LECTURES NOTES

ON

DIGITAL ELECTRIONICS AND MICROPROCESSOR

BY

Er. Jadunath murmu, Sr. lect E&tc Engg.

GOVT.POLYTECHNIC,NAYAGARH

BASICS OF DIGITAL ELECTRONICS

The branch of electronics that deals with digital data in the form of codes. There are only two codes in digital electronics, and they are 0 and 1. 0 is considered to be low logic while 1 is considered to be high logic.

Digital Electronics can also be defined as the circuit which deals with Digital Signal is knows as Digital Electronics

Advantages Of Digital Electronics

a. Digital Electronic circuits are relatively easy to design.
b. It has higher precision rate in terms of accuracy.
c. Transmitted signals are not lost over long distance.
d. Digital Signals can be stored easily.
e. Digital Electronics is more immune to 'error' and 'noise' than analog. But in case of high-speed designs, a small noise can induce error in the signal.
f. The voltage at any point in a Digital Circuit can be either high or low; hence there is less chance of confusion.
g. Digital Circuits have the flexibility that can change the functionality of digital circuits by making changes in software instead of changing actual circuit.

Disadvantages of Digital Electronics

a. The real world is analog in nature, all quantities such as light, temperature, sound etc. Digital Systems is required to translate a continuous signal to discrete which leads to small quantization errors. To reduce quantization errors a large amount of data needs to be stored in Digital Circuit.
b. Digital Circuits operate only with digital signals hence, encoders and decoders are required for the process. This increases the cost of equipment.

Number System

A digital system can understand positional number system only where there are a few symbols called digits and these symbols represent different values depending on the position they occupy in the number.

A value of each digit in a number can be determined using

a. The digit
b. The position of the digit in the number
c. The base of the number system (where base is defined as the total number of digits available in the number system).

Type number System

1. Decimal Number System
2. Binary Number System
3. Octal Number System
4. Hexadecimal Number System

Decimal Number System

The number system that we use in our day-to-day life is the decimal number system
The decimal number system contains ten digits from 0 to 9.(0,1,2,3,4,5,6,7,8,&9)
Base=10
The position in the decimal number system specifies the power of the base (10).

Example

Mathematically, we can write it as

$$2541 = (2\times1000) + (5\times100) + (4\times10) + (1\times1)$$
$$= (2\times10^3) + (5\times10^2) + (4\times10^1) + (1\times10^0)$$
$$= 2541$$

Binary Number System

Generally, a binary number system is used in the digital computers. In this number system, it carries only two digits, either 0 or 1
The binary number system contains 2 digits from 0 &1
Base=10
The position in the binary number system specifies the power of the base (2)

Mathematically, we can write it as

$$1101.011 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) +(1 \times 2^{-2}) + (1 \times 2^{-3})$$

Octal Number System

The octal number system contains 8 digits from 0 to 7(i.e. 0,1,2,3,4,5,6&7)
Base=8
The position in the octal number system specifies the power of the base (8)

Mathematically, we can write it as

$$12570= (1 \times 8^4) + (2 \times 8^3) + (5 \times 8^2) + (7 \times 8^1) + (0 \times 8^0)$$

Hexadecimal Number System

Uses 10 digits and 6 letters, 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.
Letters represents numbers starting from 10. A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.
Base =16

The position in the Hexadecimal Number System number system specifies the power of the base (8)

Mathematically, we can write it as

$19FDE_{16} = (1 \times 16^4) + (9 \times 16^3) + (F \times 16^2) + (D \times 16^1) + (E \times 16^0)$

Number System and Base Conversions

Electronic and Digital systems may use a variety of different number systems, (e.g. Decimal, Hexadecimal, Octal, Binary).

A number N in base or radix b can be written as:

(N)b = dn-1 dn-2 -- -- -- -- d1 d0 . d-1 d-2 -- -- -- -- d-m

In the above, dn-1 to d0 is the integer part, then follows a radix point, and then d-1 to d-m is the fractional part.

dn-1 = Most significant bit (MSB)

d-m = Least significant bit (LSB)

Base	Representation
2	Binary
8	Octal
10	Decimal
16	Hexadecimal

1. Decimal to Binary

Convert $(34.25)_{10}$ to Binary equivalent

Step 1: Divide the number 34 and its successive quotients with base 2.

	Remainder
2)34	
2)17	0
2) 8	1
2) 4	0
2) 2	0
2) 1	0
0	1

Step 2:

Now, perform the multiplication of 0.25 and successive fraction with base 2.

Operation	Result	carry
0.25×2	0.50	0
0.50×2	0	1

$(0.25)_{10}=(.01)_2$

Final Result is

$(\mathbf{34.25})_{\mathbf{10}}$ =$(\mathbf{100010.01})_{\mathbf{2}}$

2. Binary to Decimal

Convert $(1010.01)_2$ to equivalent Decimal No.

$(1010.01)_2 = 1x2^3 + 0x2^2 + 1x2^1 + 0x2^0 + 0x2^{-1} + 1x2^{-2} = 8+0+2+0+0+0.25 = 10.25$

$= (10.25)_{10}$

3. Decimal to Octal

Convert $(86)_{10}$ to Octal equivalent

Step 1: Divide the number 34 and its successive quotients with base 8.

	Remainder
8)86	
8)10	6
8) 1	2
0	1

Step 2:

Now perform the multiplication of 0.35 and successive fraction with base 8.

Operation	Result	carry
0.35X8	2.8	2
0.8X8	6.4	6
0.4X8	3.2	3
0.3X8	2.4	2

$(0.35)_{10}=(2632)_8$

So, the octal number of the decimal number 86.35 is 126.2632

4. Octal to Decimal

$(12.2)_8$

$1 \times 8^1 + 2 \times 8^0 + 2 \times 8^{-1} = 8+2+0.25 = 10.25$

$(12.2)_8 = (10.25)_{10}$

5. Hexadecimal to Binary

To convert from Hexadecimal to Binary, write the 4-bit binary equivalent of hexadecimal.

Binary equivalent	Hexadecimal
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

Example

$(3A)_{16} = (00111010)_2$

6. Binary to Hexadecimal

To convert from Binary to Hexadecimal, start grouping the bits in groups of 4 from the right-end and write the equivalent hexadecimal for the 4-bit binary. Add extra 0's on the left to adjust the groups.

1111011011

$\underline{0011}\ \underline{1101}\ \underline{1011}$

$(001111011011\ )_2 = (3DB)_{16}$

7. Hexa-decimal to Decimal Conversion

The process of converting hexadecimal to decimal is the same as binary to decimal. The process starts from multiplying the digits of hexadecimal numbers with its corresponding positional weights. And lastly, we add all those products.

Example 1: $(152A.25)_{16}$

$\mathbf{(152A.25)_{16}}=(1\times16^3)+(5\times16^2)+(2\times16^1)+(A\times16^0)+(2\times16^{-1})+(5\times16^{-2})$

$=5418.14453125$

8. Decimal to Hexadecimal

	Remainder Dec.	Remainder Hex.
16) 2861		
16) 178	13	D
16) 11	2	2
0	11	B

$\mathbf{2861_{10} = B2D_{16}}$

Binary addition, subtraction, Multiplication and Division

1.Binary addition

Case	A + B	Sum	Carry
1	0 + 0	0	0
2	0 + 1	1	0
3	1 + 0	1	0
4	1 + 1	0	1

In fourth case, a binary addition is creating a sum of (1 + 1 = 10) i.e. 0 is written in the given column and a carry of 1 over to the next column.

Example − Addition

0011010 + 001100 = 00100110

```
   1 1          carry
  0 0 1 1 0 1 0   = 26₁₀
 +0 0 0 1 1 0 0   = 12₁₀
 ---------------
  0 1 0 0 1 1 0   = 38₁₀
```

2. Binary Subtraction

Subtraction and Borrow, these two words will be used very frequently for the binary subtraction. There are four rules of binary subtraction.

Case	A - B	Subtract	Borrow
1	0 - 0	0	0
2	1 - 0	1	0
3	1 - 1	0	0
4	0 - 1	0	1

Example − Subtraction

0011010 - 001100 = 00001110

$$
\begin{array}{rl}
1\;1 & \text{borrow} \\
0\,0\,\cancel{1}\,\cancel{1}\,0\,1\,0 & = 26_{10} \\
-0\,0\,0\,1\,1\,0\,0 & = 12_{10} \\
\hline
0\,0\,0\,1\,1\,1\,0 & = 14_{10}
\end{array}
$$

3.Binary Multiplication

Binary multiplication is similar to decimal multiplication. It is simpler than decimal multiplication because only 0s and 1s are involved. There are four rules of binary multiplication.

Case	A x B	Multiplication
1	0 x 0	0
2	0 x 1	0
3	1 x 0	0
4	1 x 1	1

Example − Multiplication

Example:

0011010 x 001100 = 100111000

$$
\begin{array}{rl}
0\,0\,1\,1\,0\,1\,0 & = 26_{10} \\
\times 0\,0\,0\,1\,1\,0\,0 & = 12_{10} \\
\hline
0\,0\,0\,0\,0\,0\,0 & \\
0\,0\,0\,0\,0\,0\,0 & \\
0\,0\,1\,1\,0\,1\,0 & \\
0\,0\,1\,1\,0\,1\,0 & \\
\hline
0\,1\,0\,0\,1\,1\,1\,0\,0\,0 & = 312_{10}
\end{array}
$$

1.3 .1‘s complement and 2‘s complement numbers for a binary number

a.1‘s complement

1’s complement

1’s complement of a binary number is another binary number obtained by toggling all bits in it, i.e., transforming the 0 bit to 1 and the 1 bit to 0.

Original value		1’s complement
0	→	1
1	→	0

Examples:

1's complement of 7 (0111) is 8 (1000)

1's complement of 12 (1100) is 3 (0011)

Use of 1's complement

The main use of 1's complement is to represent a signed binary number. Apart from this, it is also used to perform various arithmetic operations such as addition and subtraction.

In signed binary number representation, we can represent both positive and negative numbers

.2’s complement

2’s complement of a binary number is 1 added to the 1’s complement of the binary number.

i.e. Original value	1’s complement	2’s complement
1011	0100	0100+1=0101
1101	0010	0010+1=0011

Use of 2's complement

Negative binary numbers are represented in 2’s complement form so that the same logic circuit can be used to perform addition as well as subtraction

1.4 Subtraction of binary numbers in 2‘s complement method.

The operation is carried out by means of the following steps:

(i) Find the 2’s complement of the subtrahend(negative no. only, because 2’s complement of positive no. is remain same) of given no..
(ii) Then it is added to the minuend.(add 2’s complemented with positive given no.)
(iii)If the final carry over of the sum is 1, it is dropped and the result is positive.
(iv) If there is no carry over, the two’s complement of the sum will be the result and it is negative.

Examples:

(i) 110110 - 10110

Solution:

Now, 2’s complement of 010110 is (101101 + 1) i.e.101010. Adding this with the minuend.

1 1 0 1 1 0 Minuend

1 0 1 0 1 0 2's complement of subtrahend

Carry over 1 1 0 0 0 0 0 Result of addition

After dropping the carry over we get the result of subtraction to be 100000.

(ii) 10110 – 11010

Solution:

2's complement of 11010 is (00101 + 1) i.e. 00110. Hence

Minued - 1 0 1 1 0

2's complement of subtrahend - 0 0 1 1 0

Result of addition - 1 1 1 0 0

As there is no carry over, the result of subtraction is negative and is obtained by writing the 2's complement of 11100 i.e.(00011 + 1) or 00100.

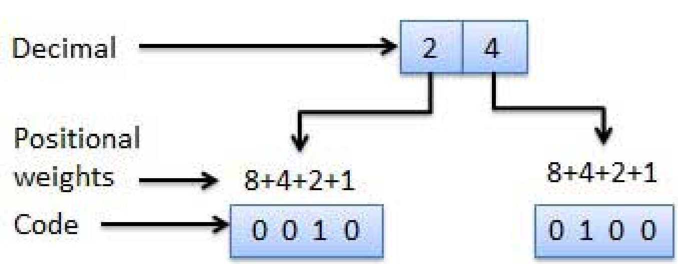
Hence the difference is – 100.

1.5 Use of weighted and Un-weighted codes & write Binary equivalent number for a number in 8421, Excess-3 and Gray Code and vice-versa.

Weighted code

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

a. BCD (8421)
b. 6311
c. 2421
d. 642-3
e. 84-2-1



Use of Weighted codes

a) Data manipulation during arithmetic operation.
b) Weighted binary code is essential for displaying numeric values in digital devices such as voltmeters and calculators
.c) To represent the decimal digits in calculators, volt meters etc.

Non-Weighted Codes

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

Non weighted codes are used in:

a) To perform certain arithmetic operations.
b) Shift position encodes.
c) Used for error detecting purpose.

Excess-3 code

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

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

Example

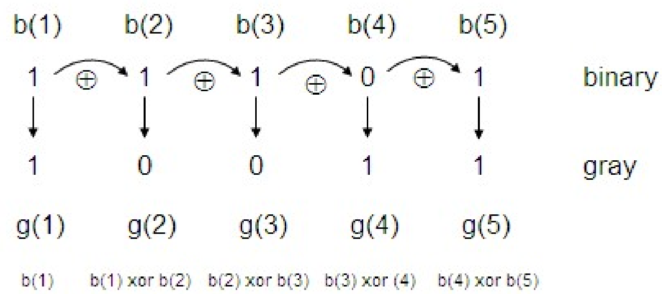
Decimal	BCD 8 4 2 1	Excess-3 BCD + 0011
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

Gray Code

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

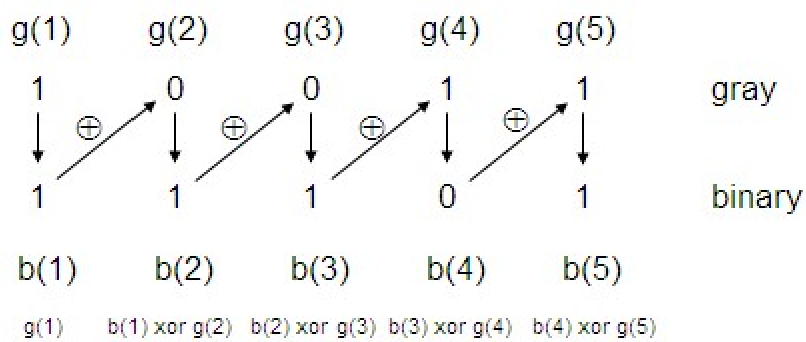
Convert a binary number to a Gray number

Let's understand the algorithm to go from binary to Gray. See the conversion from '11101' binary to its equivalent in Gray code.



Convert a Gray number to a binary number

Let's understand the algorithm to go from binary to Gray. See the conversion from '11101' binary to its equivalent in Gray code.



Application of Gray code

- Gray code is popularly used in the shaft position encoders.
- A shaft position encoder produces a code word which represents the angular position of the shaft.

1.6 Importance of parity Bit.

A parity bit is an extra bit included in binary message to make total number of 1's either odd or even. Parity word denotes number of 1's in a binary string. There are two parity system-even and odd.

Even parity system

In even parity system 1 is appended to binary string it there is an odd number of 1's in string otherwise 0 is appended to make total even number of 1's.

Odd parity system

In odd parity system, 1 is appended to binary string if there is even a number of 1's to make an odd number of 1's

Importance of parity Bit.

The purpose of a parity bit is to provide a simple way to check for

Errors

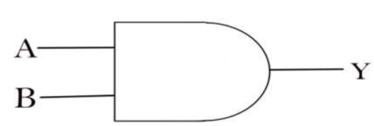
1.7 Logic Gates: AND, OR, NOT, NAND, NOR and EX-OR gates with truth table.

What is Logic Gates?

Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a certain logic. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

AND Gate

An AND gate is a logic gate having two or more inputs and a single output. An AND gate operates on logical multiplication rules

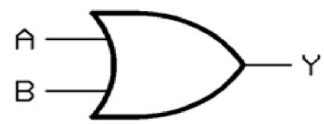


Expression for AND gate Y=A.B

Truth Table of AND gate

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

OR Gate



Expression for OR gate Y=A+B

Truth Table

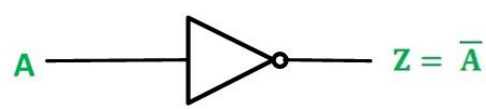
Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate

The NOT gate is the most basic logic gate of all other logic gates. NOT gate is also known as an **inverter**

NOT gate only has one input and one output

it converts 0 into 1 or 1 into 0.



Expression for NOT gate Z=$\overline{A}$

Truth Table

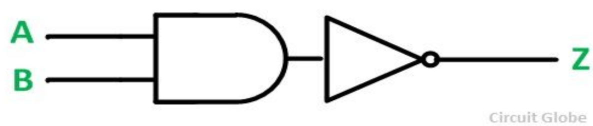
Inputs	Outputs
A	Y
0	1
1	0

NAND Gate

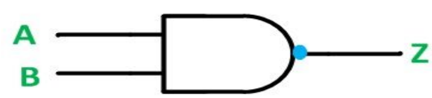
The NAND gate is a special type of logic gate in the digital logic circuit.

The NAND gate is the combination of AND -NOT gate

The NAND gate is the universal gate. It means all the basic gates such as AND, OR, and NOT gate can be constructed using a NAND gate. The output state of the NAND gate will be low only when all the inputs are high. Simply, this gate returns the complement result of the AND gate.



=



Expression for NAND gate Z=$\overline{A.B}$

Truth Table

Input		Out Put
A	B	Z=$\overline{A.B}$
0	0	0
0	1	1
1	0	1
1	1	0

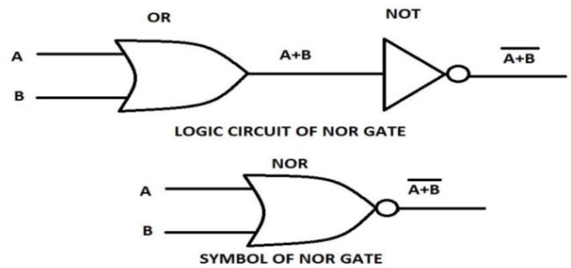
NOR Gate

The NOR gate is also a universal gate

The NOR gate is the combination of the OR -NOT gate

The NOR gate is the universal gate. It means all the basic gates such as AND, OR, and NOT gate can be constructed using a NOR gate.

The output state of the NOR gate will be high only when all of the inputs are low. Simply, this gate returns the complement result of the OR gate

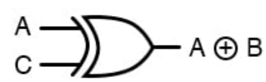


Expression for Nor gate Z=$\overline{\boldsymbol{A}+\boldsymbol{B}}$

Truth Table

Input		Out Put
A	B	**Z=$\overline{\boldsymbol{A}+\boldsymbol{B}}$**
0	0	1
0	1	0
1	0	0
1	1	0

EX-OR



Expression for EX-OR gate Z=($\bar{\boldsymbol{A}}$ B+A$\bar{\boldsymbol{B}}$)

Truth Table

Input		Out Put
A	B	**Z=A**$\oplus$B
0	0	1
0	1	0
1	0	0
1	1	0

1.8 Realize AND, OR, NOT operations using NAND, NOR gates.

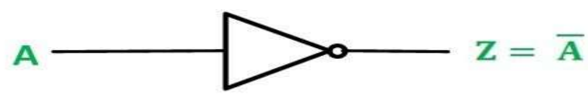
NAND and NOR gates provide the following merits in the digital logic system design

1. Fabrication of NAND and NOR gates are easier than basic gates using in the integrated digital logic families
2. Number of transistors used to design NAND and NOR gates are also less than AND and OR gates. Since the core area reduces in the integrated digital circuits.
3. The conversion of NAND and NOR are more conveniet in digital design.
4. All other logic gates can be realized completely using NAND or NOR gates.
5. Any digital ckt. can be implemented perfectly using either NAND or NOR gates thus these are called as universal gate

- **Implementation of Logic gates using NAND Gate**

i) NOT gate

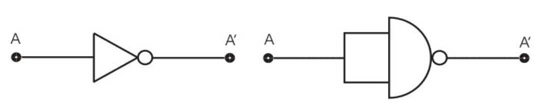
The logic symbol and Boolean expression of NOT gate is represented by



NAND equivalent representation for NOT gate is

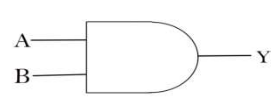
$$F=\overline{A}=\overline{A.A}$$

The above expression indicates that if the input terminals of NAND gate are Same shown in fig



ii) AND Gate

The logic symbol and Boolean expression of AND gate is represented by

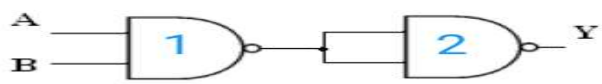


Y=A.B

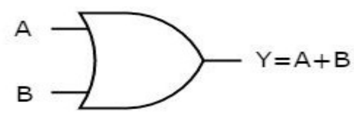
NAND equivalent representation for AND gate is

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

Now above expression can drawn as

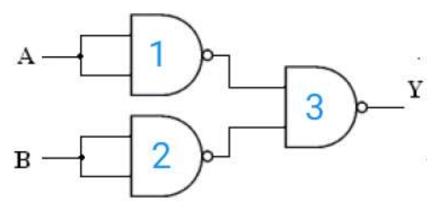


iii) OR gate:

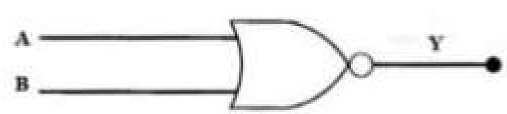


NAND equivalent representation for OR gate is

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



iv) NOR gate:



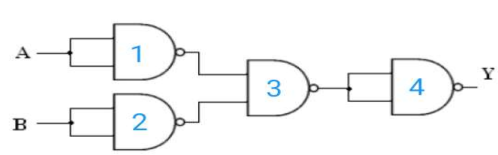
$$Y=\overline{A+B}$$

NAND equivalent representation for NOR gate is

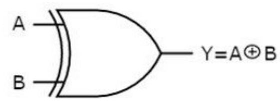
$$Y=\overline{A+B}=\overline{A}+\overline{B}$$

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

Now above expression can drawn as



v) Ex-OR gate:



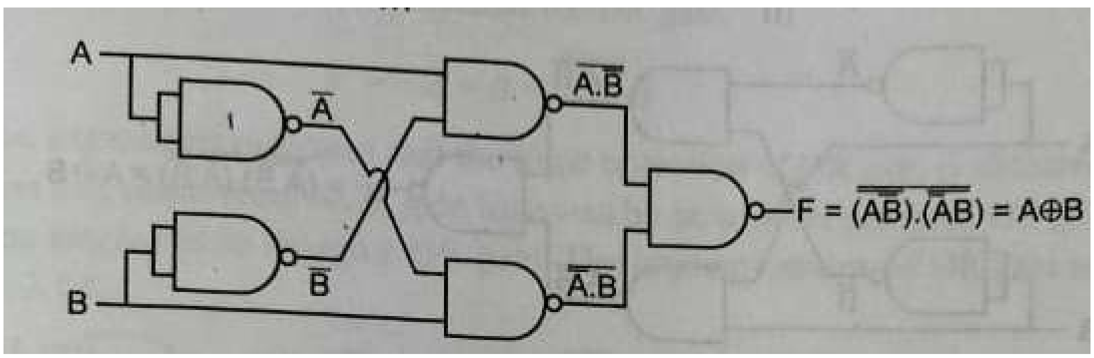
$$Y=A\bar{B}+\bar{A}B$$

NAND equivalent representation for Ex-OR gate is

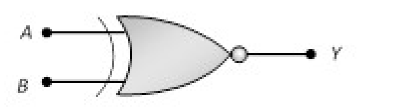
$$Y=A\bar{B}+\bar{A}B=\overline{\overline{A\bar{B}+\bar{A}B}}$$

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

Now above expression can drawn as



vi) Ex-NOR gate:



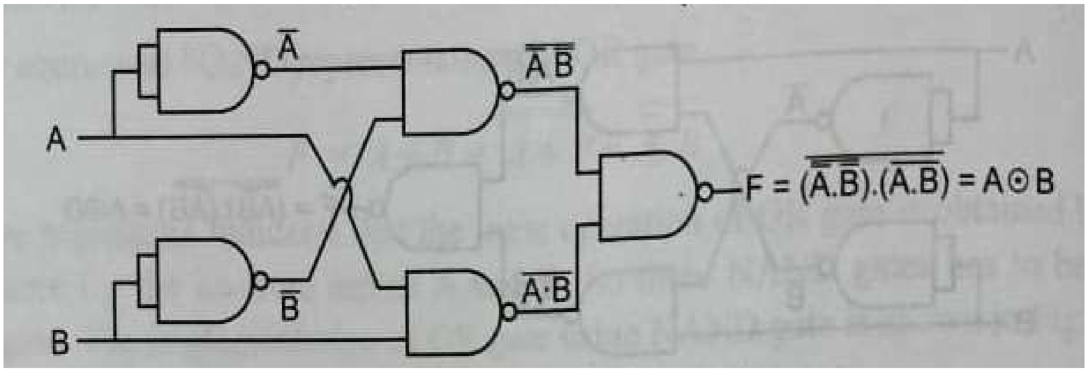
$$\mathbf{Y=A\odot B=\bar{A}\,\bar{B}+AB}$$

NAND equivalent representation for Ex-NOR gate is

$$\mathbf{Y=\bar{A}\,\bar{B}+AB=\overline{\overline{\bar{A}\,\bar{B}+AB}}}$$

$$\mathbf{=\overline{\overline{(\bar{A}\,\bar{B})}\,.\,\overline{(AB)}}}$$

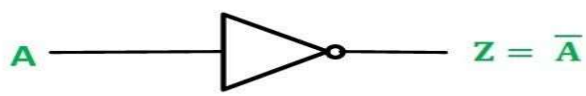
Now above expression can drawn as



- **Implementation of Logic gates using NOR Gate**

i) NOT gate

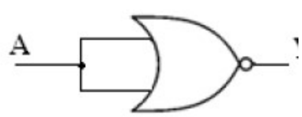
The logic symbol and Boolean expression of NOT gate is represented by



NOR equivalent representation for NOT gate is

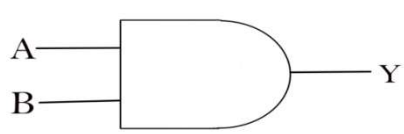
$$Z=\overline{A}$$
$$=\overline{A+A}$$

Now above expression can drawn as



ii) AND Gate

The logic symbol and Boolean expression of AND gate is represented by

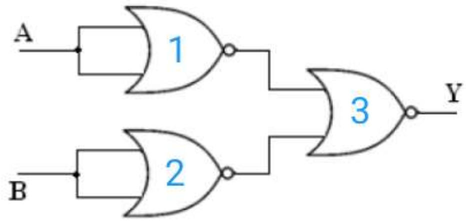


Y=A.B

NAND equivalent representation for AND gate is

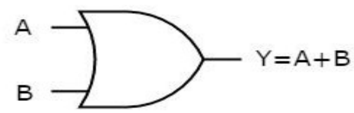
$$Y=A.B=\overline{\overline{A.B}}$$
$$=\overline{\overline{A}.+\overline{B}}$$

Now above expression can drawn as



iii) OR gate:

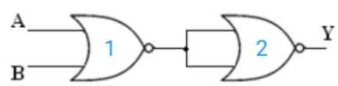
The logic symbol and Boolean expression of OR gate is represented by



NOR equivalent representation for OR gate is

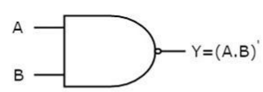
$$Y=A+B=\overline{\overline{A+B}}$$

Now above expression can drawn as



iv) NAND gate:

The logic symbol and Boolean expression of NAND gate is represented by



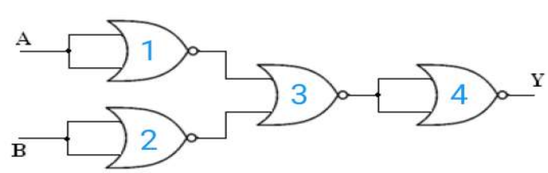
$$Y=\overline{A.B}$$

NOR equivalent representation for NAND gate is

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

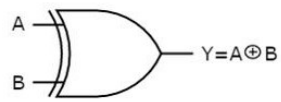
$$\overline{\overline{\overline{A}.+.\overline{B}}}$$

Now above expression can drawn as



v) Ex-OR gate:

The logic symbol and Boolean expression of **Ex-OR** gate is represented by



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

NOR equivalent representation for **Ex-OR** gate is

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

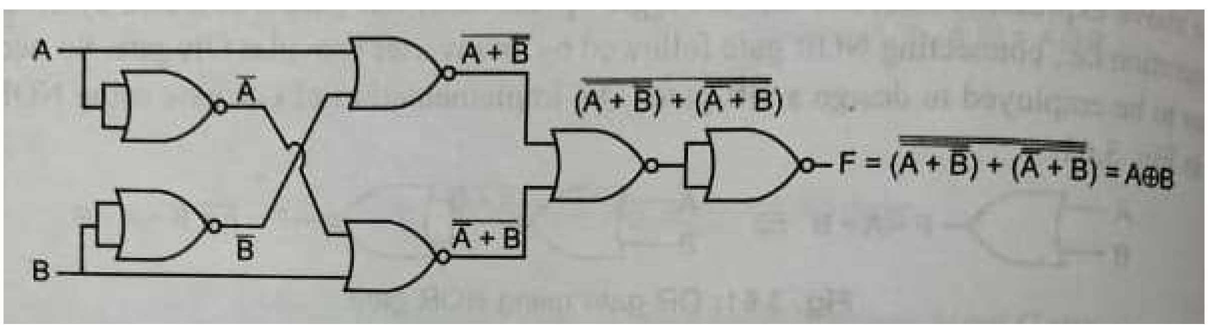
$$=\overline{\overline{(A\bar{B})}}.\overline{\overline{(\bar{A}B)}}$$

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

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

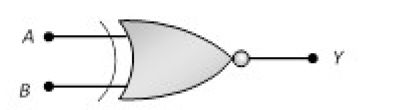
$$=\overline{\overline{\overline{(\bar{A}+B)}+\overline{(A+\bar{B})}}}$$

Now above expression can drawn as



i) Ex-NOR gate:

The logic symbol and Boolean expression of **Ex-NOR** gate is represented by



$$\mathbf{Y=A\odot B=\bar{A}\,\bar{B}+AB}$$

NOR equivalent representation for **Ex-NOR** gate is

$$\mathbf{Y=A\odot B=\overline{\overline{\bar{A}\,\bar{B}+AB}}}$$

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

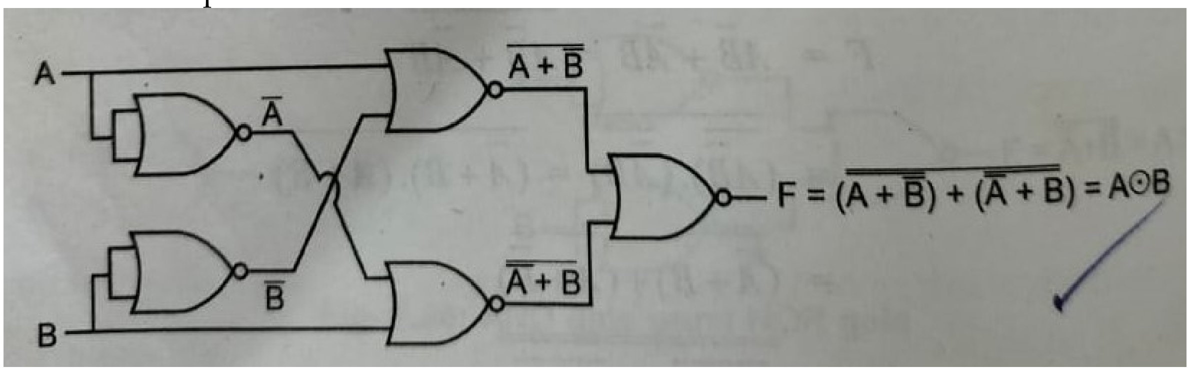
$$\mathbf{=(\overline{A\bar{B}}).(\overline{\bar{A}\,B})}$$

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

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

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

Now above expression can drawn as



PROCEDURE TO IMPLEMENT THE BOOLEAN FUNCTION USING UNIVERSAL GATE:

1. Draw a logic diagram for the Boolean f unction using basic gates i.e. AND,OR,and NOT
2. Replace the gate with equivalent NAND or NOR realization.
3. If any path has continuous two inversions, discard those terms to reduce the number of logic gates employed to implement the Boolean function.
4. Redraw the simplified logic diagram as the Universal gates implementation of Boolean function.

Example: Implement the following Boolean function using minimum number of (i)NAND gates, (ii)NOR gates

$$F=\overline{AB+CD}$$

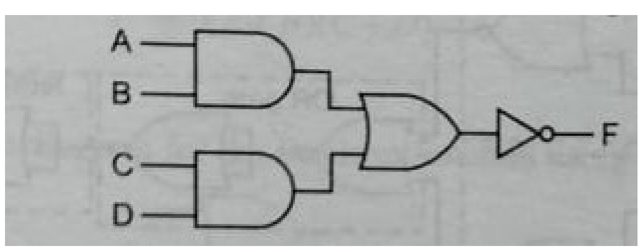
Solution:

Given Boolean function,

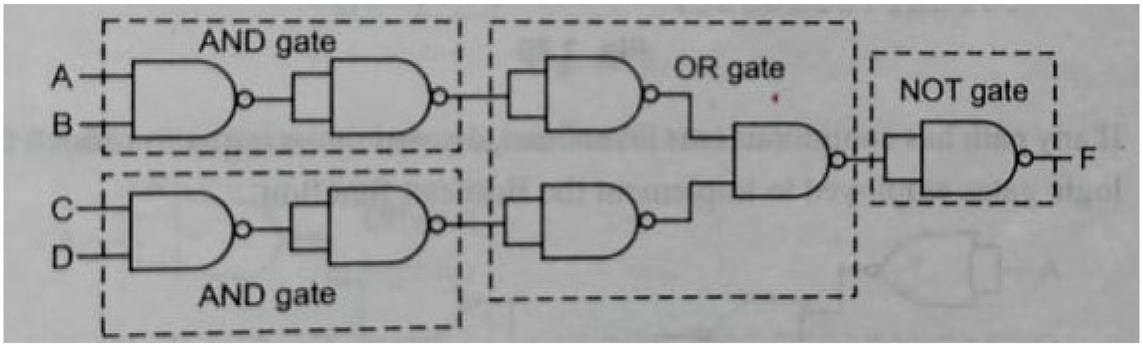
$$F=\overline{AB+CD}$$

(i) Using NAND gates:

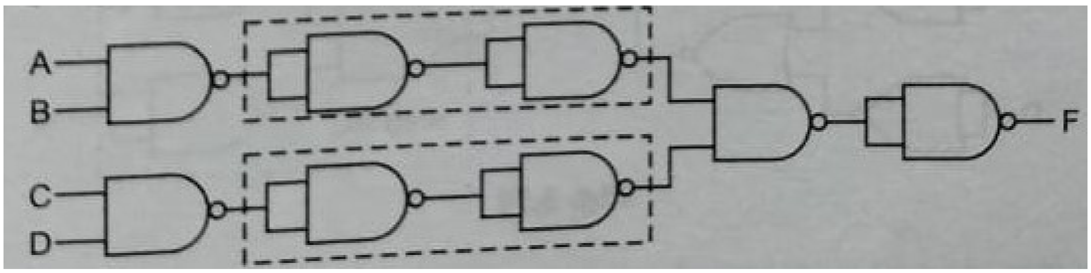
1. Step:1 Draw a logic diagram for the Boolean f unction using basic gates i.e. AND,OR,and NOT



2. Step:2 Replace the gate with equivalent NAND realization.

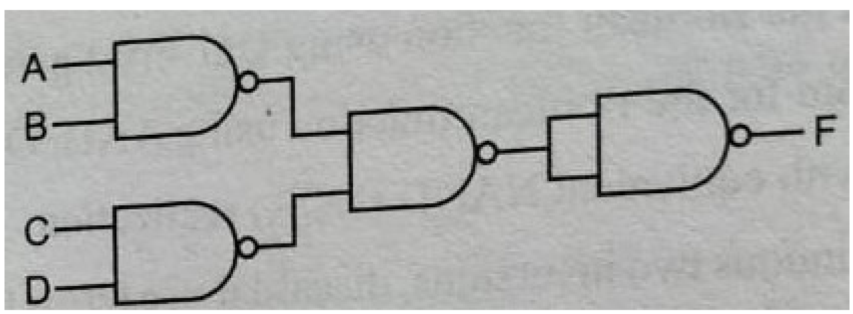


3. Step:3 If any path has continuous two inversions, discard those terms to reduce the number of logic gates employed to implement the Boolean function.



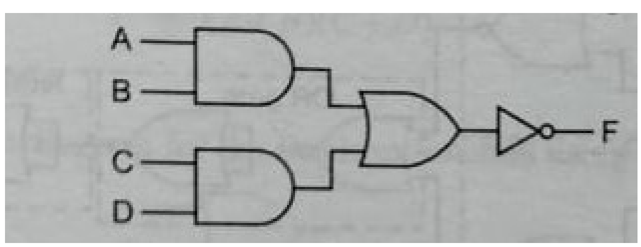
In this case, both path are having two inversions in series so discard those inverter

4. Step:4Redraw the simplified logic diagram as the Universal gates implementation of Boolean function

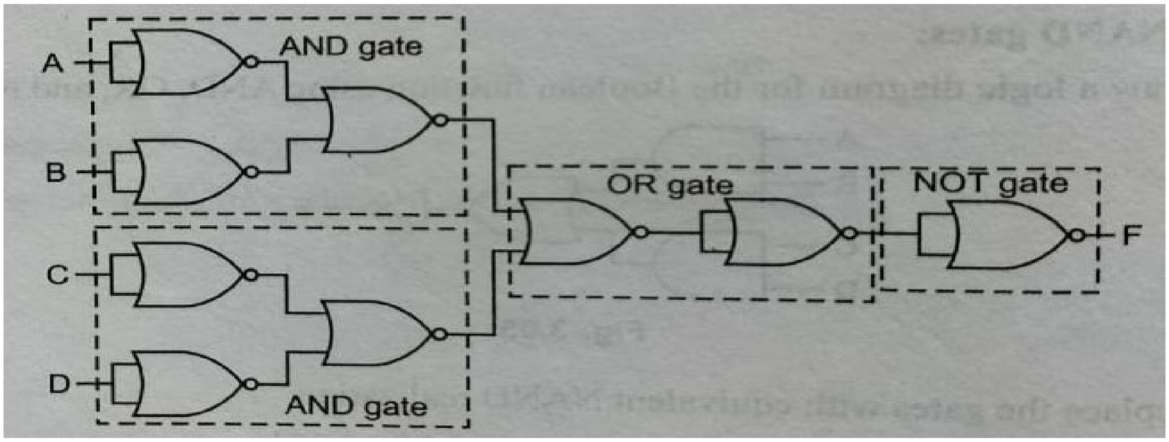


(ii) Using NOR gates:

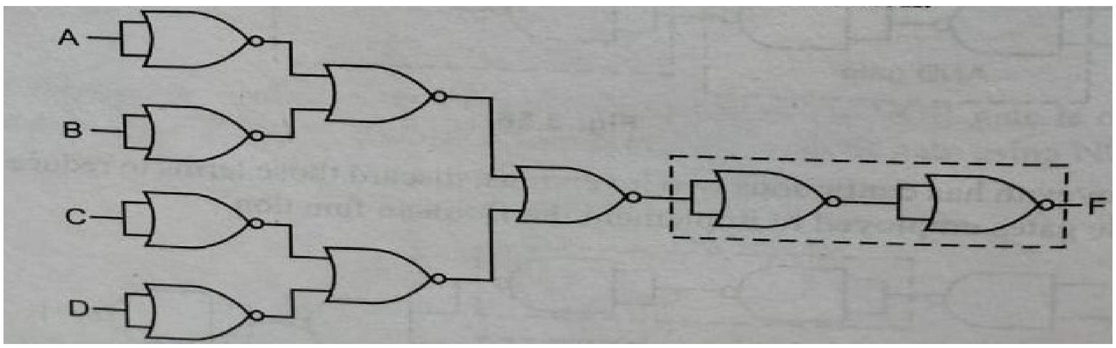
1. Step:1 Draw a logic diagram for the Boolean f unction using basic gates i.e. AND,OR,and NOT



2. Step:2 Replace the gate with equivalent NOR realization.

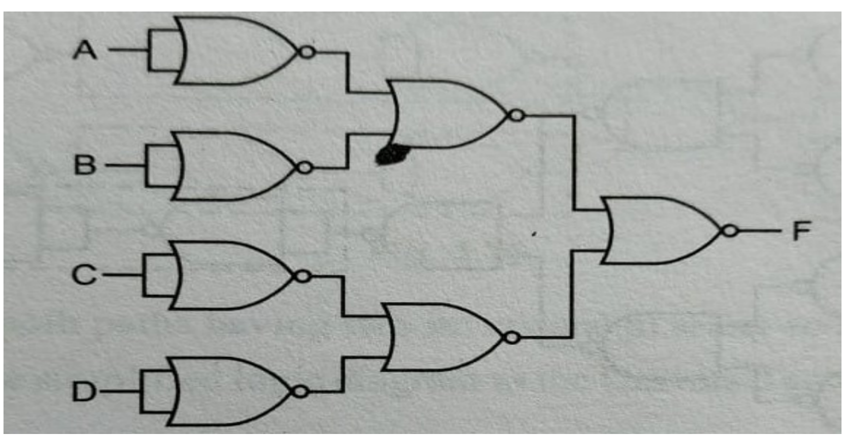


3. Step:3 If any path has continuous two inversions, discard those terms to reduce the number of logic gates employed to implement the Boolean function.



In this case, output section e having two inversions in series so discard those inverter

4. Step:4Redraw the simplified logic diagram as the Universal gates implementation of Boolean function



1.9 Different postulates and De-Morgan's theorems in Boolean algebra.

a. $\overline{\boldsymbol{AB}}$=$\overline{\boldsymbol{A}}$+$\overline{\boldsymbol{B}}$

b. $\overline{(\boldsymbol{A}+\boldsymbol{B})}=\overline{\boldsymbol{A}}.\overline{\boldsymbol{B}}$

Use Of Boolean Algebra For Simplification Of Logic Expression

What is Boolean Algebra?

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

Rule in Boolean Algebra

Following are the important rules used in Boolean algebra.

I. Variable used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.

II. Complement of a variable is represented by an overbar (-). Thus, complement of variable B is represented as $\overline{B}$. Thus if B = 0 then $\overline{B}$ = 1 and B = 1 then $\overline{B}$ = 0.

III. ORing of the variables is represented by a plus (+) sign between them. For example ORing of A, B, C is represented as A + B + C.

IV. Logical ANDing of the two or more variable is represented by writing a dot between them such as A.B.C. Sometime the dot may be omitted like ABC.

BASIC LAWS OF BOOLEAN ALGEBRA:

1. **NOT Law:**
   i. $\overline{0}$=1
   ii. $\overline{1} = 0$
   iii. A=$\overline{\overline{A}}$
2. **AND Laws**
   i. A.0=0
   ii. A. 1= A
   iii. A. A = A
   iv. A. $\overline{A}$ = 0
3. **OR Laws:**
   i. A + 0 = A
   ii. A + 1 = 1
   iii. A + A = A
   iv. A + $\overline{A}$ = 1
4. **Commutative Laws:**
   i. A + A = B + A
   ii. A . B = B . A
   iii. A + B + C = C + B + C

iv. A . B . C = B . C. A =C . A . B

5. Associative laws:

i. A +(B + C) = (A + B) + C

ii. A . (B . C) = (A . B) . C

6. Distributive law:

i. A + B C = (A + B) (A + C)

ii. A (B + C)= A B + A C

BOOLEAN THEOREM

1. A + A B = A

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

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

=AB +$A\overline{B}$

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

=A .1

=A

2. A (A + B) = A

Proof: A (A + B) = A.A+A.B

=A+AB

=A

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

Proof: A +$\overline{A}$ B = A + A B + $\overline{A}$ B

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

= A + B

4. A. ($\overline{A}$ + B) = A

Proof: A. ($\overline{A}$ + B) = A. $\overline{A}$ +A B

=0 + A B

=A B

5. AB + A. $\overline{B}$ = A

6. (A + B) . (A + $\overline{B}$) = A

7. (A + B) . (A + C) = A + B C

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

DEMORGAN'S THEOREM:

I. $\overline{\boldsymbol{A}.\boldsymbol{B}} = \overline{\boldsymbol{A}} + \overline{\boldsymbol{B}}$

II. $\overline{\boldsymbol{A}+\boldsymbol{B}} = \overline{\boldsymbol{A}}\,.\,\overline{\boldsymbol{B}}$

DUALITY THEOREM:

Duality theorem say that in the logic function applying the following changes in the AND, OR and NOT operation doesn't affect the output.

1. Swap '0' and '1' present in the expression.
2. Replacing AND operation by OR operation
3. Replacing OR operation by AND operation

Examples:

a. A + 0 = A .1 = A
b. A (B + C) = A B + A C

After applying duality theorem in the above expression, it becomes

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

ABSORPTIVE THEOREM:

a. A . ($\overline{A}$ + B) = A . B
b. A + ($\overline{A}$. B) = A + B

TRANSPOSITION THEOREM:

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

1.10. USING THE THEOREM & LAWS, SIMPLIFY THE FOLLOWING EXPRESSION

2. (A + B)(A + C)

=A.A+A.C+A.B+B.C - Distributive law

= A + A.C + A.B + B.C - Idempotent AND law (A.A = A)

=A(1+C)+A.B+B.C - Distributive law

=A+AB+BC - Identity OR law (1 + C = 1)

=A(1+B)+BC

=A.1+BC

=A+BC

3. $\overline{A}BCD + A\overline{B}CD + AB\overline{C}D + ABCD + A\overline{D} + D$

   =$\overline{A}BCD + A\overline{B}CD + AB\overline{C}D + ABCD + A\overline{D} + D$

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

   =$A + A\overline{D}$

   =$(D + A)(D + \overline{D})$

   =A+D

4. $A\overline{B}C + AB\overline{C} + \overline{A}BC + ABC$

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

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

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

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

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

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

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

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

   = $A(B + C) + BC$

   = $AB + AC + BC$

1.11. Karnaugh Map For 2,3,4 Variable, Simplification Of SOP And POS Logic Expression Using K-Map.

A. BOOLEAN FUNCTION:

Boolean function consists of a set of Boolean variables to represent a number using Boolean connectivity's logical NOT, logical AND, logical OR operations, parenthesis and equality sign. It is also known as Boolean expression.

Based on the arrangement of literals and terms Boolean expression is classified in two types such as,

1. Sum of Product (SOP) form
2. Product of Sum (POS) form

1. Sum of Product (SOP) form:

Sum of Product term is consisting of sum (OR operation) of many terms; the terms may consists of single literal or product of many literals (Variables).The sum of the terms is called SOP function.

Example:

i. F(A,B,C)=$ABC + AC + AB + A\overline{B}C$

ii. F(x,y,z)=$xy + \overline{x}z + x\overline{y}z$

iii. F(A,B,C,D)=$\overline{A}B\overline{C}D + \overline{A}BCD + A\overline{B}C\overline{D} + A\overline{B}CD + ABCD$

a. Standard Sum of Product (SOP) form:

The SOP form of expression is said to be Standard Sum of Product form or Canonical form expression if the terms present in the expression contains all the literals present in the function.

Each individual term present in the expression must have all the literals of a function.

The steps to convert non canonical SOP to Canonical or standard SOP.

1. Find the missing literal in each product term.
2. Multiply (AND) each product term to the term having missing literal by ORing the missing literal and its complement.
3. Expand the terms and rearrange the literals in the product terms.
4. Reduce the expression by omitting the repeated terms if any(i.e. A+A=A)

Example:

i) Convert the given expression F(A, B, C) = A + $\overline{\boldsymbol{B}}$C into canonical SOP form.

In the given expression, literal B and C are missing in the 1st product term. So (B+$\overline{B}$) and (C+$\overline{C}$) are multiplied (AND) with the term A. Similarly, literal A is missing in the 2nd product term. So (A+$\overline{A}$) is multiplied (AND) with the product term $\overline{B}$C.

Given;

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

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

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

$=AB\left(C + \overline{C}\right) + A\overline{B}\left(C + \overline{C}\right) + A\overline{B}C + \overline{A}\,\overline{B}C$

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

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

2. Product of Sum (POS) form:

Product of Sum (POS) term is consisting of sum (AND operation) of many terms; the terms may consists of single literal or product of many literals (Variables).The product of the set of sum terms is called POS function.

Example:

i. F(A,B,C)=$(A + B + C)(A + C)(A + B)(A + \overline{B} + C)$

ii. F(x,y,z)=$(x + y)(\overline{x} + z)(x + \overline{y} + z)$

iii. F(A,B,C,D)=$(\overline{A} + B + \overline{C} + D)(\overline{A} + B + C + D)(A + \overline{B} + C + \overline{D})(A + \overline{B} + C + D)(A + B + C + D)$

a) Standard Product of sum(POS) form:

The POS form of expression is said to be **Product of sum** form or Canonical form expression if the terms present in the expression contains all the literals present in the function.

Each individual term present in the expression must have all the literals of a function.

The steps to convert non canonical POS to Canonical or standard POS.

1. Find the missing literal in each sum term.
2. OR each sum term to the term having missing literal by ANDing(product) the missing literal and its complement.
3. Expand the terms and rearrange the literals in the sum terms.
4. Reduce the expression by omitting the repeated terms if any(i.e. A . A = A)

Let us see an example here.

Convert the given expression F(A, B, C) = (A+B)(B+C) into canonical POS form.

In the given expression, literal C is missing in the 1st sum term. So ($C.\overline{C}$) is added with the term (A+B). Similarly, literal A is missing in the 2nd sum term. So ($A.\overline{A}$) is added with the term (B+C).

Given;

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

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

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

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

3. SIMPLIFICATION OF BOOLEAN FUNCTION:

There are 3-different basic simplification methods available for minimizing Boolean function

1. Boolean algebra
2. Karnaugh map
3. Quine McCluskey method

a. KARNAUGH MAP (K-MAP):

Simplifying the Boolean functions using Boolean postulates and theorems. It is a time consuming process and to re-write the simplified expressions after each step.

To overcome this difficulty, **Karnaugh** introduced a method for simplification of Boolean functions in an easy way.

This method is a graphical method for simplification of Boolean function which consists of 2^n cells for 'n' variables. Each cell of K-map represents one of the **minterm**. The adjacent cells are differed only in single bit position.

Classification of K –Map:

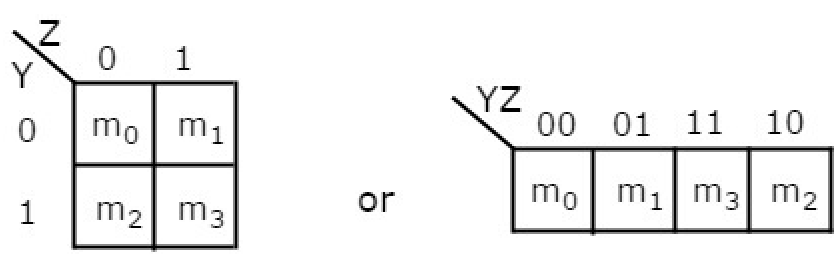
Depends on the number of variables used in the K-map it is classified as

i. **2-Variable k-map**
ii. **3-Variable k-map**
iii. **4- Variable k-map**
iv. **5- Variable k-map**

1. 2- Variable k-map:

The number of variable (n) =2

The number of cells $=2^n=2^2=4$



- The possible combinations of grouping 2 adjacent minterms are {(m_0, m_1), (m_2, m_3), (m_0, m_2) and (m_1, m_3)}.

Variable		Minterms	
A	B	Representation	m_i
0	0	$\overline{A}\,\overline{B}$	m_1
0	1	$\overline{A}\,B$	m_2
1	0	$A\,\overline{B}$	m_3
1	1	$A\,B$	m_4

(Minterms of 2-variable expression)

2. 3- Variable k-map:

The number of variable (n) =3

The number of cells $=2^n=2^3=8$

X \ YZ	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

3. 4- Variable k-map:

The number of variable (n) =4

The number of cells $=2^n=2^4=16$

WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Don’t care condition:

In some digital systems, nonessential minterms or maxterms may be introduced in the input sequences. Such nonessential minterms or maxterms are called as don’t care condition in the Boolean expression.

These nonessential terms never occur in the input sequence of the system.

Normally, in K-Map don’t care conditions are represented by symbol ‘X’. Don’t care values can be taken as either ‘0’ or ‘1’.

Don’t care conditions occur in the digital systems under the following condition:

i. If certain combinations of input variables are never occur, then the output functions of such combinations are considered as nonessential or don’t care condition.
ii. If certain combinations of variables are irrelevant even all the input combination of variables occurs, then the output functions of such combinations are considered as nonessential or don’t care condition.

Grouping cell for Minimization:

In K-map, minterms are marked by’1’

maxterm are marked by’0’

don’t care are marked by ‘d’ or ‘x’ i.e X= ‘0’ or ‘1’

In minterm function, don’t care condition is considered as ‘1’ if necessary for simplification or grouping cell. Else, it is marked by ‘0’

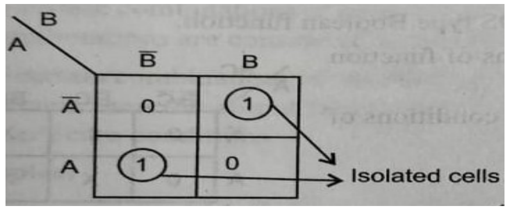
In maxterm function, don’t care condition is considered as ‘0’ if necessary for simplification or grouping cell. Else, it is marked by ‘1’

Grouping of cell or Loop of cell is process of combining adjacent cells for simplification.

Grouping is obtained by combining 1’s or 0’s of 2^i number cells, where i=0,1,2…,n (n $\longrightarrow$ number of variables used in the Boolean function.)

Isolation Cell or Single cell group(i=0):

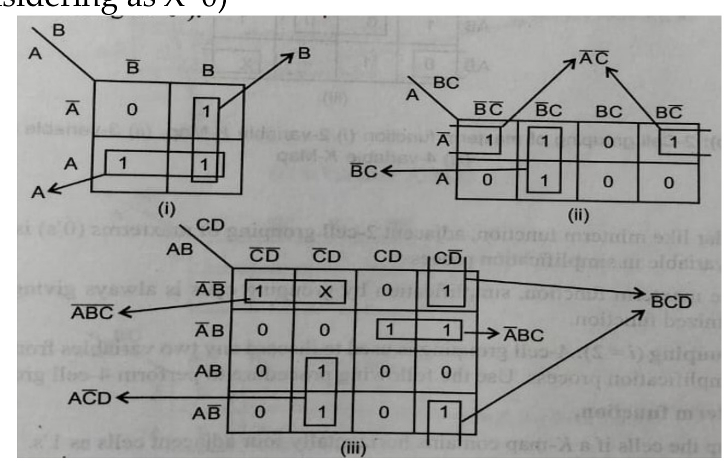
i. K-map cell is called as Isolation group when no adjacent horizontal or vertical cell is '1' for minterm and '0' for maxterm.
ii. Isolation cell can't be used for simplification, it gives the Boolean function remain as same



2- Cell group(i=1): 2 cell grouping is used to discard any variable from two adjacent cell in the simplification process

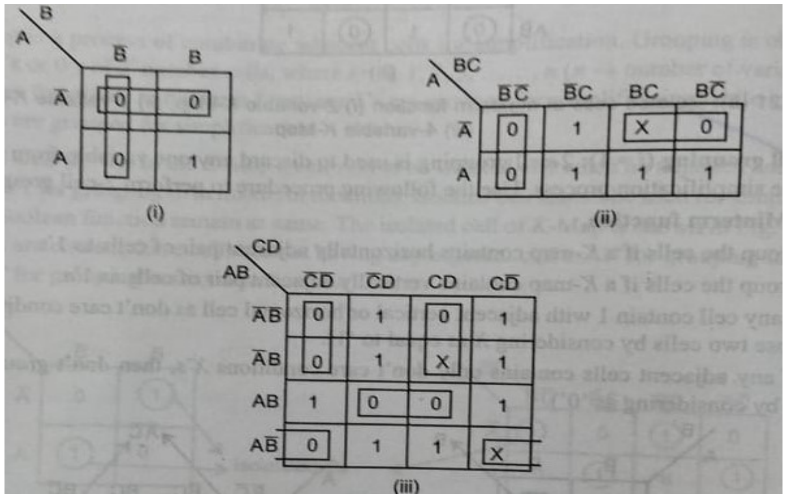
Procedure for Minterm function:

i. Group the cell if a k-map contains horizontally adjacent pair (2 cell) of cells as 1's
ii. Group the cell if a k-map contains vertically adjacent pair (2 cell) of cells as 1's
iii. If any cell contain 1 with adjacent vertical or horizontal cell as don't care condition 'X' then group those two cells by considering X=1
iv. If any cell contain only don't care condition 'X' then don't group those cells (Discard by considering as X=0)



Procedure for Maxterm function:

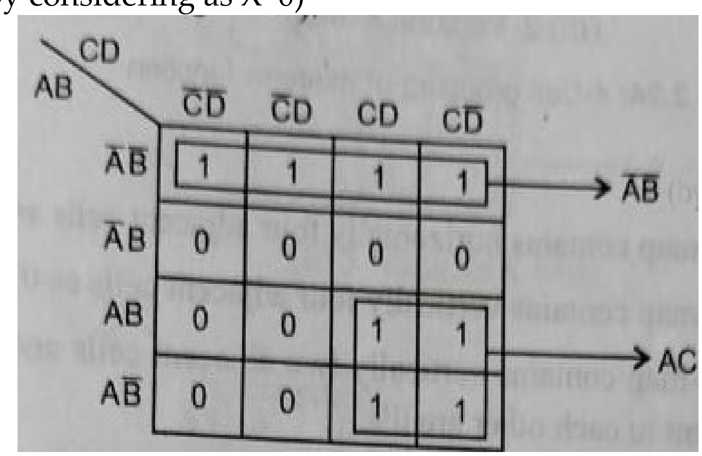
i. Group the cell if a k-map contains horizontally adjacent pair of cells as 0's
ii. Group the cell if a k-map contains vertically adjacent pair (2 cell) of cells as 0's
iii. If any cell contain 0 with adjacent vertical or horizontal cell as don't care condition 'X' then group those two cells by considering X=0
iv. If any cell contain only don't care condition 'X' then don't group those cells (Discard by considering as X=1)

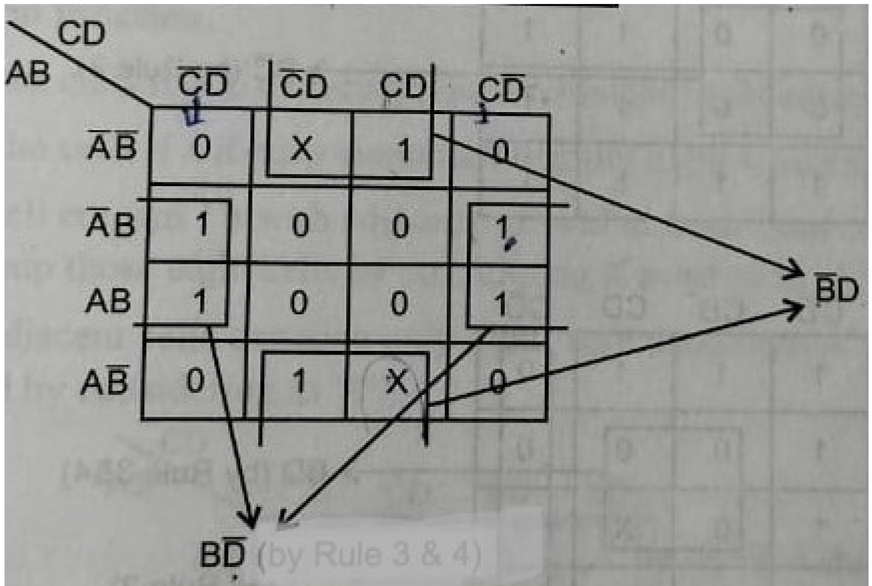


4- Cell group(i=2): 4 cell grouping is used to discard any two variables from four(4) adjacent cells in the simplification process

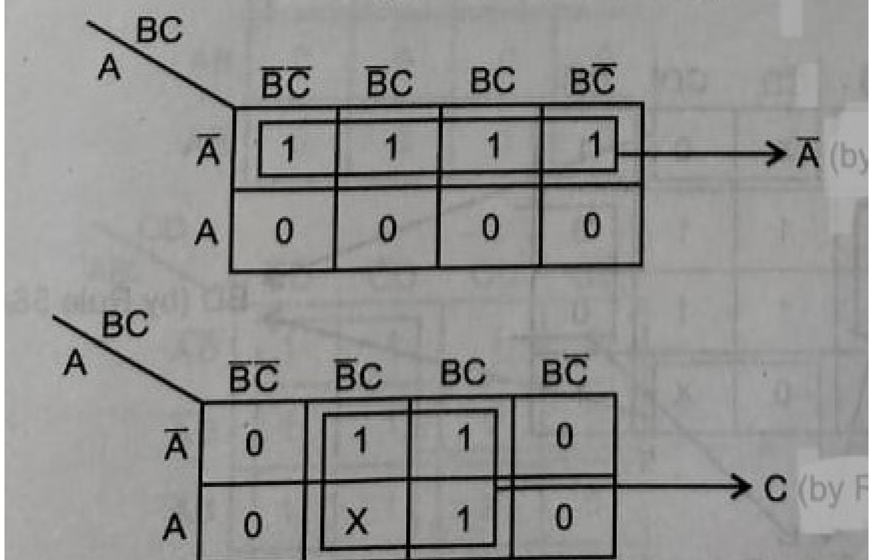
Procedure for Minterm function:

i. Group the cell if a k-map contains horizontally four (4) adjacent of cells as 1's
ii. Group the cell if a k-map contains vertically four)4) adjacent pail of cells as 1's
iii. Group the cell If a K-map contain vertically two adjacent cell and horizontal two adjacent cell which adjacent to each other are 1's.
iv. If any cell contain 1's with adjacent vertically or horizontal cell as don't care condition 'X' then group those four cell by considering X=1.
v. If any adjacent cell contain only don't care condition 'X' then don't group those cells (Discard by considering as X=0)

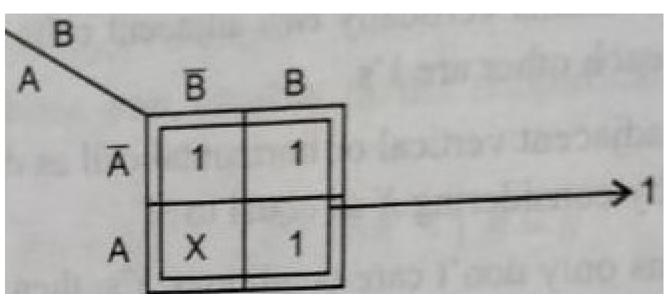




(i) 4-Variable *K*-Map



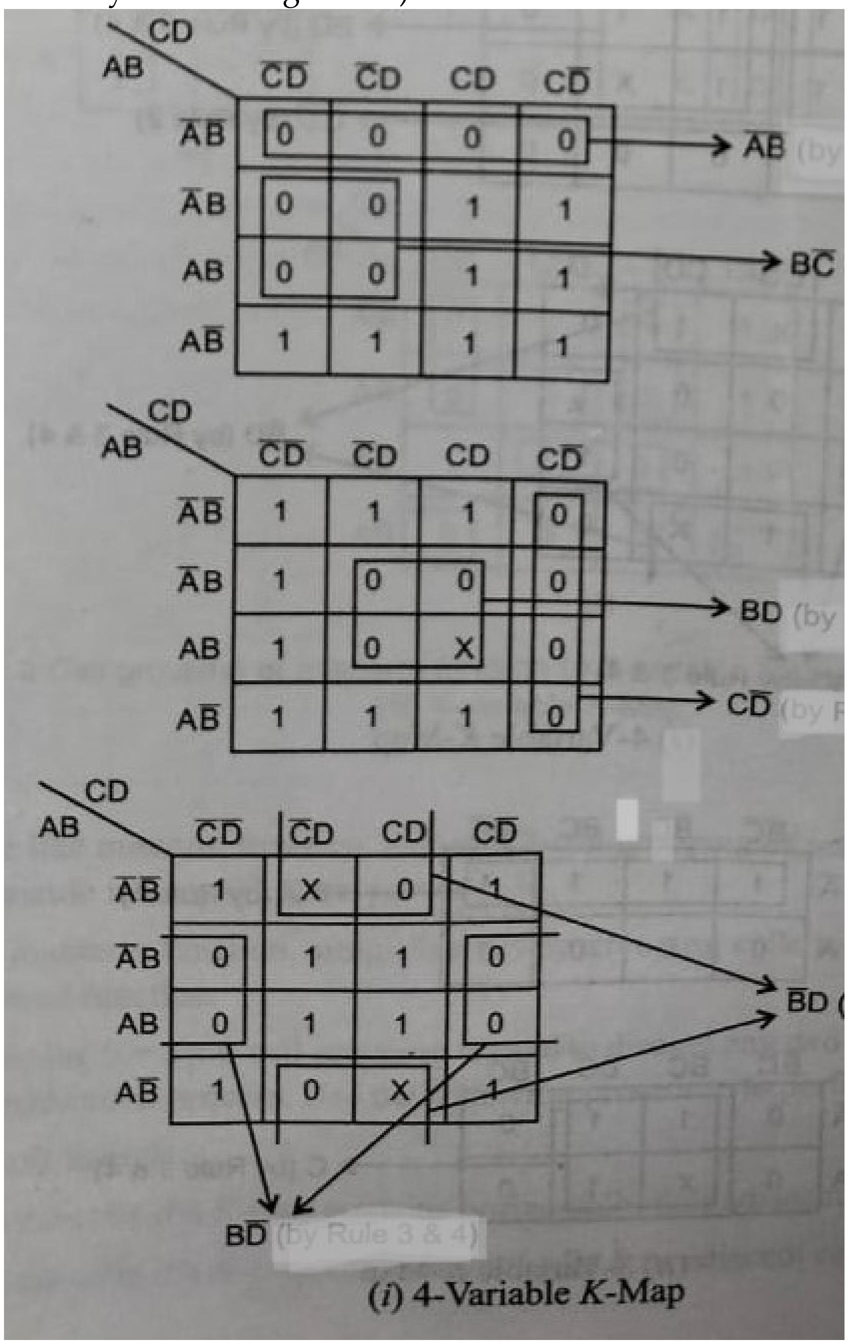
(ii) 3-Variable *K*-Map



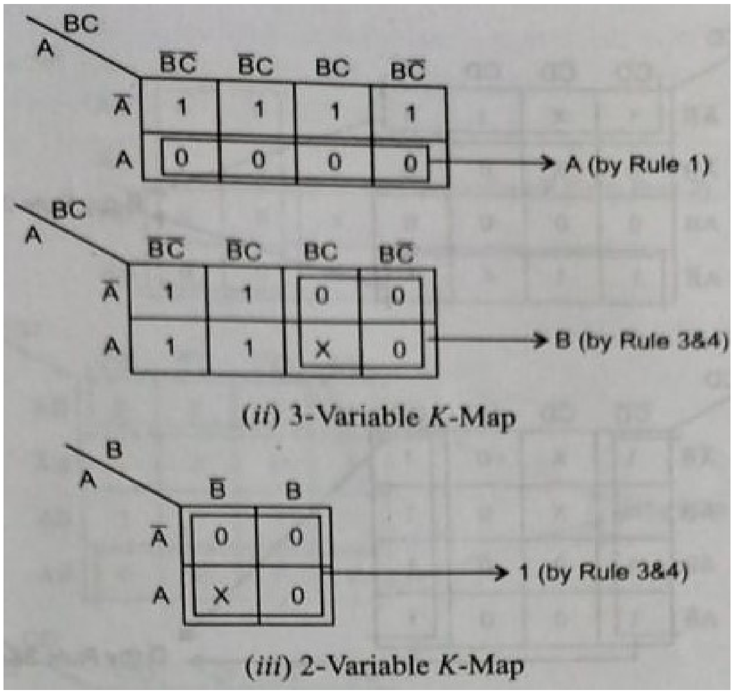
(iii) 2-Variable *K*-Map

Procedure for Maxterm function:

i. Group the cell if a k-map contains horizontally four (4) adjacent of cells as 0's
ii. Group the cell if a k-map contains vertically four)4) adjacent pail of cells as 0's
iii. Group the cell If a K-map contain vertically two adjacent cell and horizontal two adjacent cell which adjacent to each other are 0's.
iv. If any cell contain 1's with adjacent vertically or horizontal cell as don't care condition 'X' then group those four cell by considering X=0.
v. If any adjacent cell contain only don't care condition 'X' then don't group those cells (Discard by considering as X=1)



(i) 4-Variable K-Map



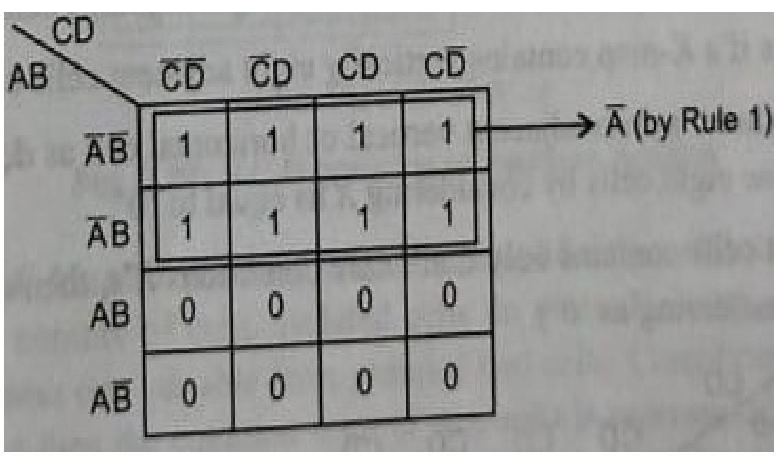
(ii) 3-Variable K-Map

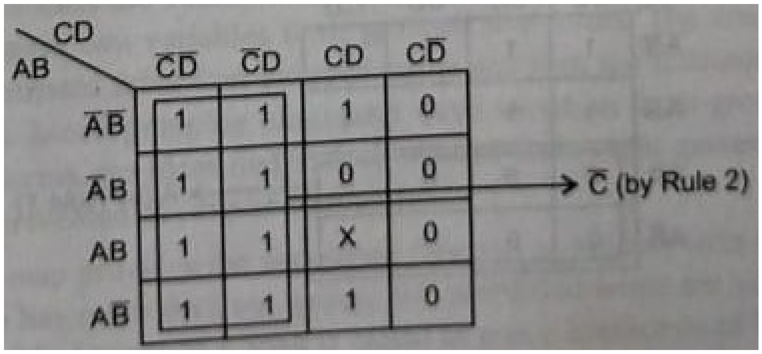
(iii) 2-Variable K-Map

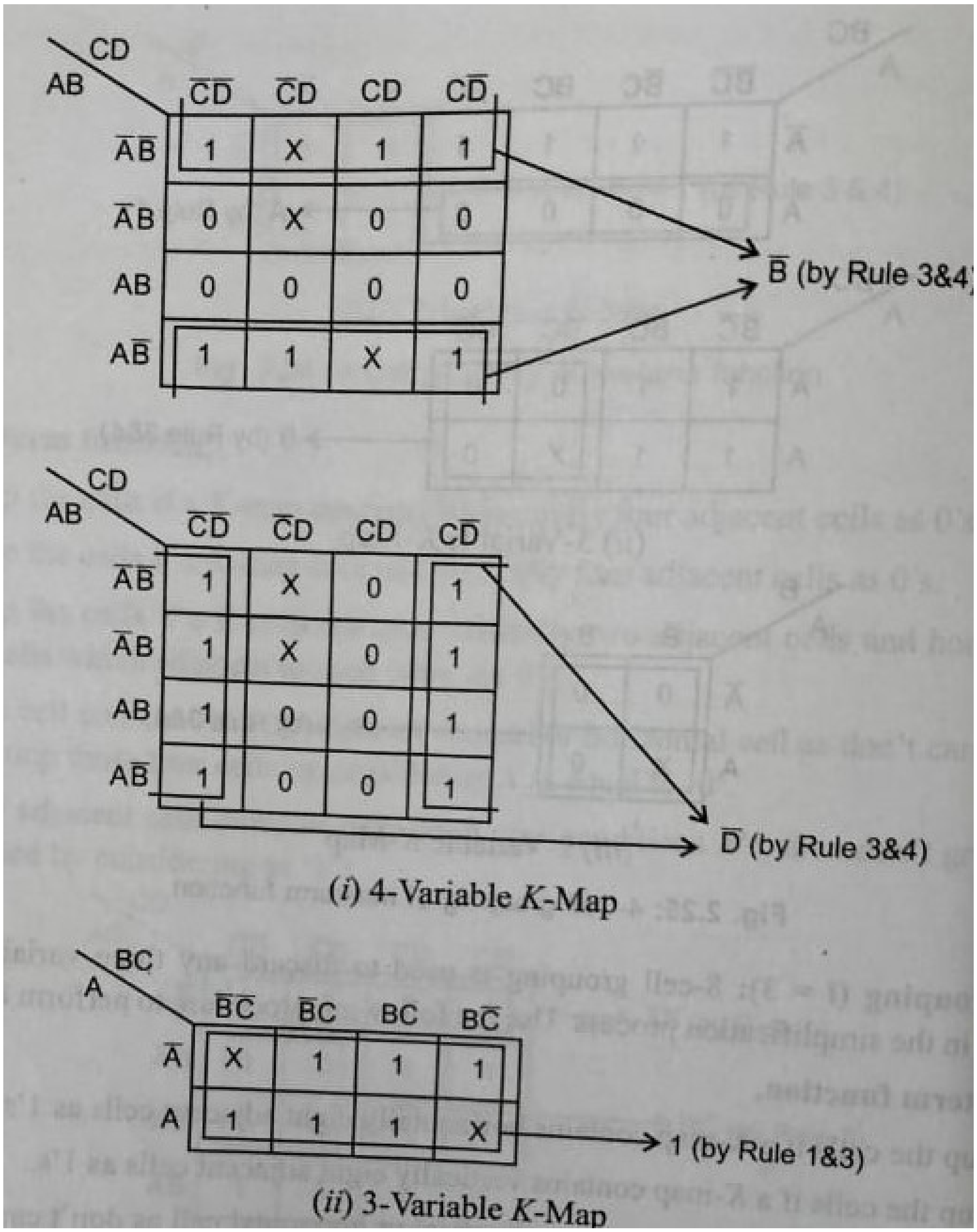
8- Cell group(i=3): 8 cell grouping is used to discard any three (3) variables from eight (8) adjacent cells in the simplification process

Procedure for Minterm function:

i. Group the cell if a k-map contains horizontally eight (8) adjacent of cells as 1's
ii. Group the cell if a k-map contains vertically eight (8)) adjacent pail of cells as 1's
iii. If any cell contain 1's with adjacent vertically or horizontal cell as don't care condition 'X' then group those eight (8) cell by considering X=1.
iv. If any adjacent cell contain only don't care condition 'X' then don't group those cells (Discard by considering as X=0)





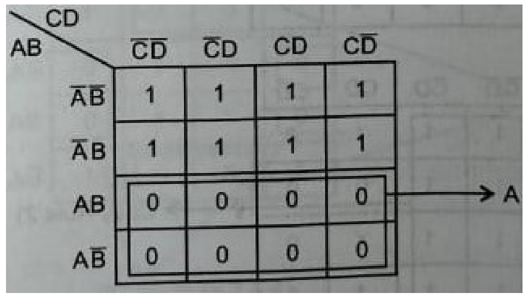


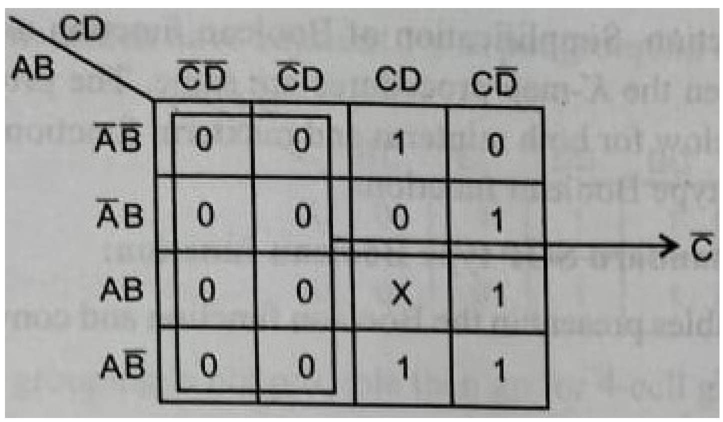
(*i*) 4-Variable *K*-Map

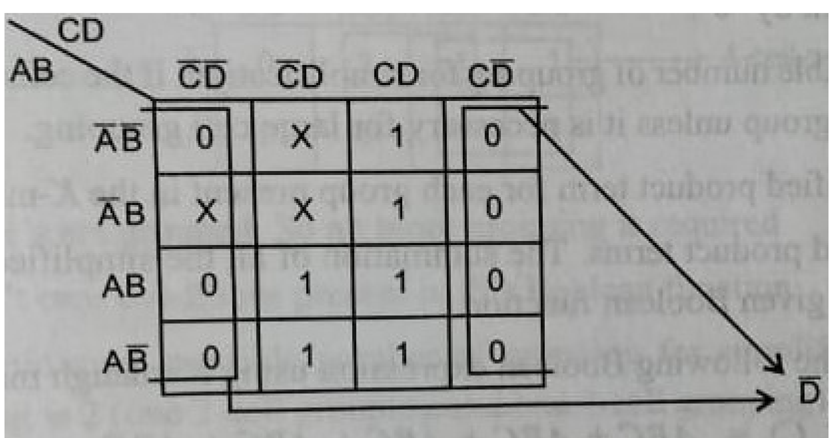
(*ii*) 3-Variable *K*-Map

Procedure for Maxterm function:

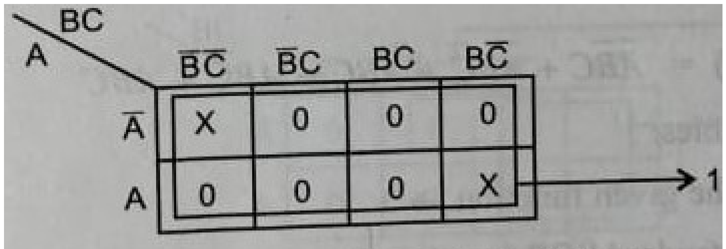
i. Group the cell if a k-map contains horizontally eight (8) adjacent of cells as 0's
ii. Group the cell if a k-map contains vertically eight (8)) adjacent pail of cells as 0's
iii. If any cell contain 0's with adjacent vertically or horizontal cell as don't care condition 'X' then group those eight (8) cell by considering X=0.
iv. If any adjacent cell contain only don't care condition 'X' then don't group those cells (Discard by considering as X=1)







(*i*) 4-Variable *K*-Map



(ii) 3-Variable K-Map

SHORT QUESTIONS AND ANSWERS

1. Define digital system?

Ans. A digital system is a system which deals with discrete signal. The input and output of this system is two binary value which is 0 and 1. Examples of digital systems are mobile phones, radio, megaphones and many more

2. List the applications of digital system?

Ans. Mobile Phones, Calculators and Digital Computers
Radios and communication Devices.

3. What is meant by bit?

Ans. Single digit that used to represent the number is called bit i.e 1 or 0

4. What is radix number system?

Ans. Radix (base) number system is a general representation of all the number system. It represent the weight of each digits present in the number system.
Example :
Base of binary no. system =2
Base of octal no. system =8
Base of hexadecimal no. system=16

5. Define binary code?

Ans. A group of binary bit that are used to represent the characters, numbers, letters or words or symbol is called as binary codes.

The digital data is represented, stored and transmitted as group of binary bits. This group is also called as binary code. The binary code is represented by the number as well as alphanumeric letter.

6. What are weighted binary codes?

Ans. A code which consists of bit weight for each digit present in the binary code is called weighted binary codes

Example:

BCD codes

7. What are non-weighted binary codes?

Ans. A code which is not having any bit weight for the digit present in the binary code is called non-weighted binary codes

Example: Excess-3 code, gray code.

8. What is gray code? Why is it called as reflective code and cyclic code?

Ans. It is the non-weighted binary code, that means there are no specific weights assigned to the bit position. only one bit position will change each time the decimal number is incremented so called reflective code. Also the adjacent gray representation differs in only binary bit hence it is referred as cyclic code.

9. State the associative property of Boolean algebra

Ans. Associative law defines that the grouping of variable in the multivariable AND and OR operation does not change the output.

i. A +(B + C) = (A + B) + C
ii. A . (B . C) = (A . B) . C

10. State the distributive property of Boolean algebra

Ans. Associative law defines that the distribution of variable with AND operation over OR operation is equal to distribution of variable with OR operation over AND operation

i. A + B C = (A + B) (A + C)
ii. A (B + C)= A B + A C

11. State the DeMorgan's theorem

i. $\overline{\boldsymbol{A}.\boldsymbol{B}} = \overline{\boldsymbol{A}} + \overline{\boldsymbol{B}}$

ii. $\overline{\boldsymbol{A}+\boldsymbol{B}} = \overline{\boldsymbol{A}}\,.\,\overline{\boldsymbol{B}}$

2.COMBINATIONAL LOGIC CIRCUITS

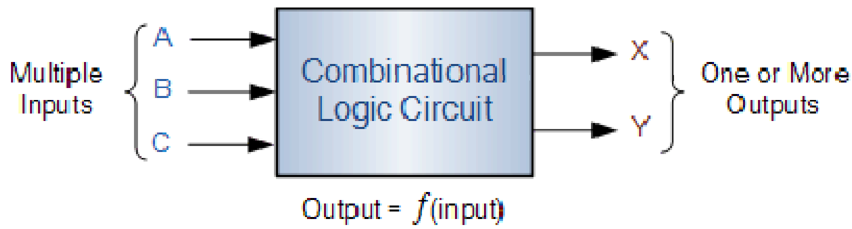
2.1 Give the concept of combinational logic circuits:

A combinational circuit is the digital logic circuit in which the output depends on the combination of present inputs applied to the circuit and It does not depend past input

Combinational circuits are developed using combination of AND, OR, NOT, NAND, and NOR logic gates.

Combinational Logic Circuits are memory less digital logic circuits whose output at any instant in time depends only on the combination of its inputs

The combinational logic circuits have no feedback circuit is used.



2.2 Half adder circuit and verify its functionality using truth table:

Half adder is a combinational circuit which consists of two binary input variables called augend and addend and two binary output variables called sum and carry. In the addition result, the lower significant bit is called as sum and the higher significant bit is called as carry.

Truth table

Input		Output	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

K –map for sum

	0	1
0	**0**	**1**
1	**1**	**0**

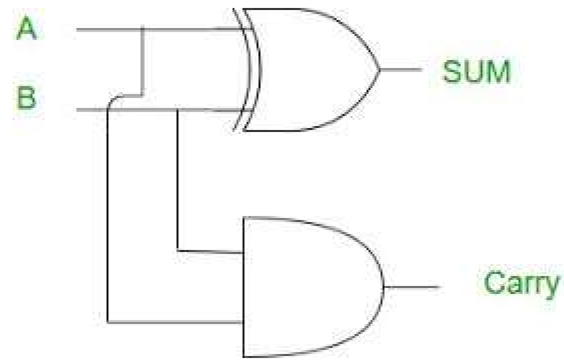
Sum= $\boldsymbol{\overline{A}B + A\overline{B} = A \oplus B}$

K –map for carry

	0	1
0	**0**	**0**
1	**0**	**1**

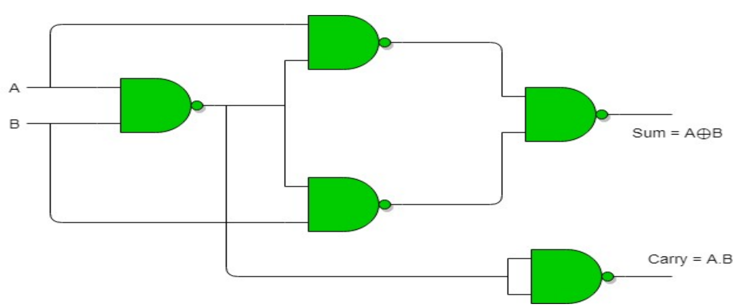
Carry=AB

Ckt diagram

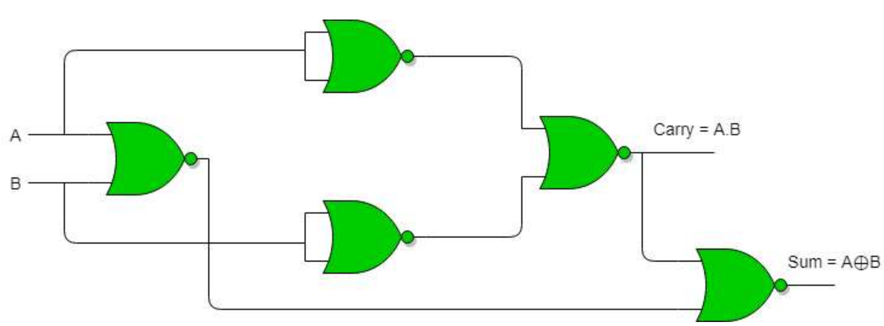


2.3 Realize a Half-adder using NAND gates only and NOR gates only.

Half-adder using NAND gates



Half-adder using NAND gates



2.4 Full adder circuit and explain its operation with truth table:

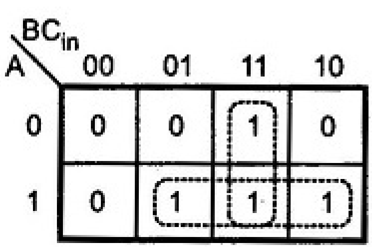
Full adder is a combinational circuit which consists of three binary input variables called augend and addend and two binary output variables called sum and carry. In the addition result, the lower significant bit is called as sum and the higher significant bit is called as carry

Truth table

Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

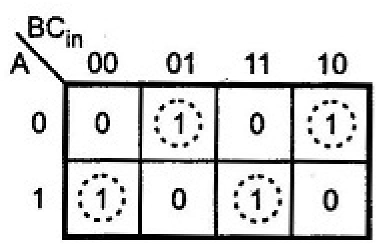
k-map

For Carry (C_{out})



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

For Sum

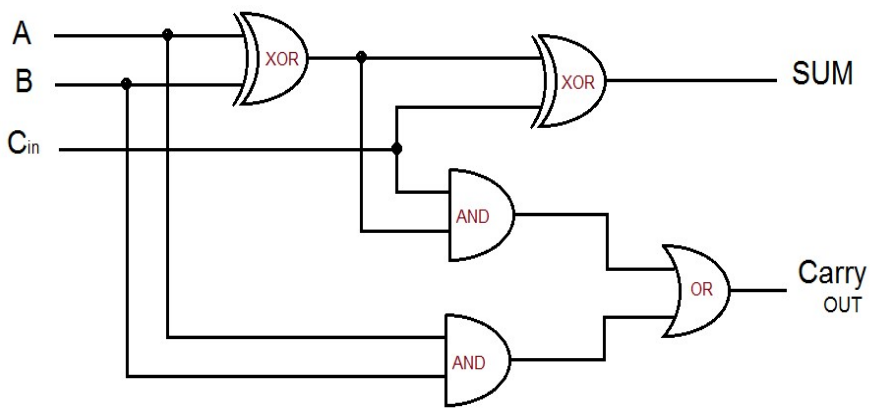


$Sum = \bar{A}\,\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\,\bar{C}_{in} + ABC_{in}$

K-map can be simplified as

$$
\begin{aligned}
\text{SUM} &= \bar{A}\,\bar{B}\,C_{in} + \bar{A}\,B\,\bar{C}_{in} + A\,\bar{B}\,\bar{C}_{in} + A\,B\,C_{in} \\
&= C_{in}\,(\bar{A}\,\bar{B} + A\,B) + \bar{C}_{in}\,(A\,\bar{B} + \bar{A}\,B) \\
&= C_{in}\,\left[(\bar{A} + B)\cdot(A + \bar{B})\right] + \bar{C}_{in}\,(A\,\bar{B} + \bar{A}\,B) \\
&= C_{in}\,(\overline{A\bar{B}}\cdot\overline{\bar{A}B}) + \bar{C}_{in}\,(A\,\bar{B} + \bar{A}\,B) \\
&= C_{in}\,(\overline{A\bar{B} + \bar{A}B}) + \bar{C}_{in}\,(A\,\bar{B} + \bar{A}\,B) \\
&= C_{in} \oplus (A\,\bar{B} + \bar{A}\,B) \\
&= C_{in} \oplus (A \oplus B)
\end{aligned}
$$

Full adder circuit diagram



2.5 Realize full-adder using two Half-adders and an OR – gate and write truth table.

The full adder can be implemented with two half adders by cascading them. The sum output of first half adder is Ex-OR of A and B. The sum output of full adder is Ex-OR of Cin and output of first half adder.

Truth table

Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

k-map

For Carry (C_{out})

A \ BC_{in}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

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

For Sum

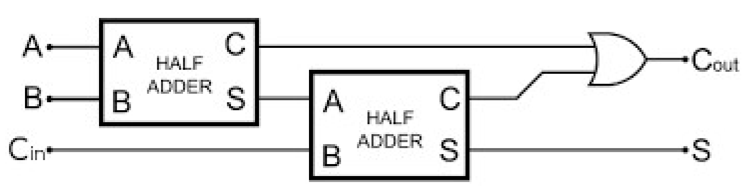
A \ BC_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$\text{Sum} = \bar{A}\,\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\,\bar{C}_{in} + ABC_{in}$

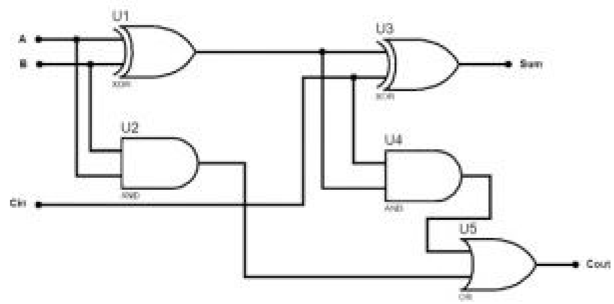
K-map can be simplified as

$$
\begin{aligned}
\textbf{SUM} &= \bar{A}\,\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\,\bar{B}\bar{C}_{in} + AB\,C_{in} \\
&= C_{in}\,(\bar{A}\,\bar{B} + AB) + \bar{C}_{in}\,(A\,\bar{B} + \bar{A}\,B) \\
&= C_{in}\,\left[(\bar{A} + B)\cdot(A + \bar{B})\right] + \bar{C}_{in}\,(A\,\bar{B} + \bar{A}\,B) \\
&= C_{in}\,(\overline{A\bar{B}}\cdot\overline{\bar{A}\,B}) + \bar{C}_{in}\,(A\,\bar{B} + \bar{A}\,B) \\
&= C_{in}\,(\overline{A\bar{B} + \bar{A}\,B}) + \bar{C}_{in}\,(A\,\bar{B} + \bar{A}\,B) \\
&= C_{in} \oplus (A\,\bar{B} + \bar{A}\,B) \\
&= C_{in} \oplus (A \oplus B)
\end{aligned}
$$

$$
\begin{aligned}
\textbf{Cout} &= BC_{in} + AC_{in} + AB \\
&= (A + \bar{A})BC_{in} + (B + \bar{B})AC_{in} + AB \\
&= ABC_{in} + \bar{A}BC_{in} + BAC_{in} + \bar{B}AC_{in} + AB \\
&= \bar{A}BC_{in} + \bar{B}AC_{in} + ABC_{in} + AB \\
&= C_{in}(\bar{A}B + \bar{B}A) \quad + ABC_{in} + AB \\
&= C_{in}(A \oplus B) \quad + AB(C_{in} + 1) \\
&= C_{in}(A \oplus B) \quad + AB
\end{aligned}
$$



Block diagram



Circuit diagram

2.6 Full subtractor circuit and explain its operation with truth table.:

a. Half adder circuit and verify its functionality using truth table:

Half subtract is a combinational circuit which consists of two binary input variables called minuend and subtrahend and two binary output variables called difference and borrow. In the two bit result, the lower significant bit is called as difference and the higher significant bit is called as borrow.

Truth table

A	B	Borrow	Difference
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

K- map

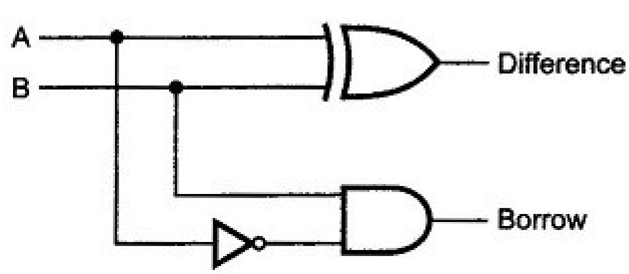
For Difference

For Borrow

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

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

Logic diagram



b.Full subtractor circuit and explain its operation with truth table.:

Full subtraction is a combinational circuit which consists of three binary input variables called minuends and subtrahends and two binary output variables called difference and borrow out. In the subraction result, the lower significant bit is called as difference and the higher significant bit is called as borrow out

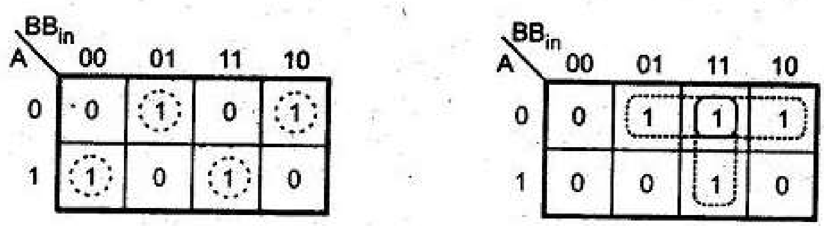
Truth table

Inputs			Outputs	
A	B	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table 3.9 Truth table for full-subtractor

K-map Simplification of Difference (D) and Borrow (B)

For D **For B_{out}**



$D = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + ABB_{in}$

$B_{out} = \bar{A}B_{in} + \bar{A}B + BB_{in}$

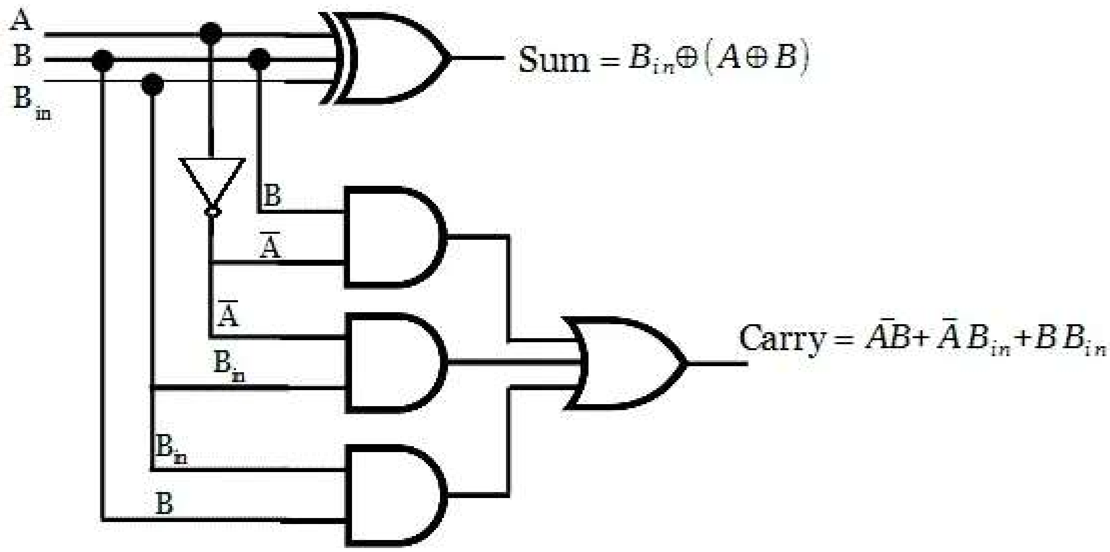
$$\text{Difference} = \bar{A}\,\bar{B}B_{in} + \bar{A}\,B\,\bar{B}_{in} + A\,\bar{B}\,\bar{B}_{in} + A\,B B_{in}$$

$$= B_{in}(\bar{A}\,\bar{B} + A\,B) + \bar{B}_{in}(\bar{A}\,B + A\,\bar{B})$$

$$= B_{in}(A \odot B) + \bar{B}_{in}(A \oplus B)$$

$$= B_{in}(\overline{A \oplus B}) + \bar{B}_{in}(A \oplus B)$$

$$= B_{in} \oplus (A \oplus B)$$



Logic circuit for Full subtractor

2.5 Realize full-subtraction using two Half-subtractor and an OR – gate and write truth table.

The full subtractor can be implemented with two half subtractors by cascading them. The difference output of first half subtractor is Ex-OR of A and B. The difference output of full subtractor is Ex-OR of Bin and output of first half subtractor.

Similarly, the borrow output of first half subtractor is ORed with the borrow output of second half subtractor to get the borrow output of full subtractor.

Simplification of Difference and Borrow

$$\text{Difference} = \bar{A}\,\bar{B}B_{in} + \bar{A}\,B\,\bar{B}_{in} + A\,\bar{B}\,\bar{B}_{in} + A\,BB_{in}$$

$$= B_{in}(\bar{A}\,\bar{B} + A\,B) + \bar{B}_{in}(\bar{A}\,B + A\,\bar{B})$$

$$= B_{in}(A \odot B) + \bar{B}_{in}(A \oplus B)$$

$$= B_{in}(\overline{A \oplus B}) + \bar{B}_{in}(A \oplus B)$$

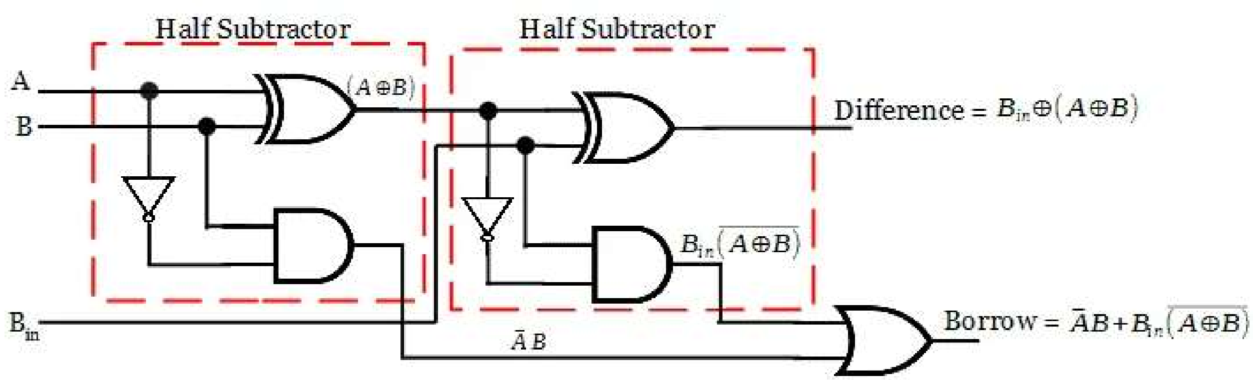
$$= B_{in} \oplus (A \oplus B)$$

$$
\begin{aligned}
\text{Borrow} &= \bar{A}\,B+\bar{A}\,B_{in}+B\,B_{in} \\
&= \bar{A}\,B+\bar{A}\,B_{in}(B+\bar{B})+B\,B_{in}(A+\bar{A}) \\
&= \bar{A}\,B+\bar{A}\,B\,B_{in}+\bar{A}\,\bar{B}\,B_{in}+A\,B\,B_{in}+\bar{A}\,B\,B_{in} \\
&= \bar{A}\,B(1+B_{in}+B_{in})+\bar{A}\,\bar{B}B_{in}+A\,B\,B_{in} \\
&= \bar{A}\,B+\bar{A}\,\bar{B}\,B_{in}+A\,B\,B_{in} \\
&= \bar{A}\,B+B_{in}(\bar{A}\,\bar{B}+A\,B) \\
&= \bar{A}\,B+B_{in}(A\odot B) \\
&= \bar{A}\,B+B_{in}\overline{(A\oplus B)}
\end{aligned}
$$

$A+A=A$

$A+1=1$

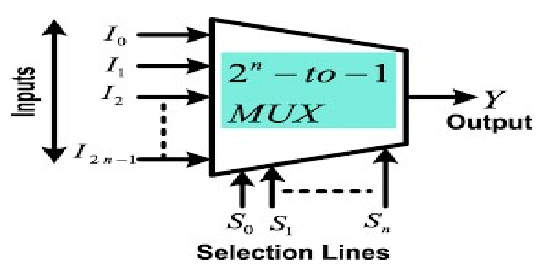
Using the simplified boolean expressions for difference and borrow output, the full subtractor can be realized



Realization of full subtractor with two half subtractors

2.7 Operation of 4 X 1 Multiplexers and 1 X 4 demultiplexer

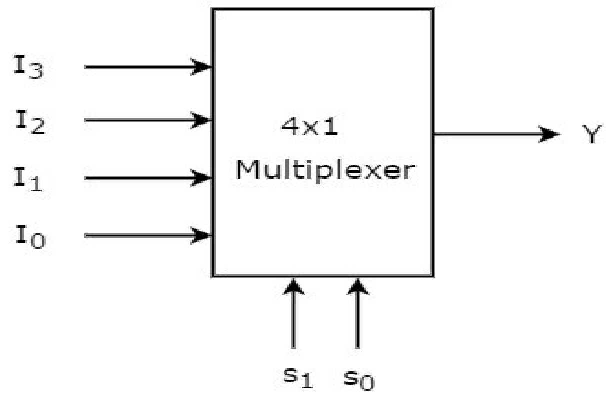
Multiplexer is a combinational circuit that has maximum of 2^n number data inputs, 'n' number of selection control lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines. Shown in figure.



Where I_0, I_1, I_3, I_4.......I_n are the input line, Y is the out put line and S_0, S_1,.....Sn are the selection line.

a. 4x1 Multiplexer

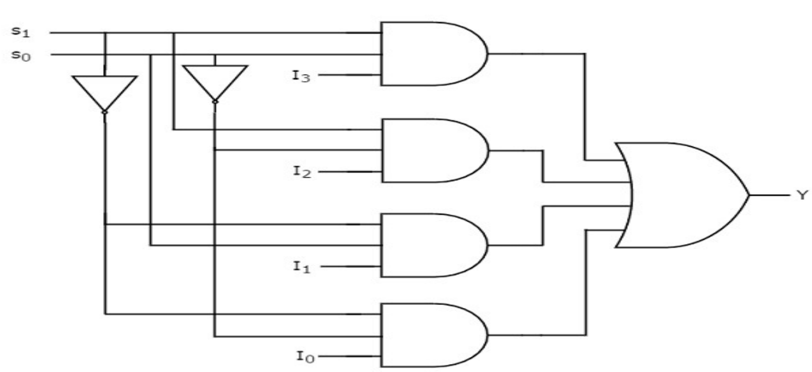
4x1 Multiplexer has four data inputs I_3, I_2, I_1 & I_0, two selection lines s_1 & s_0 and one output Y. The **block diagram** of 4x1 Multiplexer is shown in the following figure.



One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. Truth table of 4x1 Multiplexer is shown below.

Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

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



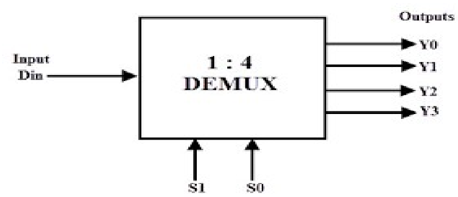
Logic circuit diagram

De-Multiplexer :

De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, 'n' selection lines and maximum of 2^n outputs. De-Multiplexer is also called as **De-Mux**.

1x4 De-Multiplexer

1x4 De-Multiplexer has one input I, two selection lines, s1 & s0 and four outputs Y3, Y2, Y1 &Y0. The block diagram of 1x4 De-Multiplexer is shown in the following figure.



The single input 'I' will be connected to one of the four outputs, Y3 to Y0 based on the values of selection lines s1 & s0. The Truth table of 1x4 De-Multiplexer is shown below.

Selection inputs		outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

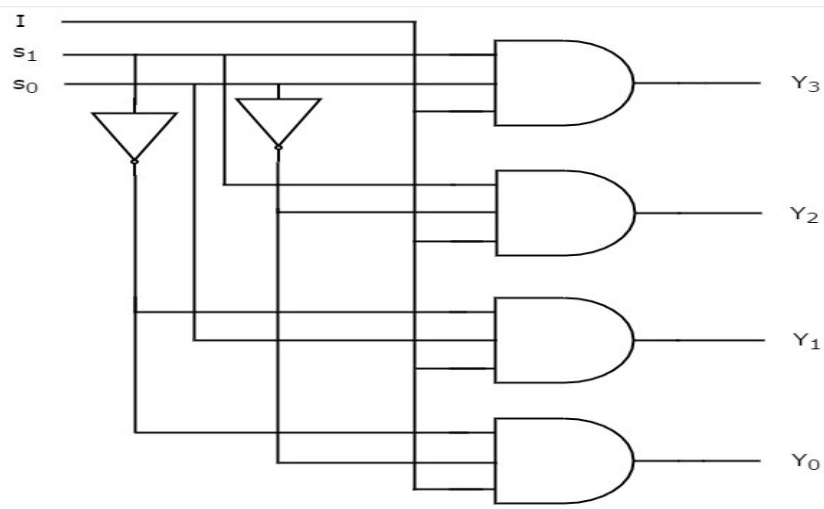
From the above Truth table, we can directly write the **Boolean functions** for each output as

$$Y_0 = I\,\overline{S_1}\ \overline{S_0}$$

$$Y_1 = I\,\overline{S_1}\ S_0$$

$$Y_2 = I\,S_1\,\overline{S_0}$$

$$Y_3 = I\,S_1\,S_0$$



Logic circuit diagram

2.8 Working of Binary-Decimal Encoder & 3 X 8 Decoder.

a. **Decoder**

Decoder is a combinational circuit that has multiple input multiple output that is 'n' number of input lines and maximum of 2^n number of output lines.

One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. i.e In the decoder, the combination of input information lines define the logic output of any one.

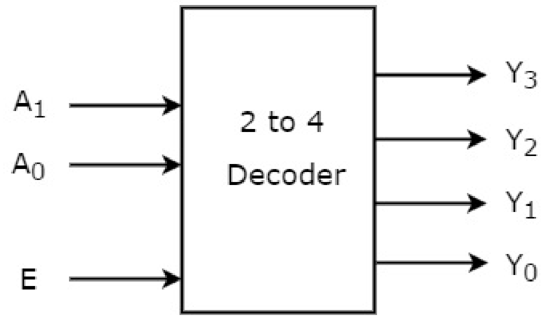
output line as logic high at a time and the rest of the output lines are being fixed to logic 0. The outputs of the decoder are nothing but the min terms of 'n' input variables lines, when it is enabled.

2 to 4 Decoder

Let 2 to 4 Decoder has two inputs A_1 & A_0 and four outputs Y_3, Y_2, Y_1 & Y_0. The **block diagram** of 2 to 4 decoder is shown in the following figure.

i.e input lines 'n'=2

output lines=2^n=2^2=4



One of these four outputs will be '1' for each combination of inputs when enable, E is '1'. The **Truth table** of 2 to 4 decoder is shown below.

Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

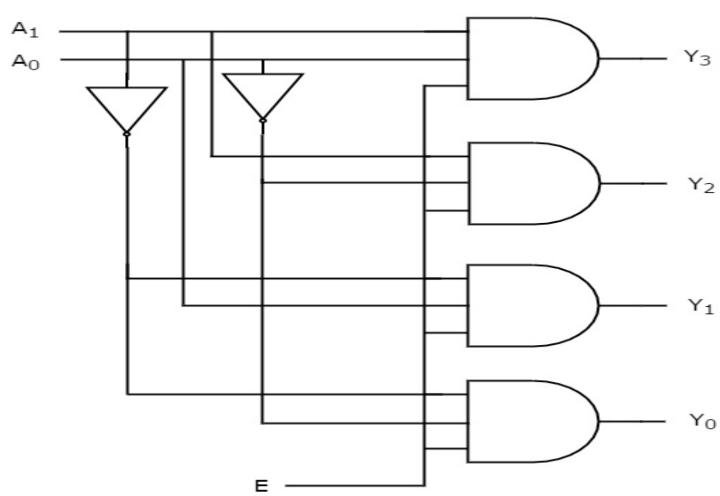
From Truth table, we can write the **Boolean functions** for each output as

$$Y_0 = E\,\overline{A_1}\,\overline{A_0}$$

$$Y_1 = E\,\overline{A_1}\,A_0$$

$$Y_2 = E\,A_1\,\overline{A_0}$$

$$Y_3 = E\,A_1\,A_0$$



3 to 8 Decoder

Let 3 to 8 Decoder has 3 inputs A_2 A_1 & A_0 and 8 outputs Y_7, Y_6, Y_5, Y_4 ,Y_3, Y_2, Y_1 & Y_0. The **block diagram** of 3 to 8 decoder is shown in the following figure.

i.e input lines 'n'=3

output lines=$2^n=2^3=8$

Enable	Inputs			Outputs							
E	A_3	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

From Truth table, we can write the **Boolean functions** for each output as

$$Y_0=E\,\overline{A}_2\,\overline{A}_1\,\overline{A}_0$$

$$Y_1=E\,\overline{A}_2\,\overline{A}_1\,A_0$$

$$Y_2=E\,\overline{A}_2\,A_1\,\overline{A}_0$$

$$Y_3=E\,\overline{A}_2\,A_1\,A_0$$

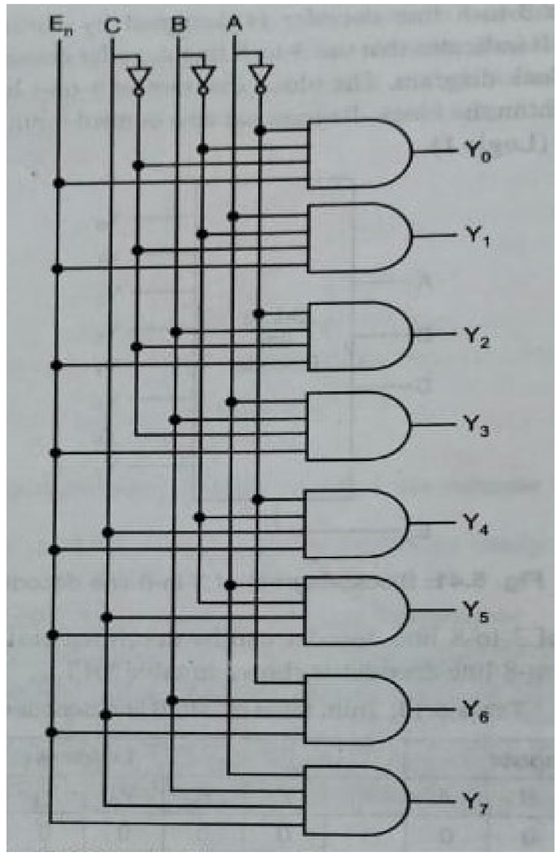
$$Y_4=E\,A_2\,\overline{A}_1\,\overline{A}_0$$

$$Y_5=E\,A_2\,\overline{A}_1\,A_0$$

$$Y_6=E\,A_2\,A_1\,\overline{A}_0$$

$$Y_7=E\,A_2\,A_1\,A_0$$

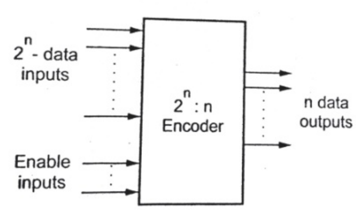
Logical circuit of the above expressions is given below:



Logic diagram 3 to 8 line decoder

Encoder:

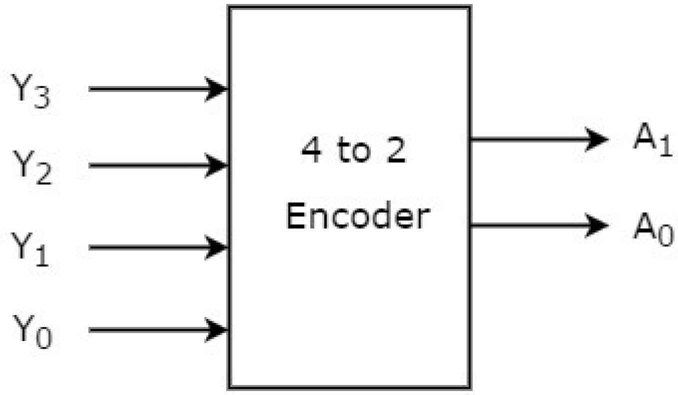
An encoder is a multiple input multi output combinational digital circuit that performs the inverse operation of a decoder. It means that an encoder converts the 2^n number of coded inputs into n number of coded outputs.



The output lines of a digital encoder generate the binary equivalent of the input line whose value is equal to 1 and are available to encode either a decimal or hexadecimal input pattern to typically a binary or B.C.D (binary coded decimal) output code

4 to 2 line Encoder:

There are four inputs (Y0, Y1, Y2, and Y3) and two outputs (A0 and A1) in the 4 to 2 line encoder. In addition, To get the respective binary code on the output side, one input line at a time is set to true in a 4-input line. The 4 to 2 line encoder's block diagram and truth table are shown below.



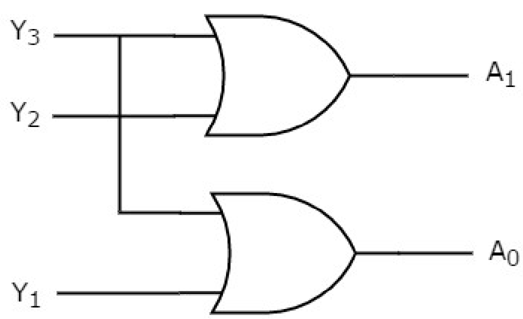
Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

The terms A_0 and A_1 are logically expressed as follows:

$$A_1=Y_3+Y_2$$
$$A_0=Y_3+Y_1$$

Circuit Diagram

Two input OR gates can be used to implement the aforementioned two Boolean functions. Further, The 4 to 2 encoder circuit diagram is given in the graphic below.



Uses of Encoder

In all digital systems, these systems are relatively simple to operate.

To convert a decimal number to a binary number, encoders are employed. The goal is to complete a binary operation like addition, subtraction, multiplication, and so on.

Disadvantages

The disadvantages of a standard encoder are listed below.

- When all of the encoder's outputs are 0, there is ambiguity. Because when only the least significant input is one or when all inputs are zero, it could be the code matching the inputs.
- When more than one input is set to high, the encoder generates an output that may or may not be the proper code. If both Y3 and Y6 are '1', for example, the encoder outputs 111. This is neither the comparable code for Y3, when it is '1', nor is it the equivalent code for Y6, when it is '1'.

2.9 Working of Two bit magnitude comparator.

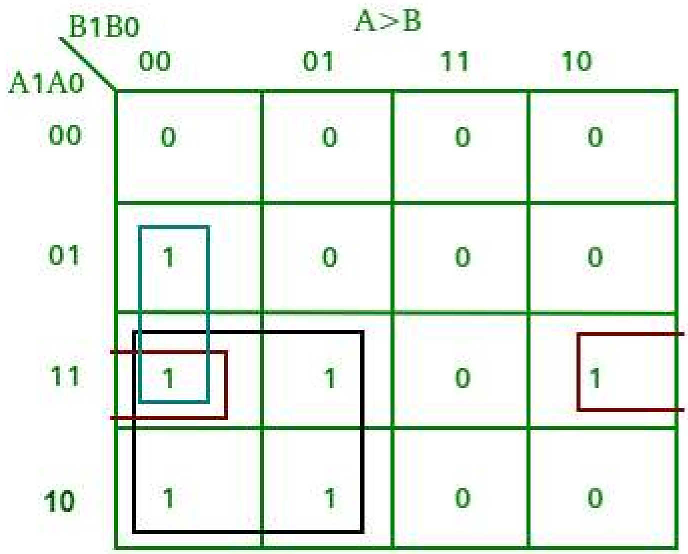
A magnitude digital Comparator is a combinational circuit that **compares two digital or binary numbers** in order to find out whether one binary number is equal, less than, or greater than the other binary number. We logically design a circuit for which we will have two inputs one for A and the other for B and have three output terminals, one for A > B condition, one for A = B condition, and one for A < B condition.

Truth table

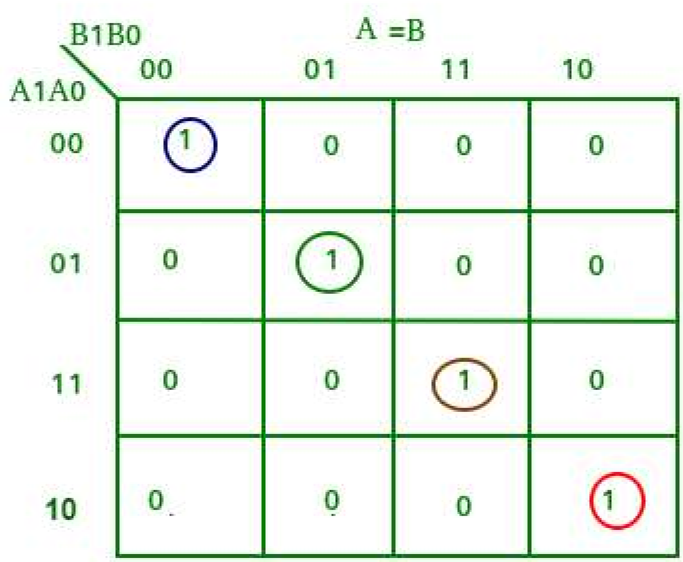
Input				Out put		
A		B		A>B	A=B	A<B
A_1	A_0	B_1	B_0			
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1

0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

From the above truth table K-map for each output can be drawn as follows:



$A>B = A_1A_0\bar{B}_0 + \bar{B}_1\bar{B}_0A_0 + A_1\bar{B}_1$

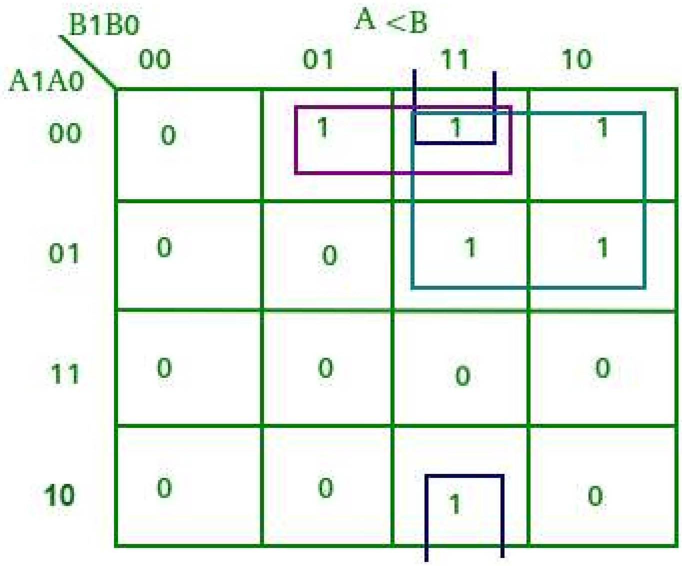


$A=B \quad = \bar{A}_1\bar{A}_0\bar{B}_1\bar{B}_0 + \bar{A}_1A_0\bar{B}_1B_0 + A_1A_0B_1B_0 + A_1\bar{A}_0B_1\bar{B}_0$

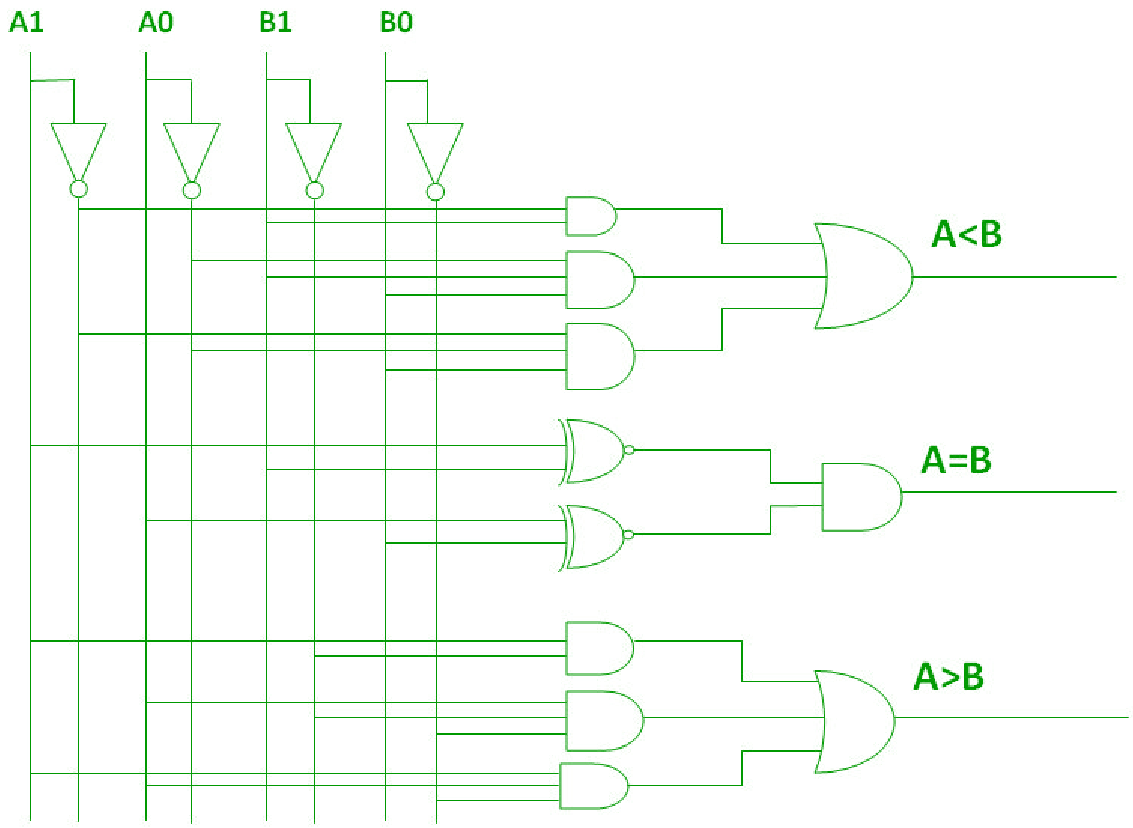
$=(\bar{A}_1\,\overline{B_1}(\bar{A}_0\bar{B}_0 + A_0\,B_0\,) + A_1B_1(A_0B_0 + \bar{A}_0\bar{B}_0)$

$=(\bar{A}_1\,\overline{B_1} + A_1B_1)(A_0B_0 + \bar{A}_0\bar{B}_0)$

$=(A_1 \odot B_1)(A_0 \odot B_0)$

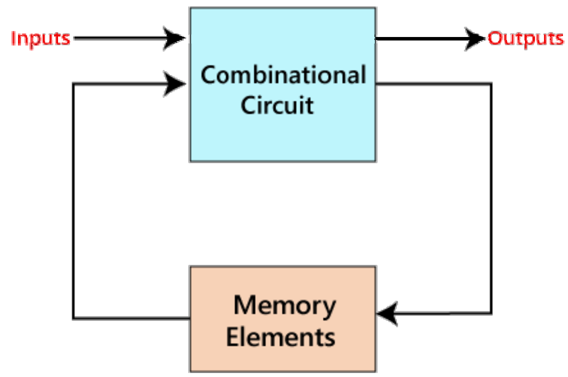


A<B $=\bar{A}_1\bar{A}_0B_1 + B_1B_0\bar{A}_0 + \bar{A}_1B_1$


A1
A0
B1
B0
A<B
A=B
A>B

3.SEQUENTIAL LOGIC CIRCUITS

The outputs of the sequential circuits depend on both the combination of present inputs and previous outputs. The previous output is treated as the present state. So, the sequential circuit contains the combinational circuit and its memory storage elements. A sequential circuit doesn't need to always contain a combinational circuit. So, the sequential circuit can contain only the memory element.



BLOCK DIAGRAM OF SEQUENTIAL CKT

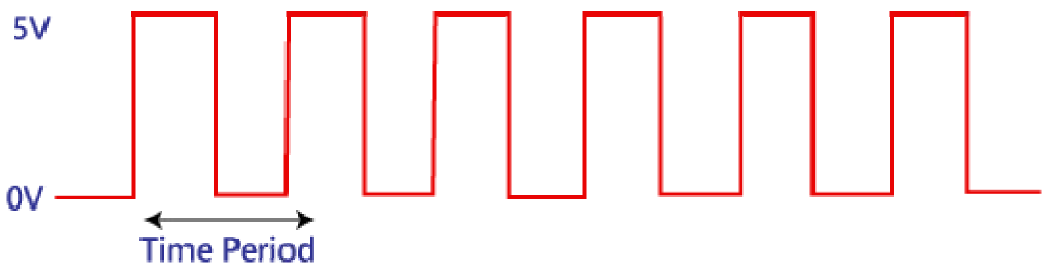
Difference between the combinational circuits and sequential circuits are given below:

	Combinational Circuits	Sequential Circuits
1	The outputs of the combinational circuit depend only on the present inputs	The outputs of the sequential circuits depend on both present inputs and present state(previous output).
2	The feedback path is not present in the combinational circuit.	The feedback path is present in the sequential circuits.
3	In combinational circuits, memory elements are not required.	In the sequential circuit, memory elements play an important role and require.
4	The clock signal is not required for combinational circuits.	The clock signal is required for sequential circuits.
5	The combinational circuit is simple to design.	It is not simple to design a sequential circuit.

3.2 State the necessity of clock and give the concept of level clocking and edge triggering,

1. **Clock:**

A clock signal is a periodic signal in which ON time and OFF time need not be the same. When ON time and OFF time of the clock signal are the same, a square wave is used to represent the clock signal. Below is a diagram which represents the clock signal:



A clock signal is considered as the square wave. Sometimes, the signal stays at logic, either high 5V or low 0V, to an equal amount of time. It repeats with a certain time period, which will be equal to twice the 'ON time' or 'OFF time'.

Types of Triggering

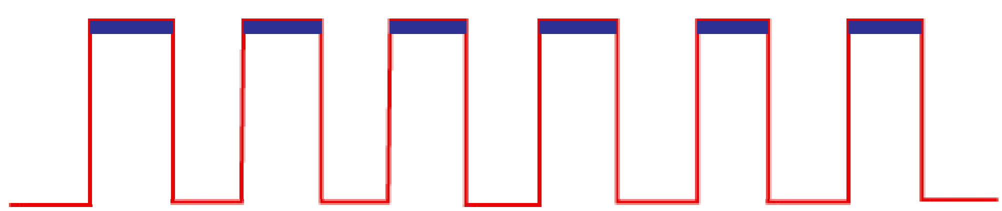
These are two types of triggering in sequential circuits:

Level triggering

The logic High and logic Low are the two levels in the clock signal. In level triggering, when the clock pulse is at a particular level, only then the circuit is activated. There are the following types of level triggering:

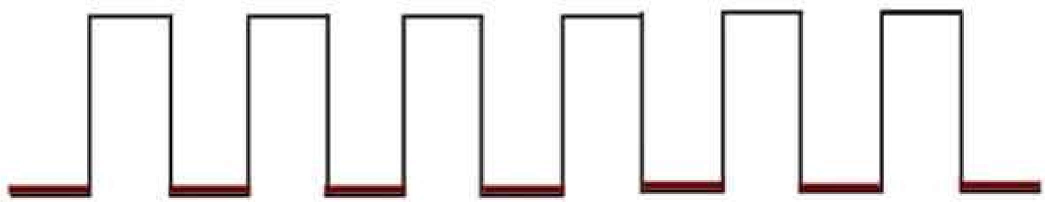
Positive level triggering

In a positive level triggering, the signal with Logic High occurs. So, in this triggering, the circuit is operated with such type of clock signal. Below is the diagram of positive level triggering:



Negative level triggering

In negative level triggering, the signal with Logic Low occurs. So, in this triggering, the circuit is operated with such type of clock signal. Below is the diagram of Negative level triggering:



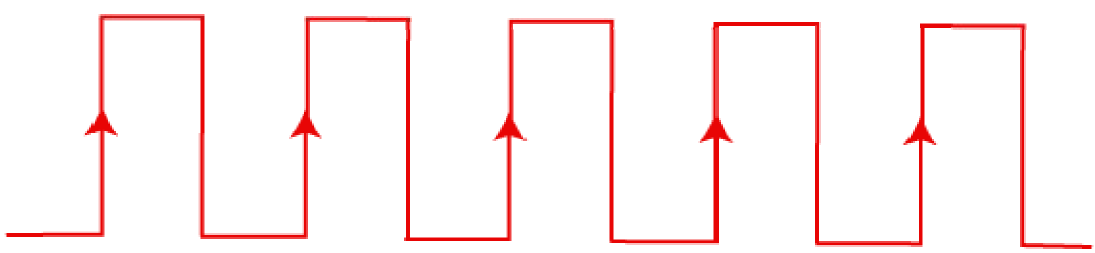
Edge triggering

In clock signal of edge triggering, two types of transitions occur, i.e., transition either from Logic Low to Logic High or Logic High to Logic Low.

Based on the transitions of the clock signal, there are the following types of edge triggering:

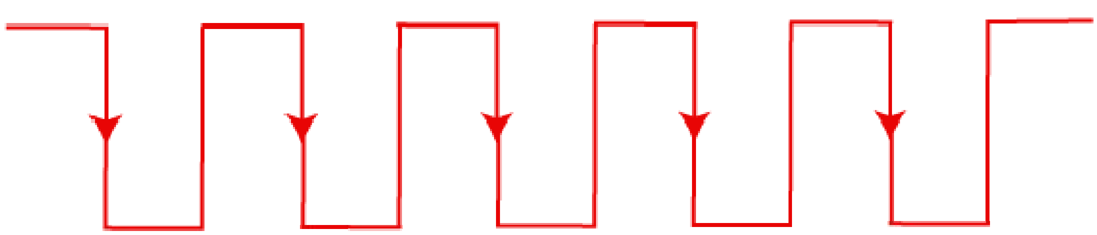
Positive edge triggering

The transition from Logic Low to Logic High occurs in the clock signal of positive edge triggering. So, in positive edge triggering, the circuit is operated with such type of clock signal. The diagram of positive edge triggering is given below.



Negative edge triggering

The transition from Logic High to Logic low occurs in the clock signal of negative edge triggering. So, in negative edge triggering, the circuit is operated with such type of clock signal. The diagram of negative edge triggering is given below.

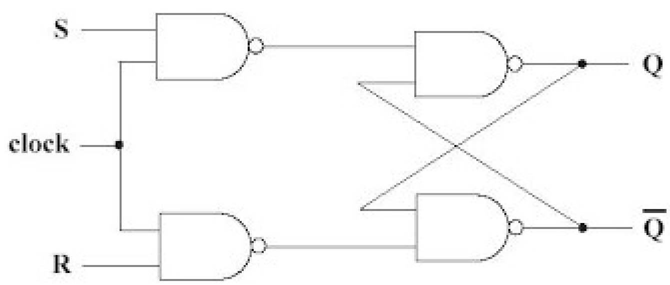


What is flip flop ?

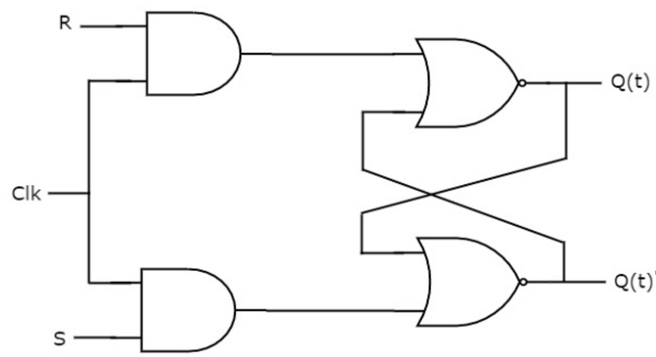
Flip-Flop is popularly known as the basic digital memory circuit. It is an edge triggered synchronous sequential logic circuit that is capable of storing single bit binary information. It has two states as logic 1(High) and logic 0(low) states. A flip flop is a sequential circuit which consists of a single binary state of information or data. The digital circuit is a flip flop which has two outputs and are of opposite states. It is also known as a Bistable Multivibrator.

3.3 Clocked SR flip flop

SR (Set-Reset) flip-flop is a clocked sequential circuit which is controlled by edge triggered CLK control signal.



Logic diagram using NAND gate



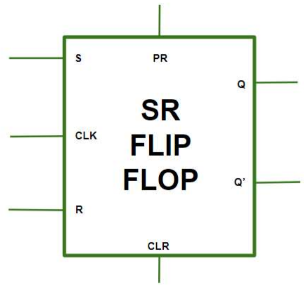
Logic diagram using AND and NOR gate

Truth Table

Inputs			Outputs		States
CLK	S	R	Q	$\bar{Q}$	
0	0	0	NC	NC	No. change
0	0	1	NC	NC	No. change
0	1	0	NC	NC	No. change
0	1	1	NC	NC	No. change
1	0	0	NC	NC	No. change
1	0	1	0	1	Reset
1	1	0	1	0	Set
1	1	1	X	X	No. change

3.3 Clocked SR flip flop with preset and clear inputs.

In SR flip flop, with the help of Preset and Clear, when the power is switched ON, the state of the circuit keeps on changing, i.e. it is uncertain. It may come to Set (Q = 1) or Reset (Q' = 0) state. In many applications, it is desired to initially Set or Reset the flip flop. This thing is accomplished by the Preset (PR) and the Clear (CLR).



BLOCK DIAGRAM OF F/F

Operations in SR Flip-Flop –

- **Case-1:**
  PR = CLR = 1

  The asynchronous inputs are inactive and the flip flop responds freely to the S, R and the CLK inputs in the normal way.

- **Case-2:**
  PR = 0 and CLR = 1

  This is used when the Q is set to 1.

- **Case-3:**
  PR = 1 and CLR = 0

  This is used when the Q' is set to 1.

- **Case-4:**
  PR = CLR = 0

  This is an invalid state.

INPUTS					OUTPUTS		Comments
PR	CLR	CLK	S	R	$Q_{(n+1)}$	$\bar{Q}_{(n+1)}$	
0	1	NA	NA	NA	1	0	Set
1	0	NA	NA	NA	0	1	Re-set
1	1	0	NA	NA	Q_n	$\bar{Q}_n$	No. change

1	1	1	0	0	Q_n	$\bar{Q}_n$	No. change
1	1	1	1	0	1	0	Set
1	1	1	0	1	0	1	Re-set
1	1	1	1	1	x	x	Not allowed

Applications of Flip-Flop :

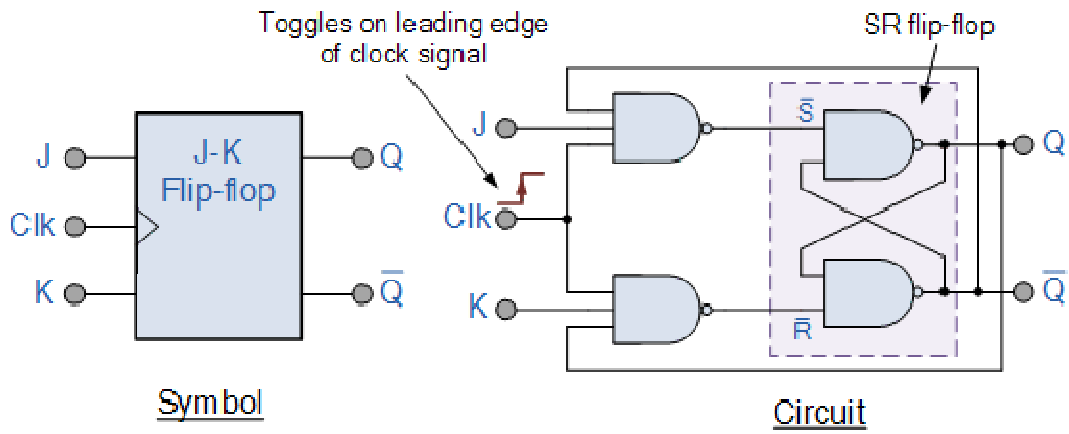
1. Flip flops are used as a bounce elimination switch.
2. They are used as a serial to parallel and parallel to serial conversion.
3. It is used for counters.
4. It is used for frequency divider and also as a latch.

3.5 Construct level clocked JK flip flop using S-R flip-flop and explain with truth table

The JK flip flop is one of the most used flip flops in digital circuits. The JK flip flop is a universal flip flop having two inputs 'J' and 'K'. In SR flip flop, the 'S' and 'R' are the shortened abbreviated letters for Set and Reset, but J and K are not. The J and K are themselves autonomous letters which are chosen to distinguish the flip flop design from other types. JK flip-flop can either be triggered upon the leading-edge of the clock or on its trailing edge and hence can either be positive- or negative- edge-triggered, respectively.

The JK flip flop work in the same way as the SR flip flop work. The JK flip flop has 'J' and 'K' flip flop instead of 'S' and 'R'. The only difference between JK flip flop and SR flip flop is that when both inputs of SR flip flop is set to 1, the circuit produces the invalid states as outputs, but in case of JK flip flop, there are no invalid states even if both 'J' and 'K' flip flops are set to 1.

The JK Flip Flop is a gated SR flip-flop having the addition of a clock input circuitry. The invalid or illegal output condition occurs when both of the inputs are set to 1 and are prevented by the addition of a clock input circuit. So, the JK flip-flop has four possible input combinations, i.e., 1, 0, "no change" and "toggle".



TRUTH TABLE

INPUTS		OUTPUT	STATES
J	K	Q^+	
0	0	Q	Previous state
0	1	0	Re-set
1	0	1	Set
1	1	$\bar{Q}$	Toggles(Complement of present state)

[Type the document title]

[Type the document subtitle]

leena marndi

Microprocessor

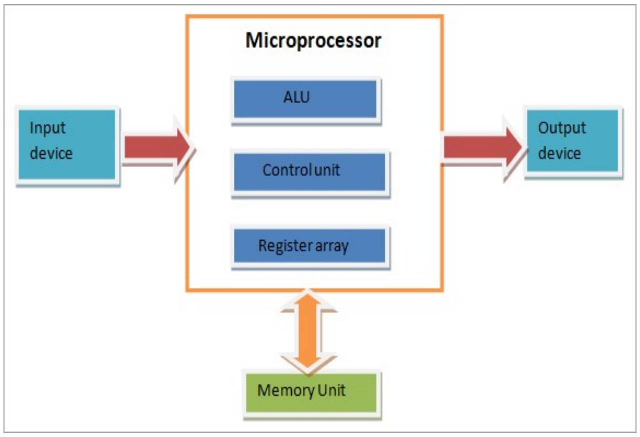
A microprocessor is a multipurpose, programmable, clock driven register based semiconductor device that read binary instructions from memory, accept binary data as input and process data according to instruction and provide result as output

It is a kind of integrated circuit (IC) unit which combines all the basic functions of a central processing unit (CPU) of the computer.

It is a programmable unit that is fabricated on the silicon chip and it consists of an ALU unit, clock, and control unit and register array which accepts the input in binary form (0's and 1's) and delivers the output after processing the input data as per the instructions fetched into the memory unit

Microcomputer

A digital computer in which one microprocessor has been provided to act as a CPU is called microcomputer



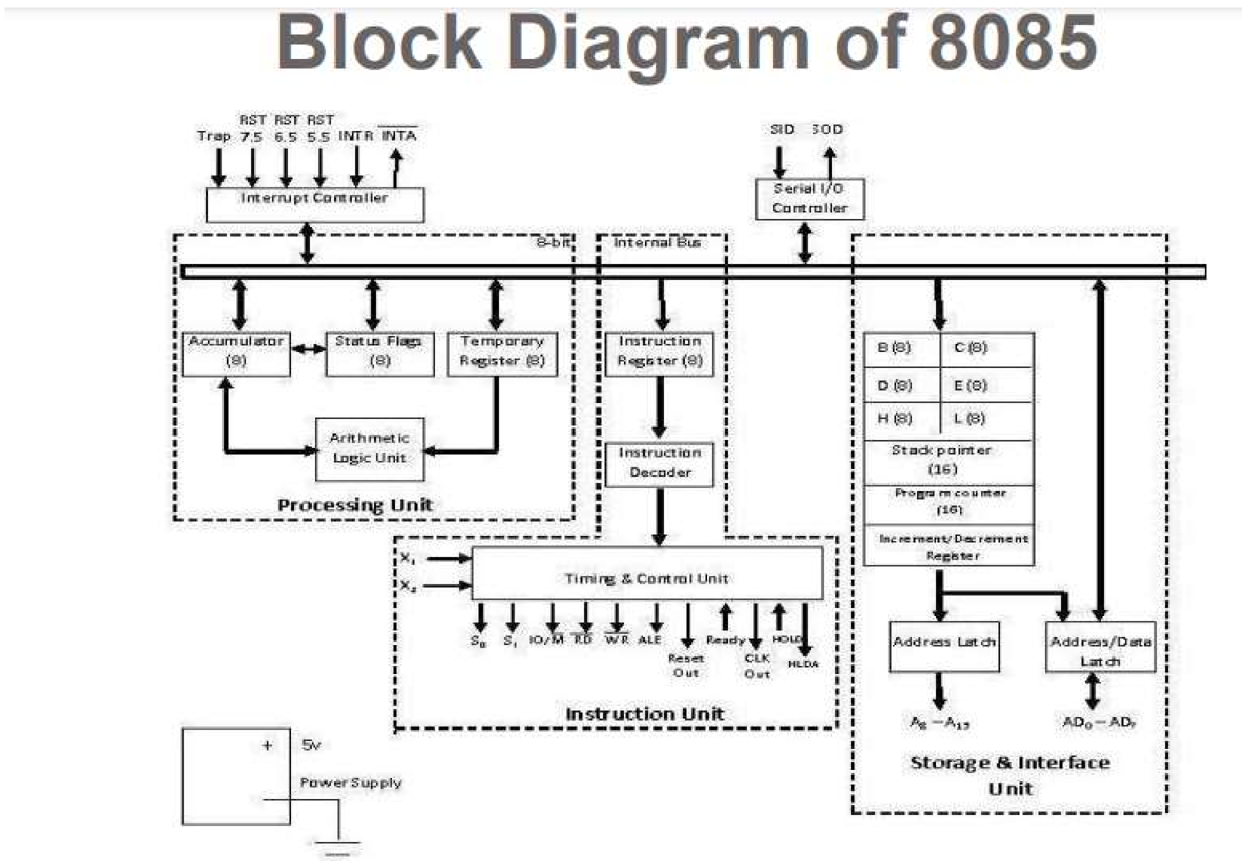
The basic building blocks of this processor are an ALU, register array, and the main control processing unit. The function of the arithmetic logical unit (ALU) is to perform the mathematical and logical operations based on the data fetched from the input units or the memory device.

Some important terms

- **Bit:** A digit of the binary number of code is called a bit
- **Nibble**: The 4bit (digit) binary number or code is called a nibble
- **Byte:** 8 bit binary no. is called Byte
- **Word:** 16 bit binary no. is called byte

Architecture of Intel 8085A Microprocessor and description of each block.

- It is a 40 pin I.C. package fabricated on a single LSI chip.
- The Intel 8085 uses a single +5Vd.c. supply for its operation.
- Intel 8085 is clock speed is about 3 MHz; the clock cycle is of 320ns.
- 8 bit data bus.
- Address bus is of 16-bit, which can address up to 64KB
- It has 80 basic instructions and 246 opcodes.



It consists of 3(Three) main section these are as follows

1. **Arithmetic &Logic Unit**
2. **Timing and Control unit**
3. **Sets of Register**

1. Arithmetic & Logic Unit

The arithmetic and logic unit performs the following arithmetic and logic operation

i) Addition

ii) Subtraction

iii) Logical AND

iv) Logical OR

v) Logical Exclusive or

vi) Increment

vii) Decrement

2. Timing and Control unit

The timing and control unit comes under the section of CPU, and it generates the timing and control signals which are necessary for the execution of Instructions. It controls flow of data from CPU to other devices. It provides status, control and timing signals which are required for the operation of memory and I/O device .It is also used to control the operations performed by the microprocessor and the devices connected to it. There are certain timing and control signals like: Control signals, DMA Signals, RESET signals, Status Signal.

3. Sets of Register

Registers are used for temporary storage and manipulation of data and instructions by the microprocessor. Data remain in the registers till they are sent to the I/O devices or memory. Intel 8085 microprocessor has the following registers:

a) One 8-bit accumulator (ACC) i.e. register A
b) Six general purpose registers of 8-bit, these are B,C, D, E, H and L
c) One 16-bit stack pointer, SP
d) One 16-bit Program Counter, PC
e) Instruction register
f) Temporary register

In addition to the above mentioned registers the 8085 microprocessor contains a set of five flip-flops which serve as flags (or status flags).

A flag is a flip-flop which indicates some conditions which arises after the execution of an arithmetic or logical instruction.

a) Accumulator (ACC):

The accumulator is an 8-bit register associated with the ALU. The register 'A' is an accumulator in the 8085. It is used to hold one of the operands of an arithmetic and logical operation. The final result of an arithmetic or logical operation is also placed in the accumulator.

b) General-Purpose Registers:

The 8085 microprocessor contains six 8-bit general purpose registers. They are: B, D, C, E, H and L register.

To hold data of 16-bit a combination of two 8-bit registers can be employed.

The combination of two 8-bit registers is called register pair.

The valid register pairs in the 8085 are: D-E, B-C and H-L. The H-L pair is used to act as a memory pointer.

c) Stack Pointer (SP):

It is a 16-bit special function register used as memory pointer. A stack is nothing but a portion of RAM i.e. it is sequence of memory location set aside by a programmer to store/ retrieve the content of accumulator, flags, program counter and general-purpose register during the execution of a program.

Stack work on LIFO(last in first out) Principle

Its operation is faster compared normal store / retrieve of memory location

The stack pointer (SP) controls the addressing of the stack. The Stack Pointer contains the address of the top element of data stored in the stack.

d) Program Counter (PC):

It is a 16-bit special purpose register. It is used to hold the address of memory of the next instruction to be executed. It keeps the track of the instruction in a program while they are being executed. The microprocessor increments the content of the next program counter during the execution of an instruction so that at the end of the execution of an instruction it points to the next instructions address in the program.

e) Instruction register

The instruction register holds the opcode (operation code or instruction code) of the instruction which is being decoded and executed.

f) Temporary register

It is an 8-bit register associated with the ALU. It holds data during an arithmetic/logical operation. It is used by the microprocessor. It is not accessible to programmer.

g) Flags:

The Intel 8085 microprocessor contains five flip-flops to serve as a status flags. The flip-flops are reset or set according to the conditions which arise during an arithmetic or logical operation.

a. Carry Flag (CS)
b. Parity Flag (P)
c. Auxiliary Carry Flag (AC)
d. Zero Flag(Z)
e. Sign Flag(S)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z	X	AC	X	P	X	CS

a) Carry Flag (CS)

Carry is generated when performing n bit operations and the result is more than n bits, then this flag becomes set i.e. 1, otherwise it becomes reset i.e. 0. During subtraction (A-B), if A>B it becomes reset and if (A<B) it becomes set. Carry flag is also called borrow flag.

1-carry out from MSB bit on addition or borrow into MSB bit on subtraction
0-no carry out or borrow into MSB bit

Example:

MVI A 30 (load 30H in register A)
MVI B 40 (load 40H in register B)
SUB B (A = A – B)
These set of instructions will set the carry flag to 1 as 30 – 40 generates a carry/borrow.

MVI A 40 (load 40H in register A)
MVI B 30 (load 30H in register B)
SUB B (A = A – B)
These set of instructions will reset the sign flag to 0 as 40 – 30 does not generate any carry/borrow.

b) Parity Flag (P)

If after any arithmetic or logical operation the result has even parity, an even number of 1 bits, the parity register becomes set i.e. 1, otherwise it becomes reset i.e. 0.

1-accumulator has even number of 1 bits
0-accumulator has odd parity

Example:

MVI A 05 (load 05H in register A)
This instruction will set the parity flag to 1 as the BCD code of 05H is 00000101, which contains even number of ones i.e. 2.

c) Auxiliary Carry Flag (AC)

This flag is used in BCD number system(0-9). If after any arithmetic or logical operation D(3) generates any carry and passes on to B(4) this flag becomes set i.e. 1, otherwise it becomes reset i.e. 0. This is the only flag register which is not accessible by the programmer1-carry out from bit 3 on addition or borrow into bit 3 on subtraction 0-otherwise

Example:

MOV A 2B (load 2BH in register A)
MOV B 39 (load 39H in register B)
ADD B (A = A + B)

These set of instructions will set the auxiliary carry flag to 1, as on adding 2B and 39, addition of lower order nibbles B and 9 will generate a carry.

d) Zero Flag(Z)

After any arithmetical or logical operation if the result is 0 (00)H, the zero flag becomes set i.e. 1, otherwise it becomes reset i.e. 0.
00H zero flag is 1.
from 01H to FFH zero flag is 0

1- zero result
0- non-zero result

Example:

MVI A 10 (load 10H in register A)
SUB A (A = A – A)
These set of instructions will set the zero flag to 1 as 10H – 10H is 00H

e) Sign Flag(S)

After any operation if the MSB (B(7)) of the result is 1, it indicates the number is negative and the sign flag becomes set, i.e. 1. If the MSB is 0, it indicates the number is positive and the sign flag becomes reset i.e. 0.
from 00H to 7F, sign flag is 0
from 80H to FF, sign flag is 1

1- MSB is 1 (negative)
0- MSB is 0 (positive)

Example:

MVI A 30 (load 30H in register A)
MVI B 40 (load 40H in register B)
SUB B (A = A – B)
These set of instructions will set the sign flag to 1 as 30 – 40 is a negative number.

MVI A 40 (load 40H in register A)
MVI B 30 (load 30H in register B)
SUB B (A = A – B)
These set of instructions will reset the sign flag to 0 as 40 – 30 is a positive number.

Pin Diagram 8085 microprocessor

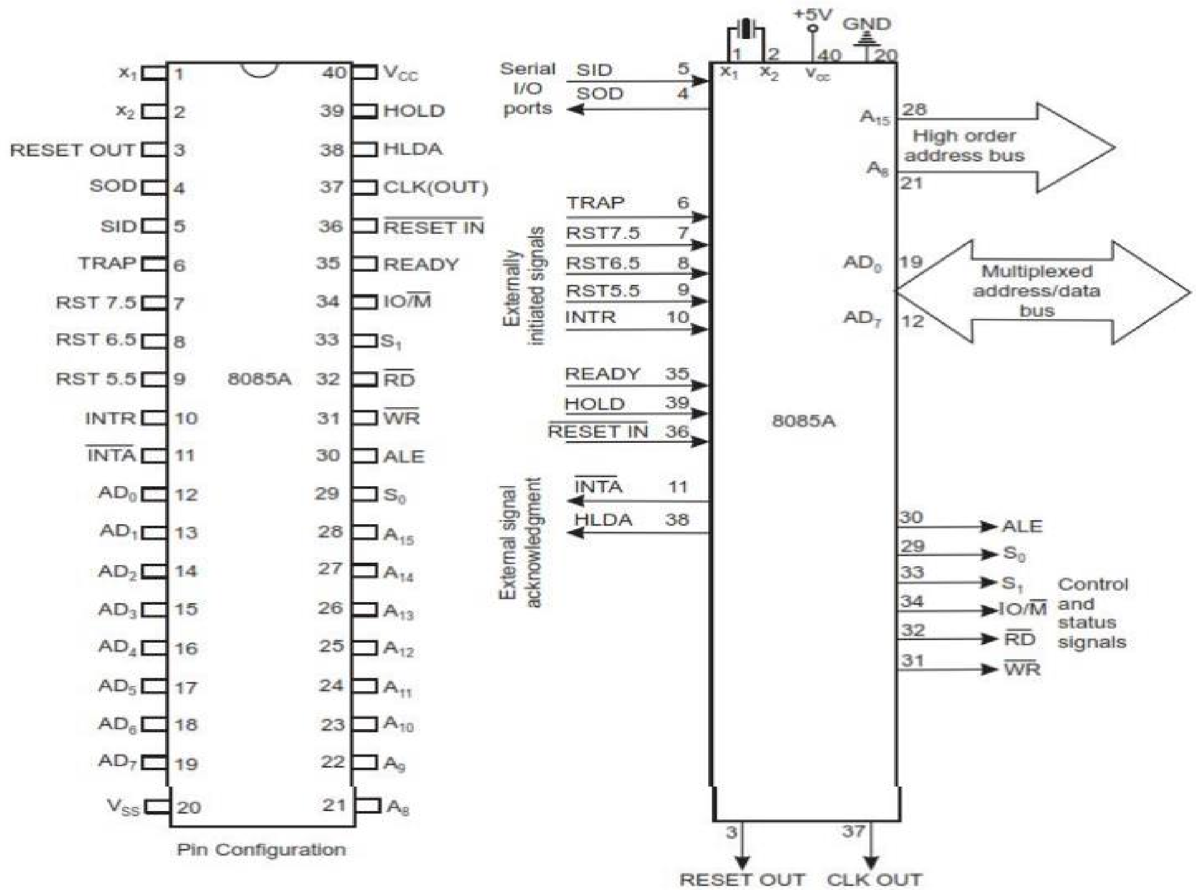


Fig 1.2 Pin Diagram of 8085

A8 - A15 (Output):

These are the Address Bus and used for most significant 8 bits of the memory address or the 8 bits of the I/0 address,

AD0 - AD7 (Input / Output)

Multiplexed Address/Data Bus it serve dual purpose. They are used for Least significant 8 bits of the memory address (or I/0 address) during the first clock cycle of a machine cycle. Then it becomes the data bus during the second and third clock cycles.

ALE (Output):

Address Latch Enable signal it goes high during the first clock cycle of a machine cycle and enables the lower 8 bit address to get latched either into the memory or external latch So when pulse goes high means ALE=1, it makes address bus enable and when ALE=0, means low pulse makes data bus enable.

IO/$\overline{M}$ (Output):

It is a status signal which distinguishes whether I/O or memory operation is being performed

When it goes high, the address on the address bus is for an I/O device.

i.e If IO/$\overline{\mathrm{M}}$ = 1 then I/O operation is being performed.

When it goes low, the address on the address bus is for an memory location

i.e If IO/$\overline{\mathrm{M}}$ = 0 then ◦ Memory operation is being performed.

SO, S1 (Output):

These are the status signals sent by the microprocessor to distinguish the various type of operation

S1	S0		
0	0	HALT	
0	1	WRITE	
1	0	READ	
1	1	FETCH	S1 can be used as an advanced R/W status.

$\overline{RD}$ (Output):

RD stands for Read.

It is an active low signal. i.e $\overline{\mathrm{RD}}$ =0 then read operation is perform

It is a control signal sent by the microprocessor to the memory/input device to control READ operation. A low signal indicates that data on the data bus must be placed either from selected memory location or from input device.

$\overline{\mathrm{RD}}$ indicates the selected memory or input device is to be read and that the Data Bus is available for the data transfer.

$\overline{WR}$(Output):

WR stands for write.

It is an active low signal. i.e $\overline{\mathrm{WR}}$ =0 then write operation is perform

It is a control signal sent by microprocessor to the memory/ output device to control Write operation A low signal indicates that data on the data bus must be written into selected memory location or into output device.

$\overline{\mathrm{WR}}$ indicates the data on the Data Bus is to be written into the selected memory or output device.

READY (Input):

It is a signal sent by an input or output device to the microprocessor.

It indicates that the input or output device is ready to send or receive data.

The microprocessor examines READY signal before it performs data transfer operation

If Ready is high, it indicates that the input or output device is ready to send or receive data.

If Ready is low, the microprocessor will wait for Ready to go high before completing the read or write cycle.

HOLD (Input):

It indicates that another device is requesting the use of the address and data bus. Having received HOLD request the microprocessor relinquishes(give up) the use of the buses as soon as the current machine cycle is completed. Internal processing may continue. After the removal of the HOLD signal the processor regains the bus.

Explain with Example

The HOLD pin specifies when any device is demanding the employ of address as well as a data bus. The two devices are LCD as well as A/D converter. Assume that if A/D converter is employing the address bus as well as a data bus. When LCD desires the utilize of both the buses by providing HOLD signal, subsequently the microprocessor transmits the control signal toward the LCD after that the existing cycle will be ended. When the LCD procedure is over, then the control signal is transmitted reverse to A/D converter.

HLDA (Output):

This is the response signal of HOLD, and it specifies whether this signal is obtained or not obtained. After the implementation of HOLD demand, this signal will go low.

INTR (Input):

It is an Interrupt signal sent by an external device to the microprocessor, when it goes high the microprocessor suspends the execution of its normal sequence of instructions i.e If it is active, the Program Counter (PC) will be inhibited from incrementing and an $\overline{\mathrm{INTA}}$ will be issued.

$\overline{INTA}$ (Output):

It is an interrupt acknowledge signal issued by the microprocessor after receiving an interrupt request from an external device. it is low active signal.

RST 5.5, 6.5, 7.5:

These pins are the restart maskable interrupts or Vectored Interrupts, used to insert an inner restart function repeatedly. All these interrupts are maskable, they can be allowed or not allowed by using programs.

TRAP (Input):

Trap interrupt is a non maskable restart interrupt. It is recognized at the same time as INTR. It is unaffected by any mask or Interrupt Enable. It has the highest priority of any interrupt.

RESET IN (Input):

Reset sets the Program Counter to zero and resets the Interrupt Enable and HLDA flipflops. None of the other flags or registers (except the instruction register) are affected The CPU is held in the reset condition as long as Reset is applied.

RESET OUT (Output):

Indicates CPU is being reset. Can be used as a system RESET.

X1, X2 (Input):

Crystal or R/C network connections to set the internal clock generator X1 can also be an external clock input instead of a crystal. The input frequency is divided by 2 to give the internal operating frequency.

CLK (Output):

Clock Output for use as a system clock when a crystal or R/ C network is used as an input to the CPU. The period of CLK is twice the X1, X2 input period.

SID (Input):

Serial input data line the data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.

SOD (output):

Serial output data line. The output SOD is set or reset as specified by the SIM instruction.

Vcc:

+5 volt supply.

Vss:

Ground Reference.

4.4. Stack, Stack pointer & stack top

- Stack is a portion of RAM memory defined by the user for temporary storage and retrieve of data while executing a program.
- The microprocessor will have dedicated internal register called a stack pointer to hold the address of the stack
- Also the processor will have facility to automatically decrement/ increment the content of SP after every Write/read into stack

- For every write operation into the stack the SP automatically decremented by two
- For every read operation into the stack the SP automatically incremented by two
- The contents register are moved to certain memory location by PUSH operation, then the register are used for other operations
- After push operation those contents which were saved in the memory are transferred back to the register by POP operation
- The set of memory location kept for this operation is called Stack
- The last memory location of the occupied portion of the Stack is called Stack top
- A special 16 bit register is known as stack pointer hold the address of stack top
- The stack pointer is initialized in beginning of the program by LXI SP or SPHL instruction
- Data are stored in the stack on Last-in-first-out(LIFO) principle
- SP register hold the address of stack top location

PUSH OPERATION

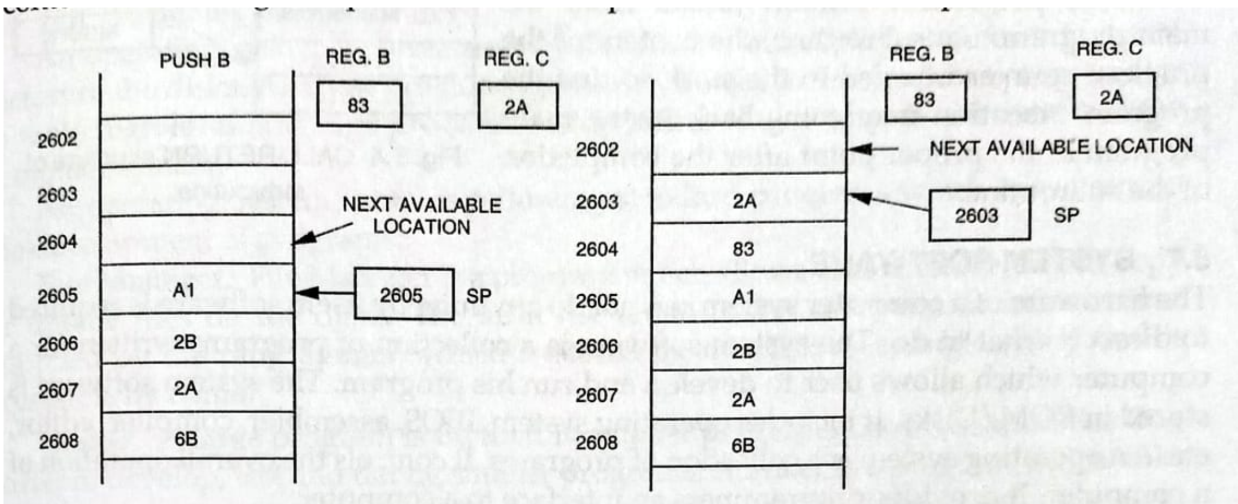


Fig. 5.2 (a) Stack before PUSH operation **Fig. 5.2 (b)** Stack after PUSH operation

POP OPERATION

POP operation is used to transfer the contents from the stack to the register

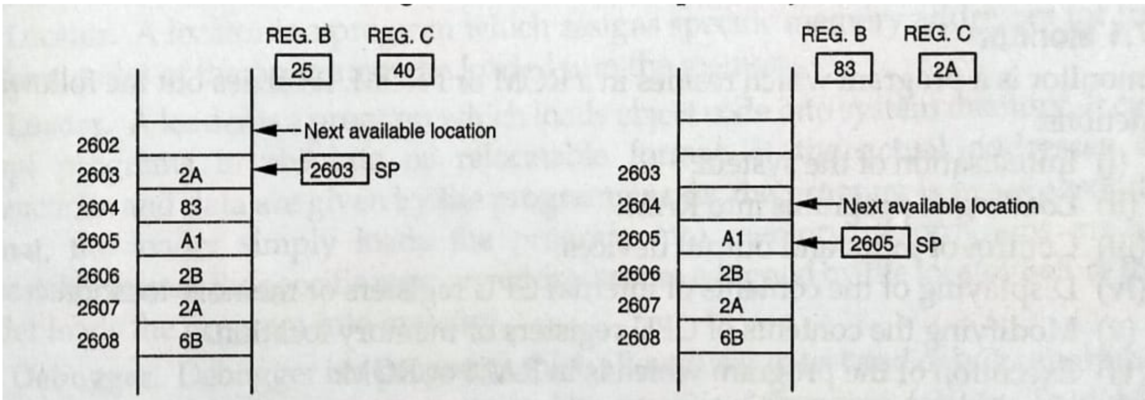


Fig. 5.3 (a) Stack before POP operation **Fig. 5.3 (b)** Stack after POP operation

Interrupts in 8085 microprocessor:

When microprocessor receives any interrupt signal from peripheral(s) which are requesting its services, it stops its current execution and program control is transferred to a sub-routine by generating CALL signal and after executing sub-routine by generating RET signal again program control is transferred to main program from where it had stopped.

When microprocessor receives interrupt signals, it sends an acknowledgement (INTA) to the peripheral which is requesting for its service.

Interrupts can be classified into various categories based on different parameters:

1. **Hardware and Software Interrupts** –

When microprocessors receive interrupt signals through pins (hardware) of microprocessor, they are known as Hardware Interrupts. There are 5 Hardware Interrupts in 8085 microprocessor.

They are – INTR, RST 7.5, RST 6.5, RST 5.5, TRAP

Software Interrupts are program instruction those which are inserted in between the program which means these are mnemonics of microprocessor. There are 8 software interrupts in 8085 microprocessor.

They are – RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6, RST 7.

2. **Vectored and Non-Vectored Interrupts** –

Vectored Interrupts are those which have fixed vector address (starting address of sub-routine) and after executing these, program control is transferred to that address.

Vector Addresses are calculated by the formula

Vector Addresses=Interrupt No.*8

INTERRUPT	VECTOR ADDRESS
TRAP (RST 4.5)	24 H
RST 5.5	2C H
RST 6.5	34 H
RST 7.5	3C H

For Software interrupts vector addresses are given by:

INTERRUPT	VECTOR ADDRESS
RST 0	00 H
RST 1	08 H
RST 2	10 H
RST 3	18 H
RST 4	20 H
RST 5	28 H
RST 6	30 H
RST 7	38 H

Non-Vectored Interrupts are those in which vector address is not predefined. The interrupting device gives the address of sub-routine for these interrupts. INTR is the only non-vectored interrupt in 8085 microprocessor.

3. <u>Maskable and Non-Maskable Interrupts</u> –

Maskable Interrupts are those which can be disabled or ignored by the microprocessor. These interrupts are either edge-triggered or level-triggered, so they can be disabled.

INTR, RST 7.5, RST 6.5, RST 5.5 are maskable interrupts in 8085 microprocessor

Non-Maskable Interrupts are those which cannot be disabled or ignored by microprocessor.

TRAP ***is a non-maskable interrupt. It consists of both level as well as edge triggering and is used in critical power failure conditions***.

Priority of Interrupts –

When microprocessor receives multiple interrupt requests simultaneously, it will execute the interrupt service request (ISR) according to the priority of the interrupts.

Instruction for Interrupts –

5. **Enable Interrupt (EI)**
6. **Disable Interrupt (DI)**
7. **Set Interrupt Mask (SIM) –** It is used to implement the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by setting various bits to form masks or generate output data via the Serial Output Data (SOD) line
8. **Read Interrupt Mask (RIM) –** This instruction is used to read the status of the hardware interrupts (RST 7.5, RST 6.5, RST 5.5) by loading into the A register a byte which defines the condition of the mask bits for the interrupts. It also reads the condition of SID (Serial Input Data) bit on the microprocessor.

4.6 Opcode & Operand,

What is Opcode?

Opcodes mean "operation codes". An opcode is the first part of an instruction which specifies the task to be performed by the computer is called opcode.

It is an instruction that tells the processor what to do with the variable or data written beside it.

What is Operand?

An operand is the second part of the instruction, is the data to be operated on and it is called **operand** .

Instruction Word Size

The 8085 instruction set is classified into the following three groups according to word size:

1. **One-word or 1-byte instructions**
2. **Two-word or 2-byte instructions**
3. **Three-word or 3-byte instructions**

One-Byte Instructions

In 1-byte instruction, the opcode and the operand of an instruction are represented in one byte.

Operand(s) are internal registers and are in the instruction in form of codes. If there is no numeral present in the instruction then that instruction will be of one-byte.

Instruction are required one Memory location to store one byte in the memory

Example, MOV C, A, RAL, and ADD B, etc.

Two-word or 2-byte instructions

Two-byte instruction is the type of instruction in which the first 8 bits indicates the opcode and the next 8 bits indicates the operand.

In a two-byte instruction, the first byte specifies the operation code and second byte specifies the operand.

Source operand is a data byte and immediately following the opcode. If an 8-bit numeral is present in the instruction then that instruction will be of two-byte. Here, the numeral may be a data or an address.

Instruction are required two Memory location to store in the memory

For example, MVI A, 35H and IN 29H, etc.

In a two-byte instruction, the first byte will be the opcode and the second byte will be for the numeral present in the instruction.

Three-word or 3-byte instructions

Three-byte instruction is the type of instruction in which the first 8 bits indicates the opcode and the next two bytes specify the 16-bit address. The low-order address is represented in second byte and the high-order address is represented in the third byte.

In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit operand.

Instruction are required three Memory location to store in the memory

Example, LXI H,3500H and STA 2500H, etc

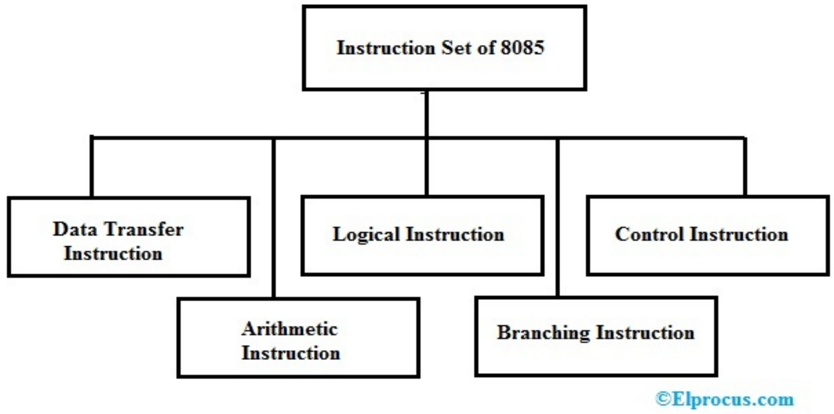
Instruction set of 8085

An instruction is a binary pattern designed inside a microprocessor to perform a specific function.

In microprocessor, the **instruction** set is the collection of the instructions that the microprocessor is designed to execute.

Classification of Instruction Set of 8085

The instruction set of 8085 microprocessor is classified into five types which include the following.



Data Transfer Instruction

An instruction that is used to transfer the data from one register to another is known as data transfer instruction. So, the data transfer can be done from source to destination without changing the source contents.

Data transfer mainly occurs from one register to another register, from memory location to register, register to memory, and between an I/O device & accumulator.

MOV M, Data
This type of instruction specifies the data transfer immediately to a location of memory. This memory location address can be specified at the H-L registers.

Example: MOV M, 28H

MVI r, Data (Move Immediate)
In this type of instruction, the transmission of data can be done immediately toward the particular register.

Example: MVI r, 32H

LDA address (Load Accumulator)
LDA is a load accumulator instruction that is mainly used for copying the data available in the address of memory indicated as the instruction's operand to the accumulator. Particularly, in this case, the available data in the 16-bit address memory is transferred toward the accumulator.

Example: LDA 500H

LDAX (Load Accumulator by extended Register Pair)

It is a load accumulator from an address in the register pair. In this type of data transfer instruction, the register holds the address of the data that needs to be loaded to the accumulator.

Example: LDAX C/D

LHLD (Load H & L Registers Direct)
LHLD instruction is a direct load instruction, where it loads the H-L register with the data from the memory. In this type of instruction, the data which is available in the address specified is copied to the L register first and then the available data within the next memory location will be loaded in the H register.

Example: LHLD 2500H

STA Address (Store Accumulator Contents in Memory)
STA stands for stored accumulator direct instruction. Once this instruction is accepted, then the available data within the accumulator can be transferred to the address of memory indicated within the operand.

Example: STA 2030H

In the above example data stored in the accumulator will be stored to memory location 2030. LSB followed by MSB will be stored in the memory location.

STAX Register (Store Accumulator by Extended Register)
It is a stored accumulator indirect instruction. In this instruction, the register is available as the operand that holds a memory address. Thus, the accumulator data can be copied to that specific memory location.

Example: STAX D

XCHG (Exchange)
This type of data transfer instruction can be used to exchange the data available within two registers.

Example: XCHG H-L & D-E. In this, the contents of H & D and L & E are exchanged.

SPHL (Stack Pointer HL Register)
In this data transfer instruction, the data of H &L can be moved to the stack pointer.

PCHL (Program Counter with HL Data)

Similar to SPHL instruction, this PCHL instruction simply copies the H-L register's data into the SP by loading the high order bytes at H & low order bytes at L.

PUSH

In this type of instruction, the stack can be loaded with the available data within the register provided in the operand. Initially, the stack pointer gets decreased & high order bytes are copied to the stack. Further stack pointer gets decreased to load the low order register bytes.

Example: PUSH D

POP

This instruction indicates the data transfer from the top of the stack to the register provided as the operand.

Example: POP C

OUT

In this type of data transfer instruction, the data available at the accumulator can be copied toward the I/O port. An 8-bit port address at the operand is present.

Example: OUT 36 H

IN

This type of instruction is used to load the data available at the I/O port to the accumulator. The operand simply holds the port address from where the data can be copied.

Example: IN, 6B H

Arithmetic Instruction of 8085

The arithmetic instructions perform different operations like addition, subtraction, increment & decrement on the data within memory & register in the 8085 microprocessor.

ADD r

This arithmetic instruction adds the data which is available in the register to the data available within the accumulator & the final result will be stored in the accumulator.

Example: ADD C

ADD M

This type of instruction is mainly used to add the date in the memory address data denoted at the operand to the data available at the accumulator. So the addition result will be stored within the accumulator.

Example: ADD 28H

ADI Data (Add Immediate)
In this instruction, the 8-bit data is specified as an operand is added immediately to the data available at the accumulator & the result is stored at the accumulator.

Example: ADI 24 H

ACI Data (Add with Carry Immediate)
This type of instruction simply adds the 8-bit data available at the operand & carries the flag by the data available at the accumulator. After every addition, the flag reproduces the output of the addition.

Example: ACI 35H

ADC r (Add with Carry)
In this type of instruction, the data present at the register can be added to the data available at the accumulator with the carry bit & output is simply reflected at the accumulator.

Example: ADC D

AMC M
This type of instruction is mainly used to add the available data at the location of memory whose address is denoted within the operand specified & the carry bit with the data available within the accumulator. So the output of addition can be stored within the accumulator.

Example: AMC 25H

SUB r
This type of instruction is used to subtract the available data at the register given at the operand from the data present in the accumulator. The final result will be stored at the accumulator.

Example: SUB C
SUB M

This instruction is used to subtract the available data at the location of memory whose address is provided by the H-L register from the data present at the accumulator.

Example: SUB 128H

SUI Data (Subtract Immediate from Accumulator)
This type of instruction is mainly used to instantly subtract the data available as operand within the instruction from the available data at the accumulator. After every subtraction, the flag can be changed to show the result of subtraction.

Example: SUI 35H

SBI Data (Subtract with Borrow Immediate from Accumulator)
This type of instruction helps subtract the 8-bit data provided as the operand & the borrow bit from the available data at the accumulator, and the result will be stored within the accumulator.

Example: SBI 24H

SBB r
This instruction is used to subtract the data present at the register & the borrow bit from the data present at the accumulator.

Example: SBB C

SBB M (Subtraction with Borrow)
This instruction is used to specify the subtraction of data available at the memory location, whose address is available at the H-L register & the borrow bit from the data present at the accumulator.

Example: SBB 1000H

INX r (Increment Extended Register)
This type of instruction is used to increase the data by 1 which is available at the register provided at the operand. The result will be stored at the same register.

Example: INX C

DCX r (Decrement Extended Register)
This type of instruction decreases the data available at the register by 1 & the result will be stored in the same register.

Example: DCX C

DCR M (Decrement Register)
In an instruction, sometimes the operand holds a location of memory. The memory location address is available at the H-L pair. Thus the data available at that specific location will be decreased by 1.

Example: DCR 28H

DAA (Decimal Adjust Accumulator)
DAA is a decimal adjust accumulator, used to break the binary number from 8-bit to two 4-bit binary-coded decimal numbers.

Logical Instruction

Logical instructions are mainly used to perform different operations like logical or Boolean over the data available in either memory or register. These instructions will modify the flag bits based on the operation executed.

CMP R/M (Compare the Register/Memory with the Accumulator)
This instruction is used to compare the data at the accumulator with the data present at the register or memory which is given as operand. According to the result obtained by the comparison, the flags are set. While the data that is compared remains unchanged.

Example: CMP B

CPI Data (Compare immediate through the Accumulator)
This type of instruction compares the 8-bit data provided as operand within the instruction by the data available within the accumulator. This result is shown through the flags.

Example: CPI 50

ANA R/M (Logical AND register or memory with the accumulator)
This instruction executes the AND operation of the data available within the accumulator to the data available in the memory or register. After the operation of AND, S, P, Z will be changed to show the outcome of the comparison.

Example: ANA C

ANI data (And Immediate with Accumulator)

This instruction executes AND operation for the immediate 8-bit data provided as operand by the data available in the accumulator.

Example: ANI 35H

ORA R/M (OR Accumulator Register or Memory)

This instruction is used to perform OR operation of the data available within the accumulator by the data available in the memory location or register.

Example: ORA C

ORI data (OR Immediate Data)

The 8-bit data provided as an operand is ORed logically with the data within the accumulator. So, the output of this instruction can be saved within the accumulator.

Example: ORI 36H

XRA R/M (Exclusive OR Immediate with Accumulator)

This instruction is used to execute XOR operation through data available at the accumulator & the data present at the memory or register.

Example: XRA 2030

XRI data (Exclusive OR Accumulator)

This type of instruction is used to execute the XOR operation of the 8-bit data specified as operand & the data present at the accumulator. The output will be stored at the accumulator.

Example: XRI 30

RLC (Rotate Left Accumulator)

This instruction holds significance when there exists a need to rotate the bits present in the accumulator. Basically, for an 8-bit value, each bit is rotated or shifted left by one position. Also, the rotation of the last bit of the sequence i.e., D7, sets the CY flag.

RRC (Right Rotate Accumulator)

This instruction is used to rotate the bit toward the right with one position. So, in this case, D0 sets the CY flag.

Example: RRC

RAL (Rotate Accumulator Left)

This type of instruction is used to rotate the bits toward the left with one of the data available within the accumulator through the carry flag. Here, D7 can be shifted to hold the flag & the bit within the carry flag can be shifted to D0.

Example: RAL

RAR (Rotate Accumulator Right)

This type of instruction is mainly used to rotate the data bits to the right which are available within the accumulator by the carry flag. Here, D0 can be shifted to hold the flag & the carry bit can be moved to the D7 position.

Example: RAR

STC (Set the Carry Flag)

This type of instruction is used to set the carry flag (CF) to 1 by not affecting any other flags.

Example: STC

CMA (Complement the Accumulator)

This type of instruction generates the complement of data at the accumulator. So, this function does not change any of the flags.

Example: CMA

CMC (Complement the Carry Flag)

This type of instruction is used to complement the data available at the carry flag (CF). So this instruction does not affect any other flag.

Example: CMC

Branching Instruction

These types of instructions are mainly used to transfer or switch the microprocessor from one location to another. So, it simply changes the general sequential flow.

JMP address (Jump unconditionally)
This type of instruction is mainly used to transfer the series of the current program to that location of memory whose 16-bit address can be simply specified within the operand of the instruction.

Example: JMP 2014H

Jx Address
This is a conditional branching type instruction, where the series of current programs can be transferred to that specific location whose address can be provided at the operand. However this transferring mainly depends on the specified PSX flag.

Example: JZ 1200H

CALL address
This instruction shifts the control of a series of current programs toward the memory address available at the operand. However the PC gets decreased before transferring,

Example: CALL 2400H

RET (Return from the Subroutine)
This type of instruction can cause the unconditional return of the sub-routine to the actual program.

RST(Restart Instruction)
This type of instruction is mainly used to transfer the series from the main program to the interrupt service routine. Mostly, the transfer can be performed above one of the 8-bits which are indicated within the operand.

Control Instruction

These instructions are mainly used to control the microprocessor operations. These instructions are discussed below.

NOP (No operation)
NOP stands for no operation. Once the 8085 microprocessor gets this instruction, then it does not perform any operation based on execution.

DI (Disable Interrupts)
DI is the disabling of the interrupt that is generated within the microprocessor. Interrupt resetting will allows to disable all the interrupts apart from TRAP.

EI (Enable Interrupts)
This type of instruction is mainly used to allow the interrupt. Once the interrupt enable pin is set then leads to enabling the interrupts within the system.

HLT (Halt & Enter Wait State)
Once the HLT interrupt is decoded through the microprocessor, it stops the current operation and waits for further instruction. To escape from the halt condition either a reset or an interrupt is necessary.

SIM (Set Interrupt Mask)
SIM is the set interrupt mask, which is used to execute the hardware interrupts programming & serial output.
RIM (Read Interrupt Mask)
RIM is the read interrupt mask that is used to situate the preferred data at the accumulator based on the serial input & interrupt.

Addressing mode

These are the instructions used to transfer the data from one register to another register, from the memory to the register, and from the register to the memory without any alteration in the content

The term addressing mode refers to the way in which the operand of the instruction is specified

Types of Addressing Modes

Intel 8085 uses the following addressing modes:

1. **Direct Addressing Mode**
2. **Register Addressing Mode**
3. **Register Indirect Addressing Mode**
4. **Immediate Addressing Mode**
5. **Implicit Addressing Mode**

1. Direct Addressing Mode:-

The address of the operand(data) is directly available in the instruction itself.

In direct addressing mode, the data to be operated is available inside a memory location and that memory location is directly specified as an operand

Examples:
LDA 2050 (load the contents of memory location into accumulator A)
LHLD address (load contents of 16-bit memory location into H-L register pair)
IN 35 (read the data from port whose address is 35)

2. Register Addressing Mode:-

In register addressing the operand is one of the general purpose registers. the opcode specifies the address of the register in addition to the operation to be performed.

In register addressing mode, the data to be operated is available inside the register(s) and register(s) is operands. Therefore the operation is performed within various registers of the microprocessor.

Examples:
MOV A, B (move the contents of register B to register A)
ADD B (add contents of registers A and B and store the result in register A)
INR A (increment the contents of register A by one)

3. Register Indirect Addressing Mode

In this mode of addressing the address of the operand is specified by a register pair.

In register indirect addressing mode, the data to be operated is available inside a memory location and that memory location is indirectly specified by a register pair.

Examples:
MOV A, M (move the contents of the memory location pointed by the H-L pair to the accumulator)
LDAX B (move contents of B-C register to the accumulator)
LXIH 9570 (load immediate the H-L pair with the address of the location 9570)

4. Immediate Addressing Mode

In immediate addressing mode the source operand is always data. If the data is 8-bit, then the instruction will be of 2 bytes, if the data is of 16-bit then the instruction will be of 3 bytes.

Examples:

MVI B 45 (move the data 45H immediately to register B)

LXI H 3050 (load the H-L pair with the operand 3050H immediately)

JMP address (jump to the operand address immediately)

5. Implicit Addressing Mode

In implied/implicit addressing mode the operand is hidden and the data to be operated is available in the instruction itself.

Examples:

CMA (finds and stores the 1's complement of the contents of accumulator A in A)

RRC (rotate accumulator A right by one bit)

RLC (rotate accumulator A left by one bit)

Timing Diagram:

Timing Diagram is a graphical representation. It represents the execution time taken by each instruction in a graphical format. The execution time is represented in T-states.

Instruction Cycle:

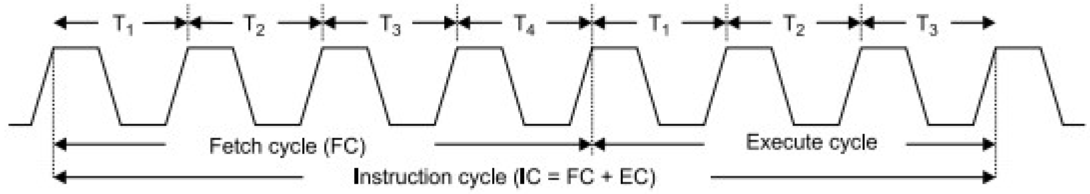
The time required to execute an instruction is called instruction cycle.

or

The time taken by the processor to complete the execution of an instruction. An instruction cycle consists of one to six machine cycles.

Fetch cycle:

The fetch cycle in a microprocessor comprises(consist) of several time states during which the next instruction to be executed is copied (fetched) from the memory location (whose address is in the Program Counter) to the Instruction Register.



IC=FC+EC

Machine Cycle:

The time required to access the memory or input/output devices is called machine cycle.

or

The time required to complete one operation; accessing either the memory or I/O device. A machine cycle consists of three to six T-states.

T-State:

The machine cycle and instruction cycle takes multiple clock periods. A portion of an operation carried out in one system clock period is called as T-state.

Or

Time corresponding to one clock period. It is the basic unit to calculate execution of instructions or programs in a processor.

Rules to identify number of machine cycles in an instruction:

1. If an addressing mode is direct, immediate or implicit then No. of machine cycles = No. of bytes.

2. If the addressing mode is indirect then No. of machine cycles = No. of bytes + 1. Add +1 to the No. of machine cycles if it is memory read/write operation.

3. If the operand is 8-bit or 16-bit address then, No. of machine cycles = No. of bytes +1.

4. These rules are applicable to 80% of the instructions of 8085.

CONCEPT OF TIMING DIAGRAM:

The 8085 microprocessor has 5 (seven) basic machine cycles. They are

1. **Opcode fetch cycle (4T)**
2. **Memory read cycle (3 T)**
3. **Memory write cycle (3 T)**
4. **I/O read cycle (3 T)**
5. **I/O write cycle (3 T**

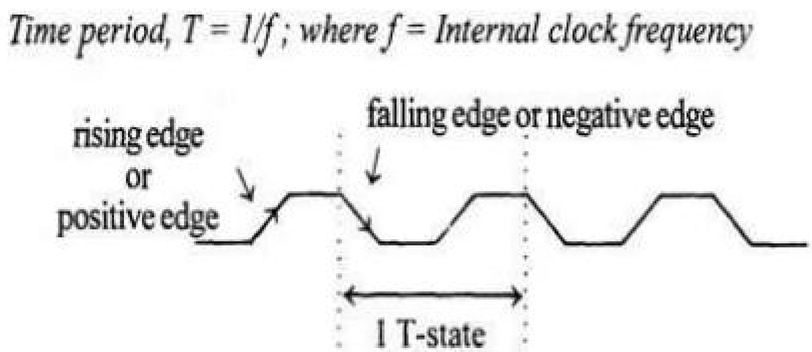


Fig 1.7 Clock Signal

Timing Diagram of Opcode fetch of 8085 :

The microprocessor requires instructions to perform any particular action. In order to perform these actions microprocessor utilizes Opcode which is a part of an instruction which provides detail(ie. Which operation µp needs to perform) to microprocessor.

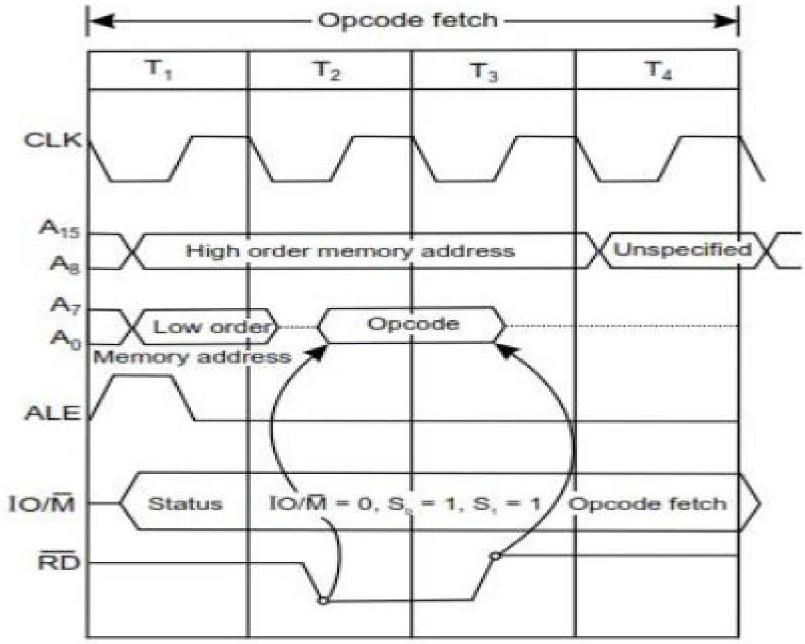


Fig 1.8 Opcode fetch machine cycle

Each instruction of the processor has one byte opcode.

The opcodes are stored in memory. So, the processor executes the opcode fetch machine cycle to fetch the opcode from memory.

Hence, every instruction starts with opcode fetch machine cycle.

The time taken by the processor to execute the opcode fetch cycle is 4T.

In this time, the first, 3 T-states are used for fetching the opcode from memory and the remaining T-states are used for internal operations by the processor.

Timing Diagram of Memory Read

The memory read machine cycle is executed by the processor to read a data byte from memory.

The processor takes 3T states to execute this cycle.

The instructions which have more than one byte word size will use the machine cycle after the opcode fetch machine cycle.

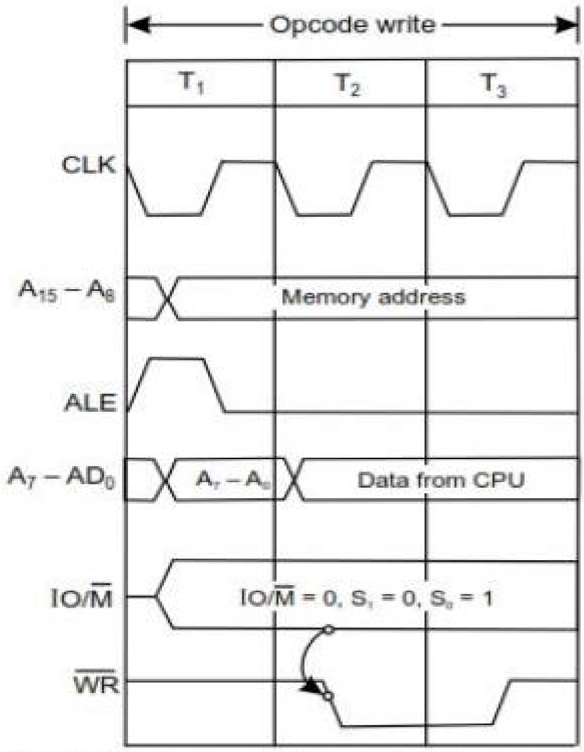


Fig 1.10 Memory Write Machine Cycle

Timing Diagram of Memory Write

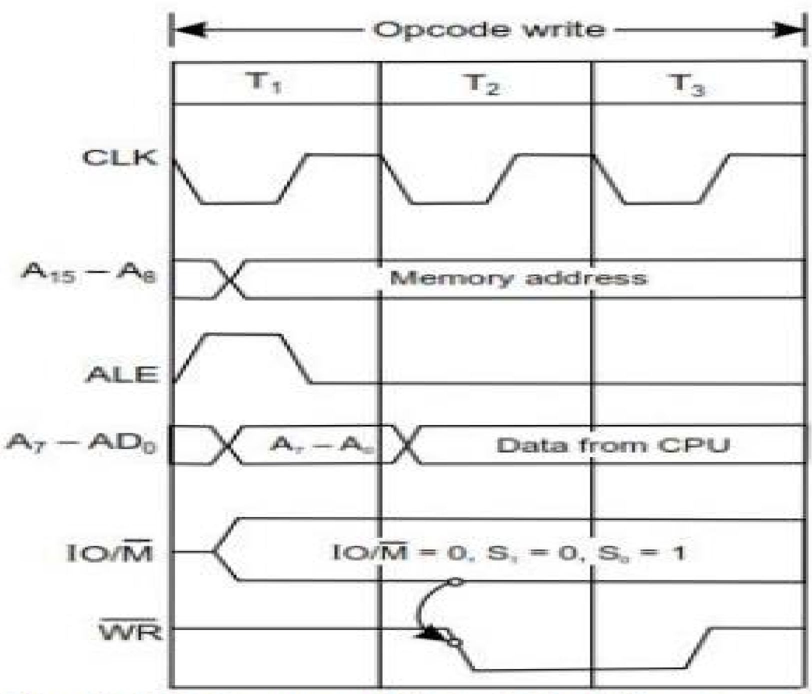


Fig 1.10 Memory Write Machine Cycle

The memory write machine cycle is executed by the processor to write a data byte in a memory location.

The processor takes, 3T states to execute this machine cycle.

Timing Diagram of I/O Read

The I/O Read cycle is executed by the processor to read a data byte from I/O port or from the peripheral, which is I/O, mapped in the system.

The processor takes 3T states to execute this machine cycle.

The IN instruction uses this machine cycle during the execution.

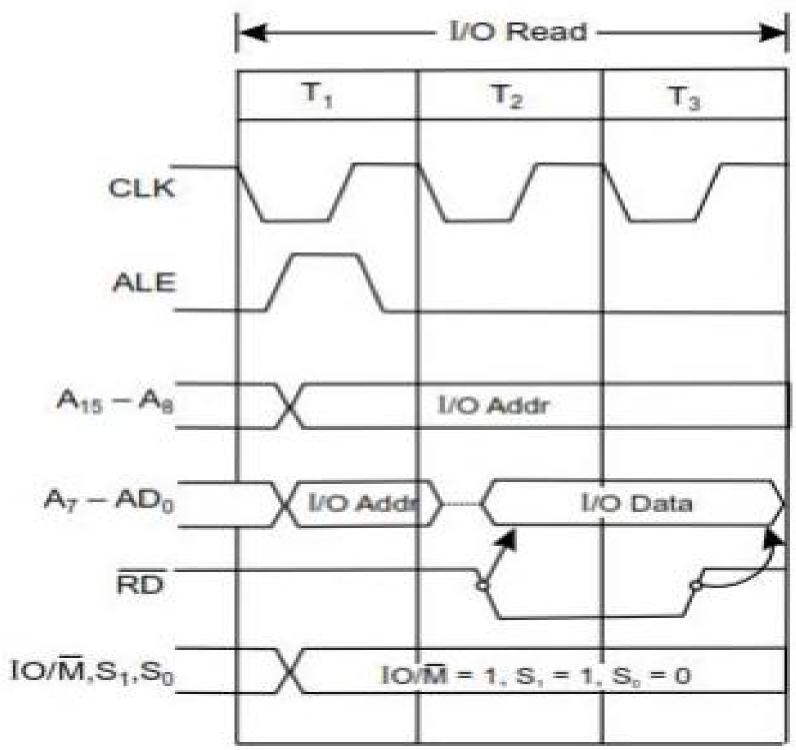


Fig 1.11 I/O Read Cycle

Timing diagram for STA 526AH

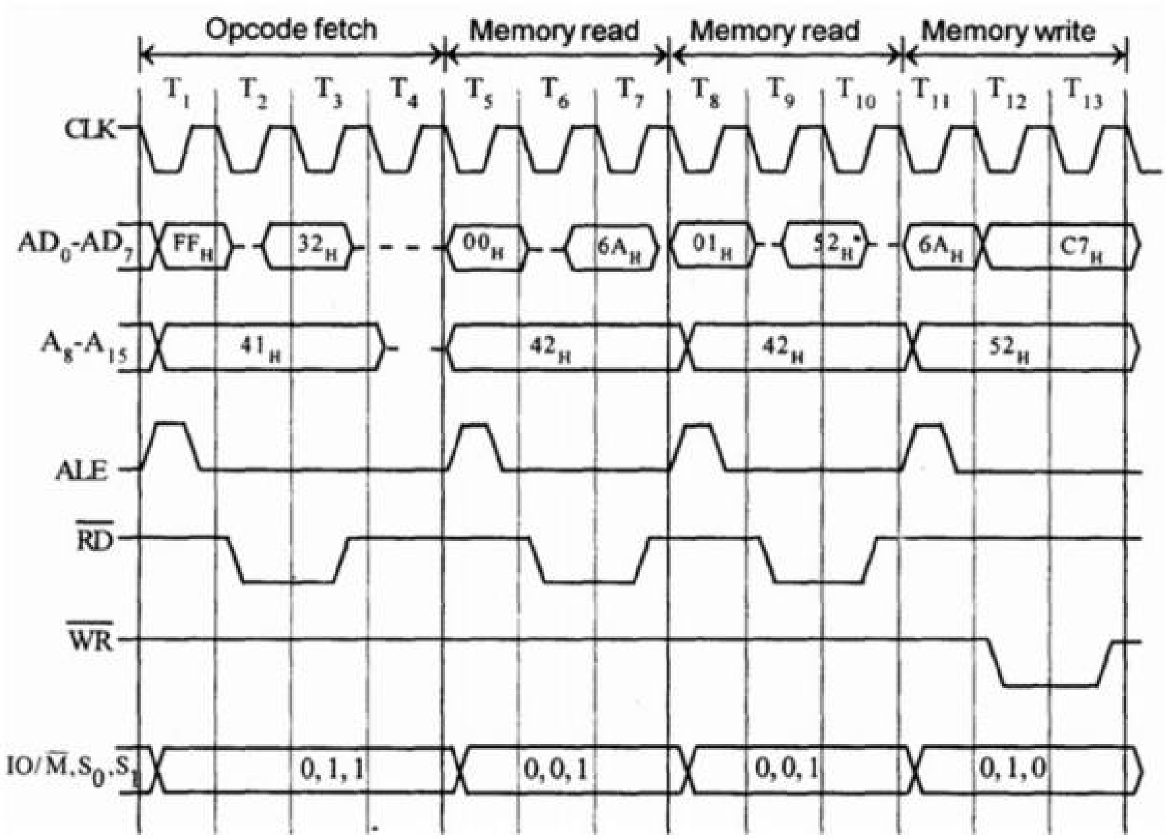


Fig 1.12 Timing Diagram for STA 526A H

Address	Mnemonics	Opcode
41FF	STA 526AH	32H
4200		6AH
4201		52H

STA means Store Accumulator -The contents of the accumulator is stored in the specified address (526A).

The opcode of the STA instruction is said to be 32H. It is fetched from the memory 41FFH (see fig). - OF machine cycle

Then the lower order memory address is read (6A). - Memory Read Machine Cycle

Read the higher order memory address (52).- Memory Read Machine Cycle

The combination of both the addresses are considered and the content from accumulator is written in 526A. - Memory Write Machine Cycle

Assume the memory address for the instruction and let the content of accumulator is C7H. So, C7H from accumulator is now stored in 526A.

Timing diagram for INR M

Fetching the Opcode 34H from the memory 4105H. (OF cycle)

Let the memory address (M) be 4250H. (MR cycle -To read Memory address and data)

Let the content of that memory is 12H.

Increment the memory content from 12H to 13H. (MW machine cycle)

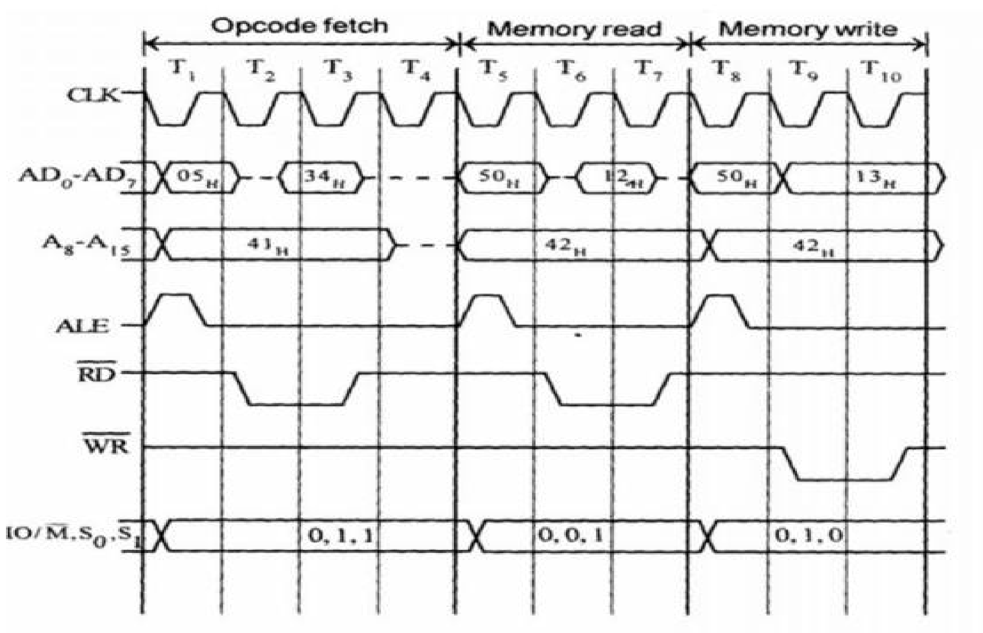


Fig 1.13 Timing Diagram for INR M

Address	Mnemonics	Opcode
4105	INR M	34_H

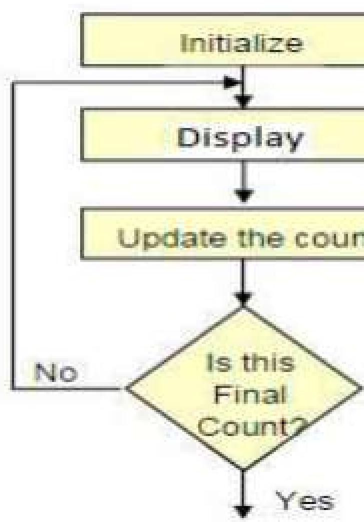
Counter and time delay.

Counter:

A counter is designed simply by loading appropriate number into one of the registers and using INR or DNR instructions.

Loop is established to update the count.

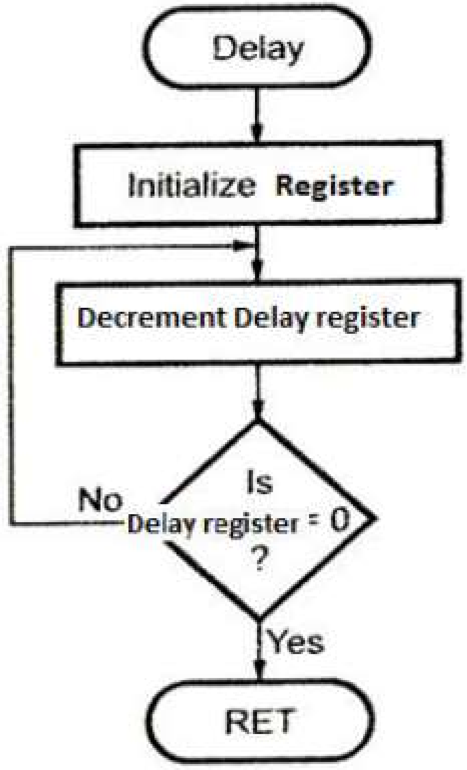
Each count is checked to determine whether it has reached final number ;if not, the loop is repeated.



Time delay:

Procedure used to design a specific delay.′

A register is loaded with a number , depending on the′ time delay required and then the register is decremented until it reaches zero by setting up a loop with conditional jump instruction.



Simple assembly language programming of 8085.

Example 1. Object: Place 05 in register B.

PROGRAM

Memory address	*Machine codes*	*Mnemonics*	*Operands*	*Comments*
FC00	06,05	MVI	B, 05	Get 05 in register B.
FC02	76	HLT		Stop.

Example 2 Object: Get 05 in register A; then move it to register B.

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
FC00	3E, 05	MVI	A, 05	Get 05 in register A.
FC02	47	MOV	B, A	Transfer 05 from register A
FC03	76	HLT		Stop.

Example 3

Object: Load the content of the memory location FC50 H directly to the accumulator, then transfer it to register B. The content of the memory location FC50 H is 05.

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
FC00	3A, 50, FC	LDA	FC50	Get the content of the memory location FC50 H into accumulator.
FC03	47	MOV	B,A	Move the content of register A to B.
FC04	76	HLT		Halt.

DATA

FC50 — 05

Addition of Two 8-bit No.; sum 8-bit

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
2000	21, 01, 25	LXI	H, 2501 H	Get address of 1st number in H-L pair.
2003	7E	MOV	A,M	1st number in accumulator.
2004	23	INX	H	Increment content of H-L pair.
2005	86	ADD	M	Add 1st and 2nd numbers.
2006	32, 03, 25	STA	2503 H	Store sum in 2503 H.
2009	76	HLT		Stop

DATA

2501 — 49 H

2502 — 56 H

The sum is stored in the memory location 2503 H.

Result

2503 — 9F H.

8-bit Subtraction

PROGRAM

Memory address	Machine Codes	Mnemonics	Operands	Comments
2000	21, 01, 25	LXI	H, 2501 H	Get address of 1st number in H-L pair.
2003	7E	MOV	A, M	1st number in accumulator.
2004	23	INX	H	Content of H-L pair increases from 2501 to 2502 H.
2005	96	SUB	M	1st number - 2nd number.
2006	23	INX	H	Content of H-L pair becomes 2503 H.
2007	77	MOV	M, A	Store result in 2503 H.
2008	76	HLT		Halt

Example 1

DATA

2501 — 49 H

2502 — 32 H

Result is stored in the Memory location 2503 H

2503 — 17 H

Example 2

DATA

2501 — F8 H

2502 — 9B H

Result

2503 — 5D H

ADDITION OF TWO 8-BIT NO.;SUM:16-BIT

PROGRAM

Memory address	*Machine Codes*	*Labels*	*Mnemonics*	*Operands*	*Comments*
2000	21, 01, 25		LXI	H, 2501 H	Address of 1st number in H-L pair.
2003	0E, 00		MVI	C,00	MSBs of sum in register C. Initial value = 00.
2005	7E		MOV	A, M	1st number in accumulator.
2006	23		INX	H	Address of 2nd number 2502 in H-L pair.
2007	86		ADD	M	1st number + 2nd number.
2008	D2, 0C, 20		JNC	AHEAD	Is carry? No, go to the label AHEAD.
200B	0C		INR	C	Yes, increment C.
200C	32, 03, 25	AHEAD	STA	2503 H	LSBs of sum in 2503 H.
200F	79		MOV	A, C	MSBs of sum in accumulator.
2010	32, 04, 25		STA	2504 H	MSBs of sum in 2504 H.
2013	76		HLT		Halt

Example 1

DATA

2501 — 98 H

2502 — 9A H

Result

2503 — 32 H, LSBs of sum.

2504 — 01 H, LSBs of sum.

Example 2

DATA

2501 — F5 H

2502 — 8A H

Result

2503 — 7F H, LSBs of sum.

2504 — 01 H, MSBs of sum.

The 1st instruction of th...

CHAPTER-5

INTERFACING AND SUPPORT CHIPS

Basic Interfacing Concepts

Interface is the path for communication between two components. Interfacing is of two types, memory interfacing and I/O interfacing.

Memory mapping & I/O mapping

Functional block diagram and description of each block of Programmable peripheral interface Intel 8255

The parallel input-output port chip 8255 is also called as programmable peripheral input- output port. The Intel's 8255 is designed for use with Intel's 8-bit, 16-bit and higher capability microprocessors.

The 8255A is a general purpose programmable I/O device designed to transfer the data from I/O to interrupt I/O under certain conditions as required. It can be used with almost any microprocessor.

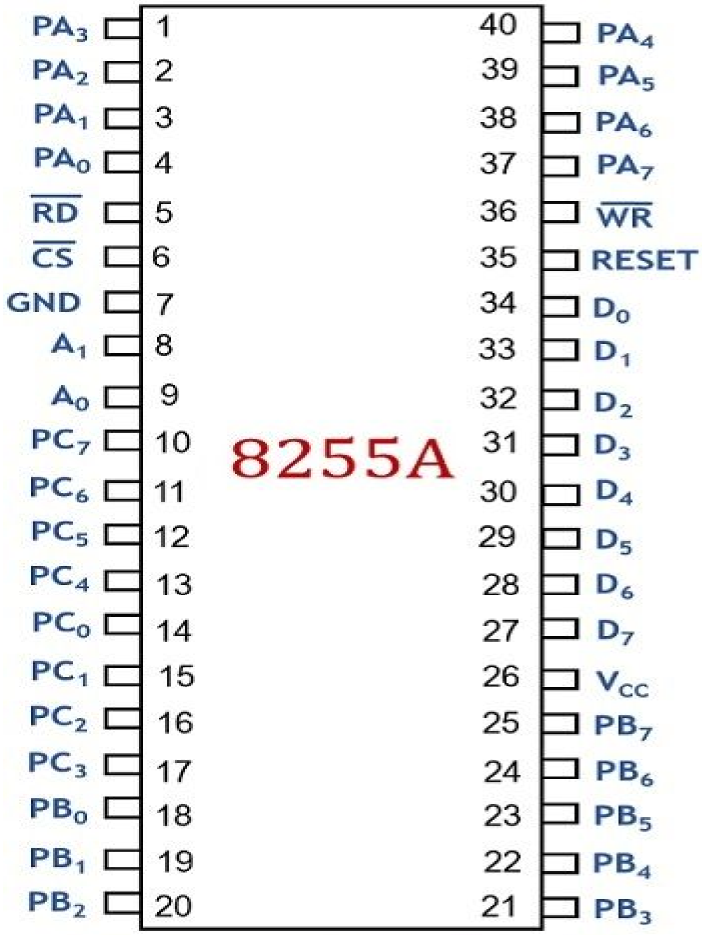
Ports of 8255A

8255A has three ports, i.e., PORT A, PORT B, and PORT C.

Port A contains one 8-bit parallel port i.e $\mathrm{PA}_0 - \mathrm{PA}_7$

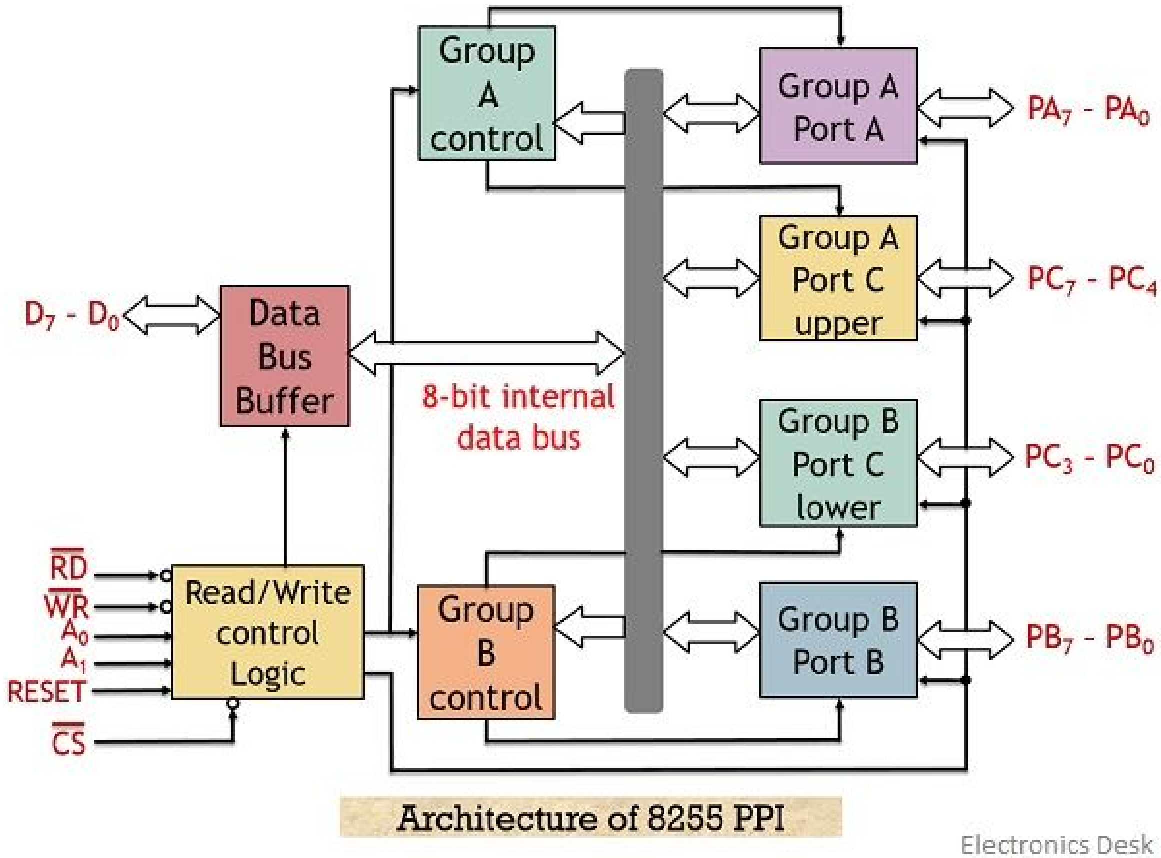
Port B contains one 8-bit parallel port i.e $\mathrm{PB}_0 - \mathrm{PB}_7$

Port C can be split into two parts, i.e. PORT C lower ($\mathrm{PC}_0 - \mathrm{PB}_3$) and PORT C upper (e $\mathrm{PC}_4 - \mathrm{PC}_7$) by the control word.



Pin Diagram of 8255 PPI

Electronics Desk



Architecture of 8255 PPI

Data Bus Buffer

It is a tri-state 8-bit buffer, which is used to interface the microprocessor to the system data bus. Data is transmitted or received by the buffer as per the instructions by the CPU. Control words and status information is also transferred using this bus.

Read/ Write control logic:

This unit manages the internal operations of the system. This unit holds the ability to control the transfer of data and control or status words both internally and externally.

Whenever there exists a need for data fetch then it accepts the address provided by the processor through the bus and immediately generates command to the 2 control groups for the particular operation.

Group A and Group B control:

These two groups are handled by the CPU and functions according to the command generated by the CPU. The CPU sends control words to the group A and group B control and they in turn sends the appropriate command to their respective port.

the group A has the access of the port A and higher order bits of port C. While group B controls port B with the lower order bits of port C.

$\overline{CS}$: It stands for chip select. A low signal at this pin shows the enabling of communication between the 8255 and the processor. More specifically we can say that the data transfer operation gets enabled by an active low signal at this pin.

$\overline{RD}$ – It is the signal used for read operation. A low signal at this pin shows that CPU is performing read operation at the ports or status word. Or we can say that 8255 is providing data or information to the CPU through data buffer.

$\overline{WR}$ – It shows write operation. A low signal at this pin allows the CPU to perform write operation over the ports or control register of 8255 using the data bus buffer.

A_0 and A_1: These are basically used to select the desired port among all the ports of the 8255 and it do so by forming conjunction with RD and WR. It forms connection with the LSB of the address bus.

The table below shows the operation of the control signals:

For the 1st unit of 8255, i.e 8255.1

A_1	A_0	Port/Control word Register address	Device selected
0	0	00	Port-A
0	1	01	Port-B
1	0	02	Port-C
1	1	03	Control Register

For the 2nd unit of 8255, i.e 8255.2

A_1	A_0	Port/Control word Register address	Device selected
0	0	08	Port-A
0	1	09	Port-B
1	0	0A	Port-C
1	1	0B	Control Register

Reset: It is an active high signal that shows the resetting of the PPI. A high signal at this pin clears the control registers and the ports are set in the input mode. Initializing the ports to input mode is done to prevent circuit breakdown. As in case of reset condition, if the ports are initialized to output mode then there exist chances of destruction of 8255 along with the processor.

Operating mode of 8255

Operating mode can be classified as follows
Mode 0: Simple input/output
Mode 1: Input output with handshaking
Mode 2: Bidirectional I/O handshaking

Mode 0: **Simple input/output:-**
In this mode, all the three ports can be programmed either as the input or the output port. Each port can be programmed in either input mode or output mode where outputs are latched and inputs are not latched. Ports do not have interrupt capability. The ports in mode-0 can be used to interface DIP switches, hexa-keypad, LEDs and 7-segment LEDs to the processor

Mode 1: Input output with handshaking
In mode 1 , only port A and B can be programmed either as the input or output port . the port-C are used for handshaking and interrupt control signals. Input and output data are latched. Interrupt driven data transfer scheme is possible

Mode 2: Bidirectional I/O handshaking

In this mode , all the port will be a bidirectional port(i.e. the processor can perform both read and write operations with an IO device connected to a port in mode-2) only port-A can be programmed to work in mode-2. Five pins of port-C are used for handshake signals. This mode is used primarily in applications such as data transfer between two computers or floppy disk controller interface

Control Word:-

7.7.4 Control Word

According to the requirement a port can be programmed to act either as an input port or an output port. For programming the ports of 8255 a control word is formed. The bits of the control word are as shown in Fig. 7.15. Control word is written into the control word register which is within 8255. No read operation of the control word register is allowed. The control word bit corresponding to a particular port is set to either 1 or 0 depending upon the definition of the port, whether it is to be made an input port or output port. If a particular port is to be made an input port, the bit corresponding to that port is set to 1. For making a port an output port, the corresponding bit for the port is set to 0. The detailed description of the bits of the control word is as follows:

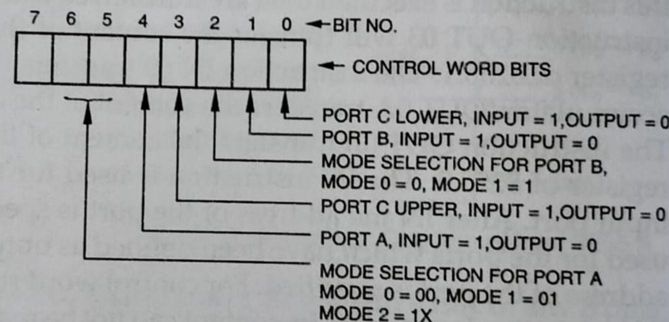


Fig. 7.15 Control Word Bits for Intel 8255

Bit No. 0. It is for Port C_{lower}.
To make Port C_{lower} an input port, the bit is set to 1.
To make Port C_{lower} an output port, the bit is set to 0.

Bit No. 1. It is for Port B.
To make Port B an input port, the bit is set to 1.
To make Port B an output port, the bit is set to 0.

Bit No. 2. It is for the selection of the mode for the Port B. If the Port B has to operate in Mode 0, the bit is set to 0. For Mode 1 operation of the port B, it is set to 1.

Bit No. 3. It is for the Port C_{upper}.
To make Port C_{upper} an input port, the bit is set to 1.
To make Port C_{upper} an output port, the bit is set to 0.

Bit No. 4. It is for Port A.
To make Port A an input port, the bit is set to 1.
To make Port A an output Port, the bit is set to 0.

Bit No. 5 and 6. These bits are to define the operating mode of the Port A. For the various modes of Port A these bits are defined as follows:

Mode of Port A	*Bit No. 6*	*Bit No. 5*
Mode 0	0	0
Mode 1	0	1
Mode 2	1	0 or 1

For mode 2 bit No. 5 is set to either 0 or 1; it is immaterial.

Bit No. 7. It is set to 1 if Port A, B and C are defined as input/output port. It is set to 0 if the individual pins of the Port C are to be set or reset.

Table 7.5 shows control words for various configurations of the ports of 8255 for Mode 0 operation. The following examples will illustrate how to make control words:

Table 7.5 Control Words for 8255A for Mode 0 Operation

Control Word Bits								*Control word*	*Port A*	*Port C_{upper}*	*Port B*	*Port C_{lower}*
7	6	5	4	3	2	1	0					
1	0	0	0	0	0	0	0	80	Output	Output	Output	Output
1	0	0	0	0	0	0	1	81	Output	Output	Output	Input
1	0	0	0	0	0	1	0	82	Output	Output	Input	Output
1	0	0	0	0	0	1	1	83	Output	Output	Input	Input
1	0	0	0	1	0	0	0	88	Output	Input	Output	Output
1	0	0	0	1	0	0	1	89	Output	Input	Output	Input
1	0	0	0	1	0	1	0	8A	Output	Input	Input	Output
1	0	0	0	1	0	1	1	8B	Output	Input	Input	Input
1	0	0	1	0	0	0	0	90	Input	Output	Output	Output
1	0	0	1	0	0	0	1	91	Input	Output	Output	Input
1	0	0	1	0	0	1	0	92	Input	Output	Input	Output
1	0	0	1	0	0	1	1	93	Input	Output	Input	Input
1	0	0	1	1	0	0	0	98	Input	Input	Output	Output
1	0	0	1	1	0	0	1	99	Input	Input	Output	Input
1	0	0	1	1	0	1	0	9A	Input	Input	Input	Output
1	0	0	1	1	0	1	1	9B	Input	Input	Input	Input

the ports of Intel 8255 are defined as follows:

Example are as follows:

Example 2. Form control word for the following configuration of the ports of Intel 8255 for Mode 0 operation:

Port A — output

Port B — output

Port C_{lower} —output

Port C_{upper} —input

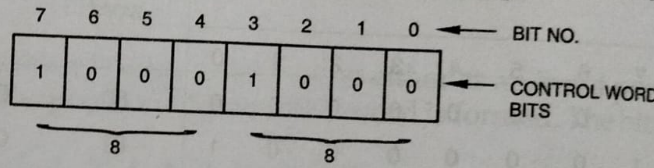


Fig. 7.17

The control word bits for the above configuration of the ports are as shown in Fig. 7.17. The control word for the above definition of the ports of Intel 8255 is 88 H.

Example 3. Make control word for the following arrangement of the ports of Intel 8255 for mode 0 operation:

Port A—output

Port B—output

Port C_{upper} —output

Port C_{lower} —output

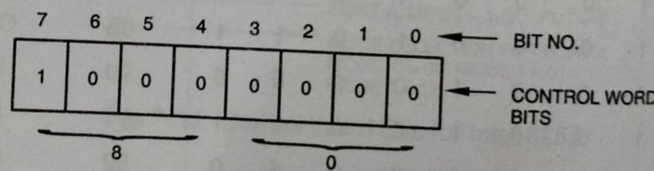


Fig. 7.18

The control word bits for the above configuration of the ports are shown in Fig. 7.18.

The control word for the above definition of the ports of Intel 8255 is 80H.

Example 4. Frame control word for the following configuration of the ports of Intel 8255 for Mode 0 operation:

Port A—input

Port B—input

Port C_{upper} —input

Port C_{lower} —input

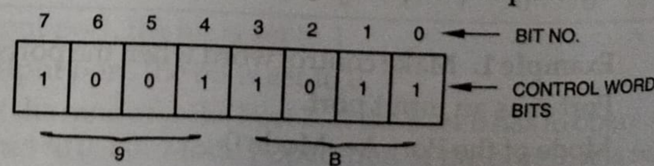
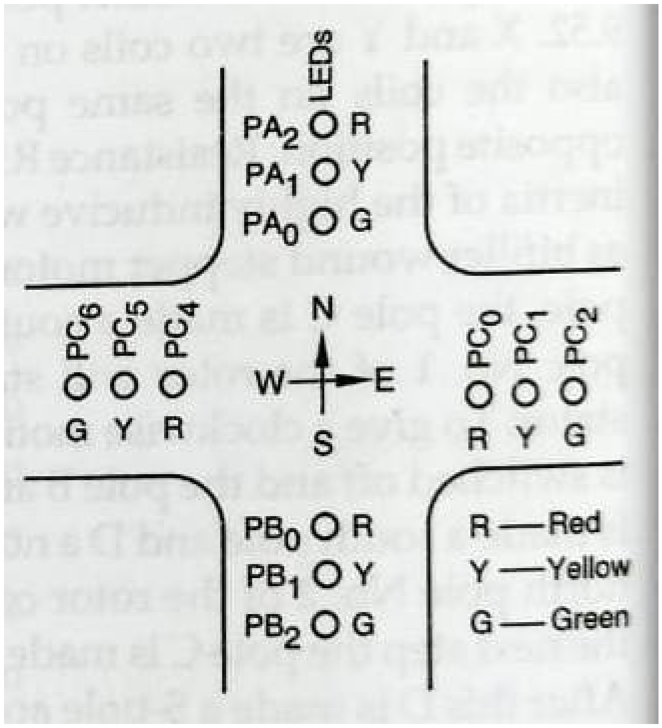


Fig. 7.19

The control word bits for the above configuration of the ports of Intel 8255 are shown in Fig. 7.19.

The control word for the above definition of the ports of Intel 8255 is 9B.

Program for Traffic light Control using 8085 microprocessor



PROGRAM

Memory address	Machine Codes	Lables	Mnemonics	Operands	Comments
FC00	3E, 80		MVI	A, 80 H	Get control word for 8255.
FC02	D3, 0B		OUT	0B	Initialize ports of 8255.2.
FC04	3E, 01	LOOP	MVI	A, 01	
FC06	D3, 09		OUT	09	Red ON for south.
FC08	D3, 08		OUT	08	Red ON for north.
FC0A	3E, 44		MVI	A, 44	Green ON for east and west.
FC0C	D3, 0A		OUT	0A	
FC0E	CD, 00, FD		CALL	DELAY I	
FC11	3E, 22		MVI	A, 22	Yellow ON for east and west.

	D3, 0A		OUT	0A	
FC13	3E, 02		MVI	A, 02	
FC15	D3, 09		OUT	09	Yellow ON for south
FC17	D3, 08		OUT	08	Yellow ON for north.
FC19	CD, 13, FD		CALL	DELAY II	
FC1B	3E, 11		MVI	A, 11	
FC1E	D3, 0A		OUT	0A	Red ON for east and west.
FC20	3E, 04		MVI	A, 04	
FC22	D3, 08		OUT	08	Green ON for north.
FC24	D3, 09		OUT	09	Green ON for south.
FC26	CD, 00, FD		CALL	DELAY I	
FC28	3E, 22		MVI	A, 22	Yellow ON for east and west.
FC2B	D3, 0A		OUT	0A	
FC2D	3E, 02		MVI	A, 02	
FC2F	D3, 09		OUT	09	Yellow ON for south.
FC31	D3, 08		OUT	08	Yellow ON for north.
FC33	CD, 13, FD		CALL	DELAY II	
FC35	C3, 04, FC		JMP	LOOP	
FC38					
DELAY I					
FD00	06, 20		MVI	B, 20 H	
FD02	0E, FF	G03	MVI	C, FF	
FD04	16, FF	G02	MVI	D, FF	
FD06	15	G01	DCR	D	
FD07	C2, 06, FD		JNZ	G01	
FD0A	0D		DCR	C	
FD0B	C2, 04, FD		JNZ	G02	
FD0E	05		DCR	B	
FD0F	C2, 02, FD		JNZ	G03	
FD12	C9		RET		
DELAY II					
FD13	06, 10		MVI	B, 10	
FD15	C3, 02, FD		JMP	FD02	To G03 lable in Delay I

Program for Square wave generator using 8085 microprocessor

9.62

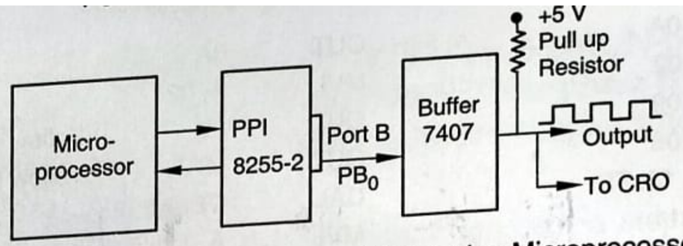


Fig. 9.55 To Generate Square Wave using Microprocessor

PROGRAM

Memory address	Machine Codes	Labels	Mnemonics	Operands	Comments
2400	3E, 98		MVI	A, 98 H	Get control word.
2402	D3, 0B		OUT	0B	Initialize ports.
2404	3E, 00	LOOP	MVI	A, 00	
2406	D3, 09		OUT	09	Make PB_0 LOW.
2408	CD, 00, 25		CALL	DELAY I	
240B	3E, 01		MVI	A, 01	
240D	D3, 09		OUT	09	Make PB_0 HIGH.
240F	CD, 09, 25		CALL	DELAY II	
2412	C3, 04, 24		JMP	LOOP	
SUBROUTINSES					
DELAY I					
2500	06, 02		MVI	B, 02	Get count for delay.
2502	05	GO	DCR	B	
2503	C3, 02, 25		JNZ	GO	
2506	C9		RET		
DELAY II					
2509	0E, 02		MVI	C, 02	Get count for delay.
250B	0D	BACK	DCR	C	
250C	C2, 0B, 25		JNZ	BACK	
250F	C9		RET		