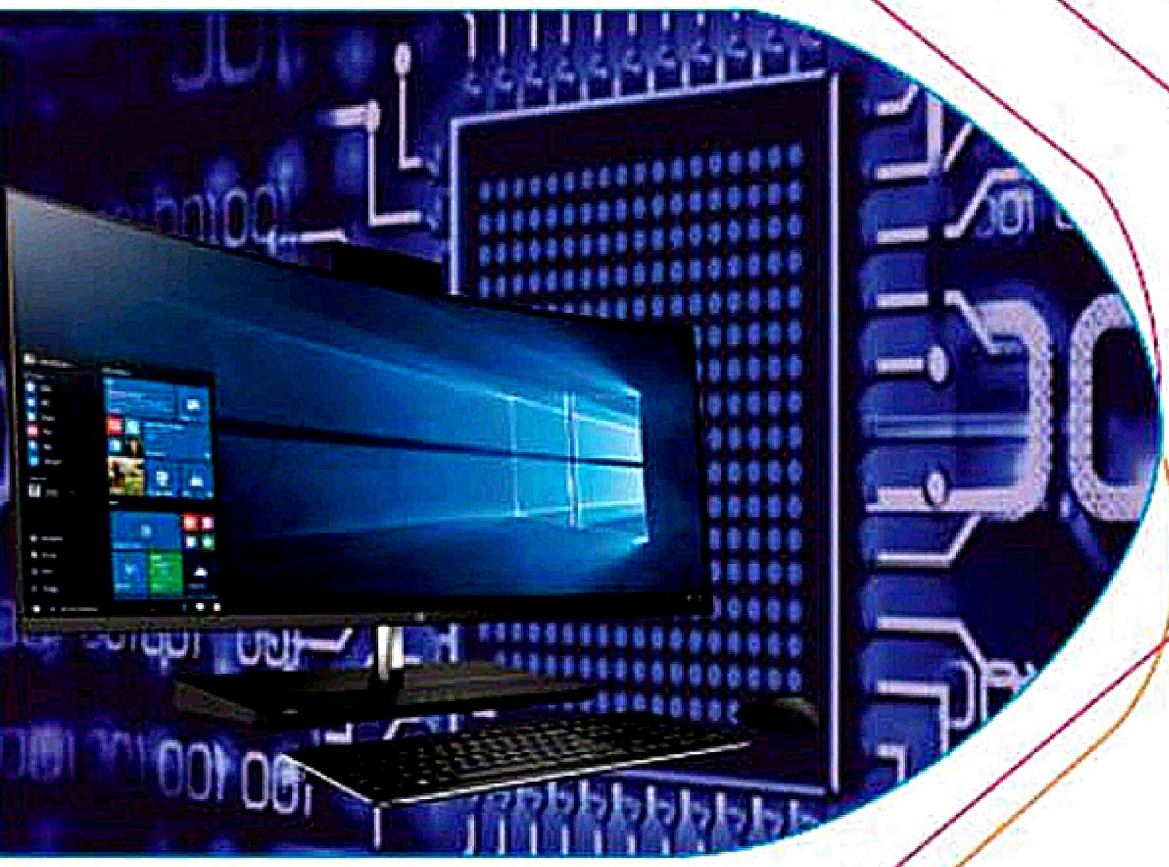
F.Y.B.C.A. SCIENCE : SEMESTER-II

NEW SYLLABUS
CBCS PATTERN

COMPUTER ORGANIZATION

Mrs. MEGHANA TIKHE-PALKAR Mrs. VARSHA BHAGATPATIL
RAHUL PATIL



NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

SPPU New Syllabus

A Book Of

COMPUTER ORGANIZATION

For B.C.A.(Science) : Semester - II

[Course Code 121 : Credit - 4]

CBCS Pattern

As Per New Syllabus, Effective from June 2019

Mrs. Meghana Tikhe-Palkar
M.Sc.(Electronic Science), NET
Associate Professor in Electronics
Dept. of Computer Science
Abashaheb Garware College, Pune

Mrs. Varsha Bhagatpatil
M.E.(E&TC) Digital System

Rahul Patil
M.C.S.
Dept. of Computer Sc.,
N.D.M.V.P. Samaj's K.T.H. M College, Nasik.

Price ₹ 200.00

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

N5111

Computer Organization **ISBN 978-93-89686-33-3**

First Edition : **November 2019**

Published By :
NIRALI PRAKASHAN
Abhyudaya Pragati, 1312, Shivaji Nagar,
Off J.M. Road, Pune – 411005
Tel - (020) 25512336/37/39, Fax - (020) 25511379
Email : niralipune@pragationline.com

Polyplate

Printed By :
YOGIRAJ PRINTERS AND BINDERS
Survey No. 10/1A, Ghule Industrial Estate
Nanded Gaon Road
Nanded, Pune - 411041
Mobile No. 9404233041/9850046517

➢ DISTRIBUTION CENTRES

PUNE

Nirali Prakashan (For orders within Pune) : 119, Budhwar Peth, Jogeshwari Mandir Lane, Pune 411002, Maharashtra
Tel : (020) 2445 2044; Mobile : 9657703145
Email : niralilocal@pragationline.com

Nirali Prakashan (For orders outside Pune) : S. No. 28/27, Dhayari, Near Asian College Pune 411041
Tel : (020) 24690204; Mobile : 9657703143
Email : bookorder@pragationline.com

MUMBAI

Nirali Prakashan : 385, S.V.P. Road, Rasdhara Co-op. Hsg. Society Ltd.,
Girgaum, Mumbai 400004, Maharashtra; Mobile : 9320129587
Tel : (022) 2385 6339 / 2386 9976, Fax : (022) 2386 9976
Email : niralimumbai@pragationline.com

➢ DISTRIBUTION BRANCHES

JALGAON

Nirali Prakashan : 34, V. V. Golani Market, Navi Peth, Jalgaon 425001, Maharashtra,
Tel : (0257) 222 0395, Mob : 94234 91860; Email : niralijalgaon@pragationline.com

KOLHAPUR

Nirali Prakashan : New Mahadvar Road, Kedar Plaza, 1st Floor Opp. IDBI Bank, Kolhapur 416 012
Maharashtra. Mob : 9850046155; Email : niralikolhapur@pragationline.com

NAGPUR

Nirali Prakashan : Above Maratha Mandir, Shop No. 3, First Floor,
Rani Jhanshi Square, Sitabuldi, Nagpur 440012, Maharashtra
Tel : (0712) 254 7129; Email : niralinagpur@pragationline.com

DELHI

Nirali Prakashan : 4593/15, Basement, Agarwal Lane, Ansari Road, Daryaganj
Near Times of India Building, New Delhi 110002 Mob : 08505972553
Email : niralidelhi@pragationline.com

BENGALURU

Nirali Prakashan : Maitri Ground Floor, Jaya Apartments, No. 99, 6th Cross, 6th Main,
Malleswaram, Bengaluru 560003, Karnataka; Mob : 9449043034
Email: niralibangalore@pragationline.com

Other Branches : Hyderabad, Chennai

Preface ...

We take this opportunity to present this book entitled as **"Computer Organization"** to the students of Second Semester - BCA (Science). The object of this book is to present the subject matter in a most concise and simple manner. The book is written strictly according to the New Syllabus (CBCS Pattern).

The book has its own unique features. It brings out the subject in a very simple and lucid manner for easy and comprehensive understanding of the basic concepts, its intricacies, procedures and practices. This book will help the readers to have a broader view on Computer Organization. The language used in this book is easy and will help students to improve their vocabulary of Technical terms and understand the matter in a better and happier way.

We sincerely thank Shri. Dineshbhai Furia and Shri. Jignesh Furia of Nirali Prakashan, for the confidence reposed in us and giving us this opportunity to reach out to the students of computer applications studies.

We have given our best inputs for this book. Any suggestions towards the improvement of this book and sincere comments are most welcome on niralipune@pragationline.com.

Authors

■■■

Syllabus ...

Unit I **Data Representation and Computers Arithmetic** **[08 Hrs]**

- Review of Decimal, Binary, Octal, Hexadecimal Number System and their Inter-conversion, BCD Code, Gray Code, Excess-3 Code, ASCII, EBCDIC, Unicode, Concept of Parity Code. Signed and Unsigned Numbers, 1's and 2's complement of Binary Numbers, Binary Arithmetic (Addition, Subtraction and Subtraction using 1's Complement and 2's Complement).

Unit II **Boolean Algebra & Logic Gates** **[08 Hrs]**

- Introduction, Logic (AND OR NOT), Boolean theorems, Boolean Laws, De Morgan's Theorem, Reduction of Logic expression using Boolean Algebra, Deriving Boolean expression from given Circuit, Exclusive OR and Exclusive NOR Gates, Universal Logic Gates, Implementation of other Gates using Universal Gates.

Minterm, Maxterm and Karnaugh Maps:

- Introduction, Minterms and sum of Minterm Form, Maxterm and Product of Maxterm Form, Reduction technique using Karnaugh Maps – 2/3/4 variable K-Maps, Grouping of variables in K-maps, K-maps for Product of Sum Form, Minimize Boolean Expression using K-map and obtain K-map from Boolean Expression.

Unit III **Combinational Circuits** **[08 Hrs]**

- **Introduction:** Multi-input, Multi-output Combinational Circuits, Code Converters Design and Implementations.
- **Arithmetic Circuits:** Introduction, Adder, BCD Adder, Excess-3 Adder, Binary Subtractors, BCD Subtractor, Multiplier, Comparator.
- **Multiplexer, Demultiplexer, ALU, Encoder and Decoder:** Introduction, Multiplexer, De-multiplexer, Decoder, ALU, Encoders.

Unit IV **Sequential Circuits** **[08 Hrs]**

- Introduction, Terminologies used, S-R flip-flop, D Flip-Flop, JK flip-flop, Race-around Condition, Master – Slave JK Flip-Flop, T Flip-Flop, Conversion from one Type of Flip-Flop to another, Application of Flip-Flops.
- **Counters:** Introduction, Asynchronous Counter, Terms related to Counters, IC7493 (4-bit Binary Counter), Synchronous Counter, Bushing, Type T Design, Type JK Design, Pre-settable Counter, IC 7490, IC 7492, Synchronous Counter ICs, Analysis of Counter Circuits.
- **Shift Register:** Introduction, Parallel and Shift Registers, Ring Counter, Johnson Counter.

Unit V **CPU, Memory and I/O Organization** **[08 Hrs]**

- Block Diagram of CPU, Functions of CPU, General Register Organization, Flags, Concept of RISC and CISC, Introduction to Hardwired and Micro-programmed CPU.
- Memory System Hierarchy, Cache Memory, Internal Memory, External Memory, Concept of Virtual Memory.
- **Input/Output:** Types of I/O Data Transfers - CPU Initiated, Interrupt Initiated and DMA, Need of I/O Interfaces, Parallel and Serial Communication (Asynchronous and Synchronous Data Transfer).

Unit VI **Introduction to Microprocessors and Microcontrollers** **[10 Hrs]**

- Block Diagram of Pentium, Functional Units, Concept of Pipeline and Parallelism, Programmers Model.
- Introduction to Microcontroller Intel 8051 – Functional Block Diagram, Introduction to Multi-Core Processors.

■■■

Contents ...

■■■

1... Data Representation and Computers Arithmetic

Objectives...

- To learn about various Number Systems such as Binary, Decimal, Octal, Hexadecimal and their inter-Conversions.
- To study about Binary Code and Binary Arithmetic.

1.1 INTRODUCTION

- **Data representation** refers to the internal method used to represent various types of data stored on a computer. Computers use different types of numeric codes to represent various forms of data, such as text, number, graphics and sound.
- In all modern computers, storage and processing units are made of a set of silicon chips and each containing a large number of transistors.
- A transistor is a two-state electronic device that can be put OFF and ON by passing an electric current through it. Since, the transistors are sensitive to currents and act like switches; we can communicate with the computers using electric signals, which are represented as a series of "pulse" and "no-pulse" conditions.
- For the sake of convenience and ease of use, a pulse is represented by the code "1" and a no-pulse by the code "0". They are called bits, an abbreviation of "binary digits". A series of 1's and 0's are used to represent a number or a character and thus they provide a way for humans and computers to communicate with each other.
- Information is therefore organized in groups of bits, group of 4 bits is called as nibble and a group of eight bits is called as a byte. A byte may represent a number, a letter, a mathematical symbol and so on.
- **Computer Arithmetic** deals with the study, design, optimization and validation of theoretical and practical means of computing operations such as addition, subtraction, multiplication, division etc.
- Various representations of numbers (or number systems) can be used depending on the application and implementation constraints like integer, fixed point, floating point etc.

- In our daily life, we see electronic calculators; computers, microprocessors etc. perform operations like addition, subtraction, multiplication and division. These devices use various number systems.
- A number system is defined as, "a set of values used to represent quantity". In other words, the system in which an ordered set of digits are used to specify any number is called Number System.
- There are two types of number system as explained below:

1. Non Positional Number Systems:

- In ancient times, people used their fingers for counting. When fingers became insufficient for counting, stones and pebbles were used to indicate values. This method of counting is called the non-positional number system.
- In non-positional number system, we have symbols like I for 1, II for 2, III for 3, IIII for 4 etc. Each symbol represents the same value regardless of its position in a number and to find the value of a number, one has to count the number of symbols present in the number.
- Non-positional number system is also called Unitary Number System.
- The most common non-positional number system is the Roman Number System.

2. Positional Number Systems:

- A positional number system is any system that requires a finite number of symbols/digits of the system to represent arbitrarily large numbers.
- When using these systems the execution of numerical calculations becomes simplified, because a finite set of digits is used. The value of each digit in a number is defined not only by the symbol, but also by the symbol's position.
- The most popular positional number system being used today is the Decimal Number System.

Base or Radix in Number System:

- The base or radix of a number system defines the range of possible values that a digit may have.
- A number in the number system of base or radix (r) is represented by a set of symbols from r distinct symbols.
- The decimal number system uses 10 digits from 0 to 9, thus its base is 10.
- The binary number system uses two distinct digits 0 and 1, thus its base is 2. For octal number system (base r = 8), a number is represented by 8 distinct digits 0 to 7.
- The 16 symbols used in hexadecimal number system (base 16) are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. Here, the symbols A, B, C, D, E, and F correspond to the decimal numbers 10, 11, 12, 13, 14, and 15 respectively.

1.2 NUMBER SYSTEMS AND THEIR INTER-CONVERSIONS

- Successful programming for digital services requires precise understanding of data formats, number systems and their conversion. The four number systems are Binary, Octal, Decimal and Hexadecimal number system.

1.2.1 Decimal Number System

- The number system that uses ten digits (or symbols), viz. 0,1,2,3,4,5,6,7,8 and 9 is called a Decimal Number System.
- Deci means 10. There are only 10 basic digits ranging from 0 to 9.
- The decimal position values as power of 10 are as shown in Fig. 1.1.

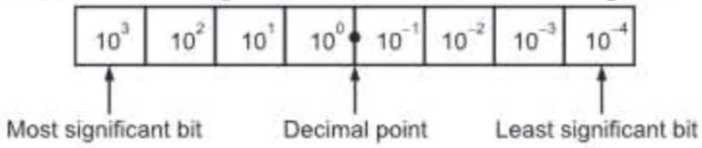
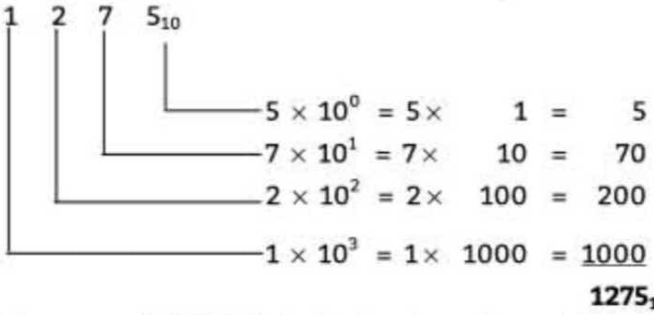


Fig. 1.1: Representation of Decimal Position Values.

- In this case, the Most Significant Digit (MSD) and Least Significant Digit (LSD) are the left most and the right most digit respectively.
- For example, the decimal number 1275 (written 1275_{10}) can be expanded as follows:



- For example, $(4234)_{10}$ in decimal number system can be written as,

$$= (4 \times 10^3) + (2 \times 10^2) + (3 \times 10^1) + (4 \times 10^0)$$
$$= (4 \times 1000) + (2 \times 100) + (3 \times 10) + (4 \times 1)$$
$$= 4000 + 200 + 30 + 4 = 4234$$

1.2.2 Binary Number System [S-17, 18, W-17]

- The number system which uses only two digits (or symbols), viz. 0 and 1 and the base is 2 is called a Binary Number System.
- Naturally, the base (or radix) in this number system is 2. It uses only two digits namely 0 and 1.
- The bit positions in a binary system have weights as shown in Fig. 1.2.

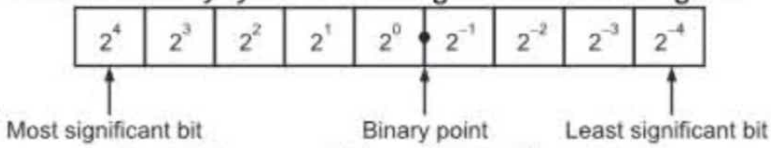


Fig. 1.2: Representation of Bit Position Values

- The leftmost bit in a given binary number with highest weight is called a Most Significant Bit (MSB). The right most bit in a given binary number with the lowest weight is called Least Significant Bit (LSB). Similarly, the point in a given binary number which acts as the indicator between positive and negative exponents is called binary point.

- For example, value of $(11100)_2$

$$= (1\times2^4) + (1\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0)$$
$$= 16 + 8 + 4 + 0 + 0$$
$$= 28$$

1. Binary to Decimal Conversion

- Any binary number can be converted to its equivalent decimal number by adding together the weights of various positions in the binary number which contain 1.
- **Examples:**

(i) $(1011)_2 = (?)_{10}$

$$(1011)_2 = \begin{matrix}1 & 0 & 1 & 1\\ \downarrow & \downarrow & \downarrow & \downarrow\\ 2^3 & 2^2 & 2^1 & 2^0\end{matrix}$$
$$= (1\times2^3) + (0\times2^2) + (1\times2^1) + (1\times2^0)$$
$$= 8 + 0 + 2 + 1$$
$$= (11)_{10}$$

(ii) $(0111.11)_2 = (?)_{10}$

$$(0111.11)_2 = \begin{matrix}0 & 1 & 1 & 1 & . & 1 & 1\\ \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow\\ 2^3 & 2^2 & 2^1 & 2^0 & & 2^{-1} & 2^{-2}\end{matrix}$$
$$= (0\times2^3) + (1\times2^2) + (1\times2^1) + (1\times2^0) + (1\times2^{-1}) + (1\times2^{-2})$$
$$= 0 + 4 + 2 + 1 + \frac{1}{2} + \frac{1}{4}$$
$$= (7.75)_{10}$$

(iii) $(110110)_2 = (?)_{10}$

$\therefore$
$$\begin{matrix}1 & 1 & 0 & 1 & 1 & 0\\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow\\ 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0\end{matrix}$$
$$= (1\times2^5) + (1\times2^4) + (0\times2^3) + (1\times2^2) + (1\times2^1) + (0\times2^0)$$
$$= 32 + 16 + 4 + 2$$
$$= (54)_{10}$$

(iv) $(10001011)_2 = (?)_{10}$

$\therefore$
$$\begin{matrix}1 & 0 & 0 & 1 & 0 & 1 & 1\\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow\\ 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0\end{matrix}$$
$$= (1\times2^6) + (0\times2^5) + (0\times2^4) + (1\times2^3) + (0\times2^2) + (1\times2^1) + (1\times2^0)$$
$$= 64 + 8 + 2+1$$
$$= (75)_{10}$$

(v) $(1100.010)_2 = (?)_{10}$

$\therefore$
$$\begin{matrix}1 & 1 & 0 & 0 & . & 0 & 1 & 0\\ \downarrow & \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow\\ 2^3 & 2^2 & 2^1 & 2^0 & & 2^{-1} & 2^{-2} & 2^{-3}\end{matrix}$$
$$= (1\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0) + (0\times2^{-1}) + (1\times2^{-2}) + (0\times2^{-3})$$
$$= 8 + 4 + \frac{1}{4}$$
$$= (12.25)_{10}$$

2. Decimal to Binary Conversion:

- For converting decimal numbers to binary, the decimal number is repeatedly divided by 2 and writing the remainder after each division, until a quotient 0 is obtained.
- Binary equivalent is obtained by writing the last remainder as the most significant bit and first remainder as the least significant bit, as the direction of arrow points upwards.
- **Examples:**

(i) $(11)_{10} = (?)_2$

2	11	1
2	5	1
2	2	0
2	1	1
	0	

LSB

MSB

$(11)_{10} = (1011)_2$

(ii) $(29)_{10} = (?)_2$.

2	29	1
2	14	0
2	7	1
2	3	1
2	1	1
	0	

LSB

MSB

$(29)_{10} = (11101)_2$

1.2.3 Octal Number System [S-17, 18, W-17]

- The number system which has the base (or radix) 8 and uses only eight digits or symbols, viz. 0, 1, 2, 3, 4, 5, 6 and 7 is called Octal Number System.
- The various digit positions in this system have weights as shown in Fig. 1.3.

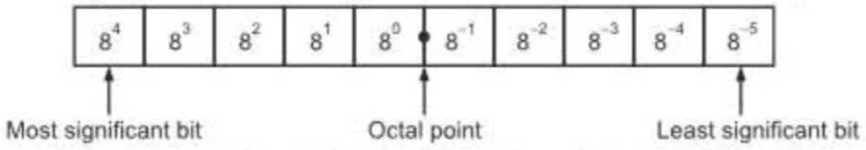


Fig. 1.3: Representation of digit Position Values in Octal Number System.

- For example, $(4131)_8$ is,

$$= (4 \times 8^3) + (1 \times 8^2) + (3 \times 8^1) + (1 \times 8^0)$$
$$= (4 \times 512) + (1 \times 64) + (3 \times 8) + (1 \times 1)$$
$$= 2048 + 64 + 24 + 1$$
$$= 2137$$

1. Octal to Binary Conversion

- The octal number system has the primary advantage of the ease with which it can be converted into Binary Number System.

- It is converted to binary by converting each octal digit to its 3-bit binary equivalents.

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

- **Examples:**

(i) $(652)_8 = (?)_2$

6 5 2
↓ ↓ ↓
110 101 010

$(652)_8 = (110101010)_2$

(ii) $(5432.76)_8 = (?)_2$

5 4 3 2 . 7 6
↓ ↓ ↓ ↓ ↓ ↓
101 100 011 010 111 110

$(5432.76)_8 = (101100\ 011010 \cdot 111\ 110)_2$

(iii) $(437.21)_8 = (?)_2$

4 3 7 . 2 1
↓ ↓ ↓ ↓ ↓
100 011 111 010 001

$(437.21)_8 = (100\ 011\ 111 \cdot 010\ 001)_2$

2. Binary to Octal Conversion

- The bits of the binary number are grouped into groups of three bits starting from the LSB.
- **Examples:**

(i) $(110\ 111\ 010)_2 = (?)_8$

110 111 010
6 7 2

$(110\ 111\ 010)_2 = (672)_8$

- Sometimes, the binary number may not have even groups of 3 bits. In such cases, we can add one or two zeros to the left of the MSB of the binary number to complete the last group.

(ii) $(1101\ 010)_2 = (?)_8$

$\underbrace{001}_{1}\ \underbrace{101}_{5}\ \underbrace{010}_{2}$

We added two zeros to the left of MSB to obtain even groups of 3-bits.

(iii) $(1101 \cdot 110\ \ 11)_2 = (?)_8$

Adding 0s to the left of MSB and to the right of LSB after the decimal point,

$\underbrace{001}\ \underbrace{101}\ \cdot\ \underbrace{110}\ \underbrace{110}$

$(1101 \cdot 11011)_2 = (15.66)_8$

3. Octal to Decimal Conversion:

- An octal number can be converted to its decimal equivalent by multiplying each octal digit by its equivalent position weight.
- **Examples:**

(i) $(250)_8 = (?)_{10}$

$$\begin{matrix} 2 & 5 & 0 \\ \downarrow & \downarrow & \downarrow \\ 8^2 & 8^1 & 8^0 \end{matrix}$$

$= (2 \times 8^2) + (5 \times 8^1) + (0 \times 8^0)$

$= 128 + 40 + 0 = (168)_{10}$

(ii) $(35.7)_8 = (?)_{10}$

$$\begin{matrix} 3 & 5 & . & 7 \\ \downarrow & \downarrow & & \downarrow \\ 8^1 & 8^0 & & 8^{-1} \end{matrix}$$

$= (3 \times 8^1) + (5 \times 8^0) + (7 \times 8^{-1})$

$= 24 + 5 + 0.875$

$= (29.875)_{10}$

4. Decimal to Octal Conversion:

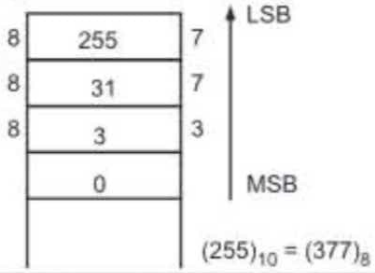
- To convert a decimal number to octal, repeated division method is used i.e. the decimal number is divided repeatedly till the quotient becomes zero.
- **Examples:**

(i) $(6458)_{10} = (?)_8$

8	6458	2	↑ LSB
8	807	7	
8	100	4	
8	12	4	
8	1	1	
	0		MSB

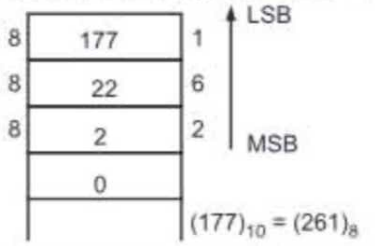
i.e. $(6458)_{10} = (14472)_8$

(ii) $(255)_{10} = (?)_8$



(iii) $(177.25)_{10} = (?)_8$

To convert the integer part, we use repeated division by 8.



To convert the fractional part,

Decimal fraction	Product	Integer bit
0.25×8	2.00	2 (MSB)
0.00×8	0.00	0 (LSB)

i.e. $(177.25)_{10} = (261.20)_8$

- The Octal Number System is used to express large binary numbers that are used in computers. While dealing with large binary numbers with many bits, it becomes easier to write the numbers in octal rather than binary.

1.2.4 Hexadecimal Number System [S-17, 18, W-17]

- Hexadecimal means 16. The number system which uses the radix (or base) 16 and 16 digits (or symbols), viz. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F is called Hexadecimal Number System.
- Relationship between hexadecimal, decimal and binary numbers is as given below:

Hexadecimal	Decimal	Binary	Octal
0	0	0000	0
1	1	0001	1
2	2	0010	2
3	3	0011	3
4	4	0100	4
5	5	0101	5
6	6	0110	6

contd. ...

7	7	0111	7
8	8	1000	10
9	9	1001	11
A	10	1010	12
B	11	1011	13
C	12	1100	14
D	13	1101	15
E	14	1110	16
F	15	1111	17

- For example, $(2BD)_{16}$ is equivalent to:

$$= (2 \times 16^2) + (B \times 16^1) + (D \times 16^0)$$
$$= (2 \times 256) + (11 \times 16) + (13 \times 1)$$
$$= 512 + 176 + 13$$
$$= 601$$

Thus, $(2BD)_{16} = (601)_{10}$

1. Hexadecimal to Decimal Conversion

- A hexadecimal number can be converted to its equivalent decimal number by keeping in mind the fact that each hexadecimal digit has a weight that is a power of sixteen.

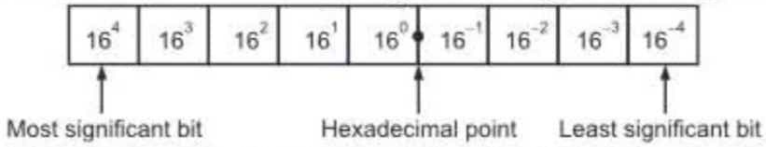


Fig. 1.4: Representation of Hexadecimal digit's position values

- **Examples:**

(i) $(268)_{16} = (?)_{10}$

$$(\underset{16^2}{2}\ \underset{16^1}{6}\ \underset{16^0}{8})_{16}$$
$$= (2 \times 16^2) + (6 \times 16^1) + (8 \times 16^0)$$
$$= 512 + 96 + 8 = 616$$

$\therefore (268)_{16} = (616)_{10}$

(ii) $(456\ E)_{16} = (?)_{10}$

$$\underset{16^3}{4}\ \underset{16^2}{5}\ \underset{16^1}{6}\ \underset{16^0}{E}$$
$$= (4 \times 16^3) + (5 \times 16^2) + (6 \times 16^1) + (14 \times 16^0) \quad \text{(Value of E = 14)}$$
$$= 16384 + 1280 + 96 + 14$$
$\therefore = (17774)_{10}$

(iii) $(11A \cdot 62)_{16} = (?)_{10}$:

$$\underset{16^2}{1}\ \underset{16^1}{1}\ \underset{16^0}{A} \cdot \underset{16^{-1}}{6}\ \underset{16^{-2}}{2}$$

$= (1 \times 16^2) + (1 \times 16^1) + (10 \times 16^0) + (6 \times 16^{-1}) + (2 \times 16^{-2})$ (Value of A = 10)

$= 256 + 16 + 10 + \frac{6}{16} + \frac{2}{256}$

$\therefore \quad = (282 \cdot 382)_{10}$

2. Decimal to Hexadecimal Conversion

- To convert a decimal number into hexadecimal, the decimal number is divided by 16 repeatedly till the quotient becomes zero.
- **Examples:**

(i) $(200)_{10} = (?)_{16}$

16	200	8	↑ LSB
16	12	C	MSB
	0		

$(200)_{10} = (C8)_{16}$

(ii) $(792)_{10} = (?)_{16}$

16	792	8	↑ LSB
16	49	C	
16	3		
	0		MSB

$(792)_{10} = (318)_{16}$

- To convert decimal fraction into hexadecimal, the fraction must be multiplied by 16. The integer part of the product is separated from the fractional part and the fractional part is multiplied by 16 repeatedly.

(iii) $(0.675)_{10} = (?)_{16}$

Decimal fraction	Product	Integer bit	
0.675 × 16	10.8	10 (A)	MSB
0.8 × 16	12.8	12 (C)	
0.8 × 16	10.8	10 (A)	LSB

$(0.675)_{10} = (0.ACA)_{16}$

3. Hexadecimal to Binary Conversion:

- Since hexadecimal systems have a base radix of 16 (i.e. $16 = 2^4$), we can therefore, represent each hexadecimal digit by a group of four binary bits.

- **Examples:**

(i) $(D283)_{16} = (?)_{12}$

D 2 8 3
↓ ↓ ↓ ↓
1101 0010 1000 0011

$(D283)_4 = (1101\ 0010\ 1000\ 0011)_2$

(ii) $(57EB \cdot AD)_{16} = (?)_2$

5 7 E B . A D
↓ ↓ ↓ ↓ ↓ ↓
0101 0111 1110 1011 1010 1101

$(57EB \cdot AD)_{16} = (0101\ 0111\ 1110\ 1011 \cdot 1010\ 1101)_2$

4. Binary to Hexadecimal Conversion:

- To convert a binary number to hexadecimal, the binary numbers are arranged in a group of 4 bits, starting from right to left and by adding zeros to left of MSB if required.
- Each group of four bits is then represented by an equivalent hexadecimal number.
- **Examples:**

(i) $(100111010)_2 = (?)_{16}$

0001 0011 1010
1 3 A

$(100111010)_2 = (13A)_{16}$

(ii) $(100101110 \cdot 11101)_2 = (?)_{16}$

Adding zeros on the left side of MSB and after the decimal point on the right side of LSB,

0001 0010 1110 . 1110 1000
1 2 E E 8

$\therefore \quad (100101110 \cdot 11101)_2 = (12E \cdot E8)_{16}$

- Hexadecimal numbers are used for representing large binary numbers and programming digital equipments. Microprocessors deal with instructions and data which use hexadecimal numbers for programming.

5. Octal to Hexadecimal Conversion

Steps to be follow to convert octal number to hexadecimal.

Step 1 : Represent the given octal number using equivalent 3-bit binary number.

Step 2 : Convert this binary number into equivalent hexadecimal number by grouping 4 binary bits and representing it by equivalent hexadecimal number.

- **Examples:**

(i) $(146)_8 = (?)_{16}$

Step 1: Convert octal to binary.

1 4 6
↓ ↓ ↓
001 100 110

$(146)_8 = (001100\ 110)_2$

Step 2: Convert binary to hexadecimal.

$$\underbrace{0000}_{0}\ \underbrace{0110}_{6}\ \underbrace{0110}_{6}$$

$\therefore \quad (146)_8 = (066)_{16}$

---

6. Hexadecimal to Octal Conversion:

- Steps for converting hexadecimal number to octal number.

Step 1 : Write the 4-bit binary number for each hexadecimal digit.

Step 2 : Form groups of 3 binary bits, starting from LSB.

Step 3 : Represent the 3-bit binary by equivalent octal digit.

- **Examples:**

(i) $(3DB)_{16} = (?)_8$

Step 1: Write the equivalent 4-bit binary number.

$$\begin{array}{ccc} 3 & D & B \\ \downarrow & \downarrow & \downarrow \\ 0011 & 1101 & 1011 \end{array}$$

$(3DB)_{16} = (0011\ 1101\ 1011)_2$

Step 2: Convert binary to octal.

$$\underbrace{001}_{1}\ \underbrace{111}_{7}\ \underbrace{011}_{3}\ \underbrace{011}_{3}$$

$\therefore \quad (3DB)_{16} = (1733)_8$

(ii) $(5CB.12E)_{16} = (?)_8$

Step 1: Converting hexadecimal to binary.

$$\begin{array}{ccccccc} 5 & C & B & . & 1 & 2 & E \\ \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow \\ (0101 & 1100 & 1011 & & 0001 & 0010 & 1110)_2 \end{array}$$

Step 2: Converting binary to octal.

$$(\underbrace{010}_{2}\ \underbrace{111}_{7}\ \underbrace{001}_{1}\ \underbrace{011}_{3} \cdot \underbrace{000}_{0}\ \underbrace{100}_{4}\ \underbrace{101}_{5}\ \underbrace{110}_{6})$$

$\therefore (5CB.12E)_{16} = (2713 \cdot 0456)_8$

(iii) $(78.4B)_{16} = (?)_8$

Step 1: Converting octal to binary.

$$\begin{array}{cccccc} 7 & 8 & . & 4 & B \\ \downarrow & \downarrow & & \downarrow & \downarrow \\ 0111 & 1000 & & 0100 & 1011 \end{array}$$

Step 2: Adding zeros to the left of MSB and to the right of LSB.

$$\underbrace{001}_{1}\ \underbrace{111}_{7}\ \underbrace{000}_{0} \cdot \underbrace{010}_{2}\ \underbrace{010}_{2}\ \underbrace{110}_{6}$$

$\therefore \quad (78.4B)_{16} = (170.226)_8$

1.3 BINARY CODES [S-17, 18, W-17]

- Computers and other digital circuits process data in binary format. To achieve this, a process of coding is required where each alphabet, special character or numeral is coded in a unique combination of 0's and 1's using a coding scheme known as code.
- Various binary codes are used to represent data which may be numeric, alphabets or special characters. Codes are also used for error detection and error correction in digital systems.

Classification of Binary Codes:

1. Weighted Codes:

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

2. 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.

3. Binary Coded Decimal (BCD) Code:

- The numeric codes used to represent the decimal digits in BCD numbers (The numbers obtained after encoding the each digit 0, 1, 2, ..., 9 in decimal number system in the form of sequence of at least four binary digits) are called BCD codes.

4. Alphanumeric Codes:

- The codes which are used to encode the characters of alphabet and decimal digits in addition to special symbols are called alphanumeric codes.
- A binary digit or bit can represent only two symbols as it has only two states '0' or '1'. But this is not enough for communication between two computers because there we need many more symbols for communication. These symbols are required to represent 26 alphabets with capital and small letters, numbers from 0 to 9, punctuation marks and other symbols.
- The following alphanumeric codes are very commonly used for the data representtation.
  (i) EBCDIC (Extended Binary Coded Decimal Interchange Code).
  (ii) ASCII (American Standard Code for Information Interchange).

5. Error-Detecting Codes:

- Error is a condition when the output information does not match with the input information. During transmission, digital signals suffer from noise that can introduce errors in the binary bits travelling from one system to other. That means a 0 bit may change to 1 or a 1 bit may change to 0.
- Whenever, a message is transmitted, it may get scrambled by noise or data may get corrupted. To avoid this, we use error-detecting codes which are additional data added to a given digital message to help us detect if an error occurred during transmission of the message.
- A simple example of error-detecting code is parity check.

6. Error-Correcting Codes:

- Along with error-detecting code, we can also pass some data to figure out the original message from the corrupt message that we received. This type of code is called an error-correcting code.
- Error-correcting codes also deploy the same strategy as error-detecting codes but additionally, such codes also detect the exact location of the corrupt bit.
- In error-correcting codes, parity check has a simple way to detect errors along with a sophisticated mechanism to determine the corrupt bit location. Once the corrupt bit is located, its value is reverted (from 0 to 1 or 1 to 0) to get the original message.

1.3.1 BCD Code

- In BCD code each decimal digit is represented by a 4-bit binary number. BCD is a way to express each of the decimal digits with a binary code. These codes continue the features of decimal and binary.

Advantages of BCD Code:

1. We need to remember binary equivalent of decimal numbers 0 to 9 only.
2. It is very fast and efficient system to convert the decimal numbers into binary numbers.

Disadvantages of BCD Code:

1. The BCD arithmetic is little more complicated.
2. The addition and subtraction of BCD have different rules.
3. BCD needs more number of bits than binary to represent the decimal number. So BCD is less efficient than binary.

Applications of BCD Code:

1. It is used in digital systems such as digital voltmeters, electronic calculators, digital computers etc.
2. It is used for arithmetic operations.
3. It is used to simplify binary number system.

- Two most popular 4-bit BCD Code systems are:
  - Weighted 4-bit BCDcode
  - Excess-3 BCD Code

1.3.1.1 Weighted 4-bit BCD Code

- This is commonly known as 8421 weighted code because it encodes the decimal system into binary number system by using concept of positional weighting into consideration.
- In this code, each decimal digit is encoded into its 4-bit binary number in which the bits from left to right have the weights 8,4,2,1 respectively.
- In the BCD, with four bits we can represent sixteen numbers (0000 to 1111). But in BCD code only first ten of these are used (0000 to 1001).
- The remaining six code combinations i.e. 1010 to 1111 are invalid in BCD.
- Following table shows weighted 4-bit BCD codes for decimal 0 to 9.

Table: BCD Code

Decimal Digit	0	1	2	3	4	5	6	7	8	9
Weighted 4-bit BCD code	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Example: Convert decimal number 5327 in weighted 4-bit BCD code

Solution: 5 → 0101
3 → 0011
2 → 0010
7 → 0111

Result is 0101 0011 0010 0111

Example: Convert BCD code 0100 1001 into decimal number.

Solution: 0100 1001
4 9

Result is $(49)_{10}$

1.3.1.2 Excess-3 Code [S-18]

- It is a self complementary binary-coded decimal code and numeral system.
- The non-weighted BCD code which is derived from 8421 BCD by adding 3 to each coded number is called Excess-3 code.
- The Excess-3 code is also called as XS-3 code or 3XS or X3 code.
- It is non-weighted code used to express decimal numbers. This is non- weighted code because it does not use the principle of positional weights into consideration while converting the decimal numbers to 4-bit BCD code System.
- The Excess-3 code words are derived from the 8421 BCD code words adding $(0011)_2$ or $(3)_{10}$ to each digit of number and then convert the Excess digits.
- The excess-3 codes are obtained as shown in Fig. 1.5.

Decimal Number → 8421 BCD —(Add 0011)→ Excess-3

Fig. 1.5: Excess-3 Code from Decimal Number

Table: Excess-3 BCD Codes

Decimal	BCD	Excess-3 BCD
	8 4 2 1	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

Example: Convert the decimal number 85 to XS-3 BCD Code.

Solution: Number = 85
Now add 3 to each digit,
8+3 = 11→ 1011
5+3= 8→ 1000
Result is 1011 1000

1.3.2 8-bit BCD Code Systems

- To overcome the limitations of 4-bit and 6-bit BCD codes, 8-bit BCD code systems have been developed.
- This system can handle numeric as well as non-numeric data with almost all the special characters such as +,-,*,/.@.$. etc., therefore the various codes under this category are also known as alphanumeric codes.
- The three most popular codes are:
  1. Gray Code.
  2. ASCII
  3. EBCDIC

1.3.2.1 Gray Code [S-18]

- Gray Code 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. 1.6.
- As only one bit changes at a time, the gray code is called as a unit distance code or a cyclic code. Gray code also called as Reflected Binary Code (RBC) after Frank Gray.
- Following table shows comparison of 8421 code and Gray code.

Decimal	8421 Code	Gray
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1
11	1 0 1 1	1 1 1 0
12	1 1 0 0	1 0 1 0
13	1 1 0 1	1 0 1 1
14	1 1 1 0	1 0 0 1
15	1 1 1 1	1 0 0 0

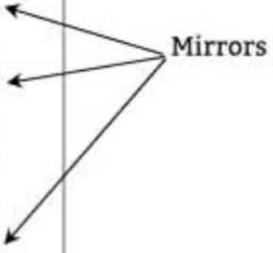


Fig. 1.6: Gray Code and 8421 Code

- The main advantage of Gray code is its ease of conversion to and from binary numbers.
- The drawback of Gray code is that it cannot be used for any arithmetic operations.

Applications of Gray Code:

It is used in instrumentation and data acquisition systems where linear or angular displacement is measured.

It is used in shaft encoders, input and output devices, A to D converters and other peripheral devices.

- We can convert the Gray coded number to its binary equivalent by remembering the following two major rules:

Rule 1 : The Most Significant Bit (MSB) of the Gray coded number and the equivalent binary number is always the same.

Rule 2 : The next-to-most significant bit of the binary number can be determined by adding the MSB of the binary number to the next-to-most significant bit of the gray coded number.

Example: Convert the Gray coded number 10101110 to its binary equivalent.

Solution: The given Gray coded number is 10101110.

The following table lists the steps showing the conversion of the Gray coded number into its binary equivalent:

Sr. No.	Gray coded digit	Binary addition operation	Binary digit
1.	1		1
2.	0	0 + 1	1
3.	1	1 + 1	0
4.	0	0 + 0	0
5.	1	1 + 0	1
6.	0	1 + 1	0
7.	1	1 + 0	1
8.	0	0 + 1	1

Hence, the binary equivalent of Gray coded number 10101110 is 11001011.

We can also convert a number represented in the binary form to Gray coded representation. For carrying out this conversion, we need to remember the following two rules:

Rule 1 : The Most Significant Digit (MSD) of the binary number and the gray coded number is always the same.

Rule 2 : The next MSD of the gray coded number can be obtained by adding the subsequent pair of bits of the binary number starting form the left.

Note: We need to ignore the carry, if it is generated white adding the subsequent pairs of the bits the binary number.

Example: Convert the binary number 01010010 to its equivalent Gray coded number.

Solution: The given binary is 01010010.

The following table lists the steps showing the conversion of binary number to its equivalent Gray coded number:

Sr. No.	Binary	Binary addition operation	Gray Coded
1.	0		0
2.	1	0 + 1	1
3.	0	1 + 0	1
4.	1	0 + 1	1
5.	0	1 + 0	1
6.	0	0 + 0	0
7.	1	0 + 1	1
8.	0	1 + 0	1

Hence, the Gray coded equivalent of binary number 01010010 is 01111011.

Comparison of BCD Code and Gray Code:

Sr. No.	BCD Code	Gray Code
1.	It is a weighted code.	It is an unweighted code.
2.	Each decimal is represented by their binary equivalent using 4 bits.	A decimal digit is not directly represented in gray code, but it is first converted into binary and then into gray code.
3.	Many bits can be changed in times.	Only one bit can be changed at a time.
4.	It is good to simplify binary number system.	It is used for converting a data from its analog to digital form.
5.	It is used in arithmetic process.	It is not used in arithmetic process, but used in A/D (Analog to Digital) converter.

1.3.2.2 ASCII Code [W-17]

- The name of the code ASCII (American Standard Code for Information Interchange) and this is pronounced as 'ask-ee'.
- This code used to represent alphanumeric data in computers, in communication equipment, in electronic device to represent the input and output in a more scientific manner.
- The ASCII code set uses 7 bits per character, allowing 128 different characters. This is enough for the English alphabet in uppercase and lowercase, the symbols on a regular English typewriter, and some combinations reserved for internal use.

- An extended ASCII code set uses 8 bits per character, which adds another 128 possible characters. This larger code set allows for foreign languages symbols like letters with accents and several graphical symbols.
- This code used to handle 128 characters but later it was modified to an 8-bit code. For example: We can check the value of any ASCII code by just holding down the ALT key and typing the ASCII code. For example, when we hold down the ALT key and type 66 from the keyboard, then the character B appears on the screen. This shows that the ASCII decimal code 66 represents the character B.

1.3.2.3 EBCDIC Code

- The Extended Binary Coded Decimal Interchange Code (EBCDIC) pronounced as "ebb-see-dick" is another frequently used code by computers for transferring alpha-numeric data.
- The EBCDIC code is an 8-bit alphanumeric code that was developed by IBM to represent alphabets, decimal digits and special characters, including control characters.
- Control characters are the special characters that are used to perform a specific function. For example, the control character FF is used to feed the next page into the printer or eject the current page from the printer.
- The EBCDIC codes are generally the decimal and the hexadecimal representation of different characters. This code is rarely used by non IBM-compatible computer systems.
- Following table lists ASCII - 7 Bit Code and 8 Bit EBCDIC Code:

Character	6-bit Internal Code	7-bit ASCII Code	8-bit EBCDIC Code
A	010001	1000001	11000001
B	010010	1000010	11000010
C	010011	1000011	11000011
D	010100	1000100	11000100
E	010101	1000101	11000101
F	010110	1000110	11000110
G	010111	1000111	11000111
H	011000	1001000	11001000
I	011001	1001001	11001001
J	100001	1001010	11010001
K	100010	1001011	11010010
L	100011	1001100	11010011
M	100100	1001101	11010100
N	100101	1001110	11010101
O	100110	1001111	11010110

contd. ...

P	100111	1010000	11010111
Q	101000	1010001	11011000
R	101001	1010010	11011001
S	110010	1010011	11100010
T	110011	1010100	11100011
U	110100	1010101	11100100
V	110101	1010110	11100101
W	110110	1010111	11100110
X	110111	1011000	11100111
Y	111000	1011001	11101000
Z	111001	1011010	11101001
0	000000	0110000	11110000
1	000001	0110001	11110001
2	000010	0110010	11110010
3	000011	0110011	11110011
4	000100	0110100	11110100
5	000101	0110101	11110101
6	000110	0110110	11110110
7	000111	0110111	11110111
8	001000	0111000	11111000
9	001001	0111001	11111001
Blank	110000	0100000	01000000
	011011	0101110	01001011
(	111100	0101000	01001101
+	010000	0101011	01001110
*	101100	0101010	01011100
$	101011	0100100	01011011
)	011100	0101001	01011101
/	110001	0101111	01100001
'	111011	0111100	01101011
"	001011	0111101	01111110
–	100000	0101101	01100000

1.3.3 Unicode

- The ASCII and EBCDIC encodings and their variants have some limitations:
  1. These encodings do not have a sufficient number of characters to be able to encode alphanumeric data of all forms, scripts and languages. As a result, they do not permit multilingual computer processing.

2. These encoding suffer from incompatibility. For example: code 7A (in hex) represents the lowercase letter 'Z' in ASCII code and the semicolon sign ';' in EBCDIC code.

- To overcome these limitations, UNICODE is developed, it is also known as Universal Code was developed jointly by the Unicode Consortium and the International Organization for Standardization (ISO). It may be 8 bit, 16 bit or 32 bit.
- The 16-bit Unicode is an International 16-bit character set that contains a maximum of 2^{16} = 65,536 different characters. These characters are sufficient to represent almost all the technical and special symbols used by the major languages of the world.
- The 16-bit Unicode, (also called 16-bit Universal Character Set), encodes the different characters by assigning them a unique value. In computer terminology, this unique value is referred as Code Point.
- The l6-bit Unicode is a character code that is supported by almost all the operating systems such as MS Windows, Linux and Mac OS X. For example, MS Windows operating system allows the use of all the Unicode characters through an accessory program called 'Character Map'.
- Unicode also uses 32 bits to represent a symbol in the data and allows 2^{32} = 4164895296 (~ 4 billion) combinations.
- Unicode covers almost all characters used in various languages used in the world. Unicode is compactable with ASCII and extended ASCII text codes.

1.3.4 Concept of Parity Code [S-18]

- An error in a digital system is the corruption of data from its correct value to some other value. The Parity Code is an error checking code in which binary numbers are transmitted with an additional bit that is used for error detection.
- It is a simplest code, where number of check bit is 1 only and the check bit is derived to generate a particular parity in the entire codeword. In other words, Parity refers to the number of 1s in the binary word.
- Parity may be of two types as explained below:
  1. **Odd Parity:** The parity of a binary word is 'odd' if it contains an odd number of 1's. So, the check bit (parity bit) in an odd-parity code is so chosen to give the resulting codeword an odd parity.
  2. **Even Parity:** The parity of a binary word is known as 'even' if it contains even number of 1's. So, for even-parity code, the check bit is so chosen that whatever may be the number of 1's in the K number of information bits, the codeword contains an even number of 1's.
- The following example explains the Parity Check Code:

Messages	Even-parity Code		Odd-parity Code	
	Checkbit	Codeword	Checkbit	Codeword
0110100	1	01101001	0	01101000

- If a single error occurs in the message at receiver, the simple parity check code can detect that error. However, it cannot detect the position of that error. Hence, it cannot correct that error. So, simple parity code is basically an error detecting code and is used in detection and transmission type of schemes.

1.4 SIGNED AND UNSIGNED NUMBERS

- The numbers having either positive (+) or negative (–) signs are called signed numbers. The numbers does not contain any positive (+) or negative (–) signs are called unsigned numbers.
- In computer systems, numbers can be represented in two ways, unsigned representation and signed representation.
- The binary number system can be used to represent the following two types of numbers:
  1. **Signed Number:** In this representation, the Most Significant Bit (MSB) of the number represents the sign of the number. In a number, if the value of MSB is 0 then the number is considered as a positive number and if the value of MSB is 1 then the number is considered as a negative number. In signed number representation, the remaining bits show the absolute value of the number. For example, if we represent an 8-bit number as a signed number then the MSB of the number represents the sign of the number and the remaining 7 bits represents the absolute value of the number that ranges from 0 to 127.
  2. **Unsigned Number:** In this representation, the number does not consist of any sign bit and therefore all the 8 bits represent the value of the number.

Bit Representation	Unsigned	Signed
00000000	0	+0
00000001	1	+1
.....	.....	.....
.....	.....	.....
.....	.....	.....
01111111	127	+127
10000000	128	–0
10000001	129	–1
.....	.....	.....
.....	.....	.....
.....	.....	.....
11111111	255	–127

Fig. 1.7: Signed and Unsigned Numbers

1.5 1'S AND 2'S COMPLEMENTS OF BINARY NUMBERS

- As the binary system has base r = 2. So the two types of complements for the binary system are 2's complement and 1's complement.

- 1's complement of binary number is that number which is obtained from a positive binary number by complementing all the bits, that is replacing each 1 by 0 and each 0 by 1, while 2's complement of binary number is that number which is obtained by adding 1 to the '1' complement of the binary number.

1's Complement of Binary Numbers:

- The 1's complement of a number is found by changing all 1's to 0's and all 0's to 1's. 1's complement is called as taking complement or 1's complement.
- Example, of 1's Complement is as shown in Fig. 1.8.

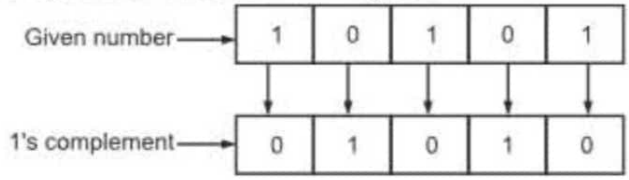


Fig. 1.8: 1's Complement of Binary Number

2's Complement of Binary Numbers:

- The 2's complement of binary number is obtained by adding 1 to the Least Significant Bit (LSB) of 1's complement of the number.
- 2's complement = 1's complement + 1

  Example, of 2's Complement is as shown Fig. 1.9.

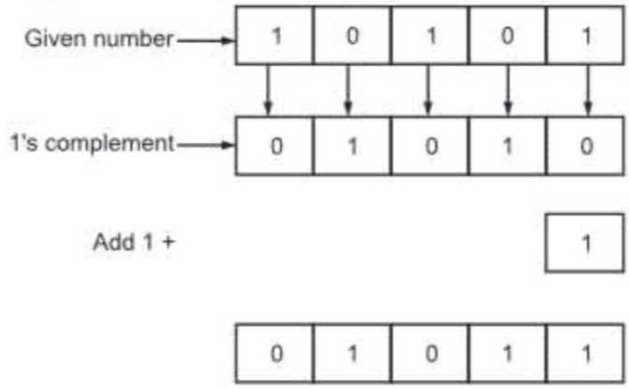


Fig. 1.9: 2's Complement of Binary Number

1.6 BINARY ARITHMETIC (Addition, Subtraction, Subtraction Using 1's Complement and Subtraction Using 2's Complement)

[S-17, W-17]

- Binary arithmetic is essential part of all the digital computers and many other digital systems. The majority of arithmetic performed by computers is binary arithmetic, i.e., arithmetic on base two numbers, (0 and 1).
- The computer arithmetic is also referred as binary arithmetic because the computer system stores and processes the data in the binary form only.
- Various binary arithmetic operations can be performed in the same way as the decimal arithmetic operations, but by predefined set of rules.

1. Binary Addition

- It is a key for binary subtraction, multiplication, division.
- There are four rules of binary addition as given in following table:

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:

0011010 + 001100 = 00100110

```
   11          carry
  0011010   = (26)10
+ 0001100   = (12)10
  0100110   = (38)10
```

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 as given in following table:

Case	A − B	Difference	Borrow
1	0 − 0	0	0
2	0 − 1	1	1
3	1 − 0	1	0
4	1 − 1	0	0

Example:

0011010 − 001100 = 00001110

```
    11          borrow
  00~~1~~1010   = (26)10
- 0001100       = (12)10
  0001110       = (14)10
```

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 as given in the following table:

Case	A × B	Product
1	0 × 0	0
2	0 × 1	0
3	1 × 0	0
4	1 × 1	1

Example:

0011010 × 001100 = 100111000

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

4. Binary Division:

- Binary division is similar to decimal division. It is called as the long division procedure.

Example:

101010 / 000110 = 000111

$$
\begin{array}{rll}
 & 111 & = 7_{10} \\
000110 \overline{)} & 101010 & = 42_{10} \\
 & -\ 110 & = 6_{10} \\
\hline
 & 1001 & \\
 & -110 & \\
\hline
 & 110 & \\
 & -110 & \\
\hline
 & 0 &
\end{array}
$$

5. Subtraction by 1's Complement:

- 1's complemented value represents the negative of the original binary number. This system is extremely simple to implement in computer hardware by only feeding all the bits through transistor invertor circuits.
- The advantage of this method is that a negative binary number may be added to a positive number in a direct manner.
- Procedure adopted for subtraction by 1's complement is as follows:

Step 1 : Find 1's complement of the binary number to be subtracted, i.e. subtrahend.

Step 2 : Add this subtrahend to the minuend.

Step 3 : If there is a carry in the extreme left position (MSB), then transfer it to the extreme right position (LSB). This is called End Around Carry (EAC).

Step 4 : Add this end around carry to the remainder. Ignore the carry 1 in its original position.

Step 5 : If there is no end around carry, the answer is negative and it is in the 1's complement form, while the answer is positive, if there is carry 1.

- **Examples:**

(i) $13_{10} - 12_{10}$.

13 →	001101	001101	
−12 →	−001100	+110011	(1's complement)
01		1000000	
		+ ↘ 1	End around carry
		000001	

As there is a carry 1, the result $(000001)_2$ is positive.

(ii) $-12_{10} - 13_{10}$.

−12 →	110011	(1's complement of $12_{10} = 001100_2$)
−13 →	+ 110010	(1's complement of $13_{10} = 001100_2$)
−25	1100101	
	+ ↘ 1	End around carry
	100110	

The number $(100110)_2$ is $-(25)_{10}$ in 1's complement form because the complement of $(100110)_2$ is $(011001)_2$ which is $(25)_{10}$.

6. Subtraction by 2's Complement:

- The problem of end around carry in addition (or subtraction) in 1's complement is overcome by using 2's complement for negative binary numbers. Further the ambiguity of positive and negative numbers is also resolved. Because of these reasons, most digital computers use 2's complement method.
- The 2's complement represents the negative binary number.
- The advantage of this method is that the subtraction can be carried out just like addition and no additional circuitry is needed.
- The procedure followed in subtraction 2's complement method is in this method are as follows:

Step 1 : Find the 1's complement of the binary number to be subtracted.

Step 2 : Add 1 to the above and get the 2's complement of the number.

Step 3 : Add the two numbers, i.e. the minuend and the 2's complement of subtrahend. Ignore the last carry in MSB position.

Examples:

(i) $13_{10} - 12_{10}$

13 →	001101	001101			001101	
−12 →	−001100	+110011	(1's complement)		+110100	(2's complement)
01					1000001	Ignore last carry

Hence, the result $(000001)_2$, i.e. $(1)_{10}$.

(ii) $-13_{10} - 12_{10}$

−13 →	001101	110010	(1's complement)		110011	(2's complement)
−12 →	−001100	+110011	(1's complement)		+110100	(2's complement)
−25					1100111	Ignore last carry

Hence, the result $(1100111)_2$ is − (25) in 2's complement form because 2's complement $(011001)_2$ is $(100111)_2$ i.e., $(25)_{10}$.

Summary

- Data is represented and stored in a computer using groups of binary digits called words.
- In binary number system, there are 2^n different unique bit patterns with an n-bit word that can be used to represent information.
- Hexadecimal numbers are formed using four-bit groups called nibbles (or nybbles).
- Binary code is represented entirely by a binary system of digits consisting of a string of consecutive zeros and ones.
- Gray code is defined as an ordering of the binary number system such that each incremental value can only differ by one bit.
- Excess-3, also called XS3, is a non-weighted code used to express decimal numbers.
- The most common alphanumeric codes used these days are ASCII code, EBCDIC code and Unicode.
- In error-correcting codes, parity check has a simple way to detect errors along with a sophisticated mechanism to determine the corrupt bit location.
- 1's complement of a binary number is another binary number obtained by toggling all bits in it. 2's complement of a binary number is 1 added to the 1's complement of the binary number.
- Binary arithmetic is essential part of all the digital computers and many other digital systems. It is possible to add and subtract binary numbers in a similar way to base 10 numbers.
- Signed integers can be stored in one's complement; two's complement, or signed magnitude representation.
- Floating-point numbers are usually coded using the IEEE 754 floating-point standard.
- Character data is stored using ASCII, EBCDIC, or Unicode.

Check Your Understanding

1. Which system is used to refer amount of things _____.
   (a) Number words (b) Number system
   (c) Number symbols (d) All of these
2. MSB stand for _____
   (a) Most significant bit (b) Many significant bit
   (c) Both a and b (d) None of these
3. _____ of a number from another can be accomplished by adding the complement of the subtrahend to the minuend:
   (a) Subtraction (b) Multiply
   (c) Divide (d) All of these
4. A number system that uses eight digits,0,1,2,3,4,5,6, and 7 is called an _____.
   (a) Binary number system (b) Decimal number system
   (c) Octal number system (d) None of these

5. Counting in hex, each digit can be increment from _____.
   (a) 0 to J (b) 0 to G
   (c) 0 to H (d) 0 to F

Answers

1. (b)	2. (a)	3. (a)	4. (c)	5. (d)

Practice Questions

Q.1 Answer the following Question in brief:

1. What is number system?
2. Enlist types of number system.
3. Explain binary number with example.
4. Explain octal number with example.
5. Explain decimal number with example.
6. Explain hexadecimal number with example.

Q.2 Answer the following Question:

1. Describe 1's complement of binary system in detail.
2. Explain 2's complement of binary system in detail.
3. Write short notes on Excess-3 Code.
4. What are ASCII and EBCDIC code?
5. Explain binary arithmetic in detail.
6. Describe Gray Code with example.
7. Explain concept of Parity Code.
8. What are the types of binary codes? Explain two of them in detail.
9. What is Unicode?
10. Convert the following:
    (i) $(23)_{10} = (?)_2$
    (ii) $(10101101)_2 = (?)_{10}$
    (iii) $(100110)_2 = (?)_8$
    (iv) $(EBC)_{16} = (?)_2$
    (v) $(59.6825)_{10} = (?)_2$
11. Solve the following:
    (i) 100111 + 11011
    (ii) 10101 – 01110
    (iii) 1010×1001
    (iv) $100001 \div 110$

Q.3 Define Terms:

Radix, Parity code, Binary code, Weighted code, Error detecting code, Gray code, 1's complement, 2's complement, Parity, Error Correcting code, Signed number.

Previous Exam Questions

Summer 2017

1. The binary equivalent of $(11.55)_{10}$ is _____ **(1 M)**

(a) 1011.1001 (b) 1011.100001

(c) 101.10001 (d) 1101.10001

Ans. Refer to Section 1.2.2.

2. Explain weighted and unweighted codes. **(3 M)**

Ans. Refer to Section 1.3.

3. Convert the following: **(3 M)**

(i) $(701.108)_8 = (?)_{16}$

(ii) $(3A2.5B)_{16} = (?)_{10}$

(iii) $(85.CA)_{16} = (?)_8$

Ans. Refer to Section 1.2.4.

4. Solve the following: **(4 M)**

(i)
$$\begin{array}{r} 1111 \cdot 1101 \\ 10111 \cdot 1110 \\ +\ 100 \cdot 1010 \\ \hline \\ \hline \end{array}$$

(ii)
$$\begin{array}{r} 110011.10 \\ -\ 11110.11 \\ \hline \\ \hline \end{array}$$

(iii) $11110 \div 10$

(iv)
$$\begin{array}{r} 11.01 \\ \times\ 11.10 \\ \hline \\ \hline \end{array}$$

(4 M)

Ans. Refer to Section 1.6.

Winter 2017

1. The decimal equivalent of $(821)_8$ is _____ **(1 M)**

(a) 592 (b) 529

(c) 630 (d) 295

Ans. Refer to Section 1.2.3.

2. Convert the following **(4 M)**

(a) $(AF9.BOD)_{16}=(?)_2$ (b) $(23.85)_{10} = (?)_2$

(c) $(1101101)_2 = (?)_{10}$ (d) $(457.65)_8 = (?)_{10}$

Ans. (a) $(AF9.BOD)_{16}=(?)_2$: Refer to Section 1.2.4.
(b) $(23.85)_{10} = (?)_2$: Refer to Section 1.2.2.
(c) $(1101101)_2 = (?)_{10}$: Refer to Section 1.2.2.
(d) $(457.65)_8 = (?)_{10}$: Refer to Section 1.2.3.

3. What are character codes? Explain ASCII code with example? **(3 M)**

Ans. Refer to Section 1.3.2.2.

4. Solve the following: **(3 M)**
(i) 11011.101 + 1010.111
(ii) Perform 11110 – 1010 using 2's complement

Ans. Refer to Section 1.6.

Summer 2018

1. Which of the following is the example of weighted number system? **(1 M)**
(a) Gray (b) Hexadecimal
(c) ASCII (d) EBCDIC

Ans. Refer to Section 1.3.

2. Write a short note on weighted and unweighted codes with an example. **(4 M)**

Ans. Refer to Section 1.3.

3. Perform the following conversions. **(4 M)**
(a) $(234)_{10} = (?)_{16}$ (b) $(142)_{10} = (?)_2$
(c) $(1011)_2 = (?)_{Gray}$ (d) $(1101)_{Gray} = (?)_2$

Ans. (a) $(234)_{10} = (?)_{16}$: Refer to Section 1.2.4.
(b) $(142)_{10} = (?)_2$: Refer to Section 1.2.2.
(c) $(1011)_2 = (?)_{Gray}$: Refer to Section 1.3.2.1.
(d) $(1101)_{Gray} = (?)_2$: Refer to Section 1.3.2.1.

4. Define term: Parity **(1 M)**

Ans. Refer to Section 1.3.4.

5. Write a note on Excess-3 code. **(3 M)**

Ans. Refer to Section 1.3.1.2.

■■■

2... Boolean Algebra and Logic Gates

Objectives...

- To learn about different Logic Gates and Boolean Algebra.
- To study about De-Morgan's Theorems and Inter-conversion of Logic Gates.
- To understand the concepts of Minterm, Maxterm and Karanaugh Maps.

2.1 INTRODUCTION

- Boolean algebra and De Morgan's theorems are used for simplifying the logical expressions. The logical expression can be implemented by using logic gates.
- Signal can be defined as, "a physical quantity, which contains some information". Signals are of two types i.e., Analog signal and Digital signal.
- Analog signal can have infinite number of different values. In real world scenario, most of the things observed in nature are analog. Examples of the analog signals are temperature, pressure, distance, sound, voltage, current and power.
- A digital signal is defined as, "the signal which has only a finite number of distinct values, i.e. 0 and 1". These signals are not continuous signals. The examples of digital signals are square waveforms.

Comparison of Analog and Digital Signal:

Sr. No.	Analog Signal	Digital Signal
1.	Analog signal has a continuous nature.	Digital signal has a discrete nature.
2.	Analog signal has infinite values.	Digital signal has a finite number of values.
3.	It can take infinite values within the specified range of time.	It can take only two discrete values within the specified range of time.
4.	Analog signal is generated by transducers and signal generators.	Digital signal is generated by A to D converter.
5.	Examples of analog signal are sine wave, triangular waves.	Example of digital signal is square wave.

Some Basic Terms:

1. **Logic:** Logic is defined as, "a statement which is true, if some condition is satisfied and false, if that condition is not satisfied". The logic which has only two possible outcomes is called Binary logic. In binary logic, two voltage levels represent the two binary digits, 1 and 0. If the higher of the two voltages represents a 1 and the lower voltage represents a 0, the system is called a positive logic system. On the other hand, if the lower voltage represents a 1 and the higher voltage represents a 0, we have a negative logic system.
2. **Boolean Expression:** The operation of a logic gate can be explained by writing its logic (or expression) equation using Boolean algebra. It is also called as Boolean equation (or expression). The mathematical relation between the inputs and the outputs with reference to Boolean operator is called a Boolean Expression. The Boolean equation has three basic elements:
   (i) Input variables.
   (ii) Output variables.
   (iii) Boolean operator.
3. **Truth Table:** A truth table defines the function of a logic gate by providing a list that shows all the output states in tabular form for each possible combination of input variable.
4. **Logic Diagram:** This is a graphical representation of a logic circuit that shows the wiring and connections of each individual logic gate. This is represented by a specific graphical symbol that implements the logic circuit.
5. **Graphical Symbols:** It is a pictogram used to show various electrical and electronic devices or functions.

2.2 BINARY LOGIC [S-17, W-17]

- In binary logic, logic 1 and logic 0 voltage levels represents two binary digits. The high and low voltage levels can be used to represent logic 1 and logic 0 levels, by using two types of logic systems viz.
  1. **Positive Logic System:** Logic 1 level represents a more positive of the two voltage levels while the least positive of the two voltage levels represents a logic 0 level.
     For example, if+5 V represents a logic 1 level and 0 V represents a logic 0 level.
  2. **Negative Logic System:** Logic 1 level represents a most negative of the two voltage levels, while the least negative of the two voltage levels represents a logic 0 level.
     For example, if 0 V represents a logic 1 level and +5 V represents a logic 0 level.

2.3 LOGIC GATES

- Logic gates are the basic building blocks of any digital system. Logical gates are the electronic devices that make logical decisions based on the different combinations of digital signals present on its inputs.
- A logic gate 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 certain logic.
- Based on this logic the logic gates can be classified as shown in Fig. 2.1.

- There are total 7 logic gates. Among all the gates AND, OR, NOT gates are Basic gates because these gates have basic operations such as sum, multiplication and inverse (compliment) operation respectively.
- NOR and NAND gates are known as universal gates. We will see explanation about these universal gates in next section.

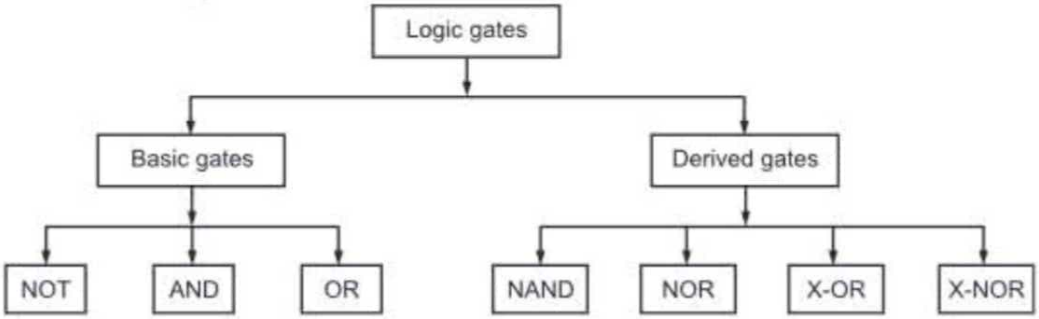


Fig. 2.1: Types of Logic Gates

2.3.1 AND Gate [W-17]

- The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high.
- A dot (·) is used to show the AND operation. The logic equation of AND gate is $Y = A \cdot B$.
- Fig. 2.2 shows logic symbol of AND gate.

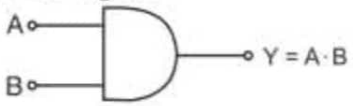


Fig. 2.2: Symbol of AND Gate

- Following table shows truth table for AND gate.

Inputs		Output
A	**B**	**Y = A · B**
0	0	0
0	1	0
1	0	0
1	1	1

2.3.2 NOT Gate [S-17]

- The NOT gate is one of the basic logic gate. Its equivalent in Boolean algebra is known as NOT operator.
- The NOT gate is called an inverter because it only inverts the input, i.e. logic 1 state at the input becomes logic 0 at the output and vice-versa.
- The NOT gate is an electronic circuit with only one input and one output signal, the output is always complement of the input.

- The logic equation for NOT gate is Y = NOT Y= $\overline{Y}$. It is read as "Y equals NOT Y" or "Y equals complement of A".
- The NOT operation is also referred as an inversion or complementation. A small circle in symbol known as bubble. It shows inversion in digital circuits.
- Fig. 2.3 shows symbol of NOT gate.

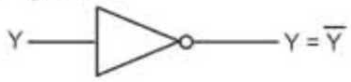


Fig. 2.3: Symbols of NOT Gate

- Following table shows truth table of NOT gate.

Input	Output
Y	**$Y = \overline{Y}$**
0	1
1	0

2.3.3 OR Gate [S-17]

- OR gate is an electronic logic circuit in which the output is in logic 1 state, when any one or both the inputs are in logic 1 state.
- A plus (+) is used to show the OR operation.
- Fig. 2.4 shows logic symbol for OR gate.

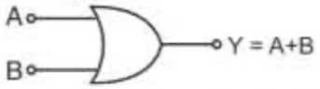


Fig. 2.4: Symbol of OR Gate

- Logic equation of OR gate is Y = A + B. It is read as "Y equals A OR B"
- Following table shows truth table of OR gate.

Inputs		Output
A	**B**	**Y = A+B**
0	0	0
0	1	1
1	0	1
1	1	1

- The logic gates which are derived from the basic logic gates are called derived logic gates.

2.4 BOOLEAN THEOREM

- **Boolean theorems** and **laws** are used to simplify the various logical expressions.
- In a digital designing problem, a unique logical expression is evolved from the truth table. If this logical expression is simplified the designing becomes easier.

- The Boolean algebra is mainly used in set theory and digital electronics. It is also used in all modern programming languages.
- There are few basic laws and theorems of Boolean algebra.
- Following table shows List of Boolean Theorems.

Table 2.1: Boolean Algebraic Theorems

Theorem No.	Theorem
1.1	$A + 0 = A$
1.2	$A \cdot 1 = A$
1.3	$A + 1 = 1$
1.4	$A \cdot 0 = 0$
1.5	$A + A = A$
1.6	$A \cdot A = A$
1.7	$A + \bar{A} = 1$
1.8	$A \cdot \bar{A} = 0$
1.9	$A \cdot (B + C) = AB + AC$
1.10	$A + BC = (A + B)(A + C)$
1.11	$A + AB = A$
1.12	$A(A + B) = A$
1.13	$A + \bar{A}B = (A + B)$
1.14	$A(\bar{A} + B) = AB$
1.15	$AB + A\bar{B} = A$
1.16	$(A + B) \cdot (A + \bar{B}) = A$
1.17	$AB + \bar{A}C = (A + C)(\bar{A} + B)$
1.18	$(A + B)(\bar{A} + C) = AC + \bar{A}B$
1.19	$AB + \bar{A}C + BC = AB + \bar{A}C$
1.20	$(A + B)(\bar{A} + C)(B + C) = (A + B)(\bar{A} + C)$
1.21	$\overline{A \cdot B \cdot C \cdot \ldots} = \bar{A} + \bar{B} + \bar{C} + \ldots$
1.22	$\overline{A + B + C + \ldots} = \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \ldots$

- Theorems 1.9 to 1.20 involve more than one variable and can be proved by making a truth table. For example, Theorem 1.10 can be proved by making the truth table given in Table.

Table 2.2: Truth Table to Prove Theorem 1.10

A	B	C	BC	A + BC	A + B	A + C	(A + B) · (A + C)
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

- From the table we observe that there are 8 (= 2^3) possible combinations of the three variables A, B and C. For each combination, the value of A + BC is the same as that of (A + B) (A + C), which proves the theorem. Theorems 1.21 and 1.12 are known as De Morgan's theorems. These theorems can be proved by first considering the two variable case and then extending this result.

2.5 BOOLEAN LAWS [S-17, W-17]

- Boolean algebra was invented by George Boole in 1854.
- Boolean Algebra is used to analyze and simplify the digital (logic) circuits. It uses only the binary numbers i.e. 0 and 1.
- Boolean algebra differs from ordinary algebra in a way that Boolean constants and variables are allowed to have only two values (0 or 1).
- These variables represent a voltage level of a circuit. Boolean variables 0 and 1 do not represent the actual number but they represent the state of the voltage variable.
- There are three basic operations in Boolean algebra:
  1. Logical addition (+) or OR operation.
  2. Logical multiplication (·) or AND operation.
  3. Logical inversion (–) or NOT operation.

Rules used in Boolean Algebra:

- Variable used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.
- Complement of a variable is represented by an overbar (–). Thus, complement of variable B is represented as $\overline{B}$. Thus if B = 0 then $\overline{B}$ = 1 and B = 1 then $\overline{B}$ = 0.
- 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.
- Logical ANDing of the two or more variable is represented by writing a dot between them such as A·B·C. Sometimes the dot may be omitted like ABC.

- Anything ANDed with a 0 is equal to 0. $A \cdot 0 = 0$.
- Anything ANDed with a 1 is equal to itself. $A \cdot 1 = A$.
- Anything ORed with a 0 is equal to itself. $A + 0 = A$.
- Anything ORed with a 1 is equal to 1. $A + 1 = 1$.
- Anything ANDed with itself is equal to itself. $A \cdot A = A$.
- Anything ORed with itself is equal to itself. $A + A = A$.

Boolean Laws:

- There are six types of Boolean Laws as explained below:

1. Commutative Law:

Statement:

- Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

  Using OR operator → $A + B = B + A$

  Using AND operator → $A \cdot B = B \cdot A$

- This law is also has more priority in Boolean algebra.

Example:

Take 2 variables 1 and 0, then

$1 + 0 = 0 + 1$

$1 = 1$

Similarly,

$1 \cdot 0 = 0 \cdot 1$

$0 = 0$

2. Associative Law:

- This law states that the order in which the logic operations are performed is irrelevant as their effect is the same.

(a) Associate Law of Addition:

Statement:

- Associative law of addition states that ORing more than two variables i.e. mathematical addition operation performed on variables will return the same value irrespective of the grouping of variables in an equation. It involves in swapping of variables in groups.

  The Associative law using OR operator can be written as,

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

(b) Associate Law of Multiplication:

Statement:

- Associative law of multiplication states that ANDing more than two variables i.e. mathematical multiplication operation performed on variables will return the same value irrespective of the grouping of variables in an equation.
- The Associative law using AND operator can be written as,

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

3. Distributive Law:

- This law is the most used and most important law in Boolean algebra, which involves 2 operators: AND, OR.

Statement 1:

- The multiplication of two variables and adding the result with a variable will result in same value as multiplication of addition of the variable with individual variables.
- In other words, ANDing two variables and ORing the result with another variable is equal to AND of ORing of the variable with the two individual variables.
- Distributive law can be written as

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

This is called OR distributes over AND.

- The distributive law can be understood by the corresponding logic equivalence shown below.
- The four basic identities of OR operations are:

(i) $A + 0 = A$

(ii) $A + 1 = 1$

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

(iv) $A + A = A$

- The authentication of the above all equations can be checked by substituting the value of $A = 0$ or $A = 1$.
- The four basic identities of AND operations is given below.

(i) $A \cdot 1 = A$

(ii) $A \cdot A = A$

(iii) $A \cdot 0 = 0$

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

- One can check the validity of the above identities by substituting the value of $A = 0$ or $A = 1$

4. AND Law:

- These laws use the AND operation. Therefore they are called as AND laws.

(i) $A \cdot 0 = 0$ (ii) $A \cdot 1 = A$ (iii) $A \cdot A = A$ (iv) $A \cdot \bar{A} = 0$

5. OR Law:

- These laws use the OR operation. Therefore they are called as OR laws.

(i) $A + 0 = A$ (ii) $A + 1 = 1$ (iii) $A + A = A$ (iv) $A + \bar{A} = 1$

6. Double Inversion Law:

- This law uses the NOT operation. The inversion law states that double inversion /complement of variable results in the original variable itself. The double inversion law is shown by the equation below,

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

2.6 De MORGAN'S THEOREMS [S-18]

- De Morgan's Theorem is mainly used to solve the various Boolean algebra expressions.
- The De Morgan's theorem defines the uniformity between the gate with same inverted input and output.
- It is used for implementing the basic gate operation likes NAND gate and NOR gate. The De Morgan's theorem mostly used in digital programming and for making digital circuit diagrams.
- There are two De Morgan's Theorems. They are described below in detail.

De Morgan's First Theorem:

- According to De Morgan's first theorem, a NOR gate is equivalent to a bubbled AND gate.
- The Boolean expressions for the bubbled AND gate can be expressed by the equation shown below.
- For NOR gate, the equation is,

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

- For the bubbled AND gate the equation is,

$$Z = \bar{A} \cdot \bar{B}$$

- As the NOR and bubbled AND gates are interchangeable, i.e., both gates have exactly identical outputs for the same set of inputs.
- Therefore, the equation can be written as shown below.

$$\overline{A + B} = \bar{A} \cdot \bar{B} \quad \ldots (1)$$

- This equation (1) or identity shown above is known as DeMorgan's Theorem.
- The symbolic representation of the theorem is shown in the figure below.

NOR gate Bubbled AND gate

Fig. 2.5 (a)

- It states that the complement of sum is equal to the product of the complements.

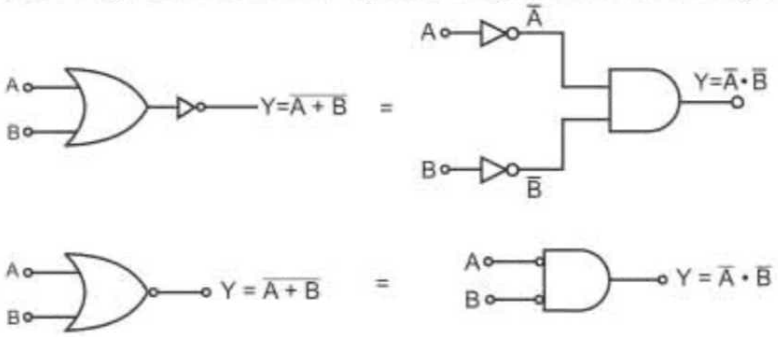


Fig. 2.5 (b)

Proof of De Morgan's First Theorem:

A	B	$\bar{A}$	$\bar{B}$	A + B	$\overline{A+B}$ (LHS)	$\bar{A}\cdot\bar{B}$ (RHS)
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

Hence, L.H.S. = R.H.S.

De Morgan's Second Theorem:

- De Morgan's Second Theorem states that the NAND gate is equivalent to a bubbled OR gate.
- The Boolean expression for the NAND gate is given by the equation shown below.

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

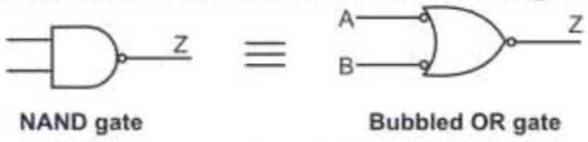
- The Boolean expression for the bubbled OR gate is given by the equation shown below.

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

- Since NAND and bubbled OR gates are interchangeable, i.e., both gates have identical outputs for the same set of inputs.
- Therefore, the equations become as given below.

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

- This identity or equation (2) shown above is known as De Morgan's Second Theorem. The symbolic representation of the theorem is shown in the figure below.



NAND gate Bubbled OR gate

Fig. 2.5 (c)

- It states that the complement of product is equal to the sum of the complements.

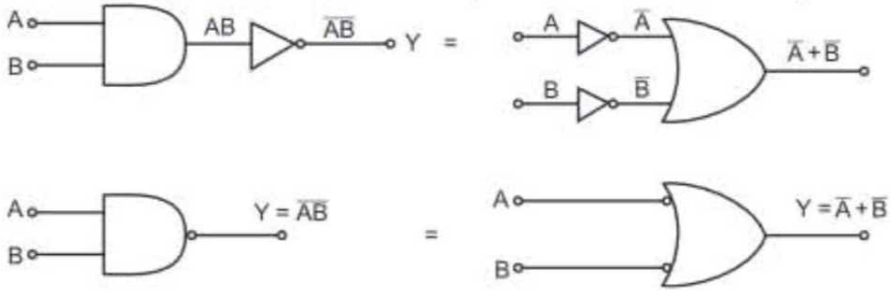


Fig. 2.5 (d)

Proof of De Morgan's Second Theorem:

A	B	$\bar{A}$	$\bar{B}$	A · B	$\overline{A \cdot B}$ (LHS)	$\bar{A} + \bar{B}$ (RHS)
0	0	1	1	0	1	1
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	0

Hence, L.H.S. = R.H.S.

Solved examples on DE-Morgans Theorem

Example 1: Implement the logic equation $\overline{A \cdot B \cdot C \cdot D}$ using NAND gates.

Solution: By De-Morgan's Second Theorem

i.e. $\overline{A \cdot B} = \bar{A} + \bar{B}$

We can write,

$$\overline{AB} + \overline{CD} = \overline{\overline{(AB)} \cdot \overline{(CD)}}$$

Hence the circuit is as shown in Fig. 2.6.

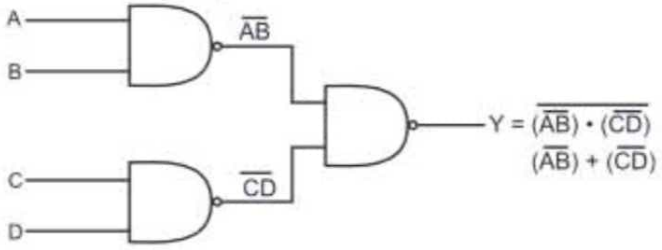


Fig. 2.6

Example 2: Use De-Morgan's theorem to simplify the following Boolean expression $\overline{\bar{A}B + A\bar{B}}$.

Solution:

$$\overline{\bar{A}B + A\bar{B}} = (\overline{\bar{A}B}) \cdot \overline{(A\bar{B})} \quad \text{(by De-Morgan's 1st theorem)}$$

$$= (\bar{\bar{A}} + \bar{B}) \cdot (\bar{A} + \bar{\bar{B}}) \quad \text{(by De-Morgan's 2nd theorem)}$$

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

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

$$= AB + \bar{A}\bar{B} \quad (\because A\bar{A} = B\bar{B} = 0)$$

Example 3: Prove the following use De-Morgan's theorem:

1. $AB + CD = \overline{\overline{AB} \cdot \overline{CD}}$
2. $\overline{(A + B) \cdot (C + D)} = (\bar{A} \cdot \bar{B}) \cdot (\bar{C} + \bar{D})$

Solution:

1. $AB + CD = \overline{\overline{AB + CD}}$

$= \overline{\overline{AB} \cdot \overline{CD}}$ ... De-Morgan's theorem

2. $\overline{(A + B) \cdot (C + D)} = \overline{(A + B)} + \overline{(C + D)}$ ... De-Morgan's theorem

$= (\bar{A} \cdot \bar{B}) + (\bar{C} \cdot \bar{D})$ ... De-Morgan's theorem

2.7 REDUCTION OF LOGIC EXPRESSION USING BOOLEAN ALGEBRA

- We can use these "Laws of Boolean" to both reduce and simplify a complex Boolean expression in an attempt to reduce the number of logic gates required.

Steps for reduction:

Step 1 : The first step for reducing a logic circuit is to write the Boolean Equation for the logic function.

Step 2 : The next step is to apply as many rules and laws as possible in order to decrease the number of terms and variables in the expression.

Step 3 : To apply the rules of Boolean Algebra it is often helpful to first remove any parentheses or brackets.

Step 4 : After removal of the parentheses, common terms or factors may be removed leaving terms that can be reduced by the rules of Boolean Algebra.

Step 5 : The final step is to draw the logic diagram for the reduced Boolean Expression.

EXAMPLES

Example 1: $A(A + \bar{A}) + B$

$= AA + A\bar{A} + B$		by the distributive law
$= A + 0 + B$		because $AA = A$ and $A\bar{A} = 0$
$= A + B$		because $A + 0 = A$

Example 2: $(A + B)(\bar{A} + B)\bar{B}$

$= (A + B)(\bar{A}\bar{B} + B\bar{B})$	by the distributive law
$= (A + B)(\bar{A}\bar{B} + 0)$	because $B\bar{B} = 0$
$= (A + B)(\bar{A}\bar{B})$	because $\bar{A}\bar{B} + 0 = \bar{A}\bar{B}$
$= A\bar{A}\bar{B} + B\bar{A}\bar{B}$	by the distributive law
$= \bar{B}0 + \bar{A}0$	because $A\bar{A} = 0$ and $B\bar{B} = 0$
$= 0$	because 0 ANDed with anything is 0

Example 3: Simplify $\overline{AB}(\bar{A} + B)(\bar{B} + B)$

Expression	Rule(s) Used
$\overline{AB}(\bar{A} + B)(\bar{B} + B)$	Original Expression
$\overline{AB}(\bar{A} + B)$	Complement law, Identity law.

contd. ...

$(\bar{A} + \bar{B})(\bar{A} + B)$	DeMorgan's Law
$\bar{A} + \bar{B}B$	Distributive law. This step uses the fact that or distributes over and. It can look a bit strange since addition does not distribute over multiplication.
$\bar{A}$	Complement, Identity.

Example 4: Simplify $(A + C)(AD + A\bar{D}) + AC + C$

Expression	Rule(s) Used
$(A + C)(AD + A\bar{D}) + AC + C$	Original Expression
$(A + C)A(D + \bar{D}) + AC + C$	Distributive Law.
$(A + C)A + AC + C$	Complement, Identity.
$A((A + C) + C) + C$	Commutative, Distributive.
$A(A + C) + C$	Associative, Idempotent.
$AA + AC + C$	Distributive.
$A + (A + 1)C$	Idempotent, Identity, Distributive.
$A + C$	Identity, twice.

2.8 DERIVING BOOLEAN EXPRESSION FROM GIVEN CIRCUIT

1. To convert a Logic gate circuit to a Boolean expression, label each gate output with a Boolean sub-expression corresponding to the gates' input signals, until a final expression is reached at the last gate.
2. Remember that OR gates are equivalent to Boolean addition, while AND gates are equivalent to Boolean multiplication.

- To derive the Boolean expression for a given logic circuit,
  1. Start by splitting up given diagram into parts.
  2. Begin from the leftmost input and worked out what the logic expression was after each logic gate.
  3. Continue doing this from left to right until we got the final logic gate.

Example 1:

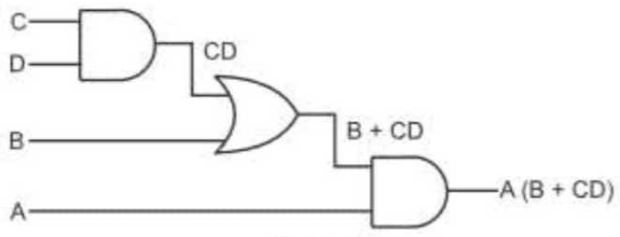


Fig. 2.7

- For the example circuit in Figure, the Boolean expression is determined as follows: The expression for the left-most AND gate with inputs C and D is CD.
- The output of the left-most AND gate is one of the inputs to the OR gate and B is the other input. Therefore, the expression for the OR gate is B + CD.
- The output of the OR gate is one of the inputs to the right-most AND gate and A is the other input. Therefore, the expression for this AND gate is A (B + CD), which is the final output expression for the entire circuit.

Example 2:

Consider the following logic gate diagram.

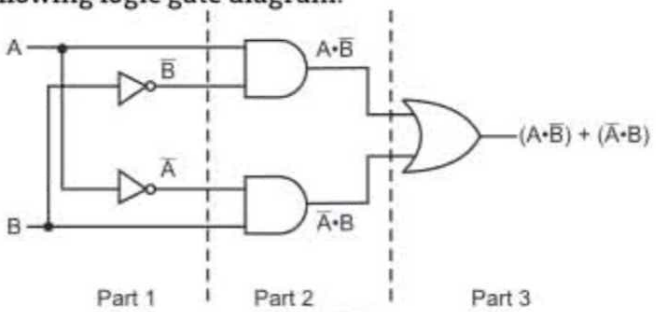


Fig. 2.8

Example 3:

Convert the following logic gate circuit into a Boolean expression, writing Boolean sub-expressions next to each gate output in the diagram:

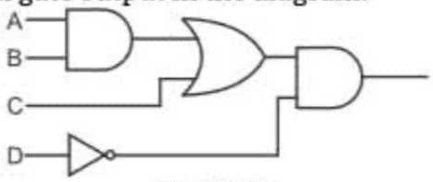


Fig. 2.9 (a)

Solution:

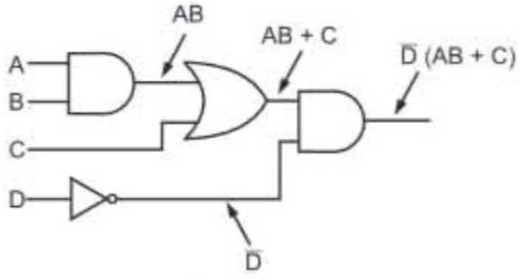


Fig. 2.9 (b)

Example 4:

Convert the following logic gate circuit into a Boolean expression, writing Boolean sub-expressions next to each gate output in the diagram:

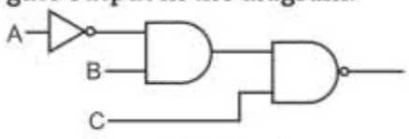


Fig. 2.10 (a)

Solution:

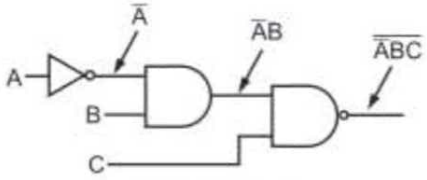


Fig. 2.10 (b)

2.9 EXCLUSIVE OR GATE

- Exclusive - OR (EX-OR) gate is neither a basic gate nor an universal gate. It is a derived gate, but it is a special case of OR gate.
- It can be simulated by using the basic gates (AND, OR and NOT) or universal gates (NAND or NOR). The X-OR gate is also called EX-OR gate.
- If the inputs are having even number of logic 1 states or logic 0 states, then the output will be only in logic 0 state.
- The Exclusive - OR gate is an electronic circuit having two or more number of inputs and only one output which recognizes only inputs that have an odd number of logic 1 state.
- An encircled plus sign (⊕) is used to show the EX-OR operation.
- Fig. 2.11 shows logic symbol of EX-OR gate.

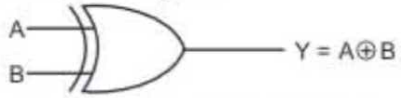


Fig. 2.11: Symbol of X-OR Gate

- The logic equation for EX-OR gate is $Y = A \oplus B$ or $Y = \overline{A}B + A\overline{B}$ or A EX-OR B
- Following table shows truth table for X-OR gate.

Inputs		Output
A	**B**	**$Y = A \oplus B$ OR $Y = \overline{A}B + A\overline{B}$**
0	0	0
0	1	1
1	0	1
1	1	0

2.10 EXCLUSIVE - NOR GATE [S-17]

- The Exclusive - NOR gate is a combination of an EX-OR gate and a NOT gate. The EX - NOR gate is also called as an EX-NOR gate or equality gate.
- The EX-NOR gate is an electronic circuit having two or more number of inputs and only one output and whose output assumes logic 1 state, if and only if the inputs are having an even number of logic 1 states.

- Fig. 2.12 shows logic symbol EX-NOR gate.
- The symbol is an EX-NOR gate with a small circle on the output. The small circle represents inversion.

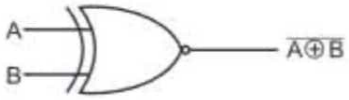


Fig. 2.12: Symbol of EX-NOR Gate

- The logic equation for EX-NOR gate is $Y = \overline{A \oplus B}$ OR $Y = \overline{A}\overline{B} \oplus A\overline{B}$.
- Following table shows truth table for EX-NOR Gate.

Inputs		Output
A	**B**	**$Y = \overline{A \oplus B}$ OR $Y = \overline{A}\overline{B} \oplus A\overline{B}$**
0	0	1
0	1	0
1	0	0
1	1	1

- The 'Exclusive-NOR' gate circuit does the opposite to the EX-NOR gate. It will give a low output if either, but not both, of its two inputs are high.

2.11 UNIVERSAL LOGIC GATES [S-18]

- Universal gates are the logic gates which are capable of implementing any Boolean function without requiring any other type of gate.
- Because universal gates can realize all the binary operations and all the basic logic gates can be derived from them, that is why they are called as "**Universal Gates**".
- The NAND and NOR gates are known as **Universal Gates** because any logic function and any basic gate can be implemented using NAND and NOR gates.

2.11.1 NAND Gate

- When a NOT gate is combined with the AND gate in cascade, the resultant gate is known as NAND gate.
- The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

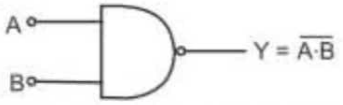


Fig. 2.13: Symbol of NAND Gate

- The logic equation of NAND is $Y = \overline{A \cdot B}$ and is read as "Y equals NOT(A and B)".

- Following table shows truth table of NAND gate.

Inputs		Output
A	B	$Y = \overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

- The outputs of all NAND gates are high if any of the inputs are low.

2.11.2 NOR Gate

- When a NOT gate is combined with the OR gate in cascade, the resultant gate is known as NOR gate.
- The NOR gate is an electronic circuit in which the output is in logic 1 state, when both inputs are in logic 0 state, and the output is in logic 0 state when both or any one of the inputs are in logic 1 state.
- This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if any of the inputs are high.
- The symbol is an OR gate with a small circle on the output. The small circle represents inversion.

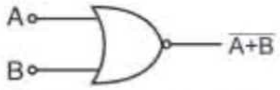


Fig. 2.14: Symbol of NOR Gate

- The logic equation for NOR gate is $Y = \overline{A + B}$ and is read as "Y equal NOT (A OR B)".
- Following table shows truth table for NOR gate.

Inputs		Output
A	B	$Y = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

Properties of Universal Gates:

- Universal gates are not associative in nature.
- Universal gates are commutative in nature.

2.12 IMPLEMENTATION OF OTHER GATES USING UNIVERSAL GATES [S-17, 18]

1. NOT Gate using NAND Gate:

- A NOT gate can be constructed using only NAND gate by tying all its inputs together.
- The expression for NOT gate is $Y = \overline{A}$.

Expression for NAND gate is $Y = \overline{A \cdot B}$
If we tie both the inputs together, A = B

$\therefore$ $Y = \overline{A \cdot A} = \overline{A}$ = NOT gate

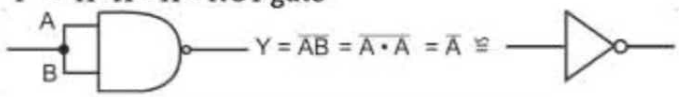


Fig. 2.15: NOT Gate using NAND Gate

2. AND Gate using NAND Gate:

- An AND gate is generated by simply inverting the output of NAND gate.
- The expression for AND gate is $Y = A \cdot B = \overline{\overline{AB}}$ ($\because \overline{\overline{A}} = A$)

Expression for NAND gate is $Y = \overline{A \cdot B}$.

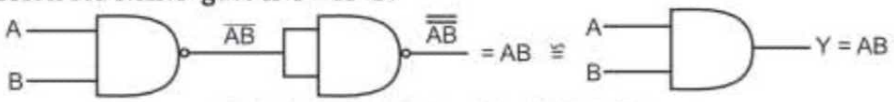


Fig. 2.16: AND Gate using NAND Gate

3. OR Gate using NAND Gates:

- The Boolean expression for OR gate is $Y = A + B$.
- Taking double inversion $\overline{\overline{A + B}}$ ($\because \overline{\overline{A}} = A$)

Applying De Morgan's first theorem ($\because \overline{A + B} = \overline{A} \cdot \overline{B}$)

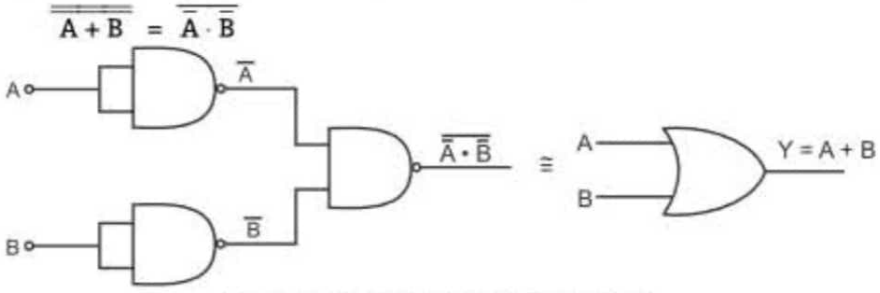


Fig. 2.17: OR Gate using NAND Gates

4. NOR Gate using NAND Gates:

- The Boolean expression for NOR gate is $Y = \overline{A + B}$

Applying De Morgan's first theorem, $Y = \overline{A} \cdot \overline{B}$

$\therefore$ $Y = \overline{\overline{A} \cdot \overline{B}}$ ($\because \overline{\overline{A}} = A$)

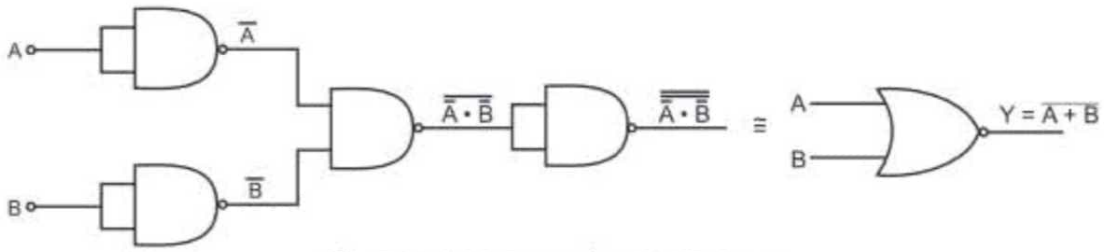


Fig. 2.18: NOR Gate using NAND Gates

5. EX-OR Gate using NAND Gates:

- The Boolean expression for EX-OR gate is $Y = \overline{A}B + A\overline{B}$.

  i.e. $Y = \overline{\overline{A}B \cdot A\overline{B}}$

  Using De Morgan's first theorem,

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

- There are product terms with a complement, so now we can implement using only NAND gates at this stage.

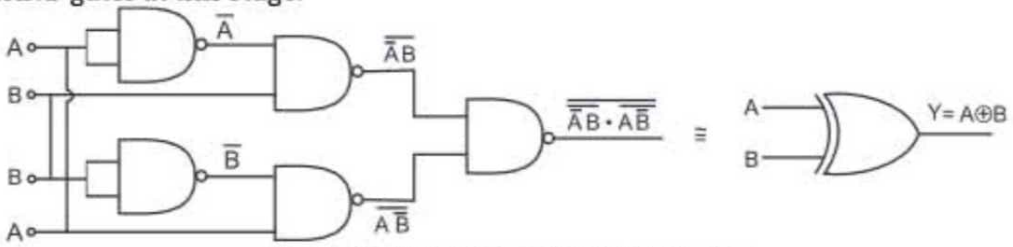


Fig. 2.19: EX-OR Gate using NAND Gates

6. EX-NOR Gate using NAND Gates:

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

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

Applying De Morgan's first theorem

$$Y = \overline{\overline{A}B} \cdot \overline{A\overline{B}} \quad (\because \overline{\overline{A}} = A)$$

Product terms with a complement, so we can implement using only NAND gates at this stage.

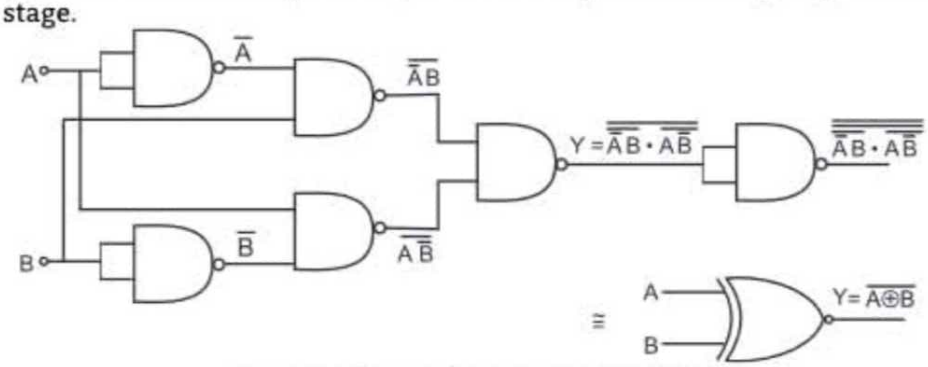


Fig. 2.20: EX-NOR Gate using NAND Gates

7. NOT Gate using NOR Gate:

Expression for NOR gate is $Y = \overline{A + B}$

Expression for NOT gate is $Y = \overline{A}$

If we short, both the inputs of NOR gate, then A = B.

$\therefore$ $Y = \overline{A + A} = \overline{A}$ ($\because A + A = A$)

= Expression for NOT Gate

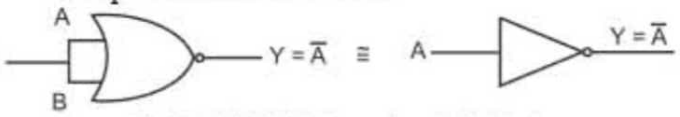


Fig. 2.21: NOT Gate using NOR Gate

8. AND Gate using NOR Gate:

- Expression for AND gate is $Y = AB = \overline{\overline{AB}}$ ($\because \overline{\overline{A}} = A$)

Applying De Morgan's second theorem, $Y = \overline{\overline{A} + \overline{B}}$, we can implement using NOR gates at this stage.

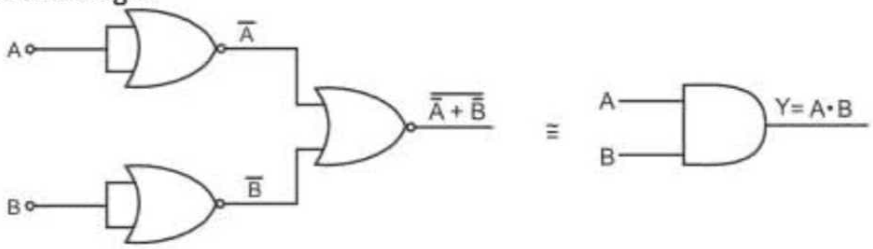


Fig. 2.22: AND Gate using NOR Gate

9. OR Gate using NOR Gate:

- Expression for OR gate is $Y = A + B = \overline{\overline{A + B}}$

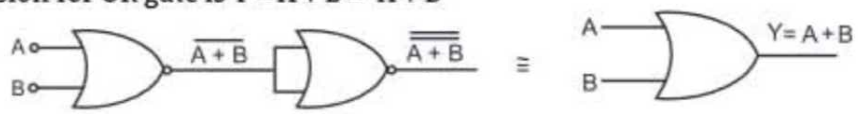


Fig. 2.23: OR Gate using NOR Gate

10. EX-OR Gate using NOR Gate:

- Expression for EX-OR Gate is $Y = \overline{A}B + A\overline{B}$

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

Applying De Morgan's theorem,

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

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

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

$$Y = \overline{(A + \overline{B}) \cdot (\overline{A} + B)} = \overline{\overline{A + \overline{B}} + \overline{\overline{A} + B}} \quad (\because \overline{\overline{A}} = A)$$

We can implement using only NOR gates at this stage.

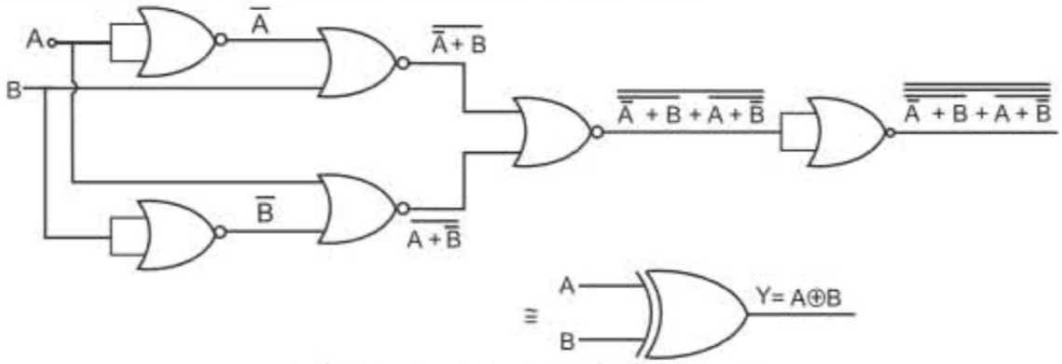


Fig. 2.24: EX-OR Gate using NOR Gate

11. EX-NOR Gate using NOR Gate:

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

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

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

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

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

We can implement using NOR gates at this stage.

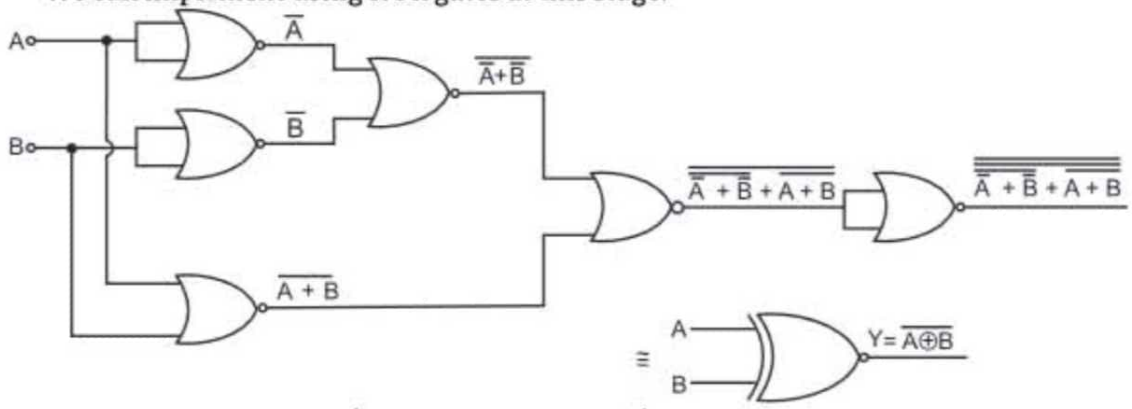


Fig. 2.25: EX-NOR Gate using NOR Gate

EXAMPLES [W-17]

Example 1: Implement the following Boolean expressions using NAND gates only:

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

Solution:

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

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

Applying De Morgan's theorem

$$Y = \overline{\bar{A} \cdot \overline{\bar{B}C} \cdot \overline{AC}}$$

This can be implemented using NAND gates.

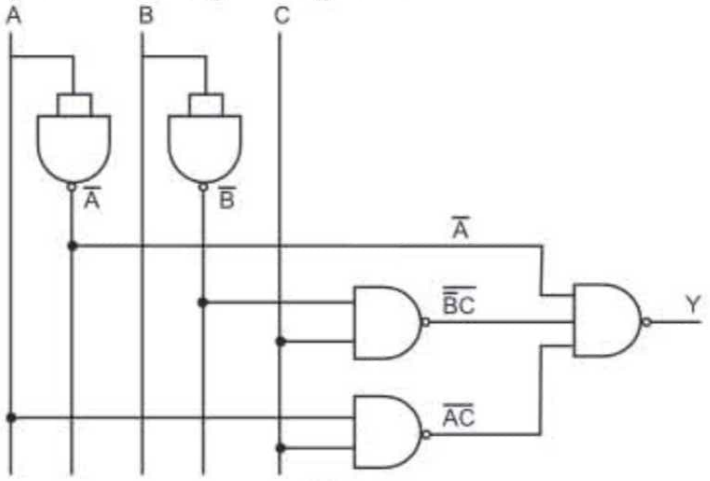


Fig. 2.26

Example 2: Realise the following Boolean expression using NAND gates only, $Y = A + B$.

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

$= \overline{\bar{A} \cdot \bar{B}}$ (Using De Morgan's theorem)

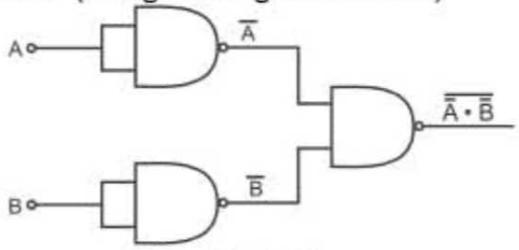


Fig. 2.27

Example 3: Implement the following Boolean expression using NOR gates only $Y = \bar{A}\bar{B}C + AC$.

Solution: $Y = \bar{A}\bar{B}C + AC$

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

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

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

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

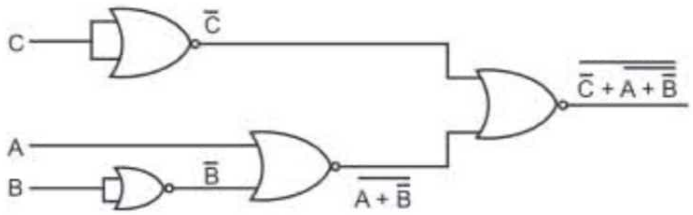


Fig. 2.28

Example 4: Implement the following Boolean equation. Use one AND gate and one OR gate only, Y = (A + B) (A + C).

Solution:

$$\begin{aligned} Y &= (A + B)(A + C) \\ &= A \cdot A + AC + AB + BC \\ &= A + AC + AB + BC \\ &= A(1 + C + B) + BC \quad (\because 1 + A + C = 1) \\ &= A + BC \end{aligned}$$

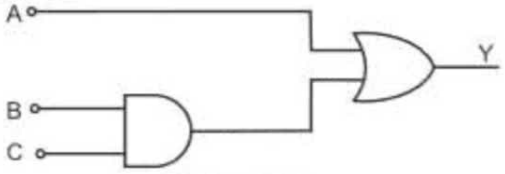


Fig. 2.29

2.13 MINTEM, MAXTERM AND KARNAUGH MAPS

2.13.1 Introduction

- A Boolean function is an algebraic form of Boolean expression. A Boolean function of n-variables is represented by $f(x_1, x_2, x_3 .... x_n)$.
- By using Boolean laws and theorems, we can simplify the Boolean functions of digital circuits.
- Any boolean expression may be expressed in terms of either minterms or maxterms. To do this we must first define the concept of a literal.
- A literal is a single variable within a term which may or may not be complemented
- Different ways of representing a Boolean function is shown below.
  - Sum-of-Products (SOP) Form
  - Product-of-Sums (POS) Form
  - Canonical Forms
- There are two types of Canonical Forms:
  1. Sum-of-min terms or Canonical SOP
  2. Product-of- max terms or Canonical POS
- Boolean functions can be represented by using NAND gates and also by using K-map (Karnaugh map) method.
- Standardization of Boolean equations will make the implementation, evolution and simplification easier and more systematic.

2.13.2 Sum of Products (SOP) Form

- The word sum and product are derived from the symbolic representations of the OR and AND functions by + and *(addition and multiplication) respectively.
- In this SOP form of Boolean function representation, the variables are operated by AND (product) to form a product term and all these product terms are ORed (summed or added) together to get the final function.
- **Example:**
  (a) Product term is any group of literals that are ANDed together example ABC, XY, PQR and so on.
  (b) A sum term is any group of literals that are ORed together such as A + B + C, X + Y, P + Q + R and so on.
  (c) A sum of product (SOP) is a group of product terms ORed tighter.
  (i) AB + ABC + CDE
  (ii) $(A\bar{B}) + ABC + CD\bar{E}$

2.13.3 Product of Sums (POS) Form

- The product of sums form is a method (or form) of simplifying the Boolean expressions of logic gates.
- In this POS form, all the variables are ORed, i.e. written as sums to form sum terms.
- All these sum terms are ANDed (multiplied) together to get the product-of-sum form. This form is exactly opposite to the SOP form. So this can also be said as "Dual of SOP form".
- Here the sum terms are defined by using the OR operation and the product term is defined by using AND operation.
- When two or more sum terms are multiplied by a Boolean OR operation, the resultant output expression will be in the form of product-of-sums form or POS form.
- The product-of-sums form is also called as **Conjunctive Normal Form** as the sum terms are ANDed together and Conjunction operation is logical AND.
- **Examples:**
  - (A + B) * (A + B + C) * (C + D)
  - $(A + \bar{B}) * (C + D + \bar{E})$

2.13.4 Conversion from Non-standard SOP to standard SOP Form

- Convert the expression $Y = AB + A\bar{C} + BC$ into standard SOP from.

Step 1 : Find the missing literal for each term:

$$Y = AB + A\bar{C} + BC$$
$$\downarrow \quad \downarrow \quad \downarrow$$
$$C \quad B \quad A \quad \leftarrow \text{Missing literal}$$

Step 2 : AND each term with (Missing literal + Its complement):

↓ ANDing

$$Y = AB \cdot (C + \bar{C}) + A\bar{C}(B + \bar{B}) + BC(A + \bar{A})$$

Original product form — (Missing literal + its complement)

Step 3 : Simplify the expression to get the standard SOP:

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

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

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

But $A + A = A \quad \therefore (ABC + ABC) = ABC$ and $(AB\bar{C} + AB\bar{C}) = AB\bar{C}$

$$\therefore \quad Y = \underbrace{ABC + AB\bar{C} + A\bar{B}\bar{C} + \bar{A}BC}_{\text{Each term contains all the literals}} \rightarrow \textbf{Standard SOP form}$$

2.13.5 Conversion from Non- standard POS to standard POS Form

- Convert the expression $Y = (A + B)(A + C)(B + \bar{C})$ into standard POS form.

Step 1 : Find the missing literal for each term:

$$Y = \underbrace{(A + B)}_{C} \quad \underbrace{(A + C)}_{B} \quad \underbrace{(B + \bar{C})}_{A} \quad \leftarrow \text{Missing literal}$$

Step 2 : OR each term with (Missing literal. Its complement):

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

This term is ORed with the term formed by ANDing the missing literal with its complement

Missing literal ANDed with its complement

Step 3 : Simplify the expression to get the standard POS:

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

But $A + BC = (A + B)(A + C)$

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

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

But $A \cdot A = A \quad \therefore (A + B + C)(A + C + B) = (A + B + C)$

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

$$\therefore \quad Y = \underbrace{(A + B + C)(A + B + \bar{C})(A + \bar{B} + C)(\bar{A} + B + \bar{C})}_{\text{Each term contains all the literals}} \textbf{ Standard POS form}$$

2.13.6 Canonical Form (Standard SOP and POS Form)

- Any Boolean function that is expressed as a sum of minterms or as a product of max terms is said to be in its "Canonical Form".
- It mainly involves in two Boolean terms, "minterms" and "maxterms".

- When the SOP form of a Boolean expression is in canonical form, then each of its product term is called 'minterm'. So, the canonical form of sum of products function is also known as "minterm canonical form" or Sum-of-minterms or Standard Canonical SOP form.
- Similarly, when the POS form of a Boolean expression is in canonical form, then each of its sum term is called 'maxterm'. So, the canonical form of product of sums function is also known as "maxterm Canonical Form or Product-Of Sum or Standard Canonical POS Form".
- Each row of a truth table can be associated with a minterm and a maxterm.

2.13.7 Minterm

- A minterm is a product (AND) of all variables in the function, in direct or complemented form.
- A minterm has the property that it is equal to 1 on exactly one row of the truth table.
- Any Boolean function can be expressed as the sum (OR) of its 1- min terms. The representation of the equation will be

  F(list of variables) = Σ(list of 1-min term indices)

  Example: F (x, y, z) = Σ (3, 5, 6, 7)
- The inverse of the function can be expressed as a sum (OR) of its 0- min terms. The representation of the equation will be

  F(list of variables) = Σ(list of 0-min term indices)

  Example: F' (x, y, z) = Σ (0,1, 2, 4)
- Examples of canonical form of sum of products expressions (min term canonical form):

  (i) $Z = XY + XZ'$

  (ii) $F = XYZ' + X'YZ + X'YZ' + XY'Z + XYZ$

2.13.8 Maxterm

- A maxterm is a sum (OR) of all the variables in the function, in direct or complemented form.
- A maxterm has the property that it is equal to 0 on exactly one row of the truth table.
- A max term is defined as the product of n variables, within the range of $0 \leq i < 2^n$.
- The max term is denoted as Mi.
- In max term, each variable is complimented, if its value is assigned to 1, and each variable is un-complimented if its value is assigned to 0.
- For a 2-variable (x and y) Boolean function, the possible max terms are:

  $x + y$, $x + y'$, $x' + y$ and $x' + y'$.
- For a 3-variable (x, y and z) Boolean function, the possible maxterms are:

  $x + y + z$, $x + y + z'$, $x + y' + z$, $x + y' + z'$, $x' + y + z$, $x' + y + z'$, $x' + y' + z$ and $x' + y' + z'$.
- 1 – Max terms = max terms for which the function F = 1.
- 0 – max terms = max terms for which the function F = 0.

- Any Boolean function can be expressed the product (AND) of its 0 – max terms. The representation of the equation will be:

  F(list of variables) = π (list of 0-max term indices)

  Example: F (x, y, z) = π (0, 1, 2, 4)
- The inverse of the function can be expressed as a product (AND) of its 1 – max terms. The representation of the equation will be

  F(list of variables) = π (list of 1-max term indices)

  Example: F' (x, y, z) = π (3, 5, 6, 7)
- Examples of canonical form of product of sums expressions (max term canonical form):

  (i) $Z = (X + Y)(X + Y')$

  (ii) $F = (X' + Y + Z')(X' + Y + Z)(X' + Y' + Z')$

Truth table representing minterm and maxterm :

X	Y	Z	Minterms	Maxterms
			Product Terms	Sum Terms
0	0	0	$m_0 = \bar{X} \cdot \bar{Y} \cdot \bar{Z} = \min(\bar{X}, \bar{Y}, \bar{Z})$	$M_0 = X + Y + Z = \max(X, Y, Z)$
0	0	1	$m_1 = \bar{X} \cdot \bar{Y} \cdot Z = \min(\bar{X}, \bar{Y}, Z)$	$M_1 = X + Y + \bar{Z} = \max(X, Y, \bar{Z})$
0	1	0	$m_2 = \bar{X} \cdot Y \cdot \bar{Z} = \min(\bar{X}, Y, \bar{Z})$	$M_2 = X + \bar{Y} + Z = \max(X, \bar{Y}, Z)$
0	1	1	$m_3 = \bar{X} \cdot Y \cdot Z = \min(\bar{X}, Y, Z)$	$M_3 = X + \bar{Y} + \bar{Z} = \max(X, \bar{Y}, \bar{Z})$
1	0	0	$m_4 = X \cdot \bar{Y} \cdot \bar{Z} = \min(X, \bar{Y}, \bar{Z})$	$M_4 = \bar{X} + Y + Z = \max(\bar{X}, Y, Z)$
1	0	1	$m_5 = X \cdot \bar{Y} \cdot Z = \min(X, \bar{Y}, Z)$	$M_5 = \bar{X} + Y + \bar{Z} = \max(\bar{X}, Y, \bar{Z})$
1	1	0	$m_6 = X \cdot Y \cdot \bar{Z} = \min(X, Y, \bar{Z})$	$M_6 = \bar{X} + \bar{Y} + Z = \max(\bar{X}, \bar{Y}, Z)$
1	1	1	$m_7 = X \cdot Y \cdot Z = \min(X, Y, Z)$	$M_7 = \bar{X} + \bar{Y} + \bar{Z} = \max(\bar{X}, \bar{Y}, \bar{Z})$

- From the above table it is clear that minterm is expressed in product format and maxterm is expressed in sum format.

2.13.9 Representing Logical Expression using Minterms and Maxterms

- As shown in above table, each minterm is represented by m_i and each maxterm is represented by M_i, where subscript i is the decimal number equivalent of the natural binary number. With these shorthand notations logical function can be represented as follows:

1. $$f(A, B, C) = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + AB\bar{C}$$
$$= m_0 + m_1 + m_3 + m_6 = \sum m(0, 1, 3, 6)$$

2. $f(A, B, C) = (A + B + \bar{C})(A + \bar{B} + \bar{C})(\bar{A} + \bar{B} + C)$
$= M_1 + M_3 + M_6 = \pi M(1, 3, 6)$

where Σ denotes *sum of product* while π denotes *product of sum.*

3. $Y = ABC + \underbrace{\bar{A}BC}_{m_3} + \underbrace{A\bar{B}\bar{C}}_{m_4}$ ← Given logic expression

m_7 ← Corresponding minterms

$\therefore \quad Y = m_7 + m_3 + m_4$

$\therefore \quad Y = \Sigma m(3, 4, 7)$ ← Other way of representation

where Σ denotes sum of products

4. $Y = \underbrace{(A + \bar{B} + C)}_{M_2}\ \underbrace{(A + B + C)}_{M_0}\ \underbrace{(\bar{A} + \bar{B} + C)}_{M_6}$ ← Given expression

← Corresponding maxterms

$\therefore \quad Y = M_2\ M_0\ M_6$

$\therefore \quad Y = \pi M(0, 2, 6)$ ← other way of representgation

where π denotes product of sums.

Problem on Minterm and Maxterms:

Problem: For the truth table of two variables write the minterms and maxterms.

Solution:

Variables / Literals		Minterms	Maxterms
A	**B**	m_i	M_i
0	0	$m_0 = \bar{A}\bar{B}$	$M_0 = A + B$
0	1	$m_1 = \bar{A}B$	$M_1 = A + \bar{B}$
1	0	$m_2 = A\bar{B}$	$M_2 = \bar{A} + B$
1	1	$m_3 = A\,B$	$M_3 = \bar{A} + \bar{B}$

2.13.10 To Write Standard SOP Expression for a Given Truth Table

Procedure is,

Step 1 : Consider only those combinations of inputs which correspond to Y = 1.

Step 2 : Write down a product term for each such combination.

Step 3 : OR all these product terms to get the standard SOP.

Example: From the given truth table obtain the logical expression in the standard SOP form.

Given truth table

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

Solution:

Step 1 : Consider only those combinations of inputs which correspond to Y = 1.

Step 2 and 3 : For the second and the third entries in given table. Write the product terms.

Truth Table

	A	B	Y
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

$Y_1 = \bar{A}B$

$Y_2 = A\bar{B}$

Boolean expressions in the product forms

OR (Add) all the product terms to write the final expression in standard SOP form as follows:

$\therefore \quad Y = Y_1 + Y_2 = \bar{A}B + A\bar{B}$

$\therefore \quad Y = m_1 + m_2 = \sum m\,(1, 2)$

2.13.11 To Write a Standard POS Expression for a given Truth Table

Procedure is,

Step 1 : Consider only those combinations of inputs which produce a law output (y = 0).

Step 2 : Write the maxterms only for such combinations.

Step 3 : AND those maxtgerms to obtain the expression in standard POS for.

Example: Write the logic expression in standard POS form for the truth table shown in Table.

Given truth table

A	B	C	Y	
0	**0**	**0**	**0**	$A + B + C(M_0)$
0	0	1	1	
1	1	0	1	
0	**1**	**1**	**0**	$A + \bar{B} + \bar{C}\,(M_3)$
1	0	0	1	
1	**0**	**1**	**0**	$\bar{A} + B + \bar{C}\,(M_5)$
1	**1**	**0**	**0**	$\bar{A} + \bar{B} + C\,(M_6)$
1	1	1	1	

Solution:

Step 1 : Write maxterms for the combinations of input which produce Y = 0.

Step 2 : AND (take product of) all the maxterms to get standard POS form:

- ANDing (taking product of) all the maxterms written in step 1 we get,

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

$$\therefore \quad Y = M_0 \cdot M_3 \cdot M_5 \cdot M_6 = \pi(0, 3, 5, 6)$$

2.13.12 Conversions of Canonical Forms

- We can represent the one canonical formed equation in other canonical form i.e. we can represent the SOP form of equation in POS form and POS form equation in SOP form.
- The important thing to remember about Boolean functions is that, the SOP and POS forms are Duals to each other. There are 2 steps to follow to convert the canonical form of the equations. These are:

Step 1 : Interchanging the operational symbols, $\sum$ and π in the equation.

Step 2 : Use the De Morgan's principle of Duality to the index numbers of the Boolean function or writing the indexes of the terms that are not presented in the given form of equation.

(a) Conversion of SOP form to POS form:

- To convert the SOP form into POS form, first we should change the Σ to π and then write the numeric indexes of missing variables of the given Boolean function.

Example: The SOP function

$f = \sum A, B, C (0, 2, 3, 5, 7) = A' B' C' + A B' C' + A B' C + ABC' + ABC$ is written in POS form by

Step 1 : Changing the operational sign to π

Step 2 : Writing the missing indexes of the terms, 001, 100 and 110. Now write the sum form for these noted terms.

001 = (A + B + C)

100 = (A + B' + C')

110 = (A + B' + C')

Writing down the new equation in the form of POS form,

$f = \pi A, B, C (1, 4, 6) = (A + B + C) * (A + B' + C') * (A + B' + C')$

(b) Conversion of POS form to SOP form:

- To convert the POS form into SOP form, first we should change the π to Σ and then write the numeric indexes of missing variables of the given Boolean function.
- **Example:** The POS function $f = \pi A, B, C (2, 3, 5) = A B' C' + A B' C + ABC'$ is written in SOP form by

Step 1 : Changing the operational sign to Σ

Step 2 : Writing the missing indexes of the terms, 000, 001, 100, 110, and 111. Now write the product form for these noted terms.

000 = A * B * C 001 = A' * B' * C 100 = A * B' * C'

110 = A * B* C' 111 = A * B * C

Step 3 : Writing down the new equation in the form of SOP form,

$f = \Sigma A, B, C (0, 1, 4, 6, 7) = (A' * B' * C') + (A' * B' * C) + (A * B' * C') + (A * B * C') + (A * B * C)$

(c) Conversion from SOP to POS and Vice-Versa:

- A product of sums form derived from a truth table is logically equivalent to a sum of products form derived from the following truth table. Let us write the standard SOP and POS.

Truth Table

Minterms	A	B	C	Y	Maxterms
	0	0	0	0	$A + B + C$
$\bar{A}\bar{B}C$	**0**	**0**	**1**	**1**	
	0	1	0	0	$A + \bar{B} + C$
	0	1	1	0	$A + \bar{B} + \bar{C}$
$A\bar{B}\bar{C}$	**1**	**0**	**0**	**1**	
	1	0	1	0	$\bar{A} + B + \bar{C}$
	1	1	0	0	$\bar{A} + \bar{B} + C$
ABC	**1**	**1**	**1**	**1**	

SOP form : $f(A, B, C) = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + AB\bar{C} + ABC$

POS form : $f(A, B, C) = (A + \bar{B} + C)(\bar{A} + B + \bar{C})$

- Now by simplifying equation in the POS form we have

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

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

- Converting to standard sum of product we have,

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

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

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

- Rearranging terms we have,

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

- Therefore, we can say that POS and SOP derived from the same truth table are logically equivalent. In terms of minterms and maxterms we can then write,

$$f(A, B, C) = m_0 + m_1 + m_3 + m_4 + m_6 + m_7 = M_2 + M_5$$

$$\therefore \quad f(A, B, C) = \sum m(0, 1, 3, 4, 6, 7)$$

$$= \pi M(2, 5)$$

- From the above expressions we can easily notice that there is a complementary type of relationship between a function expressed in terms of maxterms. Using this complementary relationship we can find logical function in terms of maxterms if function in minterms is known or vice-versa. For example, for a four variables if

$$f(A, B, C, D) = \sum m(0, 2, 4, 6, 8, 10, 12, 14)$$

then $$f(A, B, C, D) = \pi M(1, 3, 5, 7, 9, 11, 13, 15)$$

2.14 KARNAUGH MAP

- We have simplified the Boolean functions using Boolean postulates and theorems. It is a time consuming process and we have 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 known as Karnaugh map method or K-map method. It is a graphical method, which consists of 2^n cells for 'n' variables. The adjacent cells are differed only in single bit position.

2.14.1 One-Variable, Two-Variable, Three-Variable and Four-Variable Maps

- The basis of this method is a graphical chart known as Karnaugh map (K-map). It contains boxes called cells. Each of the cell represents one of the 2^n possible products that can be formed from n variables. Thus, a 2-variable map contains $2^2 = 4$ cells, a 3-variable map contains $2^3 = 8$ cells, and so forth. Fig. 2.30 shows outlines of 1, 2, 3 and 4 variable maps.

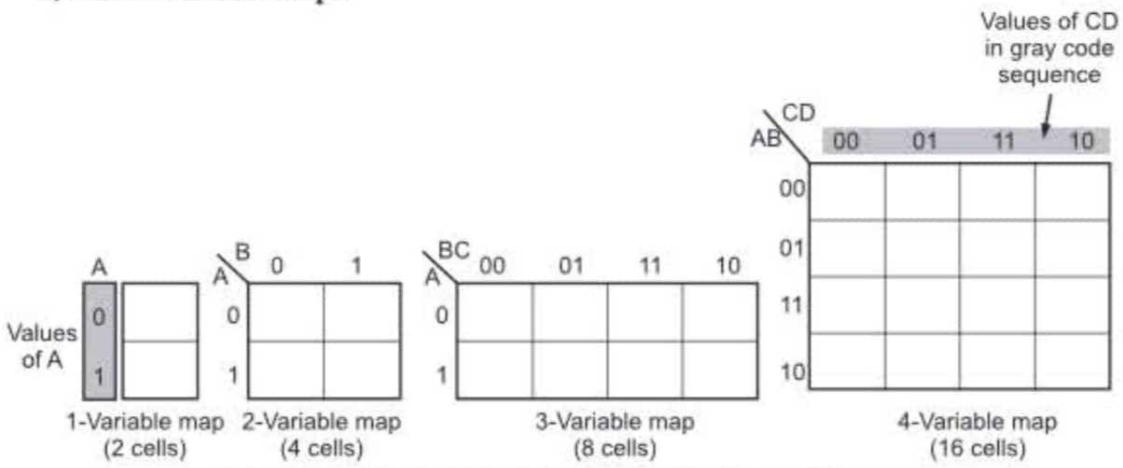
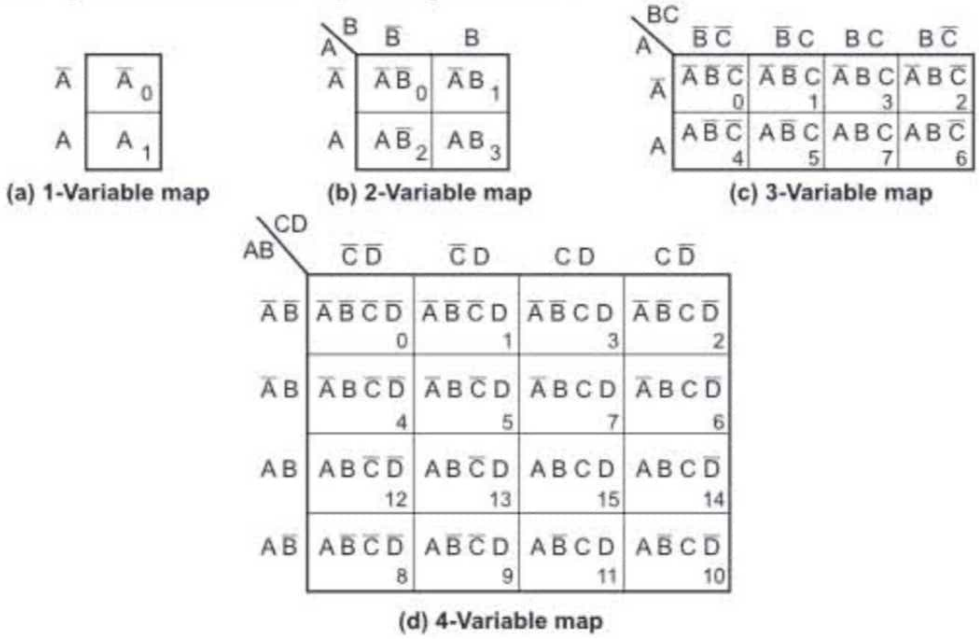


Fig. 2.30: Outlines of 1, 2, 3 and 4 variable Karnaugh maps

- Product terms are assigned to the cells of a Karnaugh map by labeling each row and each column of the map with a variable, with its complement, or with a combination of variables and complements. The product terms corresponding to a given cell is then the product of all variable in the row and column where the cell is located. Fig. 2.31 shows the way to label the rows and columns of a 1, 2, 3 and 4 variable maps and the product terms corresponding to each cell.



(a) 1-Variable map (b) 2-Variable map (c) 3-Variable map

(d) 4-Variable map

Fig. 2.31: 1, 2, 3 and 4 variable maps with product terms

- It is important to note that when we move from one cell to the next along any row or from one cell to the next along any column, one and only one variable in the product term changes (to a complemented or to an uncomplemented form).
- For example, in Fig. 2.31 (b) the only change that occurs in moving along the bottom row from $A\bar{B}$ to AB is the change from $\bar{B}$ to B. Similarly, the only change that occurs in moving down the right column from $\bar{A}B$ to AB is the change from $\bar{A}$ to A. Irrespective of number of variables the labels along each row and column must conform to the single-change rule.
- We know that the Gray code has same properties (only one variable change when we proceed to next number or previous number) hence gray code is used to label the rows and columns of K-map as shown in Fig. 2.32.
- The Fig. 2.32 shows label of the rows and columns of 1, 2, 3 and 4 variable maps using gray code and the product terms corresponding to each cell.
- Here, instead of writing actual product terms, corresponding shorthand minterm notations are written in the cell. Row and columns are marked with Gray code instead of variables.

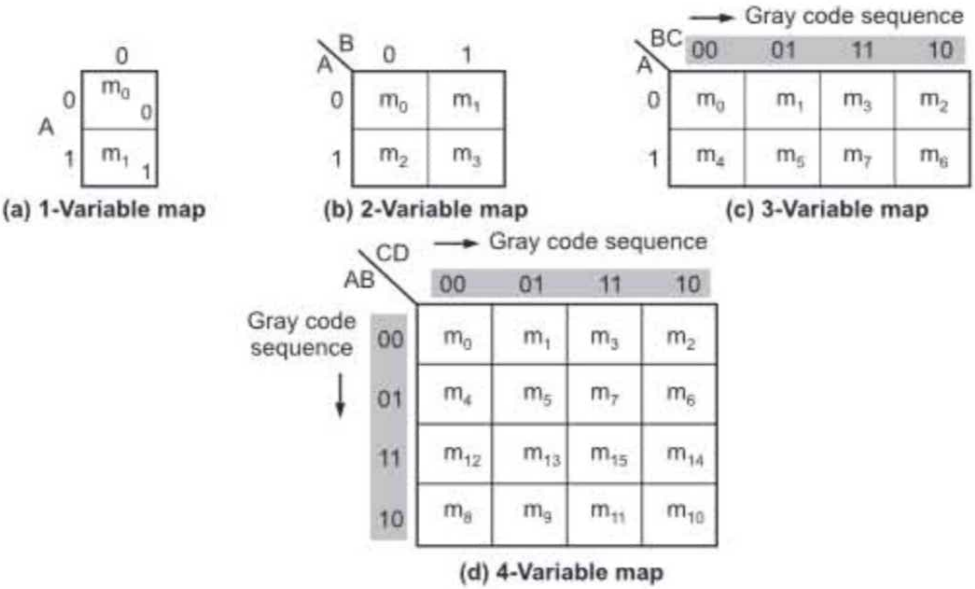


Fig. 2.32: Another way to represent 1, 2, 3 and 4 variable maps for SOP expressions

- In case of POS expressions we assign maxterms (sum terms) to the cells of a Karnaugh map. The Fig. 2.33 shows the way to label the rows and columns of 1, 2, 3 and 4 variable maps and the sum terms corresponding to each cell. The Fig. 2.32 rows and columns of 1, 2, 3 and 4 variable maps using Gray code and the sum terms corresponding to each cell. Here, instead of writing actual sum terms, corresponding shorthand maxterm notations are written in the cell, and row and columns are marked with gray code instead of variables.

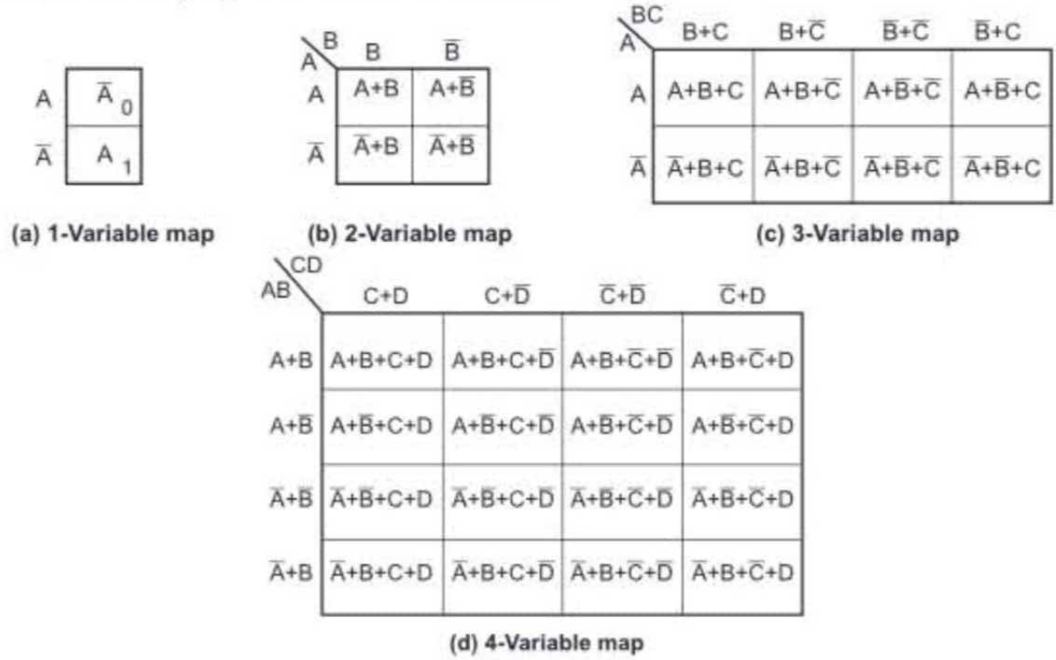


Fig. 2.33: 1, 2, 3 and 4 variable maps with sum terms

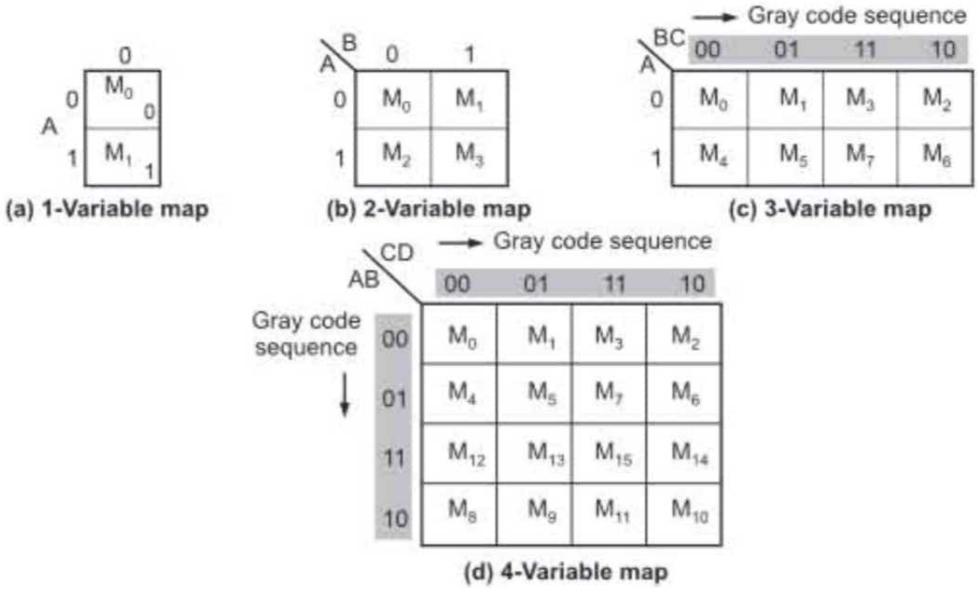


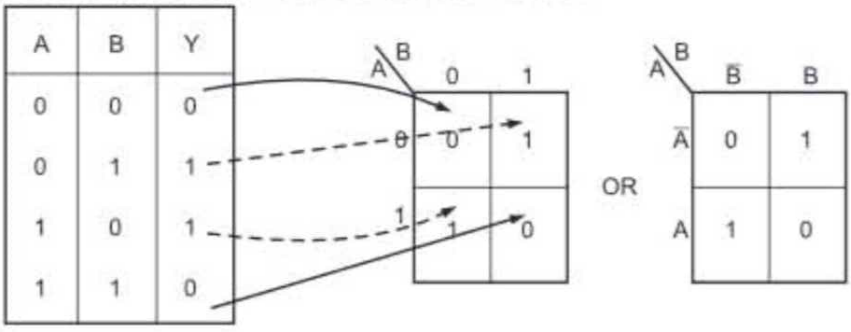
Fig. 2.34: 1, 2, 3 and 4 variable maps using Gray Code

2.14.2 Plotting a Karnaugh Map

- We know that logic function can be represented in various forms such as truth table, SOP Boolean expression and POS Boolean expression. Let us see the procedures to plot the given logic function in any form on the Karnaugh map.

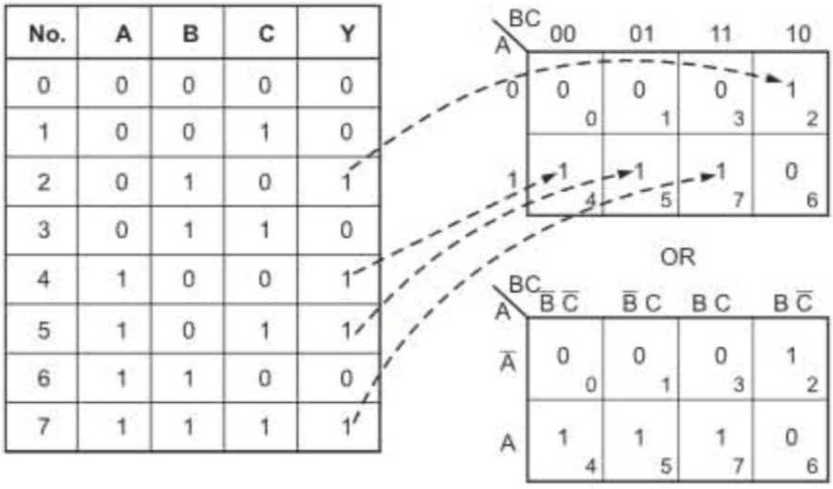
Representation of Truth Tabel on Karnaugh Map

- **Cell:** The smallest unit of a Karnaugh map, corresponding to one time of a truth table. The input variables are the cell's coordinates, and the output variable is the cell's contents.
- Fig. 2.35 shows K-maps plotted from truth tables with 2, 3 and 4 variables. We can easily notice that the terms which are having output 1, have the corresponding cells marked with 1s. The other cells are marked with zeros.



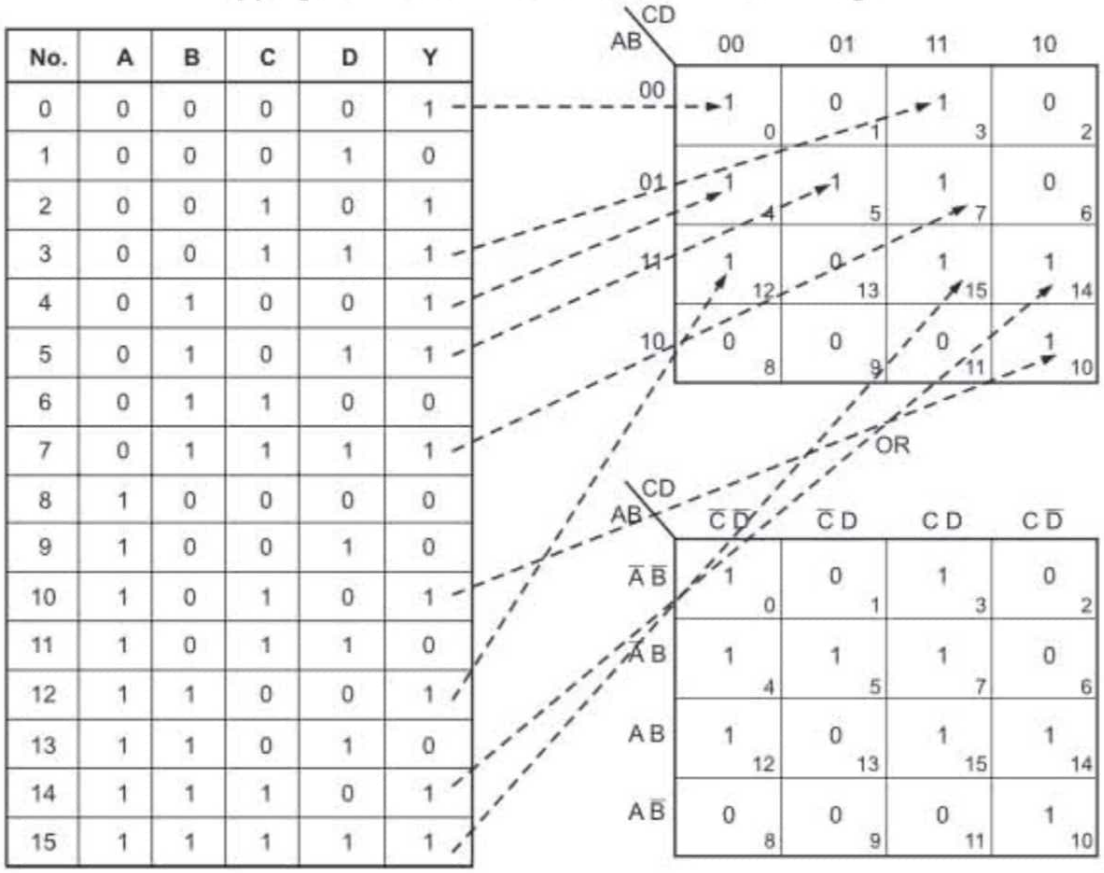
(a) Representation of 2-variable truth table on K-map

No.	A	B	C	Y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1



(b) Representation of 3-variable truth table on K-map

No.	A	B	C	D	Y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1



(c) Representation of 4-variable truth table of K-map

Fig. 2.35

Representation of Standard SOP form on K-map:

- The logical expression in standard SOP form can be represented on a K-map by simply entering 1's in the cells (boxes) of the K-map corresponding to each minterm present in the equation.
- The remaining cells (boxes) are filled with zeros.

Example 1: Illustrates the concept of transferring a canonical SOP expression on K-map.

Represent the equation given below on Karnaugh map.

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

Solution: The given expression is in the standard SOP form. Each term represents a minterm.

We have to enter 1's in the boxes corresponding to each minterm, as shown in Fig. 2.36.

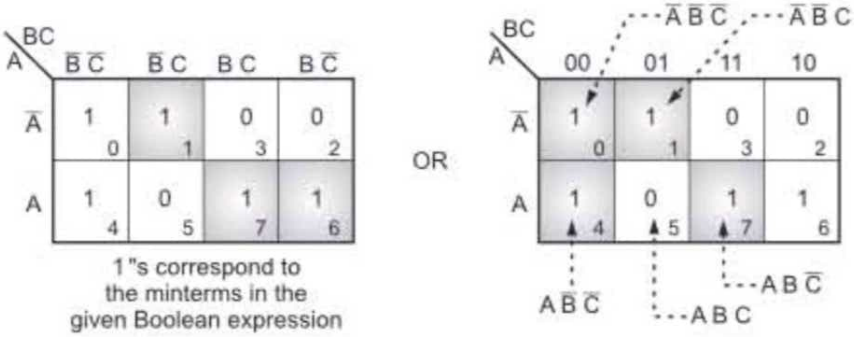


Fig. 2.36: Representation of standard SOP on K-map

Example 2: Plot the following Boolean expression on K-map.

$$Y = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}BC\bar{D} + AB\bar{C}\bar{D}$$

Solution:

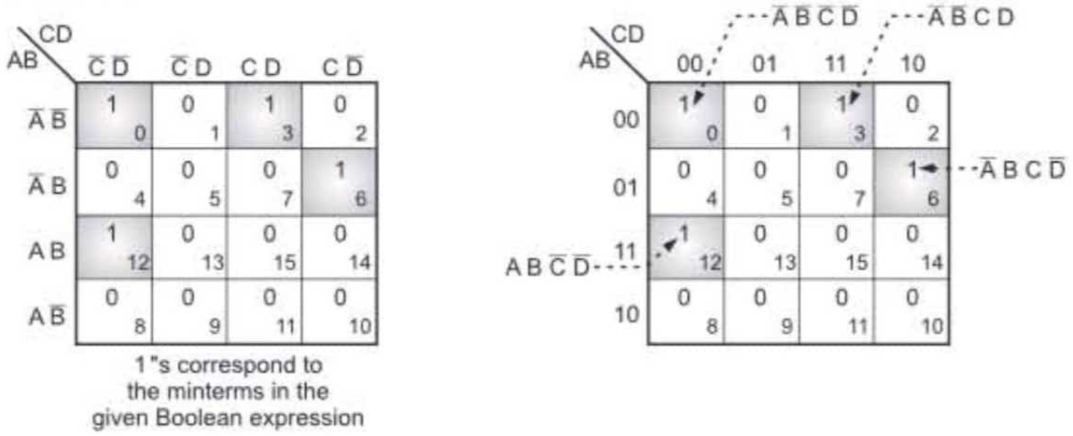


Fig. 2.37: Representation of Canonical SOP on Karnaugh map

2.14.3 Grouping of Variables in K-Maps

How does Simplification Takes Place?

- Once we plot the logic function or truth table on a Karnaugh map, we have to use the grouping technique for simplifying the logic function.
- Grouping means combining the terms in the adjacent cells.
- The grouping of either adjacent 1's or adjacent 0's results in the simplification of Boolean expression.
- If we group the adjacent 1's then the result of simplification is in SOP (sum of products) form And if the adjacent 0's are grouped, then the result of simplification is in the POS (product of sums)form.
- In this section we are going to use the SOP form. So we will group 1's and not the 0's.

Way of Grouping (Pairs, Quads and Octets);

- While grouping, we should group most number of 1's.
- The grouping follows the binary rule i.e. we can group 1, 2, 4, 8, 16, 32 ... number of 1's.
- We cannot group 3, 5, 6, 7, ... number of 1's.
  1. **Pairs:** A group of two adjacent 1's is called as a pair.
  2. **Quad:** A group of four adjacent 1's is called as a quad.
  3. **Octet:** A group of eight adjacent 1's 0 is called as octet.

Grouping Two Adjacent One's (Pairs):

- If we group two adjacent 1's on a K-map to form a pair, then the resulting term will have one less literal (variable) than the original term. That means by pairing two adjacent 1's we can eliminate one variable.

EXAMPLES

Example 1: Given K-map.

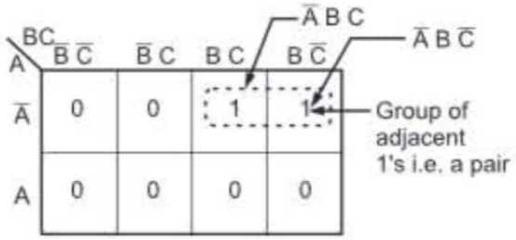


Fig. 2.38

Simplification:

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

$$= \bar{A}B \text{ ..... since } C + \bar{C} = 1$$

Thus C is eliminated

Conclusion:

- Due to formation of pair the variable C is eliminated. So henceforth just by looking at the pair we should be table to identify the variable that will be eliminated.
- The other types of pairs and corresponding simplifications are as shown in Fig. 2.38.

Example 2: Given K-map

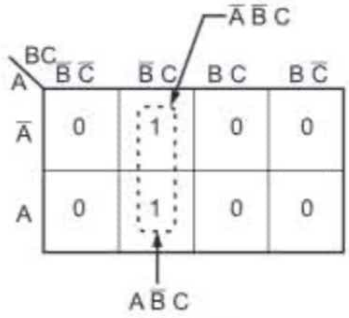


Fig. 2.39

Simplification:

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

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

$$Y = \bar{B}C \quad \text{... since } (\bar{A} + A) = 1$$

Conclusion: A is eliminated.

Example 3: Given K-map

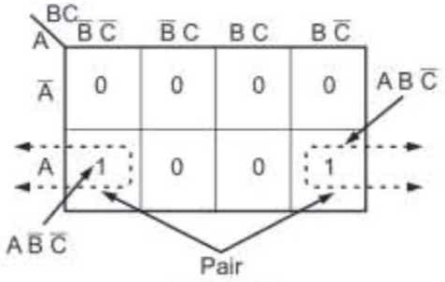


Fig. 2.40

Simplification:

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

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

$$Y = A\bar{C} \quad \text{... since } (B + \bar{B}) = 1$$

Conclusion: B is eliminated.

Example 4: Given K-map

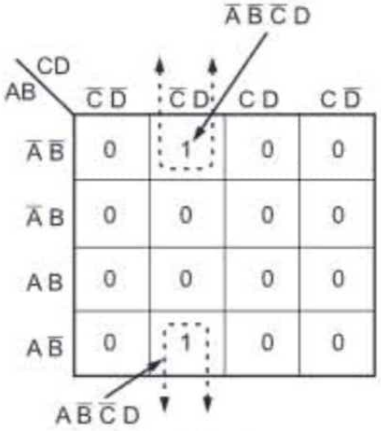


Fig. 2.41

Simplification:

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

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

$$\therefore \quad Y = \bar{B}\bar{C}D \quad \text{... since } (A + \bar{A}) = 1$$

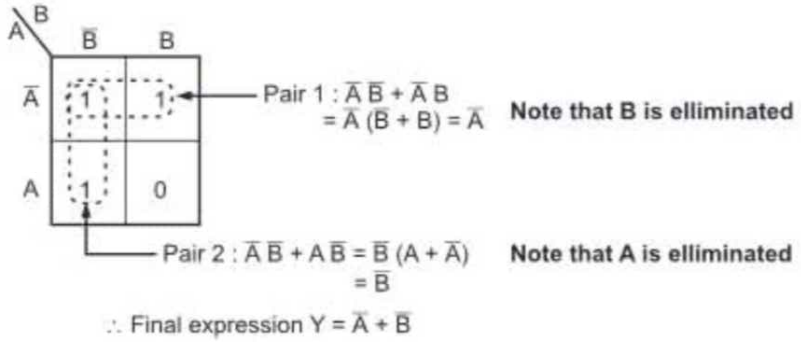
Conclusion: A is eliminated.

Different types of pairs and the corresponding simplifications:

Note: In this K-map three pairs were possible to be formed. However only two pairs are sufficient to include all the 1's present in the K-map. Then the third pair is not required.

Example 5: Overlap

Given K-map:



∴ Final expression Y = Pair 1 + Pair 2 = $\overline{A} + \overline{B}$

Fig. 2.42

Note that in order to cover all the 1's, we have to overlap two pairs as shown in Fig. 2.42.

Example 6: Given K-map

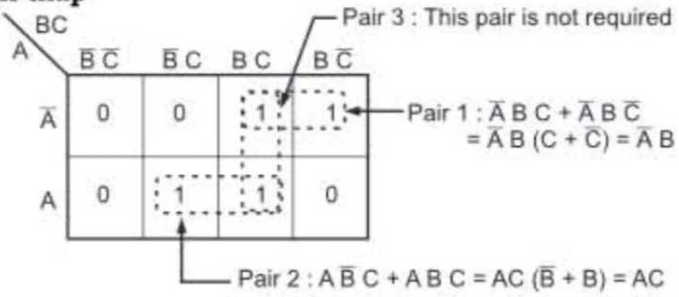


Fig. 2.43

Add the two product terms to get,

Y = Pair 1 + Pair 2

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

Grouping Four Adjacent Ones (Quad):

- If we group four 1's from the adjacent cells of a K-map, then the group is called as Quad.
- In such a quad two variables associated with the minterms are same and the other two are not the same.
- After forming a Quad, the simplification takes place in such a way that the two variables which are not same will be eliminated. Thus a quad eliminates two variables.
- Fig. 2.44 shows various types of Quads and the corresponding mathematical simplification.
- Note that overlapping is possible in Quads.

Example 7: Given K-map

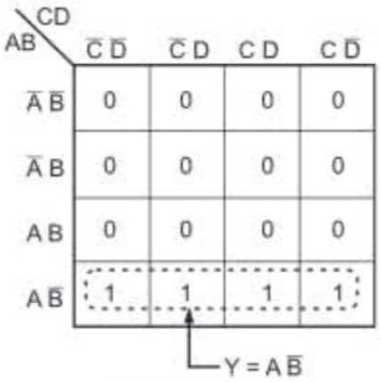


Fig. 2.44 (a)

Simplification:

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

$$= A\bar{B}[\bar{C}\bar{D} + \bar{C}D + CD + C\bar{D}]$$

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

$$\therefore \quad Y = A\bar{B}[\bar{C} + C] = A\bar{B}$$

Thus C and D are eliminated.

Example 8: Given K-map

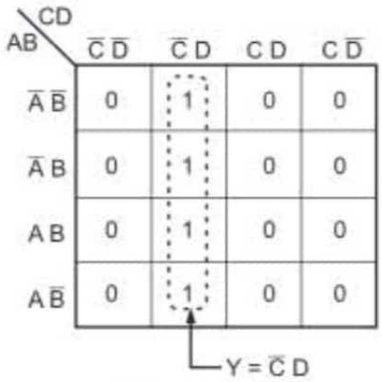


Fig. 2.44 (b)

Simplification:

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

$$= \bar{C}D[\bar{A}\bar{B} + \bar{A}B + AB + A\bar{B}]$$

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

$$= \bar{C}D[\bar{A} + A] = \bar{C}D$$

Thus A and B are eliminated.

Example 9: Four adjacent ones forming a square

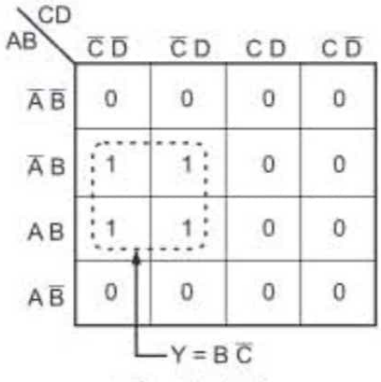


Fig. 2.44 (c)

Simplification:

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

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

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

$$\therefore \quad Y = B\bar{C}$$

Thus A and D are eliminated.

Example 10: Top and bottom 1's forming a Quad

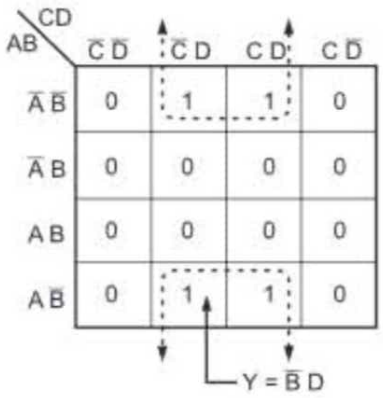


Fig. 2.44 (d)

Simplification:

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

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

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

$$\therefore \quad Y = \bar{B}D(\bar{A} + A) = \bar{B}D$$

Thus A and C are eliminated.

Example 11: Leftmost and rightmost 1's forming of Quad

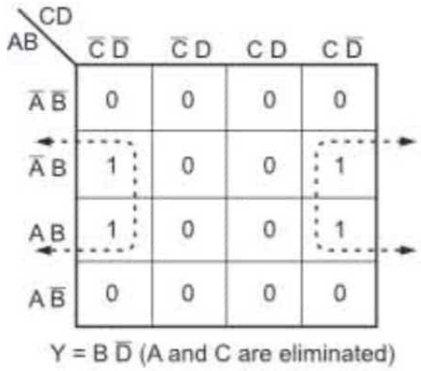


Fig. 2.44 (e)

Simplification:

$$Y = \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + \bar{A}BC\bar{D} + ABC\bar{D}$$

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

$$= B\bar{C}\bar{D} + BC\bar{D}$$

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

Thus A and C are eliminated.

Example 12: 1's corresponding to corners forming a Quad

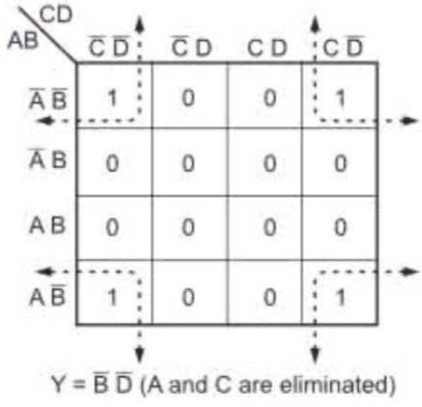


Fig. 2.44 (f)

Simplification:

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

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

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

$$\therefore \quad Y = \bar{B}\bar{D}$$

Thus A and C are eliminated.

Example 13: Overlapping of Quads and Pairs

Simplification:

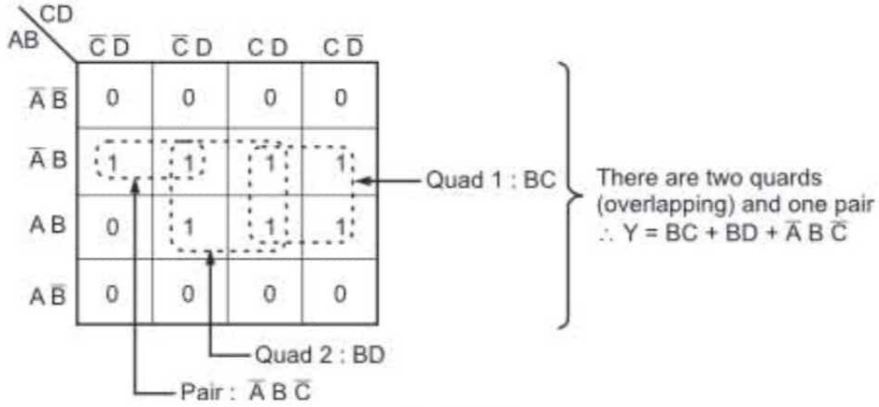


Fig. 2.44 (g)

Grouping Eight Adjacent Ones (Octet):

- It is possible to form a group of eight adjacent ones. Such a group is called as octet.
- When an octet is formed three variables will change and only one variable will remain same in all the minterms.

Example 14:

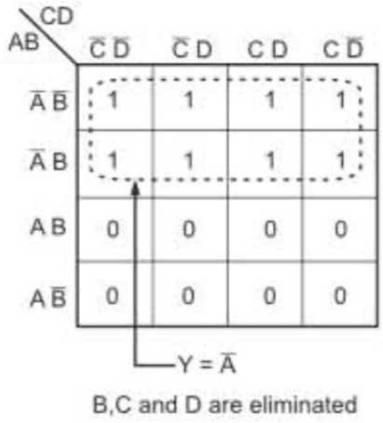


Fig. 2.45 (a)

Simplification :

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

$$= + \bar{A}BC\bar{D} + \bar{A}B\bar{C}D + \bar{A}BCD + \bar{A}BC\bar{D}$$

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

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

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

$$\therefore \quad Y = \bar{A}(\bar{B} + B) = \bar{A}$$

- The only variable that remains same is $\bar{A}$. So it appears as output.
- The three variables that change will be eliminated and the variable which does not change appear as output.
- Thus octet eliminates three variables.
- Fig. 2.45 shows various types of octets and the corresponding output.

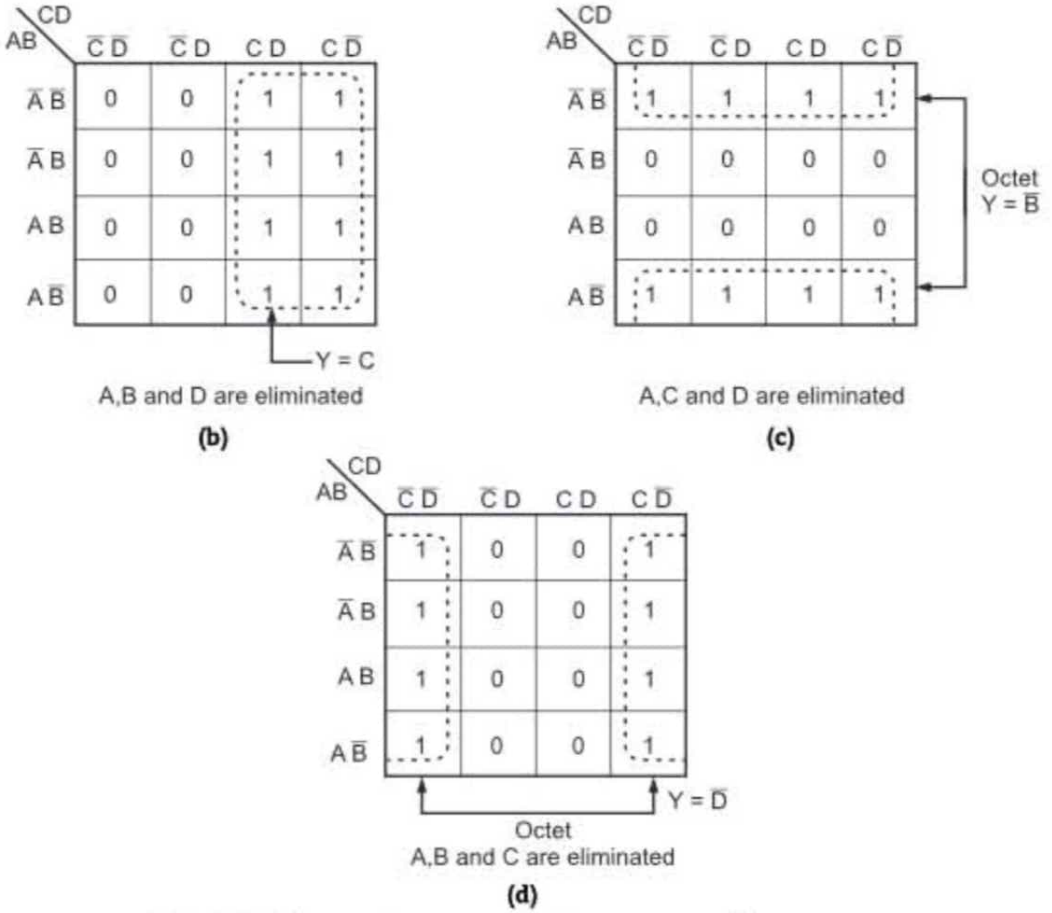


Fig. 4.45: Types of octets and the corresponding output

2.14.4 Obtain K-map for Boolean Expression

Procedure:

1. Logical expressions in the standard POS form can be represented on K-map by entering 0's in the cells of K-map corresponding to each maxterm present in the given equation.
2. The remaining cells are filled with 1's.

Example 1: Represent the following standard POS expression on Karnaugh map.

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

Solution:

- Each term in the given logical equation is a maxterm.
- Enter a 0 corresponding to each maxterm as shown in Fig. 2.46.
- The given expression has three maxterms as follows:

$$(A + B + C) = M_0 \cdot (A + \overline{B} + C) = M_2 \cdot (\overline{A} + \overline{B} + C) = M_6$$

- Hence we have to write the structure of 3 variable K-map as usual and enter 0's at $M_0 \cdot M_2$ and M_6 as shown in Fig. 2.46.

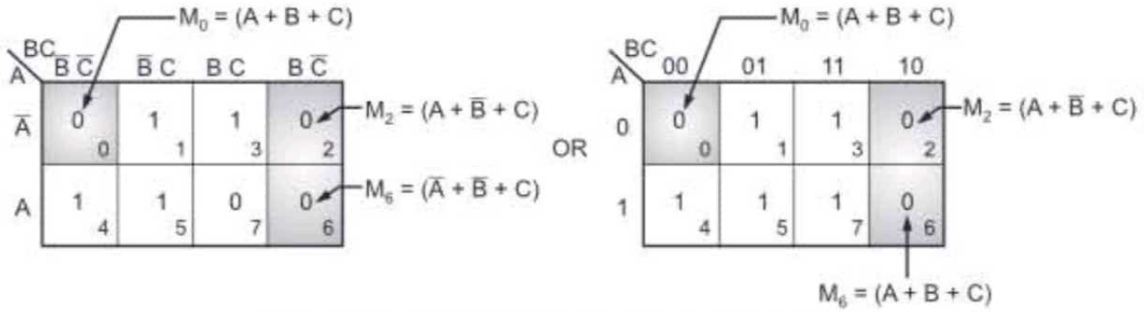


Fig. 2.46: Representation of standard POS on K-map

Example 2: Represent the following standard POS equation on the Karnaugh map.

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

Solution: The given expression contains four maxterms as follows:

$(A + B + C + \overline{D}) = M_1$ $\qquad$ $(A + \overline{B} + \overline{C} + D) = M_2$

$(A + B + \overline{C} + \overline{D}) = M_3$ $\qquad$ $(\overline{A} + \overline{B} + C + D) = M_{12}$

Enter the 0 corresponding to each maxterm as shown in Fig. 2.47.

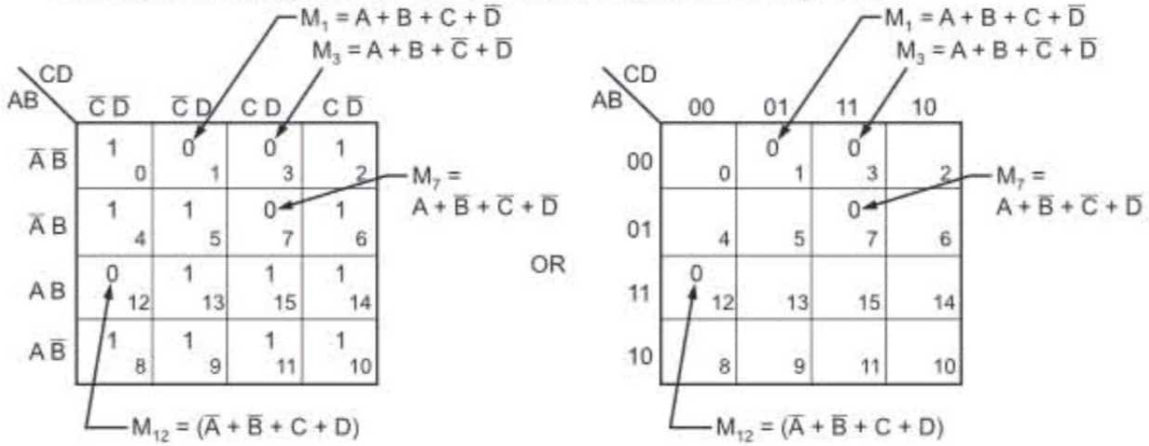


Fig. 2.47: Representation of standard POS on K-map

2.14.5 Minimize Boolean Expression using K-map

Minimization procedure:

1. The given POS expression consists of maxterms.
2. Corresponding to every maxterm enter a 0 in the K-map.
3. Enter 1's in the remaining cells of K-map.
4. Encircle/group 0's instead of 1's for carrying out the simplification.
5. Rules of simplification are exactly same those for the SOP form.

EXAMPLES

Example 1: Find the expression in the POS form for the K-map shown in Fig. 2.48.

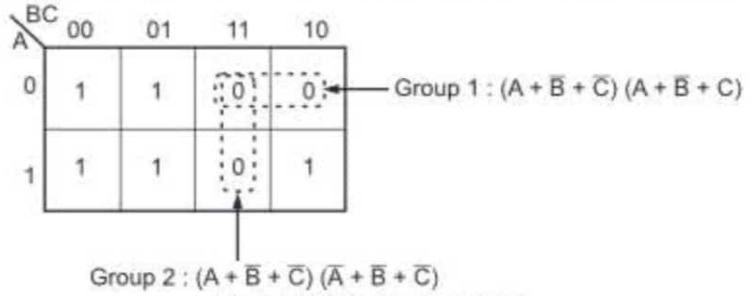


Fig. 2.48 (a): Given K-map

Solution: Group 1 $\rightarrow (A + \bar{B} + \bar{C})\ (A + \bar{B} + C)$

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

But $AA = A,\ \bar{B}\bar{B} = \bar{B},\ C\bar{C} = 0,\ A\bar{B} + A\bar{B} = A\bar{B}$

$\therefore$ $$\text{Group 1} = A + A\bar{B} + AC + \bar{B} + \bar{B}C + A\bar{C} + \bar{B}\bar{C}$$

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

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

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

$$\text{Group 1} = A + \bar{B}$$

Conclusion:

When a pair of 0's is formed, the variable which changes will get eliminated e.g. C changes for the pair of 0's in group 1. Hence it is eliminated.

$\therefore$ Group 2 $\rightarrow \bar{B} + \bar{C}$

$\therefore$ Final minimized equation is as follows:

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

(1) (2)

Group 2 : $(A + \bar{B} + \bar{C})\ (\bar{A} + \bar{B} + \bar{C})$

Remain unchanged

Variable that changes its value hence it gets eliminated

Fig. 2.48 (b)

Example 2: Minimize the following standard POS expression using K-map.

$$Y = \pi M\,(0, 2, 3, 5, 7)$$

Solution: Minimized expression is as follows:

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

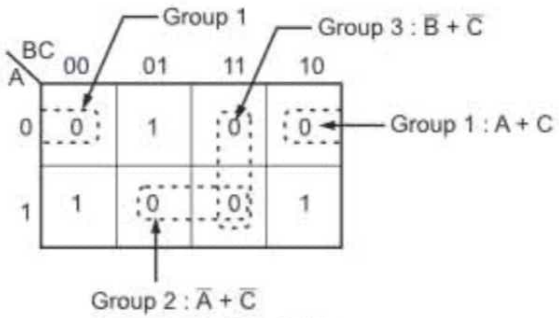


Fig. 2.49

Example 3: Find the expression for the output in the POS form for the K-map shown in given K-map.

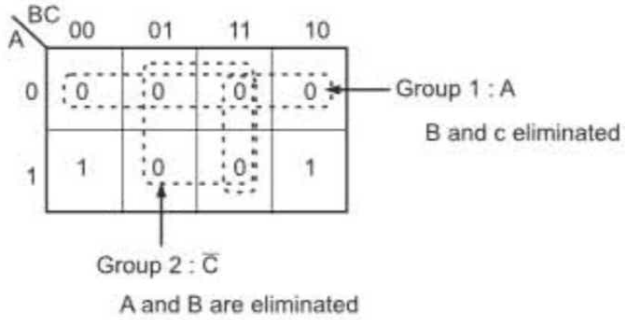


Fig. 2.50 (a): Given K-map

Solution: Minimized expression is as follows:

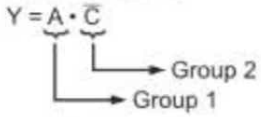


Fig. 2.50 (b)

Example 4: Write the simplified expression for output in the POS form for the following expression.

$$Y = \pi M (0, 2, 3, 7)$$

Solution: Group of zeros as shown in Fig. 2.51 for the further simplifications.

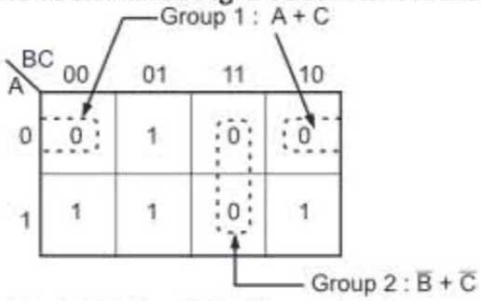


Fig. 2.51: Simplifications using K-map

Simplified equation using K-maps is as follows:

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

Example 5: Write the expression for output in the POS form for the K-map shown in Fig. 2.52.

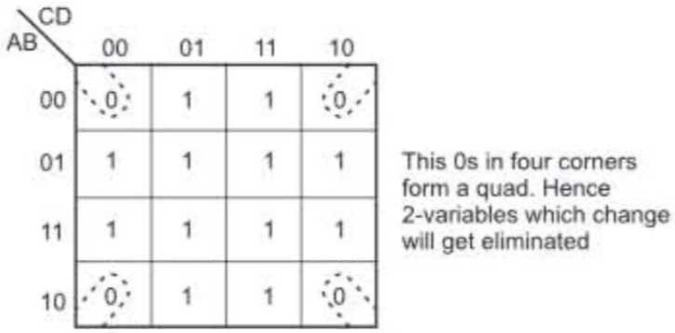


Fig. 2.52

Solution: Expression for output:

$$Y = B + D$$

Example 6: For the K-map shown in given figure. Write the expression for output in the POS form.

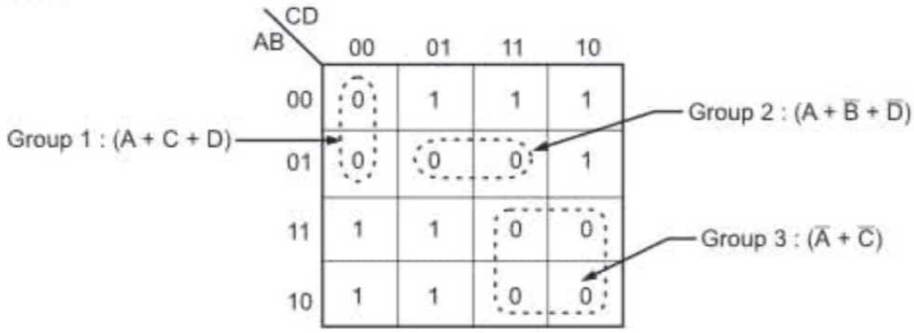


Fig. 2.53

Solution: The expression for output is:

$$Y = (A + C + D)\,(A + \overline{B} + \overline{D})\,(\overline{A} + \overline{C})$$
$$\quad\quad (1) \quad\quad\quad (2) \quad\quad (3)$$

Example 7: Group the 0's given in the K-map of the given figure, in different ways to obtain the expression for output in the POS form.

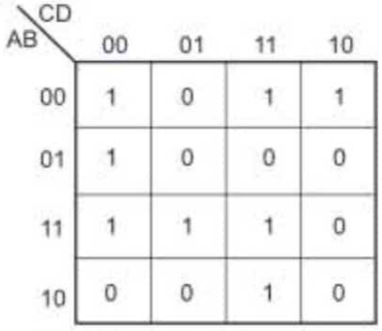


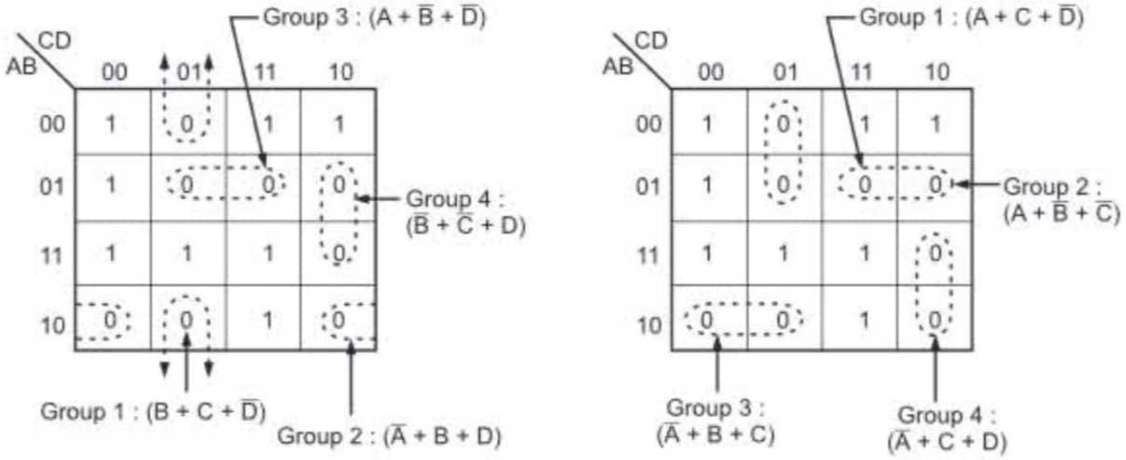
Fig. 2.54 (a): Given K-map

Solution:

1. Figs. 2.54 (b) and (c) shows two different but valid ways for grouping 0's.
2. The corresponding equations for the output in the POS form are obtained after that.
3. Referring to the grouping shown in Fig. 2.54 we get,

$$Y = (B + C + \overline{D})\,(\overline{A} + B + D)\,(A + \overline{B} + \overline{D})\,(\overline{B} + \overline{C} + D)$$
$$\quad\quad (1) \quad\quad (2) \quad\quad (3) \quad\quad (4)$$

This shows that for the same K-map we can get completely different answers and all of them are correct.



(b) One way of grouping **(c) Another way of grouping**

Fig. 2.54

Example 8: Implement the following POS expression using NOR-NOR logic.

$$F = \pi M\,(0, 1, 2, 4, 6)$$

Solution: The given Boolean function is,

$$F = M_0M_1M_2M_4M_6$$

Hence the corresponding K-map is as shown in Fig. 2.55 (a).

∴ Simplified Boolean equation is as follows:

$$F = (A + B)\,C \quad \ldots (1)$$

∴ $$F = \overline{\overline{(A + B) \cdot C}} = \overline{\overline{(A + B)} + \overline{C}}$$ ... DeMorgan's theorem

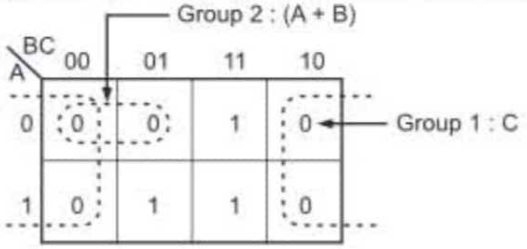


Fig. 2.55 (a)

The expression can be implemented using only NOR gates as shown in Fig. 2.55 (b).

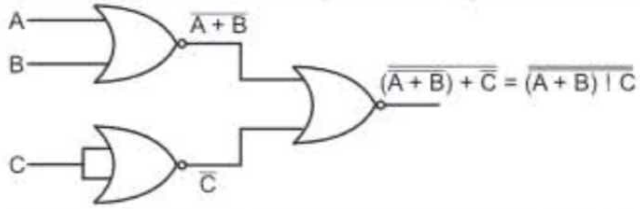


Fig. 2.55 (b): Implementation using NOR-NOR logic

Example 9: Minimize the function using K-map and implement using only NOR gates:
$f(w, x, y, z) = \pi M (1, 3, 5, 7, 13, 15)$

Solution:

Step 1 : To simplify the given function using K-map:

The given Boolean function is

$$f(w, x, y, z) = M_1M_3M_5M_7M_{13}M_{15}$$

Hence the corresponding K-map is as shown in Fig. 2.56 (a).

$\therefore$ The simplified equation is

$$f(w, x, y, z) = (w + \overline{z})\ (\overline{x} + \overline{z})$$

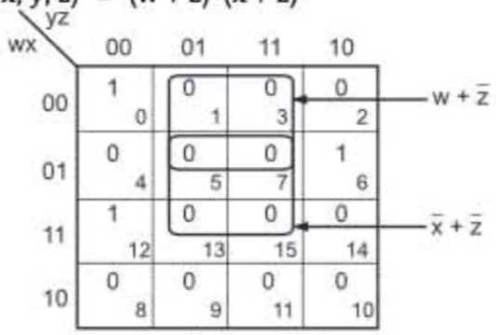


Fig. 2.56 (a)

Step 2 : Implementation using NOR gates:

$$f(w, x, y, z) = \overline{\overline{(w + \overline{z})\ (\overline{x} + \overline{z})}} \quad (\because \overline{\overline{Y}} = Y)$$

$$= \overline{\overline{(w + \overline{z})}\ \overline{(\overline{x} + \overline{z})}}$$

The implementation using NOR gates is as shown in Fig. 2.56 (b).

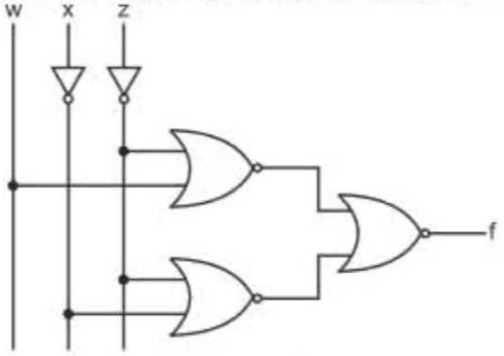


Fig. 2.56 (b)

Example 10: Minimize the function using K-map and implement using NOR gates only:

$$f(A, B, C, D) = \pi M(0, 1, 3, 4, 5, 7, 9)$$

Solution: ∴ The minimized expression is as follows:

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

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

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

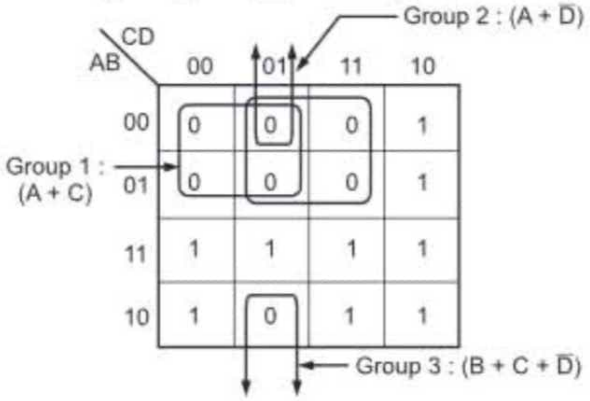


Fig. 2.57 (a)

Implementation using NOR gates:

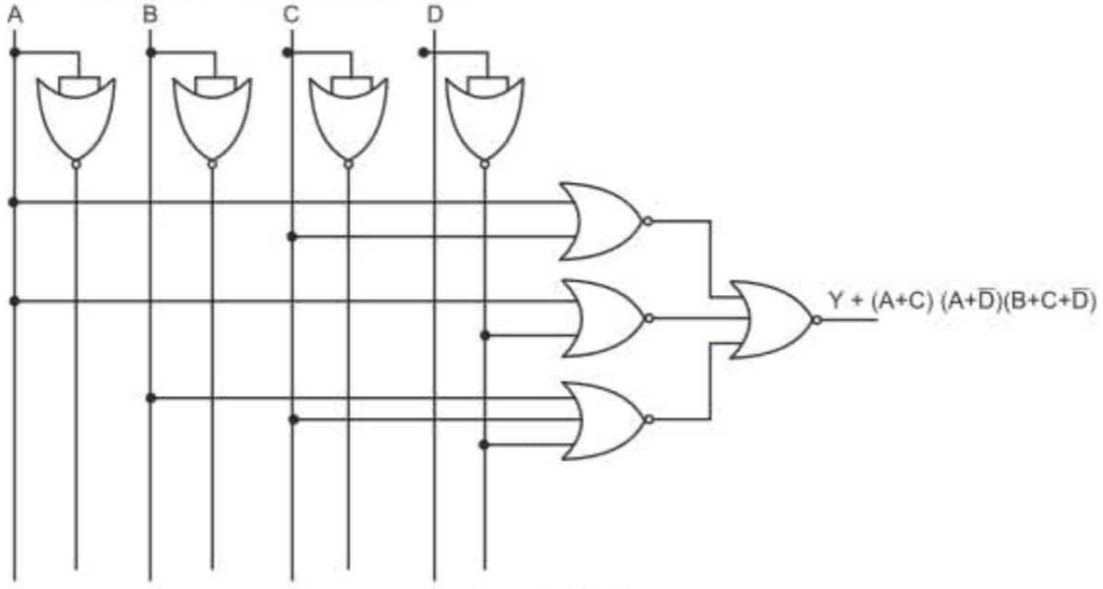


Fig. 2.57 (b)

Summary

- Boolean Algebra is the mathematical foundation of digital circuits. Boolean Algebra specifies the relationship between Boolean variables which is used to design combinational logic circuits using Logic Gates.
- Logic gates are electronic circuits that implement the basic functions of Boolean Algebra.
- The truth table shows a logic circuit's output response to all of the input combinations.
- If the input value for a NOT gate is 0, the output value is 1, and if the input value is 1, the output is 0. A NOT gate is sometimes referred to as an inverter because it inverts the input value.
- An AND gate accepts two input signals. If the two input values for an AND gate are both 1, the output is 1; otherwise, the output is 0.
- In OR gate, If the two input values are both 0, the output value is 0; otherwise, the output is 1.
- The two input XOR gate produces 0 if its two inputs are the same, and 1 otherwise.
- The NAND and NOR gates are essentially the opposite of the AND and OR gates, respectively.
- A product term in which all the variables appear exactly once, either complemented or uncomplemented, is called a minterm. A minterm represents exactly one combination of the binary variables in a truth table. It has the value of 1 for that combination and 0 for the others
- A sum term in which all the variables appear exactly once, either complemented or uncomplemented, is called a maxterm. A maxterm represents exactly one combination of the binary variables in a truth table. It has the value of 0 for that combination and 1 for the others.
- **Karnaugh Map:** A graphical technique for simplifying an expression into a minimal sum of products (MSP) form. There are a minimal number of product terms in the expression. Each term has a minimal number of literals.
- A Boolean function can be represented algebraically from a given truth table by forming the logical sum of all the minterms that produce a 1 in the function. This expression is called a sum of minterms.
- A Boolean function can be represented algebraically from a given truth table by forming the logical product of all the maxterms that produce a 0 in the function. This expression is called a product of maxterms.
- The De Morgan's theorem mostly used in digital programming and for making digital circuit diagrams. There are two De Morgan's Theorems.
- De Morgan's First Theorem : $\overline{A + B} = \bar{A} \cdot \bar{B}$
- De Morgan's Second Theorem : $\overline{A \cdot B} = \bar{A} + \bar{B}$
- A universal gate is a gate which can implement any Boolean function without need to use any other gate type. The NAND and NOR gates are universal gates.

Check Your Understanding

1. To perform product of maxterms Boolean function must be brought into _____
   (a) AND terms (b) OR terms
   (c) NOT terms (d) NAND terms
2. Maxterms are also called _____
   (a) standard sum (b) standard product
   (c) standard division (d) standard subtraction
3. Symbol representing AND operation _____
   (a) (+) (b) (.)
   (c) (–) (d) (/)
4. X + 0 = 0 + X = X is an example of _____
   (a) commutative property (b) inverse property
   (c) associative property (d) Identity element
5. Complement of a function expressed as _____
   (a) sum of minterms (b) product of minterms
   (c) sum of maxterms (d) product of maxterms

Answers

1. (b)	2. (a)	3. (b)	4. (a)	5. (a)

Practice Questions

Q.1 Answer the following Question in brief:

1. What is logic gate?
2. Write short note on: Boolean algebra.
3. Enlist types of logic gates? Name only.
4. Compare analog and digital signals.
5. What is K-map?

Q.2 Answer the following Question:

1. What is binary logic? Explain positive and negative logic in detail.
2. Describe De Morgan's theorem in detail.
3. Explain the following gates with truth table and symbols: (i) AND (ii) OR (iii) X-OR (iv) NOT (v) X-NOR (vi) NAND.
4. Enlist types of laws used in Boolean algebra.
5. Describe inter-conversion of logic gates with example.
6. Explain K-map technique to minimize Boolean expression.

Q.3 Define Terms:

1. Define the terms: (i) Signal, (ii) Logic, (iii) Truth table, (iv) Binary logic, (v) Maxterm, (vi) Minterm, (vii) K-Map, (viii) POS form, (ix) Analog signal, (x) Digital Signal.

Previous Exam Questions

Summer 2017

1. $A + \bar{A}B =$ _____
   (a) B (b) A + B
   (c) AB (d) None of the above

Ans. Refer to Section 2.5.

2. What is positive and negative logic? **(1 M)**

Ans. Refer to Section 2.2.

3. Draw the symbol for NOT gate and OR gate. **(1 M)**

Ans. Refer to Sections 2.3.2 and 2.3.3.

4. Build AND, OR, NAND and EX-OR gate using NOR gate. **(5 M)**

Ans. Refer to Section 2.12.

5. Simplify using Boolean algebra and draw simplification diagram

$$A\bar{B}C + \bar{A}\bar{C}D + \bar{C}A$$ **(3 M)**

Ans. Refer to Section 2.14.3.

6. Draw logic gate, and give Boolean function and truth table for EX-NOR gate. **(4 M)**

Ans. Refer to Section 2.10.

Winter 2017

1. $A \cdot (\bar{A} + B) =$ _____ **(1 M)**

(a) AB (b) A + B

(c) B (d) A

Ans. Refer to Section 2.5.

2. What is positive and negative logic? **(1 M)**

Ans. Refer to Section 2.2.

3. Explain AND gate. Also draw the diode diagram of an AND gate. **(5 M)**

Ans. Refer to Section 2.7.1.

4. Draw the logic gate to implement: **(3 M)**

(i) $AB + AC + A\bar{B}C$

(ii) $\overline{(A + B) \cdot (C + D)}$

(iii) $(A + BC) \cdot (\bar{C} + B)$

Ans. Refer to Examples in Section 2.12.

Summer 2018

1. State DeMorgan's 2nd theorem and prove it. **(3 M)**

Ans. Refer to Section 2.6.

2. Build OR and AND gate using NOR gate. **(3 M)**

Ans. Refer to Section 2.12.

3. Draw logic gate, Boolean equation and truth table for all 3 I/P universal logic gates. **(4 M)**

Ans. Refer to Section 2.11.

■■■

3... Combinational Circuits

Objectives...

- To study about Multi-input, Multi-output Combinational circuits and Code Convertors.
- To get knowledge about Arithmetic Circuits such as Adder, Subtractor, Multilpier, Comparator.
- To Learn about Multiplexer, Demultiplexer, ALU, Encoder and Decoder.

3.1 INTRODUCTION [W-17]

- **Definition:** A circuit in which the output/outputs depend on the present state of combination of the logic inputs is called as Combinational Circuits.
- Some of the characteristics of combinational circuits are:
  1. The combinational circuits do not use any memory. The previous state of input does not have any effect on the present state of the circuit.
  2. A combinational circuit can have n number of inputs and m number of outputs.
  3. The output of combinational circuit at any instant of time depends only on the levels present at input terminals.
  4. The combinational circuit consists of only logic gates. Logic gates are the basic building blocks of these circuits.
  5. Combinational circuit is a circuit in which we combine the different gates in the circuit such as Adders, Encoder, Decoder, Multiplexer and Demultiplexer.
- Fig. 3.1 shows block diagram of Combinational Circuits.

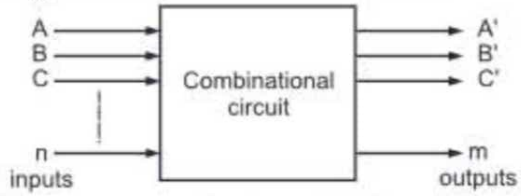


Fig. 3.1: A Generalized Combinational Circuit

- In above block diagram, we can see that combinational logic circuit has "n" inputs that means it can take 2^n combination of input values. The output "m" is depending on the Boolean expression of combinational logic circuit.

- The main difference between sequential circuits and combinational circuits is that sequential circuits compute their output based on input and state, and that the state is updated based on a clock. Combinational logic circuits implement Boolean functions and are functions only of their inputs.

3.1.1 Representing Combinational Logic Functions

- There are 3 ways to represent combinational logic functions:
  1. **Logic gates:** Logic gates are used as the building blocks in the design of combinational logic circuits. These gates are the AND, OR, NOT, NAND, NOR gates.
  2. **Boolean Algebra:** Boolean Algebra specifies the relationship between Boolean variables which is used to design digital circuits using Logic Gates. Every logic circuit can be completely described using the Boolean operations, because the OR, AND gate, and NOT gates are the basic building blocks of digital systems.
  3. **Truth table:** A truth table is used in logic to compute the functional values of logical expressions on each combination of values taken by their logical variables. If a combination logic block have more than one bit output, each single-bit output gets its own truth-table. Often they are combined into a single table with multiple output columns, one for each single-bit output.

3.1.2 Classification of Combinational Logic Circuits

- The combinational logic circuits can be classified into various types based on the purpose of usage, such as arithmetic and logical functions, data transmission, and code converters.
- To solve the arithmetic and logical functions we generally use adders, subtractors, and comparators which are generally realized by combining various logic gates called as combinational logic circuits. Similarly, for data transmission, we use multiplexers, demultiplexers, encoders, and decoders which are also realized using combinational logic.
- The code converters such as binary, BCD, and 7-segment are designed using various logic circuits.

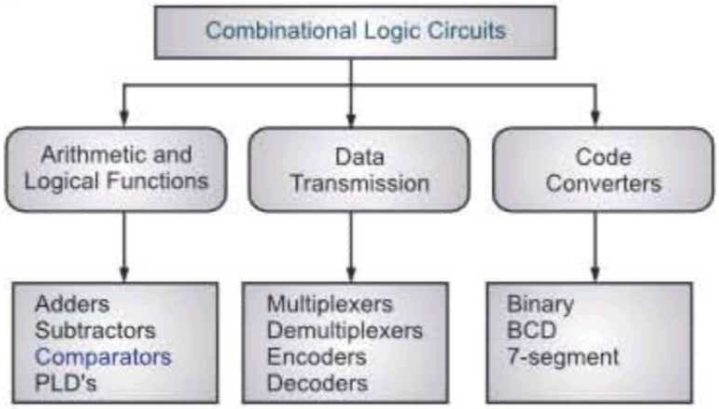


Fig. 3.2: Classification of Combinational Logic Circuits

- In fact, combinational logic is most frequently used in multiplexer and demultiplexer type circuits. If multiple inputs or outputs are connected to the common signal line, then the logic gates are used for decoding an address in order to select single data input or output switch.
- A combinational circuit can be designed using the following steps.

  Step 1 : Identification and determination of number of available input variables and required output variables.

  Step 2 : Representing symbols (alphabets) for each and every input and output variables.

  Step 3 : Expressing the input and output variable's relationship.

  Step 4 : Construction of truth table indicating the relationship between the input and output variables.

  Step 5 : Obtaining the Boolean expression for each output variable in terms of input variables.

  Step 6 : Minimizing the Boolean expressions of various output variables.

  Step 7 : Obtaining the logic diagram by the implementation of minimized Boolean expressions.

3.1.2.1 Multi-input Combinational Circuit

The Use of Logic Gate:

- Adding more input terminals to a logic gate increases the number of input state possibilities. With a single-input gate such as the inverter or buffer, there can be only two possible input states: either the input is "high" (1) or it is "low" (0).
- A two input gate has four possibilities (00, 01, 10, and 11).
- A three-input gate has eight possibilities (000, 001, 010, 011, 100, 101, 110, and 111) for input states.
- The number of possible input states is equal to two to the power of the number of inputs:

  $$\text{Number of possible input states} = 2^n$$

  where, n = Number of inputs
- This increase in the number of possible input states allows for more complex gate behavior. Now, instead of merely inverting or amplifying (buffering) a single "high" or "low" logic level, the output of the gate will be determined by whatever combination of 1's and 0's is present at the input terminals.
- Since so many combinations are possible with just a few input terminals, there are many different types of multiple-input gates, unlike single-input gates which can only be inverters or buffers.

Making Multi Input Gates:

1. The AND Gate

- Multi input gates can be made by joining gates of the same type with less inputs. The diagrams below shows how a three input AND gate and a four input AND gate can be made out of two input AND gates.

- The output of AND gate will be "high" (1) if and only if all inputs (first input and the second input and . . .) are "high" (1). If any input(s) is "low" (0), the output is guaranteed to be in a "low" state as well.

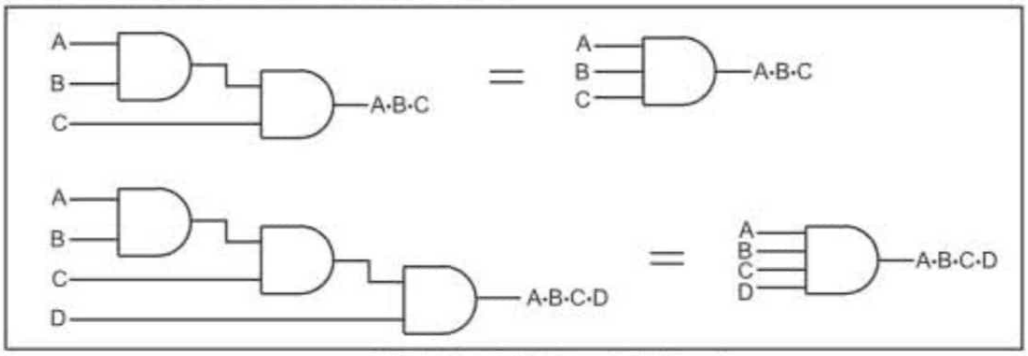


Fig. 3.3: Multi-Input AND gate

2. The NAND Gate:

- As with AND gates, NAND gates are made with more than two inputs. In such cases, the same general principle applies: the output will be "low" (0) if and only if all inputs are "high" (1). If any input is "low" (0), the output will go "high" (1).

3. The OR Gate:

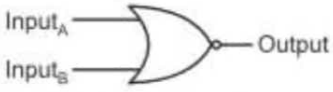
- The output of OR gate will be "high" (1) if any of the inputs (first input or the second input or . . .) are "high" (1). The output of an OR gate goes "low" (0) if and only if all inputs are "low" (0).

(a) 2-input OR gate **(b) 3-input OR gate**

Fig. 3.4: Multi-Input OR gate

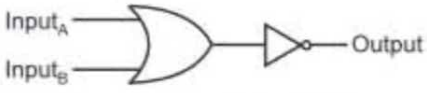
4. The NOR Gate:

- As you might have suspected, the NOR gate is an OR gate with its output inverted, just like a NAND gate is an AND gate with an inverted output.



(a) 2-input NOR gate

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0



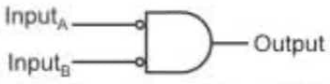
(b) Equivalent gate circuit

Fig. 3.5: NOR gate

- For NOR gates the same logical principle applies: the output goes "low" (0) if any of the inputs are made "high" (1). The output is "high" (1) only when all inputs are "low" (0).

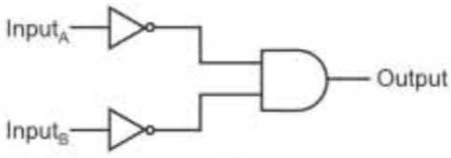
5. The Negative-AND Gate

- A Negative-AND gate functions the same as an AND gate with all its inputs inverted (connected through NOT gates). In keeping with standard gate symbol convention, these inverted inputs are signified by bubbles. Contrary to most peoples' first instinct, the logical behavior of a Negative-AND gate is *not* the same as a NAND gate. Its truth table is identical to a NOR gate:



(a) 2-input Negative-AND gate

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0



(b)

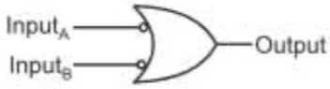
(c)

Equivalent gate circuits

Fig. 3.6: Negative-AND gate

6. The Negative-OR Gate:

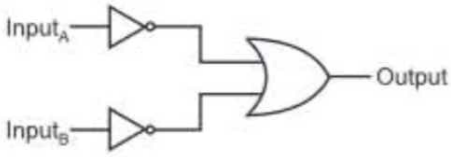
- Following the same pattern, a Negative-OR gate functions the same as an OR gate with all its inputs inverted. In keeping with standard gate symbol convention, these inverted inputs are signified by bubbles. The behavior and truth table of a Negative-OR gate is the same as for a NAND gate:



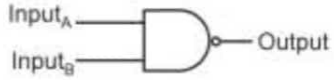
2-input Negative-OR gate

(a)

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0



(b)



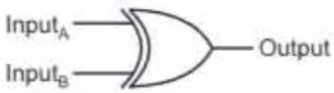
(c)

Equivalent gate circuits

Fig. 3.7: Negative-OR gate

7. The Exclusive-OR Gate:

- The last six gate types are all fairly direct variations on three basic functions: AND, OR, and NOT. The Exclusive-OR gate, however, is something quite different.
- Exclusive-OR gates output a "high" (1) logic level if the inputs are at different logic levels, either 0 and 1 or 1 and 0. Conversely, they output a "low" (0) logic level if the inputs are at the same logic levels. The Exclusive-OR (sometimes called XOR) gate has unique symbol and a truth table pattern.

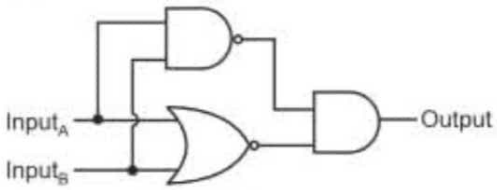


(a)

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 3.8: Exclusive-OR gate

- There are equivalent circuits for an Exclusive-OR gate made up of AND, OR, and NOT gates, just as there were for NAND, NOR, and the negative-input gates. A rather direct approach to simulating an Exclusive-OR gate is to start with a regular OR gate, then add additional gates to inhibit the output from going "high" (1) when both inputs are "high" (1):

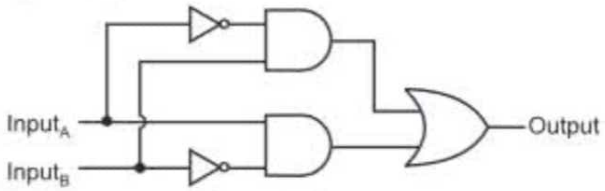


(b)

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 3.8: Exclusive-OR equivalent circuit

- In this circuit, the final AND gate act as a buffer for the output of the OR gate. Whenever the NAND gate's output is high, for the first three input state combinations (00, 01, and 10). However, when both inputs are "high" (1), the NAND gate outputs a "low" (0) logic level, which forces the final AND gate to produce a "low" (0) output.
- Another equivalent circuit for the Exclusive-OR gate uses a strategy of two AND gates with inverters. It sets up to generate "high" (1) outputs for input conditions 01 and 10. A final OR gate then allows either of the AND gates' "high" outputs to create a final "high" output:



(c)

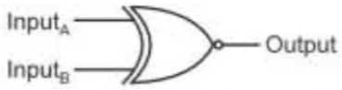
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 3.8: Exclusive-OR equivalent circuit

- Exclusive-OR gates are very useful for circuits where two or more binary numbers are to be compared bit-for-bit, and also for error detection (parity check) and code conversion (binary to Grey and vice versa).

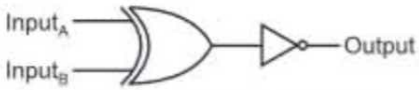
8. The Exclusive-NOR Gate

- The Exclusive-NOR gate known as the XNOR gate. It is equivalent to an Exclusive-OR gate with an inverted output. The truth table for this gate is exactly opposite as for the Exclusive-OR gate:



(a) Exclusive-NOR Gate

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1



(b) Equivalent gate circuit

Fig. 3.9

- As indicated by the truth table, the purpose of an Exclusive-NOR gate is to output a "high" (1) logic level whenever both inputs are at the same logic levels (either 00 or 11).

3.1.2.2 Multi-output Combinational Circuits

- A combinational circuit may have more than one output. In such a situation, each output must be expressed as a function of the inputs. A digital circuit called the "code converter" is an example of multiple-output circuits.
- A code converter transforms information from one binary code to another.
- Basic logic gates can be combined to form a variety of higher-level units:

1. **Routing:**
   - **Multiplexers:** These logic gates have several data input signals, as well as a control input. The output is identical to one of the inputs. The value of the control signal determines which input to take into account.
   - **Demultiplexers:** These logic gates have one data input signal, a control input, and several output signals. All of the output signals are 0 (false) except for the one selected by the control input. The selected output is identical to the data input.
2. **Computational:**
   - **Full adders:** This type of logic gate performs a single column of a binary addition. Full adders are the primary building block for multi-bit adders and subtractor.
   - **Adders and Subtractors:** These circuits used to add or subtract two binary or two complement numbers. A subtractor is just an adder with extra circuitry, which allows it to perform a two-complement operation on one of the inputs. They are typically designed to do *either* addition *or* subtraction, as directed by a control signal.
   - **Comparators:** These compare either two binary or two complement numbers.
3. **State Circuitry:**
   - This type of gate performs like an object method. The output is not only based on the input. It is also based on the historical inputs. This is made possible by the memory embedded in the circuitry.
   - State circuitry contains anything that can recollect bits of information, including memory, registers, and program counters.
   - The basic element of state circuitry is a flip-flop. A flip-flop stores one bit of data. Multiple flip-flops can be combined to form a multi-bit state element called a register. Multiple registers can be combined into a register bank.
4. **Processor Datapath:**
   - A processor's datapath is conceptually organized into two parts:
     1. **Combinational logic** determines the state of the processor for the next clock cycle. The ALU is combinational logic circuit.
     2. **State elements** hold information about the state of the processor during the current clock cycle. All registers are state elements.

3.2 CODE CONVERTOR'S DESIGN AND IMPLEMENTATION

- It may happen that the use of output of one number system as the input to another number system. Hence it is necessary to design a conversion circuit and insert it between the two systems.
- Thus a code converter is a circuit, which accepts the input information in one binary code, converts it and produces an output into another binary code i.e., which makes the two systems compatible, even though each uses a different binary code.
- To convert from binary Code x to binary Code y, the input lines must supply the bit of combination of elements as specified by Code x and the output lines must generate the corresponding bit combination of Code y. A combinational circuit performs this transformation by means of logic gates.
- The general block diagram of a code converter is shown in Fig. 3.10.

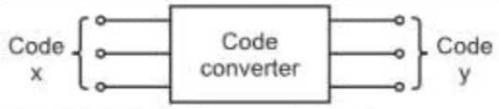


Fig. 3.10: Block diagram of code converter

- There are a wide variety of binary codes used in digital systems. Some of these codes are BCD, EX-3, gray and so on. Now we will see the design of different code converters.

3.3 ARITHMETIC CIRCUITS [S-18]

- Arithmetic operations are one of the main functionalities of most computers and calculators. These operations are carried by the logic gates or simply combinational circuits which combines the several logic gates to perform the required function. These arithmetic functionalities of the combinational circuits include addition, subtraction, multiplication, etc.
- Some of the combinational circuits used for these operations are Half- adder, Full-adder, Half-subtractors, Full subtractors, Adder-subtractors, Comparators, PLDs (Programmable Logic Devices), etc.

3.3.1 Adders

- Addition is the most fundamental operation. A simple addition consists of four possible elementary operations as follows:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

- The binary addition in the first three operations results in a sum of one digit. But in the last binary addition, the sum consists of two binary digits. The lower significant bit of this result is called *sum* and the most significant bit is called the *carry bit*.
- There are two types of Adders: Half Adder and Full Adder.

3.3.1.1 Half Adder [S-17]

- A binary half adder is a combinational logic circuit which adds two bits of binary data, producing a sum bit and a carry bit as the two output signals.
- It has two one bit inputs A and B and two outputs sum and carry.

Fig. 3.11 (a): Symbol of Half Adder

Truth Table

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

Logic Implementation of Half adder:

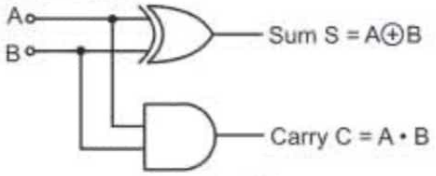


Fig. 3.11 (b)

Logic Implementation of Half adder Using Basic Gates:

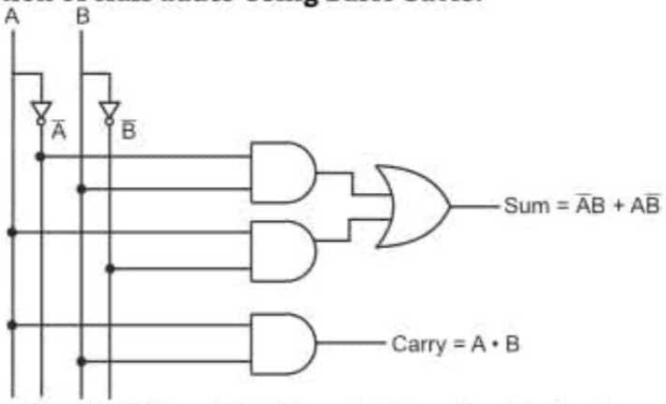


Fig. 3.11 (c): Logic Implementation using Basic gates

3.3.1.2 Full Adder [S-18]

- A full adder is a combinational circuit that performs sum of three input bits. This circuit has three inputs and two outputs.
- The inputs A and B represent the two bits to be added. The third input C_{in} represents the carry from the previous lower significant position. There are two outputs: sum S and carry C_{out}.

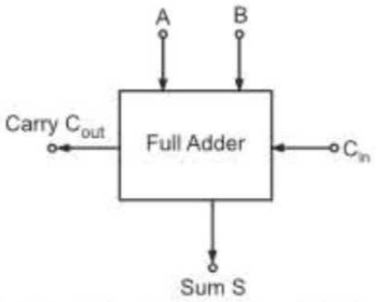


Fig. 3.12 (a): Symbol of Full Adder

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

Logic Implementation of Full Adder:

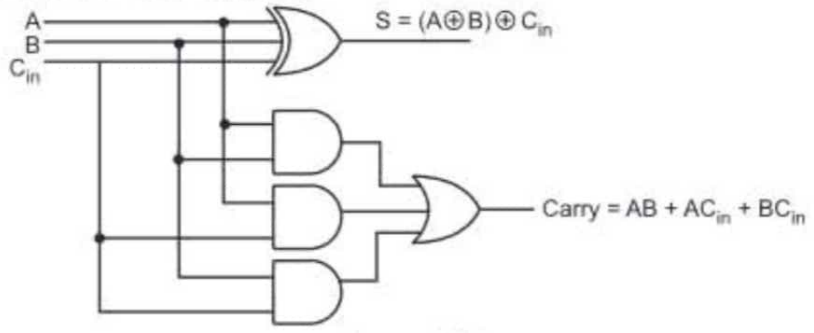


Fig. 3.12 (b)

Full Adder using Half Adders:

- Fig. 3.12 (c) shows implementation of full adder using half adder.

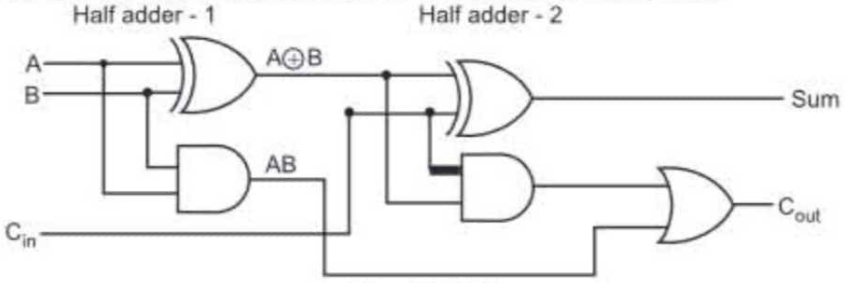


Fig. 3.12 (c)

3.3.2 Subtractor [S-17]

- The subtraction of two binary numbers can be done by taking the complement of the subtrahend and adding it to minuend.
- The rules of binary subtraction are as follows:

$0 - 0 = 0$
$0 - 1 = 1$ (Borrow = 1)
$1 - 0 = 1$
$1 - 1 = 0$

- There are two types of binary subtractors:
  1. Half subtractor.
  2. Full subtractor.

3.3.2.1 Half Subtractor

- Half subtractor is a combinational circuit with two inputs and two outputs (difference and borrow).

Truth Table

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

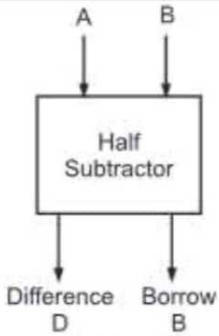


Fig. 3.13 (a) Symbol of Half Subtractor

Logic implementation of half subtractor:

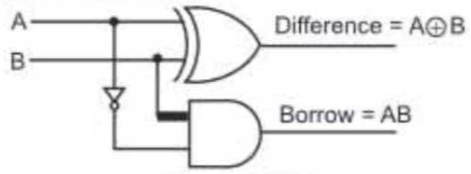
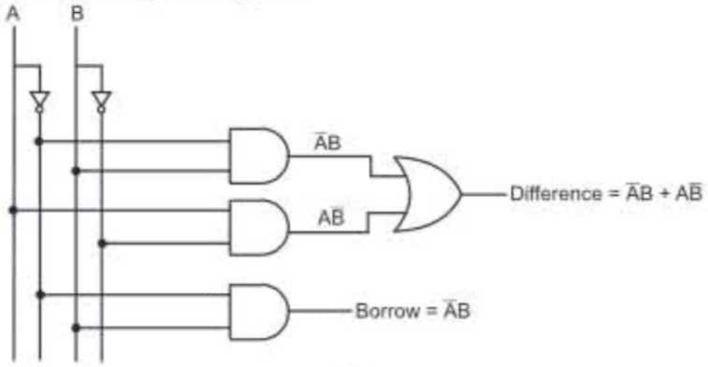


Fig. 3.13 (b)

Logic implementation using basic gates



(c)

Fig. 3.13

3.3.2.2 Full Subtractor

- The full subtractor is a combinational circuit which performs subtraction between two bits considering that 1 has been borrowed from the previous stage.

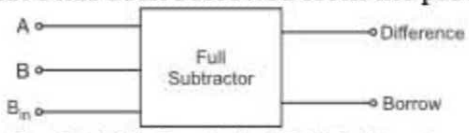


Fig. 3.14 (a): Symbol of Full Subtractor

Truth Table

Inputs			Outputs	
A	**B**	**B_{in}**	**Difference ($A - B - B_{in}$)**	**B_0**
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

Logic implementation of full subtractor:

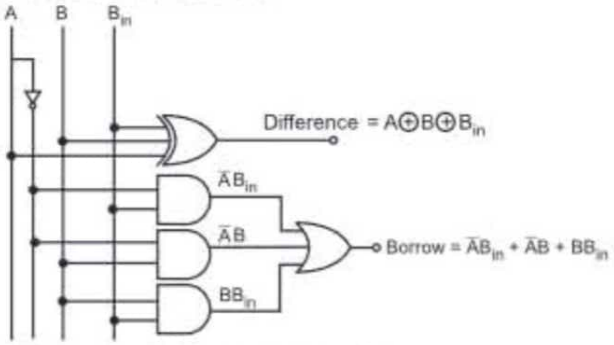


Fig. 3.14 (b)

Full subtractor using Half subtractor

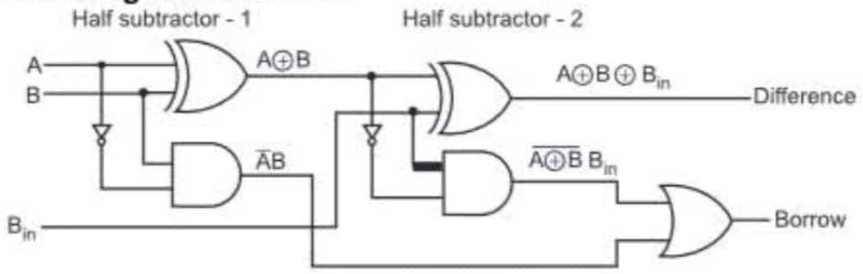


Fig. 3.14 (c)

3.4 BCD ADDER

Decimal Adder / BCD Adder:

- A Decimal Adder is the digital systems that handles the decimal number in the form of binary coded decimal numbers (BCD). A BCD adder is a circuit that adds two BCD digits and produces a sum digit also in BCD. BCD numbers use 10 digits, 0 to 9 which are represented in the binary form 0 0 0 0 to 1 0 0 1, i.e. each BCD digit is represented as a 4-bit binary number. A carry that is generated if this sum exceeds a decimal value of 9.
- When we write BCD number say 526, it can be represented as

5	2	6
↓	↓	↓
0 1 0 1	0 0 1 0	0 1 1 0

- Here, we should note that BCD cannot be greater than 9.
- In BCD addition of two numbers involve following rules:-
  1. Maximum value of the sum for two digits = 9 (max digit 1) + 9 (max digit 2) + 1 (previous addition carry) = 19
  2. If sum of two BCD digits is less than or equal to 9 (1001) without carry then the result is a correct BCD number.
  3. If sum of two BCD digits is greater than or equal to 10 (1010) the result is in-correct BCD number. Perform steps 4 for correct BCD sum.
  4. Add 6 (0110) to the result.

Example: Add 599 and 984 using BCD numbers.

	1	**2**	**3**
599 →	0101	1001	1001
+984 →	1001	1000	0100
	1110	10001	1101

Binary Sum 1, 2 and 3 are grater than 1010.
So, from Step 3 and 4 add '6' to sum.

Carry	**1**		
Result	1110	10001	1101
+6	0110	0110	0110
	0101	1000	0011
	(5)	(8)	(3)
End Carry		1	

Answer: Result of BCD addition is 1583.

Example : Add 9 and 7.

$$\begin{array}{rll} & 111 & \\ \therefore\ 9 \Rightarrow & 1001 & \\ +\ 7 \Rightarrow & 0111 & \\ \hline & {}^{1}0000 \Rightarrow & \text{As carry generated, add '6' to them} \\ & +\ 0110 & \\ \hline & \boxed{1}\ \boxed{0110} & \\ & 1\quad 6 \Rightarrow & \text{16 in BCD} \end{array}$$

$\therefore$ 9 + 7 = **16**

3.5 BCD SUBTRACTOR

- There are several methods of **BCD Subtraction**.
- BCD subtraction can be done by 1's complement method and 9's complement method or 10's complement method.
- Among all these methods 9's complement method or 10's complement method is the most easiest.
- We will clear our idea on both the methods of **BCD Subtraction**.

Method 1 of BCD Subtraction: 1's Complement Method

- In 1st method we will do **BCD Subtraction** by 1's complement method. There are several steps for this method shown below. They are:
  - At first, 1's complement of the subtrahend is done.
  - Then the complemented subtrahend is added to the other number from which the subtraction is to be done. This is called adder 1.
  - Now, in BCD subtraction, there is a term **'EAC (end-around-carry)'**.
  - If there is a carry i.e if EAC = 1, the result of the subtraction is +ve and if EAC = 0 then the result is –ve. A following table has shown the rules of EAC.

Table: Rules of EAC

Carry of individual groups	EAC = 1	EAC = 0
1	Transfer real result of adder 1 and add 0000 in adder 2	Transfer 1's complement result of adder 1 and add 1010 in adder 2
0	Transfer real result of adder 1 and add 1010 in adder 2	Transfer 1's complement result of adder 1 and add 0000 to adder 2

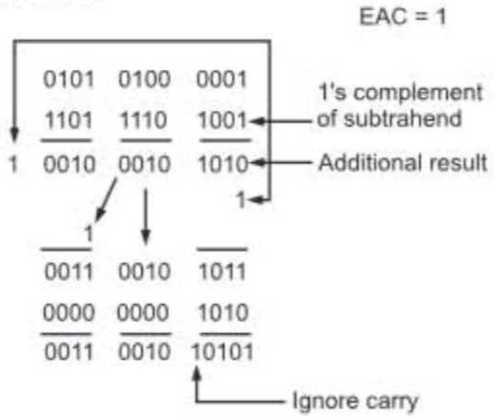
- In the final result, if any carry bit occurs, it will be ignored.

Example 1: In this example 0010 0001 0110 is subtracted from 0101 0100 0001.

Solution:

- At first, 1's complement of the subtrahend is done, which is 1101 1110 1001 and is added to 0101 0100 0001. This step is called adder 1.

- Now after addition, if any carry occurs, then it will be added to the next group of numbers towards MSB. Then EAC will be examined.
- Here, EAC = 1. So the result of addition is positive and true result of adder 1 will be transferred to adder 2.
- Now, notice from LSB. There are three groups of four bit numbers.
- 1010 is added to 1011 which is the first group of numbers because it do not have any carry. The result of the addition is the final answer.
- Carry 1 will be ignored as it is from the rule.
- Now, move to the next group of numbers. 0000 is added to 0010 and gives the result 0010. It is the final result again.
- Now, again move to the next group. Here 0000 is also added to 0011 to give the final result 0011.
- You may have noticed that, in this two groups 0000 is added, because result of first adder do not contain any carry. Thus, the results of the adder 2 is the final result of BCD Subtraction.



∴ (0101 0100 0001) – (0010 0001 0110) = (0011 0010 0101)

- Now you can check yourself.

$$(0101\ 0100\ 0001) = (541)_{10}$$
$$(0010\ 0001\ 0110) = (216)_{10}$$
$$(0011\ 0010\ 0101) = (325)_{10}$$

- We know that, 541 – 216 = 325. Thus, we can say that our result of **BCD Subtraction** is correct.

Example 2: In this example, let 0101 0001 be subtracted from 0100 1001.

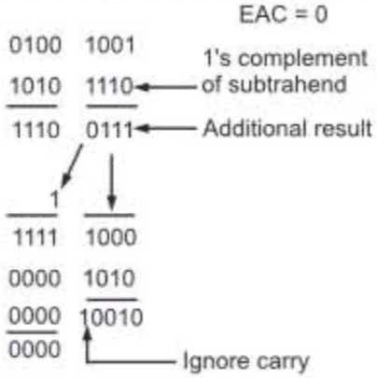
Solution:

- As per rule, first 1's complement of the subtrahend is done. Then the addition is done and the result is checked. Here EAC = 0, so the overall result will be –ve.
- Now see the result of adder 1 from LSB. 1's complement value of 0111 is transferred to adder 2 and it is added with 1010 since no carry is added with it as per the rule. The answer is the final result.

- Now move to the next result of adder 1 i.e. 1110. Here 1 is added to it which is the carry of the previous result. Then it's value is 1's complemented i.e. 0000 and it is added to 0000. Result of adder 2 is the final result. This is the final result of BCD Subtraction.

  $(0100\ 1001)_{BCD} - (0101\ 0001)_{BCD} = (-\ 0000\ 0010)_{BCD}$

- Now you can again check yourself. Decimal equivalent of the given numbers of subtraction is 49 and 51. Therefore 49 – 51 = – 2. So our result is correct.

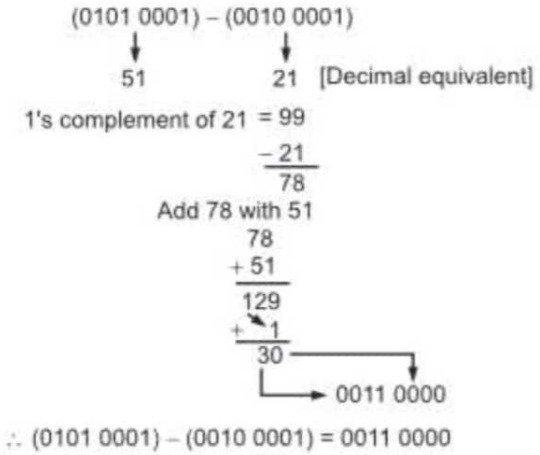


Method 2 of BCD Subtraction:

- In 2nd method, we will do BCD subtraction in **9's complement** method.
  - Here the method is very simple. At first, the decimal equivalent of the given Binary Coded Decimal (BCD) codes are found out.
  - Then the 9's complement of the subtrahend is done and then that result is added to the number from which the subtraction is to be done.
  - If there is any carry bit then the carry bit may be added to the result of the subtraction.

  Idea may be cleared from an example given below:
- Let (0101 0001) – (0010 0001) be the given subtraction.
- As we can see 51 and 21 are the decimal values of the given BCD codes. Then the 9's complement of the subtrahend is done i.e. 99 – 21 = 78.
- This complemented value is added with the 51. i.e. 51 + 78 = 129.
- In this result, the MSB i.e. 1 is the carry. This carry will be added to 29. Therefore 29 + 1 = 30, which is the final answer of **BCD Subtraction**.
- The decimal result will be changed into **BCD** codes to get the result in BCD. Therefore from the example, we can conclude the final result of BCD subtraction i.e.

$$(0101\ 0001)_{BCD} - (0010\ 0001)_{BCD} = (0011\ 0000)_{BCD}$$



∴ (0101 0001) – (0010 0001) = 0011 0000

- Binary Coded Decimal Subtraction using 10's complement is same as in case of 9's complement. Here, the only difference is that, instead of 9's complement we have to do 10's complement of the subtrahend.

3.6 EXCESS-3 (XS-3) ADDER

- To perform Excess-3 additions, we have to
  1. Add two XS-3 code groups.
  2. If carry = 1, add 0011 (3) to the sum of those two code groups.
     If carry = 0, subtract 0011 (3), i.e. add 1101 (13 in decimal) to the sum of those two code groups.

Example: (a) Add 9 and 5 and (b) 4 and 3 in XS-3.

Solution: (a)

	1 1 0 0	9 in XS-3
	+ 1 0 0 0	5 in XS-3
1	0 1 0 0	There is a carry
+ 0 0 1 1	0 0 1 1	and 3 to each group
0 1 0 0	0 1 1 1	14 in XS-3

Result: (1) (4)

Solution: (b)

	0 1 1 1	4 in XS-3
	+ 0 1 1 0	3 in XS-3
	1 1 0 1	There is a carry
	+ 1 1 0 1	and 3 to each group
Ignore carry	1 1 0 1 1	7 in XS-3

Result: (7)

3.7 MULTIPLIER

- A binary multiplier is a combinational logic circuit or digital device used for multiplying two binary numbers. The two numbers are more specifically known as multiplicand and multiplier and the result is known as a product.
- The multiplicand and multiplier can be of various bit size. The product's bit size depends on the bit size of the multiplicand and multiplier. The bit size of the product is equal to the sum of the bit size of multiplier and multiplicand.

- Binary multiplication method is same as decimal multiplication. Binary multiplication of more than 1-bit numbers contains 2 steps. (i) single bit-wise multiplication known as partial product and (ii) add all partial products into a single product.
- Partial products or single bit products can be obtained by using AND gates. However, to add these partial products we need full adders and half adders.
- The schematic design of a digital multiplier differs with bit size. The design becomes complex with the increase in bit size of the multiplier.
- Multipliers are most commonly used in various applications especially in the field of digital signal processing to perform the various algorithms.

Types of Binary Multipliers:

(a) **2 × 2 Bit Multiplier:** This multiplier can multiply two numbers having bit size = 2 i.e. the multiplier and multiplicand can be of 2 bits. The product bit size will be the sum of the bit size of the input i.e. 2 + 2 = 4. The maximum range of its output is $3 \times 3 = 9$. So we can accommodate decimal 9 in 4 bits. It is another way of finding the bit size of the product.

(b) **3 × 3 Bit Multiplier:** This multiplier can multiply two numbers having a maximum bit size of 3 bits. The bit size of the product will be 6. The maximum range of its product is $7 \times 7 = 49$. It can be accommodated in 6 bits which is the size of its output product.

(c) **4 × 4 Bit Multiplier:** This multiplier can multiply a binary number of 4-bit size and gives a product of 8-bit size because the bit size of the product is equal to the sum of bit size of multiplier and multiplicand. The maximum number it can calculate us $15 \times 15 = 225$. You can also evaluate the number of bits from the maximum output range.

3.8 COMPARATOR

- The Digital Comparator is another very useful combinational logic circuit used to compare the value of two binary digits.
- Digital or Binary Comparators are made up from standard AND, NOR and NOT gates that compare the digital signals present at their input terminals and produce an output depending upon the condition of those inputs.
- For example, along with being able to add and subtract binary numbers we need to be able to compare them and determine whether the value of input A is greater than, smaller than or equal to the value at input B etc. The digital comparator accomplishes this using several logic gates that operate on the principles of Boolean Algebra.
- A comparator will evaluate two binary strings and output a 1 if the two strings are exactly the same.
- The Exclusive-NOR (Equality gate) is used to perform the comparison.
- One Exclusive-NOR is used per pair of Binary bits and the outputs of all Exclusive-NORS are ANDed together.

- There are two main types of Digital Comparator available and these are.
  - **(a) Identity Comparator:** an Identity Comparator is a digital comparator with only one output terminal for when A = B, either A = B = 1 (HIGH) or A = B = 0 (LOW). It is also called as Equality Comparator.
  - **(b) Magnitude Comparator:** A Magnitude Comparator is a digital comparator which has three output terminals, one each for equality, A = B greater than, A > B and less than A < B

Following comparators shows different bit comparing configurations:

1. **1-bit Digital Comparator Circuit:** This comparator used to compare two bits.

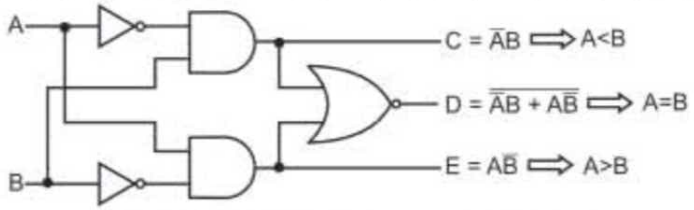


Fig. 3.15: 1-bit Digital Comparator Circuit

2. **4-bit Magnitude Comparator:** Used to compare two 4-bit words.

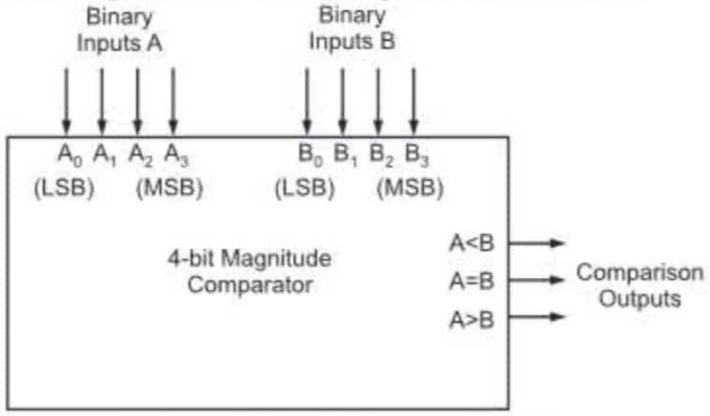


Fig. 3.16: 4-bit Magnitude Comparator

- Some commercially available digital comparators such as the TTL 74LS85 or CMOS 4063 4-bit magnitude comparator have additional input terminals that allow more individual comparators to be "cascaded" together to compare words larger than 4-bits with magnitude comparators of "n"-bits being produced. These cascading inputs are connected directly to the corresponding outputs of the previous comparator.

3. **8-bit Word Comparator:** Compares the two 8-bit numbers by cascading of two 4-bit comparators.

- To save time the comparator starts by comparing the highest-order bit (MSB) first. If equality exists, A = B then it compares the next lowest bit and so on until it reaches the lowest-order bit, (LSB). If equality still exists then the two numbers are defined as being equal.

- If inequality is found, either A > B or A < B the relationship between the two numbers is determined and the comparison between any additional lower order bits stops. Digital Comparator is used widely in Analogue-to-Digital converters, (ADC) and Arithmetic Logic Units, (ALU) to perform a variety of arithmetic operations.

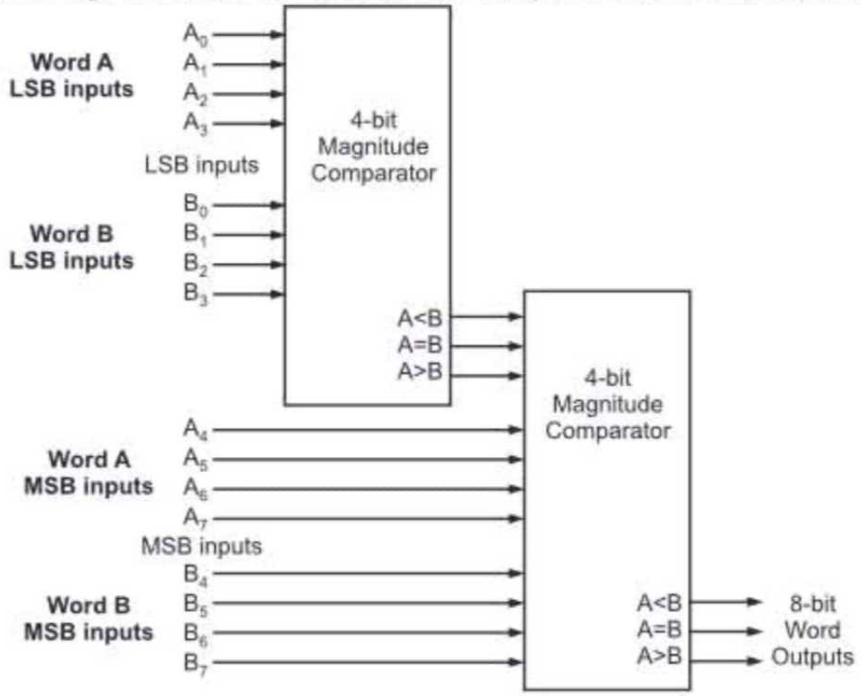


Fig. 3.17: 8-bit Word Comparator

Uses:

1. In address decoding circuits in computers.
2. Process controllers.
3. Servo-motor control.
4. In password verification and biometric applications.

3.9 MULTIPLEXER [W-17]

- Multiplexer is a special type of combinational circuit. Multiplex means 'many into one'.
- The multiplexer is a digital circuit which has many input lines and one output line. The function of the multiplexer is to select one of the input lines and connect it to the output.
- The multiplexer is also known as data selector. The selection of desired input is done by means of selection lines.
- Generally there are 2^m input lines and n selection lines whose bit combinations determine which input is to be selected. The functional block diagram for multiplexer (abbreviated as MUX) is shown in Fig. 3.18.

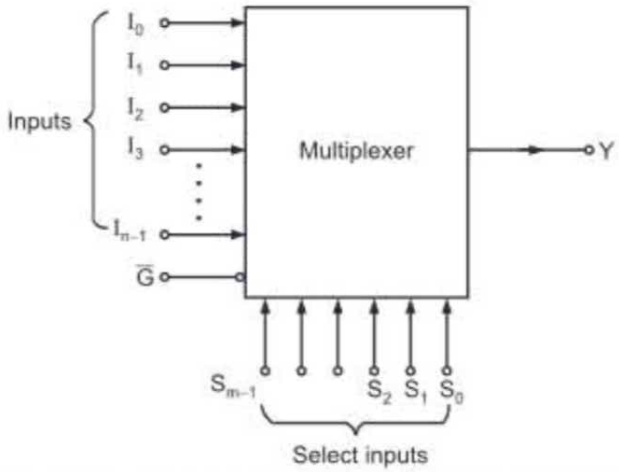


Fig. 3.18: Functional Block Diagram of a Multiplexer

- Generally, a strobe (or enable) input is incorporated which helps in cascading and it is generally active low which means it performs its intended operation when it is low.
- Depending on the number of data inputs, the multiplexers are classified as under:
  1. 2 lines to 1 line (2:1) multiplexer.
  2. 4 lines to 1 line (4:1) multiplexer.
  3. 8 lines to 1 line (8:1) multiplexer.
  4. 16 lines to 1 line (16:1) multiplexer.
- Use of multiplexers offers the following advantages:
  1. Simplification of logic expression is not required.
  2. It minimizes the IC package count.
  3. Logic diagram is simplified.

3.9.1 4 : 1 Multiplexer

- A four input multiplexer is shown in Fig. 3.19. There are four inputs I_0, I_1, I_2 and I_3 which are selectively transmitted to output Y depending on select input combinations.
- Here, two select inputs are required as $2^n = 4$, where n is number of select lines i.e., n = 2. The truth table shows various combinations of S_0 and S_1 and selected output.

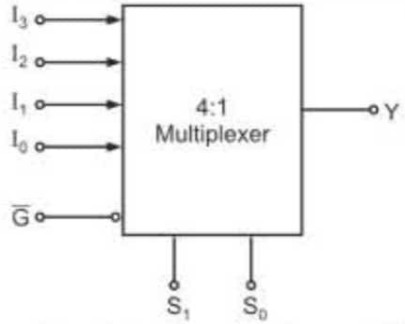


Fig. 3.19 (a): Block Diagram of 4:1 Multiplexer

Truth Table

Select Inputs		Output
S_0	S_1	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

- The Boolean expression for output is,

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

- The logic diagram for 4:1 multiplexer is shown in Fig. 3.19 (b).

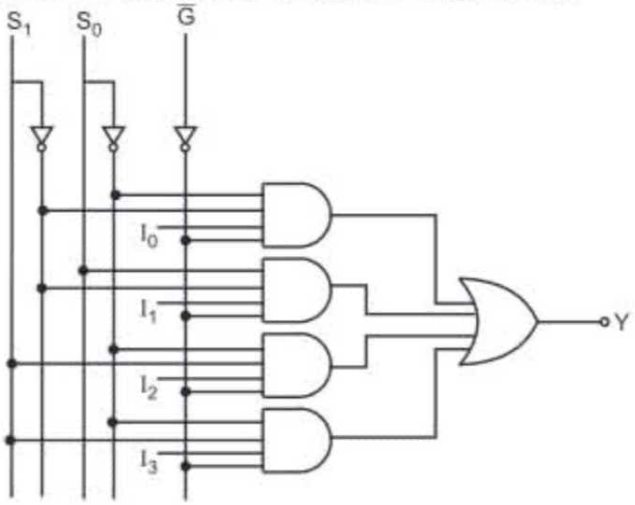


Fig. 3.19 (b): Logic Diagram for 4:1 Multiplexer

3.10 DEMULTIPLEXER

- A demultiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs. It has only one input, n outputs. Demultiplex means 'one into many'.
- The demultiplexer is used to send a single input on one of the output lines.
- The functional block diagram for a demultiplexer is shown in Fig. 3.20.
- It has one input and N outputs. The select input code (bit pattern) determines to which output line the data input will be transmitted.
- In other words, the demultiplexer takes one input data source and selectively distributes it to 1 of N output channels. The number of select lines is n where $2^n = N$.

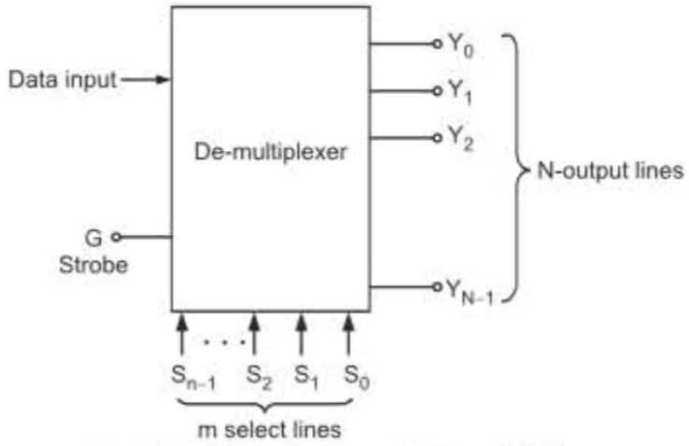


Fig. 3.20: Block Diagram of a Demultiplexer

- The demultiplexers are classified as follows:
  1. 1 line to 2 line (1:2) demultiplexer.
  2. 1 line to 4 line (1:4) demultiplexer.
  3. 1 line to 8 line (1:8) demultiplexer.
  4. 1 line to 16 line (1:16) demultiplexer.

Applications of Multiplexers:

- Multiplexers are used in various applications wherein multiple-data need to be transmitted by using single line.
  1. **Computer Memory:** Multiplexers are used in computer memory to maintain a huge amount of memory in the computers, and also to reduce the number of copper lines required to connect the memory to other parts of the computer.
  2. **Communication System:** A communication system has both a communication network and a transmission system. By using a multiplexer, the efficiency of the communication system can be increased by allowing the transmission of data, such as audio and video data from different channels through single lines or cables.
  3. **Telephone Network:** In telephone networks, multiple audio signals are integrated on a single line of transmission with the help of a multiplexer.
  4. **Data Transmission:** Multiplexer is used to transmit the data signals from the computer system of a spacecraft or a satellite to the ground system by using a GSM satellite.

3.10.1 1:4 Demultiplexer [S-18]

- The block diagram of a 1:4 demultiplexer is as shown in Fig. 3.21. It has only one data input D, two select inputs, one strobe G (or enable E) input and four outputs Y_0, Y_1, Y_2 and Y_3. The strobe G input may be active low (0) or active high (1) and it is used for cascading. But the strobe (G) input is normally active low (0).
- The truth table of a 1:4 demultiplexer is as shown in Table 3.6. From this table it is clear that D is connected to Y_0 when $S_1S_0 = 00$, it is connected to Y_1 when $S_1S_0 = 01$ and so on.

- The other outputs will remain 0. Here $\overline{G} = 0$. The strobe G input needs to be low i.e. $\overline{G} = 0$ in order to enable the demultiplexer.
- If G = 1, then all the outputs will be 0, irrespective of any data input and select inputs.

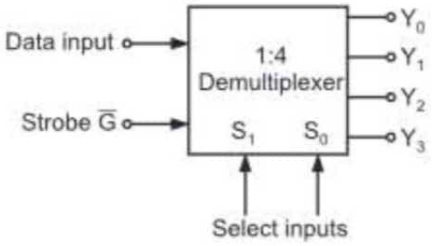


Fig. 3.21 (a): Block Diagram of a 1:4 Demultiplexer

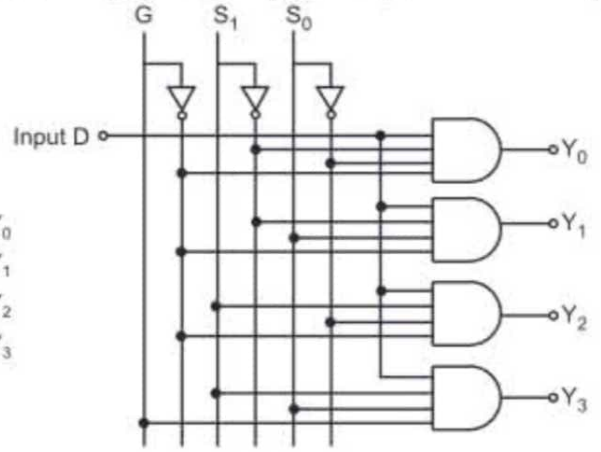


Fig. 3.21 (b): Logic Diagram of a 1:4 Demultiplexer

Truth Table for 1:4 Demultiplexer

Strobe G	Inputs		Outputs			
	S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Applications of Demultiplexer:

- Demultiplexers are used to connect a single source to multiple destinations. These applications include the following:
  1. **Communication System:** Mux and demux both are used in communication system to carry out the process of data transmission. A De-multiplexer receives the output signals from the multiplexer and at the receiver end it converts them back to the original form.
  2. **Arithmetic Logic Unit:** The output of the ALU is fed as an input to the De-multiplexer, and the output of the demultiplexer is connected to a multiple register. The output of the ALU can be stored in multiple registers.
  3. **Serial to Parallel Converter:** This converter is used to reconstruct parallel data. In this technique, serial data is given as an input to the De-multiplexer at a regular interval, and a counter is attached to the demultiplexer at the control input to detect the data signal at the output of the demultiplexer. When all data signals are stored, the output of the demux can be read out in parallel.

Comparison of Multiplexer and Demultiplexer:

Sr. No.	Multiplexer	Demultiplexer
1	It is a combinational logic circuit.	It is a combinational logic circuit.
2.	Multiplexer means 'many into one'.	Demultiplexer means 'one into many'.
3.	It has n number of data inputs and a single output.	It has a single data input and n outputs.
4.	It has 'm' number of select inputs.	It has 'm' number of select inputs.
5.	It accepts many inputs and gives out a single output depending on the status of the select control.	It accepts a single input and distributes it over several outputs depending on the status of select control.
6.	The relation between input/output lines and select lines is $n = 2^m$.	The relation between input/output lines and select lines is $n = 2^m$.
7.	It is used as a data selector.	It is used as a data distributor.

3.11 ALU [S-17, 18, W-17]

- An ALU is a combinational circuit that combines many common logic circuits in one block.
- ALUs can be designed to perform a variety of different arithmetic and logic functions. Possible arithmetic functions include addition, subtraction, multiplication, comparison, increment, decrement, shift, and rotate; possible logic functions include AND, OR, XOR, XNOR, INV, CLR (for clear), and PASS (for passing a value unchanged).
- The inputs to the ALU are the data words to be operated on (called operands), status input information from previous operations, and a opcode from the control unit indicating which operation to perform.
- The basic functionality of an ALU is: it receives 2 operands on a number of bits and a select_operation signal that specifies the operation to perform and then outputs the result to another circuit like a register.

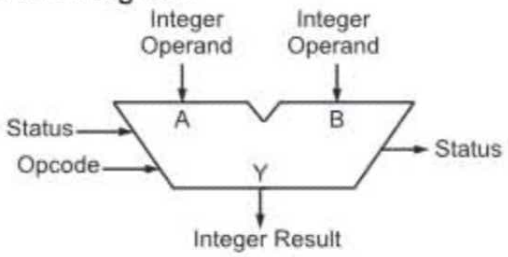


Fig. 3.22: A Symbolic Representation of an ALU

3.12 ENCODER [S-17, W-17]

- Encoder is a combinational circuit which is designed to perform the inverse operation of the decoder.
- An encoder has n number of input lines and m number of output lines. An encoder produces an m bit binary code corresponding to the digital input number.

- The encoder accepts an n input digital word and converts it into an m bit another digital word.
- Fig. 3.23 shows block diagram of encoder.

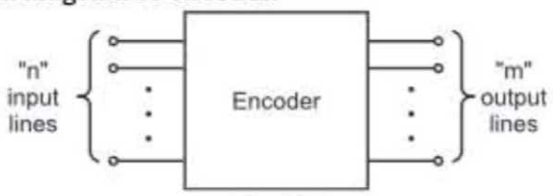


Fig. 3.23

- Examples of Encoders are Priority encoders, Decimal to BCD encoder, Octal to binary encoder, Hexadecimal to binary encoder.

Priority Encoder:

- This is a special type of encoder. Priority is given to the input lines. If two or more input line are 1 at the same time, then the input line with highest priority will be considered.
- There are four input D_0, D_1, D_2, D_3 and two output Y_0, Y_1. Out of the four input D_3 has the highest priority and D_0 has the lowest priority. That means if $D_3 = 1$ then $Y_1\ Y_1 = 11$ irrespective of the other inputs. Similarly if $D_3 = 0$ and $D_2 = 1$ then $Y_1\ Y_0 = 10$ irrespective of the other inputs.
- Fig. 3.24 shows Block diagram of priority encoder.

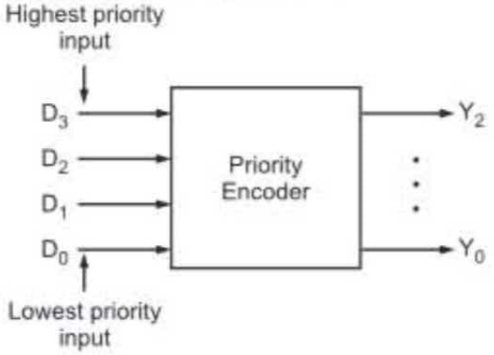


Fig. 3.24 (a) Block diagram of Priority Encoder

Truth Table

Highest	Inputs		Lowest	Outputs	
D_3	D_2	D_1	D_0	Y_0	Y_1
0	0	0	0	×	×
0	0	0	1	0	0
0	0	1	×	0	1
0	1	×	×	1	0
1	×	×	×	1	1

- Fig. 3.24 (b) shows logic circuit of priority encoder.

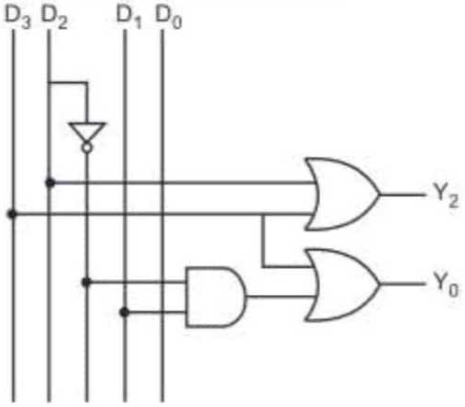


Fig. 3.24 (b)

3.12.1 8-to-3 Priority Encoder

- The priority encoders output corresponds to the currently active input which has the highest priority. So when an input with a higher priority is present, all other inputs with a lower priority will be ignored.
- The priority encoder comes in many different forms with an example of an 8-input priority encoder shown in Fig. 3.25.
- Priority encoders are available in standard IC form and the TTL 74LS148 is an 8-to-3 bit priority encoder which has eight active LOW (logic "0") inputs and provides a 3-bit code of the highest ranked input at its output.

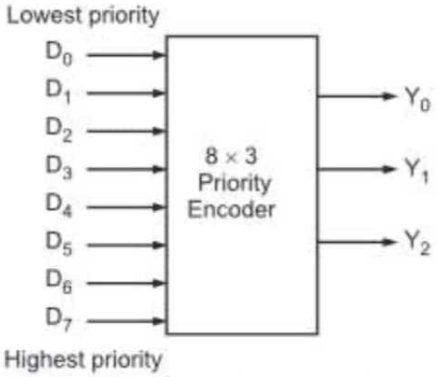


Fig. 3.25: 8 × 3 Priority Encoder

- Priority encoders output the highest order input first for example, if input lines "D_2", "D_3" and "D_5" are applied simultaneously the output code would be for input "D_5" ("101") as this has the highest order out of the 3 inputs. Once input "D_5" had been removed the next highest output code would be for input "D_3" ("011"), and so on.
- The truth table for a 8-to-3 bit priority encoder is given below.

Truth Table

Digital Inputs								Binary Output		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	×	0	0	1
0	0	0	0	0	1	×	×	0	1	0
0	0	0	0	1	×	×	×	0	1	1
0	0	0	1	×	×	×	×	1	0	0
0	0	1	×	×	×	×	×	1	0	1
0	1	×	×	×	×	×	×	1	1	0
1	×	×	×	×	×	×	×	1	1	1

Where, × equals "dont care", that is logic "0" or a logic "1".

- From this truth table, the Boolean expression for the encoder above with data inputs D_0 to D_7 and outputs Q_0, Q_1, Q_2 is given as:

Output Q_0:

$$Q_0 = \Sigma(1, 3, 5, 7)$$

$$Q_0 = \Sigma(\bar{D}_7 \bar{D}_6 \bar{D}_5 \bar{D}_4 \bar{D}_3 \bar{D}_2 \bar{D}_1 + \bar{D}_7 \bar{D}_6 \bar{D}_5 \bar{D}_4 D_3 + \bar{D}_7 \bar{D}_6 D_5 + D_7)$$

$$Q_0 = \Sigma(\bar{D}_6 \bar{D}_4 \bar{D}_2 D_1 + \bar{D}_6 \bar{D}_4 D_3 + \bar{D}_6 D_5 + D_7)$$

$$Q_0 = \Sigma(\bar{D}_6 (\bar{D}_4 \bar{D}_2 D_1 + \bar{D}_4 D_3 + D_5) + D_7)$$

Output Q_1:

$$Q_1 = \Sigma(2, 3, 5, 7)$$

$$Q_1 = \Sigma(\bar{D}_7 \bar{D}_6 \bar{D}_5 \bar{D}_4 \bar{D}_3 D_2 + \bar{D}_7 \bar{D}_6 \bar{D}_5 \bar{D}_4 D_3 + \bar{D}_7 D_6 + D_7)$$

$$Q_1 = \Sigma(\bar{D}_5 \bar{D}_4 D_2 + \bar{D}_5 \bar{D}_4 D_3 + \bar{D}_6 + D_7)$$

$$Q_1 = \Sigma(\bar{D}_5 \bar{D}_4 (D_2 + D_3) + D_6 + D_7)$$

Output Q_2:

$$Q_2 = \Sigma(4, 5, 6, 7)$$

$$Q_2 = \Sigma(\bar{D}_7 \bar{D}_6 \bar{D}_5 D_4 + \bar{D}_7 \bar{D}_6 D_5 + \bar{D}_7 D_6 + D_7)$$

$$Q_2 = \Sigma(D_4 + D_5 + D_6 + D_7)$$

- Then the final Boolean expression for the priority encoder including the zero inputs is defined as:

$$Q_0 = \Sigma(\bar{D}_6 (\bar{D}_4 \bar{D}_2 D_1 + \bar{D}_4 D_3 + D_5) + D_7)$$

$$Q_1 = \Sigma(\bar{D}_5 \bar{D}_4 (D_2 + D_3) + D_6 + D_7)$$

$$Q_2 = \Sigma(D_4 + D_5 + D_6 + D_7)$$

- In practice these zero inputs would be ignored allowing the implementation of the final Boolean expression for the outputs of the 8-to-3 priority encoder. We can constructed a simple encoder from the expression above using individual OR gates as shown in Fig. 3.26.

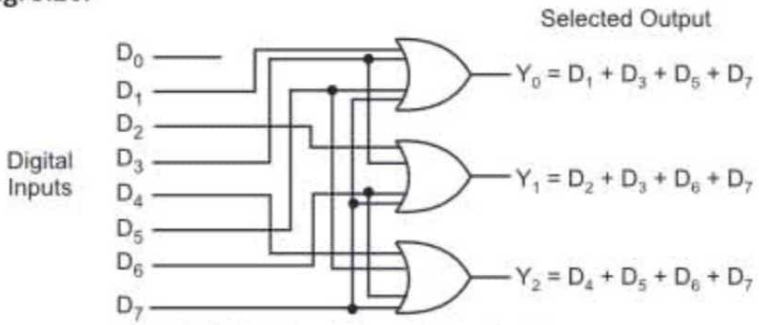


Fig. 3.26: Encoder using Logic Gates

3.13 DECODER

- A decoder is a combinational circuit. It has n input and to a maximum $m = 2^n$ outputs.
- Decoder is identical to a demultiplexer without any data input. It performs operations which are exactly opposite to those of an encoder.

- Fig. 3.27 shows block diagram of decoder.

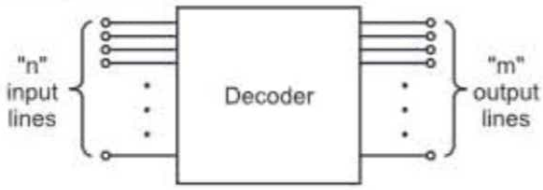


Fig. 3.27: Block diagram of Decoder

- Examples of Decoders are Code converters, BCD to seven segment decoders, Nixie tube decoders and Relay actuator.
- **Binary Decoders:** A combinational circuit that converts binary information from n coded inputs to a maximum 2^n coded outputs is called n-to-2^n decoder, more generally n-to-m decoder, $m \leq 2^n$ [17].
- **2-to-4 Decoder:** In a 2-to 4 decoder, 2 inputs, A0, A1 are decoded into $2^2 = 4$ outputs, D0 through D3. Each output represents one of the minterms of the 2 input variables.

Truth Table

X	Y	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Symbol:

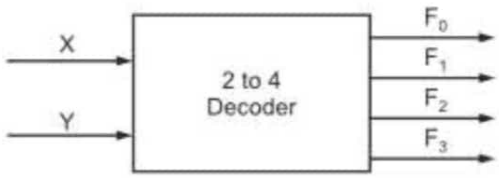


Fig. 3.28: Truth table and symbol of 2 to 4 decoder

Summary

- Combinational logic circuits are designed by combining different type of logic gates to produce a specific output for all possible input combinations.
- The code converters are used to convert the information into the code which we want. These are basically encoders and decoders which converts the data into an encoded form.
- An arithmetic circuit is a set of gates with a separate set of inputs for each number that has to be processed. The gates are connected so as to carry out an arithmetic action and the outputs of the gate circuit are the digits of the result (addition, subtraction, multiplication, or division).
- A two input logic gate is required to accomplish the addition of two binary numbers. The exclusive-OR gate is used to achieve binary addition which is slightly different from basic OR gate.

- A logic circuit block used for adding two one bit numbers or simply two bits is called as a half adder circuit. This circuit has two inputs which accept the two bits and two outputs, with one producing sum output and other produce carry output.
- A binary full adder is a multiple output combinational logic network that performs the arithmetic sum of three input bits. It consists of three inputs, in which two are input variables represent the two significant bits to be added, labeled as A and B, whereas the third input terminal is the carry from the previous lower significant position and labeled as Cin.
- A parallel adder is used for adding all bits of the two numbers simultaneously.
- A half subtractor is a multiple output combinational logic network that does the subtraction of two bits of binary data. It has input variables and two output variables. Two inputs are corresponding to two input bits and two output variables correspond to the difference bit and borrow bit.
- A combinational logic circuit performs a subtraction between the two binary bits by considering borrow of the lower significant stage is called as the full subtractor.
- Digital or Binary Comparators are made up from standard AND, NOR and NOT gates that compare the digital signals present at their input terminals and produce an output depending upon the condition of those inputs.
- A binary multiplier is an electronic circuit used in digital electronics, such as a computer, to multiply two binary numbers. It is built using binary adders.
- A multiplexer is a circuit that accepts many input but give only one output. A demultiplexer function exactly in the reverse of a multiplexer, that is a demultiplexer accepts only one input and gives many outputs.
- Encoder circuit converts human understandable language to machine understandable language.
- Encoders are combinational logic circuits and they are exactly opposite of decoders. They accept one or more inputs and generate a multibit output code.
- Decoder circuit converts machine understandable language to human understandable language.
- A binary decoder is a combinational logic circuit that converts binary information from the n coded inputs to a maximum of 2^n unique outputs.

Practice Questions

Q.1 Answer the following Question in brief:

1. What is meant by combinational logic circuits?
2. Describe full-adder with symbol and truth table.
3. Explain full subtractor.
4. What is multiplexer and demultiplexer?
5. Explain encode and decoder in short.
6. Compare multiplexer and demultiplexer.

Q.2 Answer the following Question:

1. Explain half-adder or with diagram.
2. Write short note on: Full adder.
3. With the help of diagram describe 3-to-8 decoder.
4. With the help of diagram 8-to-3 encoder.
5. Write short notes on: (i) 4:1 multiplexer (ii) 1:4 demultiplexer.
6. Explain the block diagram of combinational circuits.

Q.3 Define Terms:

Encoder, Multiplexer, Decoder, Demultiplexer, Combinational circuit, Full-Adder, Half-Adder, Half-subtractor, Binary Multiplier, Truth Table.

Check Your Understanding

1. Full Adder combinational circuits has 3 inputs and _____.
   (a) 2 outputs (b) 1 output
   (c) 3 outputs (d) None
2. Half adder circuits requires two binary _____.
   (a) Inputs (b) Outputs
   (c) Digits (d) Both (a) and (b)
3. Circuits that employs memory elements in addition to gates is called _____.
   (a) combinational circuit (b) sequential circuit
   (c) combinational sequence (d) series
4. Two bit addition is done by _____.
   (a) ripple carry adder (b) carry sum adder
   (c) full adder (d) half adder
5. A multiplexer is also called as a _____.
   (a) Coder (b) parallel adder
   (c) Data selector (d) NOR gate

Answers

1. (d)	2. (d)	3. (b)	4. (d)	5. (c)

Previous Exam Questions

Summer 2017

1. _____ subtracts two bits along with the borrow generated from previous bit position. **(1 M)**
   (a) Half adder (b) Full subtractor
   (c) Half adder (d) Full adder

Ans. Refer to Section 3.3.2.

2. ALU is _____ **(1 M)**
   (a) Adding And logic unit (b) Algorithm and Logical unit
   (c) Arithmetic and logical unit (d) Arithmetic and lexical unit

Ans. Refer to Section 3.11.

3. What is an encoder? **(1 M)**

Ans. Refer to Section 3.12.

4. Write a short note on half subtractor. **(5 M)**

Ans. Refer to Section 3.3.2.

5. With a neat diagram explain the working of 1-bit ALU. **(4 M)**

Ans. Refer to Section 3.11.

Winter 2017

1. Multiplexer is an example of _____ **(1 M)**

(a) Sequential circuit (b) Logic circuit

(c) Combinational circuit (d) Both (a) and (b)

Ans. Refer to Section 3.9.

2. _____ is responsible for performing various arithmetic and logical or shift operations

(a) CPU (b) ALU

(c) Monitor (d) None of the above **(1 M)**

Ans. Refer to Section 3.11.

3. What is combinational circuit? **(1 M)**

Ans. Refer to Section 3.1.

4. Write a short note on Half adder. **(5 M)**

Ans. Refer to Section 3.3.1.1.

5. Explain BCD encoder with its diagram. **(4 M)**

Ans. Refer to Section 3.12.

Summer 2018

1. Which combinational circuit is used for addition of two bits? **(1 M)**

(a) Half adder (b) Multiplexer

(c) Encoder (d) Subtractor

Ans. Refer to Section 3.3.

2. Which combinational circuit is used for subtraction of two bits? **(1 M)**

(a) Half adder (b) Multiplexer

(c) Encoder (d) Subtractor

Ans. Refer to Section 3.3.

3. Define the term: ALU,Encoder **(2 M)**

Ans. Refer to Section 3.11.

4. Explain 1 to 4 demultiplexer with circuit. Write applications of demultiplexer. **(5 M)**

Ans. Refer to Section 3.10.1.

5. How to implement full adder using two half adders. Draw logic diagram and its truth table. **(4 M)**

Ans. Refer to Section 3.3.1.2.

■■■

4... Sequential Circuits

Objectives...

- To understand concept of sequential circuits such as Flip-flops and its types.
- To learn about counters and its types.
- To know about parallel and shift registers.

4.1 INTRODUCTION

- Sequential circuits operate in steps. They consist of combinational circuits and registers; the latter represent a state. All registers use the same control signal - the clock - which determines when a new state is assumed. Hence, the sequential circuit is synchronous and it is called a State Machine.
- Similar to the class of combinational circuits, there exist frequently used patterns of sequential circuits. If the combinational component is an incrementer, the state machine is a counter. If the combinational part is a multiplexer with inputs from "neighbouring' elements, the result is a shift register.
- A circuit in which the output/outputs depend on both the present state and past state of input is called as sequential circuit.

4.2 TERMINOLOGIES [W-17]

- Following are terminologies used for the study of sequential circuit:
  1. **Combinational logic:** The *combinational logic* is the process of combining *logic* gates to process the given two or more inputs such that to generate at least one output signal based on the *logic* function of each *logic* gate.
  2. **Latch:** A latch is a device with exactly two stable states i.e. high-output and low-output. It has a feedback path to retain the information. Hence, latches can be memory devices and can store one bit of data.
  3. **Filp-flops**: A flip flop is an electronic circuit with two stable states that can be used to store binary data. The stored data can be changed by applying varying inputs. Flip-flops are the basic building block of a sequential circuit.
  4. **Timing analysis:** Timing analysis is the methodical analysis of a digital circuit to determine if the timing constraints imposed by components or interfaces are met.
  5. **Building blocks:** The basic *building blocks* of a *digital* circuit are '*logic* gates'. *Logic* gates are *electronic circuits* made by using various active and passive elements in.

6. **Sequential Logic:** If the present value of the output of a circuit at any instant of time is dependent not only on the present value of that inputs at that time but also on the previous value of inputs and outputs of the circuit, then the circuit is known as a Sequential Logic Circuit.

- Sequential circuits have a memory unit which stores the past state of inputs.
- Fig. 4.1 shows block diagram of sequential circuit.

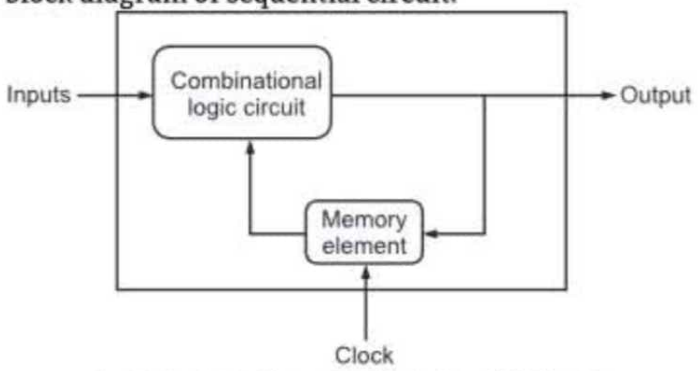


Fig. 4.1: Block diagram of Sequential Circuit

- The few examples of sequential logic circuits are as: Flip-flops, Shift registers, Counters, Calculator, Dial combinational lock, Telephone system and so on.
- The combinational circuit does not use any memory. Hence, the previous state of input does not have any effect on the present state of the circuit. But sequential circuit has memory so output can vary based on input, (See Fig. 4.1).
- The difference between the Combinational Logic System and Sequential Logic System is given in Table 4.1.

Table 4.1

Sr. No.	Combinational Logic System	Sequential Logic System
1.	The output of the system at a given instance can be determined from the knowledge of present input status.	The output of the system at a given instance cannot be determined from the knowledge of present input status.
2.	It does not require the past history of inputs to know the output of a system.	It requires the past history of inputs to know the output of a system.
3.	It has no memory capability.	It has memory capability.
4.	It does not require feedback.	It requires at least one feedback path from the output of a system.
5.	It does not require time sequence for operation of a system.	It follows a time sequence in addition to a proper combination of inputs for operation of a system.
6.	It requires more hardware for the system design but saves time.	It requires less hardware for the system design but takes longer time.
7.	The few examples are multiplexers, demultiplexers, encoders, decoders, comparators etc.	The few examples are counters, shift registers, telephone system, combination lock, digital computer etc.

- The sequential circuits can be classified depending on the timing of their signals: Synchronous sequential circuits and Asynchronous sequential circuits.
- In synchronous sequential circuits, signals can affect the memory elements only at discrete instants of time.
- In asynchronous sequential circuits change in input signals can affect memory element at any instant of time. The memory elements used in both circuits are flip-flops which are capable of storing 1-bit binary information.

Table 4.2

Sr. No.	Synchronous Sequential Circuits	Asynchronous Sequential Circuits
1.	In synchronous circuits, memory elements are clocked flip-flops.	In asynchronous circuits, memory elements are either unclocked flip-flops, time delay elements.
2.	In synchronous circuits, the change in Input signals can affect memory element upon activation of clock signal.	In asynchronous circuits change in Input signals can affect memory element at any instant of time.
3.	The maximum operating speed of clock depends on time delays Involved.	Because of absence of clock, asynchronous circuits can operate faster than synchronous circuits.
4.	Easier to design	More difficult to design.

4.3 FLIP-FLOPS [S-17, 18; W-17]

- The most important memory element is the Flip-Flop (FF), which is made up of an assembly of logic gates. Thus, a flip-flop is a binary cell (or bistable electronic circuit) which can store a bit of information. Flip-flop is a 1-bit memory cell.
- Fig. 4.2 shows the general type of symbol for a flip-flop. It has two outputs, normal output Q and inverted output $\overline{Q}$. The state of the flip-flop always refers to the state of the normal output Q. The inverted output $\overline{Q}$ is the complement of the normal output Q.

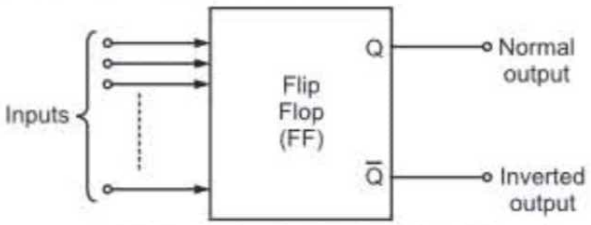


Fig. 4.2: General Flip-Flop Symbol

- A flip-flop is said to be in high state or logic '1' or Set (S) state, when Q = 1 and in low state or logic '0' state or Reset (R) state or clear state, when Q = 0.
- A flip-flop is a synchronous version of the latch.

4.3.1 S-R Flip-Flop

- The S-R (Set-Reset) flip flop, also known as a SR Latch, can be considered as one of the most basic sequential logic circuit. This is simple flip-flop is basically a one-bit memory bi-stable device.
- A basic NAND gate SR flip-flop circuit provides feedback from both of its outputs back to its opposing inputs and is commonly used in memory circuits to store a single data bit.

- S-R flip flop consists of two inputs, one called the Set, (S) and the other called the Reset, (R) with two corresponding outputs Q and its inverse or complement $\overline{Q}$ as shown in Fig. 4.3 (a).
- The modified circuit of NAND SR latch is shown in Fig. 4.3 (b).

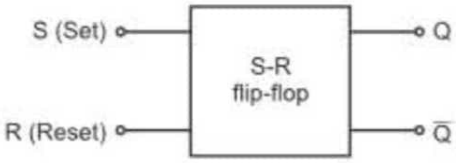


Fig. 4.3 (a): Block diagram

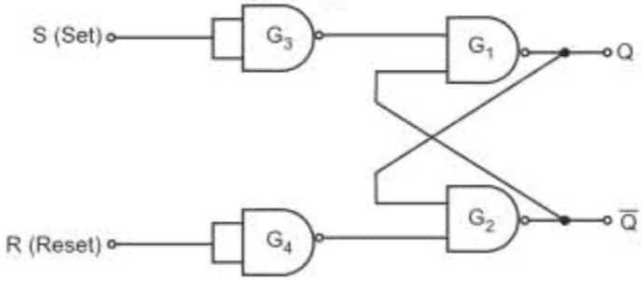


Fig. 4.3 (b): SR Latch using NAND Gate

- Following table shows truth table for S-R flip-flop.

Table 4.3: S-R flip flop Truth Table

Inputs		Outputs		Mode of operation
S	R	Q	$\overline{Q}$	
0	0	Q_n	$\overline{Q}_n$	Hold or unchanged state
0	1	0	1	Reset
1	0	1	0	Set
1	1	1	1	Prohibit

4.3.1.2 Clocked SR Flip-Flop

- It is often required to set or reset the memory cell in synchronism with a train of pulses known as clock. Such a circuit is shown in Fig. 4.4 (a) and is referred to as a clocked Set-Reset (SR) flip-flop.
- In this circuit, if a clock pulse is present (clk = 1), its operation is exactly the same as that of Fig. 4.4.
- On the other hand, when the clock pulse is not present (clk = 0), the gates G_3 and G_4 are inhibited i.e. their outputs are 1 irrespective of the values of S or R.
- In other words, the circuit responds to the inputs S and R only when the clock is present.

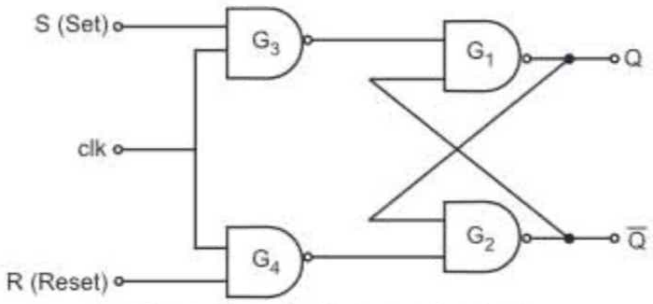


Fig. 4.4 (a): Clocked S-R Flip-Flop

Table 4.4: Truth Table of SR Flip-Flop

clk	Inputs		Output
	S	R	Q_{n+1}
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	?

- When the clock is present, the truth table of clocked S-R flip-flop is same as that of S-R flip-flop.
- The logic symbol of clocked SR flip-flop is shown in Fig. 4.4 (b).

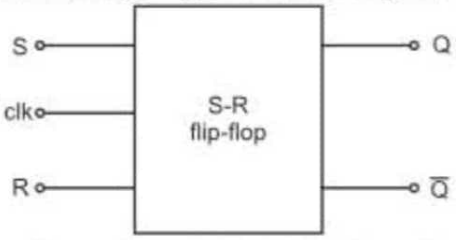


Fig. 4.4 (b): Logic Symbol of clocked S-R Flip-Flop

Clocked SR flip-flop with Preset and Clear:

- In the flip-flop when the power is Switched ON, the state of the circuit is uncertain.
- It may come to set ($Q = 1$) or reset ($\overline{Q} = 0$) state.
- In many applications it is desired to initially set or reset the flip-flop i.e. the initial state of the flip-flop is to be assigned. This is accomplished by using preset (P_r) and clear (C_r) inputs.
- These inputs may be applied at any time between clock pulses and are not in synchronism with the clock. An SR flip-flop with preset and clear is shown in Fig. 4.5 (a).

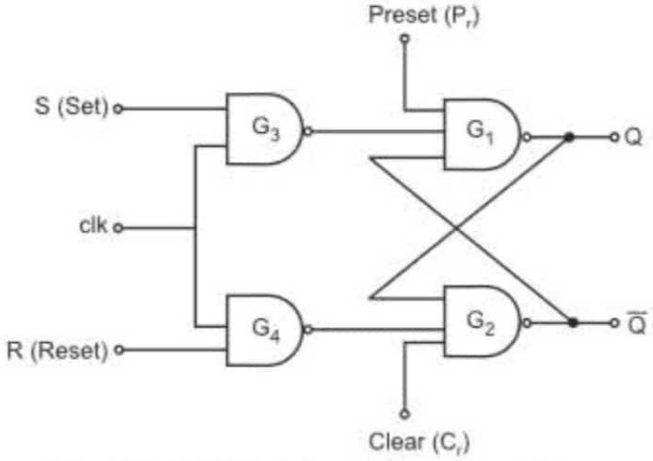


Fig. 4.5 (a): SR Flip-Flop with Preset and Clear

- If $P_r = C_r = 1$ the circuit operates in accordance with the truth table of SR flip-flop given in Table 4.5. If $P_r = 0$ and $C_r = 1$, the output of G_1 (Q) will certainly be 1.
- Consequently all the three inputs to G_2 will be 1 which will make $\overline{Q} = 0$. Hence, making $P_r = 0$ sets the flip-flop. Similarly if $P_r = 1$ and $C_r = 0$ then the flip-flop is reset.
- The condition $P_r = C_r = 0$ must not be used, since this leads to an uncertain state.

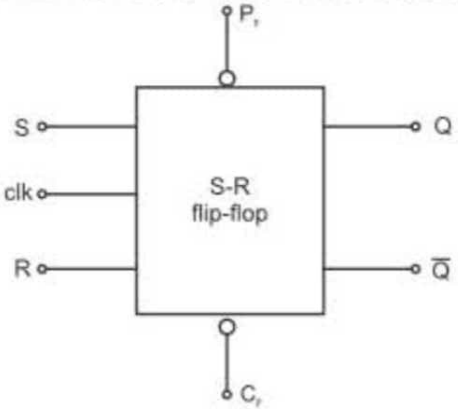


Fig. 4.5 (b): Logic Symbol of Clocked SR Flip-Flop with Preset and Clear

- In the logic symbol, bubbles are used for P_r and C_r inputs which means these are active low.

Table 4.5: Truth Table

clk	Inputs		Output	Operation
	C_r	P_r	Q	performed
1	1	1	Q_{n+1}	Normal flip-flop
0	0	1	0	Clear
0	1	0	1	Preset

4.3.2 JK (Jack-Kilby) Flip-Flop

- The name JK flip-flop is termed from the inventor Jack Kilby from texas instruments. Due to its versatility they are available as IC packages.
- The major applications of JK flip-flop are Shift registers, storage registers, counters and control circuits.
- Inspite of the simple wiring of D type flip-flop, JK flip-flop has a toggling nature. This has been an added advantage. Hence they are mostly used in counters and PWM generation, etc. Here we are using NAND gates for demonstrating the JK flip flop.
- The basic JK Flip Flop has J,K inputs and a clock input and outputs Q and $\overline{Q}$ (the inverse of Q). Optionally it may also include the PR (Preset) and CLR (Clear) control inputs.
- The uncertainty in the state of an S-R flip-flop when $S_n = R_n = 1$ can be eliminated by converting it into a JK flip-flop. The data inputs are J and K, which are ANDed with $\overline{Q}$ and Q respectively, to obtain S and R inputs i.e.

$$S = J \cdot \overline{Q}$$
$$R = K \cdot Q$$

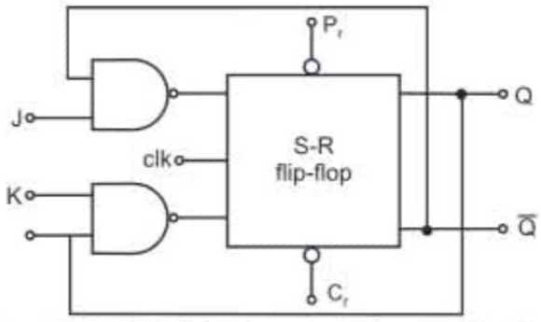


Fig. 4.6: SR Flip-Flop Converted into JK Flip-Flop

- A JK flip-flop thus obtained is shown in Fig. 4.6. Its truth table is given in Table 4.6 (a), which is reduced to Table 4.6 (b) for convenience. The table has been prepared for all the possible combinations of J and K inputs, and for each combination both the states of the output have been considered.

Table 4.6 (a): Truth Table for Fig. 4.14

Inputs		Outputs		Inputs to S-R FF		Output
J_n	K_n	Q_n	$\overline{Q}_n$	S_n	R_n	Q_{n+1}
0	0	0	1	0	0	0
0	0	1	0	0	0	1 } $= Q_n$
0	1	0	1	0	0	0
0	1	1	0	0	1	0 } $= 0$
1	0	0	1	1	0	1
1	0	1	0	0	0	1 } $= 1$
1	1	0	1	1	0	1
1	1	1	0	0	1	0 } $= \overline{Q}_n$

Table 4.6 (b): Truth Table of JK Flip-Flop

Inputs		Output
J_n	K_n	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

- We obtain JK flip-flop using NAND gates as shown in Fig. 4.7 (a) and the logic symbol of J-K flip-flop is shown in Fig. 4.7 (b).

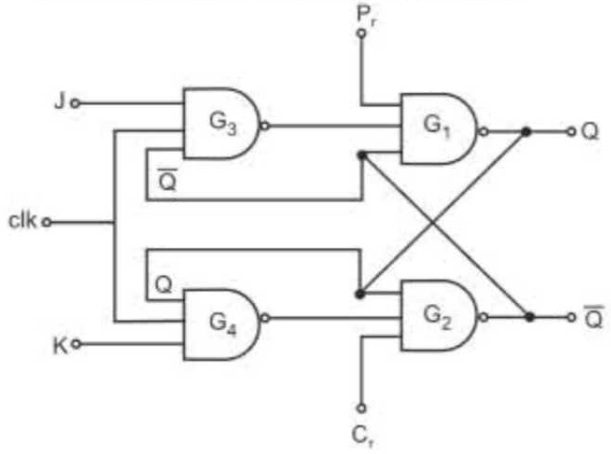


Fig. 4.7 (a): JK Flip-Flop using NAND Gates

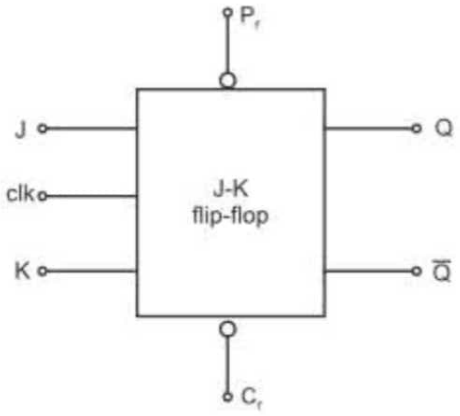


Fig. 4.7 (b): Logic symbol of JK flip-flop

4.3.3 D Flip-Flop [S-18]

- The SR and JK flip-flop can be easily converted to D flip-flop by simply addition of inverter as shown in Fig. 4.8 (a).
- The symbol of negative edge triggered D flip-flop is shown in Fig. 4.8 (b).

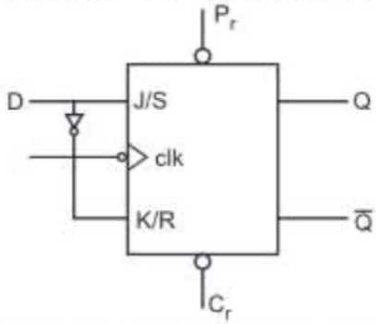


Fig. 4.8 (a): JK SR Flip-Flop converted into a D Flip-Flop

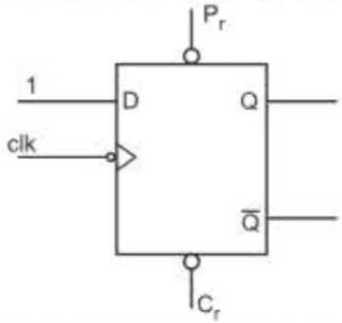


Fig. 4.8 (b): Logic Symbol of Negative Edge Triggered D Flip-Flop

- This flip-flop has only one input i.e., D input. The output Q will go to same state that is present on the D input when negative edge of clock occurs. Its truth table is given in Table 4.7.

Table 4.7: Truth Table for a D-type Flip-Flop

clk	Input	Output
	D_n	Q_{n+1}
↓	0	0
↓	1	1

- The output Q_{n+1} at the end of the clock pulse equals the input D_n. Hence, we can say that the input data appears at the output at the end of the clock pulse.
- Thus, the transfer of data from the input to the output is delayed and hence the name delay (D) flip-flop. The D-type flip-flop is either used as delay device or as a latch to store 1-bit of binary information.

4.3.3.1 Race Around Condition

- If J = K = 1 and Q = 0 and a pulse is applied at the clock input.
- After a time interval Δt equal to the propagation delay through two NAND gates in series, the output will change to Q = 1. Now we have J = K = 1 and Q = 1 and after another time interval of Δt the output will change back to Q = 0. Hence for the duration t_p of the clock pulse, the output will oscillate back and forth between 0 and 1.
- At the end of clock pulse, the value of Q is uncertain. This situation is referred to as the Race Around Condition.
- The race-around condition can be avoided if $t_p < \Delta t < T$. However, it may be difficult to satisfy the inequality because of very small propagation delays in it.
- A more practical method for overcoming this difficulty is the use of the **Master-Slave (M-S) Configuration**.

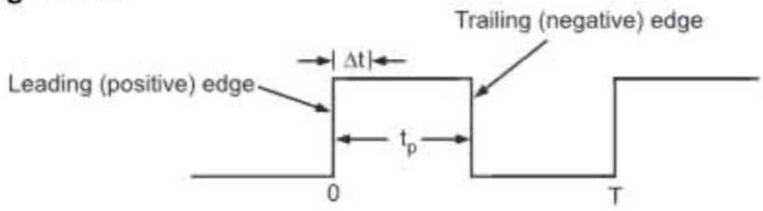


Fig. 4.9: A clock pulse

4.3.4 Master-Slave JK Flip-Flop

- A master-slave JK flip-flop is a cascade of two JK flip-flops, with feedback from the output of the second to the inputs of the first. Positive clock pulses are applied to the first flip-flop and clock pulses are inverted before these are applied to the second flip-flop.
- When clk = 1 the first flip-flop is enabled and the outputs Q_m and $\bar{Q}_m$ respond to the inputs J and K.
- At the same time, the second flip-flop is inhibited. When clk = 0, the second flip-flop is enabled and the first flip-flop is inhibited. Therefore the outputs Q and $\bar{Q}$ follow the outputs Q_M and $\bar{Q}_M$.
- Since the second flip-flop simply follows the first one, it is referred to as the slave and the first one as the Master. Hence the configuration is referred as Master-Slave (M-S) flip-flop.

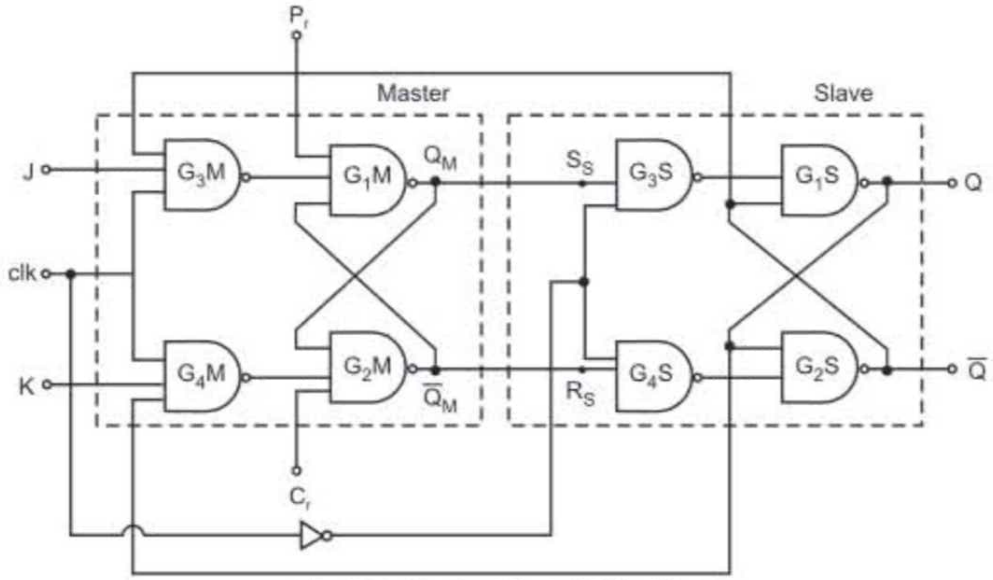


Fig. 4.10: Master-Slave JK flip-flop

- The logic symbol of a M-S J-K flip-flop is shown in Fig. 4.11.
- In this circuit, the inputs to the gates G_3M and G_4M do not change during the clock pulse, therefore the race-around condition does not exist. The state of the master-slave flip-flop changes at the negative transition of the clock pulse.

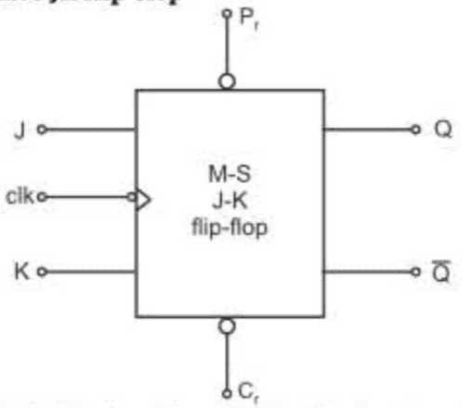


Fig. 4.11: Master-Slave JK flip-flop logic symbol

4.3.5 T Flip-Flop [S-17, 18]

- In a J-K flip-flop, if J = K, the resulting flip-flop is referred as a T-type flip-flop and is as shown in Fig. 4.12 (a). Its logic symbol is shown in Fig. 4.12 (b).

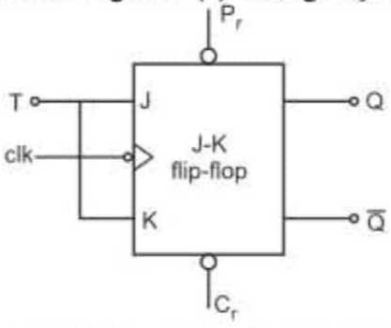


Fig. 4.12 (a): JK Flip-Flop converted into T-Type

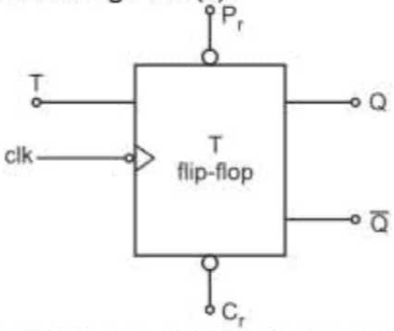


Fig. 4.12 (b): Logic Symbol of Negative Edge Triggered T Flip-Flop

- It has only one input, referred to as T-input. Its truth table is given in Table 4.8.

Table 4.8: Truth Table of a T-type Flip-Flop

clk	Input	Output
	T_n	Q_{n+1}
↓	0	Q_n
↓	1	$\overline{Q}_n$

- From above table it is clear that if T = 1, it acts as a toggle switch. For every clock pulse, the output Q changes.
- An S-R flip-flop cannot be converted into T-type flip-flop, since S = R = 1 is not allowed.

Difference between D flip-flop and T flip-flop:

- The comparison of D flip-flop and T flip-flop is as given in Table 4.9.

Table 4.9

Sr. No.	D Flip Flop	T Flip Flop
1.	It is a delay type flip-flop.	It is a toggle type flip-flop.
2	It is available commercially.	It is not available commercially.
3.	It can be constructed from SR and JK flip flops.	It cannot be constructed from SR flip-flop but from J-K flip-flop.
4.	Its J and K inputs are connected together through NOT gate.	Its J and K inputs are directly connected to each other.
5.	It is used as a delay device.	It can be used as a delay device.

Difference between Edge Triggering and Level Triggering:

Sr. No.	Edge Triggering	Level Triggering
1.	Type of triggering that allows a circuit to become active at the positive edge or the negative edge of the clock signal.	Type of triggering that allows a circuit to become active when the clock pulse is on a particular level.
2.	An event occurs at the rising edge or falling edge.	An event occurs during the high voltage level or low voltage level.
3.	Flip-flops are edge triggered.	Latches are level triggered.

4.3.6 Flip Flop Conversion

- The main purpose of the flip flop conversion is to convert a flip flop into a desired type-B flip flop using some conversion logic. The flip flop conversions are classified into different types that are:
  - SR-FF to JK-FF Conversion
  - JK-FF to SR-FF Conversion
  - SR-FF to D-FF Conversion
  - D-FF to SR-FF Conversion
  - JK-FF to T-FF Conversion
  - JK-FF to D-FF Conversion
  - D-FF to JK-FF Conversion

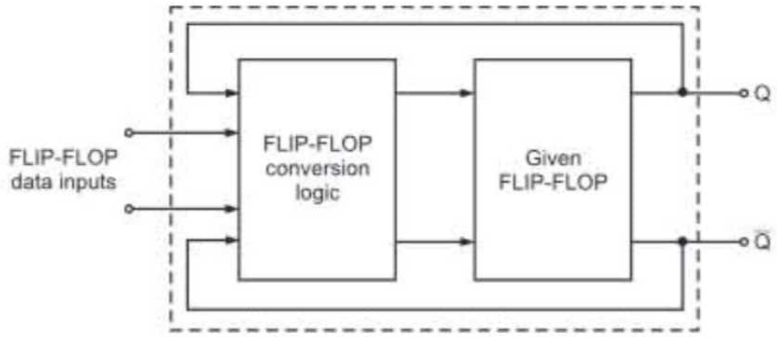


Fig. 4.13: The General Model Used to Convert One Type of Flip-Flop to another type

- To design the conversion logic we need to combine the excitation tables for both Flip-Flops and make a truth table with data input(s) and Q as the input(s) and the input(s) of the given Flip-Flops as the output(s).

4.4 APPLICATIONS OF FLIP-FLOPS [S-17]

- Application of the flip flop circuit mainly involves in bounce elimination switch, data storage, data transfer, latch, registers, counters, frequency division, memory, etc. Some of them are discussed below.

4.4.1 Counters

- A counter can be defined as "a logic circuit that counts the number of clock pulses applied at the input. Each count, a binary number is known as the state of the counter".
- Hence, a counter counting in terms of n-bits will have 2^n different states. The number of different states of the counter is known as Modulus of a counter.
- The Modulus of a counter represents the number of states through which the counter passes during its operation.
- For example, A 2-bit ripple counter is called a $2^2 = 4$ states i.e. MOD-4 counter.
  A 3-bit ripple counter is called a $2^3 = 8$ states i.e. MOD-8 counter.
- Hence, an n-bit ripple counter is known as modulo-N counter, where MOD number = 2^n.

Types:

- A counter is a sequential circuit consists of flip-flops and combinational elements and are used for counting pulses. Counters can be classified as:
  (i) Asynchronous counter (Ripple counter).
  (ii) Synchronous counter, on the basis of the manner in which the flip-flops are triggered.
- In asynchronous counters, the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the Q and $\bar{Q}$ output of the previous flip-flop. In asynchronous counter, the flip-flops are not clocked simultaneously.
- In synchronous counters, the clock input is applied to all the flip-flops simultaneously. This increases the speed of operation of synchronous counters.

Applications:

(i) In Frequency counters, Digital clock, Digital triangular wave generator.
(ii) For Time measurement.
(iii) In A to D converter.

4.4.1.1 Asynchronous or Ripple Counter

- The logic diagram of a 2-bit ripple up counter is shown in Fig. 4.14.
- The toggle (T) flip-flop are being used. But we can use the JK flip-flop also with J and K connected permanently to logic 1.
- External clock is applied to the clock input of flip-flop A and Q_A output is applied to the clock input of the next flip-flop B.

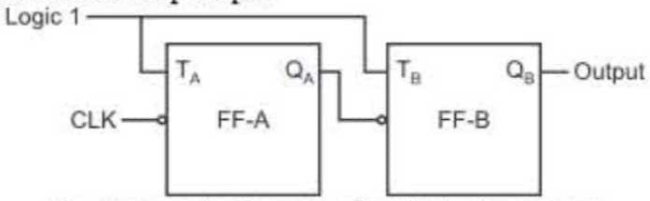


Fig. 4.14: Logic diagram of 2-bit ripple counter

- The operation of ripple counter is given below:

Table 4.10

Sr. No.	Condition	Operation
1.	Initially let both the FFs be in the reset state	Q_BQ_A = 00 initially.
2.	After 1st negative clock edge	As soon as the first negative clock edge is applied, FF-A will toggle and Q_A will be equal to 1. Q_A is connected to clock input of FF-B. Since, Q_A has changed from 0 to 1, it is treated as the positive clock edge by FF-B. There is no change in Q_B because FF-B is a negative edge triggered FF. Q_BQ_A = 01 after the first clock pulse.
3.	After 2nd negative clock edge	On the arrival of second negative clock edge, FF-A toggles again and Q_A = 0. The change in Q_A acts as a negative clock edge for FF-B. So it will also toggle, and Q_B will be 1. Q_BQ_A = 10 after the second clock pulse.
4.	After 3rd negative clock edge	On the arrival of 3rd negative clock edge, FF-A toggles again and Q_A become 1 from 0. Since this is a positive going change, FF-B does not respond to it and remains inactive. So Q_B does not change and continues to be equal to 1. Q_BQ_A = 11 after the third clock pulse.
5.	After 4th negative clock edge	On the arrival of 4th negative clock edge, FF-A toggles again and Q_A becomes 1 from 0. This negative change in Q_A acts as clock pulse for FF-B. Hence it toggles to change Q_B from 1 to 0. Q_BQ_A = 00 after the fourth clock pulse.

Table 4.11: Truth Table

Clock	Counter Output		State Number	Decimal Counter Output
	Q_B	Q_A		
Initially	0	0	–	0
1st	0	1	1	1
2nd	1	0	2	2
3rd	1	1	3	3
4th	0	0	4	0

4.4.1.2 3-Bit Asynchronous Up Counter

- Fig. 4.15 shows the diagram of a 3-bit ripple up counter.
- A 3-bit counter uses 3 flip-flops. Therefore, it has $2^3 = 8$ states.
- The clock input is applied to FF-A and Q_A output of FF-A acts as a clock input for FF-B and Q_B output of FF-B acts as the clock input for FF-C.

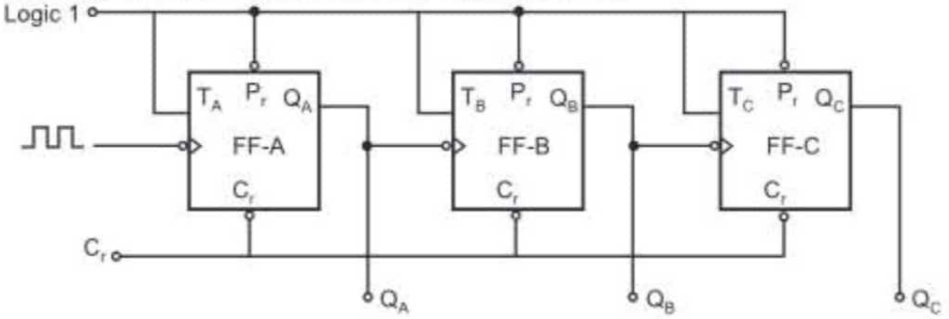


Fig. 4.15: 3-bit Ripple Counter

Working:

1. Initially, all the outputs Q_A, Q_B and Q_C are in reset condition. Therefore, $Q_C\ Q_B\ Q_A = 000$.
   All the flip-flops are negative edge-triggered flip-flops.
2. When the first negative clock edge is applied to FF-A, it toggles as $T_A = 1$. Hence, Q_A changes from 0 to 1. The Q_A output of FF-A is applied to the clock input of FF-B. Since Q_A changes from 0 to 1, it is considered as a positive edge and FF-B does not change the state. Since Q_B does not change the state, FF-C is not triggered and Q_C also does not change the state. Hence, after the first clock pulse, $Q_C\ Q_B\ Q_A = 001$.
3. When the second negative clock edge is given to FF-A, Q_A changes from 1 to 0. Hence, a 1 to 0 transition is a negative edge for FF-B, hence its Q_B output changes from 0 to 1. A 0 to 1 transition in output Q_B is treated as positive transition, hence FF-C is not triggered, hence output Q_C does not change the state. Hence, after the second clock pulse, $Q_C\ Q_B\ Q_A = 010$.
4. When the third negative edge is given to FF-A, Q_A changes from 0 to 1. Therefore, positive edge for FF-B, hence Q_B does not change the state and Q_C also does not change the state. Hence, after the third clock pulse, $Q_C\ Q_B\ Q_A = 011$.

5. When the fourth negative edge is applied to FF-A, Q_A changes from 1 to 0. Therefore, negative edge for FF-B, hence Q_B changes from 1 to 0. Therefore, negative edge for FF-C, hence Q_C changes from 0 to 1. Hence, after the fourth clock pulse, $Q_C\ Q_B\ Q_A = 100$.
6. When the fifth negative edge is applied to FF-A, Q_A changes from 0 to 1. Therefore, positive edge for FF-B, hence Q_B does not change the state and Q_C also does not change the state. Therefore, after the fifth clock pulse, $Q_C\ Q_B\ Q_A = 101$.
7. When the sixth negative edge is applied to FF-A, Q_A changes from 1 to 0. Therefore, negative edge for FF-B, hence Q_B changes the state from 0 to 1. Therefore, positive edge for FF-C, hence Q_C does not change the state. Therefore, after the sixth clock pulse, $Q_C\ Q_B\ Q_A = 110$.
8. When the seventh negative clock pulse is applied, Q_A changes from 0 to 1, hence a positive edge for FF-B, hence Q_B does not change the state and Q_C also does not change the state. Hence after the seventh clock pulse, $Q_C\ Q_B\ Q_A = 111$.

Table 4.2: Truth Table

Clock	Flip-Flop Outputs			State	Decimal Count
	Q_C	Q_B	Q_A		
Initially	0	0	0	1	0
$1^{st}\downarrow$	0	0	1	2	1
$2^{nd}\downarrow$	0	1	0	3	2
$3^{rd}\downarrow$	0	1	1	4	3
$4^{th}\downarrow$	1	0	0	5	4
$5^{th}\downarrow$	1	0	1	6	5
$6^{th}\downarrow$	1	1	0	7	6
$7^{th}\downarrow$	1	1	1	8	7
$8^{th}\downarrow$	0	0	0	1	0

4.4.1.3 Asynchronous Down Counter [S-18]

- The counters which count in descending order from maximum count to zero are called down counters.

3-bit Asynchronous Down Counter:

- Fig. 4.16 shows 3-bit asynchronous down counter.

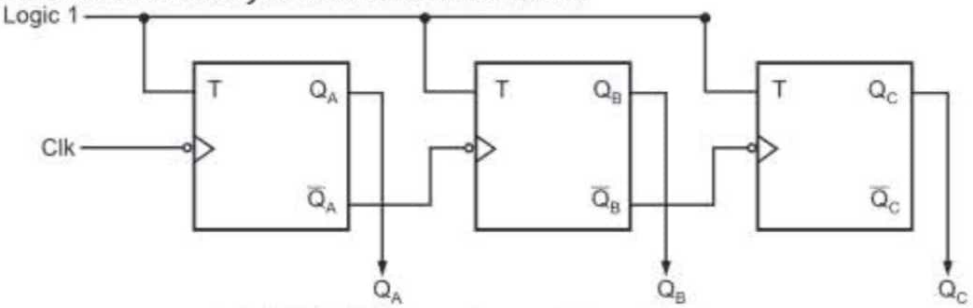


Fig. 4.16: 3-bit Asynchronous Down Counter

- The FF-A toggles at every falling edge of the clock, but FF-B will toggle only when Q_A changes from 0 to 1 i.e. $\bar{Q}_A$ changes from 1 to 0 and FF-C will toggle only when Q_B changes from 0 to 1 and $\bar{Q}_B$ will change from 1 to 0.
- The clock input is applied directly to FF-A. $\bar{Q}_A$ is connected to clock input of FF-B. $\bar{Q}_B$ is connected to clock input of FF-C.

Working:

1. Initially all the flip-flops will be in set condition (using preset terminal) $Q_C\ Q_B\ Q_A = 111$.
2. At the first negative clock edge, FF-A toggles. Therefore, Q_A changes from 1 to 0 and $\bar{Q}_A$ changes from 0 to 1. Therefore 0 to 1 transition of $\bar{Q}_A$ is treated as a positive edge. Hence Q_B does not change the state, Q_C also does not change the state. Therefore, after first clock pulse, $Q_C\ Q_B\ Q_A = 110$.
3. At the second negative clock edge, FF-A toggles. Therefore, Q_A changes from 0 to 1 and $\bar{Q}_A$ changes from 1 to 0. Therefore, 1 to 0 transition of $\bar{Q}_A$ is treated as a negative edge. Therefore, Q_B changes the state from 1 to 0 and $\bar{Q}_B$ changes from 0 to 1, which is a positive edge, hence Q_C does not change the state. Therefore, after second clock pulse, $Q_C\ Q_B\ Q_A = 101$.
4. At the third negative clock edge, FF-A toggles, Q_A changes from 1 to 0, $\bar{Q}_A$ changes from 0 to 1. Hence positive transition, therefore Q_B does not change the state and Q_C also does not change the state $Q_C\ Q_B\ Q_A = 100$.
5. At the fourth falling edge, FF-A toggles, Q_A changes from 0 to 1, $\bar{Q}_A$ changes from 1 to 0. Hence, negative transition. Therefore, Q_B changes from 0 to 1 and $\bar{Q}_B$ changes from 1 to 0. Therefore, negative edge for FF-C, hence Q_C changes from 1 to 0. Therefore, after fourth clock pulse, $Q_C\ Q_B\ Q_A = 011$.
6. At the fifth falling edge, FF-A toggles, Q_A changes from 1 to 0. Therefore, $\bar{Q}_A$ changes from 0 to 1, hence positive edge. Therefore Q_B does not change the state and Q_C also does not change the state $Q_C\ Q_B\ Q_A = 010$.
7. At the sixth falling edge, FF-A toggles, Q_A changes from 0 to 1. Therefore, $\bar{Q}_A$ changes from 1 to 0. Therefore, negative edge, hence Q_B changes from 1 to 0 and $\bar{Q}_B$ changes from 0 to 1. Therefore, positive edge, so Q_C does not change the state, $Q_C\ Q_B\ Q_A = 001$.
8. At the seventh falling edge, FF-A toggles, Q_A changes from 1 to 0. Therefore, $\bar{Q}_A$ changes from 0 to 1. Therefore positive edge, hence Q_B does not change the state and Q_C also does not change the state, $Q_C\ Q_B\ Q_A = 000$.

4.4.1.4 Synchronous Counters (3-Bit Synchronous Counter)

- If the clock pulses are applied to all the flip-flops in a counter simultaneously, then such a counter is called as synchronous counter.
- In synchronous counters all the flip-flops receive the external clock pulse simultaneously.

3-bit Synchronous Up Counter:

- Logic diagram of 3-bit synchronous counter is shown in Fig. 4.17.

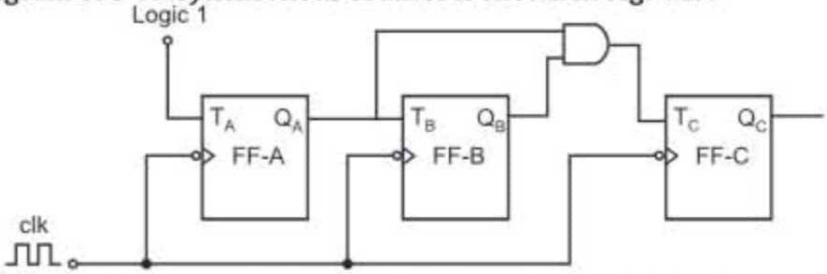


Fig. 4.17: Logic Diagram of 3-bit Synchronous Counter

Operation:

- Initially all the FFs are in reset condition. Therefore $Q_C Q_B Q_A = 000$.
  1. After the first clock pulse, FF-A will toggle and Q_A changes from 0 to 1. But since at the instant of applying the first negative edge, Q_A was 0, $T_B = 0$ and Q_B does not change the state. Therefore, $Q_B = 0$ and Q_C also remains zero.
     $\therefore$ $Q_C Q_B Q_A = 001$, after first negative edge.
  2. After the second clock pulse, FF-A toggles and Q_A changes from 1 to 0. But at the instant of applying the falling edge, Q_A was equal to 1. Therefore, $T_B = 1$, so FF-B will toggle and Q_B changes from 0 to 1. Q_C remains 0, since at the instant of applying the clock pulse Q_B was zero.
     $\therefore$ $Q_C Q_B Q_A = 010$, after second negative edge.
  3. After the third clock pulse, FF-A toggles and Q_A changes from 0 to 1. But Q_B remains unchanged, since at the instant of applying the clock pulse Q_A was 0. Q_C also remains unchanged.
     $\therefore$ $Q_C Q_B Q_A = 011$, after third negative edge.
  4. After fourth clock pulse, FF-A changes from 1 to 0. Q_A becomes 0. Q_B changes from 1 to 0, because at the instant of applying clock pulse Q_A was 1. Q_C changes from 0 to 1, because at the instant of applying clock pulse Q_B was 1.
     $\therefore$ $Q_C Q_B Q_A = 100$, after fourth negative edge.
  5. After fifth clock pulse, FF-A changes from 0 to 1. Q_B remains unchanged and Q_C also remains unchanged.
     $\therefore$ $Q_C Q_B Q_A = 101$, after fifth negative edge.
  6. After sixth clock pulse, FF-A changes from 1 to 0. Q_B changes from 0 to 1 and Q_C remains unchanged.
     $\therefore$ $Q_C Q_B Q_A = 110$, after sixth falling edge.
  7. After seventh clock pulse, FF-A changes from 0 to 1. Q_B remains unchanged and Q_C remains unchanged.
     $\therefore$ $Q_C Q_B Q_A = 111$, after seventh negative edge.

4.5.1.5 Comparison of Synchronous & Asynchronous Counters [S-18]

- Following Table shows comparison of synchronous and asynchronous counter.

Table 4.13

Sr. No.	Asynchronous Counter	Synchronous Counter
1.	Output of the previous flip-flop is connected to the clock input of the next stage.	Output of the previous flip-flop is not connected to the clock input of the next stage.
2.	Simple logic implementation.	Complex logic implementation.
3.	The flip-flops are not clocked simultaneously.	The flip-flops are clocked simultaneously.
4.	Low frequency of operation due to large propagation delay.	High frequency of operation due to low propagation delay.

4.4.1.6 Modulus Counter (MOD-N Counter) [S-17]

- The 2-bit ripple counter is called as MOD-4 counter and 3-bit ripple counter is called as MOD-8 counter. So in general, an n-bit ripple counter is called as modulo-N counter. Where, MOD number = 2^n.
- It is often desirable to construct counters which have module other than 2, 4, 8, ... etc. e.g. we may need to construct a counter having modulus 3 or 5, 6 or 7.
- A smaller modulus counter can always be constructed from a larger modulus counter by skipping states. Such counters are said to have a modified count.
- The number of flip-flops required to construct desired counter is determined by choosing the lowest natural count which is greater than the desired modified count, e.g.. A mod 9 counter requires atleast 4 flip-flops since 16 is the lowest natural count greater than 9 and requires 4 flip-flops.
- Generally, method used to skip count is to feedback a signal from some flip-flop to some previous flip-flop.

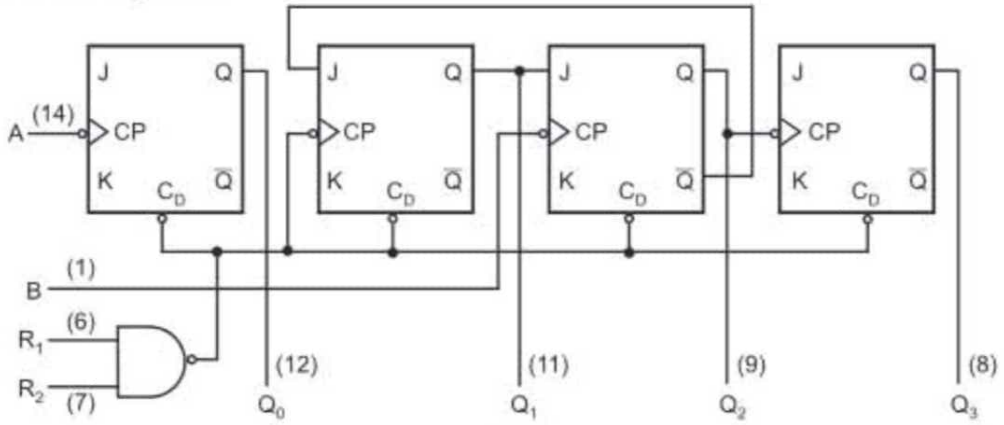
4.4.1.7 IC 7492

- The 7492 / 74LS92 counter IC is a 4-Bit ripple counter (4 cascaded counting elements). The individual counting circuits inside are partitioned into two blocks, one is a divide by two counter and the other capable of divide by 6, which when combined together effectively implements a divide by 12, perfect for the hour tracking register for a digital clock.
- The 7492 and 7493 are high speed 4-bit ripple type counters partitioned into two sections. Each counter has a divide-by-two section and either a divide-by-six (7492) or

divide-by-eight (7493) section which are triggered by a HIGH to LOW transition on the clock inputs. Each section can be used separately or tied together to form divide-by-twelve or divide-by-sixteen counters.

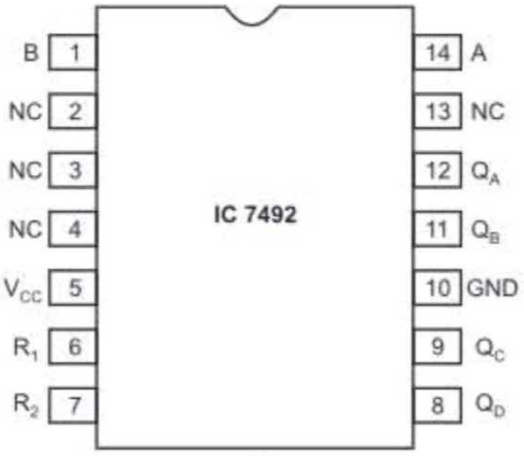
- The 7492 is 4-bit divide by twelve counter and 7493 is 4-bit binary counter. Each device consists of four master slave flip-flops which are internally connected to provide a divide-by-two section. Since the output from the divide-by-two section is not internally connected to the succeeding stages, the devices may be operated in various counting modes.

Connection diagram:



(a)

Pin Diagram:



(b)

Fig. 4.18

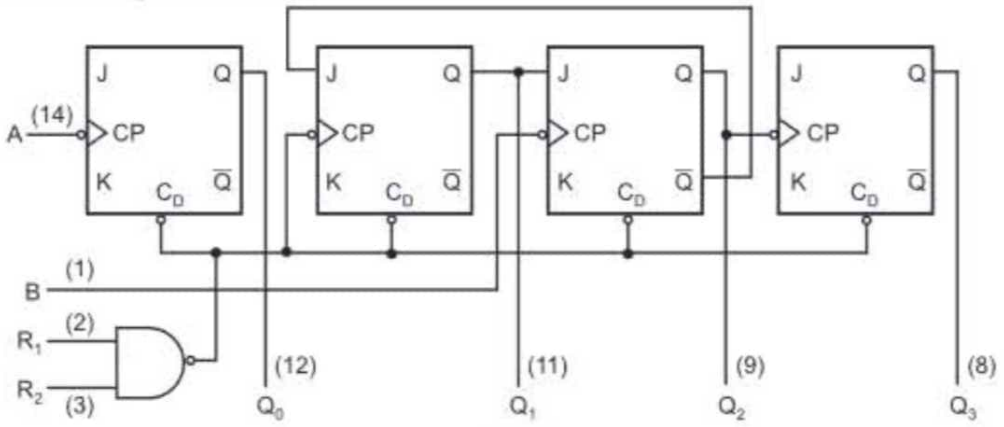
Modes of 7492:

- **Modulo 12, Divide-By-Twelve Counter:** The B input must be externally connected to the Q_A output. The A input receives the incoming count and Q_D produces a symmetrical divide-by-twelve square wave output.

- **Divide-By-Two and Divide-By-Six Counter:** No external inter-connections are required. The first flip-flop is used as a binary element for the divide-by-two function. The B input is used to obtain the divide-by-three operation at the Q_B and Q_C outputs and divide by six operation at the Q_D output.

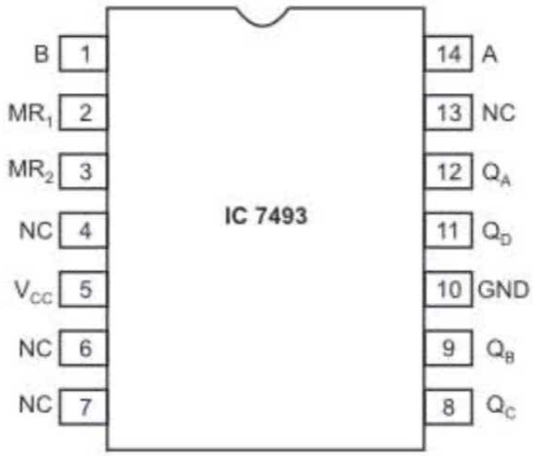
4.4.1.8 IC 7493

Connection Diagram of 7493:



(a)

Pin Diagram:



(b)

Fig. 4.19

Modes of 7493:

- **4-Bit Ripple Counter:** The output Q_A must be externally connected to input B. The input count pulses are applied to the input A. Simultaneously divisions of 2, 4, 8 and 16 are performed at the Q_A, Q_B, Q_C and Q_D outputs as shown in the truth table.
- **3-Bit Ripple Counter:** The input count pulses are applied to input A. Simultaneous frequency divisions of 2, 4 and 8 are available at the Q_B, Q_C and Q_D outputs. Independent use of the first flip-flop is available if the reset function coincides with reset of the 3-bit ripple through counter.

4.4.1.9 Type JK Design

- There are different types of flip-flop designs we could use, the S-R, the J-K, J-K Master-slave, the D-type or even the T-type flip-flop to construct a counter. But to keep things simple, we will use the D-type flip-flop, (DFF) also known as a Data Latch, because a single data input and external clock signal are used, and is also positive edge triggered.
- The D-type flip-flop, such as the TTL 74LS74, can be made from either S-R or J-K based edge-triggered flip-flops depending on whether you want it to change state either on the positive or leading edge (0 to 1 transition) or on the negative or trailing edge (1 to 0 transition) of the clock pulse. Here we will assume a positive, leading-edge triggered flip-flop.

4.4.1.10 Pre-settable Counter

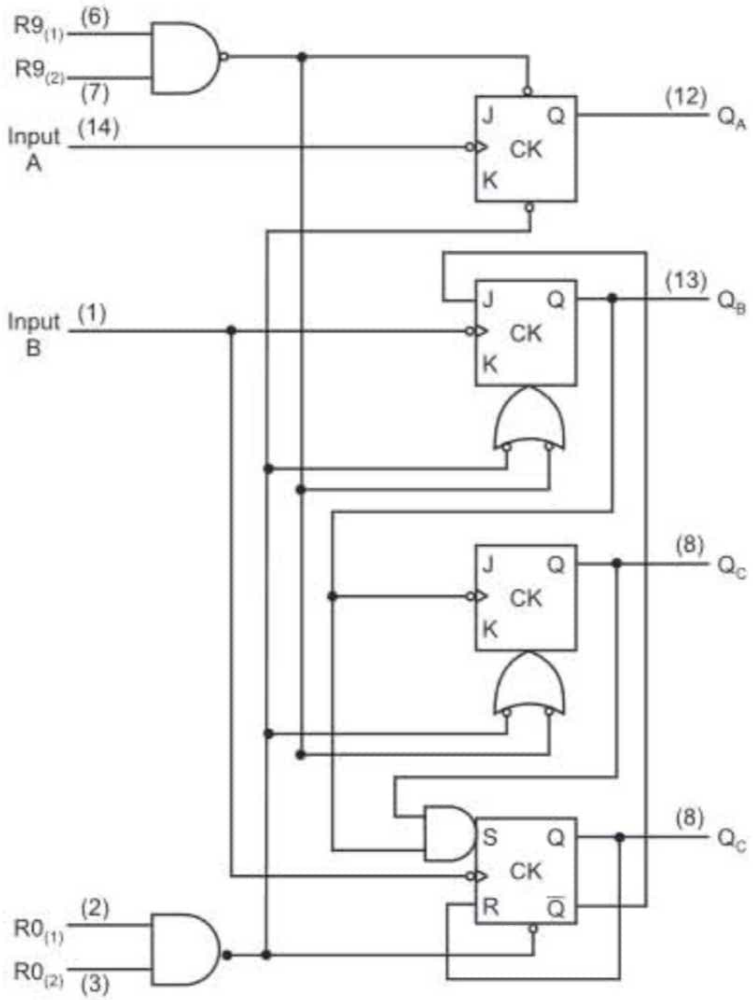
- Pre-settable Counters are those that can be preset to any starring count either asynchronously (independent of the clock signal) or synchronously (with the active transition of the clock signal). The presetting operation is achieved with the help of PRESET and CLEAR (or MASTER RESET) inputs available on the flip-flops. Presetting operation is also known as "preloading' or simply "loading' operation.
- Pre-settable counters can be UP counters, DOWN counters, or UP/DOWN counters. Additional inputs/outputs available on pre-settable counter UP/DOWN counter usually include PRESET inputs from where any desired count can be loaded; parallel load (PL) input, which when active allows the PRESET inputs to be loaded onto the counter outputs; and terminal count (TC) output which become active when the counter reaches the terminal count.

4.4.1.11 IC 7490

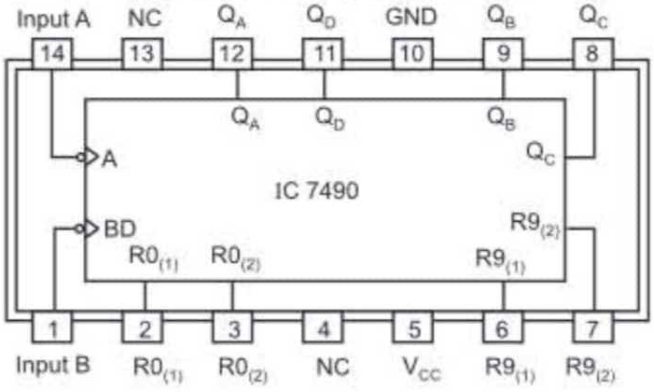
- The 7490 is a decade binary counter, meaning it is able to count from 0 to 9 cyclically. The Q_A, Q_B, Q_C and Q_D are 4-output the counting sequence is as follows:

Table 4.14

Q_D	Q_C	Q_B	Q_A
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1



(a) Connection diagram of IC7490



(b) Pin diagram of IC 7490

Fig. 4.20

Working:

- This IC 7490 contains two counters MOD2 and MOD5. Cascading of these counters will result in MOD10 counter.
- You can set the chip to count up to other maximum numbers and then return to zero. You "set it up" by changing the wiring of the $R0_1$, $R0_2$, $R9_1$ and $R9_2$ lines. If both $R0_1$ and $R0_2$ are 1 (5 volts) and either $R9_1$ or $R9_2$ are 0 (ground), then the chip will reset Q_A, Q_B, Q_C and Q_D to 0. If both $R9_1$ and $R9_2$ are 1 (5 volts), then the count on Q_A, Q_B, Q_C and Q_D goes to 1001 (5).

4.4.1.12 Analysis of Counter Circuits

- The design of asynchronous counters using flip-flops has been discussed above. Some asynchronous counters are available in MSI and are given in following Table along with some of their features. Depending upon these features, these ICs are divided into three groups A, B and C. The group to which a particular IC belongs is also indicated in the table. All these IC's consist of four master-slave positive edge-triggered flip-flops. The load, set, and reset (clear) operations are asynchronous, i.e. independent of the clock pulse.

Table 4.15: Available asynchronous counter IC in TTL

IC No.	Description	Features	Group
7490, 74290	BCD counter	Set, reset	A
7492	Divide-by-12 counter	Reset	B
7493, 74293	4-bit binary counter	Reset	B
74176, 74196	Presettable BCD counter	Reset, load	C
74177, 74197	Presettable 4-bit binary counter	Reset, load	C
74390	Dual decade counters	Reset	B
74393	Dual 4-bit binary counters	Reset	B
74490	Dual BCD counters	Set, Reset	A

4.4.1.13 Bushing

- Sometimes a counter may enter in some state which is not present in the counting sequence. As clock pulses are still applied to counter circuit the next state can be either unused state or no change. The counter then remains in the same unused state or another unused state and never arrives at used state, this condition is called as "Bushing" or "Lock out".
- To avoid lock out condition two techniques are used:
  1. An additional circuit is employed to ensure that "Lock out" does not occur. The counter is designed using the next state to be initial state from the unused state.
  2. A special circuit is used which always resets the counter whenever it got stuck into an unused state.

4.4.5 Shift Registers

- Flip-flop is a 1 bit memory cell which can be used for storing the digital data. To increase the storage capacity in terms of number of bits, we have to use a group of flip-flop. Such a group of flip-flop used to store a word is known as a Register. The n-bit register will consist of n number of flip-flop and it is capable of storing an n-bit word.
- The binary data is temporary stored and moved within the register from one flip-flop to another. The registers that allow such data transfers are called as shift registers.
- There are four mode of operations of a shift register i.e., Serial Input Serial Output, Serial Input Parallel Output, Parallel Input Serial Output, Parallel Input Parallel Output.

4.4.5.1 Types of Shift Registers

- Basic shift registers are classified by structure according to the following types:
  1. Serial In Serial-Out.
  2. Parallel In Serial Out
  3. Serial In Parallel Out
  4. Parallel In Parallel Out
  5. Bidirectional
  6. Universal Shift Register

1. Serial In Serial Out (SISO) Shift Register:

- In SISO there is only one output, the data leaves the shift register one bit at a time in a serial (sequential) pattern or manner.

(a) Shift-Left Register:

- Fig. 4.20 (a) shows shift-left register.

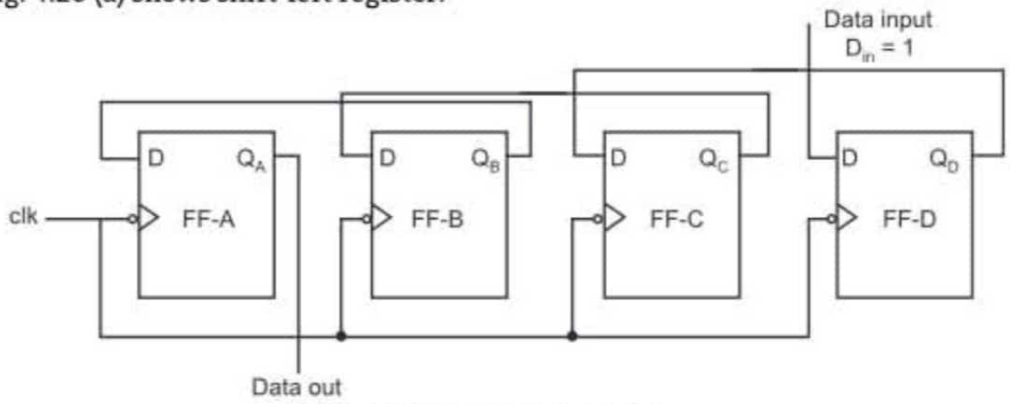


Fig. 4.21 (a) Shift-left Register

- Consider the entry of the 4-bit binary number 1111 into the register, starting with the leftmost bit.

  Initially, the register is cleared.

  $\therefore$ $Q_A Q_B Q_C Q_D = 0000$

  (i) When data 1111 is applied serially, i.e.

  $D_{in} = 1$, $Q_A Q_B Q_C Q_D = 0000$

  The application of first negative clock edge, sets the rightmost flip-flop, then the data stored is,

  $Q_A Q_B Q_C Q_D = 0001$

(ii) When the next falling edge is applied, flip-flop Q_C sets. Therefore, the data in the register is,

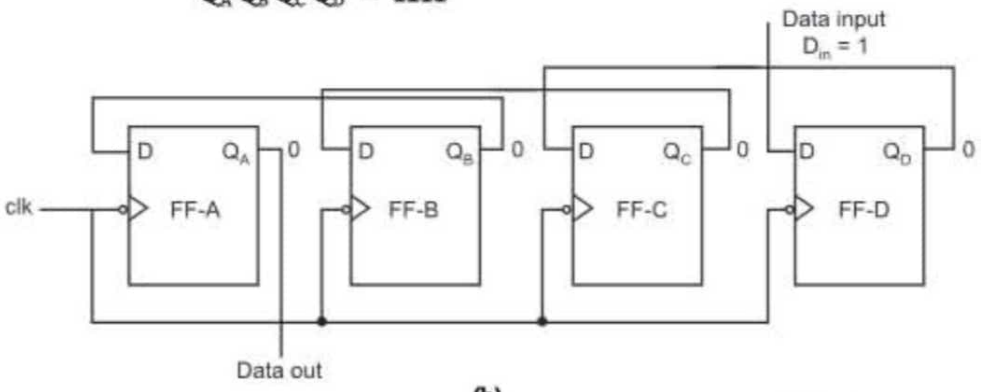
$$Q_A\ Q_B\ Q_C\ Q_D = 0011$$

(iii) When the third falling edge is applied, flip-flop Q_B sets. Therefore, the register contents become,

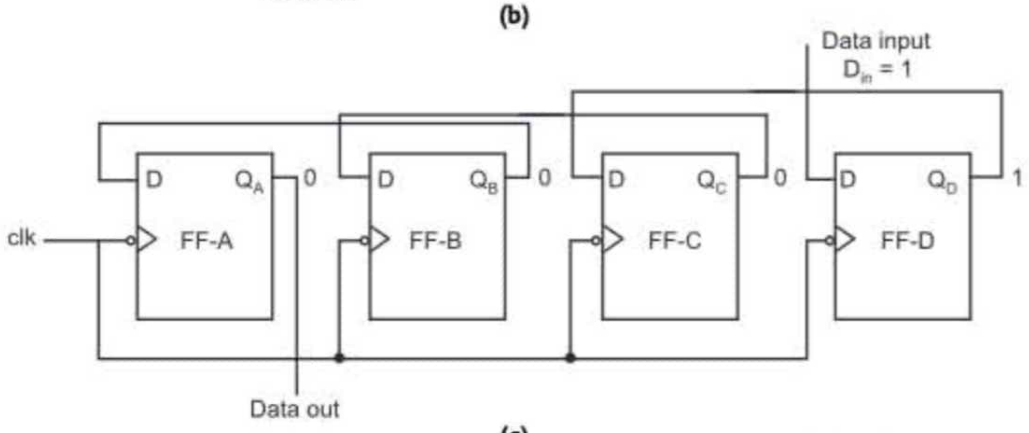
$$Q_A\ Q_B\ Q_C\ Q_D = 0111$$

(iv) The fourth negative clock edge gives,

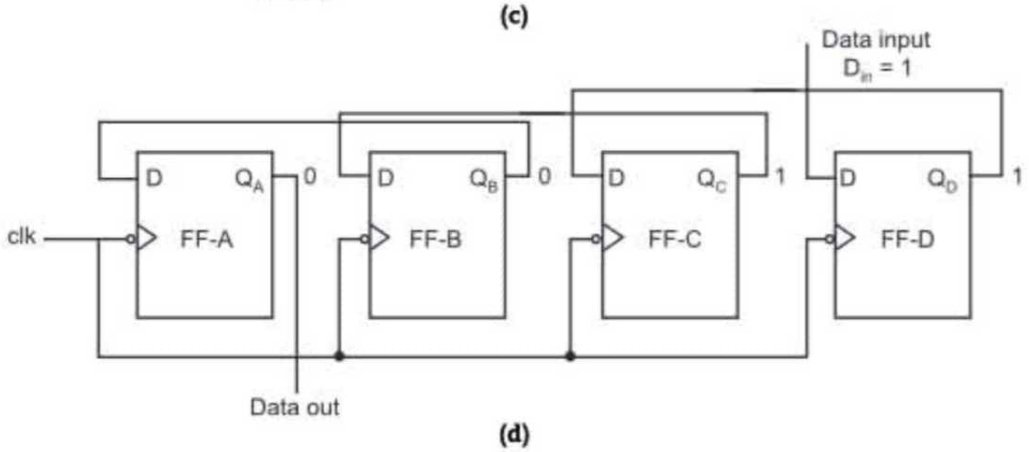
$$Q_A\ Q_B\ Q_C\ Q_D = 1111$$



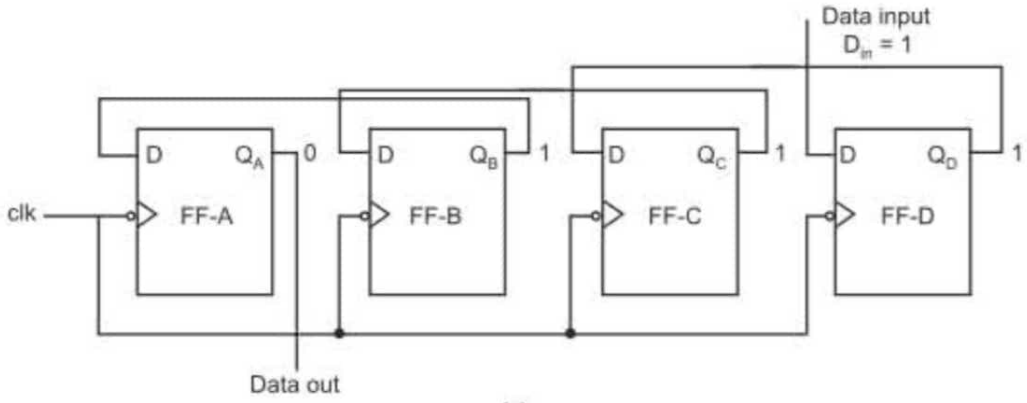
(b)



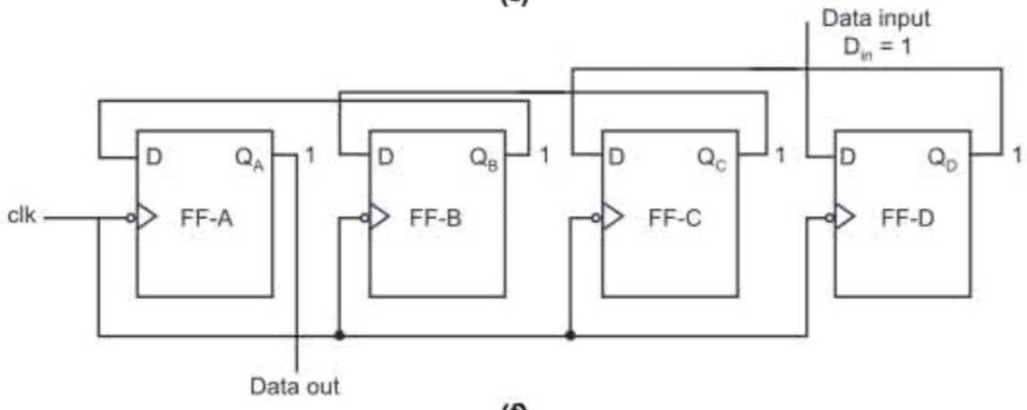
(c)



(d)



(e)



(f)

Fig. 4.21: Four Bits 1111 being Serially Entered into Shift-Left Register

(b) Shift-Right Register:

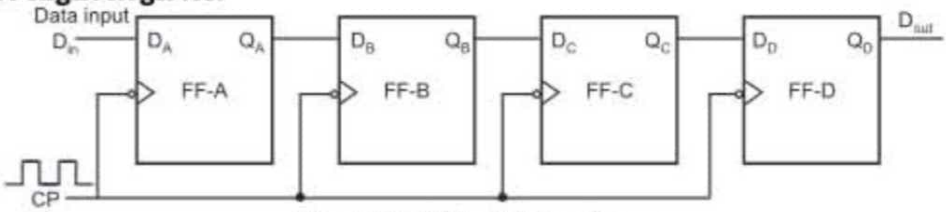


Fig. 4.22: Shift-right Register

- Initially, the register is cleared, so $Q_A\ Q_B\ Q_C\ Q_D = 0000$. When data 1111 is applied serially, leftmost bit is applied as D_{in}.

  $D_{in} = 1$

  (i) On arrival of first negative clock edge, FF-A sets and the stored word becomes $Q_A\ Q_B\ Q_C\ Q_D = 1000$.

  (ii) When the next negative clock edge is applied, FF-B sets.

  $\therefore \quad Q_A\ Q_B\ Q_C\ Q_D = 1100$

  (iii) When the third falling clock edge is applied, FF-C sets.

  $\therefore \quad Q_A\ Q_B\ Q_C\ Q_D = 1110$

  (iv) On the fourth falling clock edge,

  $Q_A\ Q_B\ Q_C\ Q_D = 1111$

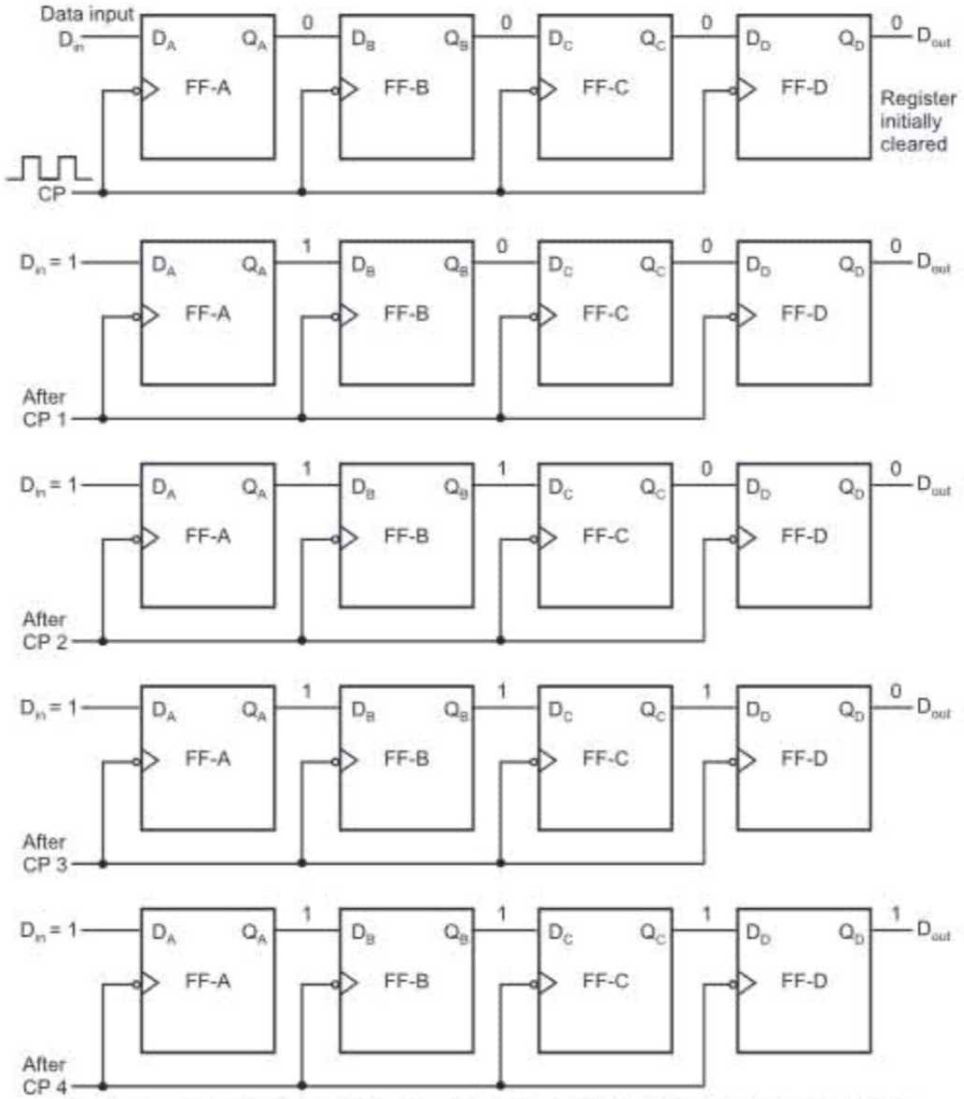


Fig. 4.23: Four Bits 1111 Being Serially Entered into Shift-Right Register

2. Serial In Parallel Out (SIPO) Shift Register:

- The data is applied serially and the output is taken parallelly.
- Once, the data is stored, each bit appears at the respective outputs and all the bits are available simultaneously.

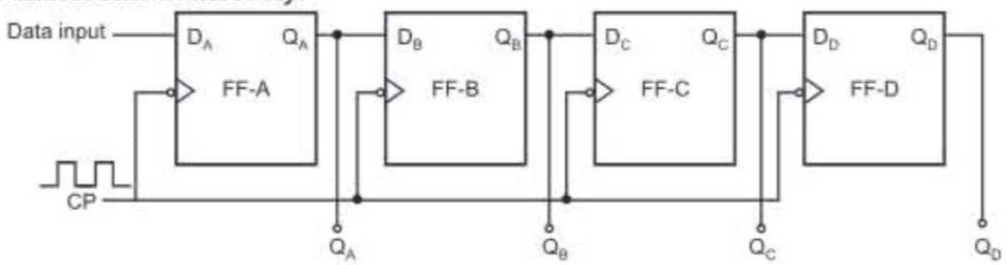


Fig. 4.24: Serial In Parallel Out (SIPO) Shift Register

3. Parallel Input Serial Output (PISO) Shift Register:

- In PISO data bits are entered in parallel fashion.

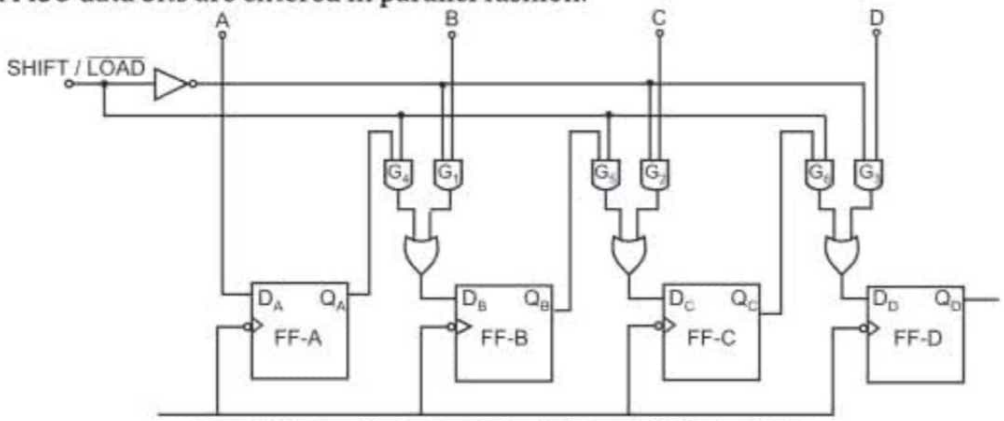


Fig. 4.25: Parallel In Serial Out (PISO) Shift Register

- In this type of shift register, the bits are loaded simultaneously into their respective stages using parallel lines. A, B, C, D are the four input lines for loading the data. The SHIFT/ $\overline{\text{LOAD}}$ is the control input which permits shifting or loading operation of the register.
  - (i) When SHIFT/ $\overline{\text{LOAD}}$ pin is low, gates G_1, G_2, G_3 are enabled and the data is loaded in the respective flip-flops. When the clock pulse is applied, the flip-flops with D = 1 will SET and with D = 0 will RESET.
  - (ii) When SHIFT/ $\overline{\text{LOAD}}$ pin is high, gates G_1, G_2, G_3 are disabled and gates G_4, G_5, G_6 are enabled. This allows the data to shift left from one stage to the next.

4. Parallel In Parallel Out (PIPO) Shift Register:

- In parallel in parallel out register, the data bits are loaded simultaneously on 4-parallel input lines and the outputs are taken simultaneously on 4 parallel output lines.
- Fig. 4.26 shows PIPO.

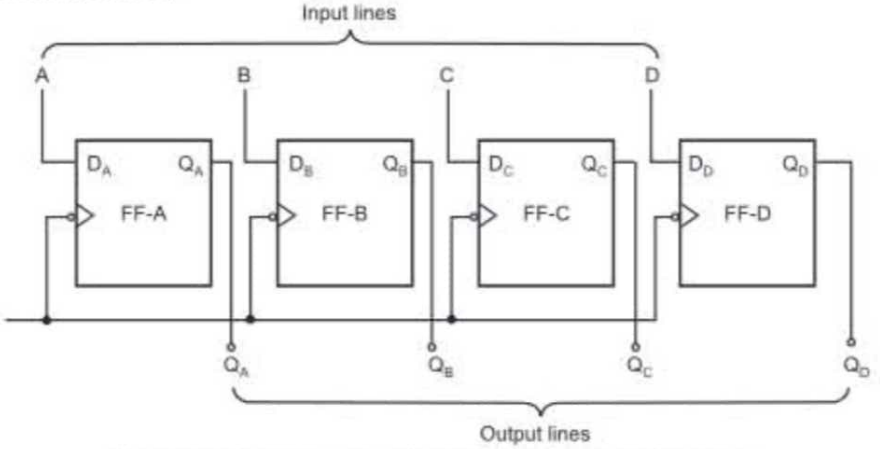


Fig. 4.26: Parallel In Parallel Out (PIPO) Shift Register

4.4.5.2 Applications Shift Register [W-17]

- As shift registers accept data and provide data, they form important link between main digital system (processor) and input output devices.
- Apart from storage and transfer of data, shift registers find application in arithmetic operations, in data conversions from serial to parallel and vice-versa and in design of special types of counters.

4.4.5.3 Ring Counter [S-18, W-17]

- A ring counter is a type of counter composed of a type of circular shift register. In ring counter the output of the last shift register is fed to the input of the first register. Ring counter is a special application of shift register.
- Fig. 4.39 shows a ring counter.

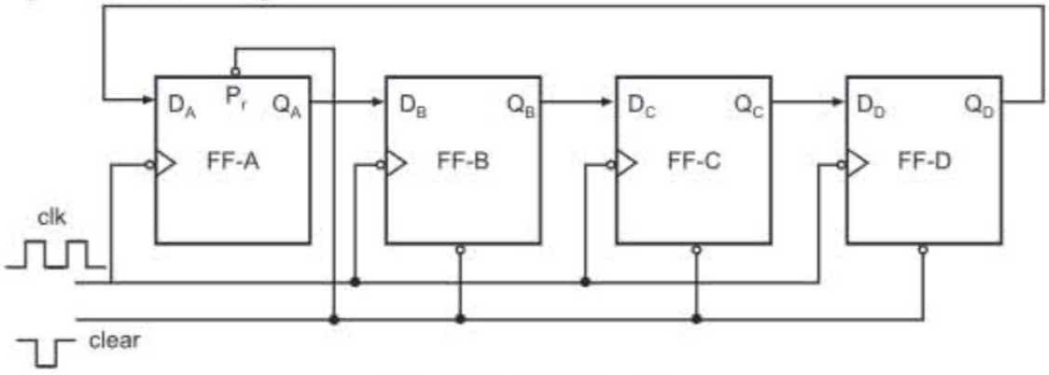


Fig. 4.27: Ring Counter

Operation:

1. A logic 0 to the clear terminal will reset the flip-flops B, C and D, whereas the Q_A output of FF-A will be set to logic 1 using the preset terminal.

   $\therefore \quad Q_D\, Q_C\, Q_B\, Q_A = 0001$

   The clear terminal is deactivated by applying a logic 1 signal to it. All the flip-flops are triggered simultaneously using the falling edge of the clock.

   (i) When the first negative edge of the clock is applied, FF-B will set because $Q_A = D_B = 1$ and (at the instant of applying the clock Q_A was 1) FF-A will reset because $Q_D = D_A = 0$ (at the instant of applying the clock pulse Q_D was 0).

   $\therefore \quad Q_D\, Q_C\, Q_B\, Q_A = 0010$

   (ii) When the second negative edge of the clock is applied, only FF-C will set because $Q_B = D_C = 1$ and FF-B will reset because $Q_A = D_B = 0$.

   $\therefore \quad Q_D\, Q_C\, Q_B\, Q_A = 0100$

   (iii) When the third negative edge of the clock is applied, only FF-D will set because $Q_C = D_D = 1$ and FF-C will be reset, because $Q_B = D_C = 0$.

   $\therefore \quad Q_D\, Q_C\, Q_B\, Q_A = 0001$

Table 4.16: Truth Table

clr	clk	Q_A	Q_B	Q_C	Q_D
⊓	×	1	0	0	0
1	↓	0	1	0	0
1	↓	0	0	1	0
1	↓	0	0	0	1
1	↓	1	0	0	0

4.4.5.4 Johnson Counter

- The Johnson counter is a modification of ring counter. In this counter the inverted output of the last stage flip-flop is connected to the input of first flip flop. The Johnson counter is also known as Twisted Ring Counter.
- Fig. 4.40 shows Johnson counter.

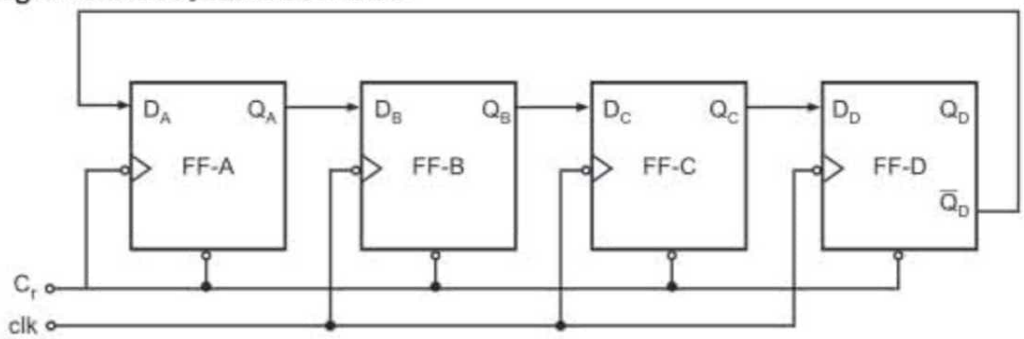


Fig. 4.28: Twisted Ring Counter

Working:

- In Johnson counter, the $\bar{Q}_D$ output of FF-D is connected to D_A input of FF-A. All the flip-flops are negative edge triggered and are clocked simultaneously. All the flip-flops are reset initially using the clear input terminal. Therefore, initially $Q_D\ Q_C\ Q_B\ Q_A = 0000$.

1. At the first falling edge of the clock, FF-A will be set ($\because D_A = \bar{Q}_D = 1$ at the instant of applying the clock pulse). Therefore, $Q_A = 1$.
   $\therefore \quad Q_D\ Q_C\ Q_B\ Q_A = 0001$
2. At the second falling edge, FF-B will also set ($\because D_B = Q_A = 1$ at the instant of applying the clock pulse). Therefore, $Q_B = 1$ and Q_A also remains 1 because $D_A = \bar{Q}_D = 1$ ($\because Q_D = 0$).
   $\therefore \quad Q_D\ Q_C\ Q_B\ Q_A = 0011$
3. At the third falling edge, FF-C will also set ($\because D_C = Q_B = 1$ at the instant of applying the clock pulse), Q_C becomes 1 and Q_B, Q_A are also set ($\because D_A = \bar{Q}_D = 1$ and $D_B = Q_A = 1$).
   $\therefore \quad Q_D\ Q_C\ Q_B\ Q_A = 0111$

4. At the fourth falling edge, FF-D will set (∵ $D_D = Q_C = 1$, at the instant of applying the clock pulse). Q_D becomes 1. Therefore $\bar{Q}_D = 0$. Q_C Q_B Q_A are also set.
   ∴ Q_D Q_C Q_B Q_A = 1111
5. When the next negative clock edge is applied, FF-A will reset (∵ $D_A = \bar{Q}_D = 0$, at the instant of applying the clock pulse). Therefore $Q_A = 0$ and Q_B Q_C $Q_D = 1$.
   ∴ Q_D Q_C Q_B Q_A = 1110
6. At subsequent negative clock edges, Q_D Q_C Q_B will also reset, following Q_A.

Table 4.17

Clear	clk	Q_D	Q_C	Q_B	Q_A
⊓	Initially	0	0	0	0
1	↓	0	0	0	1
1	↓	0	0	1	1
1	↓	0	1	1	1
1	↓	1	1	1	1
1	↓	1	1	1	0
1	↓	1	1	0	0
1	↓	1	0	0	0
1	↓	0	0	0	0

Summary

- A counter use of a group of flip-flops to count the number of clock pulses with the number of stable states in its counting cycle.
- Asynchronous and synchronous counters differ only in the method they are clocked. The maximum clock frequency for an asynchronous counter decreases as the number of bits increases. For a synchronous counter, the maximum clock frequency remains the same, regardless or the number or bits.
- In asynchronous (ripple) counters, the clock signal is applied to the LSB flip-flop, and all other flip-flops are clocked by the output or the preceding flip-flops.
- A sequential logic circuit has an output that is a function of previous sequence of inputs as well as the present input sequence. Design and analysis of these circuits can be carried out with the help of state tables and state diagrams.
- Sequential circuits are divided into asynchronous and synchronous circuits. A synchronous sequential circuits generally employ flip-flop as memory elements and their operation is synchronized by a special synchronizing pulse obtained from a system clock.
- An asynchronous sequential circuit is basically a combinational logic with direct feedback, there is no timing signal involved and one operation triggers another operation.

Check Your Understanding

1. A ripple counter's speed is limited by the propagation delay of _____.
   (a) each flip-flop (b) all flip-flops and gates
   (c) the flip-flops only with gates (d) only circuit gates
2. Memory elements in clocked sequential circuits are called _____.
   (a) latches (b) flip-flop
   (d) signals (d) gates
3. The flip-flops can be constructed with two _____.
   (a) NAND gates (b) XOR gates
   (c) AND gates (d) NOT gates
4. A counter that flows binary sequence is called _____.
   (a) ripple counter (b) edge counter
   (c) binary counter (d) level counter
5. In shift register, each bit is updated on a _____.
   (a) clock transition (b) input signal
   (c) output signal (d) source voltage

Answers

1. (a)	2. (b)	3. (a)	4. (c)	5. (a)

Practice Questions

Q.1 Answer the following Question in brief:

1. What is sequential circuit?
2. Describe JK flip-flop with diagram.
3. Explain following types of flip-flops: (i) D-type, (ii) T-type.
4. What is meant by shift registers? Enlist its types.
5. With are the types of counters?

Q.2 Answer the following Question:

1. Write short note on: (i) Ring counter, (ii) Modulus counter.
2. With the help of diagram describe asynchronous down counter.
3. Describe asynchronous counter in detail.
4. Explain pin diagram of IC 7495.
5. Explain SR flip-flop diagrammatically.
6. Write short note on 'IC 7490'.

Q.3 Define Terms:

Flip-Flop, Counter, SISO, SIPO, Latch, JK flip-flop, Asynchronous counter, Ring counter, Shift Register.

Previous Exams Questions

Winter 2017

1. The flip-flop in which the output changes at the falling edge of the clock, it is _____. **(1 M)**

(a) Negative flip-flop
(b) Negative edge triggered flip-flop
(c) Positive edge triggered flip-flop
(d) None of the above

Ans. Refer to Section 4.3.

2. Define latch. **(1 M)**

Ans. Refer to Section 4.2.

3. State the applications of shift registers. **(4 M)**

Ans. Refer to Section 4.8.2.2.

4. Distinguish between edge and level triggered flip-flops. **(3 M)**

Ans. Refer to Section 4.3.

5. Explain ring counter. **(4 M)**

Ans. Refer to Section 4.8.2.3.

Summer 2017

1. In _____ counter the clock is applied parallel to all flip-flops. **(1 M)**

(a) Asynchronous
(b) Ripple
(c) Synchronous
(d) None of the above

Ans. Refer to Section 4.3.

2. Define positive edge triggered flip-flop. **(1 M)**

Ans. Refer to Section 4.3.

3. Which are the common applications of flip-flops? Explain the bounce eliminating switch. **(4 M)**

Ans. Refer to Section 4.8.

4. What do you mean by modulus of a counter? **(4 M)**

Ans. Refer to Section 4.8.1.5.

5. Write a short note on T-flip flop. **(3 M)**

Ans. Refer to Section 4.3.6.

Summer 2018

1. Which flip flop is used to remove invalid condition? **(1 M)**

(a) SR
(b) JK
(c) Clocked SR
(d) Master slave JK

Ans. Refer to Section 4.3.

2. Define the term: Ring counter. **(1 M)**

Ans. Refer to Section 4.8.2.3.

3. Describe the working of RS flip-flop with logic diagram and truth table. **(5 M)**

Ans. Refer to Section 4.3.

4. Differentiate between synchronous and asynchronous counters with an example. **(3 M)**

Ans. Refer to Section 4.8.1.5.

5. Draw neat diagram of 3-bit combined up down asynchronous counter and explain its working. **(4 M)**

Ans. Refer to Section 4.8.1.3.

6. Explain implementation of D and T flip-flop and their applications. **(4 M)**

Ans. Refer to Sections 4.3.6 and 4.3.6.

■■■

5...

CPU, Memory and I/O Organisation

Objectives...

- To learn basic concepts of CPU.
- To provide information about Memory System of Computer.
- To understand organization of I/O System of Computer.

5.1 INTRODUCTION

- The part of a computer that performs the bulk of data processing operations is called the central processing unit and is referred as the CPU. The CPU is made-up of three major parts as shown in Fig. 5.1.
- The register set stores intermediate data used during the execution of the instructions. The Arithmetic Logic Unit (ALU) performs the required micro-operations for executing the instructions. The control unit supervises the transfer of information among the registers and instructs the ALU which operation to perform.

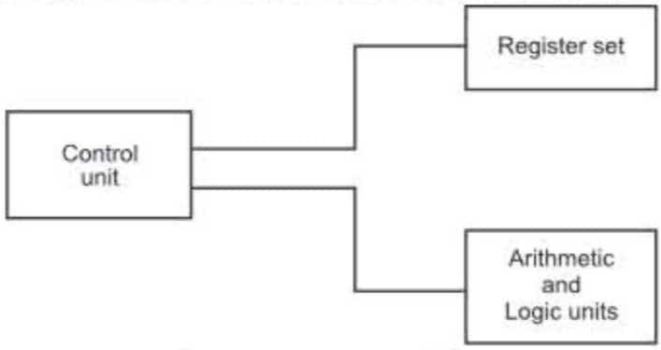


Fig. 5.1: Components of CPU

- The CPU performs the variety of functions dictated by the type of instructions that are incorporated in the computer. The computer instruction set provides the specifications for the design of the CPU. The design of CPU is a task that involves choosing the hardware for implementing the machine instructions.

- The users who writes program in machine language or assembly language must be aware of the register set, the memory structure, the type of data supported by the instructions and the function that each instruction performs.

5.2 BLOCK DIAGRAM OF CPU [S-17, W-17]

- The block diagram for a basic Central Processing Unit (CPU) for a microprocessor as shown in Fig. 5.2.

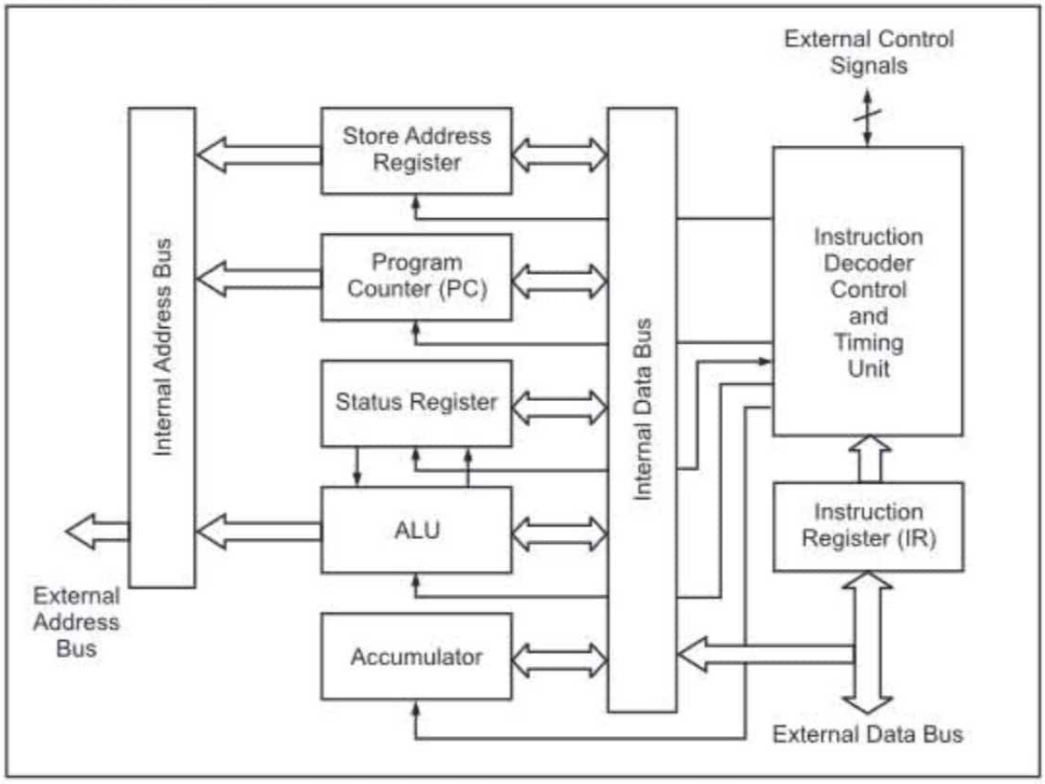


Fig. 5.2: Basic CPU Block Diagram

- The block diagram of the CPU consists of following components:
  1. **Arithmetic and Logic Unit (ALU):** Provides a set of arithmetic and logic functions.
  2. **Accumulator:** A register used to hold one of the inputs to the ALU and the results of an ALU operation. This is used for temporary storage and is one of the most used registers within the CPU.
  3. **Program Counter (PC):** This is a counter that increments after each instruction and tracks program execution to ensure that the program executes in the correct sequence.
  4. **Store Address Register:** A register that can be loaded with a single address in memory that might be required by the program,
  5. **Status Register:** Also referred to as a flag register, it is used to store information relating to the last operation undertaken by the ALU.
  6. **Instruction Decoder Timing Unit, Control and Timing Unit:** Used for organizing the data flow between the different parts of the CPU.

7. **Instruction Register (IR):** Used to store an instruction that the microprocessor is to decode and act upon.
8. **Buses:** It is a communication system that transfers data/information between components inside a computer or CPU.

- There are two types of buses: an internal bus and external bus.
  1. **Internal bus:** An internal bus within a chip which connects the processor with cache memory.
  2. **External bus:** An external bus connects CPU chip with external memory chip. Another external bus is one which connects I/O interface units with CPU/Memory. Let us see about external buses.
- The bus consists of four major group of lines: power lines, address lines, data lines, and control lines:
  1. **Data bus:** Data lines are used to transmit data and the collection of these lines is called a data bus. The width of a data bus is the number of parallel wires which carry data. Usually data bus widths are 8,16,32 or 64. Larger the bus width better will be the system performance.
  2. **Address Bus:** Address lines carry the address of data in main memory are collectively called s address bus. In this case, the bus width depends on direct addressability of memory. The increase in memory sizes address bus widths have increased. Besides addressing main memory, these lines are also used to address I/O devices.
  3. **Control Bus:** The control bus is used to control the access to and use of data and address buses by various units which share the bus. Besides this, they carry command signals to specify operations such as memory write, memory read, I/O write, I/O read, data acknowledge, bus request, bus grant, interrupt request, interrupt acknowledge and clock signal to synchronize operations on the bus.
- The interconnection lines mark the direction of data or information with the direction of the arrow. This can be one-way or two-way in direction.

5.3 FUNCTIONS OF CPU [S-17, S-18]

- The CPU processes instructions it receives in the process of decoding data. In processing this data, the CPU performs four basic functions or steps as given below:
  1. **Fetch:** Each instruction is stored in memory and has its own address. The processor takes this address number from the program counter, which is responsible for tracking which instructions the CPU should execute next.
  2. **Decode:** All programs to be executed are translated to into Assembly instructions. Assembly code must be decoded into binary instructions, which are understandable to your CPU. This step is called decoding.
  3. **Execute:** While executing instructions the CPU can do one of three things: Do calculations with its ALU, move data from one memory location to another, or jump to a different address.
  4. **Store:** The CPU must give feedback after executing an instruction and the output data is written to the memory.

5.4 GENERAL REGISTER ORGANIZATION

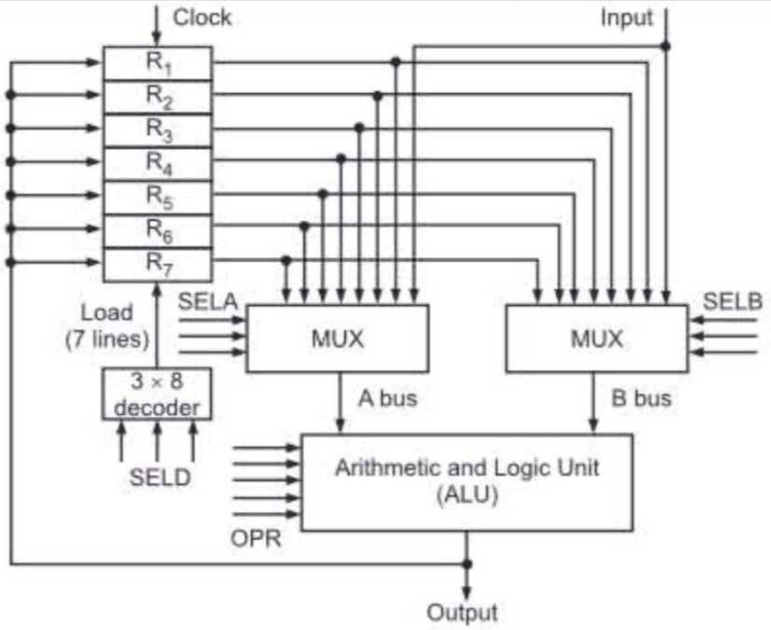


Fig. 5.3: Bus organization for seven CPU registers

- The output of each register is connected to two multiplexers the MUX to ALU through bus A and bus B.
- The selection lines in each MUX select one register or the input data for the particular bus.
- Bus A and Bus B from the inputs to a common ALU. The operation selected in the ALU determines the arithmetic or logic micro-operations that are to be performed.
- The output result of the micro-operation is available for output data and also goes into the inputs of all the registers.
- The register that receives the information from the output bus is selected by a decoder.
- A decoder provides a data path between the data in the output bus and the inputs of the selected destination register.
- The CU that operates the CPU bus system detects the information flow through the registers and ALU by selecting the various components in the system.
- For example: To perform the following operations:

  $R_1 \leftarrow R_2 + R_3$

  1. MUX A selector (SELA): to place the content to R_2 into BUS A.
  2. MUX B selector (SELB): to place the content of R_3 into BUS B.
  3. ALU Operation Selector (OPR): to provide the arithmetic addition $R_2 + R_3$.
  4. Decoder selector (SELD): to transfer the content of the output bus into R_1.
- An operation is selected by ALU operation selector (OPR).

- The results of a micro operation is directed to a destination register selected by a decoder (SELD).
- **Control word:** The 14 binary selection inputs (3 bits of SELA, 3 for SELB, 3 for SELD and 5 for OPR) as shown in Fig. 5.4.

Fig. 5.4: Control Word

ALU (Arithmetic Logic Unit):

- Individual registers performing the micro-operations directly. The computer system employs a number of storage registers connected to a common operation unit known as ALU.
- To perform or carry a micro-operation the contents of specified register are placed in the input of the common ALU.
- The ALU performs on operation and the result of the operation is then transferred to a destination register.
- The encoding of the ALU operations for the CPU is shown in following table.
- The OPR fields have five bits and each operation is designated with a symbolic name.

Table 5.1: Encoding of ALU operations

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A – B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

5.5 FLAGS [W-17]

- The FLAGS register is the status register in Intel x86 microprocessors that contains the current state of the processor.
- The Flag register is a Special Purpose Register. Depending upon the value of result after any arithmetic and logical operation the flag bits become set (1) or reset (0).
- The 5 flags are:

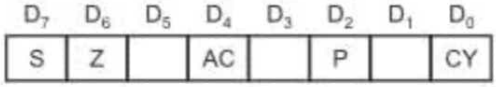


Fig. 5.5: Flags

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 is set, i.e. 1. If the MSB is 0, it indicates the number is positive and the sign flag is 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)

Zero Flag (Z):

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

    1 - zero result.

    0 - non-zero result.

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 is set i.e. 1, otherwise it is reset i.e. 0. This is the only flag register which is not accessible by the programmer

  1 - carry out from bit 3 on addition or borrow into bit 3 on subtraction.

  0 - otherwise.

Parity Flag (P):

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

  1 - accumulator has even number of 1 bits

  0 - accumulator has odd parity

Carry Flag (CY):

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

5.6 CONCEPT OF RISC AND CISC [S-18]

5.6.1 CISC Processor

- CISC stands for Complex Instruction Set Computer.
- If the control unit contains a number of micro-electronic circuitry to generate a set of control signals and each micro-circuitry is activated by a micro-code, this design approach is called CISC design.
- It is known as Complex Instruction Set Computer.

- It was first developed by Intel.
- It contains large number of complex instructions.
- In this instructions are not register based.
- Instructions cannot be completed in one machine cycle.
- Data transfer is from memory to memory.
- Micro programmed control unit is found in CISC.
- Also they have variable instruction formats

Examples of CISC processors are:

- Intel 386, 486, Pentium, Pentium Pro, Pentium II, Pentium III
- Motorola's 68000, 68020, 68040, etc.

Architecture of CISC:

- Its architecture is designed to decrease the memory cost because more storage is needed in larger programs resulting in higher memory cost.
- To resolve this, the number of instructions per program can be reduced by embedding the number of operations in a single instruction.

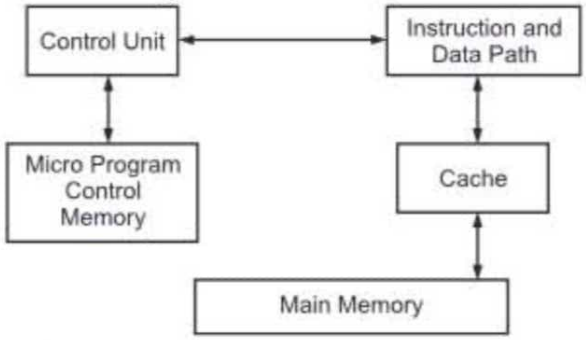


Fig. 5.6: Architecture of CISC Processor

Features of CISC:

Some features of CISC machines are:

- **Instruction set:** Simple to complex instructions and uses multiple instructions.
- **Instruction format:** Variable length instruction formats are used, so special decoding package is needed.
- **Pipelining:** CISC instruction stages are comparatively larger than those of RISC ones. This directly affects pipelining performance.
- **Addressing modes:** Large variety of addressing modes, 5 to 20 different modes are found in CISC.
- **General purpose registers:** CISC machines uses limited number of registers.
- **Micro-program:** Heavily uses micro-program to interpret the instructions.
- **Memory access:** Instruction execution generation in CISC needed to access memory frequently which makes it slow. Instructions manipulate operands in memory.

- **Control system:** Almost all CISC devices are micro programmed so no issues regarding control memory is to be dealt. The main complexity is to design the micro-program.
- **Usage:** Used in general purpose applications where speed of processor is not primary. CISC machines are preferable in such case due to their lower costs.
- **Example:** System/360 through z/Architecture, PDP-11, VAX, Motorola 68k, and Intel x86 series, Pentium series etc.

5.6.2 RISC Processor

- RISC stands for Reduced Instruction Set Computer.
- It is designed to reduce the execution time by simplifying the instruction set of the computer. Using RISC processors, each instruction requires only one clock cycle to execute results in uniform execution time.
- This reduces the efficiency as there are more lines of code, hence more RAM is needed to store the instructions.
- The compiler also has to work more to convert high-level language instructions into machine code.
- RISC chips require fewer transistors which make them cheaper to design and produce.
- In RISC, the instruction set contains simple and basic instructions from which more complex instruction can be produced. Most instructions complete in one cycle, which allows the processor to handle many instructions at same time.

Examples of the RISC processors are:

- Power PC: 601, 604, 615, 620
- DEC Alpha: 210642, 211066, 21068, 21164
- MIPS: TS (R10000) RISC Processor
- PA-RISC: HP 7100LC

Architecture of RISC:

- RISC microprocessor architecture uses highly-optimized set of instructions. It is used in portable devices like Apple iPod due to its power efficiency.

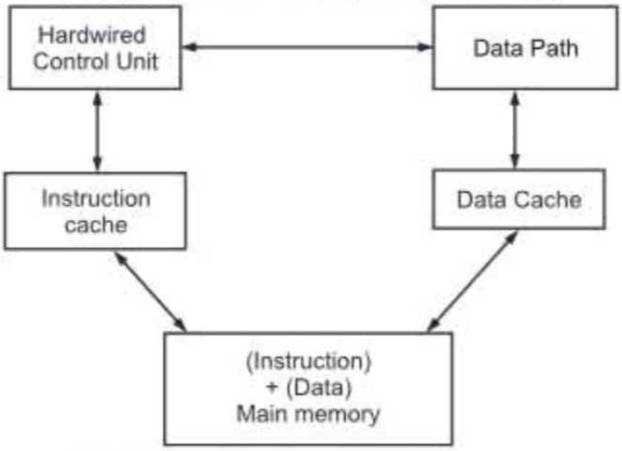


Fig. 5.7: Architecture of RISC Processor

Features of RISC:

Some features of RISC are:

1. **Instruction set:** Simple and few instructions.
2. **Instruction format:** Fixed length instructions format are used and they can be easily decoded.
3. **Pipelining:** Single cycle instruction execution so very fast. RISC instruction stages are comparatively smaller than those of CISC, thus RISC architecture permits heavy pipelining while executing instructions.
4. **Addressing modes:** Few addressing modes as compared to CISC.
5. **General purpose registers:** RISC machines uses relatively large number of registers.
6. **Micro-program:** Do not use micro-program to interpret the instructions.
7. **Memory access:** Memory access is limited to LOAD and STORE instruction for RISC architecture devices.
8. **Control system:** Control system is hardwired and memory lines are accessed making system faster. Major complexity is to design the computer with simple hardware architecture.
9. **Usage:** Used in scientific and research oriented tasks where processor speed should be very high.
10. **Example:** R4400SC (MIPS), PA7100 (HP), PowerPC (IBM), SuperSparc (SUN) etc.

Comparison of RISC and CISC:

Sr. No.	RISC	CISC
1.	Instruction takes one or two cycles.	Instruction takes multiple cycles.
2.	Instructions executed by hardware.	Instructions executed by the micro program.
3.	Few instructions.	Complex instruction set.
4.	Only load/store instructions are used to access memory.	In addition to load and store instructions, memory access is possible with other instructions also.
5.	Most of them have multiple register banks.	Single register bank.
6.	Fixed format instructions.	Variable format instructions.
7.	Few addressing modes.	Many addressing modes.
8.	Smaller set of instructions. Difficult to program.	Larger set of instructions. Easy to program.
9.	Complex design of compiler.	Simpler design of compiler, considering larger set of instructions.
10.	Low clock cycle per second.	Higher clock cycles per second.
11.	Emphasis is on software.	Emphasis is on hardware.
12.	Faster execution, as each instruction is to be executed by hardware.	Slower execution, as instructions are to be read from memory and decoded by the decoder unit.
13.	Pipelining of instructions is possible, considering single clock cycle.	Pipelining is not possible.

5.7 INTRODUCTION TO HARDWIRED AND MICRO-PROGRAMMED CPU

- The control unit of CPU selects and interrupts program instructions.
- The basic task of the control unit is:
  1. For each instruction the control unit causes the CPU to go through a sequence of control steps.
  2. In each control step the control unit issues a set of signals which cause the corresponding micro-operations to be executed.
  3. The control unit is driven by the processor clock.
  4. The signals to be generated at a certain moment depend on:
     - The actual step to be executed.
     - The condition and status flags of the processor.
     - The actual instruction executed.
     - External signals received on the system bus (for example, interrupt signals).

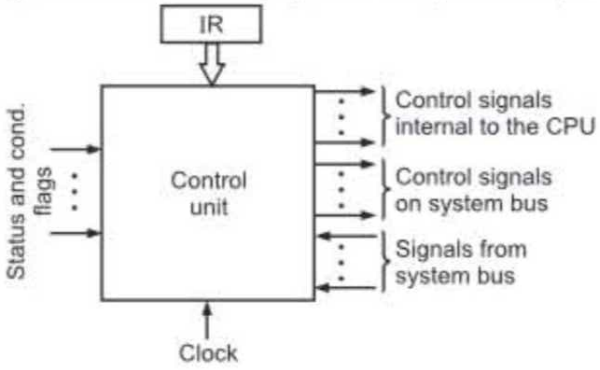


Fig. 5.8: Block diagram of a CU (Control Unit)

- Techniques for implementation of the control unit:
  1. Hardwired control, and
  2. Micro-programmed control.

5.7.1 Hardwired Control Unit

- In this case, the control unit is a combinatorial circuit, it gets a set of inputs (from IR, flags, clock, system bus) and transforms them into a set of control signals.
- Hardwired control unit is a synchronous sequential circuit that realizes all required control actions of the CPU i.e. all functions of the CU are implemented in hardware.
- With this method, the CPU is very fast but complex, very expensive and difficult to modify.
- Hardwired control provides highest speed.
- RISCs are implemented with hardwired control.

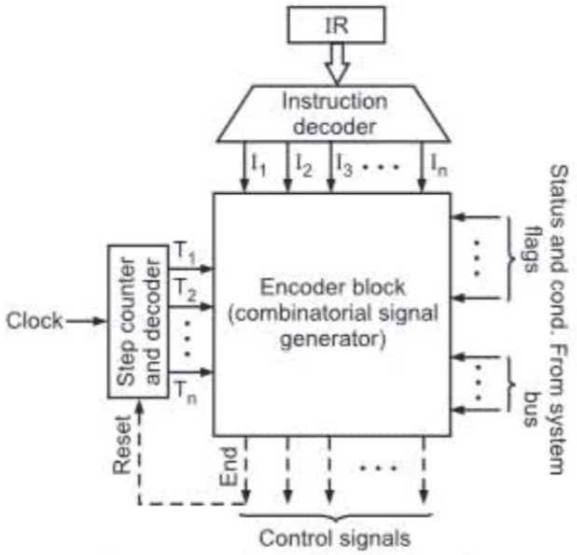


Fig. 5.9: Hardwired Control Unit

5.7.2 Micro-programmed Control Unit

- A micro-programmed control unit is implemented like another CPU inside the CPU. It executes micro-programs stored in the control store.
- A micro-program is also called as firm ware.
- In micro-programmed control unit all micro-routines corresponding to the machine instructions are stored in the control store.
- The control unit generates the sequence of control signals for a certain machine instruction by reading from the control store the CWs of the micro-routine corresponding to the respective instruction.

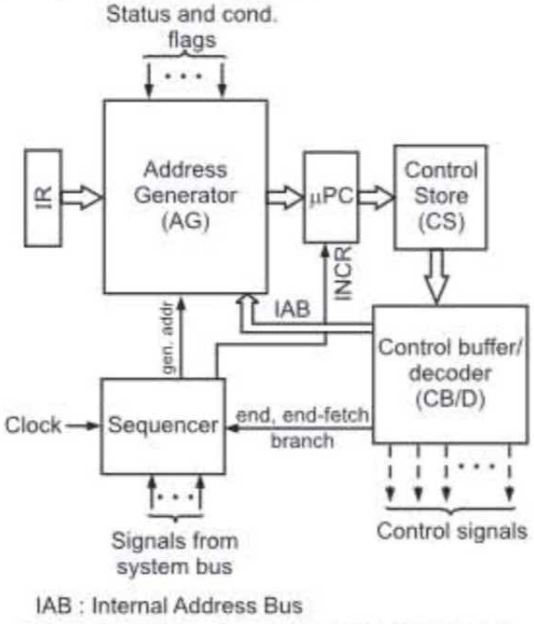


Fig. 5.10: Micro-Programmed Control Unit

- The control unit is implemented just like another very simple CPU, inside the CPU, executing micro-routines stored in the control store.
- **Control Word (CW):** A sequence of Nsig bits, where Nsig is the total number of control signals; each bit in CW corresponds to one control signal.
- Each control step during execution of an instruction defines a certain CW; it represents a combination of 1s and 0s corresponding to the active and non-active control signals.

Micro-routine:

- A sequence of CWs corresponding to the control sequence of a machine instruction. An individual CW in a micro-routine is called a Microinstruction.
- The control store contains the micro-program, sometimes called Firmware.

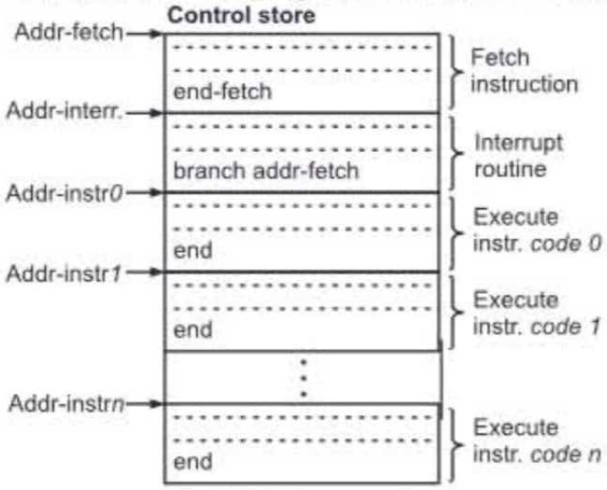


Fig. 5.11: Micro-routine

5.8 CONCEPT OF MEMORY

- The computer memory is used for storing data (information).
- A single bit of memory can be obtained either by flip-flop, a capacitor or ferrite core and the capacity of register is defined in number of bits.
- A combination of the memory bits forms memory storage. Byte is a group of 8 bits. A word can be a single byte or a group of bytes.
- A memory organization uses different types of memories, having different parameters.
- Computer memory exhibits the widest range of type, technology, organisation, performance, cost etc. of any feature of computer system. There is no single technology to satisfy memory requirements for computer system.
- Therefore, a typical computer system is equipped with hierarchy of memory sub-system.
- A computer memory is mainly classified as internal and external memory.

5.8.1 Memory System Hierarchy [W-17, S-18]

- Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time.
- The Memory Hierarchy was developed based on a program behavior known as locality of references.
- The Fig. 5.12 below clearly demonstrates the different levels of memory hierarchy:

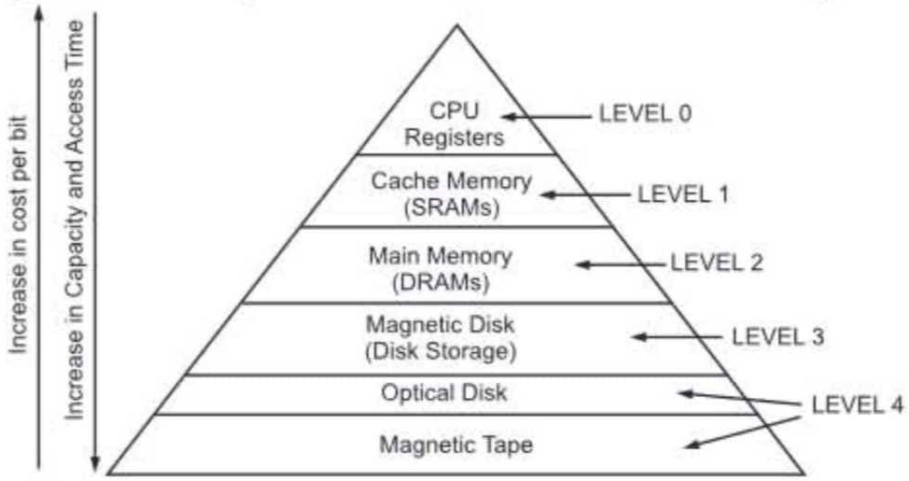


Fig. 5.12: Memory Hierarchy Design

Characteristics of Memory Hierarchy:

- Characteristics of Memory Hierarchy are following when we go from top to bottom.

1. **Capacity:**

- It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.

2. **Access Time:**

- It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.

3. **Performance:**

- Earlier when the computer system was designed without Memory Hierarchy design, the speed gap increases between the CPU registers and Main Memory due to large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.

4. **Cost per bit:**

- As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

5.8.2 Memory Access Methods

- Each memory type, is a collection of numerous memory locations. To access data from any memory, first it must be located and then the data is read from the memory location. Following are the methods to access information from memory locations:
  1. **Random Access**: Main memories are random access memories, in which each memory location has a unique address. Using this unique address any memory location can be reached in the same amount of time in any order.
  2. **Sequential Access**: This methods allows memory access in a sequence or in order.
  3. **Direct Access**: In this mode, information is stored in tracks, with each track having a separate read/write head.

5.8.3 Types of Memory [S-17]

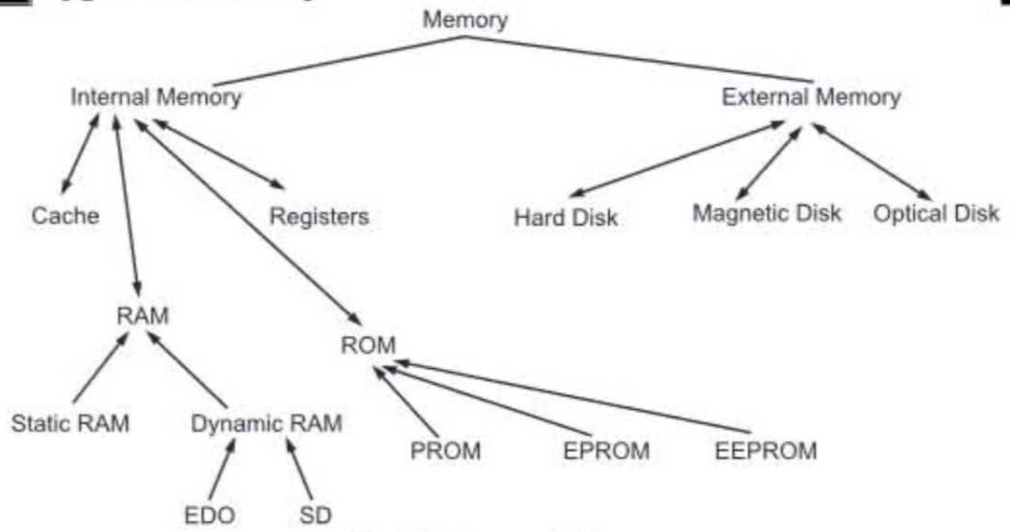


Fig. 5.13: Types of Memory

- Memory is primarily of two types:
  1. **Internal Memory or Primary Memory**: Cache memory and primary/main memory
  2. **External Memory or Secondary Memory** : Magnetic disk / optical disk etc.

5.8.3.1 Internal Memory [S-17, W-17, S-18]

1. Main Memory:

- Internal Memory is classified into main memory, cache memory.
- The memory unit that communicates directly within the CPU, Auxillary memory and Cache memory, is called main memory. It is the central storage unit of the computer system. It is a large and fast memory used to store data during computer operations. Main memory is made up of **RAM** and **ROM**, with RAM integrated circuit chips holing the major share.
- **RAM (Random Access Memory):** Following are different types of RAM.

- **DRAM:** Dynamic RAM, is made of capacitors and transistors, and must be refreshed every 10 ~ 100 ms. It is slower and cheaper than SRAM.
- **SRAM:** Static RAM, has a six transistor circuit in each cell and retains data, until powered off.
- **NVRAM:** Non-Volatile RAM, retains its data, even when turned off. Example: Flash memory.

- **ROM:** Read Only Memory, is non-volatile and is more like a permanent storage for information. It also stores the **bootstrap loader** program, to load and start the operating system when computer is turned on. **PROM**(Programmable ROM), **EPROM**(Erasable PROM) and **EEPROM**(Electrically Erasable PROM) are some commonly used ROMs.

5.8.3.2 Cache Memory [S-18, W-17]

- A Cache (Pronounced as "cash") is a small and very fast temporary storage memory.
- It is designed to speed up the transfer of data and instructions. It is located inside or close to the CPU chip.
- The data or contents of the main memory that are used again and again by CPU, are stored in the cache memory so that we can easily access that data in shorter time.
- Whenever the CPU needs to access memory, it first checks the cache memory. If the data is not found in cache memory then the CPU moves onto the main memory.
- It also transfers block of recent data into the cache and keeps on deleting the old data in cache to accommodate the new one.

5.8.3.2.1 Levels of Cache Memory

- A computer can have several different levels of cache memory. The level numbers refers to distance from CPU where Level 1 is the closest. All levels of cache memory are faster than RAM. The cache closest to CPU is always faster but generally costs more and stores less data then other level of cache.

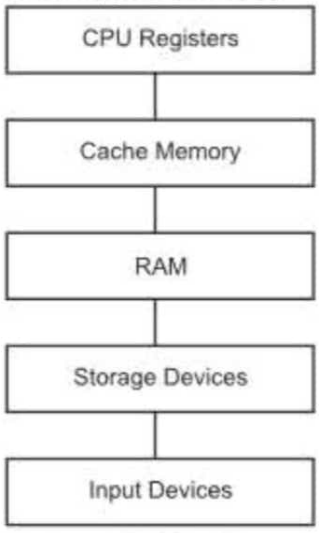


Fig. 5.14: Levels of Cache Memory

- **Level 1 or Register:** It is a type of memory in which data is stored and accepted that are immediately stored in CPU. The Instructions that are required by the CPU that are firstly searched in L1 Cache. Most commonly used register is accumulator, Program counter, address register etc. It has small capacity from 8 Km to 128 Kb.
- **Level 2 or Cache memory:** It is the fastest memory which has faster access time where data is temporarily stored for faster access. After searching the Instructions in L1 Cache, if not found then it searched into L2 cache by computer microprocessor. The high-speed system bus interconnecting the cache to the microprocessor. The common size of this cache is from 512 kb to 8 Mb.
- **Level 3 or Main Memory:** It is memory on which computer works currently. once power is off data no longer stays in this memory. The L3 cache is larger in size but also slower in speed than L1 and L2, it's size is between 1MB to 8MB.In Multicore processors, each core may have separate L1 and L2,but all core share a common L3 cache. L3 cache double speed than the RAM.
- **Level 4 or Secondary Memory:** It is external memory which is not as fast as main memory but data stays permanently in this memory.

5.8.3.2.2 Cache Performance

- When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.
  - If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from cache.
  - If the processor does not find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.
  - The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

    Hit ratio = hit / (hit + miss) = No. of hits/total accesses
- We can improve Cache Performance using higher cache block size, higher associability, reduce miss rate, reduce miss penalty, and reduce the time to hit in the cache.

5.8.4 External Memory [S-17]

(i) Auxiliary Memory:

- Devices that provide backup storage are called auxiliary memory.
- For example: Magnetic disks and tapes are commonly used auxiliary devices. Other devices used as auxiliary memory are magnetic drums, magnetic bubble memory and optical disks.
- It is not directly accessible to the CPU, and is accessed using the Input/Output channels.

(ii) Magnetic Disks:

- A magnetic disk, is a circular plate constructed of metal or plastic coated with magnetized material.
- Often, both sides of the disk are used and several disks may be stacked on one spindle with read/write heads available on each surface.
- All disks rotate together at high speed. Bits are stored in the magnetized. Surface in spots along concentric circles called tracks. The tracks are commonly divided into sections called sectors.

(iii) Magnetic Tapes:

- A magnetic tape is a medium of magnetic recording, made of a thin magnetized coating on a long, narrow strip of plastic film.
- Bits are recorded as magnetic spots on the tape along several tracks. Magnetic tapes can be stopped, started to move forward or in reverse. Read/Write heads are mounted one in each track, so that data can be recorded and read as a sequence of characters.

5.8.5 Virtual Memory

1. A virtual memory system provides a technique for translating program generated addresses into correct main memory locations.
2. Virtual memory is not a storage unit, it's a technique.
3. Virtual memory is used to give programmers the illusion that they have a very large memory even though the computer has a small main memory.
4. Virtual memory is the separation of logical memory from physical memory. This separation provides large virtual memory for programmers when only small physical memory is available.
5. In virtual memory system, programs and their data are divided into smaller pieces called "Pages". At the point where more memory is needed the operating systems decides which pages are least likely to be needed soon and writes these pages out to disk.
6. The memory space that they used is now available to the rest of the system for other application programs. When these pages are needed again, they are loaded back into real memory displacing other pages.
7. In modern computer system, the physical main memory is not a large as the address space spanned by address issued by the processor.
8. When a program does not completely fit into main memory, the parts of it not currently being executed are stored on secondary storage devices like a magnetic disk.
9. When a new segment of program is to be moved into a full memory, it must replace another segment already in memory.
10. In the modern system, the operating system moves programs and data automatically between main memory and secondary stage.
11. The conversion of the virtual or logical address into the physical address is done by a special hardware unit known as MMU (Memory Management Unit). When the required data are found in the main memory, then they are fetched to the cache memory for further processing.

12. If required data are not found in the main memory, the MMU with the help of operating system brought the data from secondary storage to main memory. This transfer of data from secondary memory to primary memory is performed using DMA (Direct Memory Access). It allows accessing the main memory by certain hardware system. This whole process is independent of the Central Processing Unit (CPU).

Difference between Virtual Memory and Cache Memory:

Sr. No.	Virtual Memory	Cache Memory
1.	Virtual memory increases the capacity of main memory.	While cache memory increase the accessing speed of CPU.
2.	Virtual memory is not a memory unit, its a technique.	Cache memory is exactly a memory unit.
3.	The size of virtual memory is greater than the cache memory.	While the size of cache memory is less than the virtual memory.
4.	Operating System manages the Virtual memory.	On the other hand hardware manages the cache memory.
5.	In virtual memory, The program with size larger than the main memory are executed.	While in cache memory, recently used data is copied into.

5.9 INPUT/OUTPUT BUS AND INTERFACE MODULES

- A typical communication link between the processor and several peripherals is shown in Fig. 5.15. The input bus consists of data lines, address line and control lines.
- Each peripheral device has associated with it an interface unit. Each interface decodes the address and control received from the I/O bus, interprets them for the peripheral, and provides signals for the peripheral controller. It also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller that operates the particular electromechanical device.
- A controller may be housed separately or may be physically integrated with the peripheral.
- To communicate with a particular device, the processor places a device address on the address lines. Each interface attached to the I/O bus contains an address decoder that monitors the address lines. When the interface detects its own address, it activates the path between the bus lines and the device that it controls. All peripherals whose address does not correspond to the address in the bus are disabled by their interface.

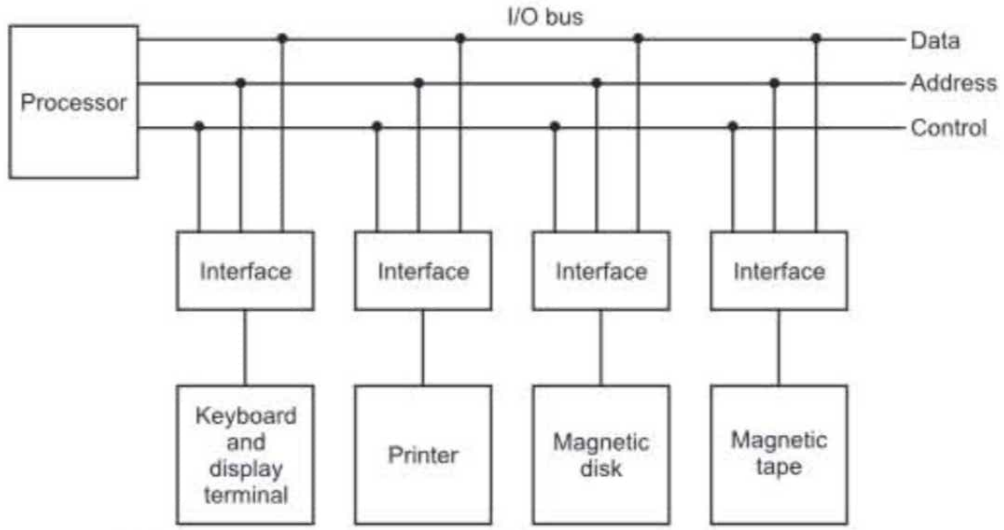


Fig. 5.15: Communication link between the processor and several peripherals

- At the same time that the address is made available in the address lines, the processor provides a function code in the control lines. The interface selected responds to the function code and proceeds to execute it.

5.9.1 Types of I/O Data Transfers [W-17]

- Binary information received from an external device is usually stored in memory for later processing.
- Data transfer between the central computer and I/O devices may be handled in a variety of modes. Some modes use the CPU as an intermediate path; others transfer the data directly to and from the memory unit. Data transfer to and from peripherals may be handled in one of three possible modes:
  1. CPU initiated
  2. Interrupt-imitated I/O
  3. Direct Memory Access (DMA)

1. **CPU initiated:** Each data item transfer is initiated by an instruction written in the program. Usually, the transfer is to and from a CPU register and peripheral. Other instructions are needed to transfer the data to and from CPU and memory.

   Transferring data under program control requires constant monitoring of the peripheral by the CPU. Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made. It is up to the programmed instructions executed in the CPU to keep close tabs on everything that is taking place in the interface unit and the I/O device.

2. **Interrupt-initiated I/O:** In this method, the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it keeps the processor busy needlessly. It can be avoided by using an interrupt facility and special commands to inform the interface to issue an interrupt request signal when the data are available from the device.

In the meantime the CPU can proceed to execute another program and the interface keeps monitoring the device. When the interface determines that the device is ready for data transfer, it generates an interrupt request to the computer. Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, braches to a service program to process the I/O transfer, and then returns to the task it was originally performing.

3. **DMA:** In Direct Memory Access(DMA), the interface transfers data into and out of the memory unit through the memory bus. The CPU initiates the transfer by supplying the interface with the starting, address and the number of words needed to be transferred and then proceeds to execute other tasks. When the transfer is made, the DMA requests memory cycles through the memory bus. When the request is granted by the memory controller, the DMA transfers the data directly into memory. CPU is not involved in data transfer.

5.9.2 Need of I/O Interfaces

1. Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.
2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU and consequently, a synchronization mechanism may be needed.
3. Data codes and formats in peripherals differ from the word format in the CPU and memory.
4. The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

5.9.3 Parallel and Serial Communication [S-17, 18, W-17]

What is data transmission?

- Data transmission refers to the process of transferring data between two or more digital devices. Data is transmitted from one device to another in analog or digital format. Basically, data transmission enables devices or components within devices to speak to each other.

How does data transmission work between digital devices?

- Data is transferred in the form of bits between two or more digital devices. There are two methods used to transmit data between digital devices: serial transmission and parallel transmission. Serial data transmission sends data bits one after another over a single channel. Parallel data transmission sends multiple data bits at the same time over multiple channels.

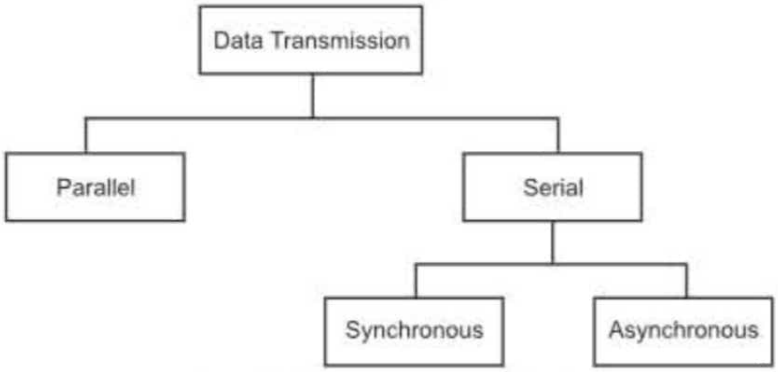


Fig. 5.16: Data Transmission Types

5.9.3.1 Parallel Transmission

- **Definition:** Within a computing or communication device, the distances between different subunits are too short. Thus, it is normal practice to transfer data between subunits using a separate wire to carry each bit of data. There are multiple wires connecting each sub-unit and data is exchanged using a parallel transfer mode. This mode of operation results in minimal delays in transferring each word.
- In parallel transmission, all the bits of data are transmitted simultaneously on separate communication lines.
- In order to transmit n bits, n wires or lines are used. Thus each bit has its own line.
- All n bits of one group are transmitted with each clock pulse from one device to another i.e. multiple bits are sent with each clock pulse.
- Parallel transmission is used for short distance communication.
- As shown in the fig, eight separate wires are used to transmit 8 bit data from sender to receiver.

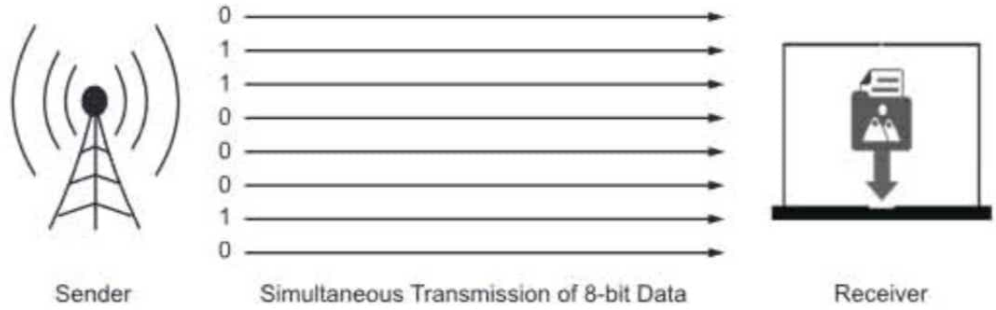


Fig. 5.17: Parallel Transmission

Advantage of Parallel Transmission:

1. It is speedy way of transmitting data as multiple bits are transmitted simultaneously with a single clock pulse.

Disadvantage of Parallel Transmission:

2. It is costly method of data transmission as it requires n lines to transmit n bits at the same time.

When is parallel transmission used to send data?

- Parallel transmission is used when:
  - A large amount of data is being sent;
  - The data being sent is time-sensitive;
  - And the data needs to be sent quickly.
- A scenario where parallel transmission is used to send data is video streaming. When a video is streamed to a viewer, bits need to be received quickly to prevent a video pausing or buffering. Video streaming also requires the transmission of large volumes of data. The data being sent is also time-sensitive as slow data streams result in poor viewer experience.

5.9.3.2 Serial Transmission

- **Definition:** When transferring data between two physically separate devices, especially if the separation is more than a few kilometers, for reasons of cost, it is more economical to use a single pair of lines. Data is transmitted as a single bit at a time using a fixed time interval for each bit. This mode of transmission is known as bit-serial transmission.
- In serial transmission, the various bits of data are transmitted serially one after the other.
- It requires only one communication line rather than n lines to transmit data from sender to receiver.
- Thus all the bits of data are transmitted on single line in serial fashion.
- In serial transmission, only single bit is sent with each clock pulse.
- As shown in fig., suppose an 8-bit data 11001010 is to be sent from source to destination. Then least significant bit (LSB) i,e. 0 will be transmitted first followed by other bits. The most significant bit (MSB) i.e. 1 will be transmitted in the end via single communication line.
- The internal circuitry of computer transmits data in parallel fashion. So in order to change this parallel data into serial data, conversion devices are used.
- These conversion devices convert the parallel data into serial data at the sender side so that it can be transmitted over single line.
- On receiver side, serial data received is again converted to parallel form so that the interval circuitry of computer can accept it.

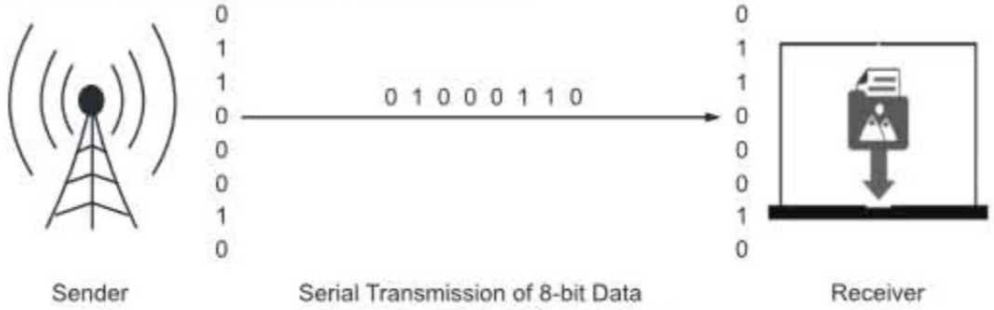


Fig. 5.18: Serial Transmission

- Serial transmission is used for long distance communication.

Advantage of Serial transmission:

1. Use of single communication line reduces the transmission line cost by the factor of n as compared to parallel transmission.

Disadvantages of Serial transmission:

1. Use of conversion devices at source and destination end may lead to increase in overall transmission cost.
2. This method is slower as compared to parallel transmission as bits are transmitted serially one after the other.

When is serial transmission used to send data?

- Serial transmission is normally used for long-distance data transfer. It is also used in cases where the amount of data being sent is relatively small. It ensures that data integrity is maintained as it transmits the data bits in a specific order, one after another. In this way, data bits are received in-sync with one another.

Types of Serial Transmission:

- There are two types of serial transmission-synchronous and asynchronous both these transmissions use 'Bit synchronization'
- Bit Synchronization is a function that is required to determine when the beginning and end of the data transmission occurs.
- Bit synchronization helps the receiving computer to know when data begin and end during a transmission. Therefore bit synchronization provides timing control.
  1. Asynchronous serial Transmission
  2. Synchronous serial Transmission

1. Asynchronous Transmission:

- Asynchronous transmission sends only one character at a time where a character is either a letter of the alphabet or number or control character i.e. it sends one byte of data at a time.
- Bit synchronization between two devices is made possible using start bit and stop bit.
- Start bit indicates the beginning of data i.e. alerts the receiver to the arrival of new group of bits. A start bit usually 0 is added to the beginning of each byte.
- Stop bit indicates the end of data i.e. to let the receiver know that byte is finished, one or more additional bits are appended to the end of the byte. These bits, usually 1s are called stop bits.

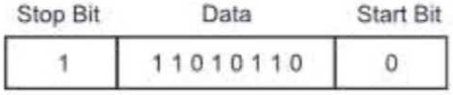


Fig. 5.19: Asynchronous transmission

- Addition of start and stop increase the number of data bits. Hence more bandwidth is consumed in asynchronous transmission.
- There is idle time between the transmissions of different data bytes. This idle time is also known as Gap.
- The gap or idle time can be of varying intervals. This mechanism is called Asynchronous, because at byte level sender and receiver need not to be synchronized. But within each byte, receiver must be synchronized with the incoming bit stream.

Advantages of Asynchronous transmission:

1. This method of data transmission is cheaper in cost as compared to synchronous e.g. If lines are short, asynchronous transmission is better, because line cost would be low and idle time will not be expensive.
2. In this approach each individual character is complete in itself, therefore if character is corrupted during transmission, its successor and predecessor character will not be affected.
3. It is possible to transmit signals from sources having different bit rates.
4. The transmission can start as soon as data byte to be transmitted becomes available.
5. Moreover, this mode of data transmission in easy to implement.

Disadvantages of Asynchronous transmission:

1. This method is less efficient and slower than synchronous transmission due to the overhead of extra bits and insertion of gaps into bit stream.
2. Successful transmission inevitably depends on the recognition of the start bits. These bits can be missed or corrupted.

Applications of Asynchronous Transmission:

- Emails
- Forums
- Letters
- Radios
- Televisions

2. Synchronous Transmission:

- Synchronous transmission does not use start and stop bits.
- In this method bit stream is combined into longer frames that may contain multiple bytes.
- There is no gap between the various bytes in the data stream.

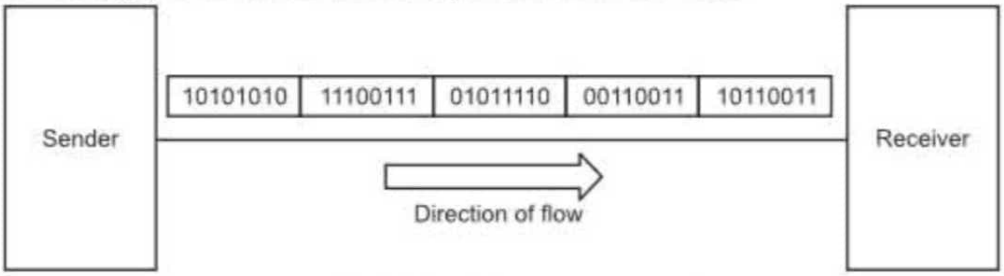


Fig. 5.20: Synchronous transmission

- In the absence of start & stop bits, bit synchronization is established between sender and receiver by 'timing' the transmission of each bit.
- Since the various bytes are placed on the link without any gap, it is the responsibility of receiver to separate the bit stream into bytes so as to reconstruct the original information.
- In order to receive the data error free, the receiver and sender operates at the same clock frequency.

Advantage of Synchronous transmission:

1. This method is faster as compared to asynchronous as there are no extra bits (start bit & stop bit) and also there is no gap between the individual data bytes.

Disadvantages of Synchronous transmission:

1. It is costly as compared to asynchronous method. It requires local buffer storage at the two ends of line to assemble blocks and it also requires accurately synchronized clocks at both ends. This lead to increase in the cost.
2. The sender and receiver have to operate at the same clock frequency. This requires proper synchronization which makes the system complicated.

Applications of Synchronous Transmission:

- Chatrooms
- Video conferencing
- Telephonic conversations
- Face-to-face interactions.

Serial Communication Terminologies:

- There are many terminologies, or 'keywords' associated with serial communication. We will discuss all of them one by one:
  1. **MSB/LSB:** MSB stands for Most Significant Bit and LSB stands for Least Significant Bit). Since data is transferred bit-by-bit in serial communication, one needs to know which bit is sent out first: MSB or LSB.
  2. **Simplex Communication:** In this mode of serial communication, data can only be transferred from transmitter to receiver and not vice versa. Examples: radio and television broadcasting.
  3. **Half Duplex Communication:** Half duplex means that data transmission can occur in only one direction at a time, i.e. either from master to slave, or slave to master, but not both. Examples :Walky-talky.

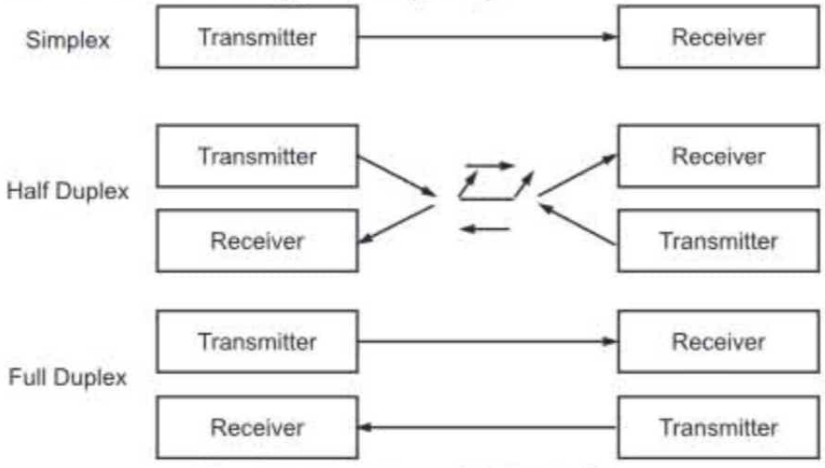


Fig. 5.21: Data Transmission Modes

4. **Full Duplex Communication:** full duplex communication means that data can be transmitted from the master to the slave, and from slave to the master as the same time. Examples: Telephone communication.
5. **Baud Rate:** Baud is synonymous to symbols per second or pulses per second. It is the unit of symbol rate, also known as baud or modulation rate. However, though technically incorrect, in the case of modem manufacturers baud commonly refers to bits per second.

Comparison between Serial and Parallel Transmission:

Sr. No.	Basis for Comparison	Serial Transmission	Parallel Transmission
1.	**Definition**	Data flows in 2 directions, bit by bit.	Data flows in multiple directions, 8 bits (1 byte) at a time.
2.	**Cost**	Economical.	Expensive.
3.	**Number of bits transferred per clock pulse**	1 bit.	8 bits or 1 byte.
4.	**Speed**	Slow.	Fast.
5.	**Applications**	Used for long distance communication.	Used for short distance communication.
6.	**Example**	Computer to computer.	Computer to printer.

Synchronous and Asynchronous Transmission:

Sr. No.	Point of Comparison	Synchronous Transmission	Asynchronous Transmission
1.	**Definition**	Transmits data in the form of chunks or frames.	Transmits 1 byte or character at a time.
2.	**Speed of Transmission**	Quick.	Slow.
3.	**Cost**	Expensive.	Cost-effective.
4.	**Time Interval**	Constant.	Random.
5.	**Gaps between the data?**	Yes.	No.
6.	**Examples**	Chat Rooms, Telephonic Conversations, Video Conferencing.	Email, Forums, Letters.

Summary

- The central processing unit of any digital computer comprises three basic elements: Arithmetic and Logic Unit (ALU), Main Memory, Control Unit (CU)
- The basic unit of main memory storage is called an accumulator. Accumulator can be considered to be a set of numbered locations.
- Registers are special purpose temporary storage locations which are quite separate from the locations in main storage (i.e. accumulator) although they can be similar In structure.
- The organization of the CPU is based on the functions it performs, which are to fetch an instruction, decode it and execute it.
- Memory is used for storing programs and data that are required to perform a specific task.
- Memory hierarchy affects performance in computer architectural design, algorithm predictions, and lower level programming constructs involving locality of reference.
- Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. The main memory holds the data and the programs that are needed by the CPU. Auxillary memory is basically used for storing the programs that are not needed in the main memory.
- Internal memory generally refers to chips rather than disks or tapes. Typically refers to main memory (RAM), but may also refer to ROM and flash memory.
- External memory refers to storage in an external hard drive or on the Internet. Portable storage can range from a portable flash drive, hard drive or a memory card that is used in a device such as a camera. Using external memory is a good way to keep files such as pictures, videos and other types of files in a safe place.
- Virtual memory is a valuable concept in computer architecture that allows you to run large, sophisticated programs on a computer even if it has a relatively small amount of RAM.
- In computer system, there are special hardware components between the CPU and peripherals to control or manage the input-output transfers. These components are called input-output interface units.
- There are three methods of I/O data transfer: 1: Program controlled or CPU initiated data transfer 2.Interrupt controlled /device initiated data transfer 3. DMA controlled data transfer.
- A DMA controller can generate memory addresses and initiate memory read or write cycles.
- The transfer of data between two units is serial or parallel. In parallel data transmission, n bit in the message must be transmitted through n separate conductor path. In serial transmission, each bit in the message is sent in sequence one at a time.
- In Asynchronous serial transfer, each bit of message is sent a sequence at a time, and binary information is transferred only when it is available. When there is no information to be transferred, line remains idle.
- In Interrupt-Initiated I/O method an interrupt facility an interrupt command is used to inform the device about the start and end of transfer.

Practice Questions

Q.1 Answer the following Question in brief:

1. What is CPU and its components?
2. What is main memory? Enlist its types.
3. What is RAM? Enlist its types.
4. What is ROM? Enlist its types.
5. What is meant by internal memory and external memory?
6. What is magnetic disk? Explain its types.
7. What is meant by serial communication?

Q.2 Answer the following Question:

1. With the help of diagram describe functions of CPU.
2. With the help of diagram describe block diagram of ALU.
3. Explain memory representation diagrammatically.
4. Describe memory hierarchy in detail.
5. Enlist characteristics of memory system.
6. What is cache memory? Describe its organisation.
7. Explain virtual memory in detail.
8. With the help of diagram describe synchronous and asynchronous serial communication.
9. Explain CPU initiated transfer mode in detail.
10. Explain synchronous and asynchronous data transfer.
11. Write short note on: RISC and CISC processors.

Q.3 Define Terms:

ALU, CU, CPU, Memory, Cache hit, Cache Miss, Hit Ratio, Cache Hit-Time, Parity Flag, Carry Flag.

Check Your Understanding

1. _____ is also called auxiliary storage.
   (a) secondary memory (b) tertiary memory
   (c) primary memory (d) cache memory
2. The ROM chips are mainly used to store _____.
   (a) System files (b) Root directories
   (c) Boot files (d) Driver files
3. The main aim of virtual memory organisation is _____.
   (a) To provide effective memory access
   (b) To provide better memory transfer
   (c) To improve the execution of the program
   (d) All of the mentioned

4. Data transmission between the keyboard and the computer is usually _____.
   (a) Parallel (b) Synchronous
   (c) Asynchronous (d) Virtual
5. What is the high speed memory between the main memory and the CPU called?
   (a) Register memory (b) Cache memory
   (c) Storage memory (d) Virtual memory

Answers

1. (a)	2. (c)	3. (d)	4. (c)	5. (c)

Previous Exam Questions

Winter 2017

1. Secondary storage is also known as _____ **(1 M)**
   (a) External memory (b) Auxiliary memory
   (c) Cache memory (d) Both (a) and (b)

Ans. Refer to Section 5.8.3.1.

2. What is address bus? **(1 M)**

Ans. Refer to Section 5.2.

3. What is burst mode of DMA transfer? **(1 M)**

Ans. Refer to Section 5.9.1.

4. What is the use of cache? Explain the terms hit and miss. **(3 M)**

Ans. Refer to Section 5.8.3.2.

5. Write a neat block diagram and explain the signals in a DMA controller. **(5 M)**

Ans. Refer to Section 5.9.1.

6. Write a short note on memory hierarchy. **(4 M)**

Ans. Refer to Section 5.8.1.

7. What is a flag register? Give its structure with explanation. **(4 M)**

Ans. Refer to Section 5.5.

8. Distinguish between serial and parallel data transfer. **(3 M)**

Ans. Refer to Section 5.9.3.

9. What is internal memory? Explain Processor registers and Processor cache. **(3 M)**

Ans. Refer to Section 5.8.3.

Summer 2017

1. EPROM is _____ **(1 M)**
   (a) Ejectable Program read Only Memory
   (b) Erasable Programmable Read Only Memory
   (c) Erasable Perishable Read Only Memory
   (d) Erasable Programmable Reject Only Memory

Ans. Refer to Section 5.8.3.1.

2. List the types of buses. **(1 M)**

Ans. Refer to Section 5.2.

3. Write or give the memory classification. **(3 M)**

Ans. Refer to Section 5.8.3.

4. Explain the functions of the CPU. **(4 M)**

Ans. Refer to Section 5.3.

5. Explain external memory in short. **(3 M)**

Ans. Refer to Section 5.8.4.

Summer 2018

1. Which of the following is not in memory hierarchy? **(1 M)**

(a) RAM (b) ROM

(c) PROM (d) None of the above

Ans. Refer to Section 5.8.3.1.

2. Define the term: Cache Memory. **(1 M)**

Ans. Refer to Section 5.8.3.2.

3. Write a short note on memory hierarchy. **(5 M)**

Ans. Refer to Section 5.8.1.

4. Differentiate between RISC and CISC. **(4 M)**

Ans. Refer to Section 5.6.

5. What are the different ways of serial data transfer? Explain the typical frame format of synchronous and asynchronous data. **(4 M)**

Ans. Refer to Section 5.9.3.

6. Distinguish between serial and parallel data transfer. **(3 M)**

Ans. Refer to Section 5.9.3.

7. Mention the functions of a CPU. **(3 M)**

Ans. Refer to Section 5.3.

■■■

6...

Introduction to Microprocessors and Microcontrollers

Objectives...

- To describe the basic organization of Intel Pentium Processor.
- To learn the basic concepts of Microprocessors such as Pipeline and Parallelism.
- To know about Microcontrollers and Multicore processors.

6.1 INTRODUCTION [S-17]

- The microprocessor is a multipurpose, programmable and clock driven IC which is capable of performing arithmetic and logical operations.
- The microprocessor has a digital circuit for data handling and computation under program control. The microprocessor is a data processing unit. Data processing includes both computation and data handling.
- Microprocessors are classified according to their word-length such as 8-bit, 16-bit, 32-bit and 64-bit microprocessors.
- Microprocessor ICs are programmable so that instructions can be executed by a microprocessor to perform given tasks within its capability.
- The basic functional block diagram of a microprocessor is shown in Fig. 6.1.
- The basic functional blocks of a microprocessor are ALU, Flag Register, Register Array, Program Counter (PC)/Instruction Pointer (IP), Instruction Decoding Unit, Timing and Control Unit.

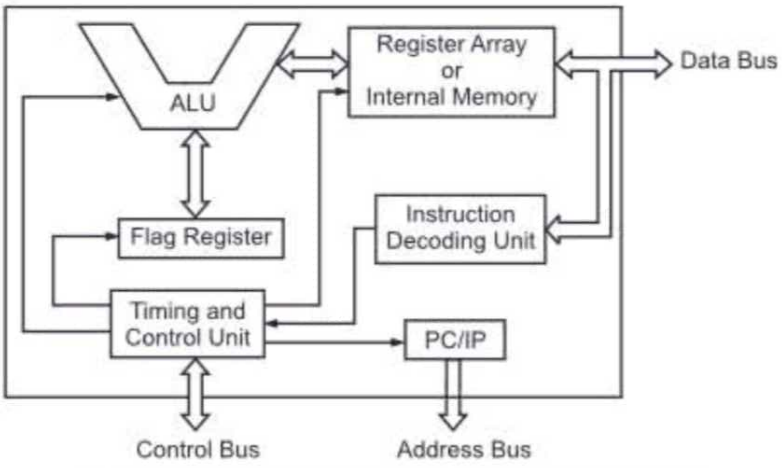


Fig. 6.1: Basic Functional Block of a Microprocessor

- Various parts of microprocessor and its working are explained below:
  1. **ALU (Arithmetical and Logical Unit)** is the computational unit of the microprocessor which performs arithmetic and logical operations on binary data.
  2. The various conditions of the result are stored as status bits called flags in the **flag register**.
  3. The **Register Array** is the internal storage device and so it is also called internal memory. The input data for ALU, the output data of ALU (result of computations) and any other binary information needed for processing are stored in the register array.
  4. **Program counter (PC)** generates the address of the instructions to be fetched from the memory and send through address bus to the memory.
  5. The memory will send the instruction codes and data through the **data bus**. Then instruction codes are decoded by the decoding unit and send information to timing and control unit. The data is stored in the register array for processing by ALU.
  6. **Control unit** will generate the necessary control signals for internal and external operations of the microprocessor.

Advantages of a Microprocessor:

- **High Speed:** Microprocessor chips can work at very high speed due to the technology involved in it. It is capable of executing millions of instructions per second.
- **Small Size:** Due to very large scale and ultra large scale integration technology, a microprocessor is fabricated in a very less footprint. This will reduce the size of the entire computer system.
- **Versatile:** Microprocessors are very versatile, the same chip can be used for a number of applications by simply changing the program (instructions stored in the memory).
- **Low Power Consumption:** Microprocessors are usually manufactured using metal oxide semiconductor technology, in which MOSFETs (Metal Oxide Semiconductor Field Effect Transistors) are working in saturation and cut-off modes. So the power consumption is very low.

- **Less Heat Generation:** Compared to vacuum tube devices, semiconductor devices would not emit that much heat.
- **Reliable:** Microprocessors are very reliable, failure rate is very less as semiconductor technology is used.
- **Portable:** Devices or computer system made with micro-processors can be made portable due to the small size and low power consumption.
- **Low Cost:** Microprocessors are available at low cost due to integrated circuit technology which will reduce the cost of a computer system.

Applications of Microprocessors:

1. **Instrumentation:**
   - Frequency counters.
   - Function generators.
   - Frequency synthesizers.
   - Spectrum analysers.
2. **Control:**

- Microprocessor based controllers are available in home appliances, such as microwave oven, washing machine etc.

3. **Communication:**

- Microprocessors are being used in a wide range of communication equipments.
- In telephone exchanges and modem etc.
- The use of microprocessor in television, satellite communication have made tele-conferencing possible.
- Railway reservation and air reservation system also uses this technology.

4. **Consumer:**
   - Calculators.
   - Accounting system.
   - Games machine.
   - Complex industrial controllers.
   - Traffic light control.
   - Data acquisition systems.
   - Multi user, multi-function environments.
   - Military applications.
   - Communication systems.

Introduction to Pentium Family:

1. 4004 = 4 bit processor
2. 8080, 8085 = 8 bit
3. 8086, 8088 = 16 bit
4. 80186 = 16 bit data bus and 20 bit address bus
5. 80286 = 24 bit address bus
6. 80386 = 32 bit processor (32 bit data + address bus)
7. 80486 = Improved version of 386
8. 80586 = Similar to 486, but with 64 bit data bus

9. Pentium Pro = 36 bit address bus, 3 way superscalar
10. Pentium II = MMX instructions
11. Pentium III, IV
12. Itanium processor = 64 bit processor, RISC design

6.2 BLOCK DIAGRAM OF PENTIUM

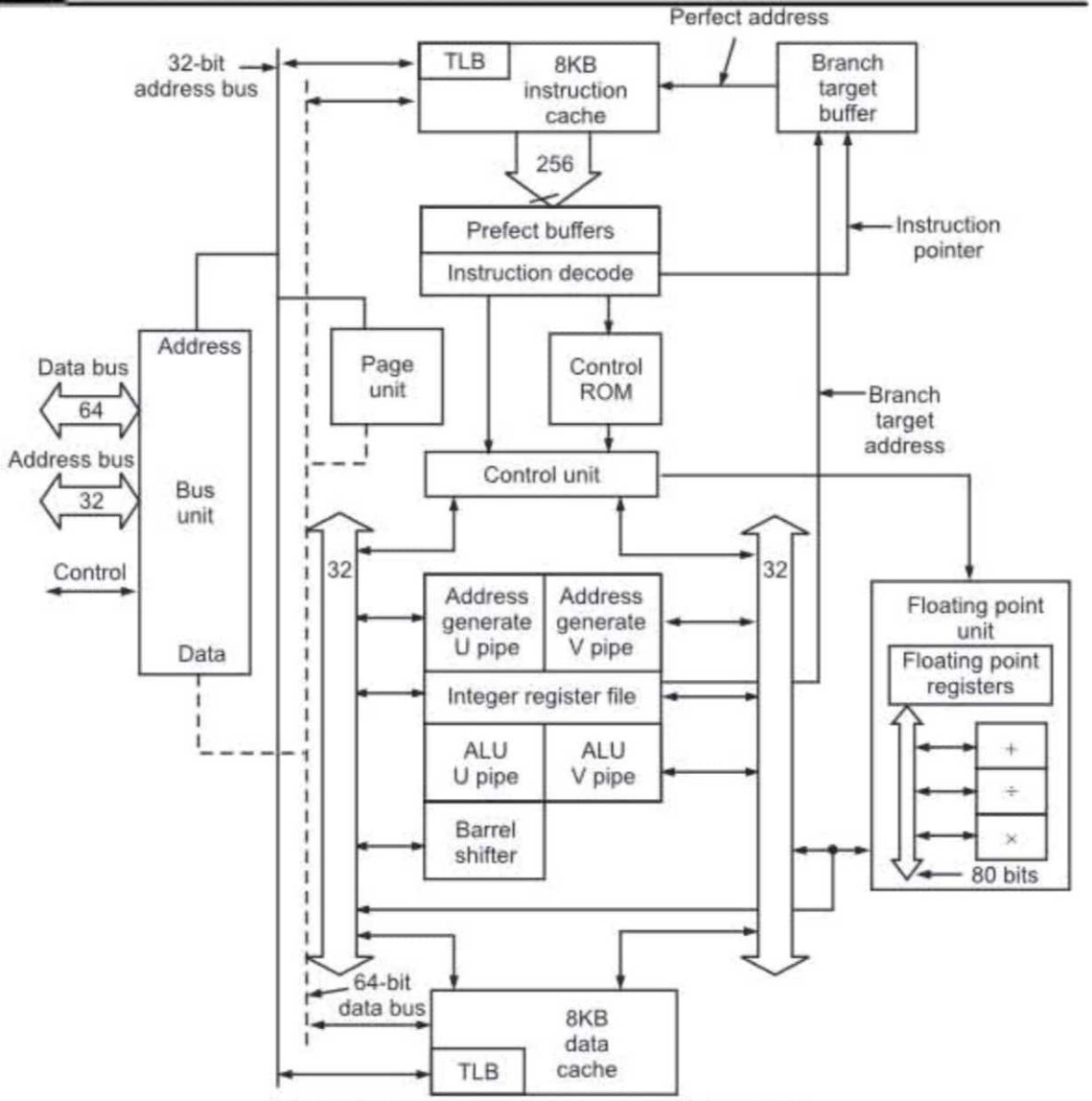


Fig. 6.2: Pentium Architecture Block Program

Architecture:

1. 64-bit bus.
2. Pentium processor uses superscalar architecture and hence can issue multiple instruction per cycle.

3. Complex Instruction Set Computer (CISC) architecture with Reduced Instruction Set Computer (RISC) performance.
4. Pentium processor executes instructions in five stages. This staging, or pipelining allows the processor to overlap multiple instructions so that it takes less time to execute two instructions in a row.
5. When data is modified, only the data in the code is changed. Memory data is changed only when the Pentium processor replaces the modified data in the cache with a different set of data.
6. Supports burst read and burst write back cycle.
7. Instruction cache.
8. Supports pipelining.

6.3 FUNCTIONAL UNITS

6.3.1 Integer Pipelines

- At the heart of the processor are the two integer pipelines, the U pipeline and the V pipeline. These pipelines are responsible for executing Pentium instructions.

U & V Pipeline:

U & V are parts of integer pipeline:

1. Pentium has 2 execution unit: U & V pipes.
2. The U pipe can execute by instruction in the Pentium architecture.
3. V pipe can execute only simple instruction.

- During execution, the U and V pipelines are capable of executing two integer instructions at the same time, under special condition, or one floating point instruction.

6.3.2 Floating Point Unit

- The floating point unit of Pentium processor maintains a set of floating point registers.
- It is designed to carry out operations on floating point numbers.
- Floating point unit carry out operations such as addition, subtraction, multiplication, division, square root, bit shifting, exponential or trigonometric calculations.
- It provides 80 bit precision which causes high speed math operators.
- Floating point unit has 8 stages of pipelining.
- The floating point unit uses U and V pipelines to perform the initial work during a floating point instruction such as fetching 64-bit operand.
- Then it uses its own pipeline to complete the operation.
- Since both integer pipelines are used, only one floating point instruction may be executed at a time.

6.3.3 Bus Unit

- It brings mixture of data and codes into the CPU.
- The Pentium communicates with the outside world via a 32-bit address bus and a 64 bit data bus. This bus unit is capable of performing **burst** reads and writes of 32 bytes to memory (Through bus cycle pipelining. It allows two bus cycles to be in progress simultaneously).

6.3.4 Instruction Cache and Data Cache

- The 8 KB instruction cache is used to provide quick access to frequently used instructions. Also the 8 KB data cache is used to provide quick access to data stored in cache. When an instruction is not found in the instruction cache it is read from the external memory. A copy is placed in instruction cache for future reference.

6.3.5 Branch Target Buffer and Branch Prediction

- The branch target buffer and prefetch buffers work together with the instruction cache to fetch instructions as fast as possible. The prefetch buffers maintain a copy of the next 32 bytes of prefetched instruction code. It can be loaded from the cache in a single clock cycle, due to 256 bit wide data output of the instruction cache.
- The Pentium uses a technique called branch prediction to maintain a steady flow of instructions into the pipeline. To support branch prediction, the branch target buffer maintains a copy of instructions in a different part of the program located at an address called the branch target.
- For example, for instruction CALL ABC the address of subroutine ABC is stored as branch target, as this instruction changes the sequence of program execution. So as and when the target address is needed, the branch target buffer feeds it to the instruction code.

6.3.6 Look Aside Buffer

- The memory accesses outside the processor are significantly longer than processor clock cycles. So it keeps a copy of memory data in a fast reading cache. The data and instruction caches can be enabled or disabled with hardware or software. Both caches use translation look aside buffer, which converts logical address into physical address when virtual memory is employed.

6.4 PROGRAMMING MODEL

- The Pentium microprocessor contains four data registers referred as EAX, EBX, ECX and EDX. These are also called as general purpose registers. All are 32 bit wide. The lower 16 bits of each register are called AX, BX, CX, DX and are split-up into two halves of 8 bits each. The Higher order 8-bits are called AH, BH, CH, DH and lower order 8-bits are called AL, BL, CL, DL.
- Five other 32-bits registers are available for use as pointer or index registers. These registers are Stack Pointer (ESP), Base Pointer (EBP), Source Index (ESI), Destination Index (EDI) and Instruction Pointer (EIP). These five registers are not divided as the data registers. Here E in a register is called as extended (32 bit wide). When the processor is in real mode only 16 bit wide registers AX, BX, CX, DX, BP, SI, DI, SP, IP are available.
- Six segment registers associated with six segments are used to access the segment memory. The code segment (CS) is used during instruction fetches, the data segment (DS) is used while reading and writing data, the stack segment (SS) is used during

stack operations such as subroutine call and return addresses. The Extra Segment (ES), FS and GS are used as supplements to CS, DS and SS if the space is insufficient for storage.

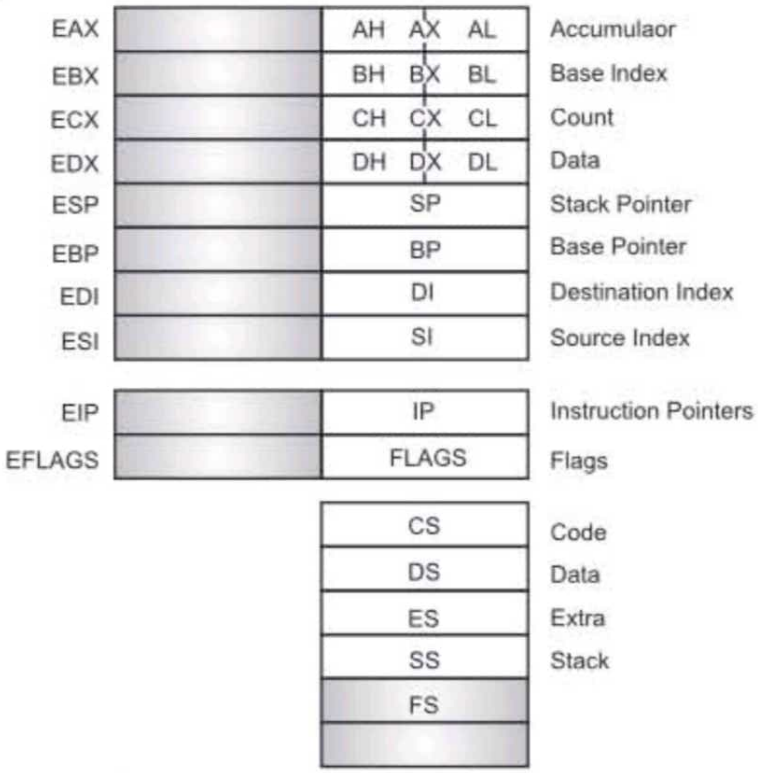


Fig. 6.3: Programming Model of Microprocessor

- The 32 bit flag registers or process status register is used to indicate the result of arithmetic and logical instructions in the form of setting or resetting flag.

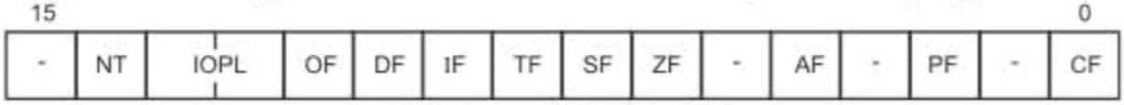


Fig. 6.4: Lower word of flag register

- The 16-bits of the flag register are divided into two groups the control flags and status flags. The control flags are IF (Interrupt Enable Flag), DF (Direction Flag), TF (Trap Flag). The status flags are CF (Carry Flag), PF (Parity Flag), AF (Auxiliary Carry Flag), ZF (Zero Flag), SF (Sign Flag), OF (Overflow Flag), NT (Nested Task) and IOPL (Input Output Privilege Level).
  - **CF:** Set when carry generated out of MSB of result.
  - **PF:** Set when result has even parity.
  - **AF:** Set when carry generated out of bit 3 in AL.
  - **ZF:** Set when result equals zero.
  - **SF:** Set when result is negative.
  - **OF:** Set when overflow occurred in result.

- **IF:** Set when interrupt request.
- **DF:** Set when pointer updates forward during string operation.
- **TF:** Set and allows single step capability for debugging.
- **IOPL:** Priority level of current task.
- **NT:** Sets and indicates if current task is nested.

- The upper 16 bits of the flag register are used for protected mode operation.

6.5 CONCEPT OF PARALLELISM [W-17]

- Parallelism is the process where several instructions are executed simultaneously. It reduces the total computational time.
- The computer system have multiple processors to handle various data from data memory like arithmetic operations and floating point operations.
- Such network of processors allows simultaneous processes to occur else the system is limited to perform specific task.
- Many processors present in a system implies many ALUs present thus many types of data can be handled simultaneously.
- A processor at a time can work with only a specific operation. Many processors perform simultaneous jobs.

Goals/Objectives of Parallelism:

- The purpose of parallel processing is to speed-up the computer processing capability or In other words, it increases the computational speed.
- Increases throughput i.e. amount of processing that can be accomplished during a given interval of time.
- Improves the performance of the computer for a given clock speed.
- The system may have two or more processors operating concurrently.

Types of Parallelism:

- Here are a variety of ways that parallel processor can be classified. It can be considered from the internal organization of the processor, from the interconnection structure between processors, or from the flow of information through the system.

1. Instruction-Level Parallelism (ILP):

- ILP is a measure of how many of the operations in a computer program can be performed simultaneously.
- Micro-architectural techniques that are used to develop ILP include:

(i) Instruction Pipelining:

- In this technique, the execution of multiple instructions can be partially overlapped.
- Pipelining is the concept of simultaneously fetching multiple instructions to be processed. While an execution unit is decoding and executing one instruction, the fetch unit is retrieving the next instruction(s) to be executed.
- This overlapping of the two units conceptually creates an assembly-line (or pipeline) of instructions so that the processor is constantly executing instructions without having to wait for them to be fetched.

(ii) Superscalar Execution:

- In this technique multiple execution units are used to execute multiple instructions in parallel. In typical superscalar processors, the instructions executing simultaneously are adjacent in the original program order.
- Superscalar processors have more than one unit of execution. Each unit has its own pipeline capabilities. But they are distinguished from scalar ones in that they are capable of executing more than one instruction at a time resulting in the execution of multiple instructions each clock cycle. This is accomplished by the processor looks for instructions that can be handled within the same clock cycle and processes them together.

2. Processor Level Parallelism (PLP):

- In Process/Processor Level Parallelism, different processes are executed on parallel processors or processor cores. This brings us to real multi-core and multi-processor systems.

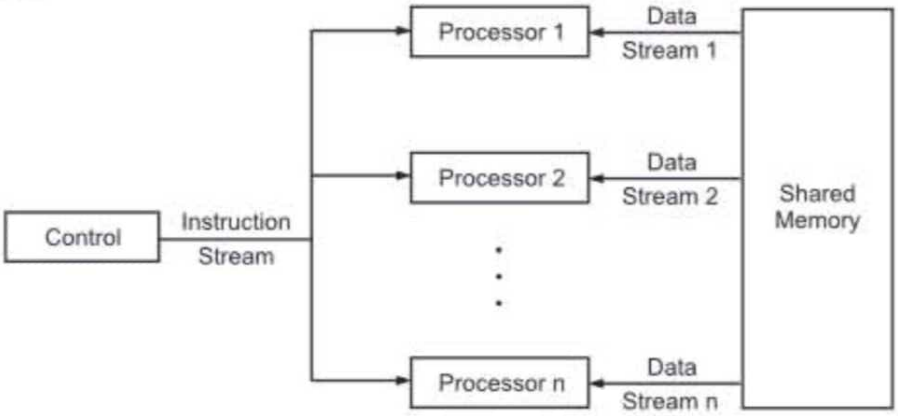


Fig. 6.5: Processor Level Parallelism

- **M.J. Flynn's Classification:** This concept based on new concept of Instruction stream and data stream. The sequence of instructions read from memory constitutes an *Instruction Stream*. The operations performed on the data in the processor constitute a *Data Stream*.

6.6 CONCEPT OF PIPELINE [S-18]

- Pipelining is a technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments.
- It allows storing and executing instructions in an orderly manner.
- A pipeline can be visualized as a collection of processing segments through which binary information flows.
- The pipeline name implies a flow of information analogous to an industrial assembly line wherein the task is subdivided into several subtasks and each subtask is performed by the stages(segments) in the pipeline.
- Pipeline is nothing but start a new task before a old task has been completed.

- According to the levels of processing, Handler (1977) has proposed the following classification scheme for pipeline processors as shown in Fig. 6.6.

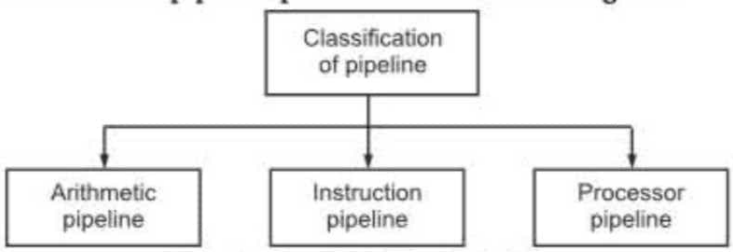


Fig. 6.6: Classification of Pipeline

- The Pentium processor sends two instructions in parallel to the two independent integer pipelines called U and V pipelines. Hence multiple instructions are executed simultaneously.
- The processor capable of parallel instruction execution is known as superscalar machine.
- Pipelining exist within a single processor.

6.6.1 Arithmetic Pipelining [W-17]

- An Arithmetic pipeline divides an arithmetic operation, such as a multiply, into multiple arithmetic steps each of which are executed one-by-one in sequence in different arithmetic sections in the ALU.
- The arithmetic pipeline represents the parts of an arithmetic operation that can be broken down and overlapped as they are performed.
- Some functions of the arithmetic logic unit of a processor can be pipelined to maximize performance.
- An arithmetic pipeline is used to implement complex arithmetic functions like floating point addition, multiplication and division.
- Pipeline arithmetic units are usually found in very high speed computers. Floating point operations are also easily decomposed into sub-operations.

6.6.2 Instruction Pipelining [S-17]

1. Instruction pipelining:

- In instruction cycle the decode and execute cycles for several instructions are performed simultaneously to reduce overall processing time. This process is referred as instruction pipelining.
- To apply the concept of instruction pipelining we must subdivide instruction processing in number of stages as given below.

  S1 : Fetch (F) : Read instruction from memory.

  S2 : Decode (D) : Decode the code and fetch source operand (S) if necessary.

  S3 : Execute (E) : Perform the operation specified by instructions.

  S4 : Store (S) : Store the result in destination.
- Here, instruction processing is divided into four stages hence it is known as four stage pipeline instruction.
- With this subdivision and assuming equal duration for each stage we can reduce the execution time for four instructions from 16 time units to 7 time units. This is illustrated in Fig. 6.7.

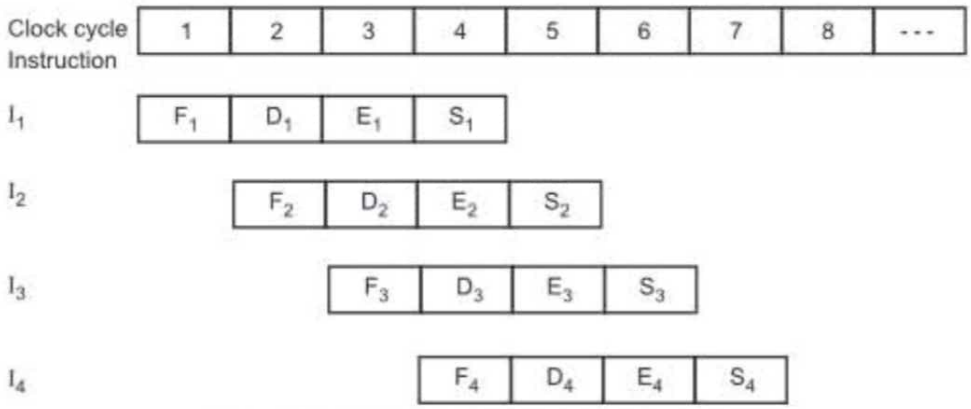


Fig. 6.7: Four stage instruction pipelining

- In this instruction pipelining four instructions are in progress at any given time this means that four distinct hardware are needed as shown in Fig. 6.8.
- These units are implemented such that they are capable of performing their tasks simultaneously and without interfering with one another. Information from the stage is passed to the next stage with the help of buffer.

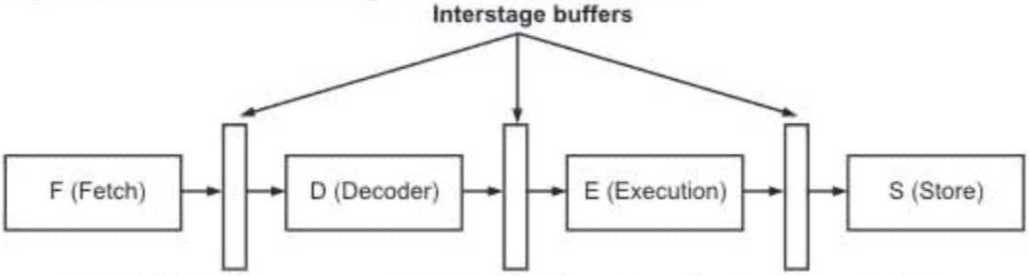


Fig. 6.8: Hardware organization for four stage instruction pipeline

- There are certain difficulties that will prevent the instruction pipeline from executing at it maximum rate.
  1. Different stages of pipeline may take different times to execute.
  2. Some instruction may skip some of the stages. For example, immediate addressing for register addressing does not need effective address calculations.
  3. Two or more stages are requiring the access of memory at the same time. For example, the instruction fetch stage and operand fetch stage will need to access memory at the same time. In this case one of these stages will have to wait while the other stage finishes its tasks.

6.6.3 Processor Pipelining

- This refers to the pipeline processing of the same data stream by a cascade processors, each of which performs a specific task.
- The data stream passes the first processor with results stored in a memory block which is also accessible by the second processor. The second processor then passes the refined results to the third, and so on.

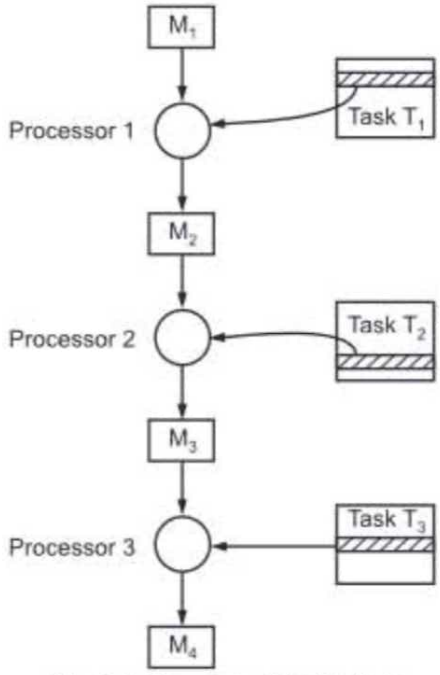


Fig. 6.9: Processor Pipelining

6.6.4 Advantages of Pipelining

- Pipelining does not reduce the time it takes to complete an instruction, instead it increases the number of instructions that can be processed simultaneously and reduces the delay between completed instructions (called as throughput).

6.6.5 Difference between Arithmetic and Instruction Pipeline

Sr. No.	Arithmetic Pipeline	Instruction Pipeline
1.	An arithmetic pipeline divides an arithmetic operation into sub operations for execution in the pipeline segments.	It is a technique for implementing instruction level parallelism within a single processor.
2.	It is non-linear. It uses more than one stage at a time.	It is a linear.
3.	It is synchronous pipelining i.e. no buffering between stages is provided. (Each stage must be ready to accept the data passed from a previous stage when that data is produced).	It is a synchronous.

Difference between "U" and "V" Pipeline

Sr. No.	U Pipeline	V Pipeline
1.	U Pipeline can execute any instruction.	V pipeline can execute simple instruction.
2.	U pipeline has a barrel shifter.	V pipeline do not have barrel shifter.
3.	U pipeline can execute the full range of Pentium instructions.	V pipeline can execute a limited number instructions.

6.7 MICROCONTROLLER

- A Microcontroller is a small and low-cost microcomputer, which is designed to perform the specific tasks of embedded systems like displaying microwave's information, receiving remote signals, etc.
- The general microcontroller consists of the processor, the memory (RAM, ROM, EPROM), Serial ports, peripherals (timers, counters), etc.

Block Diagram of a Microcontroller:

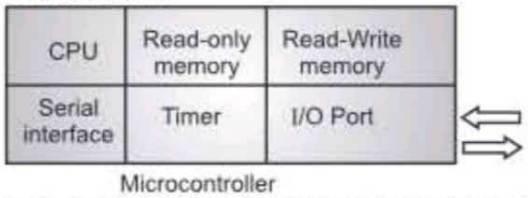


Fig. 6.10: Schematic Internal Architecture of a Microcontroller

How does a Microcontroller Work?

The Central Processing Unit:

- The CPU performs arithmetic operations, manages data flow, and generates control signals in accordance with the sequence of instructions created by the programmer.

Memory:

- The embedded system contains two types of semiconductor memory - Random Access Memory (RAM) and Read Only Memory (ROM). RAM stores user program and data. Embedded software (firmware) is stored in ROM. When power is switched ON, the processor executes the embedded software stored in ROM.

Input/Output Port:

- Normally, microcontroller is used in embedded systems to control the operation of machines in the microcontroller. Therefore, to connect it to other machines, devices or peripherals we require I/O interfacing ports in the microcontroller interface. For this purpose, microcontroller 8051 has 4 parallel and one serial input, output ports to connect it to the other peripherals.

Timers/Counters:

- 8051 microcontroller has two 16-bit timers and counters. These counters are again divided into a 8-bit registers. The timers are used for measurement of time intervals to determine the pulse width of pulses.

Advantages of Microcontroller:

- Low time required for performing operation.
- The processor chips are very small and flexibility occurs.
- Due to their higher integration, cost and size of the system is reduced.
- The microcontroller is easily used to interface additional RAM, ROM and I/O ports.
- At the same time many tasks can be performed so human efforts can be saved.
- Without any additional digital parts it can act as microcomputer.
- It is easy to use, as well as troubleshooting and system maintenance is simple.

Disadvantages of Microcontroller:

- The microcontroller cannot interface high power devices directly.
- It has more complex structure as compared to microprocessor.

- It only performs limited number of executions simultaneously.
- It is generally used in micro-equipments.
- Once microcontrollers are programmed for application specific circuitry then they cannot be reprogrammed.

Difference between Microprocessor and Microcontroller:

Sr. No.	Basis for comparison	Microprocessor	Microcontroller
1.	**Basic**	Made up of a single silicon chip comprising an ALU, CU and registers.	Consist of microprocessor, memory, I/O port, timers, interrupt control unit etc.
2.	**Characteristic**	Dependent unit.	Self-contained unit.
3.	**I/O Ports**	Does not contain built-in I/O port.	Built-in I/O ports are present.
4.	**Type of operation performed**	General purpose in design and operation.	Application oriented or domain specific.
5.	**Targeted for**	High end market.	Embedded market.
6.	**Power consumption**	Provides less power saving options.	Includes more power saving options.
7.	**Instruction set**	Processing intensive and have powerful full addressing model	Control input and output devices.

Types of Microcontrollers:

- Microcontrollers are divided into various categories based on memory, architecture, bits and instruction sets. Following is the list of their types:

  (a) Based on Bits: Based on bit configuration, the microcontroller is further divided into three categories.

  1. **8-bit Microcontroller:** This type of microcontroller is used to execute arithmetic and logical operations like addition, subtraction, multiplication division, etc. For example, Intel 8031 and 8051 are 8 bits microcontroller.
  2. **16-bit Microcontroller:** This type of microcontroller is used to perform arithmetic and logical operations where higher accuracy and performance is required. For example, Intel 8096 is a 16-bit microcontroller.
  3. **32-bit Microcontroller:** This type of microcontroller is generally used in automatically controlled appliances like automatic operational machines, medical appliances, etc.

  (b) Based on Memory: Based on the memory configuration, the microcontroller is further divided into two categories.

  1. **External memory Microcontroller**: This type of microcontroller is designed in such a way that they do not have a program memory on the chip. Hence, it is named as external memory microcontroller. For example: Intel 8031 microcontroller.

2. **Embedded memory Microcontroller:** This type of microcontroller is designed in such a way that the microcontroller has all programs and data memory, counters and timers, interrupts, I/O ports are embedded on the chip. For example: Intel 8051 microcontroller.

(c) **Based on Instruction Set**: Based on the instruction set configuration, the microcontroller is further divided into two categories.

1. **CISC**: CISC stands for Complex Instruction Set Computer. It allows the user to insert a single instruction as an alternative to many simple instructions.
2. **RISC**: RISC stands for Reduced Instruction Set Computers. It reduces the operational time by shortening the clock cycle per instruction.

6.7.1 Introduction to Microcontroller Intel 8051

- A Microcontroller is a VLSI IC that contains a CPU (Processor) along with some other peripherals like Memory (RAM and ROM), I/O Ports, Timers/Counters, Communication Interface, ADC, etc.
- The 8051 Microcontroller is one of the most popular and most commonly used microcontrollers in various fields like embedded systems, consumer electronics, automobiles, etc.
- Technically called as Intel MCS-51 Architecture, the 8051 microcontroller series was developed by Intel in the year 1980.
- Some of the 8051 Microcontrollers produced by different manufacturers are: Atmel (AT89C51, AT89S51), Phillips (S87C654), STC Micro (STC89C52), Infineon (SAB-C515, XC800), Siemens (SAB-C501), Silicon Labs (C8051), NXP (NXP700, NXP900), etc.
- 8051 is an 8 – bit Microcontroller i.e. the data bus of the 8051 Microcontroller (both internal and external) is 8 – bit wide. It is a CISC based Microcontroller with Harvard Architecture (separate program and data memory).

Applications of 8051 Microcontroller:

- Even with the development of many advanced and superior Microcontrollers, 8051 Microcontroller is still being used in many embedded system and applications.
- Some of the applications of 8051 Microcontroller are mentioned below:
  - Consumer Appliances (TV Tuners, Remote controls, Computers, Sewing Machines, etc.)
  - Home Applications (TVs, VCR, Video Games, Camcorder, Music Instruments, Home Security Systems, Garage Door Openers, etc.)
  - Communication Systems (Mobile Phones, Intercoms, Answering Machines, Paging Devices, etc.)
  - Office (Fax Machines, Printers, Copiers, Laser Printers, etc.)
  - Automobiles (Air Bags, ABS, Engine Control, Transmission Control, Temperature Control, Keyless Entry, etc)
  - Aeronautical and Space
  - Medical Equipment
  - Defense Systems
  - Robotics
  - Industrial Process and Flow Control
  - Radio and Networking Equipment
  - Remote Sensing

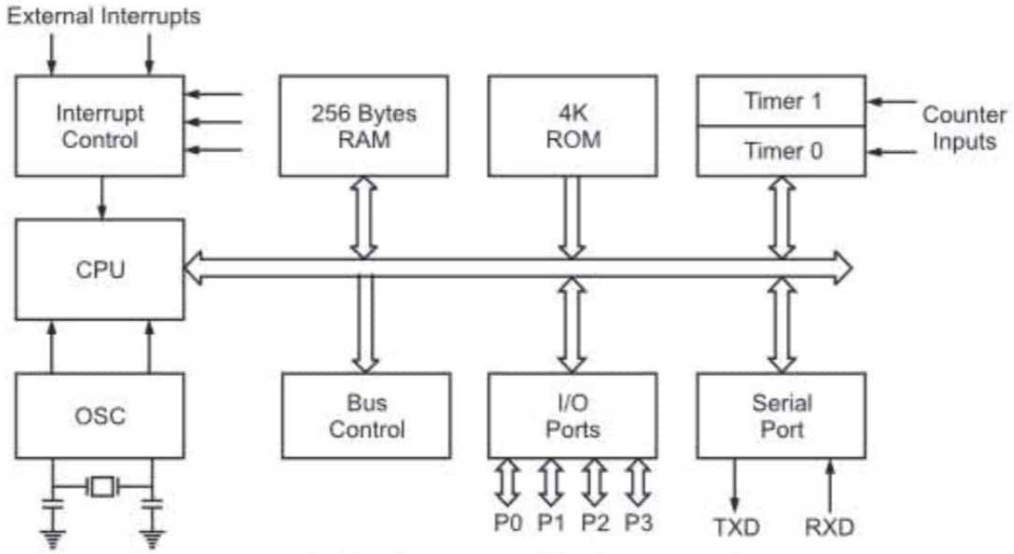


Fig. 6.11: Layout of 8051 Microcontroller

Features of 8051 Microcontroller:

- **8-Bit ALU:** ALU or Arithmetic Logic Unit is the heart of a microcontroller. It performs arithmetic and bitwise operation on binary numbers. The ALU in 8051 is an 8-Bit ALU i.e. it can perform operations on 8 – bit data.
- **8-Bit Accumulator:** The Accumulator is an important register associated with the ALU which holds one of the operated while performing arithmetic and logical instructions and also holds result of operation. The accumulator in 8051 is an 8 – bit register.
- **RAM:** 8051 Microcontroller has 128 Bytes of RAM which includes Special Function Register (SFRs) and Input / Output Port Registers.
- **ROM:** 8051 has 4 KB of on-chip ROM (Program Memory).
- **I/O Ports:** 8051 has four 8 – bit Input / Output Ports which are bit addressable and bidirectional.
- **Timers / Counters:** 8051 has two 16 – bit Timers called T0 and T1.
- **Serial Port:** 8051 supports full duplex UART Communication.
- **External Memory:** 8051Microcontroller can access two 16 – bit address line at once: one each for RAM and ROM. The total external memory that an 8051 Microcontroller can access for RAM and ROM is 64KB (2^{16} for each type).
- **Additional Features:** Interrupts, on-chip oscillator, Boolean Processor, Power Down Mode, etc.

6.8 INTRODUCTION TO CORE PROCESSORS

- The processor is the main component of a computer system. It is a logic circuitry that processes instructions.
- It is also called CPU (Central Processing Unit). It is the brain of the computer system.

- Processor is mainly responsible to do all the computational calculations, logical decision making and to control different activities of the system.
- Central Processing Unit is very complicated chip consisting of billions of electronic components.
- It is fitted on the motherboard with other electronic parts .
- The main work of the processor is to execute low level instructions loaded into the memory .
- The processor can be manufactured using different technologies - Single core processor and multicore processor.
- According to processors can be divided into three types- multiprocessors, multithreaded processors and multicore processors.
- There are new trends in the CPU manufacturing industry which are based on the idea that while clock speeds can only be increased to a limit and there is limit to number of electronic components to be used in a core.
- Many other technologies are there to speed things up and open ways for better and more powerful central processing units .
- When we are unable to increase the performance of CPU furthermore by modifying its running frequency, then new technology called multicore architecture helps.
- In multicore architecture we can put more than one core on a single silicon die.
- This new approach to enhance the speed came with some additional benefits like better performance, better power management and better cooling as the multi core processors run at a lower speed to dissipate less heat.
- It also has some disadvantages like existing programs need to be rewritten as per new architecture.
- If we do not write programs with special focus for running on parallel cores, we will not get advantage of multicores.

6.8.1 Multi-Core Processors

- Multicore processor are the latest processors which became available in the market after 2005.
- These processors use two or more cores to process instructions at the same time by using hyper threading.
- The multiple cores are embedded in the same die.
- The multicore processor may looks like a single processor but actually it contains two (dual-core), three (tricore), four (quad-core), six (hexa-core), eight (octa-core) or ten (deca-core) cores. Some processor even have 22 or 32 cores. Due to power and temperature constraint, the multicore processors are only practical solution for increasing the speed of future computers.

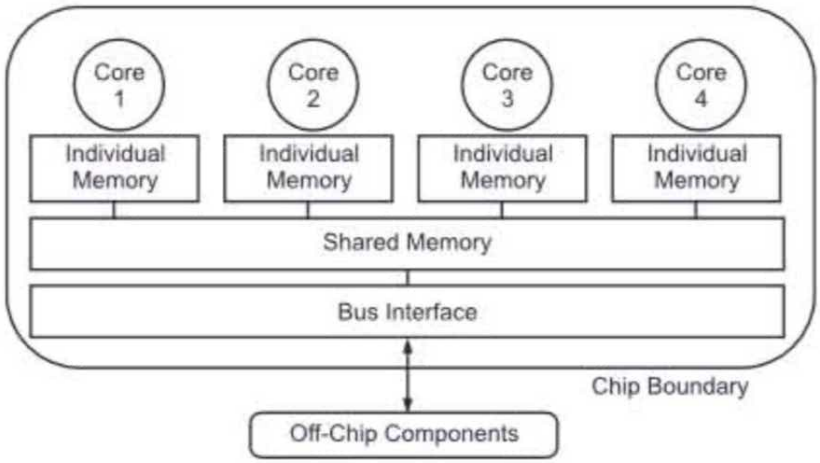


Fig. 6.12: Block diagram for a general Multi-Core processor

6.8.1.1 Problems with Multicore Processors

- According to Amdahl's law, the performance of parallel computing is limited by its serial components. So, increasing the number of cores may not be the best solution.
- There is need to increase the clock speed of individual cores.

Comparison of Single-Core Processor and Multi-Core Processor:

Parameter	Single-Core Processor	Multi-Core Processor
Number of cores on a die	Single	Multiple
Instruction Execution	Can execute Single instruction at a time	Can execute multiple instructions by using multiple cores
Gain	Speed up every program or software being executed	Speed up the programs which are designed for multi-coreprocessors
Performance	Dependent on the clock frequency of the core	Dependent on the frequency, number of cores and program to be executed
Examples	Processor launched before 2005 like 80386, 486, AMD 29000, AMD K6, Pentium I, II, III etc.	Processor launched after 2005 like Core-2- Duo, Athlon 64 X2, I3.I5 and I7 etc.

- One architecture uses single core while the other is using two or more cores on the same die for processing instructions. In todays' time people use multicore processors but single core processors are also very important as far as further speed up is required. It the single-core processors which are put together to make a multi-core processor.

6.8.1.2 Understanding Dual Core, Quad Core, Hexa Core, Octa Core and Deca Core Processors

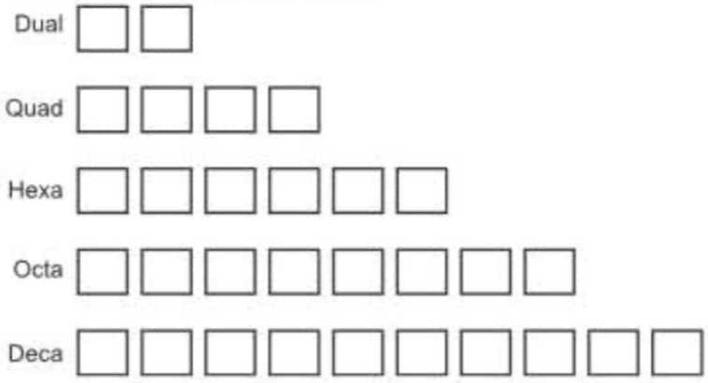


Fig. 6.13: Number of dual core processors

Dual Core Processors:

- Dual-core Processor simply means Central Processing Units which has two complete execution cores per physical processor.
- Dual core processors has two processors combined together and their caches and cache controllers onto a single integrated circuit.

Quad Core Processors:

- Quad-core processor is a chip with four independent units that read and execute information and tasks to be done in the central processing unit (CPU).
- Within the chip, each core operates in conjunction with other circuits such as cache, memory management, and input/output (I/O) ports.
- The individual cores in a quad-core processor can run multiple instructions at the same time, increasing the overall speed for programs compatible with parallel processing.

Hexa Core Processors:

- Hexa Core Processor is another multi-core processor that is built with six cores.
- It can handle tasks faster than the dual core and quad core processors. Hexa Core processors have been introduced since 2010.
- It was launched in Intel Core i7 Hexa-Core processor.

Octa Core Processors:

- The Octa Core processors are made up of eight (8) independent cores to handle and perform task efficiently.
- This implies that Octa Core processors can performs double faster than the quad core processor.
  You might find out that two sets of quad-core processors are present in a gadget, it is still Octa Core as you cannot call it double dual core processor.
- The cores will split various tasks between them according to their types and works brilliantly.

Deca Core Processors:

- As dual core processors are made up of 2 cores, quad core with 4 cores, hexa core with 6 cores and octa core processors with 8 cores.
- Deca Core is also made up of 10 complete independent units to handle and perform task very efficiently than other processors.
- Having Deca Core processor along with an efficient processor type would really make Gadgets to perform efficiently and super faster than others.
- Deca Core processors is the latest and new processor.
- We have only heard about few smartphones having the Deca Core processors.

Table: Present number of cores on a processor

Model	No. of Core	Example
Single Core	1	AMD Ordinary
Dual Core	2	Intel Duo
Quad Core	4	Intel 13.15
Hexa Core	6	AMD Phenom X6
Octa Core	8	INTEL, XENON

6.8.1.3 Advantages and Disadvantages

Advantages of Multicore Processors:

- Having a multi-core processor in a computer means that it will work faster for certain programs.
- The computer may not get as hot when it is turned on.
- The computer needs less power because it can turn off some sections if they aren´t needed.
- More features can be added to the computer.
- The signals between different CPUs travel shorter distances, therefore they degrade less.

Disadvantages of Multicore Processors:

- They do not work at twice the speed as a normal processor. They get only 60-80% more speed.
- The speed that the computer works at depends on what the user is doing with it.
- They cost more than single core processors.
- They are more difficult to manage thermally than lower-density single-core processors.
- Not all operating systems support more than one core.
- Operating systems compiled for a multi-core processor will run slightly slower on a single-core processor.

Operating System Support:

- The following operating systems support multi-core processors
  - Microsoft Windows (Windows XP or newer)
  - Linux
  - Mac OS X
  - Most BSD-based systems
  - Solaris

Summary

- The microprocessor has a digital circuit for data handling and computation under program control. The microprocessor is a data processing unit. Data processing includes both computation and data handling.
- Pentium, Family of microprocessors developed by Intel Corp. Introduced in 1993 as the successor to Intel's 80486 microprocessor, the Pentium contained two processors on a single chip and about 3.3 million transistors.
- A microcontroller is a small and low-cost microcomputer, which is designed to perform the specific tasks of embedded systems like displaying microwave's information, receiving remote signals, etc.
- 8051 microcontroller is designed by Intel in 1981. It is an 8-bit microcontroller. It is built with 40 pins DIP (dual inline package), 4KB of ROM storage and 128 bytes of RAM storage, 16-bit timers. It consists of four parallel 8-bit ports, which are programmable as well as addressable as per the requirement.
- Pipelining is a technique of decomposing a sequential process into sub-operations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments.
- Parallelism is the process of processing several set of instructions simultaneously. It reduces the total computational time.
- A Multi Core processor is an Integrated Circuit (IC) to which two or more processors have been attached for enhanced performance, reduced power consumption, and more efficient simultaneous processing of multiple tasks .

Check Your Understanding

1. Because of Pentium's superscalar architecture, the number of instructions that are executed per clock cycle is _____
   (a) 1 (b) 2
   (c) 3 (d) 4
2. The register which holds the information about the nature of results of arithmetic and logic operations is called as _____
   (a) Accumulator (b) Condition code register
   (c) Flag register (d) Process status register
3. The 8051 Microcontroller has _____ parallel I/O ports.
   (a) 3 (b) 2
   (c) 5 (d) 4
4. When multiple-instructions are overlapped during execution of program, then function performed is called
   (a) Multitasking (b) Multiprogramming
   (c) Hardwired control (d) Pipelining
5. The fetch and execution cycles are interleaved with the help of _____
   (a) modification in processor architecture
   (b) clock
   (c) special unit
   (d) control unit

Answers

1. (b)	2. (c)	3. (d)	4. (d)	5. (b)

Practice Questions

Q.1 Answer the following Question in brief:

1. Give the block diagram of 8051.
2. Write short note on Applications of Microprocessor.
3. Describe concept of parallelism in detail.
4. Explain instruction pipeline in detail.
5. Explain block diagram of Pentium.
6. Write short note on U & V pipeline.

Q.2 Answer the following Question:

1. Explain Arithmetic and Instruction Pipeline.
2. What is Microcontroller? Which are features of 8051 microcontroller?
3. Write difference between microcontroller and microprocessor.
4. What is multicore processor? Explain with its advantages.
5. Explain the programming model of microprocessor.

Q.3 Define Terms:

1. Define the following terms:
   Parallel processing, Parallelism, Pipeline, Arithmetic pipeline, Microprocessor, Microcontroller, Program counter, PLP, CISC, RISC.

Previous Exams Questions

Winter 2017

1. Write a short note on parallelism. **(4 M)**

Ans. Refer to Section 6.5.

2. Explain the working of arithmetic pipeline in brief. **(4 M)**

Ans. Refer to Section 6.6.1.

Summer 2017

1. 8086 is a _____ bit microprocessor. **(1 M)**

(a) 32 (b) 16
(c) 24 (d) 48

Ans. Refer to Section 6.1.

2. Write a short note on instruction pipeline. **(4 M)**

Ans. Refer to Section 6.6.2.

Summer 2018

1. Define term: Pipeline. **(1 M)**

Ans. Refer to Section 6.6.

■■■