Organization Lab

MCA- 108

SELF LEARNING MATERIAL

DIRECTORATE

OF DISTANCE EDUCATION

SWAMI VIVEKANAND SUBHARTI UNIVERSITY

MEERUT – 250 005,

UTTAR PRADESH (INDIA)

SLM Module Developed By :

Author:

Reviewed by :

Assessed by:

Study Material Assessment Committee, as per the SVSU ordinance No. VI (2)

DISCLAIMER

No part of this publication which is material protected by this copyright notice may be reproduced or transmitted or utilized or stored in any form or by any means now known or hereinafter invented, electronic, digital or mechanical, including photocopying, scanning, recording or by any information storage or retrieval system, without prior permission from the publisher.

Information contained in this book has been published by Directorate of Distance Education and has been obtained by its authors from sources be lived to be reliable and are correct to the best of their knowledge. However, the publisher and its author shall in no event be liable for any errors, omissions or damages arising out of use of this information and specially disclaim and implied warranties or merchantability or fitness for any particular use.

Published by: Gayatri Sales

Typeset at: Micron Computers

Printed at: Gayatri Sales, Meerut.

ORGANIZATION LAB

- Study and Bread Board Realization of Logic Gates. K-Map, Flip-Flop equation, realization of characteristic and excitation table of various Flip Flops.
- Implementation of Half Adder, Full Adder and Subtractor.
- Implementation of Ripple Counters and Registers.
- Implementation of Decoder and Encoder circuits.
- Implementation of Multiplexer and D-Multiplexer circuits.

Unit-I

(Representation of Information and Basic Building Blocks) :-

Introduction to Computer

The Bachelor of Computer Applications (BCA) and the Master of Computer Application (MCA) are undergraduate and postgraduate programs or courses which have guidelines and eligibility set out by the All India Council of Technical Education (AICTE). They are typically three-year programs that span six semesters. They are designed to bridge the gap between the studies of computers and its applications. The master's program aims to shape computer professionals with the right moral and ethical values and can prepare students to face the challenges and opportunities in the IT industry of India by building strong foundations.

The syllabus focuses on the core fundamentals of computer science, but generally undergoes revision according to the industry requirement with the aim of increasing employment opportunities for students. Admission to both the BCA and MCA can be obtained by clearing the appropriate entrance exams. Having a background in science can help in grasping concepts, and mathematics is a compulsory subject. BCA and MCA graduates can seek job opportunities in fields like software development, web design, systems management, quality assurance and software testing. A MCA or BCA graduate can work in IT companies big and small in various roles.

Various topics covered under BCA and MCA

Bachelor of Computer Applications (BCA)

The Bachelor of Computer Applications course usually consists of core courses in programming with C, algorithm and data structures, advanced programming with C, computer organization and network fundamentals, relational database management systems, Web programming, software engineering and visual programming. Some of the main course highlights are meant to heighten technological know-how, to train students to become industry specialists, to provide research-based training and to encourage software development. The syllabus is intended to not only teach students theory and applications, but can also help equip them with professional and communication skills.

A few of the topics covered under BCA courses are listed below. Each semester has 6 topics to cover so a total of 36 topics are covered in the entire duration of the course. Each semester has Computer Laboratory and Practical work based on the syllabus of that particular semester.

- Business Communication
- Principles of Management
- Programming Principles and Algorithms
- Computer Fundamental and Office Automation
- Business Accounting
- Organizational Behaviour
- Elements of Statistics
- C Programming
- Cost Accounting
- Software Engineering
- C++, Core Java, Advanced Java

Master of Computer Applications (MCA)

In Master of Computer Applications course, higher level subjects like computer organization, data and file structuring using C, operating system, computer networks, modelling and simulation, visual basic, combinatory and graph theory, computer graphics, system programming and computer based numerical and statistical techniques are generally taught. In the final semester, you will most likely need to specialize in a topic in the form of project work. The course stresses the application of theory and computing principles through project work, case studies, presentations and practical assignments. Some of the popular fields which are generally chosen by students for MCA specialization are application software, hardware technology, systems management, systems development, and management information systems.

A few of the topics covered under MCA courses are listed below. Each semester has 6 theoretical topics to cover so a total of 30 topics are covered in the first five semesters of the MCA course. In the sixth and final semester a student has to submit an industrial project. Each semester has a lot of practical work involved based on the syllabus of MCA.

- Accounting and Financial Management
- UNIX and Shell Programming
- Object Oriented Systems in C++

- Design and Analysis of Algorithm
- Modelling and Simulation
- Software Engineering
- Net Framework and C Lab
- WEB Technology

Computer hardware generation

Introduction:
A computer is an electronic device that manipulates information or data. It has the ability to store, retrieve, and process data.
Nowadays, a computer can be used to type documents, send email, play games, and browse the Web. It can also be used to edit or create spreadsheets, presentations, and even videos. But the evolution of this complex system started around 1940 with the first Generation of Computer and evolving ever since.
There are five generations of computers.

1. **FIRST GENERATION**

- **Introduction:**
  1. 1946-1959 is the period of first generation computer.
  2. J.P.Eckert and J.W.Mauchy invented the first successful electronic computer called ENIAC, ENIAC stands for “Electronic Numeric Integrated And Calculator”.
- **Few Examples are:**
  1. ENIAC
  2. EDVAC
  3. UNIVAC
  4. IBM-701
  5. IBM-650

  …
- **Advantages:**

1. It made use of vacuum tubes which are the only electronic component available during those days.

2. These computers could calculate in milliseconds.

- **Disadvantages:**

1. These were very big in size, weight was about 30 tones.

2. These computers were based on vacuum tubes.

3. These computers were very costly.

4. It could store only a small amount of information due to the presence of magnetic drums.

5. As the invention of first generation computers involves vacuum tubes, so another disadvantage of these computers was, vacuum tubes require a large cooling system.

6. Very less work efficiency.

7. Limited programming capabilities and punch cards were used to take inputs.

8. Large amount of energy consumption.

9. Not reliable and constant maintenance is required.

2. **SECOND GENERATION**

- **Introduction:**

1. 1959-1965 is the period of second-generation computer.

2. 3.Second generation computers were based on Transistor instead of vacuum tubes.

- **Few Examples are:**

1. Honeywell 400

2. IBM 7094

3. CDC 1604

4. CDC 3600
5. UNIVAC 1108

… many more

- **Advantages:**
    1. Due to the presence of transistors instead of vacuum tubes, the size of electron component decreased. This resulted in reducing the size of a computer as compared to first generation computers.
    2. Less energy and not produce as much heat as the first genration.
    3. Assembly language and punch cards were used for input.
    4. Low cost than first generation computers.
    5. Better speed, calculate data in microseconds.
    6. Better portability as compared to first generation
- **Disadvantages:**
    1. A cooling system was required.
    2. Constant maintenance was required.
    3. Only used for specific purposes.

3. THIRD GENERATION

- **Introduction:**
    1. 1965-1971 is the period of third generation computer.
    2. These computers were based on Integrated circuits.
    3. IC was invented by Robert Noyce and Jack Kilby In 1958-1959.
    4. IC was a single component containing number of transistors.
- **Few Examples are:**
    1. PDP-8

2. PDP-11

3. ICL 2900

4. IBM 360

5. IBM 370

… and many more

- **Advantages:**

1. These computers were cheaper as compared to second-generation computers.

2. They were fast and reliable.

3. Use of IC in the computer provides the small size of the computer.

4. IC not only reduce the size of the computer but it also improves the performance of the computer as compared to previous computers.

5. This generation of computers has big storage capacity.

6. Instead of punch cards, mouse and keyboard are used for input.

7. They used an operating system for better resource management and used the concept of time-sharing and multiple programming.

8. These computers reduce the computational time from microseconds to nanoseconds.

- **Disadvantages:**

1. IC chips are difficult to maintain.

2. The highly sophisticated technology required for the manufacturing of IC chips.

3. Air conditioning is required.

4. FOURTH GENERATION

- **Introduction:**

1. 1971-1980 is the period of fourth generation computer.
2. This technology is based on Microprocessor.
3. A microprocessor is used in a computer for any logical and arithmetic function to be performed in any program.
4. Graphics User Interface (GUI) technology was exploited to offer more comfort to users.

- **Few Examples are:**

  1. IBM 4341
  2. DEC 10
  3. STAR 1000
  4. PUP 11

  … and many more

- **Advantages:**

  1. Fastest in computation and size get reduced as compared to the previous generation of computer.
  2. Heat generated is negligible.
  3. Small in size as compared to previous generation computers.
  4. Less maintenance is required.
  5. All types of high-level language can be used in this type of computers.

- **Disadvantages:**

  1. The Microprocessor design and fabrication are very complex.
  2. Air conditioning is required in many cases due to the presence of ICs.
  3. Advance technology is required to make the ICs.

5. FIFTH GENERATION

- **Introduction:**

    1. The period of the fifth generation in 1980-onwards.
    2. This generation is based on artificial intelligence.
    3. The aim of the fifth generation is to make a device which could respond to natural language input and are capable of learning and self-organization.
    4. This generation is based on ULSI(Ultra Large Scale Integration) technology resulting in the production of microprocessor chips having ten million electronic component.

- **Few Examples are:**

    1. Desktop
    2. Laptop
    3. NoteBook
    4. UltraBook
    5. Chromebook

    … and many more

- **Advantages:**

    1. It is more reliable and works faster.
    2. It is available in different sizes and unique features.
    3. It provides computers with more user-friendly interfaces with multimedia features.

- **Disadvantages:**

    1. They need very low-level languages.
    2. They may make the human brains dull and doomed.

Number System:

Binary

A binary number system is one of the four types of number system. In computer applications, where binary numbers are represented by only two symbols or digits, i.e. 0 (zero) and 1(one). The binary numbers here are expressed in the base-2 numeral system. For example, $(101)_2$ is a binary number. Each digit in this system is said to be a bit. Learn about the number system here.

able of Contents:

Number System is a way to represent the numbers in the computer architecture. There are four different types of the number system, such as:

1. Binary number system (base 2)

2. Octal number system (base 8)

3. Decimal number system(base 10)

4. Hexadecimal number system (base 16).

In this article, let us discuss what is a binary number system, conversion from one system to other systems, table, positions, binary operations such as addition, subtraction, multiplication, and division, uses and solved examples in detail.

What is a Binary Number System?

Binary Number System**:** According to digital electronics and mathematics, a binary number is defined as a number that is expressed in the binary system or base 2 numeral system. It describes numeric values by two separate symbols; 1 (one) and 0 (zero). The base-2 system is the positional notation with 2 as a radix.

The binary system is applied internally by almost all latest computers and computer-based devices because of its direct implementation in electronic circuits using logic gates. Every digit is referred to as a bit.

What is Bit in Binary Number?

A single binary digit is called a "Bit**".** A binary number consists of several bits. Examples are:

- 10101 is a five-bit binary number
- 101 is a three-bit binary number
- 100001 is a six-bit binary number

Facts to Remember:

- Binary numbers are made up of only 0's and 1's.
- A binary number is represented with a base-2
- A bit is a single binary digit.

Binary Numbers Table

Some of the binary notations of lists of decimal numbers from 1 to 30, are mentioned in the below list.

Number	Binary Number	Number	Binary Number	Number	Binary Number
1	1	11	1011	21	10101
2	10	12	1100	22	10110
3	11	13	1101	23	10111
4	100	14	1110	24	11000
5	101	15	1111	25	11001
6	110	16	10000	26	11010

7	111	17	10001	27	11011
8	1000	18	10010	28	11100
9	1001	19	10011	29	11101
10	1010	20	10100	30	11110

How to Calculate Binary Numbers

For example, the number to be operated is 1235.

Thousands	**Hundreds**	**Tens**	**Ones**
1	2	3	5

This indicates,

1235 = 1 × 1000 + 2 × 100 + 3 × 10 + 5 × 1

Given,

1000	$= 10^3 = 10 \times 10 \times 10$
100	$= 10^2 = 10 \times 10$
10	$= 10^1 = 10$
1	$= 10^0$ (any value to the exponent zero is one)

The above table can be described as,

Thousands	**Hundreds**	**Tens**	**Ones**
10^3	10^2	10^1	10^0
1	2	3	5

Hence,

1235 = 1 × 1000 + 2 × 100 + 3 × 10 + 5 × 1

$= 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 5 \times 10^0$

The decimal number system operates in base 10, wherein the digits 0-9 represent numbers. In binary system operates in base 2 and the digits 0-1 represent numbers, and the base is known as **radix**. Put differently, and the above table can also be shown in the following manner.

Thousands	Hundreds	Tens	Ones	
Decimal	10^3	10^2	10^1	10^0
Binary	2^3	2^2	2^1	2^0

We place the digits in columns 10^0, 10^1 and so on in base 10. When there is a need to put a value higher than 9 in the form of $10^{(n+1)}$ for instance, to add 10 to column 10^0, you need to add 1 to the column 10^1.

We place the digits in columns 2^0, 2^1 and so on in base 2. To place a value that is higher than 1 in 2^n, you need to add $2^{(n+1)}$. For instance, to add 3 to column 2^0, you need to add 1 to column 2^1.

Position in Binary Number System

In the Binary system, we have ones, twos, fours etc…

For example 1011.110

It is shown like this:

1 × 8 + 0 × 4 + 1 × 2 + 1 + 1 × ½ + 1 × ¼ + 0 × 1⁄8

= 11.75 in Decimal

To show the values greater than or less than one, the numbers can be placed to the left or right of the point.

For 10.1, 10 is a whole number on the left side of the decimal, and as we move more left, the number place gets bigger (Twice).

The first digit on the right is always Halves ½ and as we move more right, the number gets smaller (half as big).

In the example given above:

- “10” shows ‘2’ in decimal.
- “.1” shows ‘half’.
- So, “10.1” in binary is 2.5 in decimal.

Binary Arithmetic Operations

Like we perform the arithmetic operations in numerals, in the same way, we can perform addition, subtraction, multiplication and division operations on Binary numbers. Let us learn them one by one.

Binary Addition

Adding two binary numbers will give us a binary number itself. It is the simplest method. Addition of two single-digit binary number is given in the table below.

Binary Numbers		**Addition**
0	0	0
0	1	1
1	0	1
1	1	0; Carry →1

Let us take an example of two binary numbers and add them.

For example: Add 1101_2 and 1001_2.

Solution:

Binary Subtraction

Subtracting two binary numbers will give us a binary number itself. It is also a straightforward method. Subtraction of two single-digit binary number is given in the table below.

Binary Numbers		**Subtraction**
0	0	0
0	1	1; Borrow 1
1	0	1
1	1	0

Let us take an example of two binary numbers and subtract them.

Example: Subtract $1101_{2,}$ and 1010_2.

Solution:

Binary Multiplication

The multiplication process is the same for the binary numbers as it is for numerals. Let us understand it with example.

Example: Multiply 1101_2 and 1010_2.

Solution:

Binary Division

The binary division is similar to the decimal number division method. We will learn with an example here.

Example: Divide 1010_2 by 10_2

Solution:

Uses of Binary Number System

Binary numbers are commonly used in computer applications. All the coding and languages in computers such as C, C++, Java, etc. use binary digits 0 and 1 to write a program or encode any digital data. The computer understands only the coded language. Therefore these 2-digit number system is used to represent a set of data or information in discrete bits of information.

Problems and Solutions

Let us practice some of the problems for better understanding:

Question 1: **What is binary number 1.1 in decimal?**

Solution:

Step 1: 1 on the left-hand side is on the one's position, so it's 1.

Step 2: The one on the right-hand side is in halves, so it's

$1 \times ½$

Step 3: so, 1.1 = 1.5 in decimal.

Question 2: Write 10.11_2 in Decimal?

Solution:

$10.11 = 1 \times (2)^1 + 0\ (2)^0 + 1\ (½)^1 + 1(½)^2$

$= 2 + 0 + ½ + ½$

$= 2.75$

So, 10.11 is 2.75 in Decimal.

Keep visiting BYJU'S to explore and learn more such Math-related topics in a fun and engaging way.

Frequently Asked Questions – FAQs

What is a binary number system?

A number system where a number is represented by using only two digits (0 and 1) with a base 2 is called a binary number system. For example, 10012 is a binary number.

What is a bit?

A bit is a single digit in the binary number. For example, 101 is three-bit binary numbers, where 1, 0 and 1 are the bits.

How to convert a decimal number into a binary number? Give an example.

To convert a decimal number into its equivalent binary number, we divide the decimal number by 2 each time, till we get 0 as a dividend. Let us take an example to convert 1310 into a binary number.
13 ÷ 2: 6 and remainder 1
6 ÷ 2: 3 and remainder 0
3 ÷ 2: 1 and remainder 1
1 ÷ 2: 0 and remainder 1

Now we take the bits from the last remainder to first remainder, i.e.(MSB to LSB).
Hence,
13_{10} = 1101^2

What is the use of binary numbers?

Binary numbers are commonly used in computer architecture. Since the computer understands only the language of two digits 0's and 1's, therefore the programming is done using a binary number system.

What is the value of 163 in binary?

The value of 163 in binary is 10100011.

How is 200 represented in binary?

200 is the decimal number. The binary form of 200 is 110010002.

Octal

Octal Number System has a base of eight and uses the number from 0 to 7. The octal numbers, in the number system, are usually represented by binary numbers when they are grouped in pair of three. For example, 12_8 is expressed as 001010_2, where 1 is equivalent to 001 and 2 is equivalent to 010.

Octal Number System

Base – 8

Octal Symbol – 0, 1, 2, 3, 4, 5, 6 and 7

Table of Contents:

Apart from octal number system, there are other number systems in Maths, such as:

- Binary Number System
- Hexadecimal Number System
- Decimal Number System

Definition

A number system which has its base as ‘eight’ is called an Octal number system. It uses numbers from 0 to 7. Let us take an example, to understand the concept. As we said, any number with base 8 is an octal number like 24_8, 109_8, 55_8, etc.

Like Octal number is represented with base 8, in the same way, a binary number is represented with base 2, decimal number with base 10 and the hexadecimal number is represented with base 16. Examples for these number systems are:

- 22_2 is a binary number
- 100_{10} is a decimal number
- 40_{16} is a hexadecimal number

If we solve an octal number, each place is a power of eight.

- $124_8 = 1 \times 8^2 + 2 \times 8^1 + 4 \times 8^0$

Octal Numbers Chart

We use only 3 bits to represent Octal Numbers. Each group will have a distinct value between 000 and 111.

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

Decimal to Octal Number

To convert decimal to octal number, octal dabble method is used. In this method, the decimal number is divided by 8 each time, it yields or gives a remainder. The first remainder we get is the least significant digit(LSD) and the last remainder is the most significant digit(MSD). Let us understand the conversion with the help example.

Solved Example

Problem: Suppose 560 is a decimal number. Convert it into an octal number.

Solution: If 560 is a decimal number, then,

560/8=70 and remainder is 0

70/8=8 and remainder is 6

8/8=1 and remainder is 0

And 1/8=0 and remainder is 1

So the octal number starts from MSD to LSD, i.e. 1060

Therefore, $560_{10} = 1060_8$

Problem: Convert 0.52 into an octal number.

Solution: The fraction part of the decimal number has to be multiplied by 8.

0.52 × 8=0.16 with carry 4

0.16 × 8=0.28 with carry 1

0.28 × 8=0.24 with carry 2

0.24 × 8=0.92 with carry 1

So, for the fractional octal number, we read the generated carry from up to down.

Therefore, 4121 is the octal number.

Octal to Decimal Number

Let us learn here, conversion of Octal number to Decimal Number or base 8 to base 10.

Solved Example

Example: Suppose 215_8 is an octal number, then it's decimal form will be,

$215_8 = 2 \times 8^2 + 1 \times 8^1 + 5 \times 8^0$

$= 2 \times 64+ 1 \times 8 + 5 \times 1 = 128 + 8 + 5$

$= 141_{10}$

Example: Let 125 is an octal number denoted by 125_8. Find the decimal number.

$125_8 = 1\times 8^2 + 2 \times 8^1 + 5 \times 8^0$

$= 1 \times 64 + 2 \times 8 + 5 \times 1 = 64+16+5$

$=85_{10}$

Binary To Octal Number

A binary number can be converted into an octal number, with the help of the below-given table.

Octal Number	**Equivalent Binary Number**
0	0
1	1
2	10
3	11

4	100
5	101
6	110
7	111

Solved Example

Example: Convert $(100010)_2$ to octal number.

Solution: With the help of the table we can write,

100→4

and 010→2

Therefore,$(100010)_2$ = 42

Similarly, we can convert an octal number to binary number with the help of the table.

Octal to Hexadecimal Number

Hexadecimal number consist of numbers and alphabets. It is represented with base 16. The numbers from 0-9 are represented in the usual form, but from 10 to 15, it is denoted as A, B, C, D, E, F. Conversion of the octal number to hexadecimal requires two steps.

- First, convert octal number to decimal number.
- Then, convert decimal number to hexadecimal number.

Let us understood with the help of an example. We will take the same example, where we have converted octal number to decimal, such as;

$(215)_8 = (45)_{10}$

Now, convert $(45)_{10}$ into a hexadecimal number by dividing 45 by 16 until you get remainder less than 16.

Therefore, we can write, $(45)_{10} = (2D)_{16}$

Or $(215)_8 = (2D)_{16}$

Octal Multiplication Table

*	0	1	2	3	4	5	6	7

0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

Applications

The octal Number system is widely used in computer application sectors and also in the aviation sector to use the number in the form of code.

Based on octal number system applications, several computing systems are developed. All the modern generation computing system uses 16-bit, 32-bit or 64-bit word which is further divided into 8-bit words. Similarly, for various programming languages, octal numbers are used to do coding or to write the encrypted language, which is only understood by the computing machine.

Also in the aviation sector or field or say aviation industry, Transponders used in the aircraft transmits a code which is expressed as four octal digit number. These codes are interrogated by ground radar.

Also, study-related topics on number systems by downloading BYJU’S -The Learning App.

Frequently Asked Questions – FAQs

What is an octal number system?

A number system expressed with base-8 and whose range is from 0 to 8 only, it is called octal number system. It is represented as N8.

What is the use of octal numbers?

The octal Number system is widely used in computer application sectors. All the modern generation computing system uses 16-bit, 32-bit or 64-bit word which is further divided into 8-bit words. Also in the aviation sector the octal numbers are used in the form of code.

What is the importance of octal number system?

Since, the octal numbers uses less number of digits as compared to decimal numbers and hexadecimal numbers, therefore it is easy to do computations in fewer steps and also less chances of occurrence of error.

What is the octal form of decimal number 19?

To convert a decimal number into octal number, we need to divide the given decimal number by 8 and until output is 0. At last we need to write the remainder from LSD to MSD in reverse order.
19/8 = 2, Remainder = 3
2/8 = 0, Remainder = 2
Therefore, $19_{10} = 23_8$

What are 4 types of number system?

Binary number system
Octal Number system
Decimal number system
Hexadecimal number system

What is 13_8 in binary?

For octal number 13,
1 → 001
3 → 011
Therefore, clubbing both the numbers we get:
$13_8 = 001011_2$
Or $13_8 = 1011_2$

What is the binary number 1111 equivalent to in octal number system?

1111 can be written in group of three digits by adding 0’s, such as;
001111 → 001 111 → 17
$(1111)_2 \rightarrow (17)_2$

Hexadecimal

Hexadecimal Number System is one the type of Number Representation techniques, in which there value of base is 16. That means there are only 16 symbols or possible digit values, there are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Where A, B, C, D, E and F are single bit representations of decimal value 10, 11, 12, 13, 14 and 15 respectively. It requires only 4 bits to represent value of any digit. Hexadecimal numbers are indicated by the addition of either an 0x prefix or an h suffix.

Position of every digit has a weight which is a power of 16. Each position in the Hexadecimal system is 16 times more significant than the previous position, that means numeric value of an hexadecimal number is determined by multiplying each digit of the number by the value of the position in which the digit appears and then adding the products. So, it is also a positional (or weighted) number system.

Representation of Hexadecimal Number

Each Hexadecimal number can be represented using only 4 bits, with each group of bits having a distich values between 0000 (for 0) and 1111 (for F = 15 = 8+4+2+1). The equivalent binary number of Hexadecimal number are as given below.

Hex digit	1	0	2	3	4	5	6	7
Binary	0000	0001	0010	0011	0100	0101	0110	0111

Hex digit	8	9	A = 10	B = 11	C = 12	D = 13	E = 14	F = 15
Binary	1000	1001	1010	1011	1100	1101	1110	1111

Hexadecimal number system is similar to Octal number system. Hexadecimal number system provides convenient way of converting large binary numbers into more compact and smaller groups.

Most Significant Bit (MSB)	**Hex Point**		**Least Significant Bit (LSB)**		
16^2	16^1	16^0	16^{-1}	16^{-2}	16^{-3}
256	16	1	1/16	1/256	1/4096

Since base value of Hexadecimal number system is 16, so there maximum value of digit is 15 and it can not be more than 15. In this number system, the successive positions to the left of the hexadecimal point having weights of 16^0, 16^1, 16^2, $16^3$and so on. Similarly, the successive positions to the right of the hexadecimal point having weights of 16^{-1}, 16^{-2}, $16^{-3}$and so on. This is called base power of 16. The decimal value of any hexadecimal number can be determined using sum of product of each digit with its positional value.

Example-1 − The number 512 is interpreted as

512=2x16^{2}+0x16^{1}+0x16^{0}=200

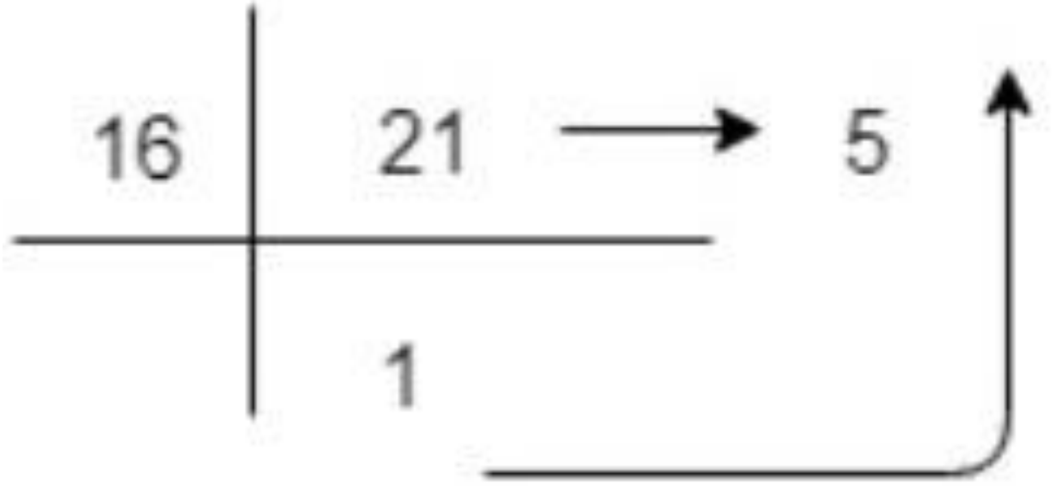
Here, right most bit 0 is the least significant bit (LSB) and left most bit 2 is the most significant bit (MSB).

Example-2 − The number 2015.0625 is interpreted as

2015.0625=7x16^{2}+13x16^{1}+15x16^{0}+1x16^{-1}=7DF.10

Here, right most bit 0 is the least significant bit (LSB) and left most bit 7 is the most significant bit (MSB).

Example-3 A a decimal number 21 to represent in Hexadecimal representation



(21)$_{10}$=1x16^{1}+5x16^{0}=(15)$_{16}$
So, decimal value 21 is equivalent to 15 in Hexadecimal Number System.

Applications of Hexadecimal Number System

Hexadecimal Number System is commonly used in Computer programming and Microprocessors. It is also helpful to describe colors on web pages. Each of the three primary colors (i.e., red, green and blue) is represented by two hexadecimal digits to create 255 possible values, thus resulting in more than 16 million possible colors. Hexadecimal number system is used to describe locations in memory for every byte. These hexadecimal numbers are also easier to read and write than binary or decimal numbers for Computer Professionals.

Advantages and Disadvantages

The main advantage of using Hexadecimal numbers is that it uses less memory to store more numbers, for example it store 256 numbers in two digits whereas decimal number stores 100 numbers in two digits. This number system is also used to represent Computer memory addresses. It uses only 4 bits to represent any digit in binary and easy to convert from hexadecimal to binary and vice-versa. It is easier to handle input and output in the hexadecimal form. There is wide number of advantages in data science field, artificial intelligence and machine learning.

The major disadvantage of Hexadecimal number system is that it may not an easy to read and write for people, and also difficult to perform operations like multiplications, divisions using hexadecimal number system. Hexadecimal numbers is most difficult number system for dealing with Computer's data.

15's and 16's Complement of Hexadecimal (Base-16) Number

Simply, 15's complement of a hexadecimal number is the subtraction of it's each digits from F(=15). For example, 15's complement of hexadecimal number 2030 is FFFF - 2030 = DFCF.

16's complement of hexadecimal number is 15's complement of given number plus 1 to the least significant bit (LSB). For example 8's complement of hexadecimal number 2020 is (FFFF - 2030) + 1 = DFDF + 1 = DFE0. Please note that maximum digit of hexadecimal number system is F(=15), so addition of F+1 will be 0 with carry 1.

Character Codes (BCD, ASCII, EBCDIC)

Computers and digital circuits processes information in the binary format. Each character is assigned 7 or 8 bit binary code to indicate its character which may be numeric, alphabet or special symbol. Example - Binary number 1000001 represents 65(decimal) in straight binary code, alphabet A in ASCII code and 41(decimal) in BCD code.

Types of codes

BCD (Binary-Coded Decimal) code :

Four-bit code that represents one of the ten decimal digits from 0 to 9.

Example - (37)10 is represented as 0011 0111 using BCD code, rather than (100101)2 in straight binary code.

Thus BCD code requires more bits than straight binary code.

Still it is suitable for input and output operations in digital systems.

Note: 1010, 1011, 1100, 1101, 1110, and 1111 are INVALID CODE in BCD code.

ASCII (American Standard Code Information Interchange) code :

It is 7-bit or 8-bit alphanumeric code.

7-bit code is standard ASCII supports 127 characters.

Standard ASCII series starts from 00h to 7Fh, where 00h-1Fh are used as control characters and 20h-7Fh as graphics symbols.

8-bit code is extended ASCII supports 256 symbols where special graphics and math's symbols are added.

Extended ASCII series starts from 80h to FFh.

EBCDIC (Extended Binary Coded Decimal Interchange Code) code

8-bit alphanumeric code developed by IBM, supports 256 symbols.

It was mainly used in IBM mainframe computers.

Gray code

Differs from leading and following number by a single bit.

Gray code for 2 is 0011 and for 3 is 0010.

No weights are assigned to the bit positions.

Extensively used in shaft encoders.

Excess-3 code

4-bit code is obtained by adding binary 0011 to the natural BCD code of the digit.

Example - decimal 2 is coded as 0010 + 0011 = 0101 as Excess-3 code.

It not weighted code.

Its self-complimenting code, means 1's complement of the coded number yields 9's complement of the number itself.

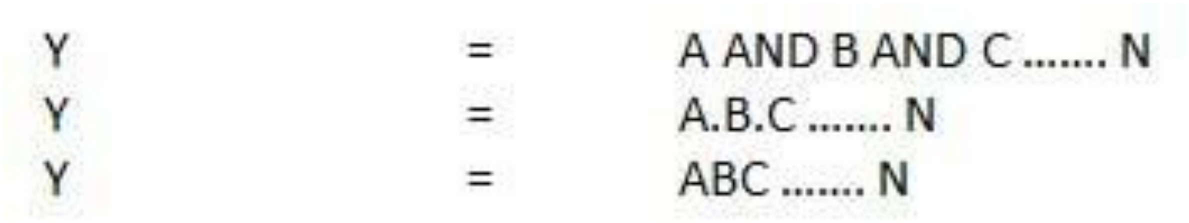
Used in digital system for performing substraction operations.

Logic gates

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

AND Gate

A circuit which performs an AND operation is shown in figure. It has n input (n >= 2) and one output.

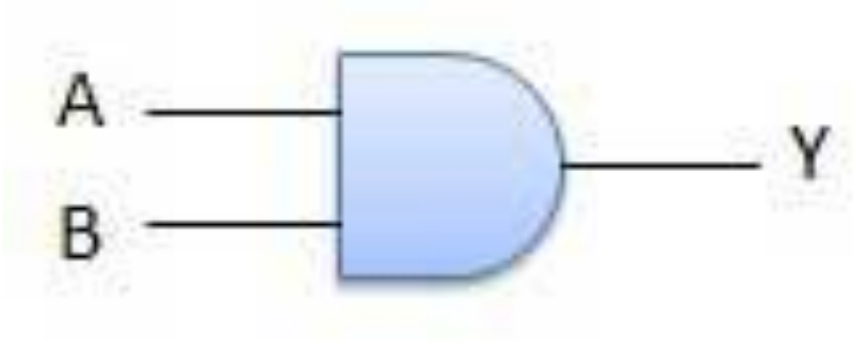


Y = A AND B AND C .... N

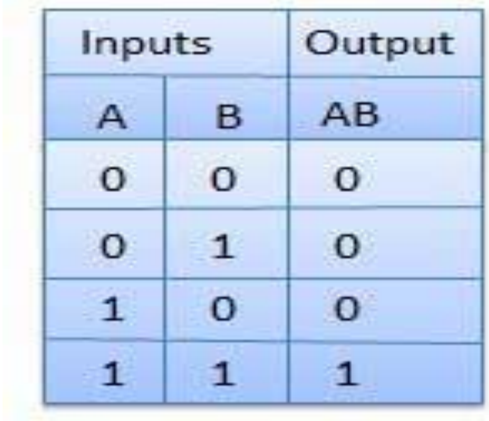
Y = A.B.C .... N

Y = ABC .... N

Logic diagram



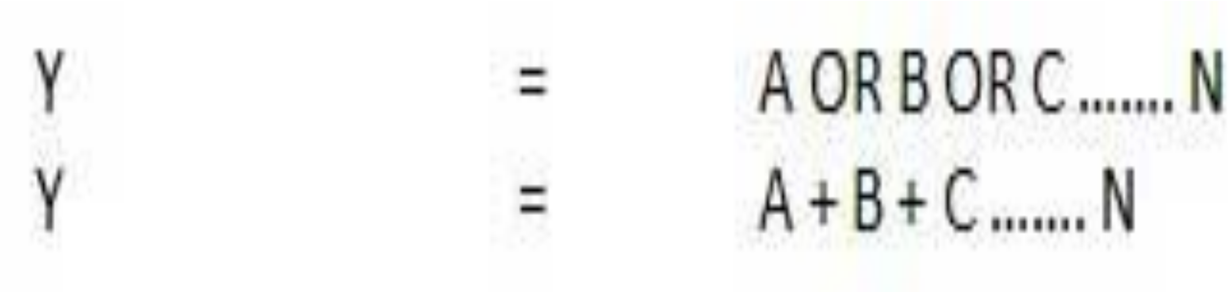
Truth Table



Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

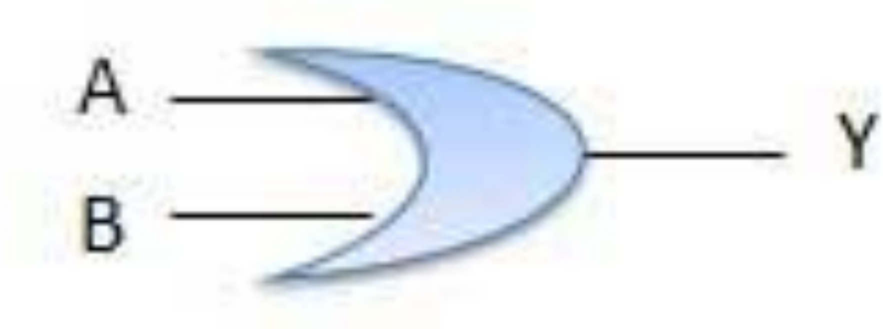
A circuit which performs an OR operation is shown in figure. It has n input (n >= 2) and one output.



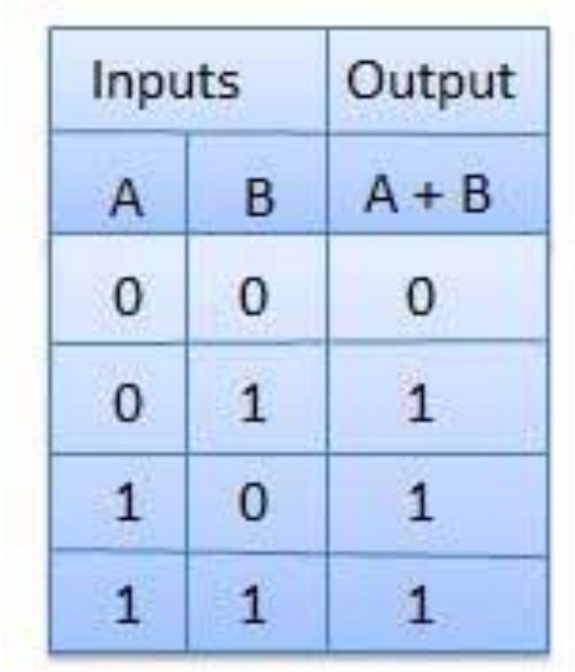
Y = A OR B OR C .... N

Y = A + B + C .... N

Logic diagram



Truth Table



Inputs		Output
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

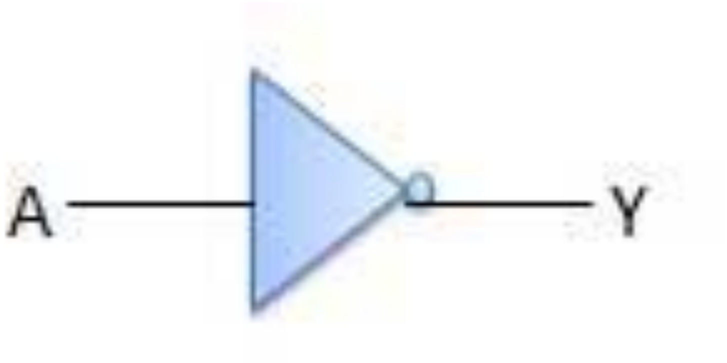
NOT Gate

NOT gate is also known as **Inverter**. It has one input A and one output Y.

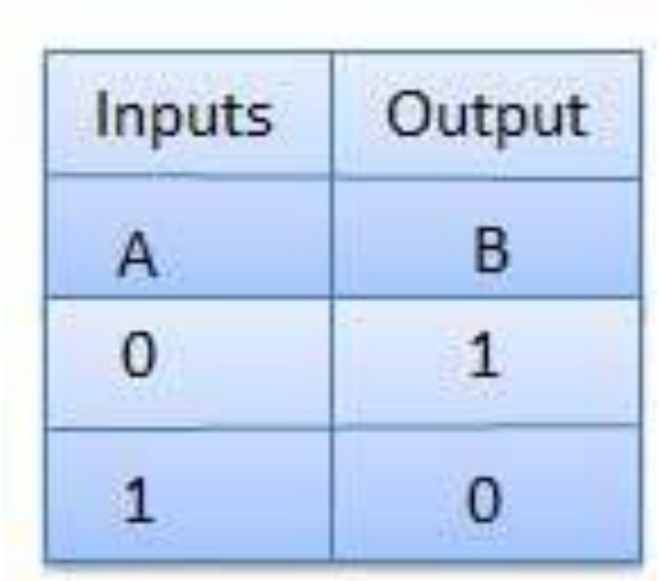
$$Y = \text{NOT } A$$

$$Y = \overline{A}$$

Logic diagram



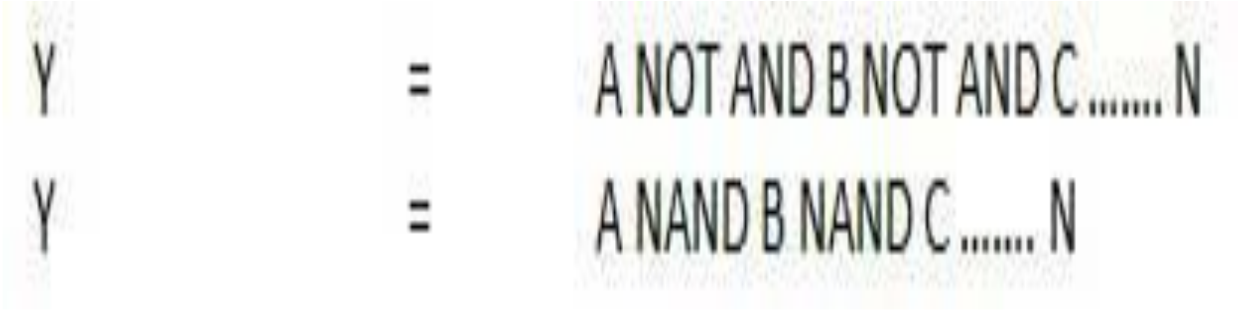
Truth Table



Inputs	Output
A	B
0	1
1	0

NAND Gate

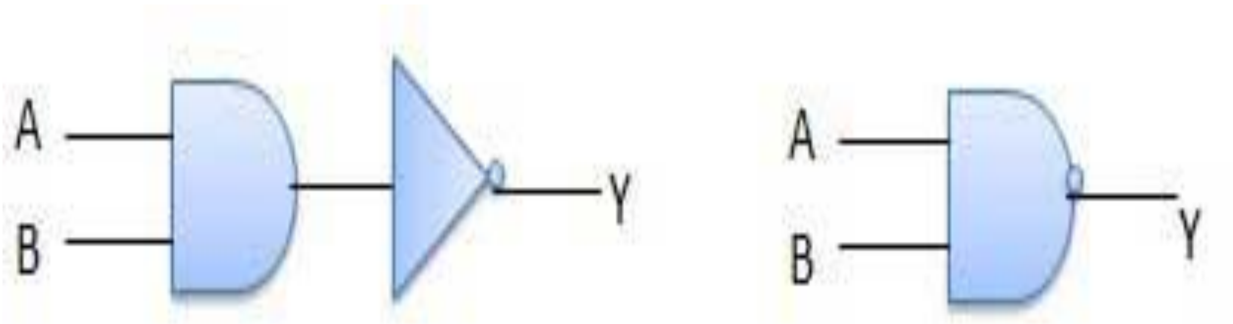
A NOT-AND operation is known as NAND operation. It has n input (n >= 2) and one output.



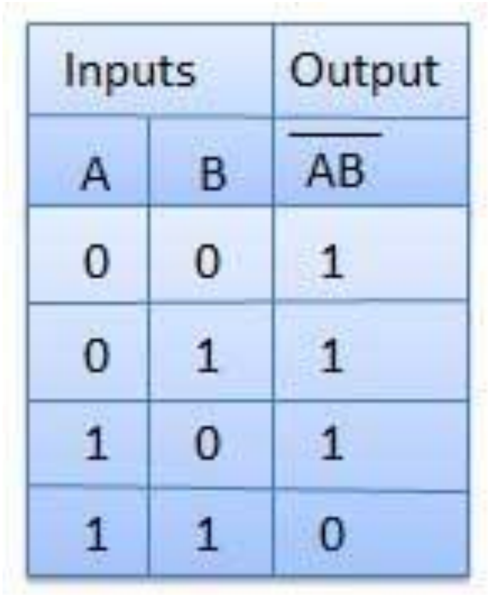
Y = A NOT AND B NOT AND C .... N

Y = A NAND B NAND C .... N

Logic diagram



Truth Table



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

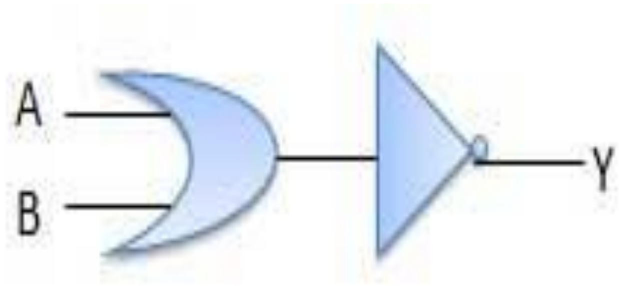
NOR Gate

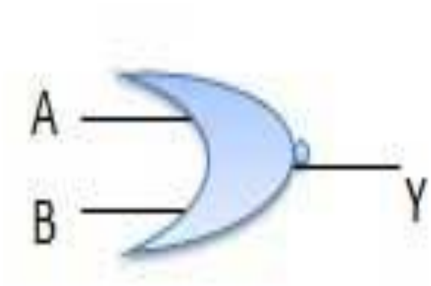
A NOT-OR operation is known as NOR operation. It has n input (n >= 2) and one output.

Y = A NOT OR B NOT OR C .... N

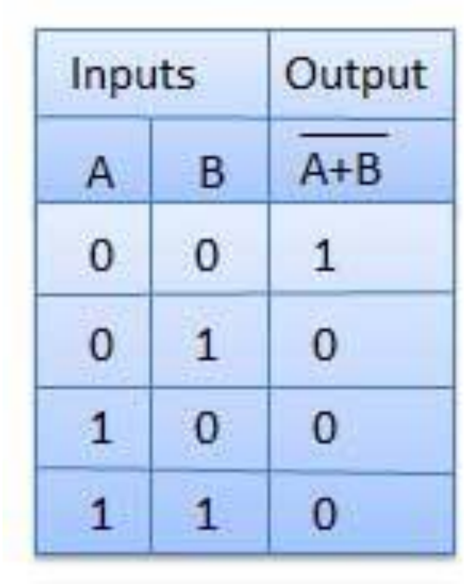
Y = A NOR B NOR C .... N

Logic diagram





Truth Table



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

XOR Gate

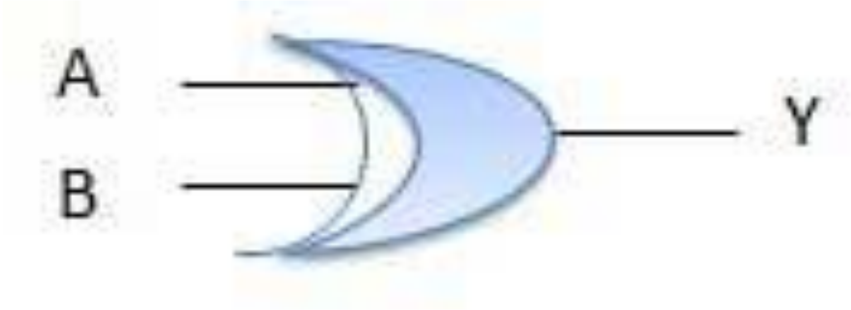
XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input (n >= 2) and one output.

$$Y = A \text{ XOR } B \text{ XOR } C \ldots\ldots. N$$

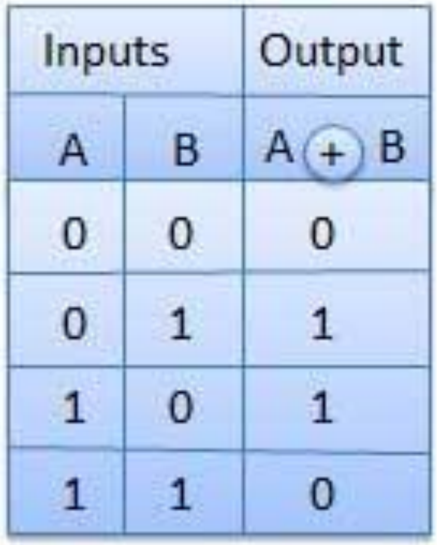
$$Y = A \oplus B \oplus C \ldots\ldots. N$$

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

Logic diagram



Truth Table



Inputs		Output
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate

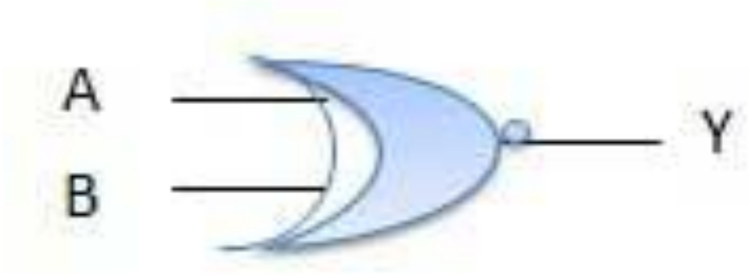
XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate. It has n input (n >= 2) and one output.

$$Y = A \text{ XOR } B \text{ XOR } C \ldots\ldots. N$$

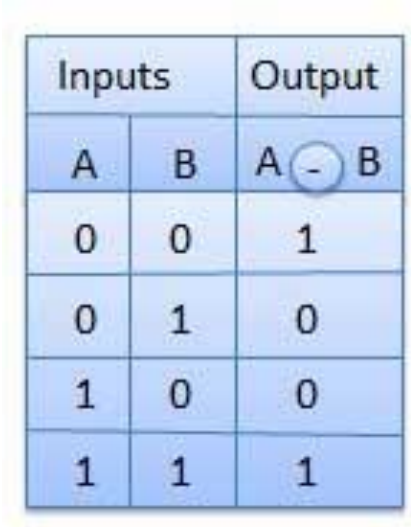
$$Y = A \odot B \odot C \ldots\ldots. N$$

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

Logic diagram



Truth Table



Inputs		Output
A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

Boolean Algebra

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

Rule in Boolean Algebra

Following are the important rules used in Boolean algebra.

- 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. Sometime the dot may be omitted like ABC.

Boolean Laws

There are six types of Boolean Laws.

Commutative law

Any binary operation which satisfies the following expression is referred to as commutative operation.

(i) $A.B = B.\ A$ (ii) $A + B = B + A$

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

Associative law

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

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

Distributive law

Distributive law states the following condition.

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

AND law

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

(i) $A.0 = 0$ (ii) $A.1 = A$

(iii) $A.A = A$ (iv) $A.\overline{A} = 0$

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) $A + \overline{A} = 1$

INVERSION law

This law uses the NOT operation. The inversion law states that double inversion of a variable results in the original variable itself.

$$\overline{\overline{A}} = A$$

Important Boolean Theorems

Following are few important boolean Theorems.

Boolean function/theorems	Description
Boolean Functions	Boolean Functions and Expressions, K-Map and NAND Gates realization
De Morgan's Theorems	De Morgan's Theorem 1 and Theorem 2

K-map simplification

In previous chapters, 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.

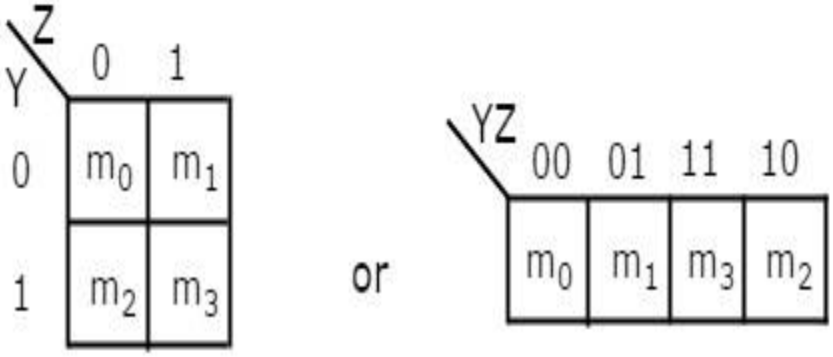
To overcome this difficulty, Karnaugh introduced a method for simplification of Boolean functions in an easy way. This method is known as Karnaugh map method or K-map method. It is a graphical method, which consists of 2^n cells for 'n' variables. The adjacent cells are differed only in single bit position.

K-Maps for 2 to 5 Variables

K-Map method is most suitable for minimizing Boolean functions of 2 variables to 5 variables. Now, let us discuss about the K-Maps for 2 to 5 variables one by one.

2 Variable K-Map

The number of cells in 2 variable K-map is four, since the number of variables is two. The following figure shows **2 variable K-Map**.

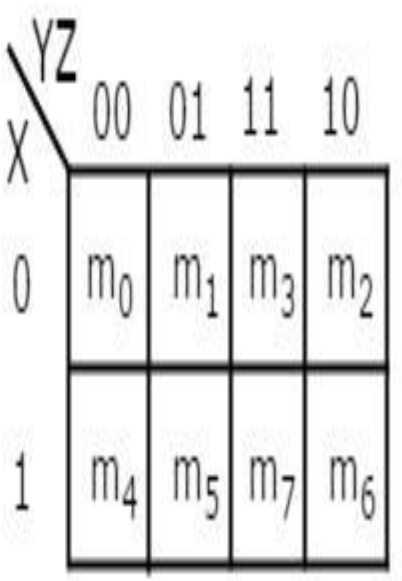


- There is only one possibility of grouping 4 adjacent min terms.

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

3 Variable K-Map

The number of cells in 3 variable K-map is eight, since the number of variables is three. The following figure shows **3 variable K-Map**.

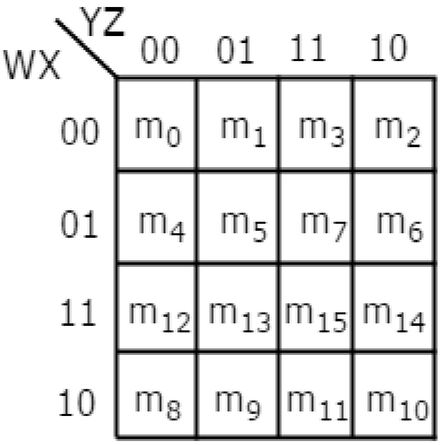


X \ YZ	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

- There is only one possibility of grouping 8 adjacent min terms.
- The possible combinations of grouping 4 adjacent min terms are {(m_0, m_1, m_3, m_2), (m_4, m_5, m_7, m_6), (m_0, m_1, m_4, m_5), (m_1, m_3, m_5, m_7), (m_3, m_2, m_7, m_6) and (m_2, m_0, m_6, m_4)}.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_1, m_3), (m_3, m_2), (m_2, m_0), (m_4, m_5), (m_5, m_7), (m_7, m_6), (m_6, m_4), (m_0, m_4), (m_1, m_5), (m_3, m_7) and (m_2, m_6)}.
- If x=0, then 3 variable K-map becomes 2 variable K-map.

4 Variable K-Map

The number of cells in 4 variable K-map is sixteen, since the number of variables is four. The following figure shows **4 variable K-Map**.

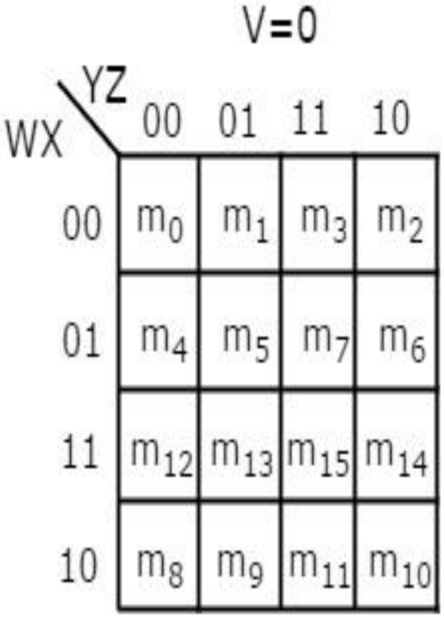


WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

- There is only one possibility of grouping 16 adjacent min terms.
- Let R_1, R_2, R_3 and R_4 represents the min terms of first row, second row, third row and fourth row respectively. Similarly, C_1, C_2, C_3 and C_4 represents the min terms of first column, second column, third column and fourth column respectively. The possible combinations of grouping 8 adjacent min terms are $\{(R_1, R_2), (R_2, R_3), (R_3, R_4), (R_4, R_1), (C_1, C_2), (C_2, C_3), (C_3, C_4), (C_4, C_1)\}$.
- If w=0, then 4 variable K-map becomes 3 variable K-map.

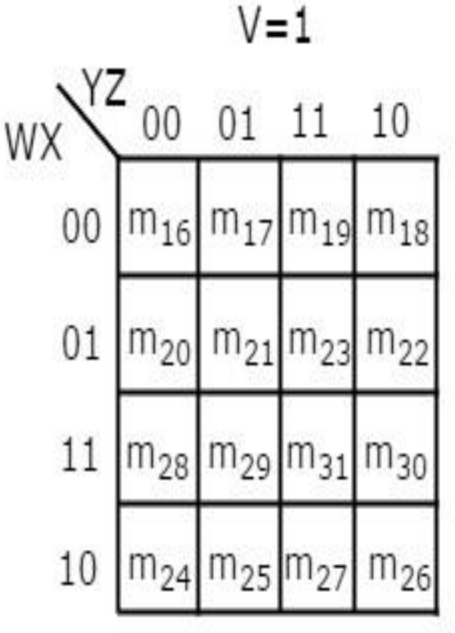
5 Variable K-Map

The number of cells in 5 variable K-map is thirty-two, since the number of variables is 5. The following figure shows 5 variable K-Map.



V=0

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



V=1

WX \ YZ	00	01	11	10
00	m_{16}	m_{17}	m_{19}	m_{18}
01	m_{20}	m_{21}	m_{23}	m_{22}
11	m_{28}	m_{29}	m_{31}	m_{30}
10	m_{24}	m_{25}	m_{27}	m_{26}

- There is only one possibility of grouping 32 adjacent min terms.
- There are two possibilities of grouping 16 adjacent min terms. i.e., grouping of min terms from m_0 to m_{15} and m_{16} to m_{31}.
- If v=0, then 5 variable K-map becomes 4 variable K-map.

In the above all K-maps, we used exclusively the min terms notation. Similarly, you can use exclusively the Max terms notation.

Minimization of Boolean Functions using K-Maps

If we consider the combination of inputs for which the Boolean function is ‘1’, then we will get the Boolean function, which is in standard sum of products form after simplifying the K-map.

Similarly, if we consider the combination of inputs for which the Boolean function is ‘0’, then we will get the Boolean function, which is in **standard product of sums** form after simplifying the K-map.

Follow these rules for simplifying K-maps in order to get standard sum of products form.

- Select the respective K-map based on the number of variables present in the Boolean function.
- If the Boolean function is given as sum of min terms form, then place the ones at respective min term cells in the K-map. If the Boolean function is given as sum of products form, then place the ones in all possible cells of K-map for which the given product terms are valid.
- Check for the possibilities of grouping maximum number of adjacent ones. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.
- Each grouping will give either a literal or one product term. It is known as primeimplicant. The prime implicant is said to be essential prime implicant, if atleast single ‘1’ is not covered with any other groupings but only that grouping covers.
- Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

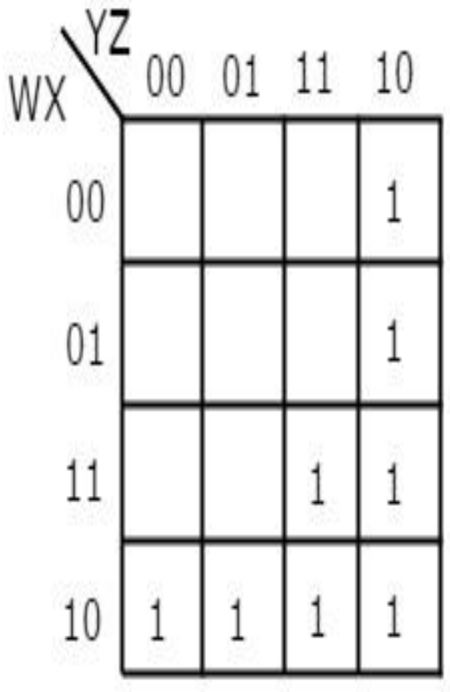
Note 1 − If outputs are not defined for some combination of inputs, then those output values will be represented with don’t care symbol ‘x’. That means, we can consider them as either ‘0’ or ‘1’.

Note 2 − If don’t care terms also present, then place don’t cares ‘x’ in the respective cells of K-map. Consider only the don’t cares ‘x’ that are helpful for grouping maximum number of adjacent ones. In those cases, treat the don’t care value as ‘1’.

Example

Let us **simplify** the following Boolean function, **f**W,X,Y,ZW,X,Y,Z**= WX’Y’ + WY + W’YZ’** using K-map.

The given Boolean function is in sum of products form. It is having 4 variables W, X, Y & Z. So, we require **4 variable K-map**. The **4 variable K-map** with ones corresponding to the given product terms is shown in the following figure.

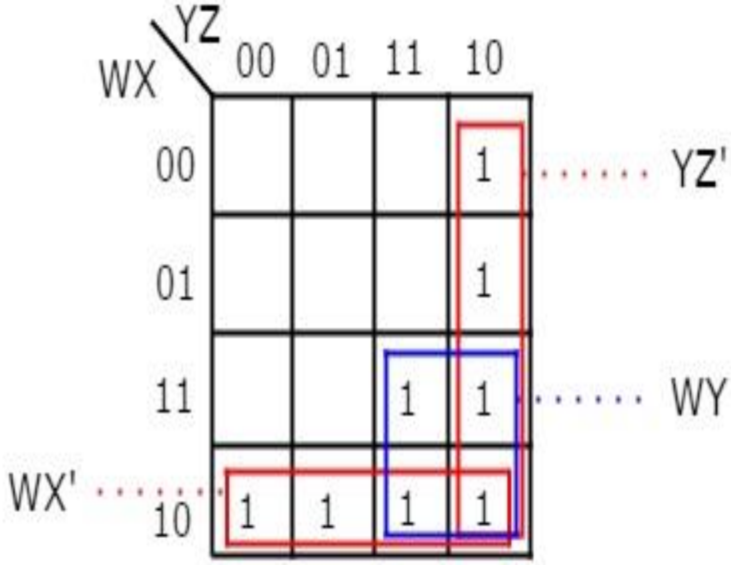


WX \ YZ	00	01	11	10
00				1
01				1
11			1	1
10	1	1	1	1

Here, 1s are placed in the following cells of K-map.

- The cells, which are common to the intersection of Row 4 and columns 1 & 2 are corresponding to the product term, **WX’Y’**.
- The cells, which are common to the intersection of Rows 3 & 4 and columns 3 & 4 are corresponding to the product term, **WY**.
- The cells, which are common to the intersection of Rows 1 & 2 and column 4 are corresponding to the product term, **W’YZ’**.

There are no possibilities of grouping either 16 adjacent ones or 8 adjacent ones. There are three possibilities of grouping 4 adjacent ones. After these three groupings, there is no single one left as ungrouped. So, we no need to check for grouping of 2 adjacent ones. The **4 variable K-map** with these three **groupings** is shown in the following figure.



Here, we got three prime implicants WX’, WY & YZ’. All these prime implicants are **essential** because of following reasons.

- Two ones **(m_8 & m_9)** of fourth row grouping are not covered by any other groupings. Only fourth row grouping covers those two ones.
- Single one **(m_{15})** of square shape grouping is not covered by any other groupings. Only the square shape grouping covers that one.
- Two ones **(m_2 & m_6)** of fourth column grouping are not covered by any other groupings. Only fourth column grouping covers those two ones.

Therefore, the **simplified Boolean function** is

f = WX’ + WY + YZ’

Follow these **rules for simplifying K-maps** in order to get standard product of sums form.

- Select the respective K-map based on the number of variables present in the Boolean function.
- If the Boolean function is given as product of Max terms form, then place the zeroes at respective Max term cells in the K-map. If the Boolean function is given as product of sums form, then place the zeroes in all possible cells of K-map for which the given sum terms are valid.
- Check for the possibilities of grouping maximum number of adjacent zeroes. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.
- Each grouping will give either a literal or one sum term. It is known as **prime implicant**. The prime implicant is said to be **essential prime implicant**, if

atleast single ‘0’ is not covered with any other groupings but only that grouping covers.

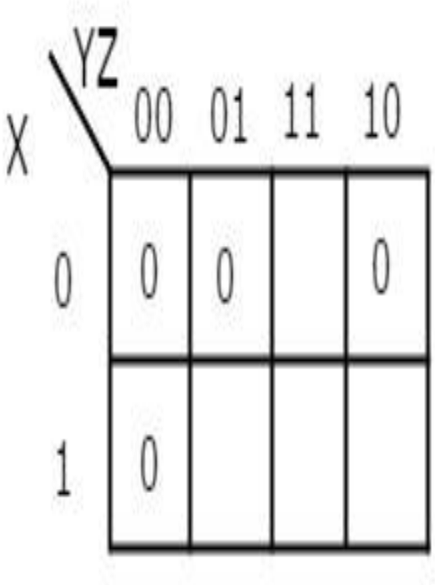
- Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

Note − If don’t care terms also present, then place don’t cares ‘x’ in the respective cells of K-map. Consider only the don’t cares ‘x’ that are helpful for grouping maximum number of adjacent zeroes. In those cases, treat the don’t care value as ‘0’.

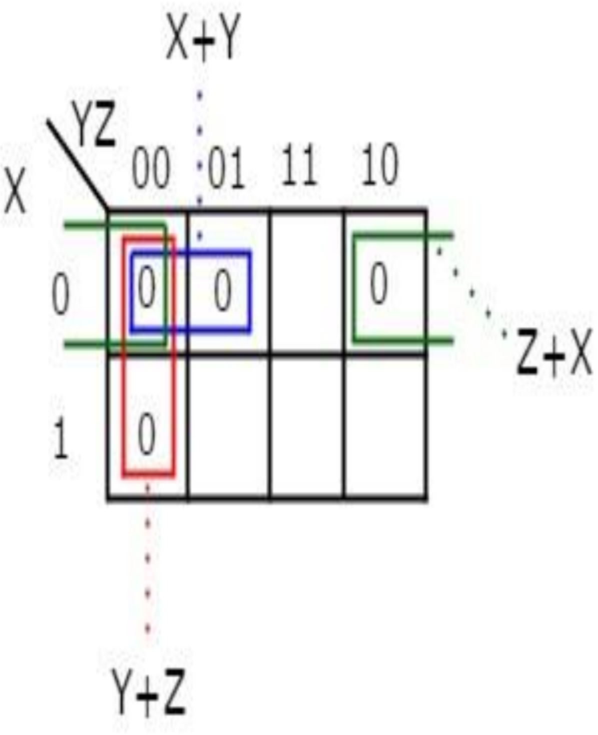
Example

Let us **simplify** the following Boolean function, $f(X,Y,Z)=\prod M(0,1,2,4)f(X,Y,Z)=\prod M(0,1,2,4)$ using K-map.

The given Boolean function is in product of Max terms form. It is having 3 variables X, Y & Z. So, we require 3 variable K-map. The given Max terms are M_0, M_1, M_2 & M_4. The 3 **variable K-map** with zeroes corresponding to the given Max terms is shown in the following figure.



There are no possibilities of grouping either 8 adjacent zeroes or 4 adjacent zeroes. There are three possibilities of grouping 2 adjacent zeroes. After these three groupings, there is no single zero left as ungrouped. The **3 variable K-map** with these three **groupings** is shown in the following figure.



Here, we got three prime implicants X + Y, Y + Z & Z + X. All these prime implicants are **essential** because one zero in each grouping is not covered by any other groupings except with their individual groupings.

Therefore, the **simplified Boolean function** is

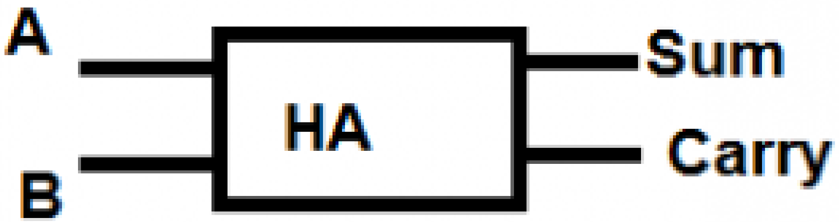
f = X+YX+Y**.**Y+ZY+Z**.**Z+XZ+X

In this way, we can easily simplify the Boolean functions up to 5 variables using K-map method. For more than 5 variables, it is difficult to simplify the functions using K-Maps. Because, the number of **cells** in K-map gets **doubled** by including a new variable.

Due to this checking and grouping of adjacent ones mintermsminterms or adjacent zeros MaxtermsMaxterms will be complicated. We will discuss **Tabular method** in next chapter to overcome the difficulties of K-map method.

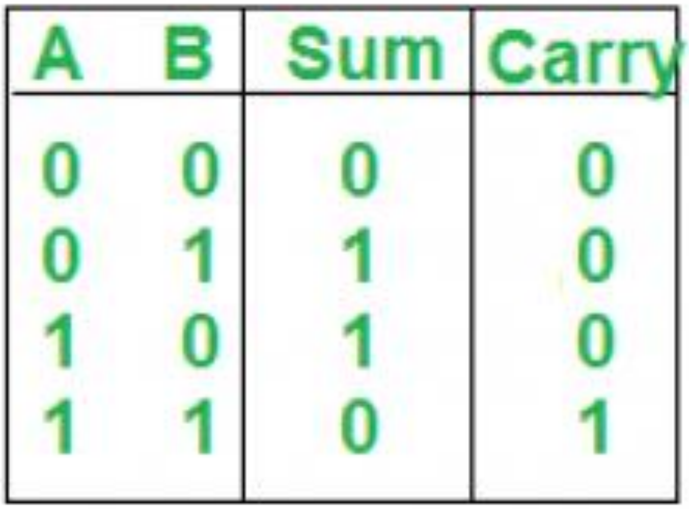
Half Adder

The addition of 2 bits is done using a combination circuit called Half adder. The input variables are augend and addend bits and output variables are sum & carry bits. A and B are the two input bits.



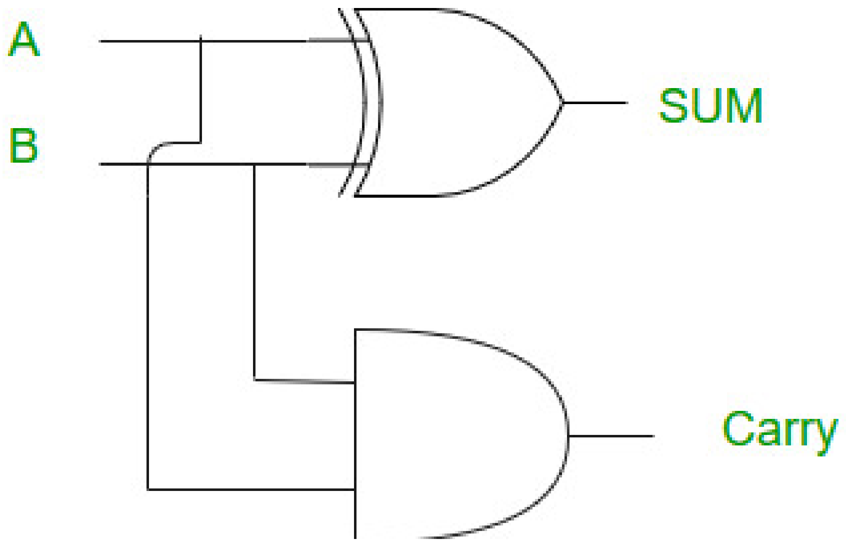
Half Adder

Truth Table:



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

Logical Expression:
Sum = A XOR B
Carry = A AND B
Implementation:

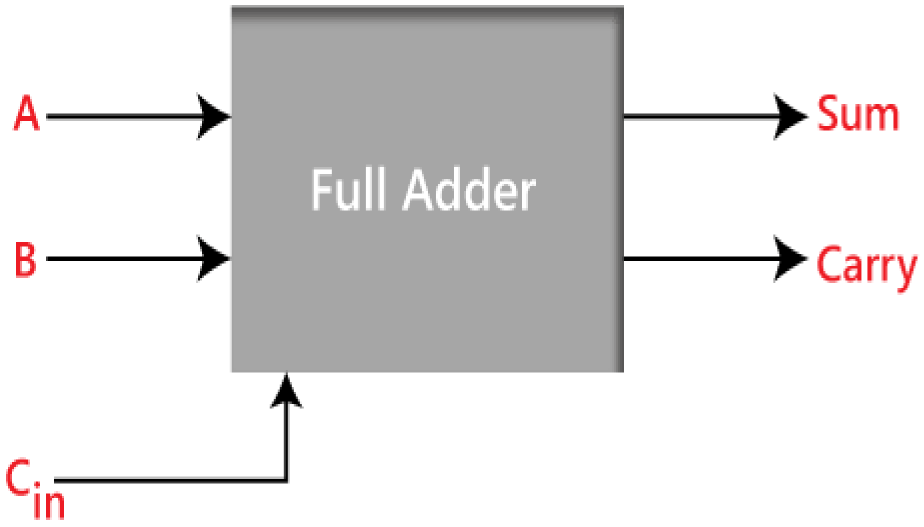


Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

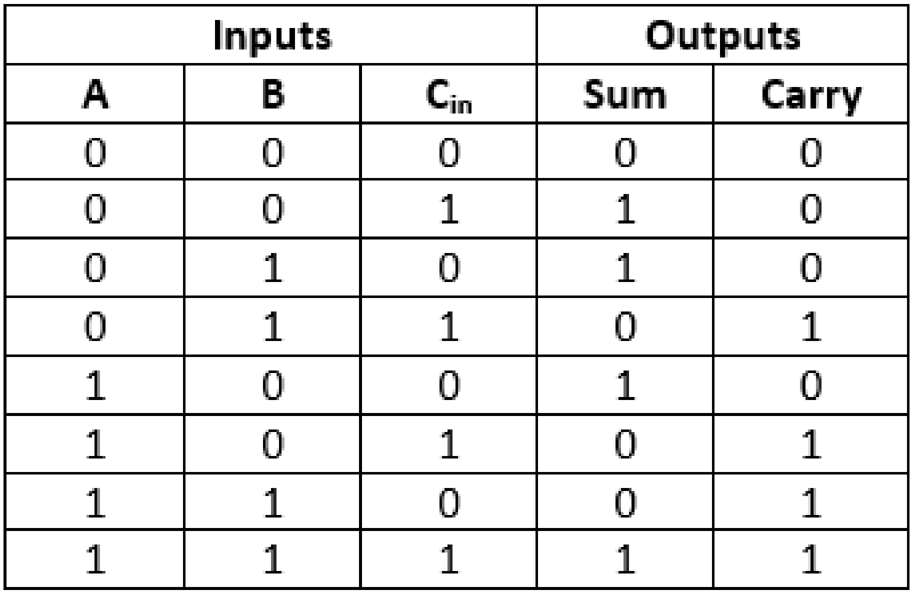
Full Adder

The half adder is used to add only two numbers. To overcome this problem, the full adder was developed. The full adder is used to add three 1-bit binary numbers A, B, and carry C. The full adder has three input states and two output states i.e., sum and carry.

Block diagram



Truth Table

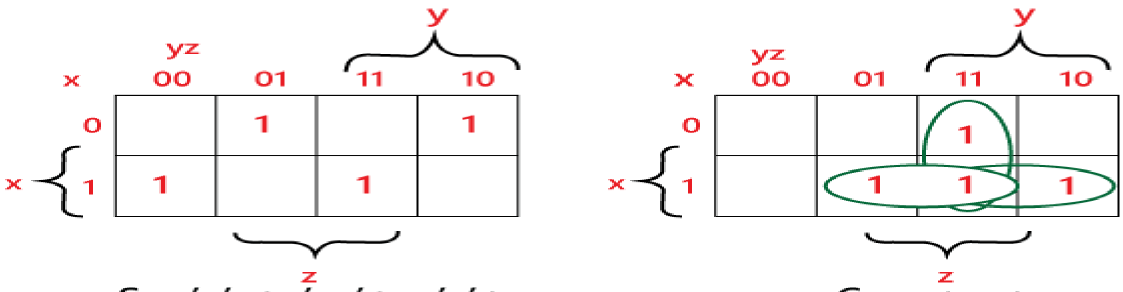


Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

In the above table,

1. 'A' and' B' are the input variables. These variables represent the two significant bits which are going to be added
2. 'C_{in}' is the third input which represents the carry. From the previous lower significant position, the carry bit is fetched.
3. The 'Sum' and 'Carry' are the output variables that define the output values.
4. The eight rows under the input variable designate all possible combinations of 0 and 1 that can occur in these variables.

The SOP form can be obtained with the help of K-map as:



S=x'y'z+x'yz'+xy'z'+xyz

C=xy+xz+yz

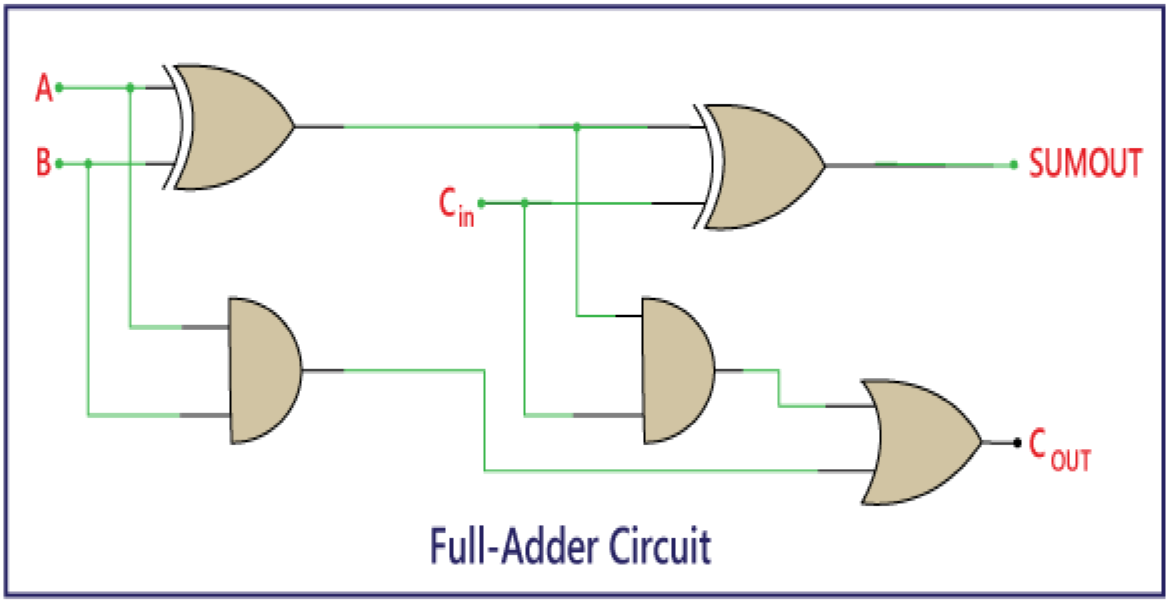
Sum = x' y' z+x' yz+xy' z'+xyz
Carry = xy+xz+yz

Construction of Half Adder Circuit:

The above block diagram describes the construction of the Full adder circuit. In the above circuit, there are two half adder circuits that are combined using the OR gate. The first half adder has two single-bit binary inputs A and B. As we know that, the half adder produces two outputs, i.e., Sum and Carry. The 'Sum' output of the first adder will be the first input of the second half adder, and the 'Carry' output of the first adder will be the second input of the second half adder. The second half adder will again provide 'Sum' and 'Carry'. The final outcome of the Full adder circuit is the 'Sum' bit. In order to find the final output of the 'Carry', we provide the 'Carry' output of the first and the second adder into the OR gate. The outcome of the OR gate will be the final carry out of the full adder circuit.

The MSB is represented by the final 'Carry' bit.

The full adder logic circuit can be constructed using the **'AND'** and **the 'XOR' gate** with an **OR gate**.



The actual logic circuit of the full adder is shown in the above diagram. The full adder circuit construction can also be represented in a Boolean expression.

Sum:

- Perform the XOR operation of input A and B.
- Perform the XOR operation of the outcome with carry. So, the sum is (A XOR B) XOR C_{in} which is also represented as:
  $(A \oplus B) \oplus C_{in}$

Carry:

1. Perform the 'AND' operation of input A and B.
2. Perform the 'XOR' operation of input A and B.
3. Perform the 'OR' operations of both the outputs that come from the previous two steps. So the 'Carry' can be represented as:
   $A.B + (A \oplus B)$

Subtractor

subtractor An electronic logic circuit for calculating the difference between two binary numbers, the minuend and the number to be subtracted, the subtrahend (see table).
A full subtractor performs this calculation with three inputs: minuend bit, subtrahend bit, and borrow bit. It produces two outputs: the difference and the borrow. Full subtractors thus allow for the inclusion of borrows generated by previous stages of subtraction when forming their output signals, and can be cascaded to form n-bit subtractors. Alternatively the subtract operation can be performed using two half subtractors, which are simpler since they contain only two inputs and produce two outputs.

Decoder

Decoder is a combinational circuit that has 'n' input lines and maximum of 2^n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the **min terms** of 'n' input variables lineslines, when it is enabled.

2 to 4 Decoder

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

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

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

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

$$Y3=E.A1.A0Y3=E.A1.A0$$

$$Y2=E.A1.A0'Y2=E.A1.A0'$$

$$Y1=E.A1'.A0Y1=E.A1'.A0$$

$$Y0=E.A1'.A0'Y0=E.A1'.A0'$$

Each output is having one product term. So, there are four product terms in total. We can implement these four product terms by using four AND gates having three inputs each & two inverters. The **circuit diagram** of 2 to 4 decoder is shown in the following figure.

Therefore, the outputs of 2 to 4 decoder are nothing but the **min terms** of two input variables A_1 & A_0, when enable, E is equal to one. If enable, E is zero, then all the outputs of decoder will be equal to zero.

Similarly, 3 to 8 decoder produces eight min terms of three input variables A_2, A_1 & A_0 and 4 to 16 decoder produces sixteen min terms of four input variables A_3, A_2, A_1 & A_0.

Implementation of Higher-order Decoders

Now, let us implement the following two higher-order decoders using lower-order decoders.

- 3 to 8 decoder
- 4 to 16 decoder

3 to 8 Decoder

In this section, let us implement **3 to 8 decoder using 2 to 4 decoders**. We know that 2 to 4 Decoder has two inputs, A_1 & A_0 and four outputs, Y_3 to Y_0. Whereas, 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0.

We can find the number of lower order decoders required for implementing higher order decoder using the following formula.

Requirednumberoflowerorderdecoders=m2m1Requirednumberoflowerorderdecoders=m2m1

Where,

m1m1 is the number of outputs of lower order decoder.
m2m2 is the number of outputs of higher order decoder.
Here, m1m1 = 4 and m2m2 = 8. Substitute, these two values in the above formula.

Requirednumberof2to4decoders=84=2Requirednumberof2to4decoders=84=2

Therefore, we require two 2 to 4 decoders for implementing one 3 to 8 decoder. The **block diagram** of 3 to 8 decoder using 2 to 4 decoders is shown in the following figure.

The parallel inputs A_1 & A_0 are applied to each 2 to 4 decoder. The complement of input A_2 is connected to Enable, E of lower 2 to 4 decoder in order to get the outputs, Y_3 to Y_0. These are the **lower four min terms**. The input, A_2 is directly connected to Enable, E of upper 2 to 4 decoder in order to get the outputs, Y_7 to Y_4. These are the **higher four min terms**.

4 to 16 Decoder

In this section, let us implement **4 to 16 decoder using 3 to 8 decoders**. We know that 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0. Whereas, 4 to 16 Decoder has four inputs A_3, A_2, A_1 & A_0 and sixteen outputs, Y_{15} to Y_0

We know the following formula for finding the number of lower order decoders required.

Requirednumberoflowerorderdecoders=m2m1Requirednumberoflowerorderdecoders=m2m1

Substitute, m1m1 = 8 and m2m2 = 16 in the above formula.

Requirednumberof3to8decoders=168=2Requirednumberof3to8decoders=168=2

Therefore, we require two 3 to 8 decoders for implementing one 4 to 16 decoder. The **block diagram** of 4 to 16 decoder using 3 to 8 decoders is shown in the following figure.

Encoders

An **Encoder** is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2^n input lines and 'n' output lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with 'n' bits. It is optional to represent the enable signal in encoders.

4 to 2 Encoder

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

At any time, only one of these 4 inputs can be ‘1’ in order to get the respective binary code at the output. The **Truth table** of 4 to 2 encoder is shown below.

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

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

$$A1=Y3+Y2A1=Y3+Y2$$

$$A0=Y3+Y1A0=Y3+Y1$$

We can implement the above two Boolean functions by using two input OR gates. The **circuit diagram** of 4 to 2 encoder is shown in the following figure.

The above circuit diagram contains two OR gates. These OR gates encode the four inputs with two bits

Octal to Binary Encoder

Octal to binary Encoder has eight inputs, Y_7 to Y_0 and three outputs A_2, A_1 & A_0. Octal to binary encoder is nothing but 8 to 3 encoder. The **block diagram** of octal to binary Encoder is shown in the following figure.

At any time, only one of these eight inputs can be ‘1’ in order to get the respective binary code. The **Truth table** of octal to binary encoder is shown below.

Inputs								**Outputs**		
Y_7	**Y_6**	**Y_5**	**Y_4**	**Y_3**	**Y_2**	**Y_1**	**Y_0**	**A_2**	**A_1**	**A_0**
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

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

A2=Y7+Y6+Y5+Y4A2=Y7+Y6+Y5+Y4

A1=Y7+Y6+Y3+Y2A1=Y7+Y6+Y3+Y2

A0=Y7+Y5+Y3+Y1A0=Y7+Y5+Y3+Y1

We can implement the above Boolean functions by using four input OR gates. The **circuit diagram** of octal to binary encoder is shown in the following figure.

The above circuit diagram contains three 4-input OR gates. These OR gates encode the eight inputs with three bits.

Drawbacks of Encoder

Following are the drawbacks of normal encoder.

- There is an ambiguity, when all outputs of encoder are equal to zero. Because, it could be the code corresponding to the inputs, when only least significant input is one or when all inputs are zero.
- If more than one input is active High, then the encoder produces an output, which may not be the correct code. For **example**, if both Y_3 and Y_6 are '1', then the encoder produces 111 at the output. This is neither equivalent code corresponding to Y_3, when it is '1' nor the equivalent code corresponding to Y_6, when it is '1'.

So, to overcome these difficulties, we should assign priorities to each input of encoder. Then, the output of encoder will be the binarybinary code corresponding to the active High inputss, which has higher priority. This encoder is called as **priority encoder**.

Priority Encoder

A 4 to 2 priority encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. Here, the input, Y_3 has the highest priority, whereas the input, Y_0 has the lowest priority. In this case, even if more than one input is '1' at the same time, the output will be the binarybinary code corresponding to the input, which is having **higher priority**.

We considered one more **output, V** in order to know, whether the code available at outputs is valid or not.

- If at least one input of the encoder is '1', then the code available at outputs is a valid one. In this case, the output, V will be equal to 1.
- If all the inputs of encoder are '0', then the code available at outputs is not a valid one. In this case, the output, V will be equal to 0.

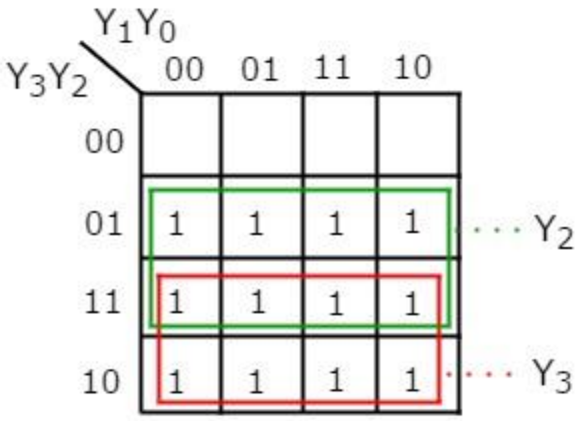
The **Truth table** of 4 to 2 priority encoder is shown below.

Inputs				**Outputs**		
Y_3	**Y_2**	**Y_1**	**Y_0**	**A_1**	**A_0**	**V**
0	0	0	0	0	0	0

0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

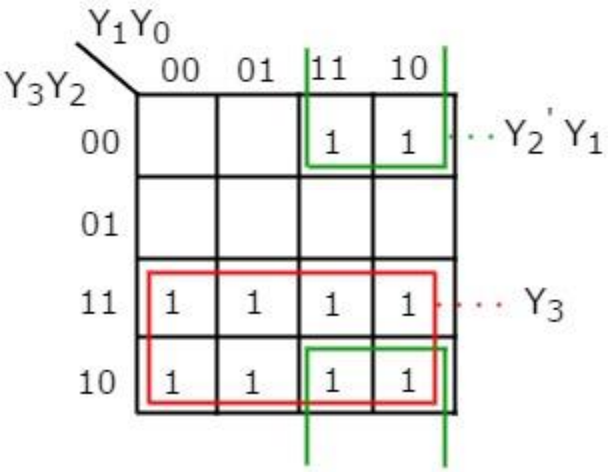
Use **4 variable K-maps** for getting simplified expressions for each output.

K-Map for A_1



Y_3Y_2 \ Y_1Y_0	00	01	11	10
00				
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

K-Map for A_0



Y_3Y_2 \ Y_1Y_0	00	01	11	10
00			1	1
01				
11	1	1	1	1
10	1	1	1	1

The simplified **Boolean functions** are

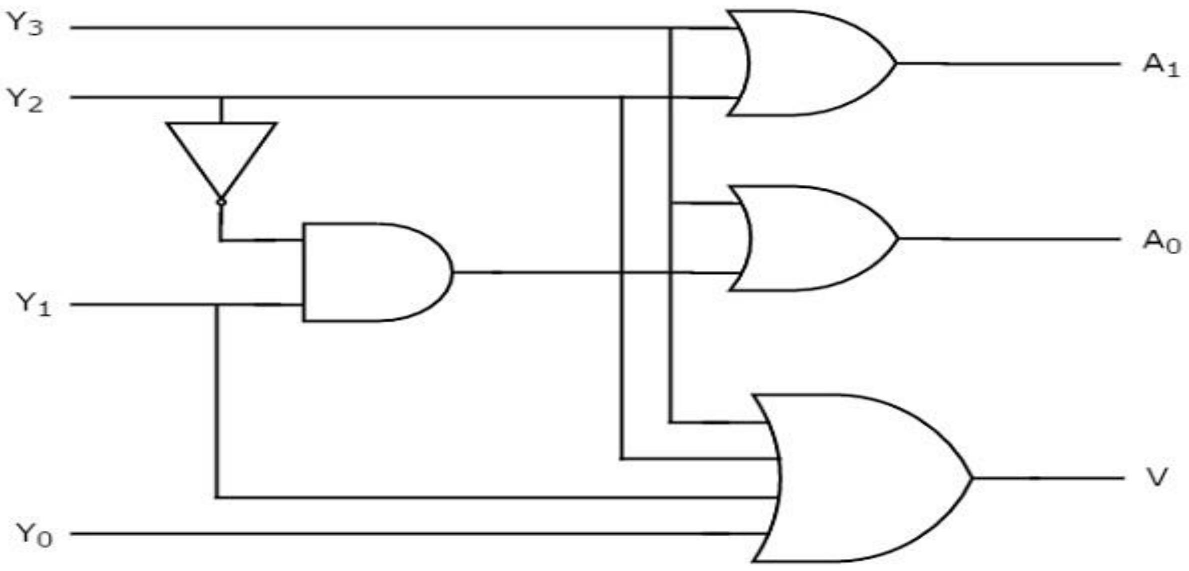
$$A1=Y3+Y2A1=Y3+Y2$$

$A0=Y3+Y2′Y1A0=Y3+Y2′Y1$

Similarly, we will get the Boolean function of output, V as

$$V=Y3+Y2+Y1+Y0V=Y3+Y2+Y1+Y0$$

We can implement the above Boolean functions using logic gates. The **circuit diagram** of 4 to 2 priority encoder is shown in the following figure.



The above circuit diagram contains two 2-input OR gates, one 4-input OR gate, one 2input AND gate & an inverter. Here AND gate & inverter combination are used for producing a valid code at the outputs, even when multiple inputs are equal to ‘1’ at the same time. Hence, this circuit encodes the four inputs with two bits based on the **priority** assigned to each input.

The parallel inputs A_2, A_1 & A_0 are applied to each 3 to 8 decoder. The complement of input, A3 is connected to Enable, E of lower 3 to 8 decoder in order to get the outputs, Y_7 to Y_0. These are the **lower eight min terms**. The input, A_3 is directly connected to Enable, E of upper 3 to 8 decoder in order to get the outputs, Y_{15} to Y_8. These are the **higher eight min terms**.

Multiplexer

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

Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called as **Mux**.

4x1 Multiplexer

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

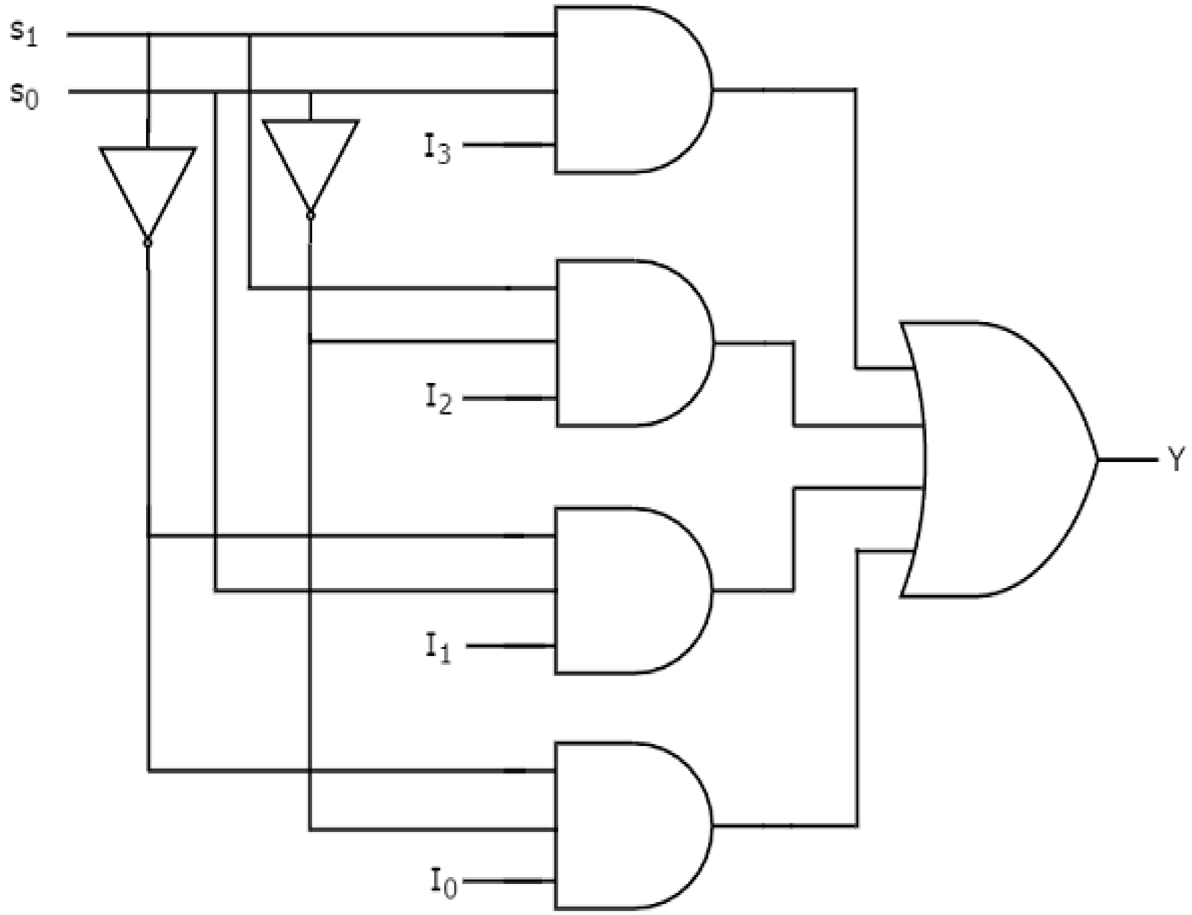
One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below.

Selection Lines		**Output**
$\mathbf{S_1}$	$\mathbf{S_0}$	**Y**
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

From Truth table, we can directly write the **Boolean function** for output, Y as

Y=S1′S0′I0+S1′S0I1+S1S0′I2+S1S0I3Y=S1′S0′I0+S1′S0I1+S1S0′I2+S1S0I3

We can implement this Boolean function using Inverters, AND gates & OR gate. The **circuit diagram** of 4x1 multiplexer is shown in the following figure.



We can easily understand the operation of the above circuit. Similarly, you can implement 8x1 Multiplexer and 16x1 multiplexer by following the same procedure.

Implementation of Higher-order Multiplexers.

Now, let us implement the following two higher-order Multiplexers using lower-order Multiplexers.

- 8x1 Multiplexer
- 16x1 Multiplexer

8x1 Multiplexer

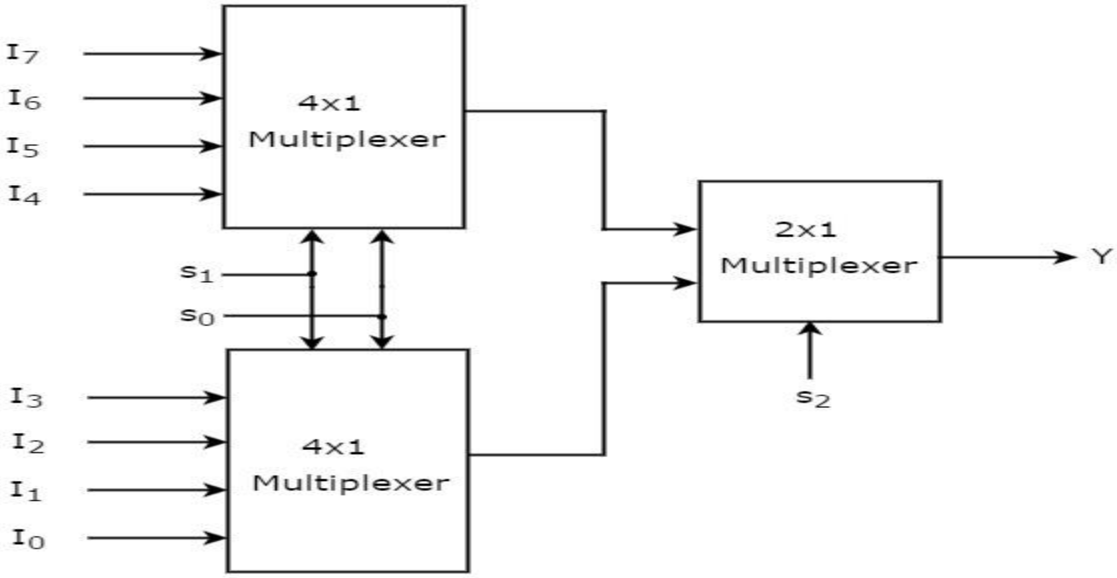
In this section, let us implement 8x1 Multiplexer using 4x1 Multiplexers and 2x1 Multiplexer. We know that 4x1 Multiplexer has 4 data inputs, 2 selection lines and one output. Whereas, 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output.

So, we require two **4x1 Multiplexers** in first stage in order to get the 8 data inputs. Since, each 4x1 Multiplexer produces one output, we require a **2x1 Multiplexer** in second stage by considering the outputs of first stage as inputs and to produce the final output.

Let the 8x1 Multiplexer has eight data inputs I_7 to I_0, three selection lines s_2, s_1 & s0 and one output Y. The **Truth table** of 8x1 Multiplexer is shown below.

Selection Inputs			**Output**
S_2	**S_1**	**S_0**	**Y**
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

We can implement 8x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 8x1 Multiplexer is shown in the following figure.



The same **selection lines, s_1 & s_0** are applied to both 4x1 Multiplexers. The data inputs of upper 4x1 Multiplexer are I_7 to I_4 and the data inputs of lower 4x1 Multiplexer are I_3 to I_0. Therefore, each 4x1 Multiplexer produces an output based on the values of selection lines, s_1 & s_0.

The outputs of first stage 4x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_2** is applied to 2x1 Multiplexer.

- If s_2 is zero, then the output of 2x1 Multiplexer will be one of the 4 inputs I_3 to I_0 based on the values of selection lines s_1 & s_0.
- If s_2 is one, then the output of 2x1 Multiplexer will be one of the 4 inputs I_7 to I_4 based on the values of selection lines s_1 & s_0.

Therefore, the overall combination of two 4x1 Multiplexers and one 2x1 Multiplexer performs as one 8x1 Multiplexer.

16x1 Multiplexer

In this section, let us implement 16x1 Multiplexer using 8x1 Multiplexers and 2x1 Multiplexer. We know that 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output. Whereas, 16x1 Multiplexer has 16 data inputs, 4 selection lines and one output.

So, we require two **8x1 Multiplexers** in first stage in order to get the 16 data inputs. Since, each 8x1 Multiplexer produces one output, we require a 2x1 Multiplexer in second stage by considering the outputs of first stage as inputs and to produce the final output.

Let the 16x1 Multiplexer has sixteen data inputs I_{15} to I_0, four selection lines s_3 to s_0 and one output Y. The **Truth table** of 16x1 Multiplexer is shown below.

Selection Inputs	Output

S_3	**S_2**	**S_1**	**S_0**	**Y**
0	0	0	0	I_0
0	0	0	1	I_1
0	0	1	0	I_2
0	0	1	1	I_3
0	1	0	0	I_4
0	1	0	1	I_5
0	1	1	0	I_6
0	1	1	1	I_7
1	0	0	0	I_8
1	0	0	1	I_9
1	0	1	0	I_{10}
1	0	1	1	I_{11}
1	1	0	0	I_{12}
1	1	0	1	I_{13}
1	1	1	0	I_{14}
1	1	1	1	I_{15}

We can implement 16x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 16x1 Multiplexer is shown in the following figure.

The **same selection lines, s_2, s_1 & s_0** are applied to both 8x1 Multiplexers. The data inputs of upper 8x1 Multiplexer are I_{15} to I_8 and the data inputs of lower 8x1 Multiplexer are I_7 to I_0. Therefore, each 8x1 Multiplexer produces an output based on the values of selection lines, s_2, s_1 & s_0.

The outputs of first stage 8x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_3** is applied to 2x1 Multiplexer.

- If s_3 is zero, then the output of 2x1 Multiplexer will be one of the 8 inputs Is_7 to I_0 based on the values of selection lines s_2, s_1 & s_0.
- If s_3 is one, then the output of 2x1 Multiplexer will be one of the 8 inputs I_{15} to I_8 based on the values of selection lines s_2, s_1 & s_0.

Therefore, the overall combination of two 8x1 Multiplexers and one 2x1 Multiplexer performs as one 16x1 Multiplexer.

Demultiplexer

De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, 'n' selection lines and maximum of 2^n outputs. The input will be connected to one of these outputs based on the values of selection lines.

Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as **De-Mux**.

1x4 De-Multiplexer

1x4 De-Multiplexer has one input I, two selection lines, s_1 & s_0 and four outputs Y_3, Y_2, Y_1 &Y_0. The **block diagram** of 1x4 De-Multiplexer is shown in the following figure.

The single input ‘I’ will be connected to one of the four outputs, Y_3 to Y_0 based on the values of selection lines s_1 & s0. The **Truth table** of 1x4 De-Multiplexer is shown below.

Selection Inputs		**Outputs**			
S_1	**S_0**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	0	0	0	0	**I**
0	1	0	0	**I**	0
1	0	0	**I**	0	0
1	1	**I**	0	0	0

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

$$Y3=s1s0IY3=s1s0I$$

$$Y2=s1s0'IY2=s1s0'I$$

$$Y1=s1's0IY1=s1's0I$$

$$Y0=s1's0'IY0=s1's0'I$$

We can implement these Boolean functions using Inverters & 3-input AND gates. The **circuit diagram** of 1x4 De-Multiplexer is shown in the following figure.

We can easily understand the operation of the above circuit. Similarly, you can implement 1x8 De-Multiplexer and 1x16 De-Multiplexer by following the same procedure.

Implementation of Higher-order De-Multiplexers

Now, let us implement the following two higher-order De-Multiplexers using lower-order De-Multiplexers.

- 1x8 De-Multiplexer
- 1x16 De-Multiplexer

1x8 De-Multiplexer

In this section, let us implement 1x8 De-Multiplexer using 1x4 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x4 De-Multiplexer has single input, two selection lines and four outputs. Whereas, 1x8 De-Multiplexer has single input, three selection lines and eight outputs.

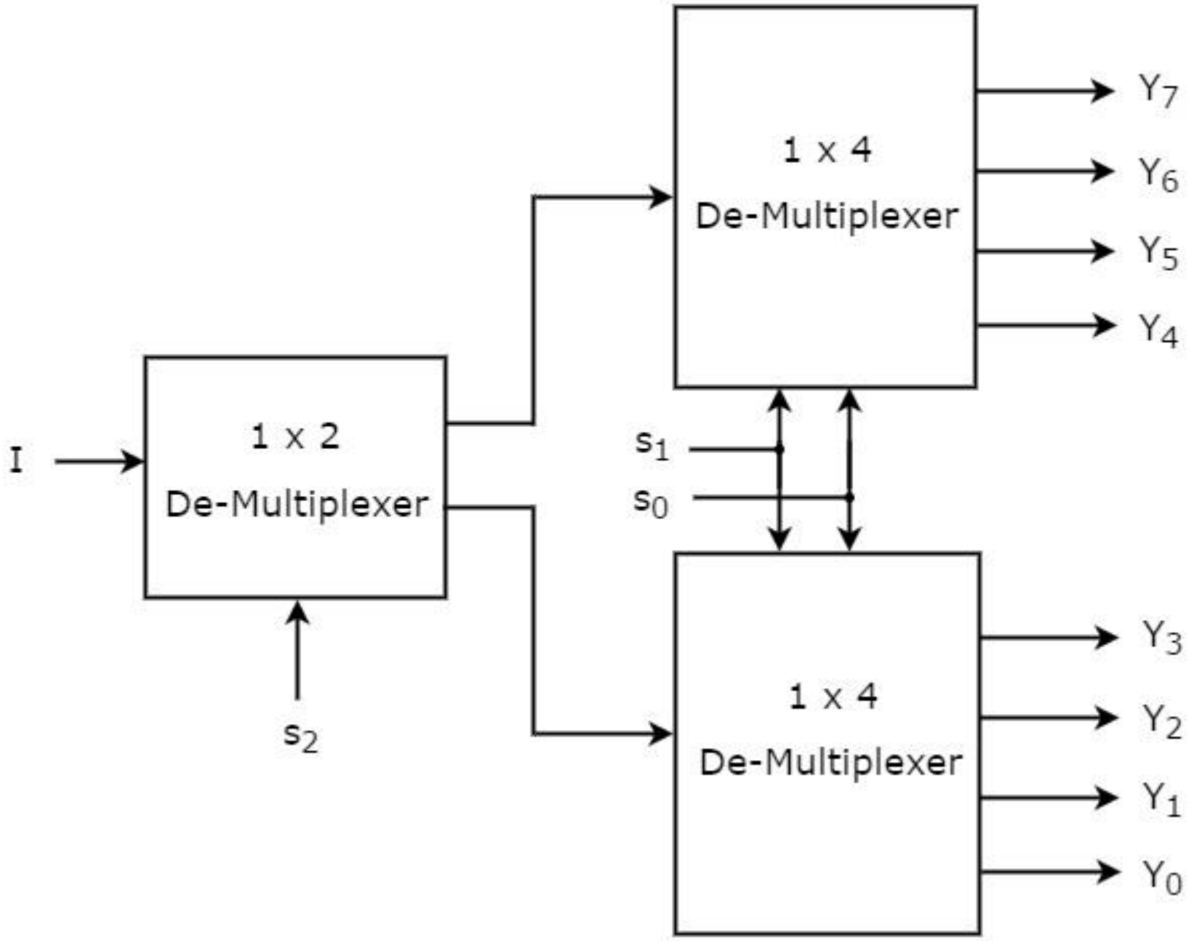
So, we require two **1x4 De-Multiplexers** in second stage in order to get the final eight outputs. Since, the number of inputs in second stage is two, we require **1x2 DeMultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x8 De-Multiplexer.

Let the 1x8 De-Multiplexer has one input I, three selection lines s_2, s_1 & s_0 and outputs Y_7 to Y_0. The **Truth table** of 1x8 De-Multiplexer is shown below.

Selection Inputs			**Outputs**							
s_2	s_1	s_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	**I**
0	0	1	0	0	0	0	0	0	**I**	0
0	1	0	0	0	0	0	0	**I**	0	0
0	1	1	0	0	0	0	**I**	0	0	0

1	0	0	0	0	0	**I**	0	0	0	0
1	0	1	0	0	**I**	0	0	0	0	0
1	1	0	0	**I**	0	0	0	0	0	0
1	1	1	**I**	0	0	0	0	0	0	0

We can implement 1x8 De-Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 1x8 De-Multiplexer is shown in the following figure.



The common **selection lines, s_1 & s_0** are applied to both 1x4 De-Multiplexers. The outputs of upper 1x4 De-Multiplexer are Y_7 to Y_4 and the outputs of lower 1x4 De-Multiplexer are Y_3 to Y_0.

The other **selection line, s_2** is applied to 1x2 De-Multiplexer. If s_2 is zero, then one of the four outputs of lower 1x4 De-Multiplexer will be equal to input, I based on the values of selection lines s_1 & s_0. Similarly, if s_2 is one, then one of the four outputs of upper 1x4 DeMultiplexer will be equal to input, I based on the values of selection lines s_1 & s_0.

1x16 De-Multiplexer

In this section, let us implement 1x16 De-Multiplexer using 1x8 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x8 De-Multiplexer has single input, three selection lines and eight outputs. Whereas, 1x16 De-Multiplexer has single input, four selection lines and sixteen outputs.

So, we require two **1x8 De-Multiplexers** in second stage in order to get the final sixteen outputs. Since, the number of inputs in second stage is two, we require **1x2 DeMultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x16 De-Multiplexer.

Let the 1x16 De-Multiplexer has one input I, four selection lines s_3, s_2, s_1 & s_0 and outputs Y_{15} to Y_0. The **block diagram** of 1x16 De-Multiplexer using lower order Multiplexers is shown in the following figure.

The common **selection lines s_2, s_1 & s_0** are applied to both 1x8 De-Multiplexers. The outputs of upper 1x8 De-Multiplexer are Y_{15} to Y_8 and the outputs of lower 1x8 DeMultiplexer are Y_7 to Y_0.

The other **selection line, s_3** is applied to 1x2 De-Multiplexer. If s_3 is zero, then one of the eight outputs of lower 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines s_2, s_1 & s_0. Similarly, if s3 is one, then one of the 8 outputs of upper 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines s_2, s_1 & s_0.

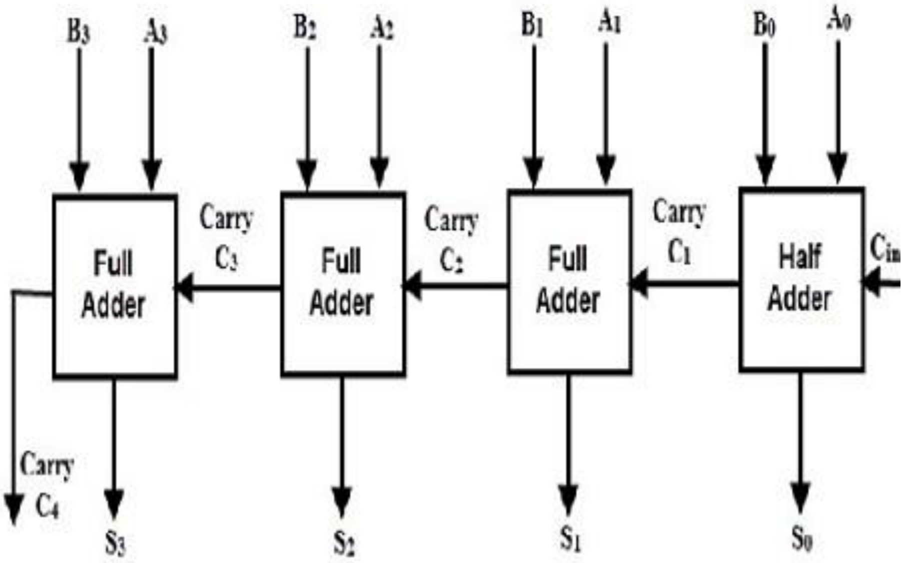
Carry lookahead adder

To perform these operations 'Adder circuits' are implemented using basic logic gates. Adder circuits are evolved as Half-adder, Full-adder, Ripple-carry Adder, and Carry Look-ahead Adder.
Among these Carry Look-ahead Adder is the faster adder circuit. It reduces the propagation delay, which occurs during addition, by using more complex hardware circuitry. It is designed by transforming the ripple-carry Adder circuit such that the carry logic of the adder is changed into two-level logic.

4-Bit Carry Look-ahead Adder

In parallel adders, carry output of each full adder is given as a carry input to the next higher-order state. Hence, these adders it is not possible to produce carry and sum outputs of any state unless a carry input is available for that state.

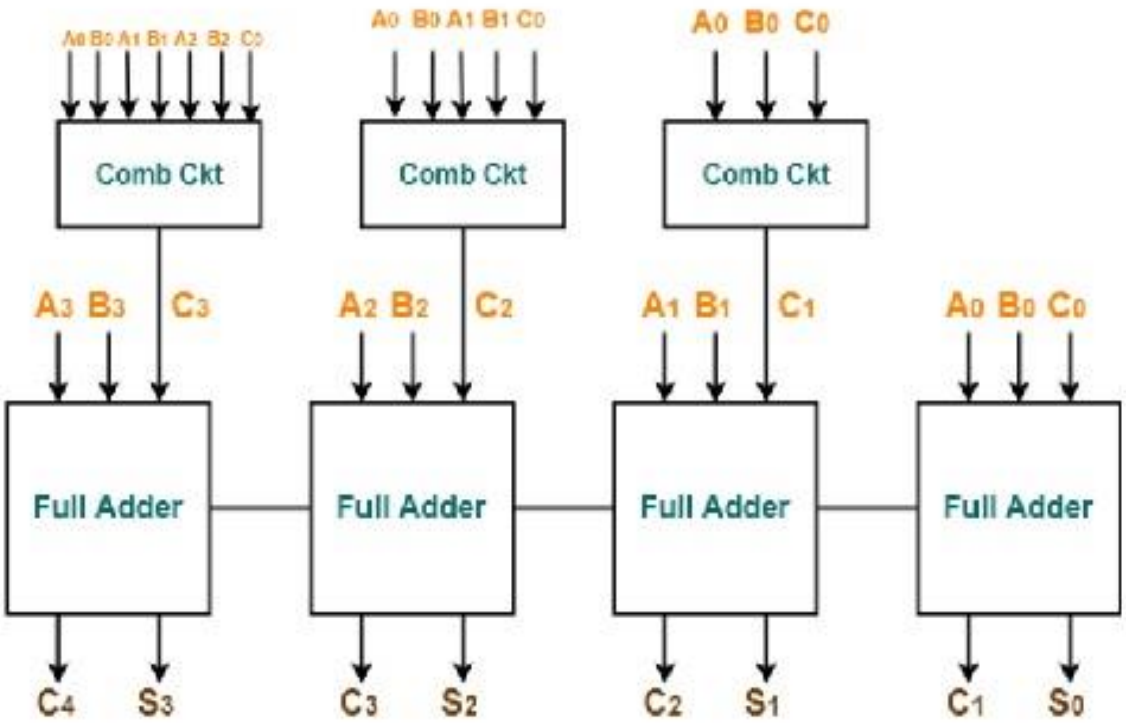
So, for computation to occur, the circuit has to wait until the carry bit propagated to all states. This induces carry propagation delay in the circuit.



Consider the 4-bit ripple carry adder circuit above. Here the sum S3 can be produced as soon as the inputs A3 and B3 are given. But carry C3 cannot be computed until the carry bit C2 is applied whereas C2 depends on C1. Therefore to produce final steady-state results, carry must propagate through all the states. This increases the carry propagation delay of the circuit.

The propagation delay of the adder is calculated as "the propagation delay of each gate times the number of stages in the circuit". For the computation of a large number of bits, more stages have to be added, which makes the delay much worse. Hence, to solve this situation, Carry Look-ahead Adder was introduced.

To understand the functioning of a Carry Look-ahead Adder, a 4-bit Carry Look-ahead Adder is described below.



In this adder, the carry input at any stage of the adder is independent of the carry bits generated at the independent stages. Here the output of any stage is dependent only on the bits which are added in the previous stages and the carry input provided at the beginning stage. Hence, the circuit at any stage does not have to wait for the generation of carry-bit from the previous stage and carry bit can be evaluated at any instant of time.

Truth Table of Carry Look-ahead Adder

For deriving the truth table of this adder, two new terms are introduced – Carry generate and carry propagate. Carry generate Gi =1 whenever there is a carry Ci+1 generated. It depends on Ai and Bi inputs. Gi is 1 when both Ai and Bi are 1. Hence, Gi is calculated as Gi = Ai. Bi.

Carry propagated Pi is associated with the propagation of carry from Ci to Ci+1. It is calculated as Pi = Ai ⊕ Bi. The truth table of this adder can be derived from modifying the truth table of a full adder.

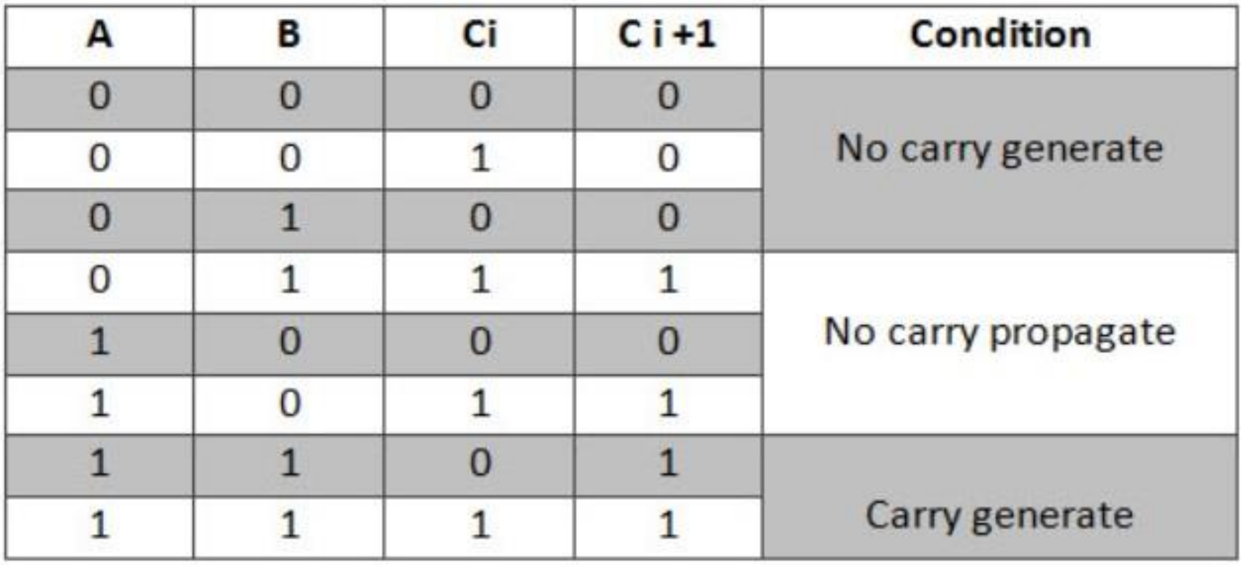
Using the Gi and Pi terms the Sum Si and Carry Ci+1 are given as below –

- Si = Pi ⊕ Gi.
- Ci+1 = Ci.Pi +Gi.

Therefore, the carry bits C1, C2, C3, and C4 can be calculated as

- C1 = C0.P0+G0.
- C2 = C1.P1+G1 = (C0.P0+G0).P1+G1.
- C3 = C2.P2+G2 = (C1.P1+G1).P2+G2.
- C4 = C3.P3+G3 = C0.P0.P1.P2.P3 + P3.P2.P1.G0 + P3.P2.G1 + G2.P3 + G3.

It can be observed from the equations that carry Ci+1 only depends on the carry C0, not on the intermediate carry bits.

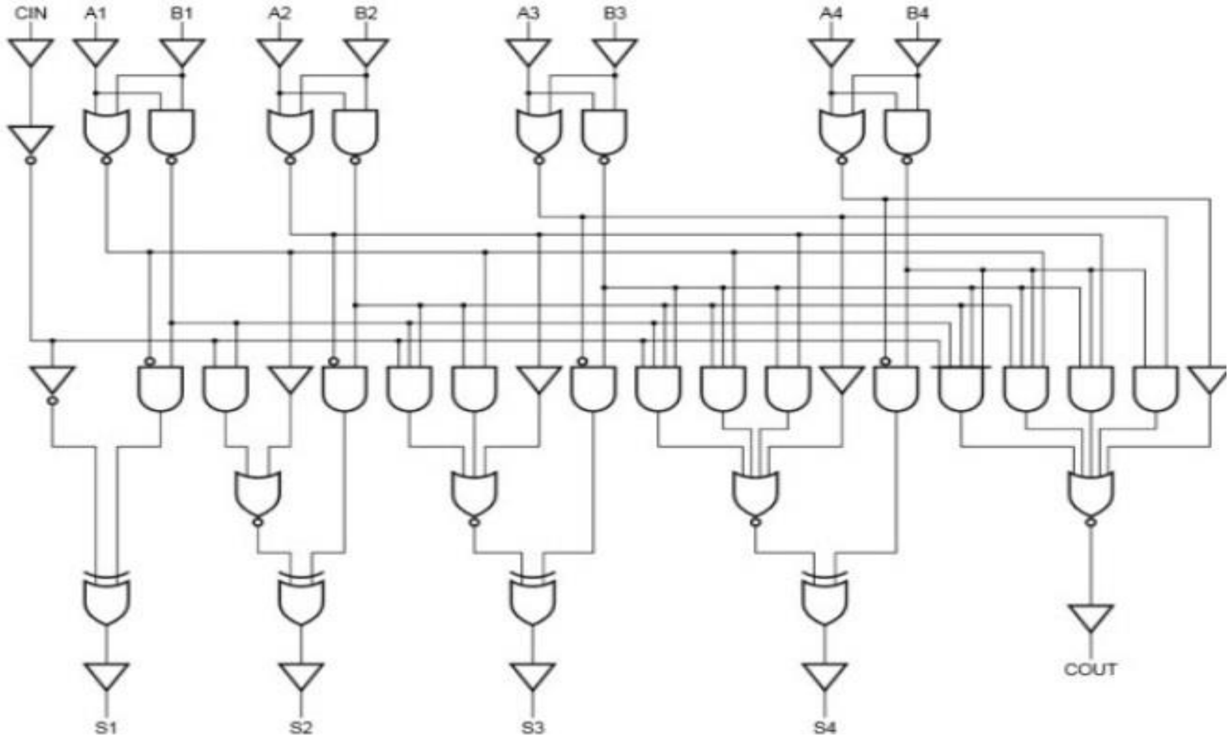


A	B	Ci	C i +1	Condition
0	0	0	0	No carry generate
0	0	1	0	
0	1	0	0	
0	1	1	1	No carry propagate
1	0	0	0	
1	0	1	1	
1	1	0	1	Carry generate
1	1	1	1	

Circuit Diagram

The above equations are implemented using two-level combinational circuits along with AND, OR gates, where gates are assumed to have multiple inputs.

The Carry Look-ahead Adder circuit fro 4-bit is given below.



8-bit and 16-bit Carry Look-ahead Adder circuits can be designed by cascading the 4-bit adder circuit with carry logic.

Advantages of Carry Look-ahead Adder

In this adder, the propagation delay is reduced. The carry output at any stage is dependent only on the initial carry bit of the beginning stage. Using this adder it is possible to calculate the intermediate results. This adder is the fastest adder used for computation.

Applications

High-speed Carry Look-ahead Adders are used as implemented as IC’s. Hence, it is easy to embed the adder in circuits. By combining two or more adders calculations of higher bit boolean functions can be done easily. Here the increase in the number of gates is also moderate when used for higher bits.

For this Adder there is a tradeoff between area and speed. When used for higher bit calculations, it provides high speed but the complexity of the circuit is also increased thereby increasing the area occupied by the circuit. This adder is usually implemented as 4-bit modules which are cascaded together when used for higher calculations. This adder is costlier compared to other adders.

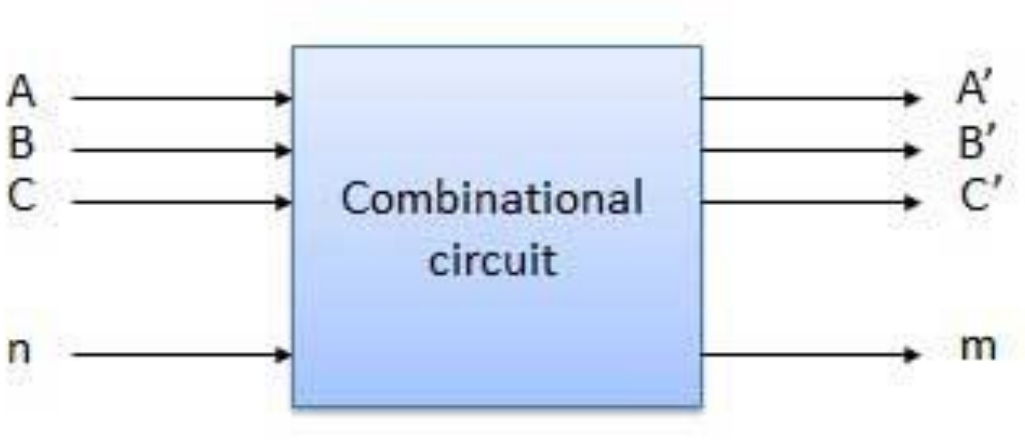
For boolean computation in computers, adders are being used regularly. Charles Babbage implemented a mechanism for anticipating the carry bit in computers, to reduce the delay caused by the ripple carry adders. While designing a system, the speed of computation is the highest deciding factor for a designer. In 1957, Gerald B. Rosenberger patented the modern Binary Carry Look-ahead Adder. Based on the analysis of gate delay and simulation, experiments are being conducted to modify the circuit of this adder to make it even faster. For an n-bit carry look-ahead adder, what is the propagation delay, when given a delay of each gate is 20?

Combinational logic Design

Combinational circuit is a circuit in which we combine the different gates in the circuit, for example encoder, decoder, multiplexer and demultiplexer. Some of the characteristics of combinational circuits are following −

- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
- The combinational circuit do not use any memory. The previous state of input does not have any effect on the present state of the circuit.
- A combinational circuit can have an n number of inputs and m number of outputs.

Block diagram

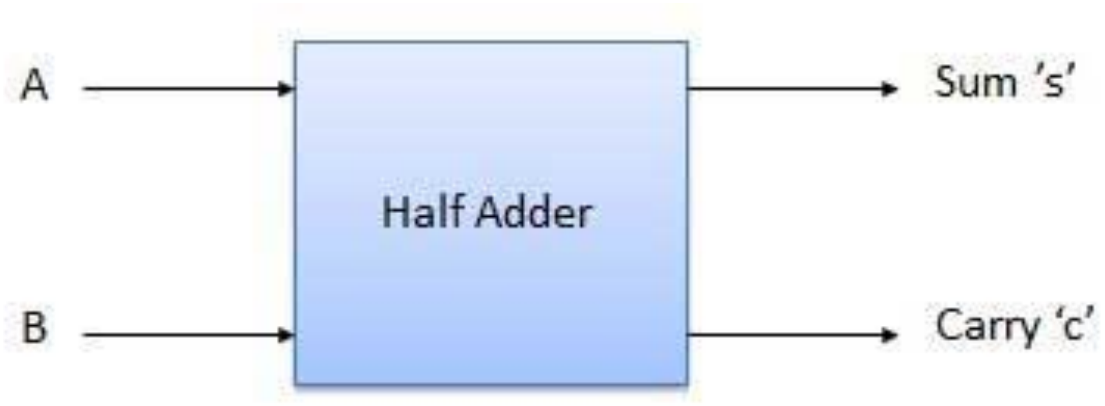


We're going to elaborate few important combinational circuits as follows.

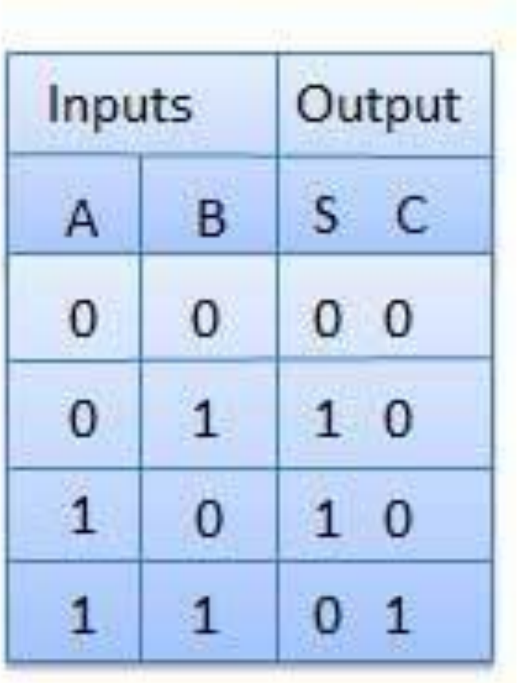
Half Adder

Half adder is a combinational logic circuit with two inputs and two outputs. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two **single** bit numbers. This circuit has two outputs **carry** and **sum**.

Block diagram

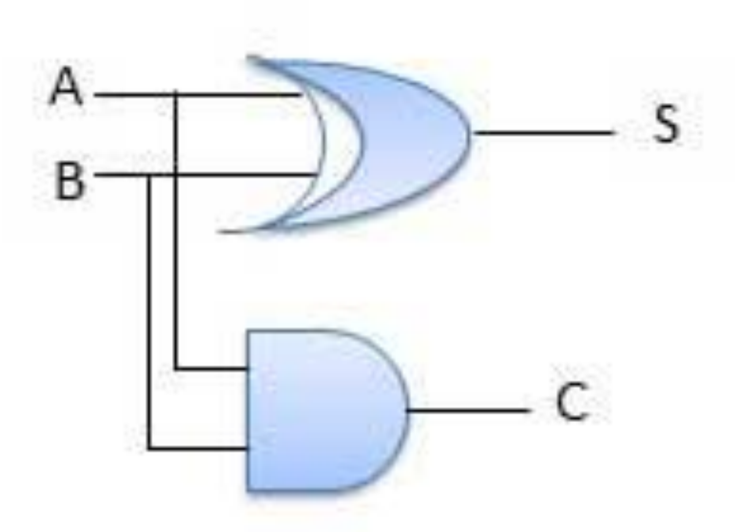


Truth Table



Inputs		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

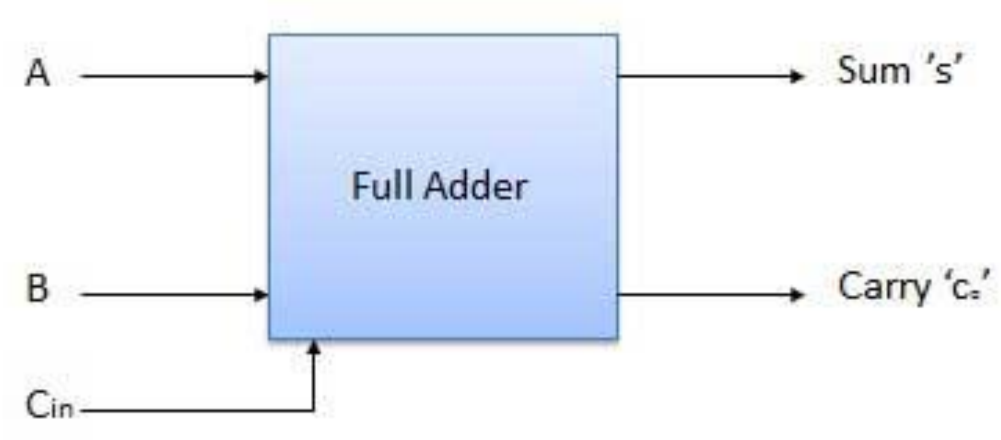
Circuit Diagram



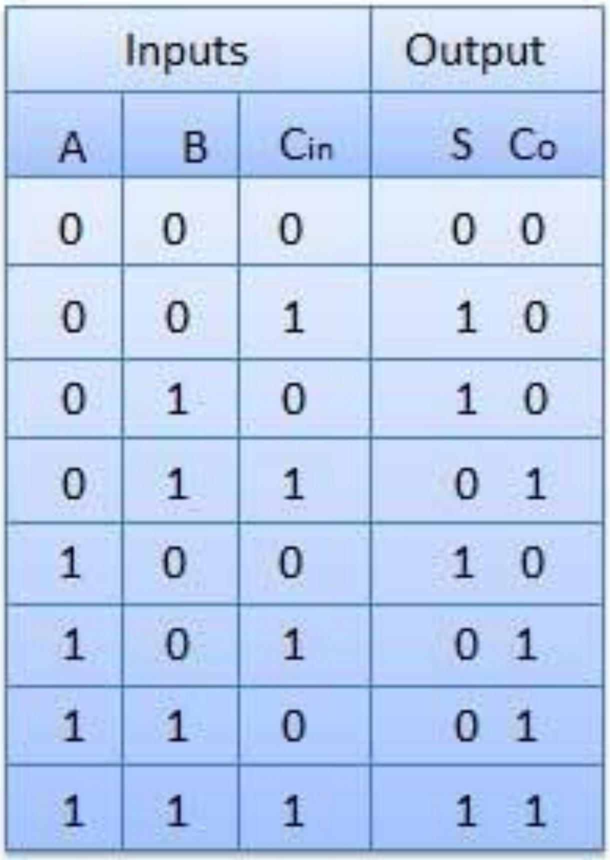
Full Adder

Full adder is developed to overcome the drawback of Half Adder circuit. It can add two one-bit numbers A and B, and carry c. The full adder is a three input and two output combinational circuit.

Block diagram

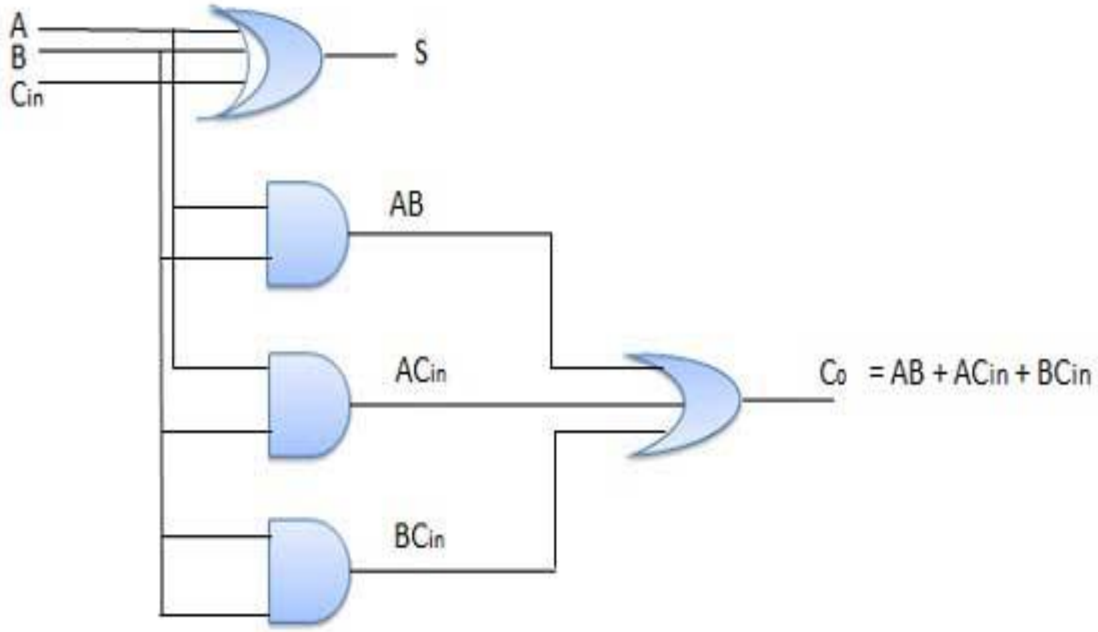


Truth Table



Inputs			Output	
A	B	C_{in}	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Circuit Diagram



N-Bit Parallel Adder

The Full Adder is capable of adding only two single digit binary number along with a carry input. But in practical we need to add binary numbers which are much longer than just one bit. To add two n-bit binary numbers we need to use the n-bit parallel adder. It uses a number of full adders in cascade. The carry output of the previous full adder is connected to carry input of the next full adder.

4 Bit Parallel Adder

In the block diagram, A_0 and B_0 represent the LSB of the four bit words A and B. Hence Full Adder-0 is the lowest stage. Hence its C_{in} has been permanently made 0. The rest of the connections are exactly same as those of n-bit parallel adder is shown in fig. The four bit parallel adder is a very common logic circuit.

Block diagram

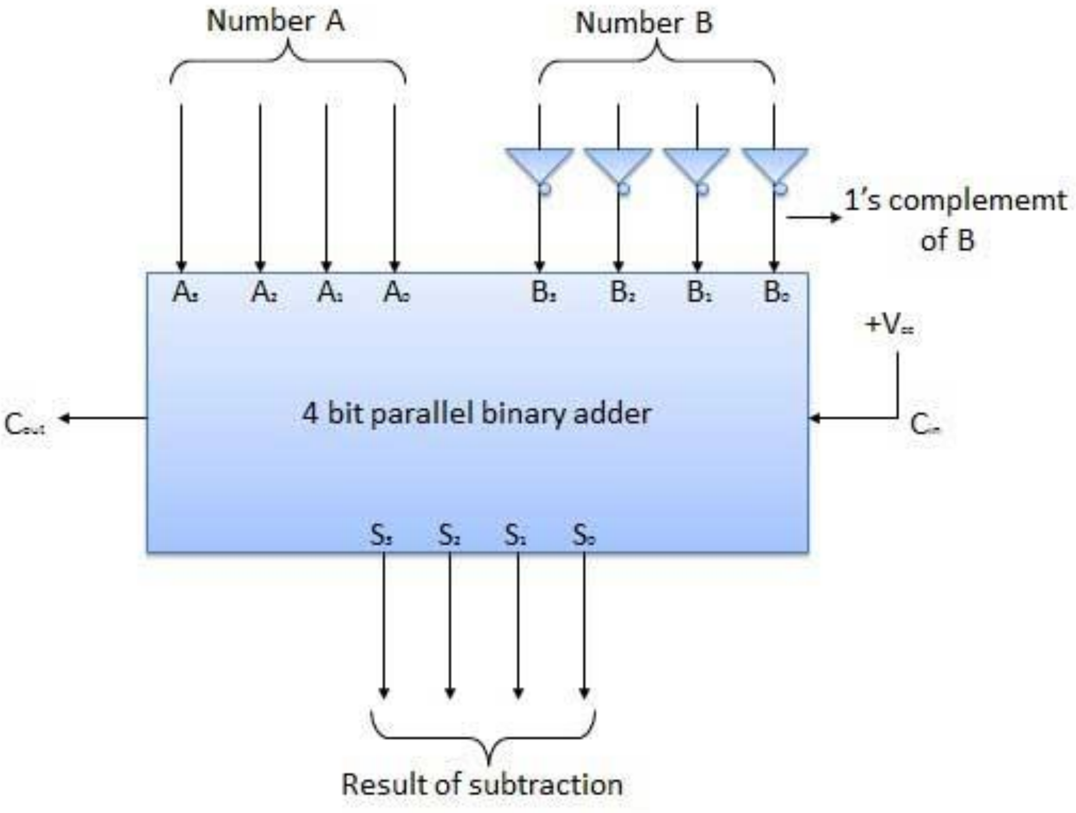
N-Bit Parallel Subtractor

The subtraction can be carried out by taking the 1's or 2's complement of the number to be subtracted. For example we can perform the subtraction (A-B) by adding either 1's or 2's complement of B to A. That means we can use a binary adder to perform the binary subtraction.

4 Bit Parallel Subtractor

The number to be subtracted (B) is first passed through inverters to obtain its 1's complement. The 4-bit adder then adds A and 2's complement of B to produce the subtraction. S_3 S_2 S_1 S_0 represents the result of binary subtraction (A-B) and carry output C_{out} represents the polarity of the result. If A > B then Cout = 0 and the result of binary form (A-B) then C_{out} = 1 and the result is in the 2's complement form.

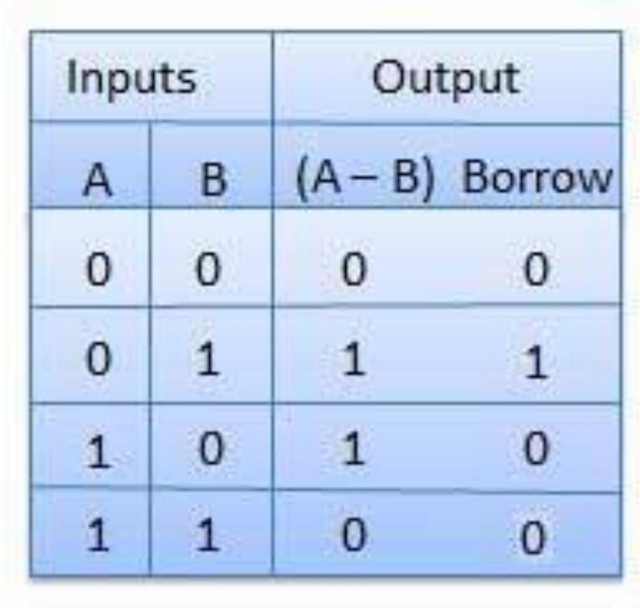
Block diagram



Half Subtractors

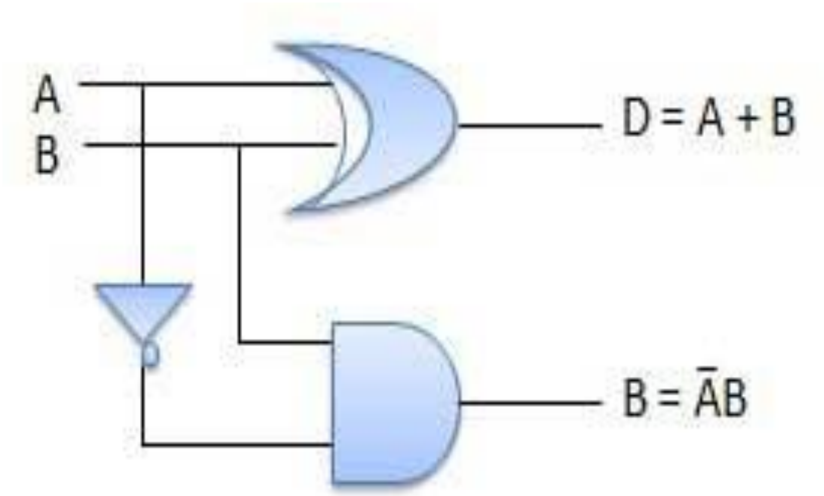
Half subtractor is a combination circuit with two inputs and two outputs (difference and borrow). It produces the difference between the two binary bits at the input and also produces an output (Borrow) to indicate if a 1 has been borrowed. In the subtraction (A-B), A is called as Minuend bit and B is called as Subtrahend bit.

Truth Table



Inputs		Output	
A	B	(A – B)	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

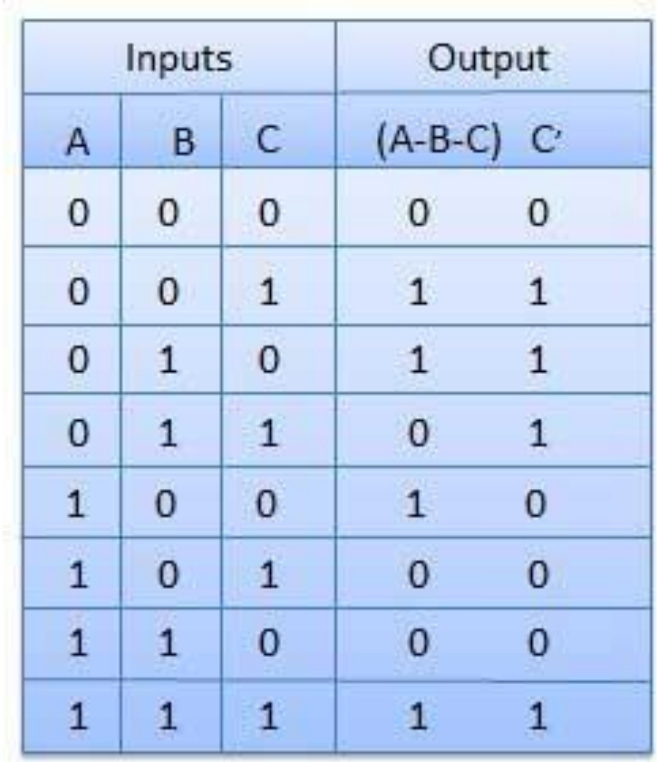
Circuit Diagram



Full Subtractors

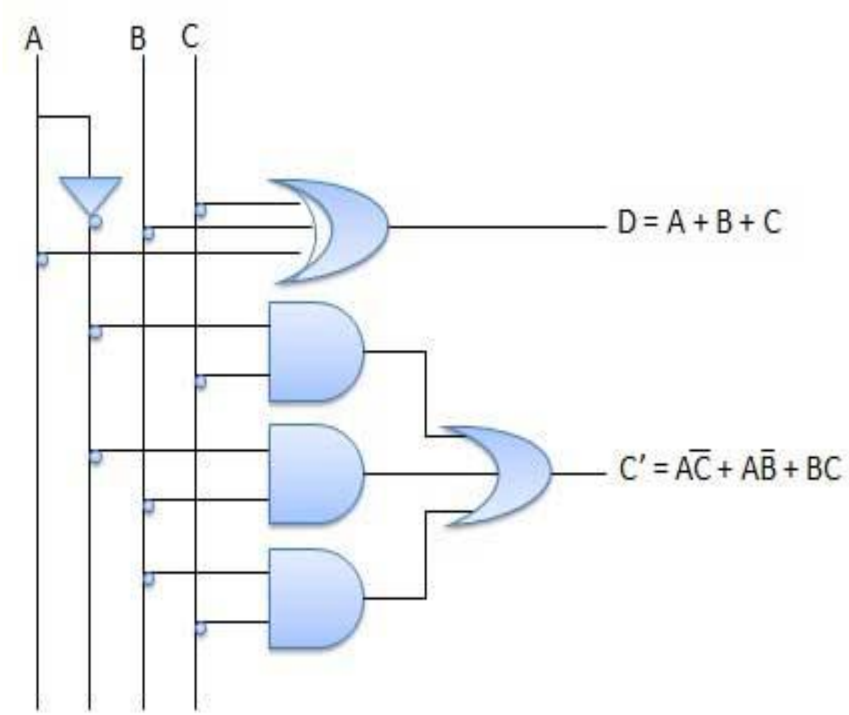
The disadvantage of a half subtractor is overcome by full subtractor. The full subtractor is a combinational circuit with three inputs A,B,C and two output D and C'. A is the 'minuend', B is 'subtrahend', C is the 'borrow' produced by the previous stage, D is the difference output and C' is the borrow output.

Truth Table



Inputs			Output	
A	B	C	(A-B-C)	C′
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

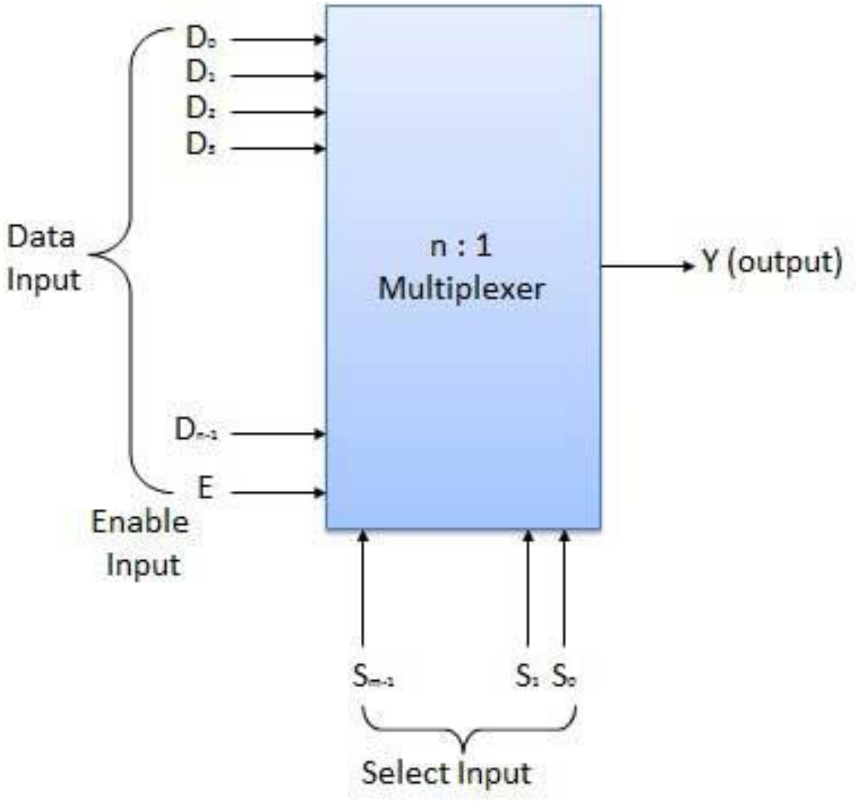
Circuit Diagram



Multiplexers

Multiplexer is a special type of combinational circuit. There are n-data inputs, one output and m select inputs with 2m = n. It is a digital circuit which selects one of the n data inputs and routes it to the output. The selection of one of the n inputs is done by the selected inputs. Depending on the digital code applied at the selected inputs, one out of n data sources is selected and transmitted to the single output Y. E is called the strobe or enable input which is useful for the cascading. It is generally an active low terminal that means it will perform the required operation when it is low.

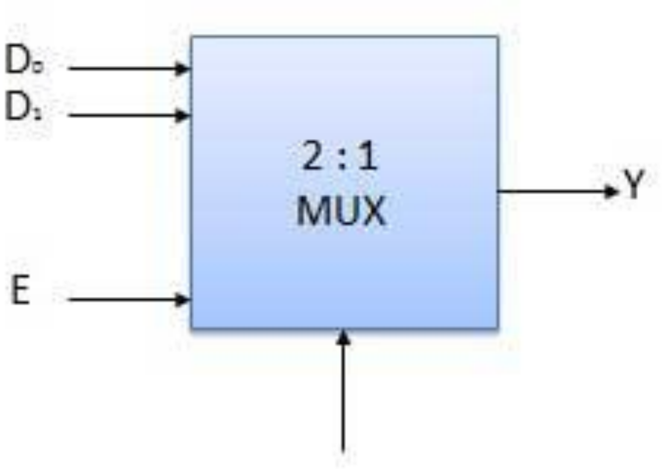
Block diagram



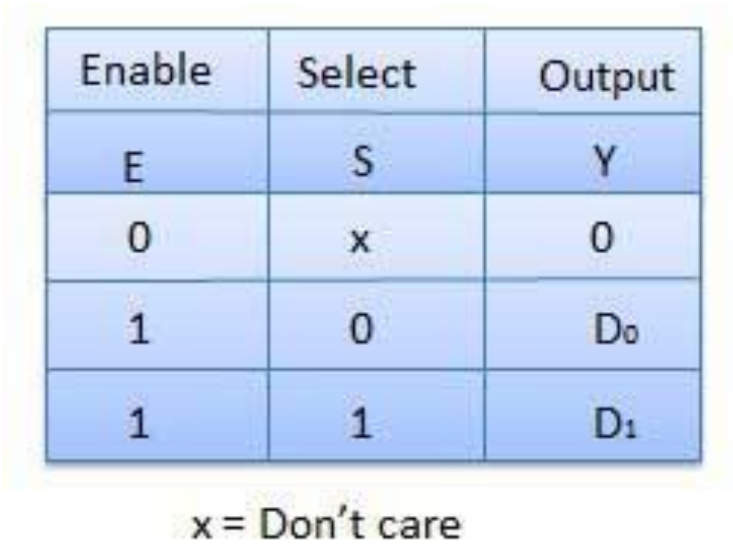
Multiplexers come in multiple variations

- 2 : 1 multiplexer
- 4 : 1 multiplexer
- 16 : 1 multiplexer
- 32 : 1 multiplexer

Block Diagram



Truth Table



Enable	Select	Output
E	S	Y
0	x	0
1	0	D_0
1	1	D_1

x = Don’t care

Demultiplexers

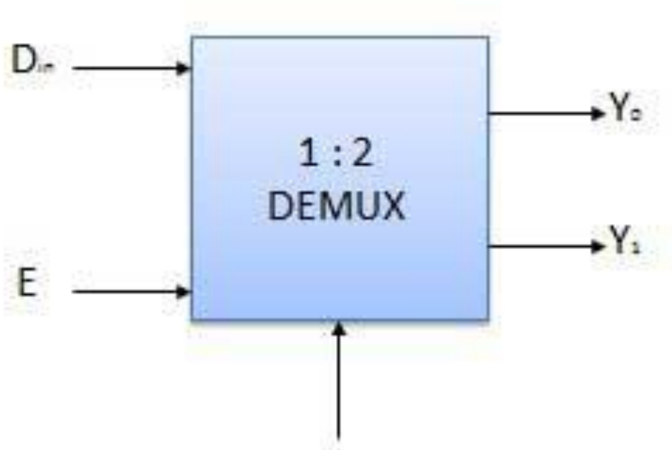
A demultiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs. It has only one input, n outputs, m select input. At a time only one output line is selected by the select lines and the input is transmitted to the selected output line. A de-multiplexer is equivalent to a single pole multiple way switch as shown in fig.

Demultiplexers comes in multiple variations.

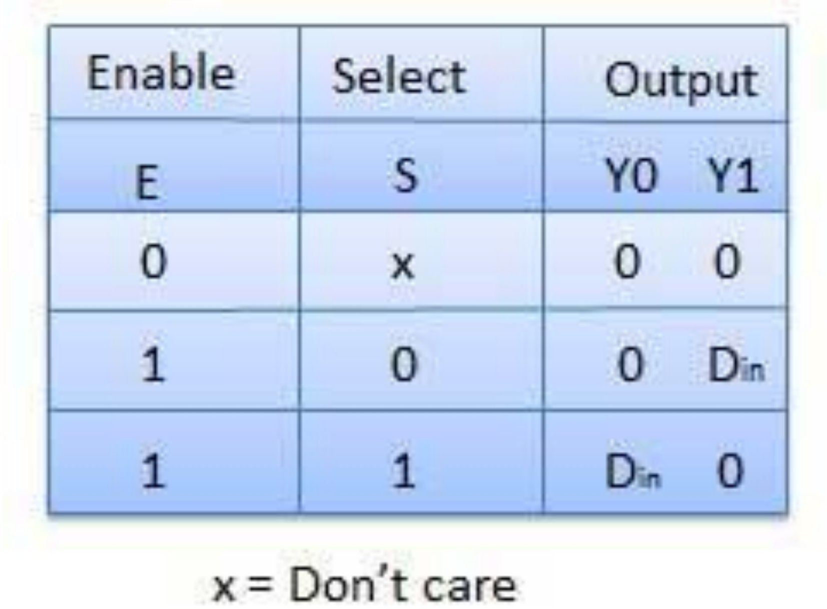
- 1 : 2 demultiplexer
- 1 : 4 demultiplexer

- 1 : 16 demultiplexer
- 1 : 32 demultiplexer

Block diagram



Truth Table



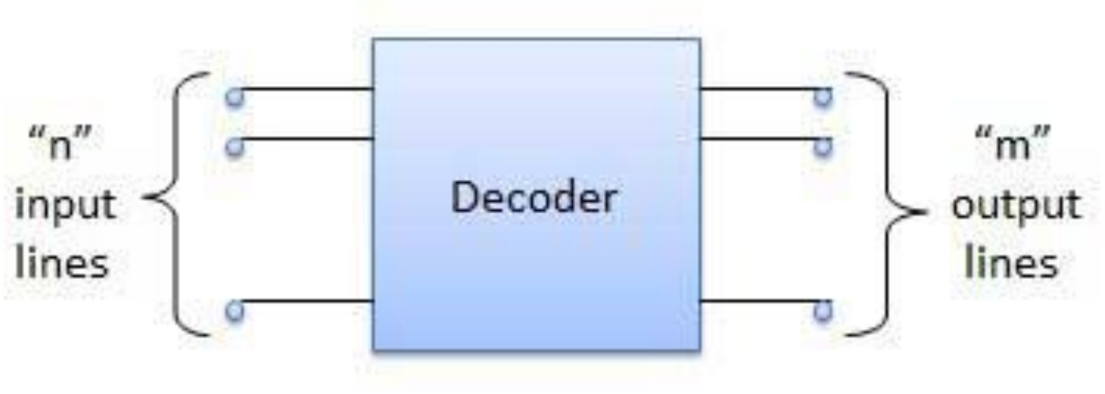
Enable	Select	Output	
E	S	Y0	Y1
0	x	0	0
1	0	0	Din
1	1	Din	0

x = Don't care

Decoder

A decoder is a combinational circuit. It has n input and to a maximum m = 2n outputs. Decoder is identical to a demultiplexer without any data input. It performs operations which are exactly opposite to those of an encoder.

Block diagram



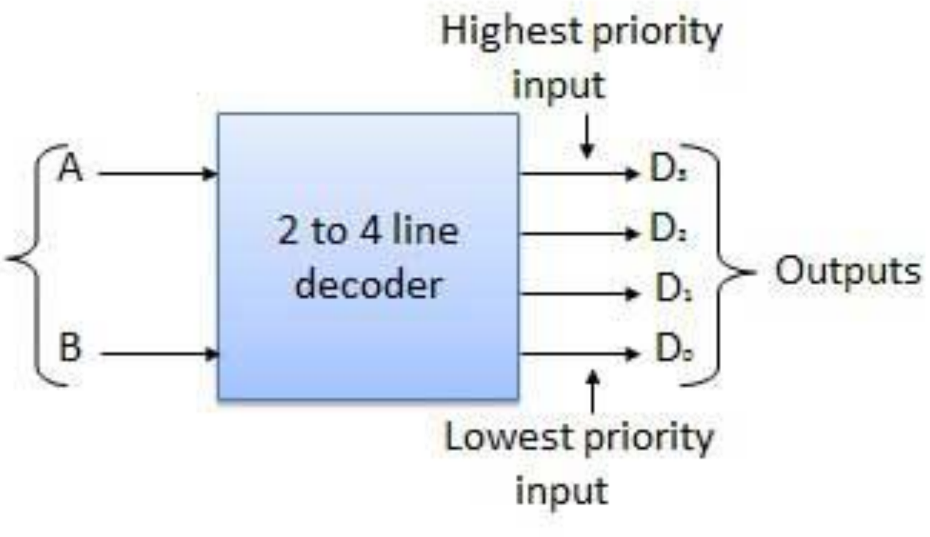
Examples of Decoders are following.

- Code converters
- BCD to seven segment decoders
- Nixie tube decoders
- Relay actuator

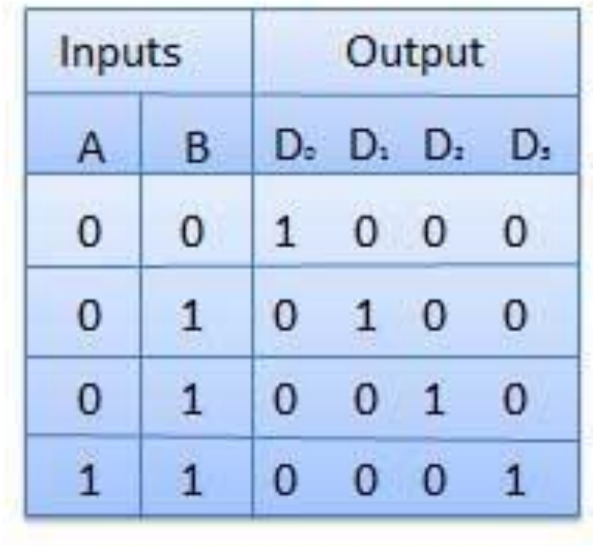
2 to 4 Line Decoder

The block diagram of 2 to 4 line decoder is shown in the fig. A and B are the two inputs where D through D are the four outputs. Truth table explains the operations of a decoder. It shows that each output is 1 for only a specific combination of inputs.

Block diagram

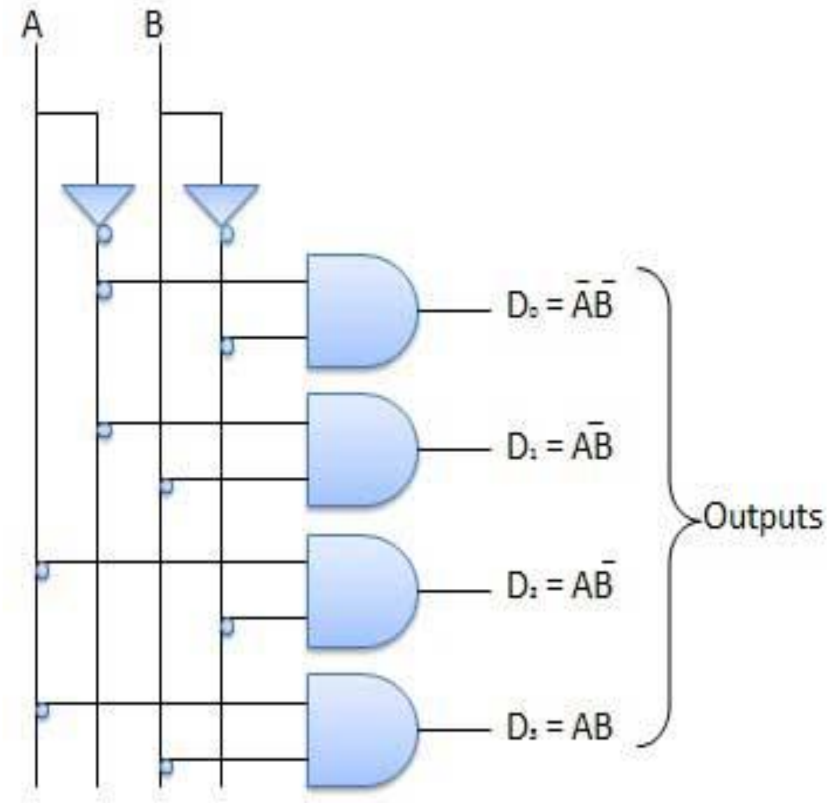


Truth Table



Inputs		Output			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
0	1	0	0	1	0
1	1	0	0	0	1

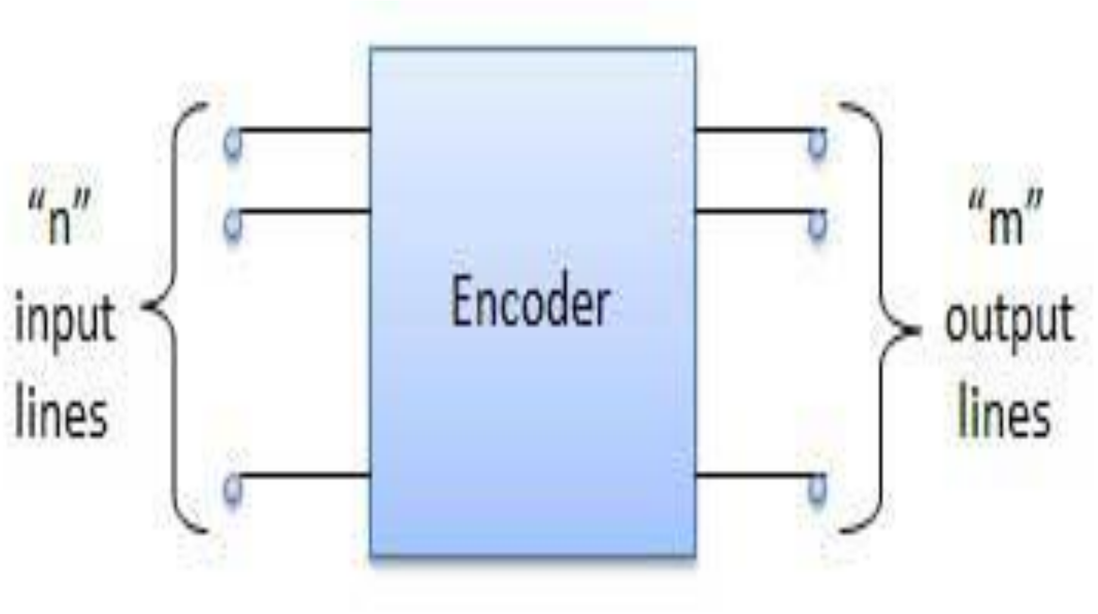
Logic Circuit



Encoder

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.

Block diagram



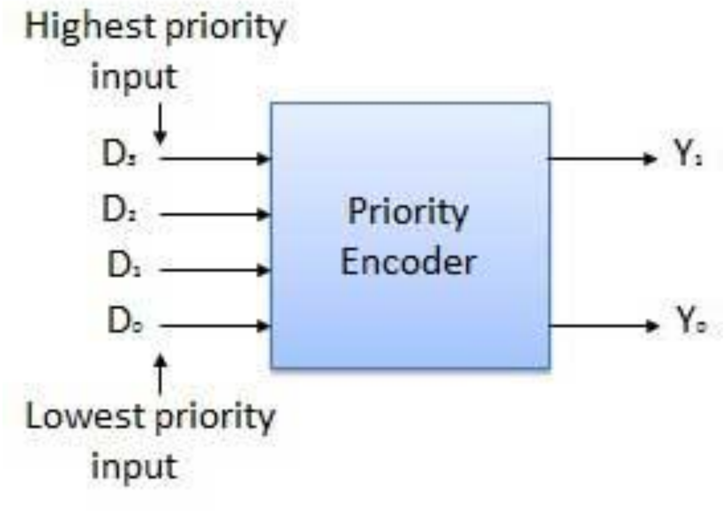
Examples of Encoders are following.

- Priority encoders
- Decimal to BCD encoder
- Octal to binary encoder
- Hexadecimal to binary encoder

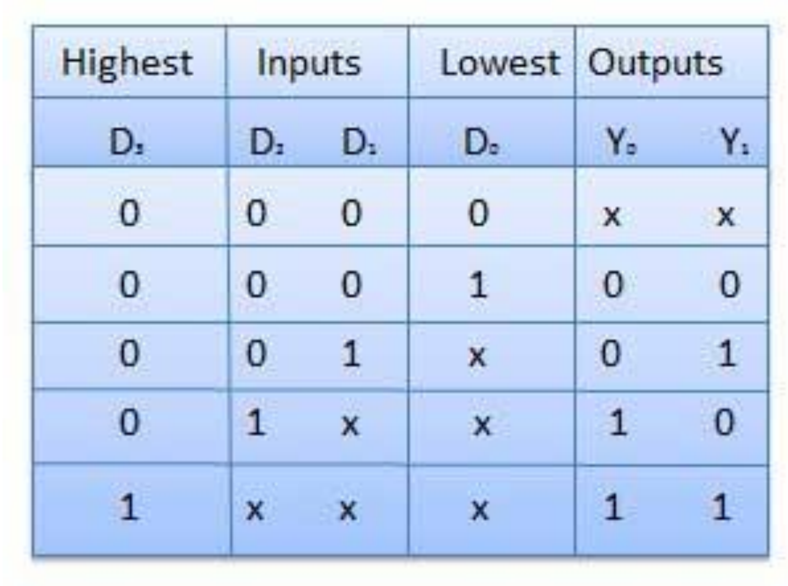
Priority Encoder

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.

Block diagram

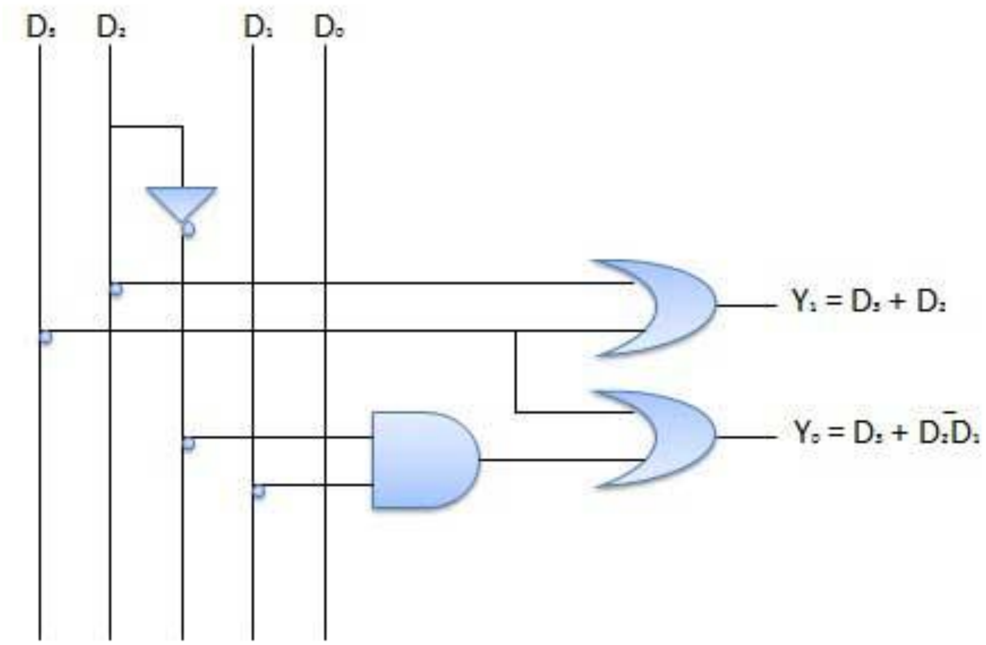


Truth Table



Highest	Inputs		Lowest	Outputs	
D_3	D_2	D_1	D_0	Y_0	Y_1
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

Logic Circuit



Flip-Flops

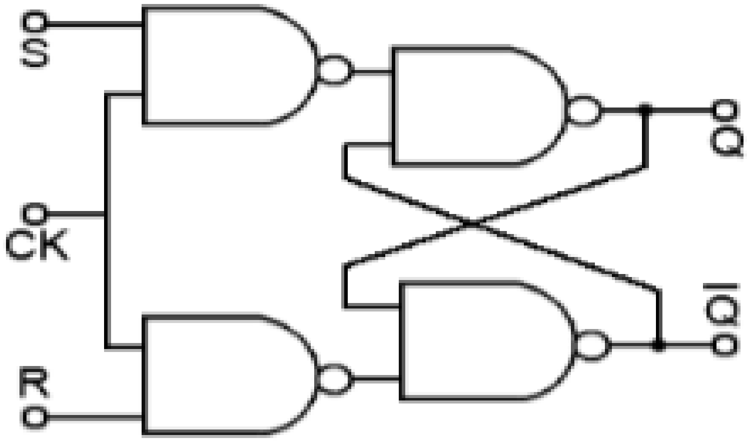
Flip-flop is a circuit that maintains a state until directed by input to change the state. A basic flip-flop can be constructed using four-NAND or four-NOR gates.

Types of flip-flops:

1. RS Flip Flop
2. JK Flip Flop
3. D Flip Flop
4. T Flip Flop

Logic diagrams and truth tables of the different types of flip-flops are as follows:

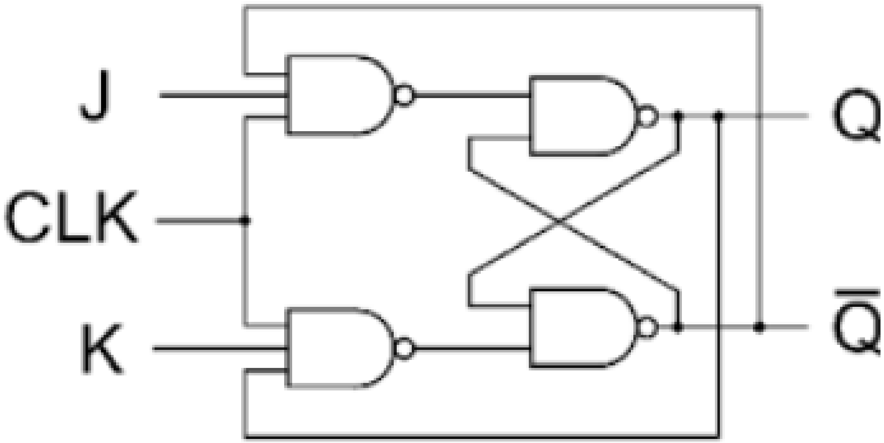
S-R Flip Flop :



TRUTH TABLE

S	R	Q_N	Q_{N+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	-
1	1	1	-

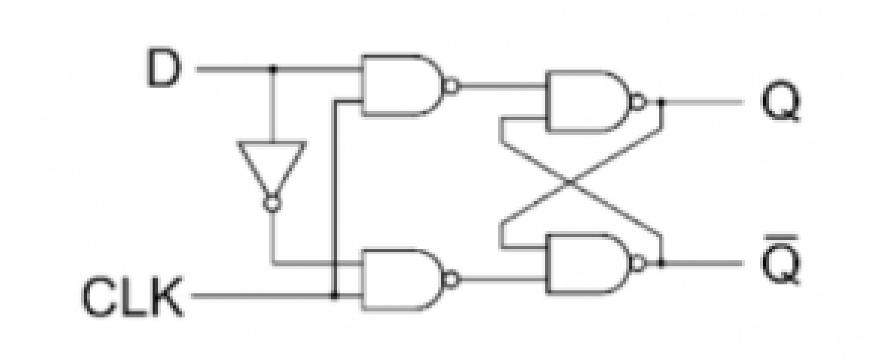
J-K Flip Flop:

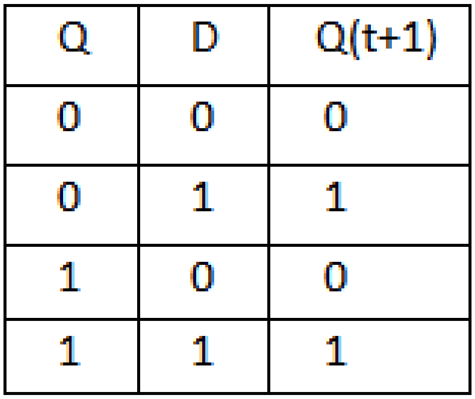


TRUTH TABLE

J	K	Q_N	Q_{N+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

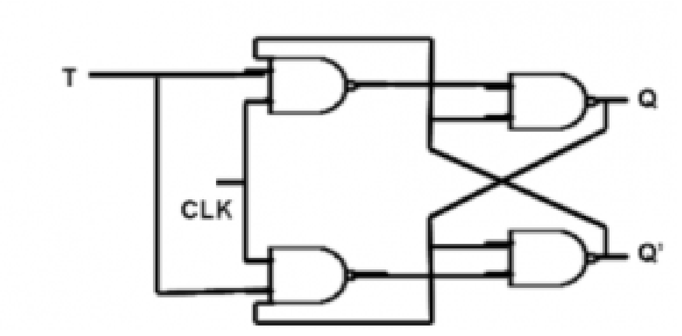
D Flip Flop :

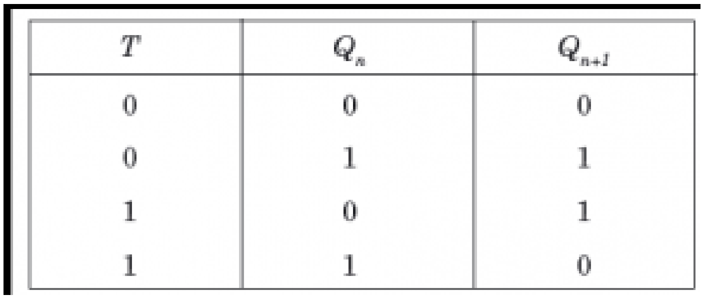




Q	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

T Flip Flop :

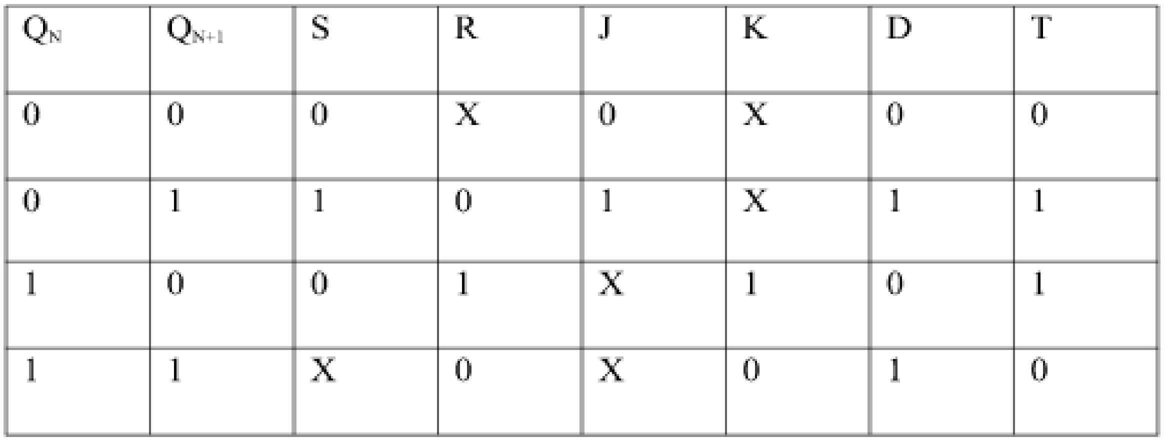




T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Conversion for FlipFlops :-

EXCITATION TABLE:



Q_N	Q_{N+1}	S	R	J	K	D	T
0	0	0	X	0	X	0	0
0	1	1	0	1	X	1	1
1	0	0	1	X	1	0	1
1	1	X	0	X	0	1	0

Steps To Convert from One FlipFlop to Other :

Let there be required flipflop to be constructed using sub-flipflop:

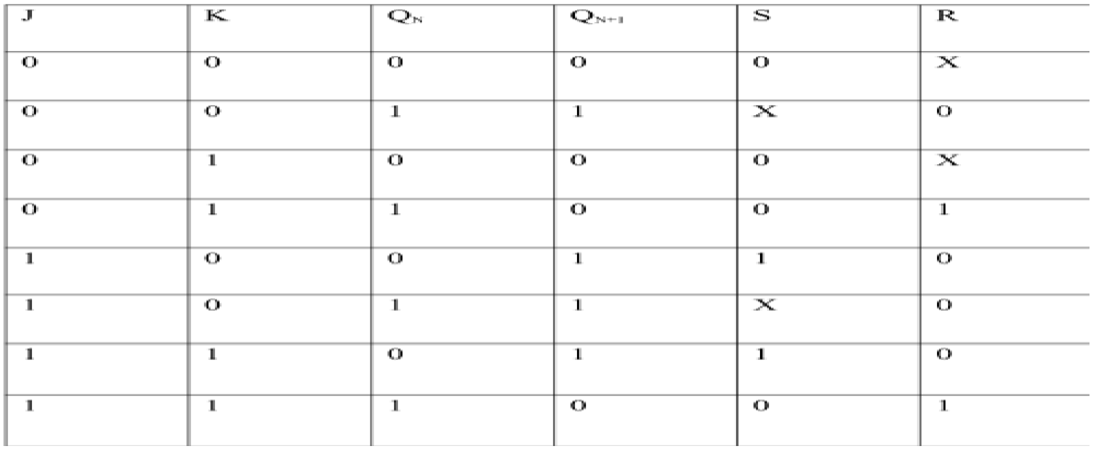
1. Draw the truth table of required flipflop.

2. Write the corresponding outputs of sub-flipflop to be used from the excitation table.

3. Draw K-Maps using required flipflop inputs and obtain excitation functions for sub-flipflop inputs.

4. Construct logic diagram according to the functions obtained.

i) Convert SR To JK FlipFlop



J	K	Q_N	Q_{N+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Excitation Functions:

KQ_N

J

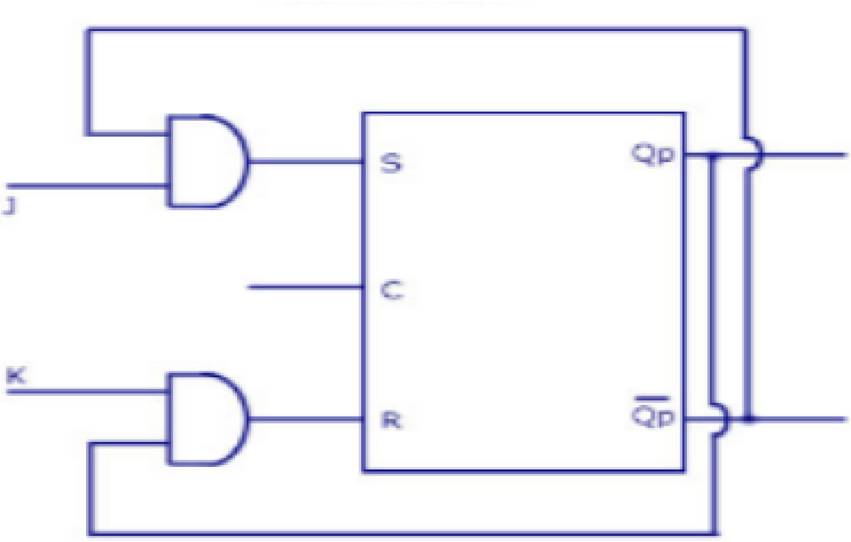
$S = JQ_N'$

0	X	0	0
1	X	0	1

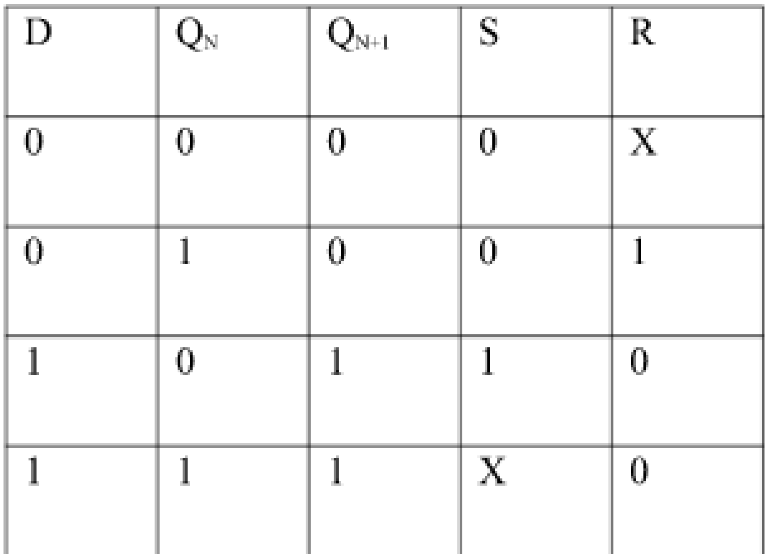
$R = KQ_N$

KQ_N

X	0	1	X
0	0	1	0



ii) Convert SR To D FlipFlop:



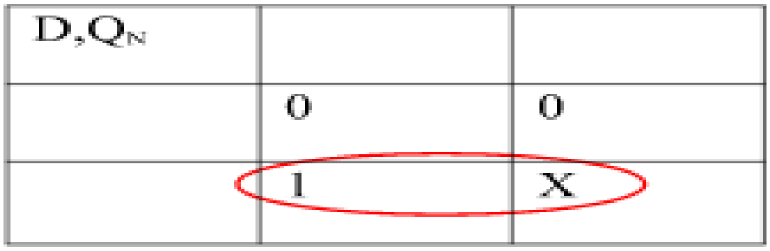
D	Q_N	Q_{N+1}	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

Excitation Functions:

S = D

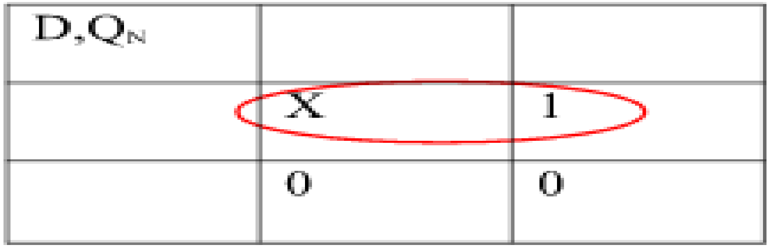
R = D'

S:

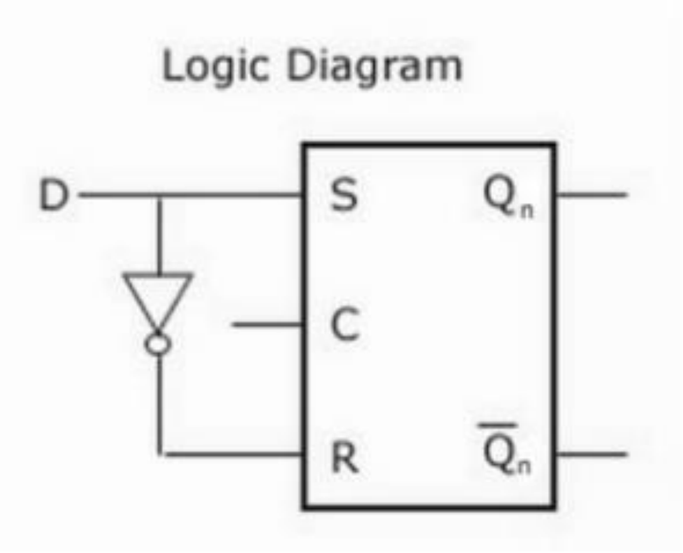


D,Q_N		
	0	0
	1	X

R:



D,Q_N		
	X	1
	0	0



Applications of Flip-Flops

These are the various types of flip-flops being used in digital electronic circuits and the applications of Flip-flops are as specified below.

- Counters
- Frequency Dividers
- Shift Registers
- Storage Registers
- Bounce elimination switch
- Data storage
- Data transfer
- Latch
- Registers
- Memory

Registers

Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU. The registers used by the CPU are often termed as Processor registers.

A processor register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).

The computer needs processor registers for manipulating data and a register for holding a memory address. The register holding the memory location is used to calculate the address of the next instruction after the execution of the current instruction is completed.

Following is the list of some of the most common registers used in a basic computer:

Register	Symbol	Number of bits	Function
Data register	DR	16	Holds memory operand
Address register	AR	12	Holds address for the memory
Accumulator	AC	16	Processor register
Instruction register	IR	16	Holds instruction code
Program counter	PC	12	Holds address of the instruction
Temporary register	TR	16	Holds temporary data
Input register	INPR	8	Carries input character

Output register	OUTR	8	Carries output character

The following image shows the register and memory configuration for a basic computer.

- The Memory unit has a capacity of 4096 words, and each word contains 16 bits.
- The Data Register (DR) contains 16 bits which hold the operand read from the memory location.
- The Memory Address Register (MAR) contains 12 bits which hold the address for the memory location.
- The Program Counter (PC) also contains 12 bits which hold the address of the next instruction to be read from memory after the current instruction is executed.
- The Accumulator (AC) register is a general purpose processing register.
- The instruction read from memory is placed in the Instruction register (IR).
- The Temporary Register (TR) is used for holding the temporary data during the processing.
- The Input Registers (IR) holds the input characters given by the user.
- The Output Registers (OR) holds the output after processing the input data.

Counters (synchronous & asynchronous)

Counters are of two types depending upon clock pulse applied. These counters are: Asynchronous counter, and Synchronous counter.

In Asynchronous Counter is also known as Ripple Counter, different flip flops are triggered with different clock, not simultaneously. While in Synchronous Counter, all flip flops are triggered with same clock simultaneously and Synchronous Counter is faster than asynchronous counter in operation.

Let’s see the difference between these two counters:

S.NO	SYNCHRONOUS COUNTER	ASYNCHRONOUS COUNTER
1.	In synchronous counter, all flip flops	In asynchronous counter, different

	are triggered with same clock simultaneously.	flip flops are triggered with different clock, not simultaneously.
2.	Synchronous Counter is faster than asynchronous counter in operation.	Asynchronous Counter is slower than synchronous counter in operation.
3.	Synchronous Counter does not produce any decoding errors.	Asynchronous Counter produces decoding error.
4.	Synchronous Counter is also called Parallel Counter.	Asynchronous Counter is also called Serial Counter.
5.	Synchronous Counter designing as well implementation are complex due to increasing the number of states.	Asynchronous Counter designing as well as implementation is very easy.
6.	Synchronous Counter will operate in any desired count sequence.	Asynchronous Counter will operate only in fixed count sequence (UP/DOWN).
7.	Synchronous Counter examples are: Ring counter, Johnson counter.	Asynchronous Counter examples are: Ripple UP counter, Ripple DOWN counter.

8.	In synchronous counter, propagation delay is less.	In asynchronous counter, there is high propagation delay.

ALU

Arithmetic Logic Unit (ALU): A sub unit within a computer’s central processing unit. ALU full form is **Arithmetic Logic Unit** , takes the data from Memory registers; **ALU** contains the logical circuit to perform mathematical operations like subtraction, addition, multiplication, division, logical operations and logical shifts on the values held in the processors registers or its accumulator.

It is the size of the word that the **ALU** can handle which, more than any other measure, determines the word-size of a processor: that is, a 32-bit processor is one with a 32-bit ALU.

After processing the instructions the result will store in Accumulator. Control unit generates control signals to ALU to perform specific operations. The accumulator is used as by default register for storing data. It is 16-bit register.

The simplest sort of ALU performs only addition, Boolean logic (including the NOT or complement operation) and shifts a word one bit to the right or left, all other arithmetic operations being synthesized from sequences of these primitive operations. For example, subtraction is performed as complement-add multiplication by a power of two by shifting, division by repeated subtraction. However, there is an increasing tendency in modern processors to implement extra arithmetic functions in hardware, such as dedicated multiplier or divider units.

The ALU might once have been considered the very core of the computer in the sense that it alone actually performed calculations. However, in modern SUPER SCALAR processor architectures this is no longer true, as there are typically several different ALUs in each of several separate integer and floating-point units. An ALU may be required to perform not only those calculations required by a user program but also many internal calculations required by the processor itself, for example to derive addresses for instructions that employ different ADDRESSING MODES, say by adding an offset to a base address. Once again, however, in modern architectures there is a tendency to distribute this work into a separate load/store unit.

The three fundamental attributes of an ALU are its operands and results, functional organization, and algorithms.

Operands and Results

The operands and results of the ALU are machine words of two kinds: arithmetic words, which represent numerical values in digital form, and logic words, which represent arbitrary sets of digitally encoded symbols. Arithmetic words consist of digit vectors (strings of digits).

Operator: Operator is arithmetic or logical operation that is performed on the operand given in instructions.

Flag: ALU uses many types of the flag during processing instructions. All these bits are stored in status or flag registers.

Functional Organization of an ALU

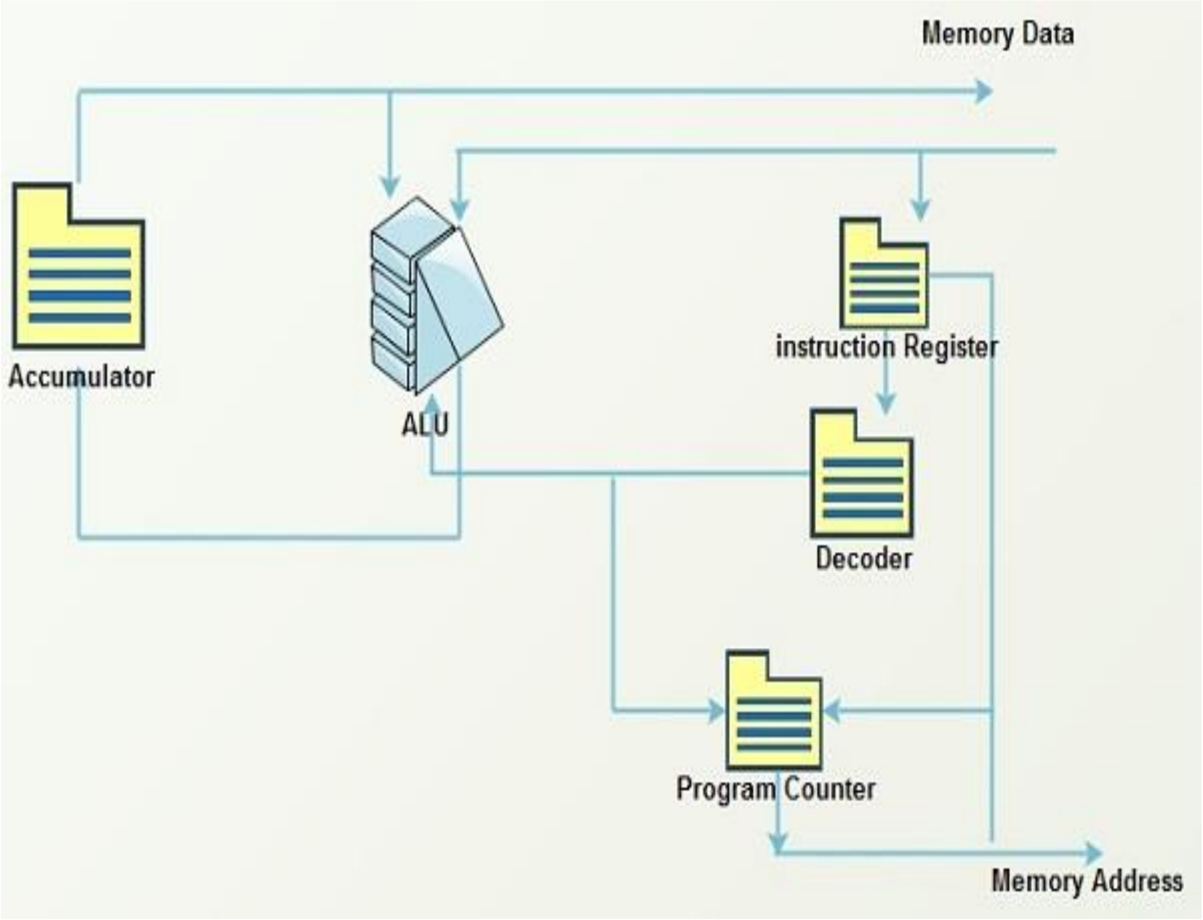
A typical ALU consists of three types of functional parts: storage registers, operations logic, and sequencing logic.

Arithmetic Logical Unit (ALU) Architecture

ALU is formed through the combinational circuit. The combinational circuit used logical gates like AND, OR, NOT, XOR for their construction. The combinational circuit does not have any memory element to store a previous data bit. Adders are the main part of the arithmetic logic unit to perform addition, subtraction by 2’s complement.

Control unit generates the selection signals for selecting the function performed by ALU.

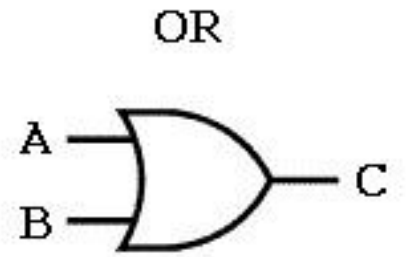
Registers : Registers are a very important component in ALU to store instruction, intermediate data, output, and input.

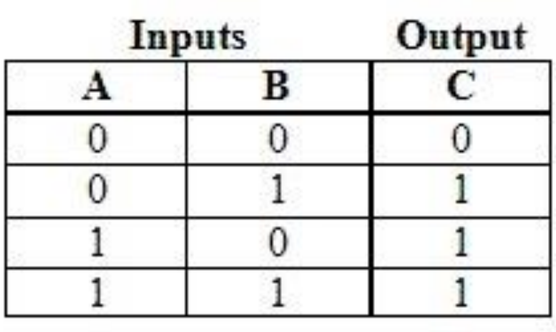


Logic Gates

Logic gates are building a block of ALU. Logic gates are constructed from diode, resistors or transistors. These gates are used in Integrated circuit represent binary input as ‘ON’ and ‘OFF’ state. Binary number 0 is represented by ‘OFF’ and Binary Number ’1’ is represented by ‘ON’ state in an integrated circuit.

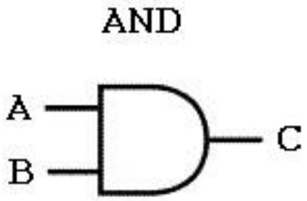
OR gate : OR gate can take two or more inputs. The output of OR gate is always 1 if any of the inputs is 1 and 0 if all the inputs are false. OR gate performs an addition operation on all operand given in instructions. It can be expressed as X=A+B or X=A+B+C.

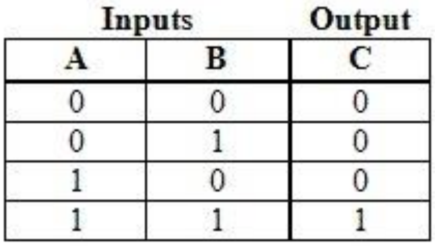




Inputs		Output
A	**B**	**C**
0	0	0
0	1	1
1	0	1
1	1	1

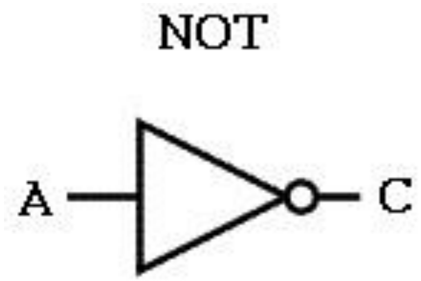
AND gate : AND gate takes two or more inputs. The output of AND gate is 1 if all inputs are 1. AND gate gives 0 results if any one of input in given data is 0. AND gate performs multiplication option on all inputs operands. It is represented by ‘.’ symbol. We can write it as- X=A.B or X=A.B.C.

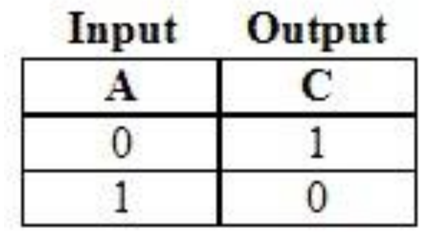




Inputs		Output
A	**B**	**C**
0	0	0
0	1	0
1	0	0
1	1	1

NOT gate : Not gate is used to reverse the result of gates or reverse Boolean state from 0 to 1 and 1 to 0.Not gate is also used with ‘AND’ and ‘OR’ gate. While using with AND or ‘OR’ gate, NOT gate is representing an as small circle in front of both gates. After using NOT gate, AND gates convert into NAND or ‘OR’ gate convert into NOR.





Input	Output
A	C
0	1
1	0

Registers: Registers provide fast memory access as a comparison to cache, RAM, hard disk. They are built on CPU. Register are small in size. Processing Intermediate data stored in registers.A number of registers used for specific purpose. ALU used four general purpose register. All these four registers are 16-bit register is divided into registers. 16-bit register implies that register can store maximum 16 bit of data.

Accumulator : Accumulator is 16 bit by default and general purpose register. By default means that any operand in instruction does not specify a particular register for holding the operand. That time operand will automatically store in AC. AC is used as two separate registers of 7 bit AL and AH. AC located inside the ALU. Intermediate data and result after execution will store in AC.AC used MBR to deal with memory.

Program Counter: PC stands for program counter. It is 16-bit register. It counts the number of instruction left for execution. It acts as a pointer for instructions and also known as Instruction pointer register. PC holds the address of next instruction to be executed. When an instruction is fetched from the register. Register get automatically incremented by one and point to the address of next instruction.

Flag register : it is also known as a Status register or Program Status register. Flag register holds the Boolean value of status word used by the process.

Auxiliary Flag : if two numbers are to be added such that if in the beginning of higher bit there is a carry. This is known as auxiliary bit.

Carry bit : Carry bit is indicate the most significant borrow or carry bit by subtracting a greater number than a smaller number or adding two numbers.

Sign Bit : Sign bit is a most significant bit in 2’s complement to show that result is negative or positive. It is also known as negative bit. If the final carry over here after the sum of last most significant bit is 1, it is dropped and the result is positive.

If there is no carry over here then 2’s complement will negative and negative bit set as 1.

Overflow bit : Overflow bit used to indicate that stack is overflow or not after processing the instruction. It is set to be 1 means that stack is overflow if it is 0 then its reverse to happen.

Parity Bit : Parity bit represent odd or even set of ’1’ bits in given string. It is used as error detecting code. Parity bit has two types: Even parity bit and an Odd parity bit.

In Even parity bit, we count the occurrence of I’s in the string. If a number of 1 bit is odd in counting than we will add even parity bit to make it even or if the number of 1 bit are even then even parity bit is 0.

Data	Number of 1 bits	even parity bit	Data including Even Parity bit
1010111	5	1	11010111

Memory Address Register: Address register holds the address of memory where data is residing. CPU fetches the address from the register and access the location to acquire data. In the same way, MAR is used to write the data into memory.

Data Register: Data registers also Known as Memory Data Register. It holds the content or instruction fetched from memory location for reading and writing purpose. It is 16-bit register means that can store 2^{16}bytes of data. From Data, register instruction moves in Instruction register and data content moves to AC for manipulation.

Instruction register: Instruction holds the instruction to be executed .control unit of CPU fetch the instruction, decode it and execute the instruction by accessing appropriate content.IR is 16-bit register. It has two fields – Opcode and operand.

PC holds the address of the instruction to be executed. Once the address is fetched it gets incremented by 1.PC hold the address of next instructions. In this situation, IR holds the address of the current instruction.

Input /output register: Input register holds the input from input devices and output register hold the output that has to give to output devices.

Micro-Operation

In general, the Arithmetic Micro-operations deals with the operations performed on numeric data stored in the registers.

The basic Arithmetic Micro-operations are classified in the following categories:

1. Addition
2. Subtraction
3. Increment
4. Decrement
5. Shift

Some additional Arithmetic Micro-operations are classified as:

1. Add with carry
2. Subtract with borrow
3. Transfer/Load, etc.

The following table shows the symbolic representation of various Arithmetic Micro-operations.

Symbolic Representation	Description
R3 ← R1 + R2	The contents of R1 plus R2 are transferred to R3.
R3 ← R1 - R2	The contents of R1 minus R2 are transferred to R3.
R2 ← R2'	Complement the contents of R2 (1's complement)
R2 ← R2' + 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2' + 1	R1 plus the 2's complement of R2 (subtraction)
R1 ← R1 + 1	Increment the contents of R1 by one
R1 ← R1 - 1	Decrement the contents of R1 by one

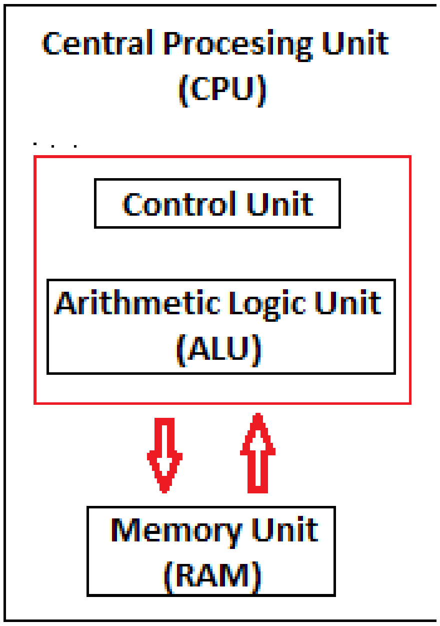
ALU- chip

A Central Processing Unit is also called a processor, central processor, or microprocessor. It carries out all the important functions of a computer. It receives instructions from both the hardware and active software and produces output accordingly. It stores all important programs like operating systems and application software. CPU also helps Input and output devices to communicate with each other. Owing to these features of CPU, it is often referred to as the brain of the computer.

CPU is installed or inserted into a CPU socket located on the motherboard. Furthermore, it is provided with a heat sink to absorb and dissipate heat to keep the CPU cool and functioning smoothly.

Generally, a CPU has three components:

- ALU (Arithmetic Logic Unit)
- Control Unit
- Memory or Storage Unit



Control Unit: It is the circuitry in the control unit, which makes use of electrical signals to instruct the computer system for executing already stored instructions. It takes instructions from memory and then decodes and executes these instructions. So, it controls and coordinates the functioning of all parts of the computer. The Control Unit's main task is to maintain and regulate the flow of information across the processor. It does not take part in processing and storing data.

ALU: It is the arithmetic logic unit, which performs arithmetic and logical functions. Arithmetic functions include addition, subtraction, multiplication division, and comparisons. Logical functions mainly include selecting, comparing, and merging the data. A CPU may contain more than one ALU. Furthermore, ALUs can be used for maintaining timers that help run the computer.

Memory or Storage Unit/ Registers: It is called Random access memory (RAM). It temporarily stores data, programs, and intermediate and final results of processing. So, it acts as a temporary storage area that holds the data temporarily, which is used to run the computer.

What is CPU Clock Speed?

The clock speed of a CPU or a processor refers to the number of instructions it can process in a second. It is measured in gigahertz. For example, a CPU with a clock speed of 4.0 GHz means it can process 4 billion instructions in a second.

Types of CPU:

CPUs are mostly manufactured by Intel and AMD, each of which manufactures its own types of CPUs. In modern times, there are lots of CPU types in the market. Some of the basic types of CPUs are described below:

Single Core CPU: Single Core is the oldest type of computer CPU, which was used in the 1970s. It has only one core to process different operations. It can start only one operation at a time; the CPU switches back and forth between different sets of data streams when more than one program runs. So, it is not suitable for multitasking as the performance will be reduced if more than one application runs. The performance of these CPUs is mainly dependent on the clock speed. It is still used in various devices, such as smartphones.

Dual Core CPU: As the name suggests, Dual Core CPU contains two cores in a single Integrated Circuit (IC). Although each core has its own controller and cache, they are linked together to work as a single unit and thus can perform faster than the single-core processors and can handle multitasking more efficiently than Single Core processors.

Quad Core CPU: This type of CPU comes with two dual-core processors in one integrated circuit (IC) or chip. So, a quad-core processor is a chip that contains four independent units called cores. These cores read and execute instructions of CPU. The cores can run multiple instructions simultaneously, thereby increases the overall speed for programs that are compatible with parallel processing.

Quad Core CPU uses a technology that allows four independent processing units (cores) to run in parallel on a single chip. Thus by integrating multiple cores in a single CPU, higher performance can be generated without boosting the clock speed. However, the performance increases only when the computer's software supports multiprocessing. The software which supports multiprocessing divides the processing load between multiple processors instead of using one processor at a time.

History of CPU:

Some of the important events in the development of CPU since its invention till date are as follows:

- In 1823, Baron Jons Jackob Berzelius discovered silicon that is the main component of CPU till date.
- In 1903, Nikola Tesla got gates or switches patented, which are electrical logic circuits.
- In December 1947, John Bardeen, William Shockley, and Walter Brattain invented the first transistor at the Bell Laboratories and got it patented in 1948.
- In 1958, the first working integrated circuit was developed by Robert Noyce and Jack Kilby.
- In 1960, IBM established the first mass-production facility for transistors in New York.
- In 1968, Robert Noyce and Gordon Moore founded Intel Corporation.
- AMD (Advanced Micro Devices) was founded in May 1969.
- In 1971, Intel introduced the first microprocessor, the Intel 4004, with the help of Ted Hoff.
- In 1972, Intel introduced the 8008 processor; in 1976, Intel 8086 was introduced, and in June 1979, Intel 8088 was released.
- In 1979, a 16/32-bit processor, the Motorola 68000, was released. Later, it was used as a processor for the Apple Macintosh and Amiga computers.
- In 1987, Sun introduced the SPARC processor.
- In March 1991, AMD introduced the AM386 microprocessor family.
- In March 1993, Intel released the Pentium processor. In 1995, Cyrix introduced the Cx5x86 processor to give competition to Intel Pentium processors.
- In January 1999, Intel introduced the Celeron 366 MHz and 400 MHz processors.
- In April 2005, AMD introduced its first dual-core processor.
- In 2006, Intel introduced the Core 2 Duo processor.
- In 2007, Intel introduced different types of Core 2 Quad processors.
- In April 2008, Intel introduced the first series of Intel Atom processors, the Z5xx series. They were single-core processors with a 200 MHz GPU.
- In September 2009, Intel released the first Core i5 desktop processor with four cores.
- In January 2010, Intel released many processors such as Core 2 Quad processor Q9500, first Core i3 and i5 mobile processors, first Core i3 and i5

desktop processors. In the same year in July, it released the first Core i7 desktop processor with six cores.

- In June 2017, Intel introduced the first Core i9 desktop processor.
- In April 2018, Intel released the first Core i9 mobile processor.

Faster Algorithm and Implementation (multiplication & Division)

Booth algorithm gives a procedure for multiplying binary integers in signed 2's complement representation in efficient way, i.e., less number of additions/subtractions required. It operates on the fact that strings of 0's in the multiplier require no addition but just shifting and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as 2^(k+1) to 2^m.

As in all multiplication schemes, booth algorithm requires examination of the multiplier bits and shifting of the partial product. Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to following rules:

1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier

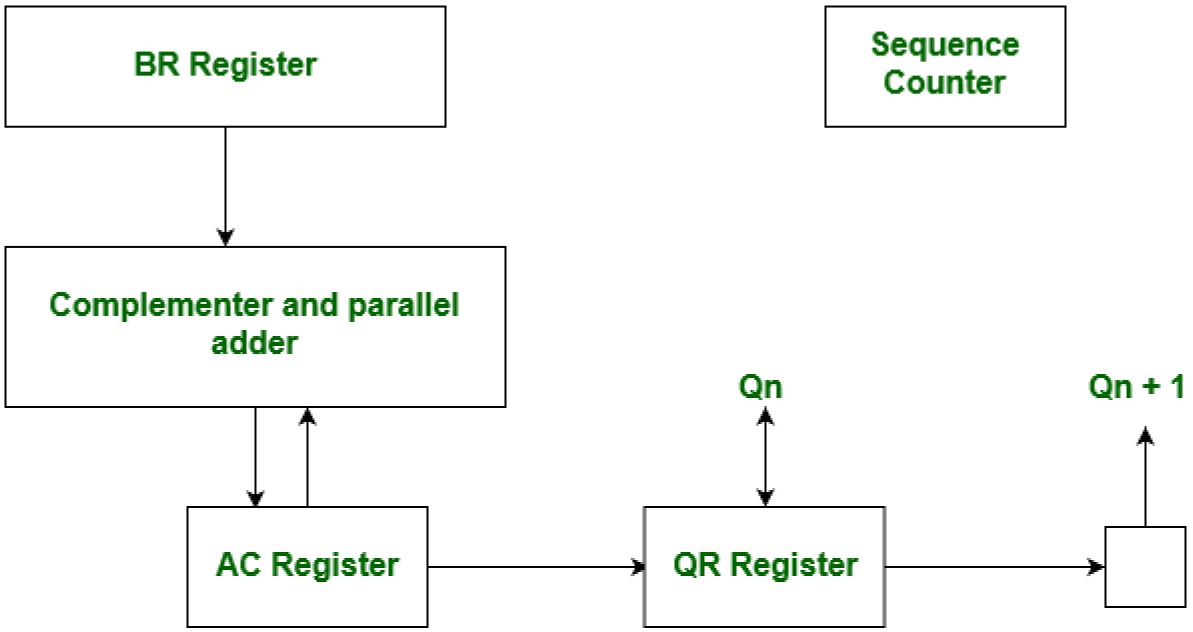
2. The multiplicand is added to the partial product upon encountering the first 0 (provided that there was a previous '1') in a string of 0's in the multiplier.

3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

Hardware Implementation of Booths Algorithm – The hardware implementation of the booth algorithm requires the register configuration shown in the figure below.

Booth's Algorithm Flowchart –

We name the register as A, B and Q, AC, BR and QR respectively. Qn designates the least significant bit of multiplier in the register QR. An extra flip-flop Qn+1is appended to QR to facilitate a double inspection of the multiplier.The flowchart for the booth algorithm is shown below.



AC and the appended bit Qn+1 are initially cleared to 0 and the sequence SC is set to a number n equal to the number of bits in the multiplier. The two bits of the multiplier in Qn and Qn+1are inspected. If the two bits are equal to 10, it means that the first 1 in a string has been encountered. This requires subtraction of the multiplicand from the partial product in AC. If the 2 bits are equal to 01, it means that the first 0 in a string of 0's has been encountered. This requires the addition of the multiplicand to the partial product in AC.

When the two bits are equal, the partial product does not change. An overflow cannot occur because the addition and subtraction of the multiplicand follow each other. As a consequence, the 2 numbers that are added always have a opposite signs, a condition that excludes an overflow. The next step is to shift right the partial product and the multiplier (including Qn+1). This is an arithmetic shift right (ashr) operation which AC and QR ti the right and leaves the sign bit in AC unchanged. The sequence counter is decremented and the computational loop is repeated n times.

Example – A numerical example of booth's algorithm is shown below for n = 4. It shows the step by step multiplication of -5 and -7.

MD = -5 = 1011, MD = 1011, MD'+1 = 0101

MR = -7 = 1001

The explanation of first step is as follows: Qn+1

AC = 0000, MR = 1001, Qn+1 = 0, SC = 4

Qn Qn+1 = 10

So, we do AC + (MD)'+1, which gives AC = 0101

On right shifting AC and MR, we get

AC = 0010, MR = 1100 and Qn+1 = 1

OPERATION	AC	MR	QN+1	SC
	0000	1001	0	4
AC + MD’ + 1	0101	1001	0	
ASHR	0010	1100	1	3
AC + MR	1101	1100	1	
ASHR	1110	1110	0	2
ASHR	1111	0111	0	1
AC + MD’ + 1	0010	0011	1	0

Product is calculated as follows:

Product = AC MR

Product = 0010 0011 = 35

Unit-II

(Basic Organization)

Von Neumann Machine (IAS Computer)

In this chapter of COA tutorial, we will learn:

- what is ias computer?
- working of ias computer,
- important points about ias computer,
- ias computer structure,
- What is von neumann bottleneck?

What is ias computer?

Von-neumann designed a new stored-program computer in 1946 with support from his collegues at Institute for Advanced Studies, Princeton. The computer is today known as IAS computer.

Most of the computers still use the stored program concept of Von-neumann.

IAS computer works on following principles:

- Same memory is used to store both the program and data.
- The program is executed in written sequence.
- A program can modify itself when computer executes the program.

IAS computer CPU used several vacuum-tubes to store operands and results.

Important facts about IAS computer

Given below are important facts about IAS computer or Von-neumann computer.

- The memory contains words that basically represents data or instruction.
- The basic data is a binary number in IAS computer.

- IAS instructions are 20 bits long.
- Instruction in IAS consists of two parts - 1) operation code or op-code which is of 8-bit. 2) address of 12-bit
- IAS instruction allow only one memory address.

IAS computer structure

The structure of IAS computer is shown below:

The IAS computer has CPU.
CPU consists of program control unit and data processing unit. It also contains control unit along with various set of high speed registers. These registers are meant for temporary storage of instructions and data.

The main memory is used for storing programs and data.

Von-Neumann bottleneck

In Von-Neumann computer the CPU has to wait longer to obtain data from the memory in comparision to recieving data from registers. Reason is quite obvious - registers are much faster.

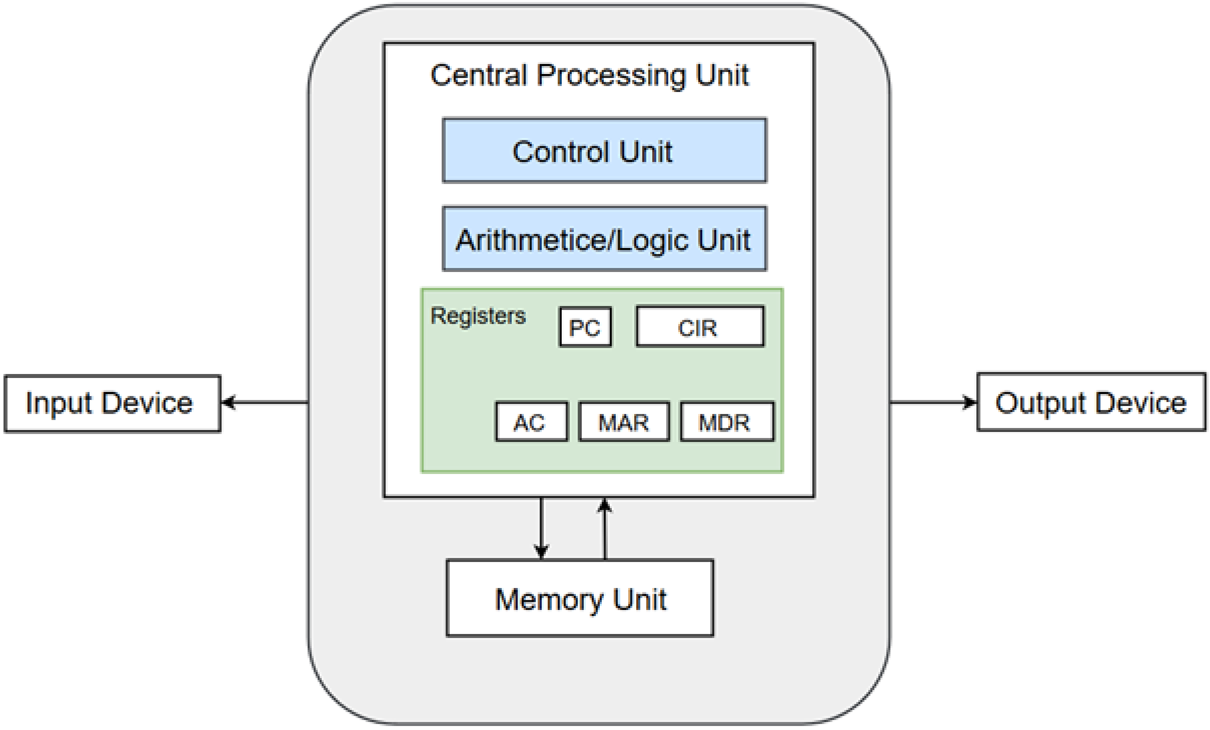
Von-Neumann proposed his computer architecture design in 1945 which was later known as Von-Neumann Architecture. It consisted of a Control Unit, Arithmetic, and Logical Memory Unit (ALU), Registers and Inputs/Outputs.

Von Neumann architecture is based on the stored-program computer concept, where instruction data and program data are stored in the same memory. This design is still used in most computers produced today.

A Von Neumann-based computer:

- Uses a single processor
- Uses one memory for both instructions and data.
- Executes programs following the fetch-decode-execute cycle

Von-Neumann Basic Structure:



Components of Von-Neumann Model:

- Central Processing Unit
- Buses
- Memory Unit

Central Processing Unit

The part of the Computer that performs the bulk of data processing operations is called the Central Processing Unit and is referred to as the CPU.

The Central Processing Unit can also be defined as an electric circuit responsible for executing the instructions of a computer program.

The CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer.

The major components of CPU are Arithmetic and Logic Unit (ALU), Control Unit (CU) and a variety of registers.

Arithmetic and Logic Unit (ALU)

The Arithmetic and Logic Unit (ALU) performs the required micro-operations for executing the instructions. In simple words, ALU allows arithmetic (add, subtract, etc.) and logic (AND, OR, NOT, etc.) operations to be carried out.

Control Unit

The Control Unit of a computer system controls the operations of components like ALU, memory and input/output devices.

The Control Unit consists of a program counter that contains the address of the instructions to be fetched and an instruction register into which instructions are fetched from memory for execution.

Registers

Registers refer to high-speed storage areas in the CPU. The data processed by the CPU are fetched from the registers.

Following is the list of registers that plays a crucial role in data processing.

Registers	Description
MAR (Memory Address Register)	This register holds the memory location of the data that needs to be accessed.
MDR (Memory Data Register)	This register holds the data that is being transferred to or from memory.
AC (Accumulator)	This register holds the intermediate arithmetic and logic results.
PC (Program Counter)	This register contains the address of the next instruction to be executed.
CIR (Current Instruction Register)	This register contains the current instruction during processing.

Buses

Buses are the means by which information is shared between the registers in a multiple-register configuration system.

A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during each particular register transfer.

Von-Neumann Architecture comprised of three major bus systems for data transfer.

Bus	Description
Address Bus	Address Bus carries the address of data (but not the data) between the processor and the memory.
Data Bus	Data Bus carries data between the processor, the memory unit and the input/output devices.
Control Bus	Control Bus carries signals/commands from the CPU.

Memory Unit

A memory unit is a collection of storage cells together with associated circuits needed to transfer information in and out of the storage. The memory stores binary information in groups of bits called words. The internal structure of a memory unit is specified by the number of words it contains and the number of bits in each word.

Two major types of memories are used in computer systems:

1. RAM (Random Access Memory)
2. ROM (Read-Only Memory)

Operational flow chart (Fetch, Execute)

The CPU carries out billions of instructions a second. Each cycle, the CPU fetches an instruction, decodes it to find out what to do and then carries out the instruction (executes it).

This process happens continuously whilst the computer is on and uses a number of registers:

PC - Program Counter
MAR - Memory Address Register
MDR - Memory Data Register
CIR - Current Instruction Register

It also uses the two main units in the CPU; the control unit and the arithmetic logic unit.

During the fetch phase the next instruction address is copied from the PC into the MAR which then fetches the instruction at that location using the address bus. The instruction is held in the MDR register and duplicated into the CIR register. The PC then increments by one.

During the decode stage, the instruction in the CIR is decoded.

Finally the instruction is carried out in the execute stage, the CPU checks for interrupts and the process starts again.

To explain what is happening during a CPU cycle we can use register transfer notation (RTN) to show the data and instructions that are being transferred without describing how it is done.

The first register notation you will normally see is when the program counter loads into the memory address register. This is represented as below:

MAR <- [PC]

The data that is being transferred is on the right and the location of where it is going is on the left. The data in the brackets is the data that is being loaded. You will also sometimes see basic arithmetic operations such as +, -, * and /. An example of this is as follows, where the program counter increments to the next memory address.

MAR <- [PC]+1

Another common register transfer is when the memory data register copies its contents to the current instruction register.

CIR <- [MDR]

If a register is written on its own within brackets, it means that it is decoded and executed. For example:

[CIR]

describe how interrupts are handled

Interrupts are generated by hardware or software and can happen at any time. These occur to tell the CPU that something needs immediate attention. This interrupt will cause the CPU to stop its current task, save its current position, and deal with the interrupt before returning back to the saved program counter.

A computer is generally trying to deal with multiple interrupts at the same time. An OS will normally have two programs to handle this. An interrupt handler prioritises the interrupts and stores them in a cue waiting for the CPU to deal with them. A another program called a scheduler deals with which program should have control next.

Interrupts allow a computer to run multiple tasks at the same time - otherwise known as multitasking.

Instruction Cycle

The generic instruction cycle for an unspecified CPU consists of the following stages:

1. Fetch instruction: Read instruction code from address in PC and place in IR. (IR ← Memory[PC])

2. Decode instruction: Hardware determines what the opcode/function is, and determines which registers or memory addresses contain the operands.

3. Fetch operands from memory if necessary: If any operands are memory addresses, initiate memory read cycles to read them into CPU registers. If an operand is in memory, not a register, then the memory address of the operand is known as the effective address, or EA for short. The fetching of an operand can therefore be denoted as Register ← Memory[EA]. On today's computers, CPUs are much faster than memory, so operand fetching usually takes multiple CPU clock cycles to complete.

4. Execute: Perform the function of the instruction. If arithmetic or logic instruction, utilize the ALU circuits to carry out the operation on data in registers. This is the only stage of the instruction cycle that is useful from the perspective of the end user. Everything else is overhead required to make the execute stage happen. One of the major goals of CPU design is to eliminate overhead, and spend a

higher percentage of the time in the execute stage. Details on how this is achieved is a topic for a hardware-focused course in computer architecture.

5. Store result in memory if necessary: If destination is a memory address, initiate a memory write cycle to transfer the result from the CPU to memory. Depending on the situation, the CPU may or may not have to wait until this operation completes. If the next instruction does not need to access the memory chip where the result is stored, it can proceed with the next instruction while the memory unit is carrying out the write operation.

An example of a full instruction cycle is provided by the following VAX instruction, which uses memory addresses for all three operands.

```
mull    x, y, product
```

1. Fetch the instruction code from Memory[PC]

2. Decode the instruction. This reveals that it's a multiply instruction, and that the operands are memory locations x, y, and product.

3. Fetch x and y from memory.

4. Multiply x and y, storing the result in a CPU register.

5. Save the result from the CPU to memory location product.

5.6.2. MIPS Instruction Cycle

Since the MIPS is a load-store architecture, all instructions except load and store get their operands from CPU registers and store their result in a CPU register. Hence, the instruction cycle for all instructions except load and store is somewhat simpler. When all operands are in CPU registers, which can be accessed within a single clock cycle, fetching operands and storing the results can occur within the same clock cycle as execution (add, subtract, etc.). For example, suppose R0, R1, R2 ... R15 are CPU registers. Then the operation

```
R0 ← R4 + R7          # One clock cycle
```

is a simple, atomic operation inside the CPU, and therefore is not regarded as multiple steps in the instruction cycle. If one of operands were in memory instead of a register, on the other hand, fetching it from memory and placing it into a register would be a separate step.

```
R4 ← Mem[address1]    # Multiple clock cycles
R0 ← R4 + R7          # One clock cycle
```

1. Fetch instruction from memory to IR
2. Decode
3. Execute (all data in CPU registers)

The specific cycle for a load instruction is:

1. Fetch instruction from memory to IR
2. Decode
3. Fetch operand from memory to a register

The specific cycle for a store instruction is:

1. Fetch instruction from memory to IR
2. Decode
3. Store operand from register to memory

5.6.3. Analysis of the Instruction Cycle

Note that in any case, most of the instruction cycle is overhead. Only the execute stage actually does something considered useful by the user, and all the other stages are fluff, either preparation or wrap-up.

One way to increase the density of useful work in a program is by making more complex instructions. If the execute cycle accomplishes more for the same amount of fetching, decoding and storing overhead, then the program will be shorter, and will run faster. This is the philosophy behind CISC architectures. A classic example of this idea is the VAX polyf instruction, which evaluates a polynomial given an array of coefficients, the order or the polynomial, and the value of x. It accomplishes in one instruction cycle what would require a loop, and hence dozens of instruction cycles otherwise.

The cost of overhead can also be alleviated without actually reducing it. The primary technique to achieve this is called pipelining. A pipelined CPU overlaps the execution of two or more instructions, so that while one instruction is executing, the next one is already being decoded, and the one after that is being fetched. Pipelining is discussed in Chapter 17, A Pipelined Implementation.

Organization of Central Processing Unit

Central Processing Unit (CPU) consists of the following features −

- CPU is considered as the brain of the computer.

- CPU performs all types of data processing operations.

- It stores data, intermediate results, and instructions (program).
- It controls the operation of all parts of the computer.

CPU itself has following three components.

- Memory or Storage Unit
- Control Unit
- ALU(Arithmetic Logic Unit)

Memory or Storage Unit

This unit can store instructions, data, and intermediate results. This unit supplies information to other units of the computer when needed. It is also known as internal storage unit or the main memory or the primary storage or Random Access Memory (RAM).

Its size affects speed, power, and capability. Primary memory and secondary memory are two types of memories in the computer. Functions of the memory unit are −

- It stores all the data and the instructions required for processing.
- It stores intermediate results of processing.
- It stores the final results of processing before these results are released to an output device.
- All inputs and outputs are transmitted through the main memory.

Control Unit

This unit controls the operations of all parts of the computer but does not carry out any actual data processing operations.

Functions of this unit are −

- It is responsible for controlling the transfer of data and instructions among other units of a computer.
- It manages and coordinates all the units of the computer.
- It obtains the instructions from the memory, interprets them, and directs the operation of the computer.
- It communicates with Input/Output devices for transfer of data or results from storage.
- It does not process or store data.

ALU (Arithmetic Logic Unit)

This unit consists of two subsections namely,

- Arithmetic Section
- Logic Section

Arithmetic Section

Function of arithmetic section is to perform arithmetic operations like addition, subtraction, multiplication, and division. All complex operations are done by making repetitive use of the above operations.

Logic Section

Function of logic section is to perform logic operations such as comparing, selecting, matching, and merging of data.

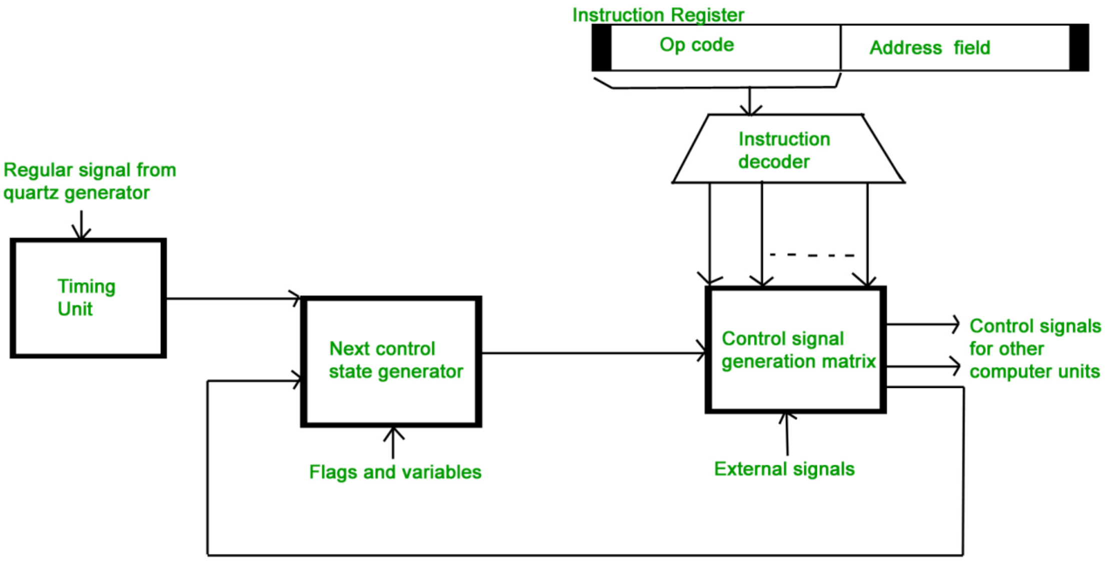
Hardwired & micro programmed control unit

To execute an instruction, the control unit of the CPU must generate the required control signal in the proper sequence. There are two approaches used for generating the control signals in proper sequence as Hardwired Control unit and Micro-programmed control unit.

Hardwired Control Unit –

The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes and the external inputs. The outputs of the state machine are the control signals. The sequence of the operation carried out by this machine is determined by the wiring of the logic elements and hence named as “hardwired”.

- Fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.
- Hardwired control is faster than micro-programmed control.
- A controller that uses this approach can operate at high speed.
- RISC architecture is based on hardwired control unit


Instruction Register
Op code
Address field
Instruction decoder
Regular signal from quartz generator
Timing Unit
Next control state generator
Control signal generation matrix
Control signals for other computer units
Flags and variables
External signals

Micro-programmed Control Unit –

- The control signals associated with operations are stored in special memory units inaccessible by the programmer as Control Words.

- Control signals are generated by a program are similar to machine language programs.

- Micro-programmed control unit is slower in speed because of the time it takes to fetch microinstructions from the control memory.

Some Important Terms –

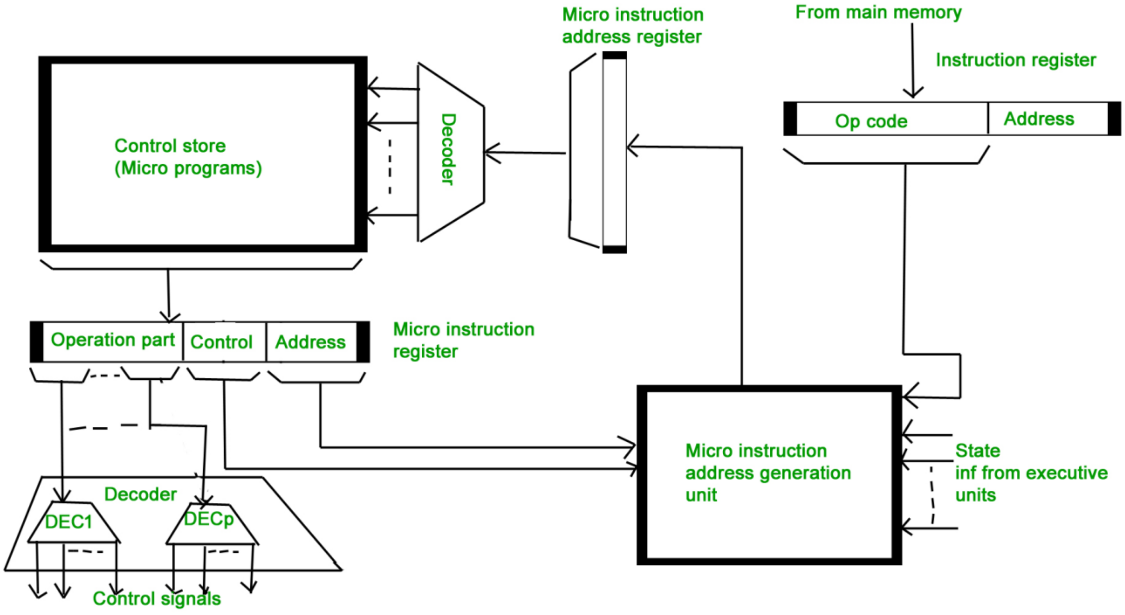
1. **Control Word :** A control word is a word whose individual bits represent various control signals.

2. **Micro-routine :** A sequence of control words corresponding to the control sequence of a machine instruction constitutes the micro-routine for that instruction.

3. **Micro-instruction :** Individual control words in this micro-routine are referred to as microinstructions.

4. **Micro-program :** A sequence of micro-instructions is called a micro-program, which is stored in a ROM or RAM called a Control Memory (CM).

5. **Control Store :** the micro-routines for all instructions in the instruction set of a computer are stored in a special memory called the Control Store.


Micro instruction address register
From main memory
Instruction register
Op code
Address
Control store (Micro programs)
Decoder
Operation part
Control
Address
Micro instruction register
Micro instruction address generation unit
State inf from executive units
Decoder
DEC1
DECp
Control signals

Types of Micro-programmed Control Unit – Based on the type of Control Word stored in the Control Memory (CM), it is classified into two types :

1. Horizontal Micro-programmed control Unit :

The control signals are represented in the decoded binary format that is 1 bit/CS. Example: If 53 Control signals are present in the processor than 53 bits are required. More than 1 control signal can be enabled at a time.

- It supports longer control word.
- It is used in parallel processing applications.
- It allows higher degree of parallelism. If degree is n, n CS are enabled at a time.
- It requires no additional hardware(decoders). It means it is faster than Vertical Microprogrammed.
- It is more flexible than vertical microprogrammed

2. Vertical Micro-programmed control Unit :

The control signals re represented in the encoded binary format. For N control signals- Log2(N) bits are required.

- It supports shorter control words.

- It supports easy implementation of new conrol signals therefore it is more flexible.
- It allows low degree of parallelism i.e., degree of parallelism is either 0 or 1.
- Requires an additional hardware (decoders) to generate control signals, it implies it is slower than horizontal microprogrammed.
- It is less flexible than horizontal but more flexible than that of hardwired control unit.

Single Organization

omputer perform task on the basis of instruction provided. An instruction in computer comprises of groups called fields. These field contains different information as for computers every thing is in 0 and 1 so each field has different significance on the basis of which a CPU decide what to perform. The most common fields are:

- Operation field which specifies the operation to be performed like addition.
- Address field which contain the location of operand, i.e., register or memory location.
- Mode field which specifies how operand is to be founded.

An instruction is of various length depending upon the number of addresses it contain. Generally CPU organization are of three types on the basis of number of address fields:

1. Single Accumulator organization
2. General register organization
3. Stack organization

In first organization operation is done involving a special register called accumulator. In second on multiple registers are used for the computation purpose. In third organization the work on stack basis operation due to which it does not contain any address field. It is not necessary that only a single organization is applied a blend of various organization is mostly what we see generally.

On the basis of number of address, instruction are classified as:

Note that we will use X = (A+B)*(C+D) expression to showcase the procedure.

1. **Zero Address Instructions –**

A stack based computer do not use address field in instruction.To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

Expression: X = (A+B)*(C+D)

Postfixed : X = AB+CD+*

TOP means top of stack

M[X] is any memory location

PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)
POP	X	M[X] = TOP

2. **One Address Instructions –**
   This use a implied ACCUMULATOR register for data manipulation.One operand is in accumulator and other is in register or memory location.Implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.

Expression: X = (A+B)*(C+D)

AC is accumulator

M[] is any memory location

M[T] is temporary location

LOAD	A	AC = M[A]
ADD	B	AC = AC + M[B]
STORE	T	M[T] = AC
LOAD	C	AC = M[C]
ADD	D	AC = AC + M[D]
MUL	T	AC = AC * M[T]
STORE	X	M[X] = AC

3. **Two Address Instructions –**
   This is common in commercial computers.Here two address can be specified in the instruction.Unlike earlier in one address instruction the result was stored in accumulator here result cab be stored at different location rather than just accumulator, but require more number of bit to represent address.

Here destination address can also contain operand.

Expression: X = (A+B)*(C+D)

R1, R2 are registers

M[] is any memory location

MOV	R1, A	R1 = M[A]
ADD	R1, B	R1 = R1 + M[B]
MOV	R2, C	R2 = C
ADD	R2, D	R2 = R2 + D
MUL	R1, R2	R1 = R1 * R2
MOV	X, R1	M[X] = R1

4. **Three Address Instructions –**
   This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.

```
Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location
```

ADD	R1, A, B	R1 = M[A] + M[B]
ADD	R2, C, D	R2 = M[C] + M[D]
MUL	X, R1, R2	M[X] = R1 * R2

General Register Organization

When we are using multiple general purpose registers, instead of single accumulator register, in the CPU Organization then this type of organization is known as General register based CPU Organization. In this type of organization, computer uses two or three address fields in their instruction format. Each address field may specify a general register or a memory word.If many CPU registers are available for heavily used variables and intermediate results, we can avoid memory references much of the time, thus vastly increasing program execution speed, and reducing program size.

For example:

```
MULT R1, R2, R3
```

This is an instruction of an arithmatic multiplication written in assembly language. It uses three address fields R1, R2 and R3. The meaning of this instruction is:

```
R1 <-- R2 * R3
```

This instruction also can be written using only two address fields as:

```
MULT R1, R2
```

In this instruction, the destination register is the same as one of the source registers. This means the operation

```
R1 <-- R1 * R2
```

The use of large number of registers results in short program with limited instructions.

Some examples of General register based CPU Organization are **IBM 360 and PDP- 11**.
The advantages of General register based CPU organization –

- Efficiency of CPU increases as there are large number of registers are used in this organization.

- Less memory space is used to store the program since the instructions are written in compact way.

The disadvantages of General register based CPU organization –

- Care should be taken to avoid unnecessary usage of registers. Thus, compilers need to be more intelligent in this aspect.

- Since large number of registers are used, thus extra cost is required in this organization.

General register CPU organisation of two type:

1. Register-memory reference architecture (CPU with less register)– In this organisation Source 1 is always required in register, source 2 can be present either in register or in memory.Here two address instruction format is the compatible instruction format.

2. Register-register reference architecture(CPU with more register)– In this organisation ALU operations are performed only on a register data. So operands are required in the register. After manipulation result is also placed in register.Here three address instruction format is the compatible instruction format.

Stack Organization

STACK ORGANIZATION. Stack is a storage structure that stores information in such a way that the last item stored is the first item retrieved. It is based on the principle of LIFO (Last-in-first-out). The stackin digital computers is a group of memory locations with a register that holds the address of top of element.

The computers which use Stack-based CPU Organization are based on a data structure called stack. The stack is a list of data words. It uses Last In First Out (LIFO) access method which is the most popular access method in most of the CPU. A register is used to store the address of the topmost element of the stack which is known as Stack pointer (SP). In this organisation, ALU operations are performed on stack data. It means both the operands are always required on the stack. After manipulation, the result is placed in the stack.

The main two operations that are performed on the operators of the stack are Push and Pop. These two operations are performed from one end only.

1. Push –

This operation results in inserting one operand at the top of the stack and it decrease the stack pointer register. The format of the PUSH instruction is:

```
PUSH
```

It inserts the data word at specified address to the top of the stack. It can be implemented as:

```
//decrement SP by 1
SP <-- SP - 1


//store the content of specified memory address
//into SP; i.e, at top of stack
SP <-- (memory address)
```

2. Pop –

This operation results in deleting one operand from the top of the stack and it increase the stack pointer register. The format of the POP instruction is:

```
POP
```

It deletes the data word at the top of the stack to the specified address. It can be implemented as:

```
//transfer the content of  SP (i.e, at top most data)
//into specified memory location
(memory address) <-- SP


//increment SP by 1
SP <-- SP + 1
```

Operation type instruction does not need the address field in this CPU organization. This is because the operation is performed on the two operands that are on the top of the stack. For example:

```
SUB
```

Addressing modes

The term addressing modes refers to the way in which the operand of an instruction is specified. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.

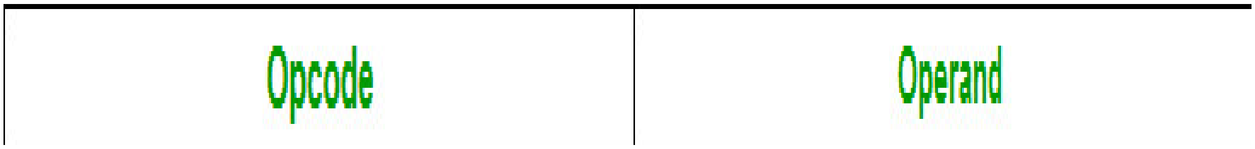
Addressing modes for 8086 instructions are divided into two categories:

1) Addressing modes for data

2) Addressing modes for branch

The 8086 memory addressing modes provide flexible access to memory, allowing you to easily access variables, arrays, records, pointers, and other complex data types. The key to good assembly language programming is the proper use of memory addressing modes.

An assembly language program instruction consists of two parts



Opcode	Operand

The memory address of an operand consists of two components:

IMPORTANT TERMS

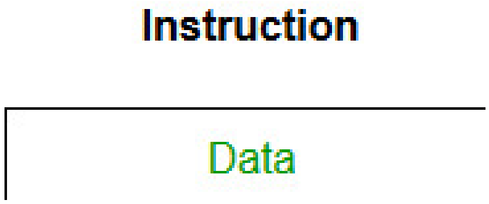
- **Starting address** of memory segment.
- **Effective address or Offset**: An offset is determined by adding any combination of three address elements: **displacement, base and index.**
    - **Displacement:** It is an 8 bit or 16 bit immediate value given in the instruction.
    - **Base**: Contents of base register, BX or BP.

- **Index**: Content of index register SI or DI.

According to different ways of specifying an operand by 8086 microprocessor, different addressing modes are used by 8086.

Addressing modes used by 8086 microprocessor are discussed below:

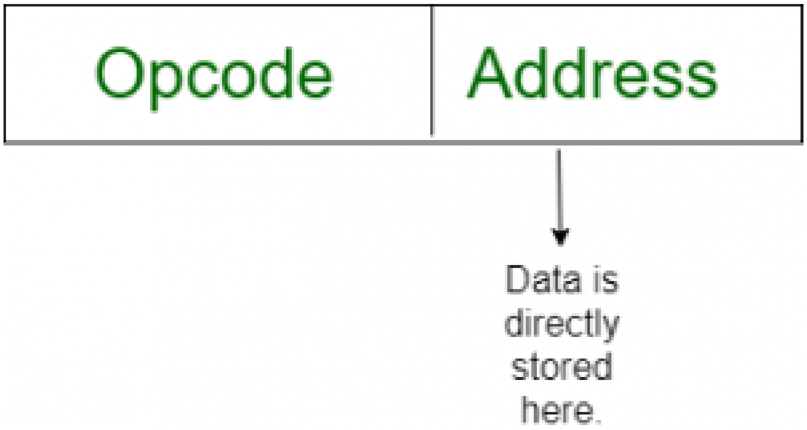
- **Implied mode:**: In implied addressing the operand is specified in the instruction itself. In this mode the data is 8 bits or 16 bits long and data is the part of instruction.Zero address instruction are designed with implied addressing mode.



Example: CLC (used to reset Carry flag to 0)

- **Immediate addressing mode (symbol #):**In this mode data is present in address field of instruction .Designed like one address instruction format.

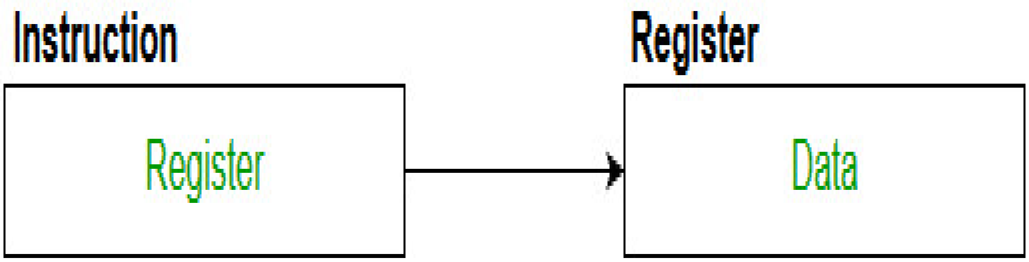
  Note: Limitation in the immediate mode is that the range of constants are restricted by size of address field.



Example: MOV AL, 35H (move the data 35H into AL register)

- **Register mode:** In register addressing the operand is placed in one of 8 bit or 16 bit general purpose registers. The data is in the register that is specified by the instruction.
-

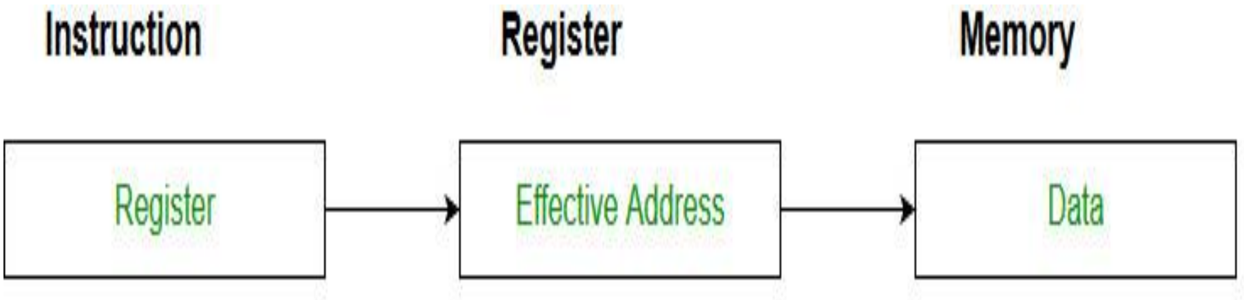
  Here one register reference is required to access the data.



Example: MOV AX,CX (move the contents of CX register to AX register)

- **Register Indirect mode**: In this addressing the operand’s offset is placed in any one of the registers BX,BP,SI,DI as specified in the instruction. The effective address of the data is in the base register or an index register that is specified by the instruction.
-

Here two register reference is required to access the data.



The 8086 CPUs let you access memory indirectly through a register using the register indirect addressing modes.

- MOV AX, [BX](move the contents of memory location s

addressed by the register BX to the register AX)

- **Auto Indexed (increment mode)**: Effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next consecutive memory location.**(R1)+**.

Here one register reference,one memory reference and one ALU operation is required to access the data.

Example:

- Add R1, (R2)+ // OR

- R1 = R1 +M[R2]

  R2 = R2 + d

  Useful for stepping through arrays in a loop. R2 – start of array d – size of an element

- **Auto indexed (decrement mode)**: Effective address of the operand is the contents of a register specified in the instruction. Before accessing the operand, the contents of this register are automatically decremented to point to the previous consecutive memory location. **–(R1)**
  Here one register reference,one memory reference and one ALU operation is required to access the data.

Example:

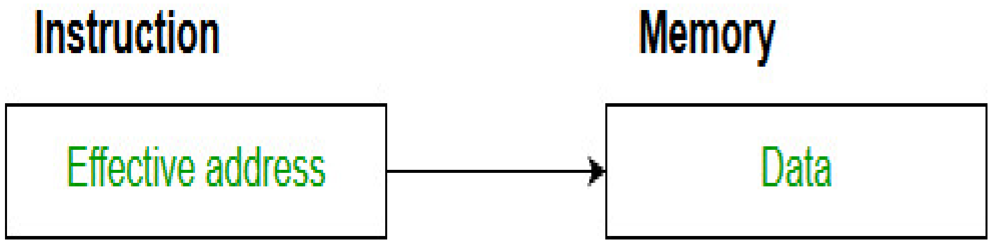
Add R1,-(R2)   //OR

R2 = R2-d
R1 = R1 + M[R2]

Auto decrement mode is same as auto increment mode. Both can also be used to implement a stack as push and pop . Auto increment and Auto decrement modes are useful for implementing “Last-In-First-Out” data structures.

- **Direct addressing/ Absolute addressing Mode (symbol [ ]):** The operand’s offset is given in the instruction as an 8 bit or 16 bit displacement element. In this addressing mode the 16 bit effective address of the data is the part of the instruction.

  Here only one memory reference operation is required to access the data.



Example:ADD AL,[0301]   //add the contents of offset address 0301 to AL

- **Indirect addressing Mode (symbol @ or ())**:In this mode address field of instruction contains the address of effective address.Here two references are required.

1st reference to get effective address.

2nd reference to access the data.

Based on the availability of Effective address, Indirect mode is of two kind:

1. Register Indirect:In this mode effective address is in the register, and corresponding register name will be maintained in the address field of an instruction.

   Here one register reference,one memory reference is required to access the data.

2. Memory Indirect:In this mode effective address is in the memory, and corresponding memory address will be maintained in the address field of an instruction.

   Here two memory reference is required to access the data.

- **Indexed addressing mode**: The operand's offset is the sum of the content of an index register SI or DI and an 8 bit or 16 bit displacement.

  Example:MOV AX, [SI +05]

- **Based Indexed Addressing:** The operand's offset is sum of the content of a base register BX or BP and an index register SI or DI.

  Example: ADD AX, [BX+SI]

Based on Transfer of control, addressing modes are:

- **PC relative addressing mode:** PC relative addressing mode is used to implement intra segment transfer of control, In this mode effective address is obtained by adding displacement to PC.

- EA= PC + Address field value

  PC= PC + Relative value.

- **Base register addressing mode:**Base register addressing mode is used to implement inter segment transfer of control.In this mode effective address is obtained by adding base register value to address field value.

- EA= Base register + Address field value.

  PC= Base register + Relative value.

Instruction formats

1. Instruction format describes the internal structures (layout design) of the bits of an instruction, in terms of its constituent parts.

2. An Instruction format must include an opcode, and address is dependent on an availability of particular operands.

3. The format can be implicit or explicit which will indicate the addressing mode for each operand.

4. Designing of an Instruction format is very complex. As we know a computer uses a variety of instructional. There are many designing issues which affect the instructional design, some of them are given are below:

    - **Instruction length**: It is a most basic issue of the format design. A longer will be the instruction it means more time is needed to fetch the instruction.

    - **Memory size**: If larger memory range is to be addressed then more bits will be required in the address field.

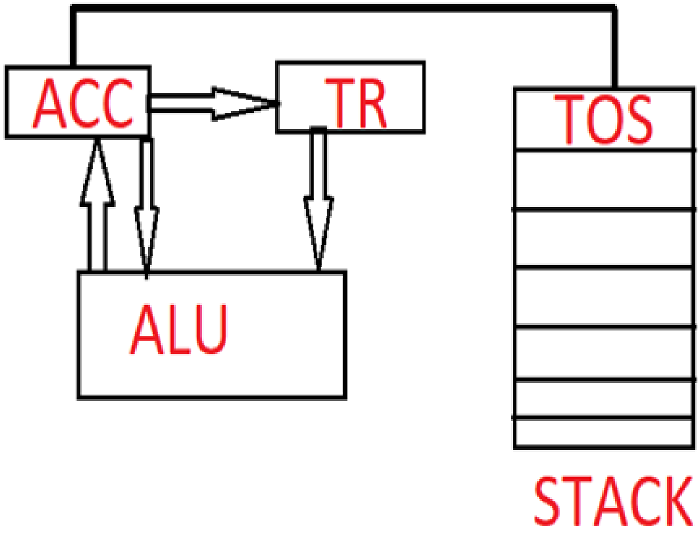
    - **Memory organization**: If the system supports the virtual memory then memory range which needs to be addressed by the instruction, is larger than the physical memory.

    - **Memory transfer length**: Instruction length should be equal to the data bus length or it should be multiple of it.

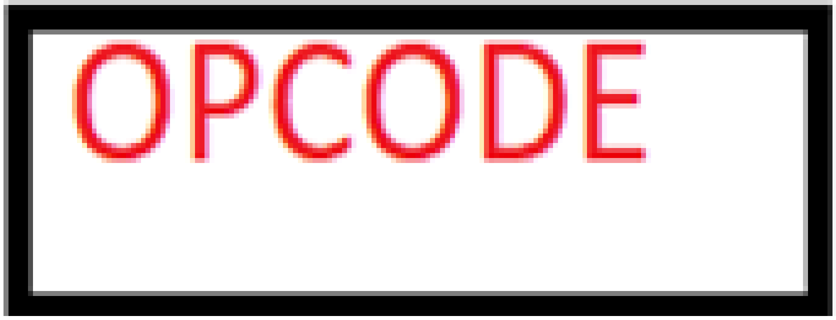
5. Instruction formats are classified into 5 types based on the type of the CPU organization. CPU organization is divided into three types based on the availability of the ALU operands, which are as follows here:

1) STACK CPU

In this organization, ALU operands are performed only on a stack data. This means that both of the ALU operations are always required in the stack. The same stack is also used as the destination. In the stack, we can perform insert and deletion operation at only one end which is called as the top of a stack. So in this format, there is no need of address because in this TOS becomes the default location.

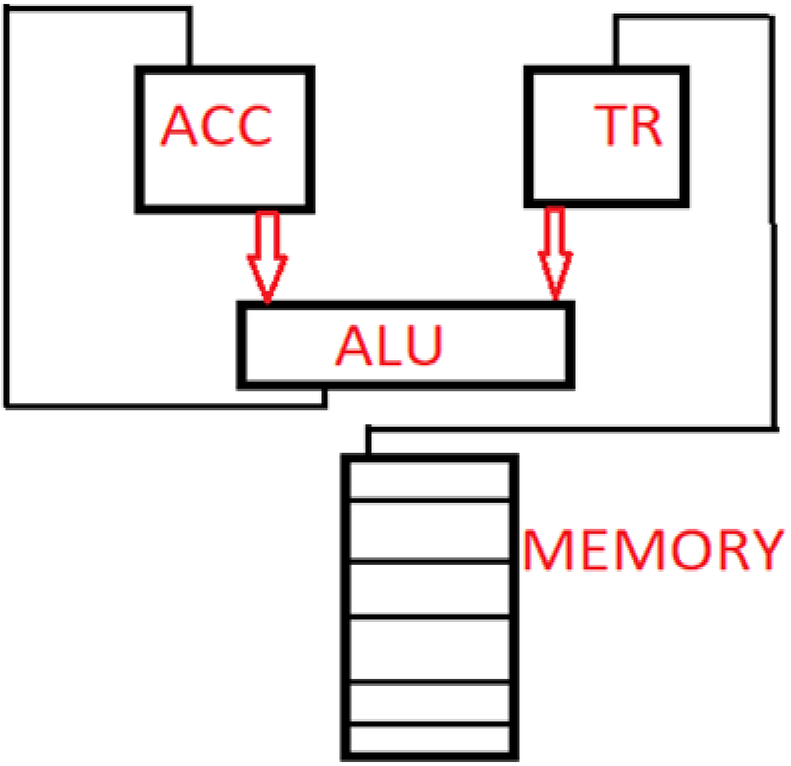


In this organization, only the ALU operands are zero address operation whereas data transfer instructions are not a zero address instruction. The computable instruction format of STACK CPU is Zero Address Instruction Format.

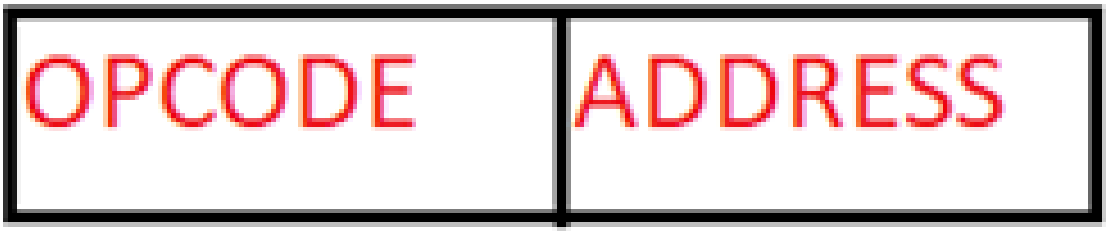


2) Accumulator CPU

In this organization, one of the ALU operands is always present in the accumulator. The same accumulator is also used as the destination. Another ALU operand is present either in the register or in memory. In processor design, only one accumulator is present so it becomes the default location.



The computable instruction format of Accumulator CPU is **One Address Instruction Format**.



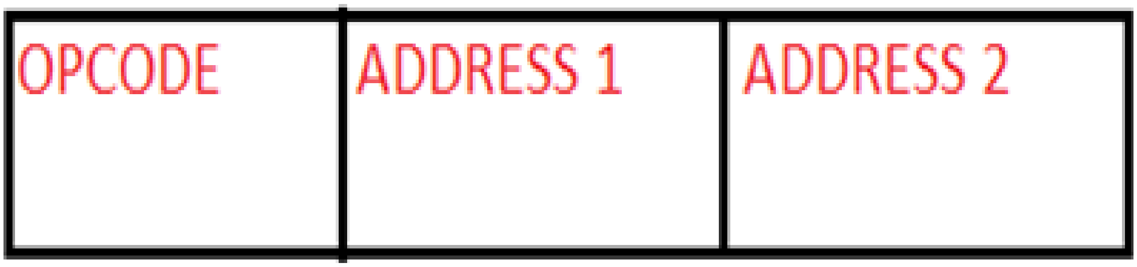
3) General Register CPU

Based on the number of the registers possible in the processors, the architecture is divided into two types:

i. Register-Memory references CPU
ii. Register-Register references CPU

i) Register-Memory Reference CPU

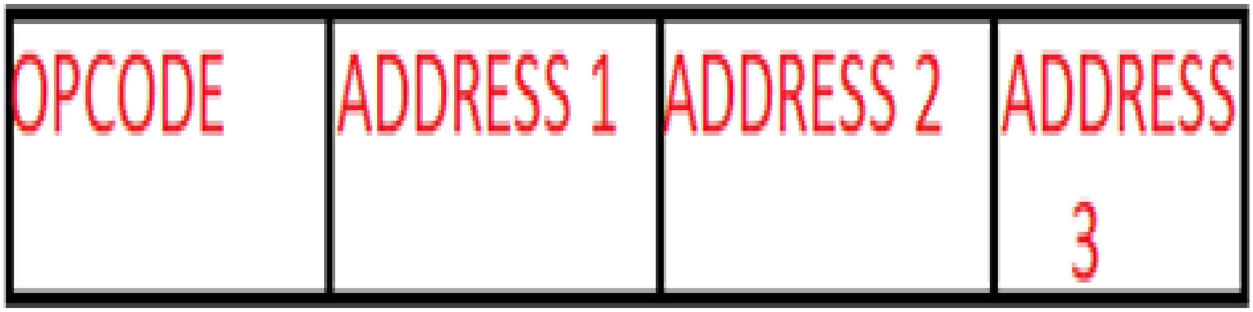
In this architecture, processors support less number of registers. Therefore register file size is small. In this organization, the first ALU operand is always required in the register. The same register can also be used as the destination. The second ALU operand is present either in a register or in memory. The computable instruction format of the register to memory reference CPU is **Two Address Instruction Format**.



OPCODE	ADDRESS 1	ADDRESS 2

ii) Register-Register Reference CPU

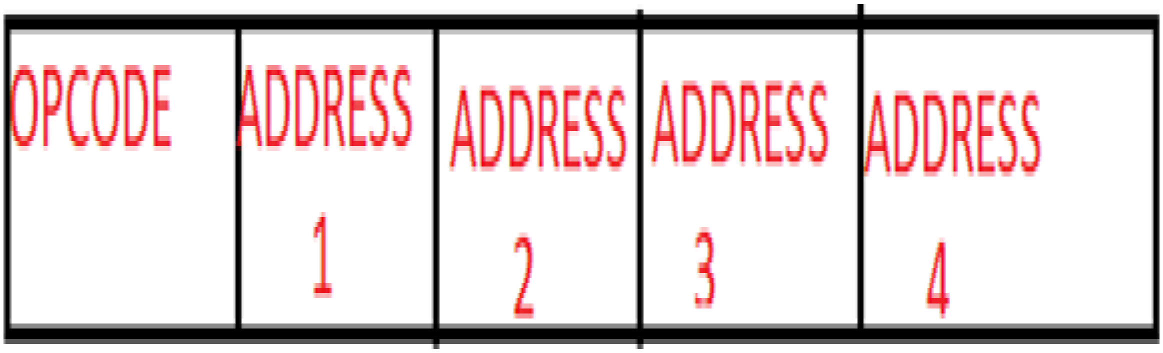
In this architecture, processors support number of registers, therefore, register file size is large. In this organization, ALU operands are performed only on a registers data that means both of the ALU operands are required in the register. Due to more number of register present in the CPU, the separate register is used to store the result. The computable instruction format of Register-Register Reference CPU is Three Address Instruction Format.



OPCODE	ADDRESS 1	ADDRESS 2	ADDRESS 3

Four Address instruction format

This format contains the 4 different address fields with an opcode. Since PC is used as the mandatory register in the CPU design which is used to hold the next instruction address. So four instruction format is not in the use.



OPCODE	ADDRESS 1	ADDRESS 2	ADDRESS 3	ADDRESS 4

data transfer & Manipulation

Data Transfer Instructions: Data transfer instructions cause transfer of data from one location to another without changing the information content.
The common transfers may be between memory and processor registers, between processor registers and input/output.

Data Manipulation Instructions:

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. There are three types of data manipulation instructions: Arithmetic instructions, Logical and bit manipulation instructions, and Shift instructions.

Program Control Instructions

Program control instructions specify conditions for altering the content of the program counter, while data transfer and manipulation instructions specify conditions for data processing operations. The change in value of a program counter as a result of the execution of a program control instruction causes a break in the sequence of instruction execution.

I/O Organization

The method that is used to transfer information between internal storage and external I/O devices is known as I/O interface. The CPU is interfaced using special communication links by the peripherals connected to any computer system. These communication links are used to resolve the differences between CPU and peripheral. There exists special hardware components between CPU and peripherals to supervise and synchronize all the input and output transfers that are called interface units.

Mode of Transfer:

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways

1. Programmed I/O.

2. Interrupt- initiated I/O.

3. Direct memory access(DMA).

Now let’s discuss each mode one by one.

1. **Programmed I/O:** It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.

   Example of Programmed I/O: In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.

2. **Interrupt- initiated I/O:** Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

Note: Both the methods programmed I/O and Interrupt-driven I/O require the active intervention of the processor to transfer data between memory and the I/O module, and any data transfer must transverse a path through the processor. Thus both these forms of I/O suffer from two inherent drawbacks.

- The I/O transfer rate is limited by the speed with which the processor can test and service a device.
- The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.

3. **Direct Memory Access**: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

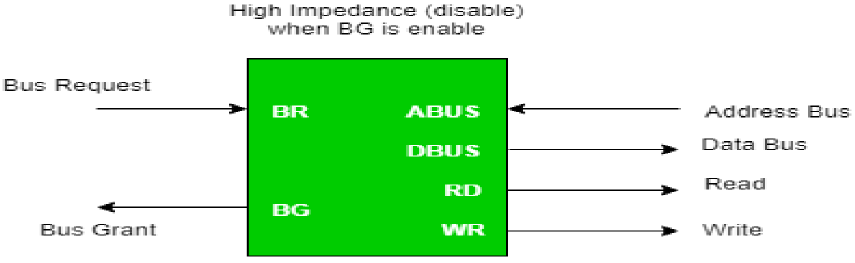


Figure - CPU Bus Signals for DMA Transfer

Bus Request : It is used by the DMA controller to request the CPU to relinquish the control of the buses.

Bus Grant : It is activated by the CPU to Inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.

Types of DMA transfer using DMA controller:

Burst Transfer :

DMA returns the bus after complete data transfer. A register is used as a byte count, being decremented for each byte transfer, and upon the byte count reaching zero, the

DMAC will

release the bus. When the DMAC operates in burst mode, the CPU is halted for the duration of the data transfer.

Steps involved are:

1. Bus grant request time.

2. Transfer the entire block of data at transfer rate of device because the device is usually slow than thespeed at which the data can be transferred to CPU.

3. Release the control of the bus back to CPU So, total time taken to transfer the N bytes

= Bus grant request time + (N) * (memory transfer rate) + Bus release control time.

Where,

X µsec =data transfer time or preparation time (words/block)

Y µsec =memory cycle time or cycle time or transfer time (words/block)

% CPU idle (Blocked)=(Y/X+Y)*100

% CPU Busy=(X/X+Y)*100

Cyclic Stealing :

An alternative method in which DMA controller transfers one word at a time after which it must return the control of the buses to the CPU. The CPU delays its operation only for one memory cycle to allow the direct memory I/O transfer to “steal” one memory cycle. Steps Involved are:

4. Buffer the byte into the buffer

5. Inform the CPU that the device has 1 byte to transfer (i.e. bus grant request)

6. Transfer the byte (at system bus speed)

7. Release the control of the bus back to CPU.

Before moving on transfer next byte of data, device performs step 1 again so that bus isn’t tied up and the transfer won’t depend upon the transfer rate of device. So, for 1 byte of transfer of data, time taken by using cycle stealing mode (T). = time required for bus grant + 1 bus cycle to transfer data + time required to release the bus, it will be N x T

In cycle stealing mode we always follow pipelining concept that when one byte is getting transferred then Device is parallel preparing the next byte. “The fraction of CPU time to the data transfer time” if asked then cycle stealing mode is used.

Where,

X µsec =data transfer time or preparation time

(words/block)

Y µsec =memory cycle time or cycle time or transfer

time (words/block)

% CPU idle (Blocked) =(Y/X)*100

% CPU busy=(X/Y)*100

Bus Architecture

Summary of functions of buses in computers

1. **Data sharing** - All types of buses found in a computer transfer data between the computer peripherals connected to it.

The buses transfer or send data either in the serial or parallel method of data transfer. This allows for the exchange of 1, 2, 4 or even 8 bytes of data at a time. (A byte is a group of 8 bits). Buses are classified depending on how many bits they can move at the same time, which means that we have 8-bit, 16-bit, 32-bit or even 64-bit buses.

2. Addressing - A bus has address lines, which match those of the processor. This allows data to be sent to or from specific memory locations.

3. Power - A bus supplies power to various peripherals connected to it.

4. Timing - The bus provides a **system clock** signal to synchronize the peripherals attached to it with the rest of the system.

The expansion bus facilitates easy connection of more or additional components and devices on a computer such as a TV card or sound card.

Bus Terminologies

Computers have two major types of buses:

1. System bus:- This is the bus that connects the CPU to the main memory on the motherboard. The system bus is also called the front-side bus, memory bus, local bus, or host bus.

2. A number of I/O Buses, (I/O is an acronym for input/output), connecting various peripheral devices to the CPU. These devices connect to the system bus via a ‘bridge’ implemented in the processors' chipset. Other names for the I/O bus include “expansion bus", "external bus” or “host bus”.

Expansion Bus Types

These are some of the common expansion bus types that have ever been used in computers:

- **ISA** - Industry Standard Architecture
- **EISA** - Extended Industry Standard Architecture
- **MCA** - Micro Channel Architecture
- **VESA** - Video Electronics Standards Association
- **PCI** - Peripheral Component Interconnect
- PCI Express (PCI-X)
- **PCMCIA** - Personal Computer Memory Card Industry Association (Also called PC bus)
- **AGP** - Accelerated Graphics Port
- **SCSI** - Small Computer Systems Interface.

ISA Bus

This is the most common type of early expansion bus, which was designed for use in the original IBM PC. The IBM PC-XT used an 8-bit bus design. This means that the data transfers take place in 8-bit chunks (i.e. one byte at a time) across the bus. The ISA bus ran at a clock speed of 4.77 MHz.

For the 80286-based IBM PC-AT, an improved bus design, which could transfer 16-bits of data at a time, was announced. The 16-bit version of the ISA bus is sometimes known as the AT bus. (AT-Advanced Technology)

The improved AT bus also provided a total of 24 address lines, which allowed 16MB of memory to be addressed. The AT bus was backward compatible with its 8-bit predecessor and allowed 8-bit cards to be used in 16-bit expansion slots.

When it first appeared the 8-bit ISA bus ran at a speed of 4.77MHZ – the same speed as the processor. Improvements done over the years eventually made the AT bus ran at a clock speed of 8MHz.

Comparison Between 8 and 16 Bit ISA Bus

8-Bit ISA card (XT-Bus)	16-Bit ISA (AT –Bus card)
8-bit data interface	16-bit data interface
4.77 MHZ bus	8-MHZ bus
62-pin connector	62-pin connector
	36-pin AT extension connection

MCA (Micro Channel Architecture)

IBM developed this bus as a replacement for ISA when they designed the PS/2 PC launched in 1987.

The bus offered a number of technical improvements over the ISA bus. For instance, the MCA ran at a faster speed of 10MHz and supported either 16-bit or 32-bit data. It also supported bus mastering - a technology that placed a mini-processor on each expansion card. These mini-processors controlled much of the data transfer allowing the CPU to do other tasks.

One advantage of MCA was that the plug-in cards were software configurable; this means that they required minimal intervention by the user when configuring.

The MCA expansion bus did not support ISA cards and IBM decided to charge other manufacturers royalties for use of the technology. This made it unpopular and it is now obsolete technology.

EISA (Extended Industry Standard Architecture)

This is a bus technology developed by a group of manufactures as an alternative to MCA. The bus architecture was designed to use a 32-bit data path and provided 32 address lines giving access to 4GB of memory.

Like the MCA, EISA offered a disk-based setup for the cards, but it still ran at 8MHz in order for it to be compatible with ISA.

The EISA expansion slots are twice as deep as an ISA slot. If an ISA card is placed in an EISA slot it will use only the top row of connectors, however, a full EISA card uses both rows. It offered bus mastering.

EISA cards were relatively expensive and were normally found on high-end workstations and network servers.

VESA Bus

It was also known as the Local bus or the VESA-Local bus. VESA (**Video Electronics Standards Association**) was invented to help standardize PCs video specifications, thus solving the problem of proprietary technology where different manufacturers were attempting to develop their own buses.

The VL Bus provided 32-bit data path and ran at 25 or 33 MHZ. It ran at the same clock frequency as the host CPU. But this became a problem as processor speeds increased because, the faster the peripherals are required to run, the more expensive they are to manufacture.

It was difficult to implement the VL-Bus on newer chips such as the 486s and the new Pentiums and so eventually the VL-Bus was superseded by PCI.

VESA slots had an extra set of connectors and thus the cards were larger. The VESA design was backward compatible with the older ISA cards.

Features of the VESA local bus card:-

- 32-bit interface
- 62/36-pin connector
- 90+20 pin VESA local bus extension

Peripheral Component Interconnect

Peripheral Component Interconnect (PCI) is one of the latest developments in bus architecture and is the current standard for PC expansion cards. Intel developed and launched it as the expansion bus for the Pentium processor in 1993. It is a local bus like VESA, that is, it connects the CPU, memory, and peripherals to a wider, faster data pathway.

PCI supports both 32-bit and 64-bit data width; it is compatible with 486s and Pentiums. The bus data width is equal to the processor, such as a 32-bit processor would have a 32 bit PCI bus, and operates at 33MHz.

PCI was used in developing Plug and Play (PnP) and all PCI cards support PnP. This means a user can plug a new card into the computer, power it on and it will "self-identify" and "self-specify" and start working without manual configuration using jumpers.

Unlike VESA, PCI supports bus mastering that is, the bus has some processing capability and thus the CPU spends less time processing data. Most PCI cards are designed for 5v, but there are also 3v and dual-voltage cards. Keying slots used help to differentiate 3v and 5v cards and also to make sure that a 3v card is not slotted into a 5v socket and vice versa.

Programming Registers

Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU. The registers used by the CPU are often termed as Processor registers.

A processor register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).

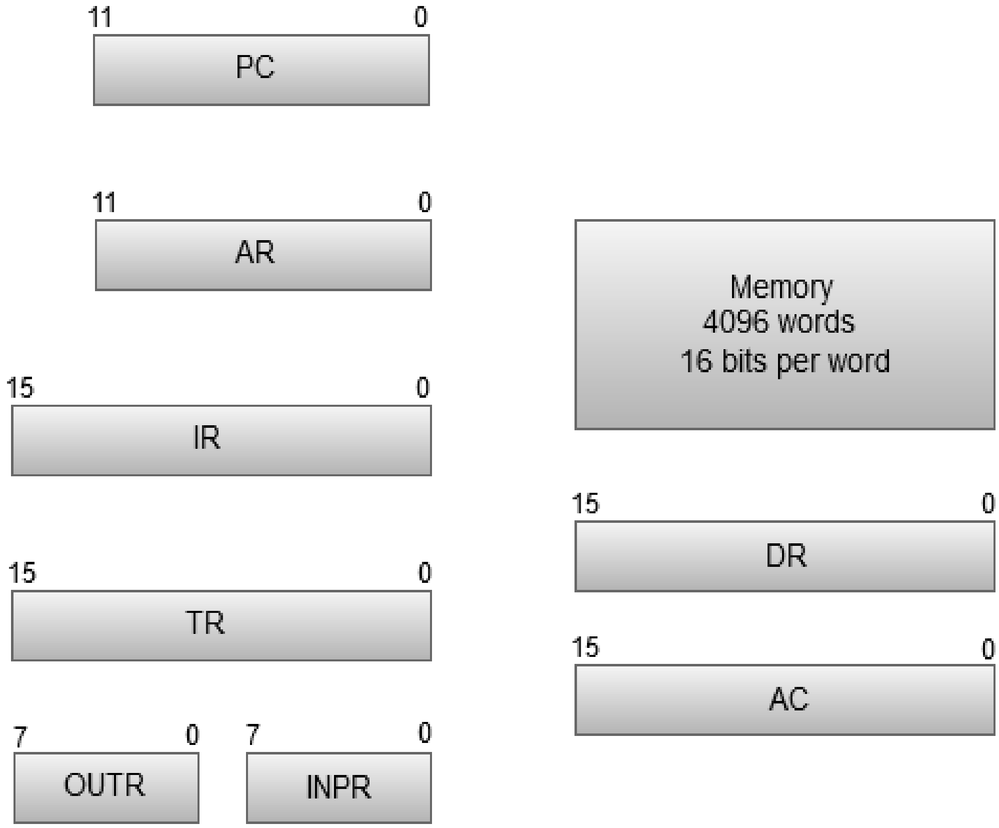
The computer needs processor registers for manipulating data and a register for holding a memory address. The register holding the memory location is used to calculate the address of the next instruction after the execution of the current instruction is completed.

Following is the list of some of the most common registers used in a basic computer:

Register	Symbol	Number of bits	Function
Data register	DR	16	Holds memory operand
Address register	AR	12	Holds address for the memory
Accumulator	AC	16	Processor register
Instruction register	IR	16	Holds instruction code
Program counter	PC	12	Holds address of the instruction
Temporary register	TR	16	Holds temporary data
Input register	INPR	8	Carries input character
Output register	OUTR	8	Carries output character

The following image shows the register and memory configuration for a basic computer.

Register and Memory Configuration of a basic computer:



- The Memory unit has a capacity of 4096 words, and each word contains 16 bits.
- The Data Register (DR) contains 16 bits which hold the operand read from the memory location.
- The Memory Address Register (MAR) contains 12 bits which hold the address for the memory location.
- The Program Counter (PC) also contains 12 bits which hold the address of the next instruction to be read from memory after the current instruction is executed.
- The Accumulator (AC) register is a general purpose processing register.
- The instruction read from memory is placed in the Instruction register (IR).
- The Temporary Register (TR) is used for holding the temporary data during the processing.
- The Input Registers (IR) holds the input characters given by the user.
- The Output Registers (OR) holds the output after processing the input data.

Unit-III

(Memory Organization) :-

Memory Hierarchy

A memory unit is an essential component in any digital computer since it is needed for storing programs and data.

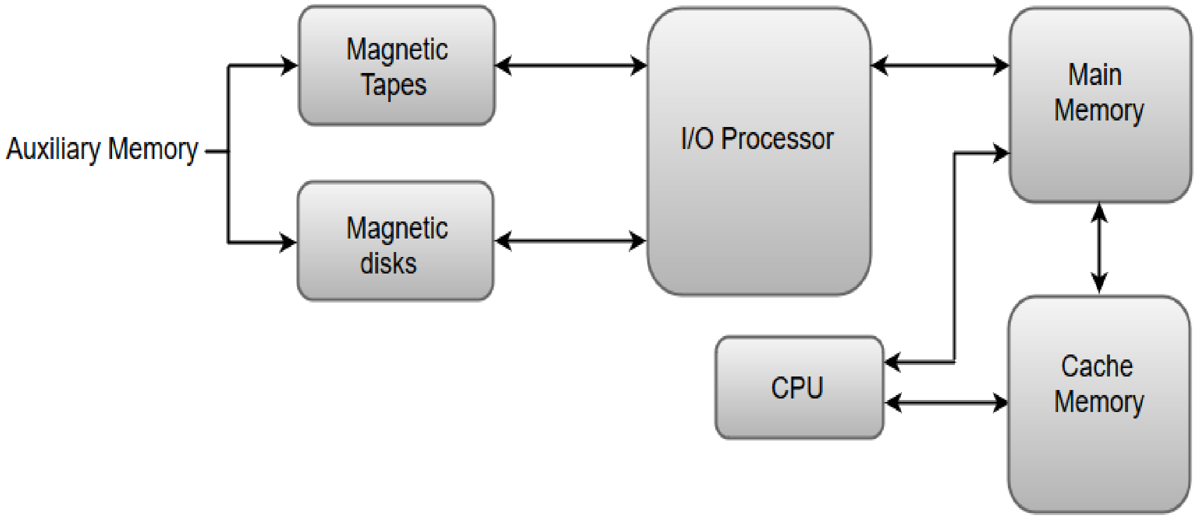
Typically, a memory unit can be classified into two categories:

1. The memory unit that establishes direct communication with the CPU is called Main Memory. The main memory is often referred to as RAM (Random Access Memory).
2. The memory units that provide backup storage are called Auxiliary Memory. For instance, magnetic disks and magnetic tapes are the most commonly used auxiliary memories.

Apart from the basic classifications of a memory unit, the memory hierarchy consists all of the storage devices available in a computer system ranging from the slow but high-capacity auxiliary memory to relatively faster main memory.

The following image illustrates the components in a typical memory hierarchy.

Memory Hierarchy in a Computer System:



Auxiliary Memory

Auxiliary memory is known as the lowest-cost, highest-capacity and slowest-access storage in a computer system. Auxiliary memory provides storage for programs and data that are kept for long-term storage or when not in immediate use. The most common examples of auxiliary memories are magnetic tapes and magnetic disks.

A magnetic disk is a digital computer memory that uses a magnetization process to write, rewrite and access data. For example, hard drives, zip disks, and floppy disks.

Magnetic tape is a storage medium that allows for data archiving, collection, and backup for different kinds of data.

Main Memory

The main memory in a computer system is often referred to as Random Access Memory (RAM). This memory unit communicates directly with the CPU and with auxiliary memory devices through an I/O processor.

The programs that are not currently required in the main memory are transferred into auxiliary memory to provide space for currently used programs and data.

I/O Processor

The primary function of an I/O Processor is to manage the data transfers between auxiliary memories and the main memory.

Cache Memory

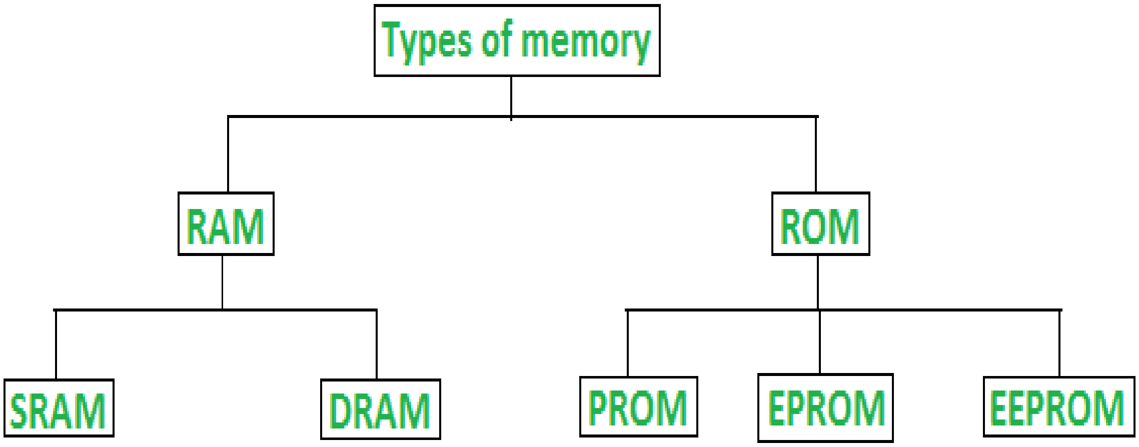
The data or contents of the main memory that are used frequently by CPU are stored in the cache memory so that the processor can easily access that data in a shorter time. Whenever the CPU requires accessing memory, it first checks the required data into the cache memory. If the data is found in the cache memory, it is read from the fast memory. Otherwise, the CPU moves onto the main memory for the required data.

We will discuss each component of the memory hierarchy in more detail later in this chapter.

Main memory (RAM/ROM chips)

Memory is the most essential element of a computing system because without it computer can’t perform simple tasks. Computer memory is of two basic type – Primary memory(RAM and ROM) and Secondary memory(hard drive,CD,etc.). Random Access

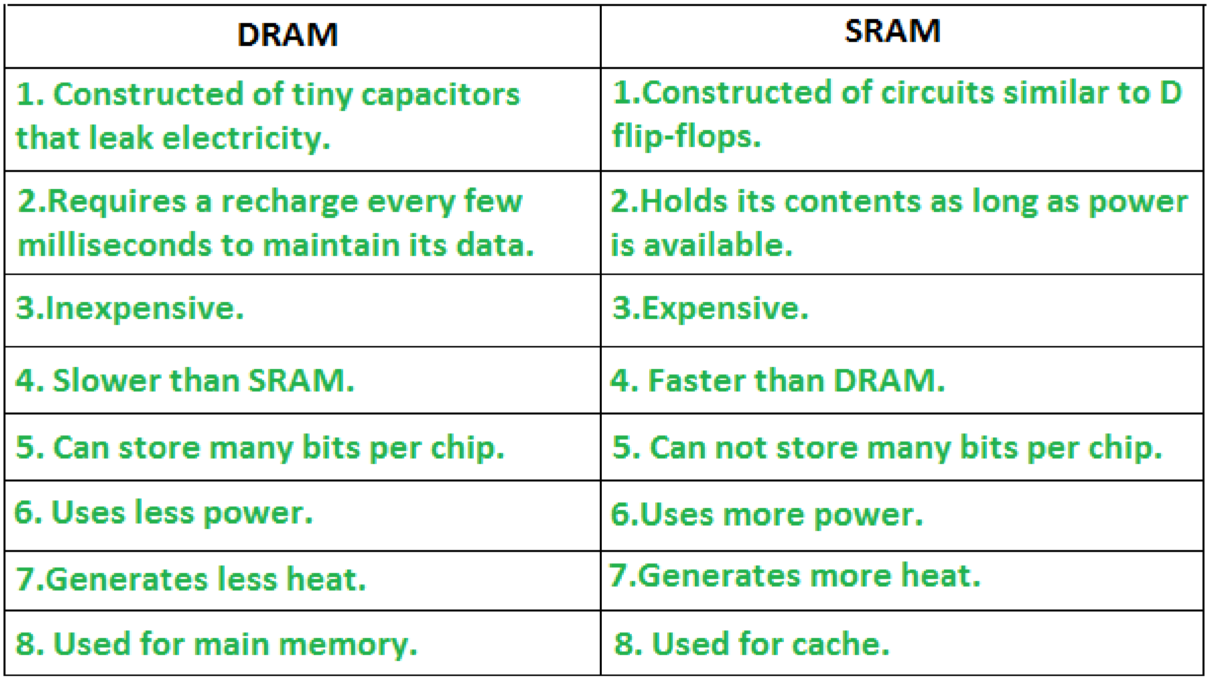
Memory (RAM) is primary-volatile memory and Read Only Memory (ROM) is primary-non-volatile memory.



Classification of computer memory

1. Random Access Memory (RAM) –

- It is also called as read write memory or the main memory or the primary memory.
- The programs and data that the CPU requires during execution of a program are stored in this memory.
- It is a volatile memory as the data loses when the power is turned off.
- RAM is further classified into two types- SRAM (Static Random Access Memory) and DRAM (Dynamic Random Access Memory).



DRAM	SRAM
1. Constructed of tiny capacitors that leak electricity.	1.Constructed of circuits similar to D flip-flops.
2.Requires a recharge every few milliseconds to maintain its data.	2.Holds its contents as long as power is available.
3.Inexpensive.	3.Expensive.
4. Slower than SRAM.	4. Faster than DRAM.
5. Can store many bits per chip.	5. Can not store many bits per chip.
6. Uses less power.	6.Uses more power.
7.Generates less heat.	7.Generates more heat.
8. Used for main memory.	8. Used for cache.

Difference between SRAM and DRAM

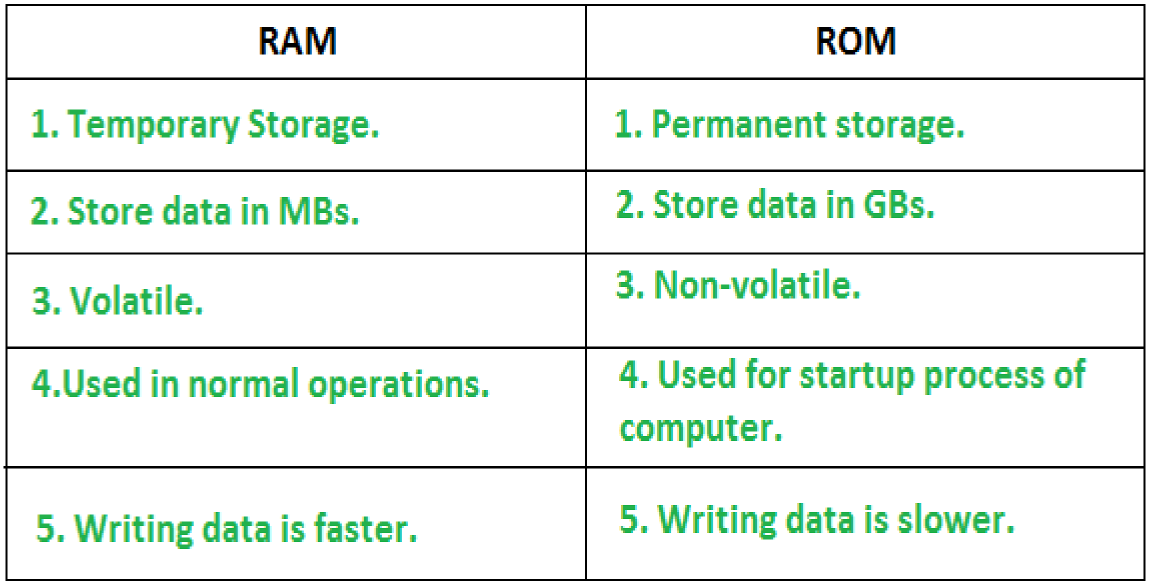
2. Read Only Memory (ROM) –

- Stores crucial information essential to operate the system, like the program essential to boot the computer.
- It is not volatile.
- Always retains its data.
- Used in embedded systems or where the programming needs no change.
- Used in calculators and peripheral devices.
- ROM is further classified into 4 types- ROM, PROM, EPROM, and EEPROM.

Types of Read Only Memory (ROM) –

1. **PROM (Programmable read-only memory)** – It can be programmed by user. Once programmed, the data and instructions in it cannot be changed.
2. **EPROM (Erasable Programmable read only memory)** – It can be reprogrammed. To erase data from it, expose it to ultra violet light. To reprogram it, erase all the previous data.

3. **EEPROM (Electrically erasable programmable read only memory)** – The data can be erased by applying electric field, no need of ultra violet light. We can erase only portions of the chip.



RAM	ROM
1. Temporary Storage.	1. Permanent storage.
2. Store data in MBs.	2. Store data in GBs.
3. Volatile.	3. Non-volatile.
4.Used in normal operations.	4. Used for startup process of computer.
5. Writing data is faster.	5. Writing data is slower.

Difference between RAM and ROM

Auxiliary memory

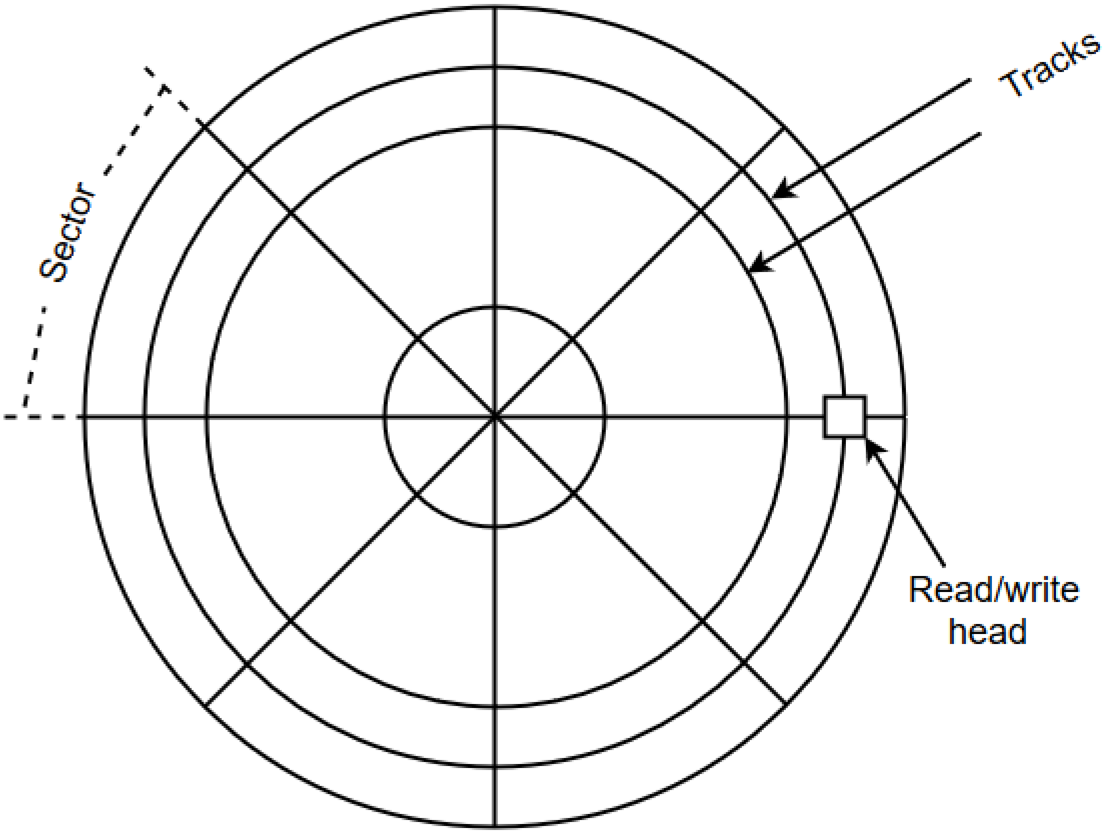
An Auxiliary memory is known as the lowest-cost, highest-capacity and slowest-access storage in a computer system. It is where programs and data are kept for long-term storage or when not in immediate use. The most common examples of auxiliary memories are magnetic tapes and magnetic disks.

Magnetic Disks

A magnetic disk is a type of memory constructed using a circular plate of metal or plastic coated with magnetized materials. Usually, both sides of the disks are used to carry out read/write operations. However, several disks may be stacked on one spindle with read/write head available on each surface.

The following image shows the structural representation for a magnetic disk.

Magnetic disks



- The memory bits are stored in the magnetized surface in spots along the concentric circles called tracks.
- The concentric circles (tracks) are commonly divided into sections called sectors.

Magnetic Tape

Magnetic tape is a storage medium that allows data archiving, collection, and backup for different kinds of data. The magnetic tape is constructed using a plastic strip coated with a magnetic recording medium.

The bits are recorded as magnetic spots on the tape along several tracks. Usually, seven or nine bits are recorded simultaneously to form a character together with a parity bit.

Magnetic tape units can be halted, started to move forward or in reverse, or can be rewound. However, they cannot be started or stopped fast enough between individual characters. For this reason, information is recorded in blocks referred to as records.

Associative memory

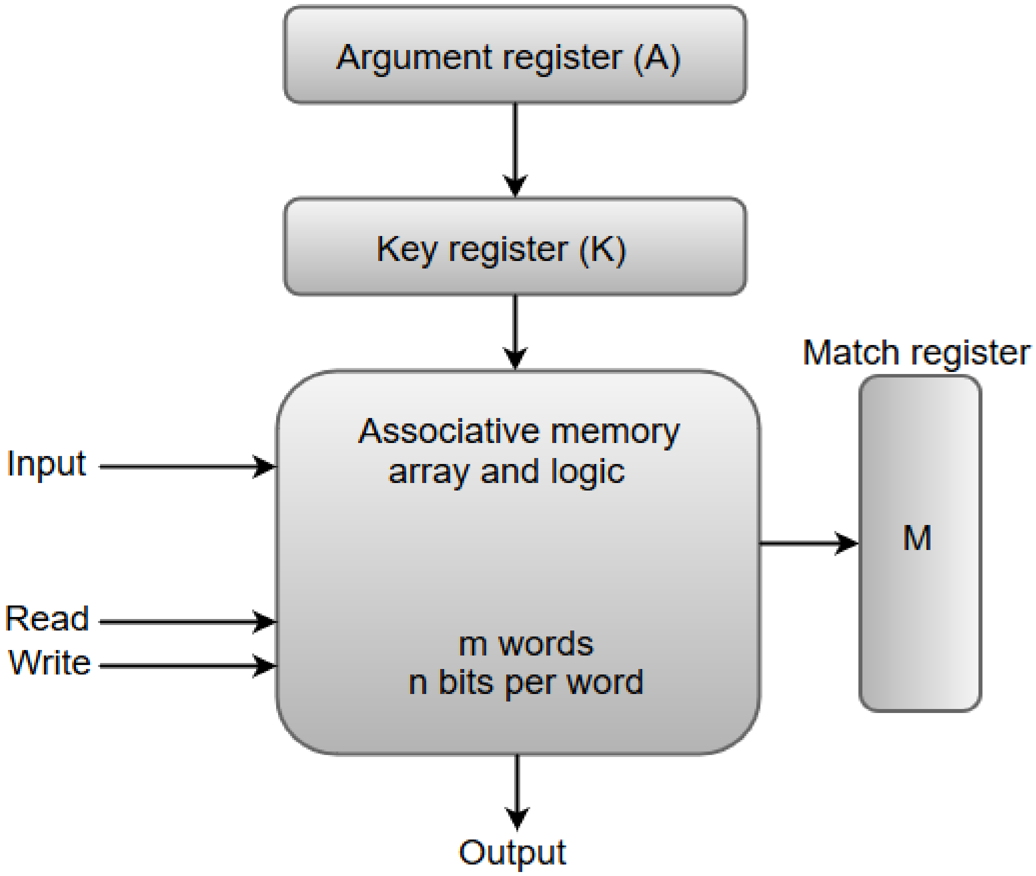
An associative memory can be considered as a memory unit whose stored data can be identified for access by the content of the data itself rather than by an address or memory location.

Associative memory is often referred to as **Content Addressable Memory (CAM)**.

When a write operation is performed on associative memory, no address or memory location is given to the word. The memory itself is capable of finding an empty unused location to store the word.

On the other hand, when the word is to be read from an associative memory, the content of the word, or part of the word, is specified. The words which match the specified content are located by the memory and are marked for reading.

The following diagram shows the block representation of an Associative memory.



From the block diagram, we can say that an associative memory consists of a memory array and logic for 'm' words with 'n' bits per word.

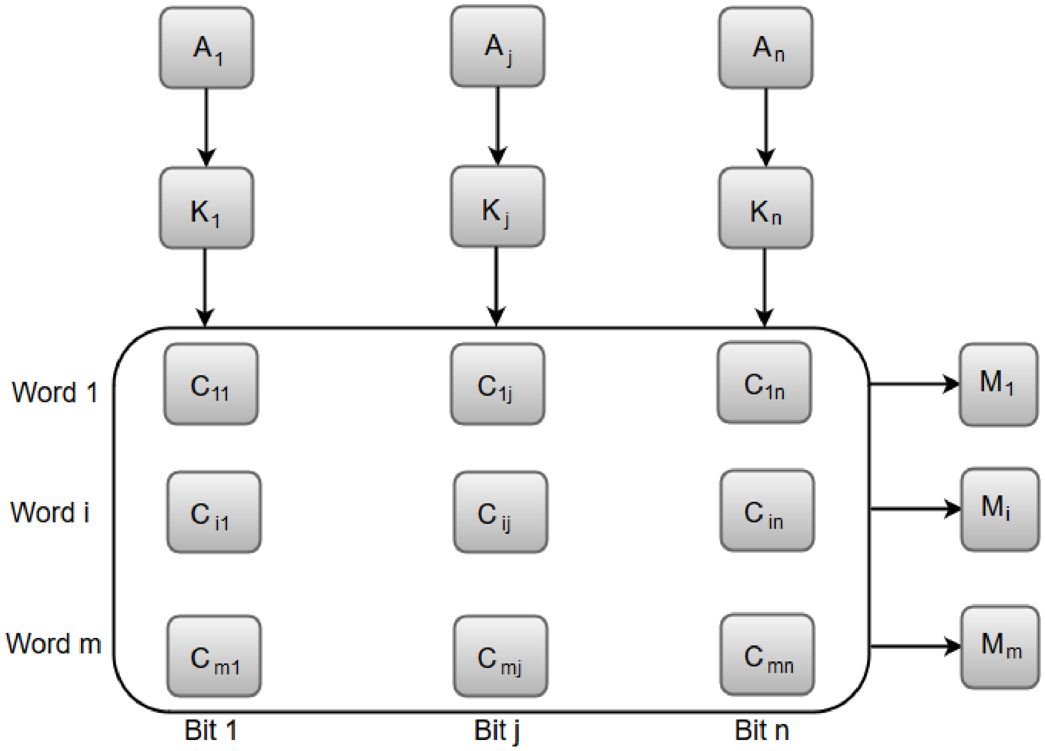
The functional registers like the argument register **A** and key register **K** each have **n** bits, one for each bit of a word. The match register **M** consists of **m** bits, one for each memory word.

The words which are kept in the memory are compared in parallel with the content of the argument register.

The key register (K) provides a mask for choosing a particular field or key in the argument word. If the key register contains a binary value of all 1's, then the entire argument is compared with each memory word. Otherwise, only those bits in the argument that have 1's in their corresponding position of the key register are compared. Thus, the key provides a mask for identifying a piece of information which specifies how the reference to memory is made.

The following diagram can represent the relation between the memory array and the external registers in an associative memory.

Associative memory of m word, n cells per word:



The cells present inside the memory array are marked by the letter C with two subscripts. The first subscript gives the word number and the second specifies the bit position in the word. For instance, the cell C_{ij} is the cell for bit **j** in word **i**.

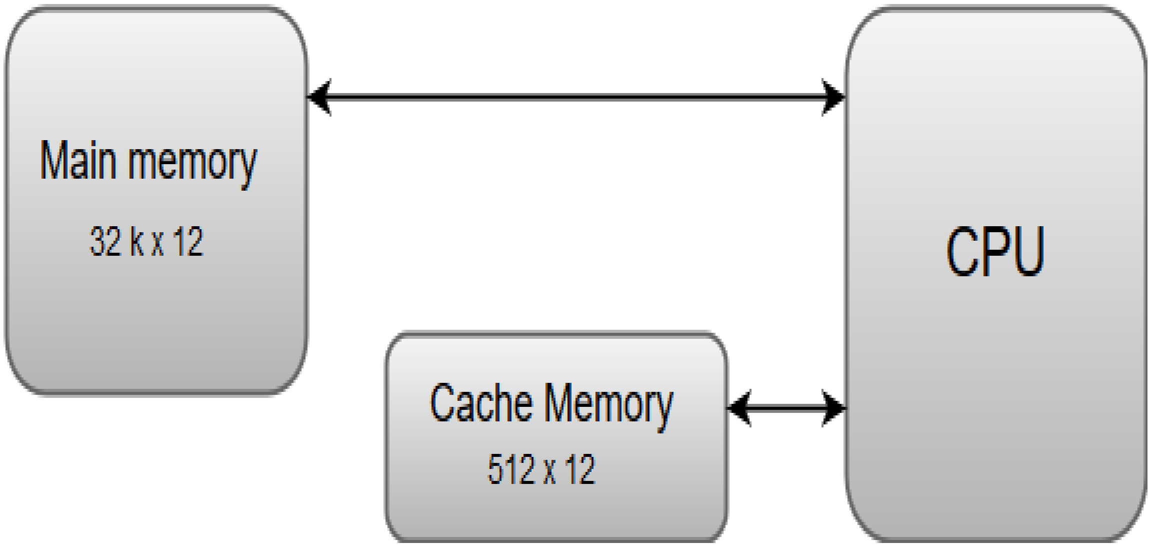
A bit A_j in the argument register is compared with all the bits in column **j** of the array provided that $K_j = 1$. This process is done for all columns **j** = 1, 2, 3......, n.

If a match occurs between all the unmasked bits of the argument and the bits in word **i**, the corresponding bit M_i in the match register is set to 1. If one or more unmasked bits of the argument and the word do not match, M_i is cleared to 0.

Cache memory

The data or contents of the main memory that are used frequently by CPU are stored in the cache memory so that the processor can easily access that data in a 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 into the main memory.

Cache memory is placed between the CPU and the main memory. The block diagram for a cache memory can be represented as:



The cache is the fastest component in the memory hierarchy and approaches the speed of CPU components.

The basic operation of a cache memory is as follows:

- When the CPU needs to access memory, the cache is examined. If the word is found in the cache, it is read from the fast memory.
- If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word.
- A block of words one just accessed is then transferred from main memory to cache memory. The block size may vary from one word (the one just accessed) to about 16 words adjacent to the one just accessed.
- The performance of the cache memory is frequently measured in terms of a quantity called **hit ratio**.
- When the CPU refers to memory and finds the word in cache, it is said to produce a **hit**.
- If the word is not found in the cache, it is in main memory and it counts as a **miss**.
- The ratio of the number of hits divided by the total CPU references to memory (hits plus misses) is the hit ratio.

Virtual Memory

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

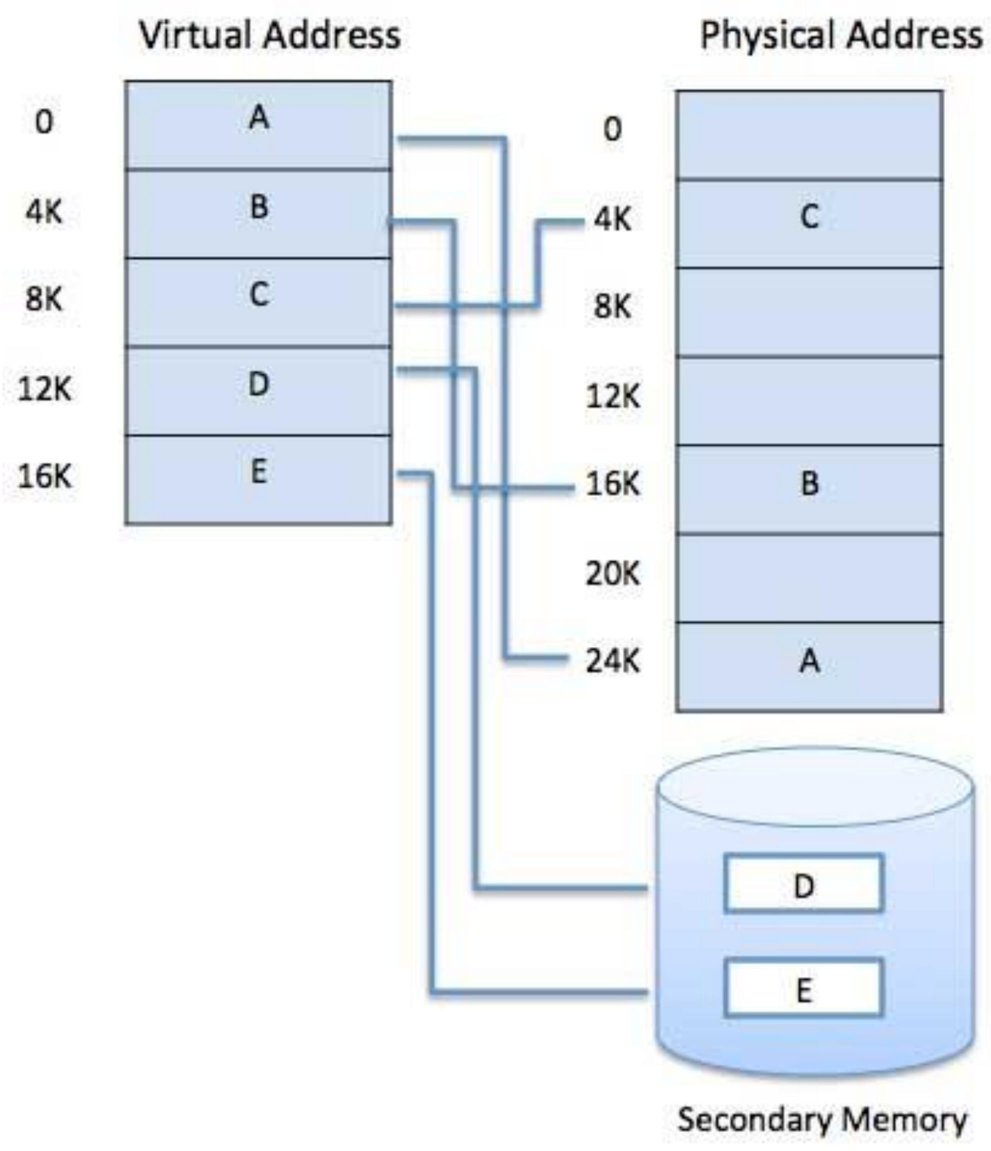
The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.

- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

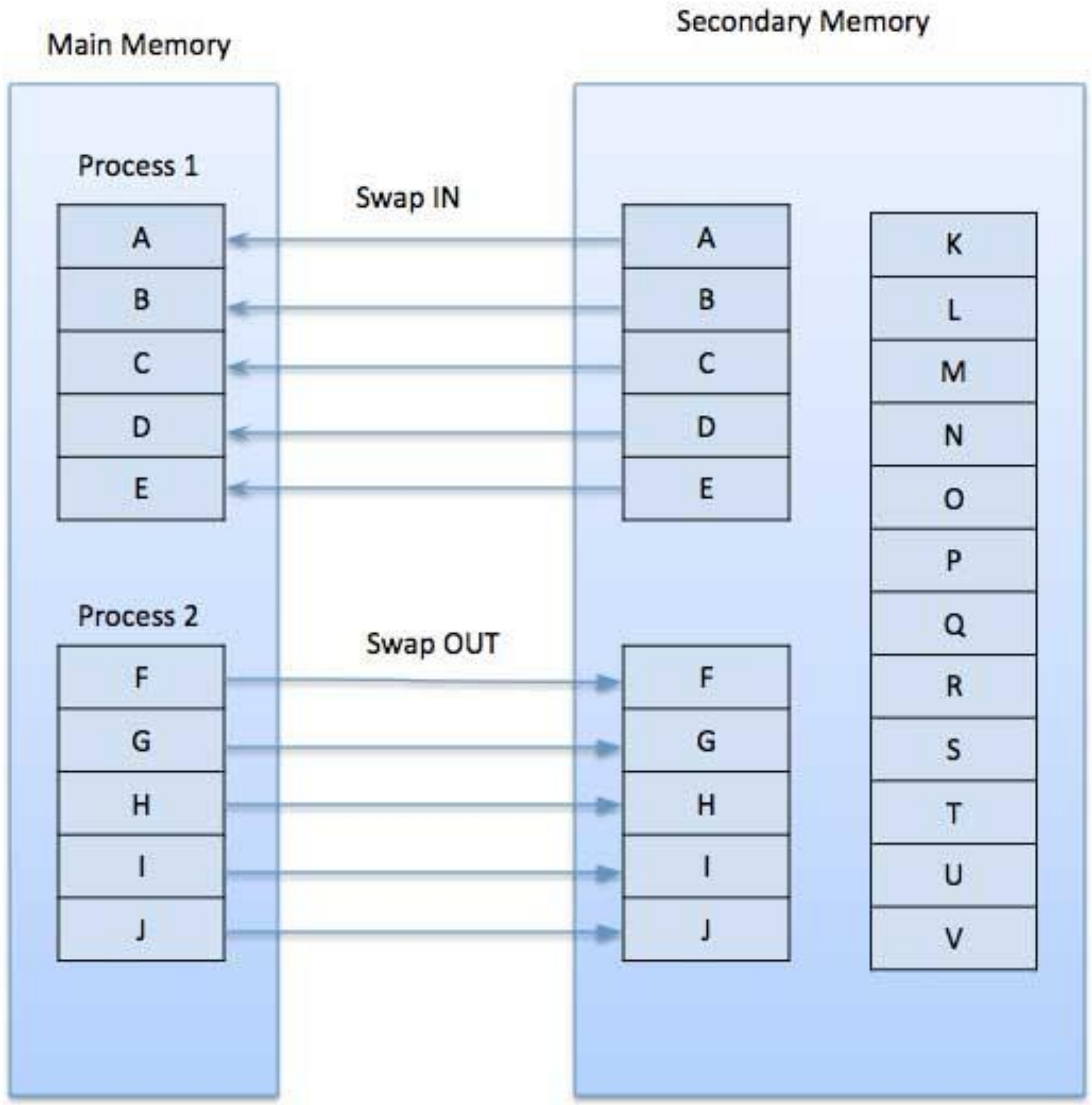
Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below −



Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

Following are the advantages of Demand Paging −

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

Reference String

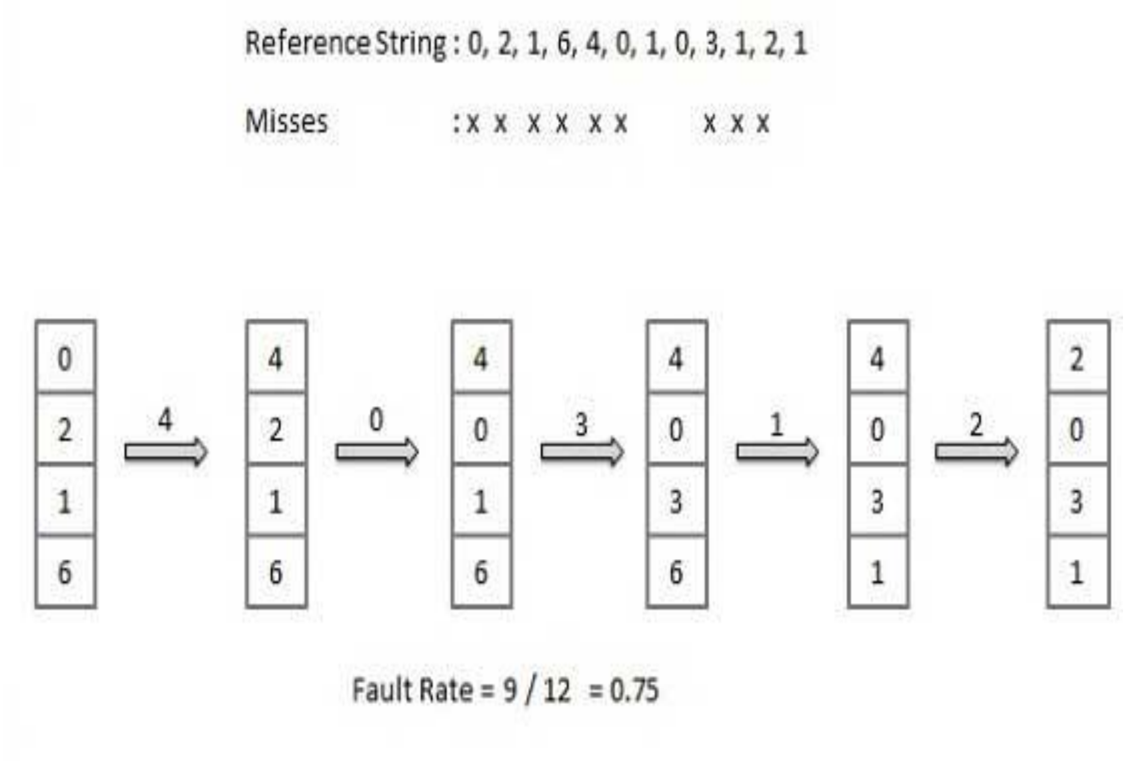
The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things.

- For a given page size, we need to consider only the page number, not the entire address.

- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
- For example, consider the following sequence of addresses − 123,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0

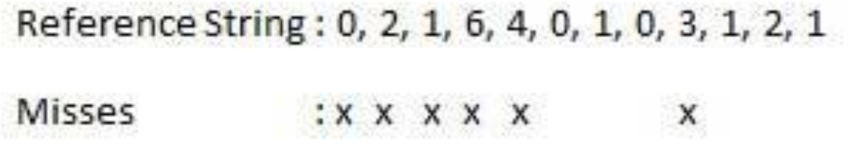
First In First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.


Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1
Misses :x x x x x x x x x
0 2 1 6
4
4 2 1 6
0
4 0 1 6
3
4 0 3 6
1
4 0 3 1
2
2 0 3 1
Fault Rate = 9 / 12 = 0.75

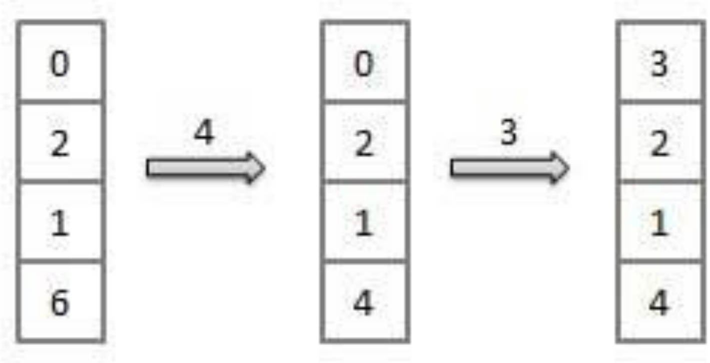
Optimal Page algorithm

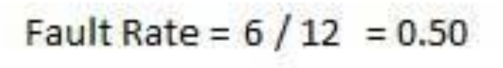
- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.



Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x





Fault Rate = 6 / 12 = 0.50

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Memory Management Hardware

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

This tutorial will teach you basic concepts related to Memory Management.

Process Address Space

The process address space is the set of logical addresses that a process references in its code. For example, when 32-bit addressing is in use, addresses can range from 0 to 0x7fffffff; that is, 2^31 possible numbers, for a total theoretical size of 2 gigabytes.

The operating system takes care of mapping the logical addresses to physical addresses at the time of memory allocation to the program. There are three types of addresses used in a program before and after memory is allocated –

S.N.	Memory Addresses & Description
1	**Symbolic addresses** The addresses used in a source code. The variable names, constants, and instruction labels are the basic elements of the symbolic address space.
2	**Relative addresses** At the time of compilation, a compiler converts symbolic addresses into relative addresses.
3	**Physical addresses** The loader generates these addresses at the time when a program is loaded into main memory.

Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.

The set of all logical addresses generated by a program is referred to as a **logical address space**. The set of all physical addresses corresponding to these logical addresses is referred to as a **physical address space.**

The runtime mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address.

- The value in the base register is added to every address generated by a user process, which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.

- The user program deals with virtual addresses; it never sees the real physical addresses.

Static vs Dynamic Loading

The choice between Static or Dynamic Loading is to be made at the time of computer program being developed. If you have to load your program statically, then at the time of compilation, the complete programs will be compiled and linked without leaving any external program or module dependency. The linker combines the object program with other necessary object modules into an absolute program, which also includes logical addresses.

If you are writing a Dynamically loaded program, then your compiler will compile the program and for all the modules which you want to include dynamically, only references will be provided and rest of the work will be done at the time of execution.

At the time of loading, with **static loading**, the absolute program (and data) is loaded into memory in order for execution to start.

If you are using **dynamic loading**, dynamic routines of the library are stored on a disk in relocatable form and are loaded into memory only when they are needed by the program.

Static vs Dynamic Linking

As explained above, when static linking is used, the linker combines all other modules needed by a program into a single executable program to avoid any runtime dependency.

When dynamic linking is used, it is not required to link the actual module or library with the program, rather a reference to the dynamic module is provided at the time of compilation and linking. Dynamic Link Libraries (DLL) in Windows and Shared Objects in Unix are good examples of dynamic libraries.

Swapping

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction**.

The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take

2048KB / 1024KB per second
= 2 seconds
= 2000 milliseconds

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

Memory Allocation

Main memory usually has two partitions −

- **Low Memory** − Operating system resides in this memory.
- **High Memory** − User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

S.N.	Memory Allocation & Description
1	**Single-partition allocation** In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.
2	**Multiple-partition allocation** In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

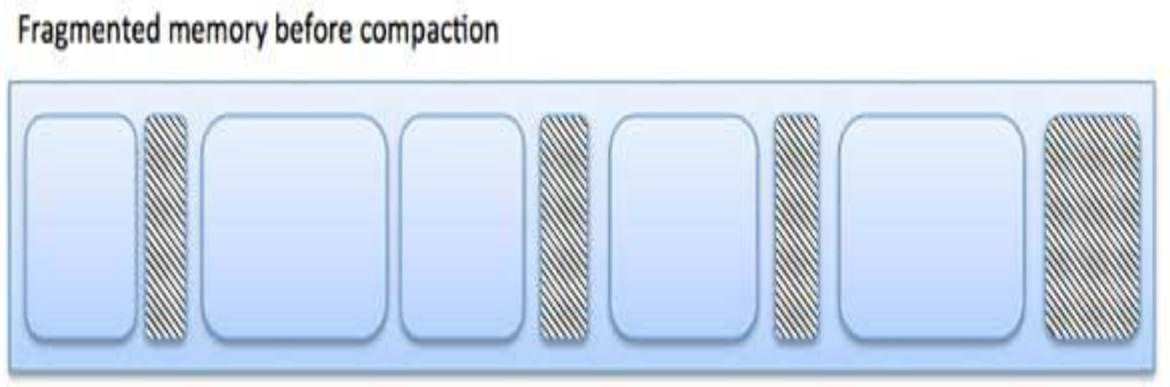
Fragmentation

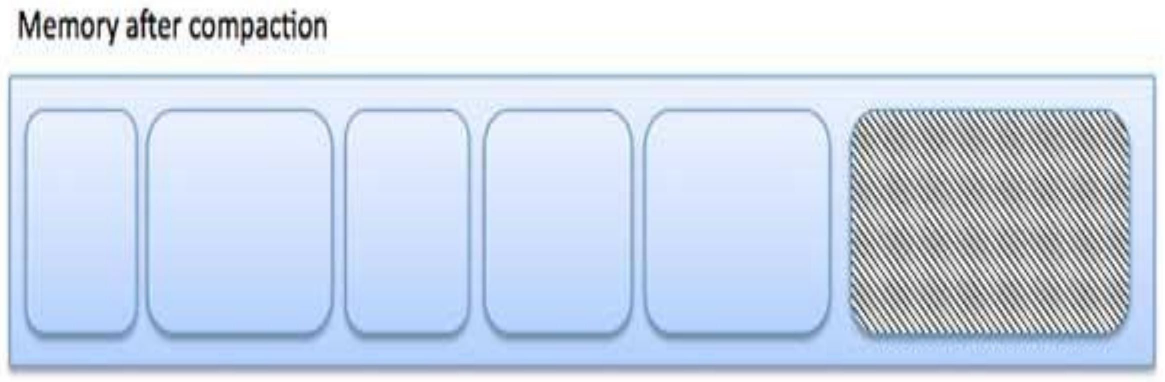
As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types –

S.N.	Fragmentation & Description
1	**External fragmentation** Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
2	**Internal fragmentation** Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory –





External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

Paging

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

Address Translation

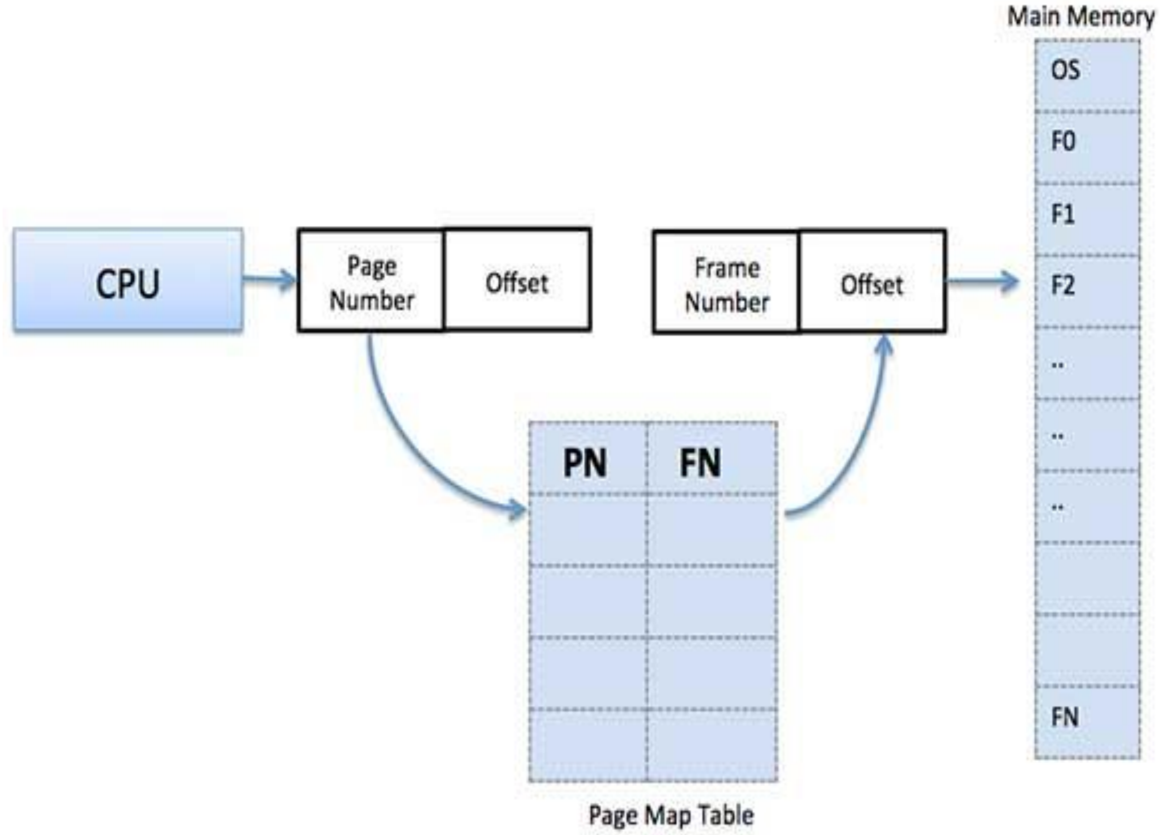
Page address is called **logical address** and represented by **page number** and the **offset**.

Logical Address = Page number + page offset

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

Physical Address = Frame number + page offset

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging −

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

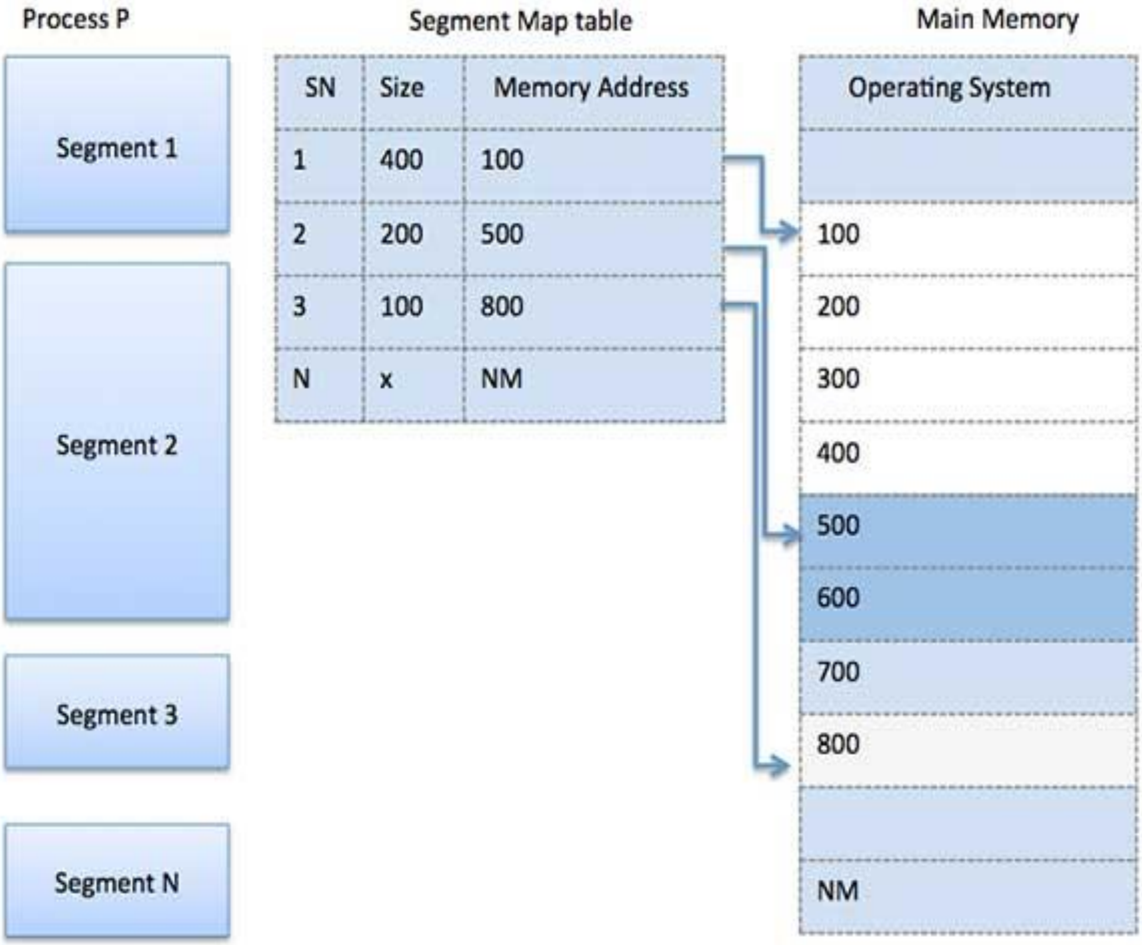
Segmentation

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

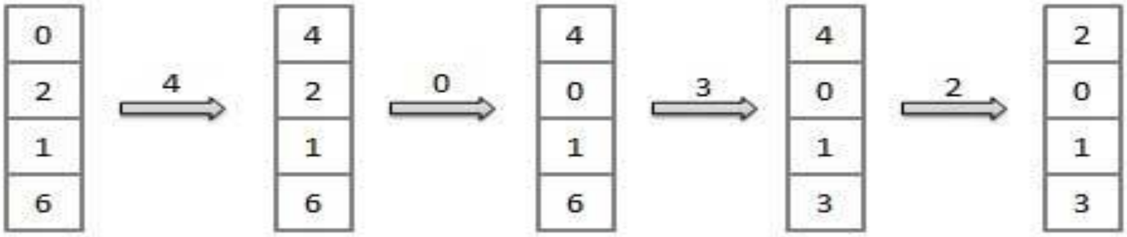
A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.



SN	Size	Memory Address
1	400	100
2	200	500
3	100	800
N	x	NM

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



Fault Rate = 8 / 12 = 0.67

Page Buffering algorithm

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write the new page in the frame of free pool, mark the page table and restart the process.
- Now write the dirty page out of disk and place the frame holding replaced page in free pool.

Least frequently Used(LFU) algorithm

- The page with the smallest count is the one which will be selected for replacement.
- This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

Most frequently Used(MFU) algorithm

- This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

hit/miss ratio

Here are some scripts related to Hit/Miss Ratios .

Buffer Hit Ratio

BUFFER HIT RATIO NOTES:

□ **Consistent Gets** - The number of accesses made to the block buffer to retrieve data in a consistent mode.

□ **DB Blk Gets** - The number of blocks accessed via single block gets (i.e. not through the consistent get mechanism).

□ **Physical Reads** - The cumulative number of blocks read from disk.

□ Logical reads are the sum of **consistent gets** and **db block gets**.

□ The **db block gets** statistic value is incremented when a block is read for update and when segment header blocks are accessed.

□ Hit Ratio should be > 80%, else increase DB_BLOCK_BUFFERS in init.ora

```
select    sum(decode(NAME, 'consistent gets',VALUE, 0)) "Consistent Gets",
          sum(decode(NAME, 'db block gets',VALUE, 0)) "DB Block Gets",
          sum(decode(NAME, 'physical reads',VALUE, 0)) "Physical Reads",
          round((sum(decode(name, 'consistent gets',value, 0)) +
               sum(decode(name, 'db block gets',value, 0)) -
               sum(decode(name, 'physical reads',value, 0))) /
              (sum(decode(name, 'consistent gets',value, 0)) +
               sum(decode(name, 'db block gets',value, 0))) * 100,2) "Hit Ratio"
from   v$sysstat
```

□ **Data Dict Hit Ratio**

DATA DICTIONARY HIT RATIO NOTES:

□ **Gets** - Total number of requests for information on the data object.

□ **Cache Misses** - Number of data requests resulting in cache misses

□ Hit Ratio should be > 90%, else increase SHARED_POOL_SIZE in init.ora

```
select    sum(GETS),

          sum(GETMISSES),

          round((1 - (sum(GETMISSES) / sum(GETS))) * 100,2)

from      v$rowcache
```

□ **SQL Cache Hit Ratio**

SQL CACHE HIT RATIO NOTES:

□ **Pins** - The number of times a pin was requested for objects of this namespace.

□ **Reloads** - Any pin of an object that is not the first pin performed since the object handle was created, and which requires loading the object from disk.

□ Hit Ratio should be > 85%

```
select    sum(PINS) Pins,
          sum(RELOADS) Reloads,

          round((sum(PINS) - sum(RELOADS)) / sum(PINS) * 100,2) Hit_Ratio
from      v$librarycache
```

Library Cache Miss Ratio

LIBRARY CACHE MISS RATIO NOTES:

- **Executions** - The number of times a pin was requested for objects of this namespace.

- **Cache Misses** - Any pin of an object that is not the first pin performed since the object handle was created, and which requires loading the object from disk.

- Hit Ratio should be < 1%, else increase SHARED_POOL_SIZE in init.ora

```
select    sum(PINS) Executions,
          sum(RELOADS) cache_misses,
          sum(RELOADS) / sum(PINS) miss_ratio

from      v$librarycache
```

magnetic disk and its performance

Magnetic Disk is type of secondary memory which is a flat disc covered with magnetic coating to hold information. It is used to store various programs and files. The polarized information in one direction is represented by 1, and vice versa. The direction is indicated by 0.

Magnetic disk are less expensive than RAM and can store large amounts of data, but data access rate is slower than main memory because of secondary memory. Data can be modified or can be deleted easily in the magnetic disk memory. It also allows random access of data.

Figure – Magnetic Disk

These are various advantages and disadvantages of magnetic disk memory.

Advantages :-

These are economical memory

The easy and direct access of data possible.

It can store large amounts of data.

It has better data transfer rate than magnetic tapes.

It has less prone to corruption of data as compared to tapes.

Disadvantages :-

These are less less expensive than RAM but more expensive than magnetic tape memories.

It need clean and dust free environment to store.

These are not suitable for sequential access.

magnetic Tape

Magnetic drums, magnetic tape and magnetic disks are types of magnetic memory. These memories use property for magnetic memory. Here, we have explained about magnetic tape in brief.

Magnetic Tape memory :

In magnetic tape only one side of the ribbon is used for storing data. It is sequential memory which contains thin plastic ribbon to store data and coated by magnetic oxide. Data read/write speed is slower because of sequential access. It is highly reliable which requires magnetic tape drive writing and reading data.

Image from Wikipedia – Magnetic Tape Memory

The width of the ribbon varies from 4mm to 1 Inch and it has storage capacity 100 MB to 200 GB.

Let’s see various advantages and disadvantages of Magnetic Tape memory.

Advantages :

1. These are inexpensive, i.e., low cost memories.
2. It provides backup or archival storage.
3. It can be used for large files.
4. It can be used for copying from disk files.
5. It is a reusable memory.
6. It is compact and easy to store on racks.

Disadvantages :

1. Sequential access is the disadvantage, means it does not allow access randomly or directly.
2. It requires caring to store, i.e., vulnerable humidity, dust free, and suitable environment.
3. It stored data cannot be easily updated or modified, i.e., difficult to make updates on data.

Unit-IV

(I/O Organization) :-

Peripheral devices

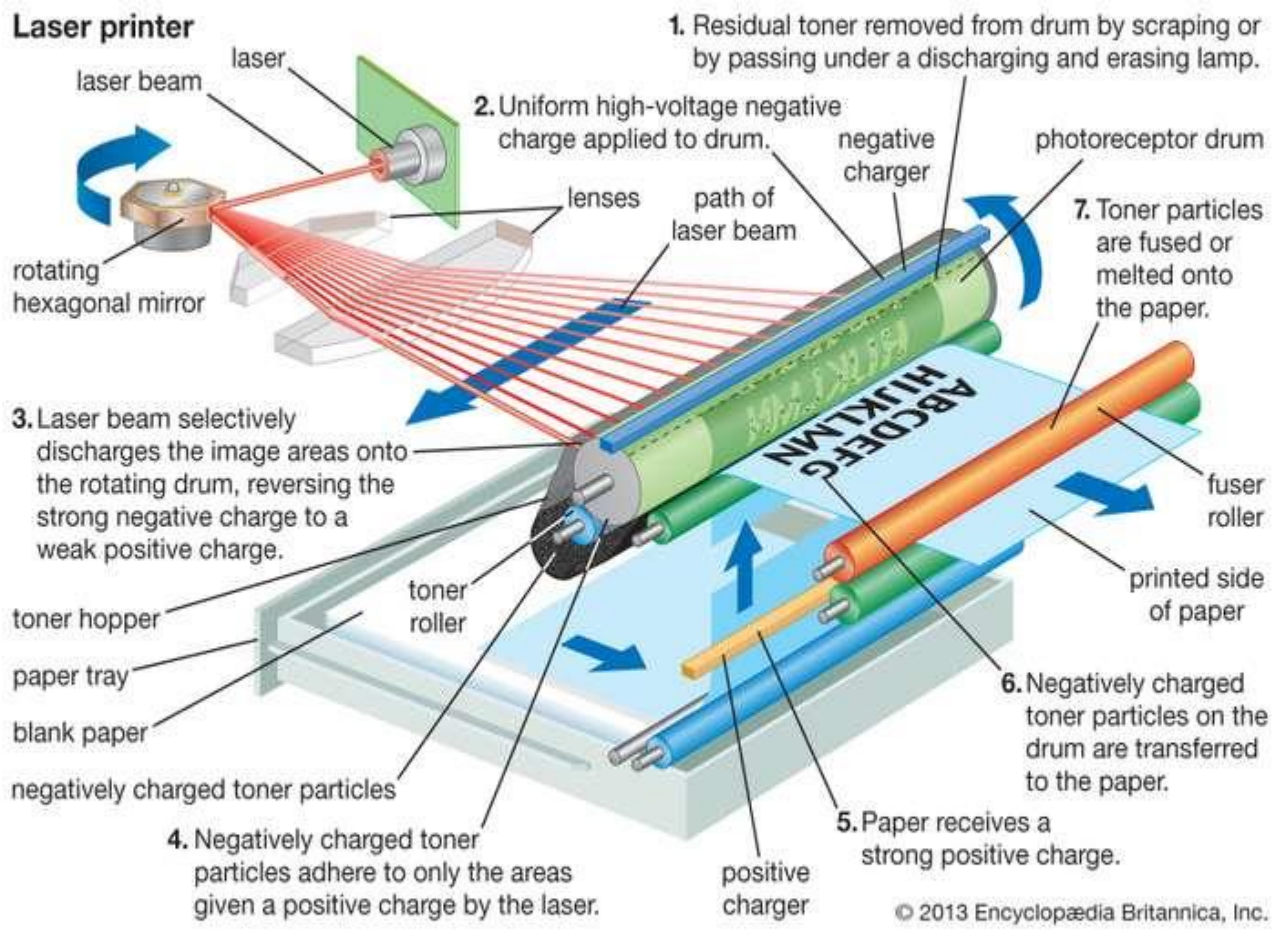
input-output device, or **input/output device**, any of various devices (including sensors) used to enter information and instructions into a computer for storage or processing and to deliver the processed data to a human operator or, in some cases, a machine controlled by the computer. Such devices make up the peripheral equipment of modern digital computer systems.

Peripherals are commonly divided into three kinds: input devices, output devices, and storage devices (which partake of the characteristics of the first two). An input device converts incoming data and instructions into a pattern of electrical signals in binary code that are comprehensible to a digital computer. An output device reverses the process, translating the digitized signals into a form intelligible to the user. At one time punched-card and paper-tape readers were extensively used for inputting, but these have now been supplanted by more efficient devices.

Input devices include typewriter-like keyboards; handheld devices such as the mouse, trackball, joystick, trackpad, and special pen with pressure-sensitive pad; microphones, webcams, and digital cameras. They also include sensors that provide information about their environment—temperature, pressure, and so forth—to a computer. Another direct-entry mechanism is the optical laser scanner (e.g., scanners used with point-of-sale terminals in retail stores) that can read bar-coded data or optical character fonts.

computer keyboard

Output equipment includes video display terminals, ink-jet and laser printers, loudspeakers, headphones, and devices such as flow valves that control machinery, often in response to computer processing of sensor input data. Some devices, such as video display terminals and USB hubs, may provide both input and output. Other examples are devices that enable the transmission and reception of data between computers—e.g., modems and network interfaces.


Laser printer
laser
laser beam
1. Residual toner removed from drum by scraping or by passing under a discharging and erasing lamp.
2. Uniform high-voltage negative charge applied to drum.
negative charger
photoreceptor drum
lenses
path of laser beam
7. Toner particles are fused or melted onto the paper.
rotating hexagonal mirror
ABCDEFG
HIJKLMN
3. Laser beam selectively discharges the image areas onto the rotating drum, reversing the strong negative charge to a weak positive charge.
fuser roller
toner roller
printed side of paper
toner hopper
paper tray
blank paper
6. Negatively charged toner particles on the drum are transferred to the paper.
negatively charged toner particles
5. Paper receives a strong positive charge.
4. Negatively charged toner particles adhere to only the areas given a positive charge by the laser.
positive charger
© 2013 Encyclopædia Britannica, Inc.

laser printer

Most auxiliary storage devices—as, for example, CD-ROM and DVD drives, flash memory drives, and external disk drives also double as input/output devices (*see* computer memory). Even devices such as smartphones, tablet computers, and wearable devices like fitness trackers and smartwatches can be considered as peripherals, albeit ones that can function independently.

USB

Various standards for connecting peripherals to computers exist. For example, serial advanced technology attachment (SATA) is the most common interface, or bus, for magnetic disk drives. A bus (also known as a port) can be either serial or parallel, depending on whether the data path carries one bit at a time (serial) or many at once (parallel). Serial connections, which use relatively few wires, are generally simpler than parallel connections. Universal serial bus (USB) is a common serial bus.

USB port

I/O interface

The I/O subsystem of a computer provides an efficient mode of communication between the central system and the outside environment. It handles all the input-output operations of the computer system.

Peripheral Devices

Input or output devices that are connected to computer are called **peripheral devices**. These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be the part of computer system. These devices are also called **peripherals**.

For example: Keyboards, display units and printers are common peripheral devices.

There are three types of peripherals:

1. **Input peripherals** : Allows user input, from the outside world to the computer. Example: Keyboard, Mouse etc.
2. **Output peripherals**: Allows information output, from the computer to the outside world. Example: Printer, Monitor etc
3. **Input-Output peripherals**: Allows both input(from outised world to computer) as well as, output(from computer to the outside world). Example: Touch screen etc.

Interfaces

Interface is a shared boundary btween two separate components of the computer system which can be used to attach two or more components to the system for communication purposes.

There are two types of interface:

1. CPU Inteface
2. I/O Interface

Let's understand the I/O Interface in details,

Input-Output Interface

Peripherals connected to a computer need special communication links for interfacing with CPU. 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** because they provide communication links between processor bus and peripherals. They provide a method for transferring information between internal system and input-output devices.

Modes of I/O Data Transfer

Data transfer between the central unit and I/O devices can be handled in generally three types of modes which are given below:

1. Programmed I/O
2. Interrupt Initiated I/O
3. Direct Memory Access

Programmed I/O

Programmed I/O instructions are the result of I/O instructions written in computer program. Each data item transfer is initiated by the instruction in the program.

Usually the program controls data transfer to and from CPU and peripheral. Transferring data under programmed I/O requires constant monitoring of the peripherals by the CPU.

Interrupt Initiated I/O

In the programmed I/O method the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is time consuming process because it keeps the processor busy needlessly.

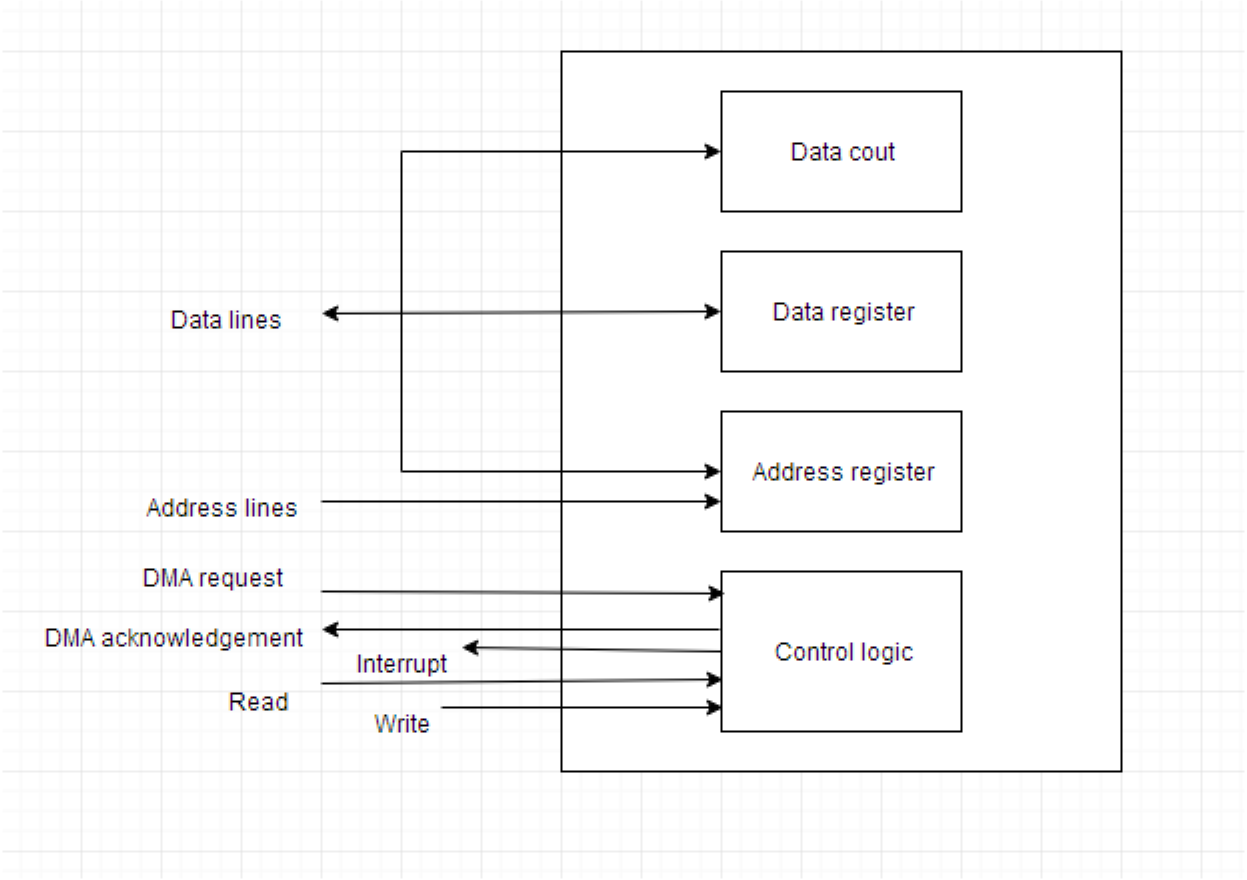
This problem can be overcome by using **interrupt initiated I/O**. In this when the interface determines that the peripheral is ready for data transfer, it generates an interrupt. After receiving the interrupt signal, the CPU stops the task which it is processing and service the I/O transfer and then returns back to its previous processing task.

Direct Memory Access

Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This technique is known as **DMA**.

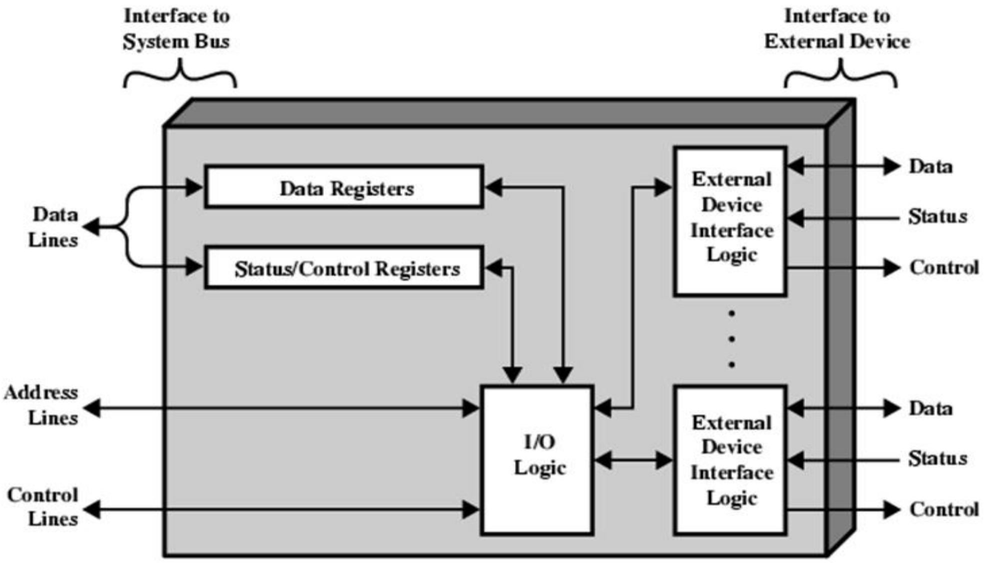
In this, the interface transfer data to and from the memory through memory bus. A DMA controller manages to transfer data between peripherals and memory unit.

Many hardware systems use DMA such as disk drive controllers, graphic cards, network cards and sound cards etc. It is also used for intra chip data transfer in multicore processors. In DMA, CPU would initiate the transfer, do other operations while the transfer is in progress and receive an interrupt from the DMA controller when the transfer has been completed.



Modes of Transfer

The mode of transferring information between internal storage and external I/O devices is known as I/O interface or input/output interface. The I/O module diagram is as follows:



I/O Module Decisions:

- Hide or reveal device properties to CPU
- Support multiple or single devices
- Control device functions or leave for CPU
- Also, O/S decisions – e.g. Unix treats everything it can as a file

Mode of Transfer:

- Data transfer between the central computer to I/O devices may be handled in variety of modes.

–Programmed I/O

–Interrupt Initiated I/O

–Direct Memory Access (DMA)

Let us discuss each in detail:

Programmed I/O:

- CPU requests I/O operation
- I/O module performs operations.
- I/O module sets status bits
- CPU checks status bits periodically
- I/O module does not inform CPU directly
- I/O module does not interrupt CPU
- CPU may wait or come back later
- Under programmed I/O data transfer is very like memory access (CPU viewpoint)
- Each device is given an unique identifier
- CPU commands contain identifier (address)

Dis-advantage of Programmed I/O:

- The problem with programmed I/O is that the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data.
- The processor, while waiting, must repeatedly interrogate the status of the I/O module.
- Performance of the entire system is severely degraded.

Interrupt Driven I/O Basic Operation:

- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data

Working of CPU in terms of interrupts:

- Issue read command.
- Do other work.
- Check for interrupt at end of each instruction cycle.
- If interrupted:-

–Save context (registers)
–Process interrupt

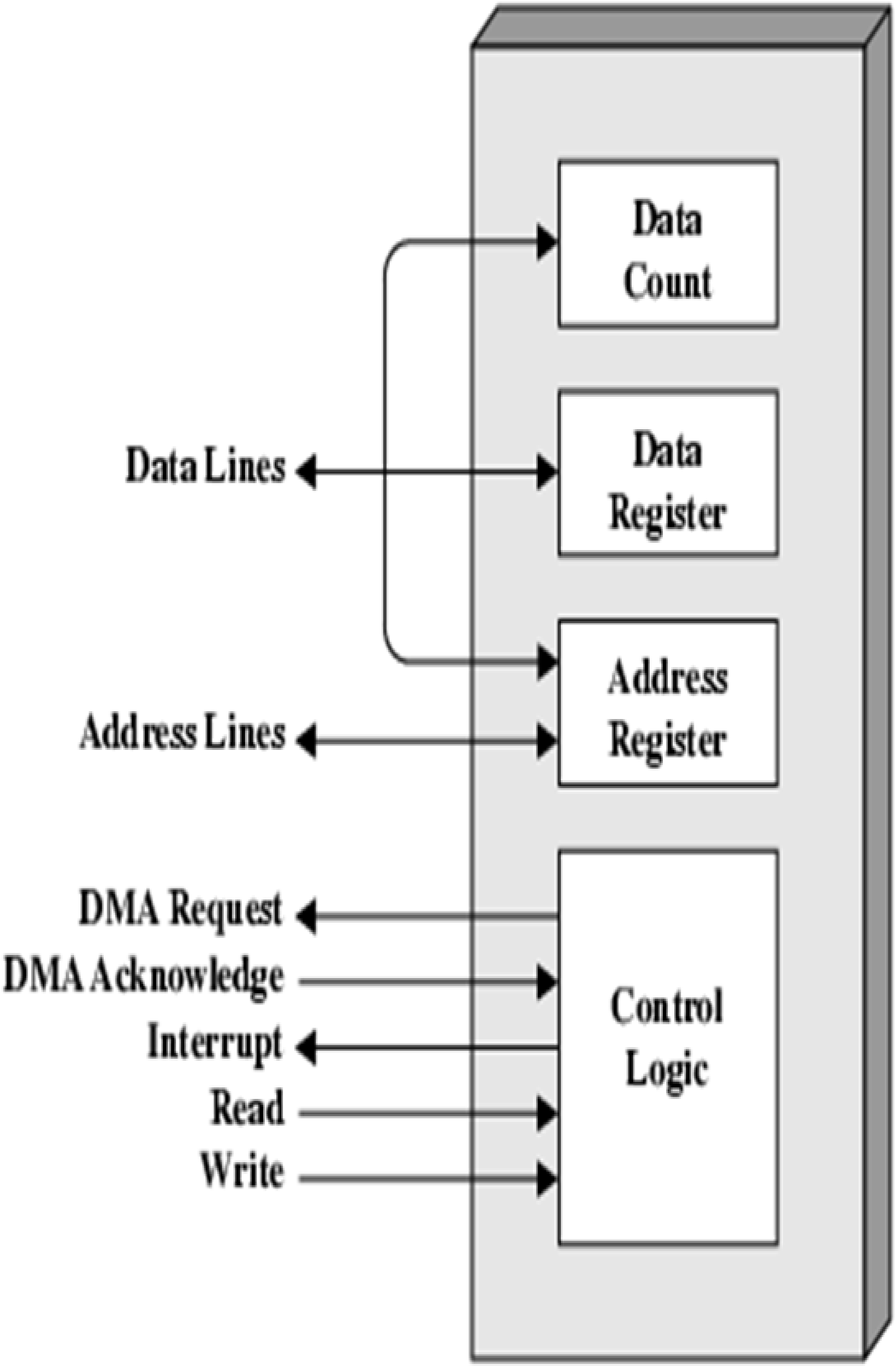
- Fetch data & store.
- See Operating Systems notes.

Multiple Interrupts:

- Each interrupt line has a priority
- Higher priority lines can interrupt lower priority lines
- If bus mastering only current master can interrupt

Direct Memory Access (DMA)

- Interrupt driven and programmed I/O require active CPU intervention
- Transfer rate is limited (processor to test and service the device)
- CPU is tied up for managing I/O transfer.
- Additional Module (hardware) on bus
- DMA controller takes over from CPU for I/ODMA is the answer.
- DMA module must use the bus only when the processor does not need it,
- It must force the processor to suspend operation temporarily. This technique is called cycle stealing


Data Count
Data Lines
Data Register
Address Register
Address Lines
DMA Request
DMA Acknowledge
Interrupt
Read
Write
Control Logic

DMA Operation:

- CPU tells DMA controller:-

–Read/Write

–Device address

–Starting address of memory block for data

–Amount of data to be transferred

- CPU carries on with other work
- DMA controller deals with transfer
- DMA controller sends interrupt when finished

Cyclic Stealing :

In this DMA controller transfers one word at a time after which it must return the control of the buses to the CPU. The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to “steal” one memory cycle.

Priority Interrupt

Data transfer between the CPU and the peripherals is initiated by the CPU. But the CPU cannot start the transfer unless the peripheral is ready to communicate with the CPU. When a device is ready to communicate with the CPU, it generates an interrupt signal. A number of input-output devices are attached to the computer and each device is able to generate an interrupt request.

The main job of the interrupt system is to identify the source of the interrupt. There is also a possibility that several devices will request simultaneously for CPU communication. Then, the interrupt system has to decide which device is to be serviced first.

Priority Interrupt

A priority interrupt is a system which decides the priority at which various devices, which generates the interrupt signal at the same time, will be serviced by the CPU. The system has authority to decide which conditions are allowed to interrupt the CPU, while some other interrupt is being serviced. Generally, devices with high speed transfer such as *magnetic disks* are given high priority and slow devices such as *keyboards* are given low priority.

When two or more devices interrupt the computer simultaneously, the computer services the device with the higher priority first.

Types of Interrupts:

Following are some different types of interrupts:

Hardware Interrupts

When the signal for the processor is from an external device or hardware then this interrupts is known as **hardware interrupt**.

Let us consider an example: when we press any key on our keyboard to do some action, then this pressing of the key will generate an interrupt signal for the processor to perform certain action. Such an interrupt can be of two types:

- **Maskable Interrupt**

The hardware interrupts which can be delayed when a much high priority interrupt has occurred at the same time.

- **Non Maskable Interrupt**

  The hardware interrupts which cannot be delayed and should be processed by the processor immediately.

Software Interrupts

The interrupt that is caused by any internal system of the computer system is known as a **software interrupt**. It can also be of two types:

- **Normal Interrupt**

  The interrupts that are caused by software instructions are called **normal software interrupts**.

- **Exception**

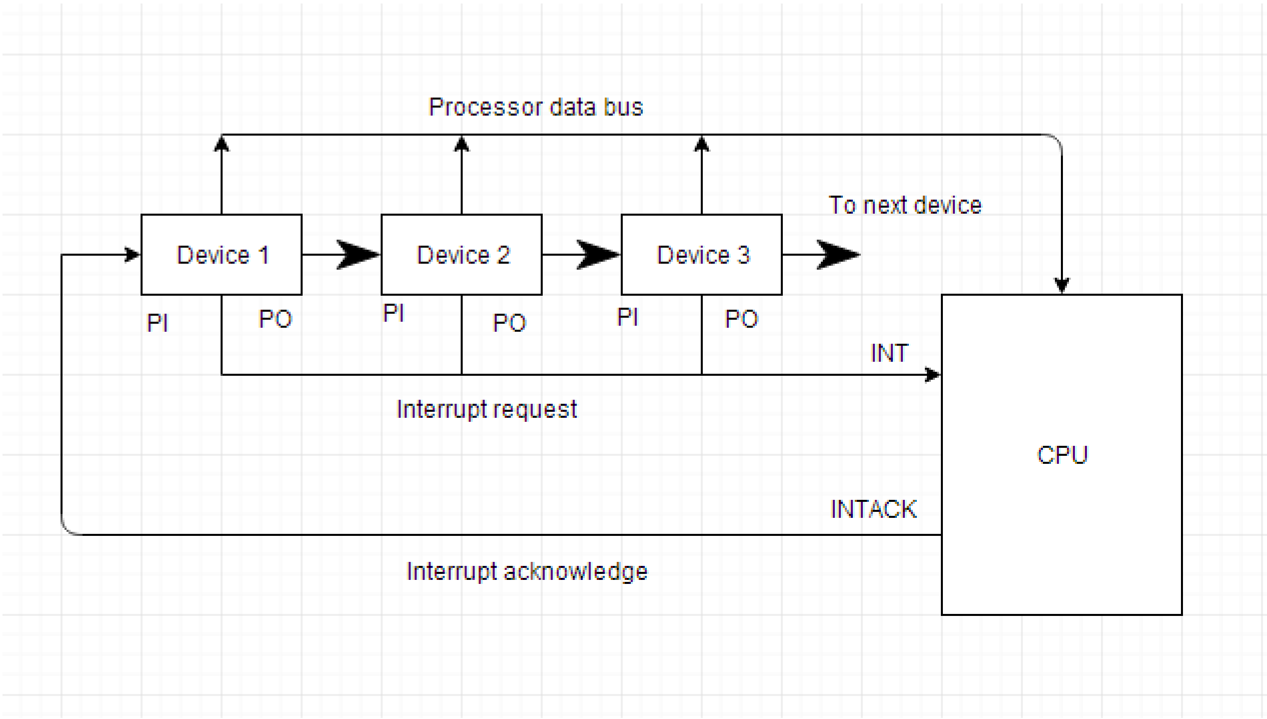
  Unplanned interrupts which are produced during the execution of some program are called **exceptions**, such as division by zero.

Daisy Chaining Priority

This way of deciding the interrupt priority consists of serial connection of all the devices which generates an interrupt signal. The device with the highest priority is placed at the first position followed by lower priority devices and the device which has lowest priority among all is placed at the last in the chain.

In daisy chaining system all the devices are connected in a serial form. The interrupt line request is common to all devices. If any device has interrupt signal in low level state then interrupt line goes to low level state and enables the interrupt input in the CPU. When there is no interrupt the interrupt line stays in high level state. The CPU respond to the interrupt by enabling the interrupt acknowledge line. This signal is received by the device 1 at its PI input. The acknowledge signal passes to next device through PO output only if device 1 is not requesting an interrupt.

The following figure shows the block diagram for daisy chaining priority system.



Direct Memory Access

Direct Memory Access (DMA) transfers the block of data between the memory and peripheral devices of the system, without the participation of the processor. The unit that controls the activity of accessing memory directly is called a DMA controller**.**

The processor relinquishes the system bus for a few clock cycles. So, the DMA controller can accomplish the task of data transfer via the system bus. In this section, we will study in brief about DMA, DMA controller, registers, advantages and disadvantages. So let us start.

Content: Direct Memory Access in Computer Architecture

1. What is DMA and Why it is used?
2. DMA Controller's Working
3. DMA Block Diagram
4. Advantages and Disadvantages
5. Key Takeaways

What is DMA and Why it is used?

Direct memory access (DMA) is a mode of data transfer between the memory and I/O devices. This happens without the involvement of the processor. We have two other methods of data transfer, programmed I/O and Interrupt driven I/O. Let's revise each and get acknowledge with their drawbacks.

In programmed I/O, the processor keeps on scanning whether any device is ready for data transfer. If an I/O device is ready, the processor fully dedicates itself in transferring the data between I/O and memory. It transfers data at a high rate, but it can't get involved in any other activity during data transfer. This is the major drawback of programmed I/O.

In Interrupt driven I/O, whenever the device is ready for data transfer, then it raises an interrupt to processor. Processor completes executing its ongoing instruction and saves its current state. It then switches to data transfer which causes a delay. Here, the processor doesn't keep scanning for peripherals ready for data transfer. But, it is fullyinvolved in the data transfer process. So, it is also not an effective way of data transfer.

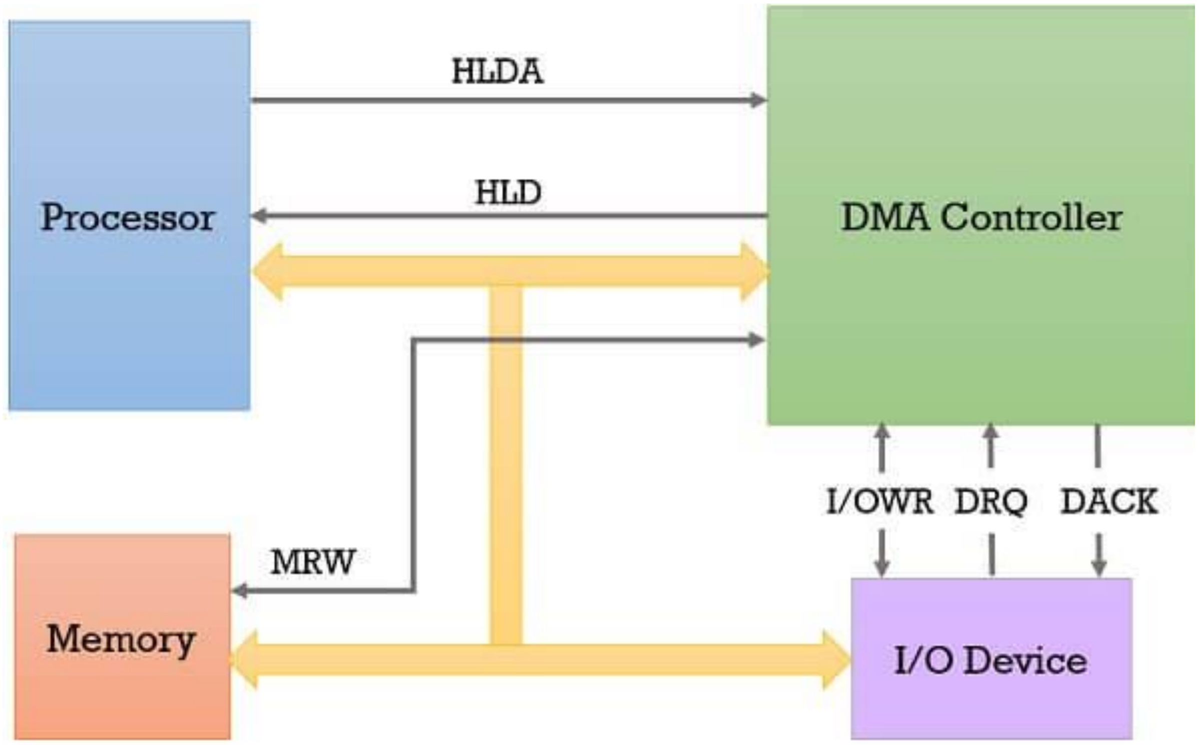
The above two modes of data transfer are not useful for transferring a large block of data. But, the DMA controller completes this task at a faster rate and is also effective for transfer of large data block.

The DMA controller transfers the data in three modes:

1. **Burst Mode:** Here, once the DMA controller gains the charge of the system bus, then it releases the system bus only after **completion** of data transfer. Till then the CPU has to wait for the system buses.
2. **Cycle Stealing Mode:** In this mode, the DMA controller **forces** the CPU to stop its operation and **relinquish the control over the bus** for a **short term** to DMA controller. After the **transfer of every byte**, the DMA controller **releases** the **bus** and then again requests for the system bus. In this way, the DMA controller steals the clock cycle for transferring every byte.
3. **Transparent Mode:** Here, the DMA controller takes the charge of system bus only if the **processor does not require the system bus**.

Direct Memory Access Controller & it's Working

DMA controller is a **hardware unit** that allows I/O devices to access memory directly without the participation of the processor. Here, we will discuss the working of the DMA controller. Below we have the diagram of DMA controller that explains its working:



DMA Controller Data Transfer

1. Whenever an I/O device wants to transfer the data to or from memory, it sends the DMA request (**DRQ**) to the DMA controller. DMA controller accepts this DRQ and asks the CPU to hold for a few clock cycles by sending it the Hold request (**HLD**).

2. CPU receives the Hold request (HLD) from DMA controller and relinquishes the bus and sends the Hold acknowledgement (**HLDA**) to DMA controller.

3. After receiving the Hold acknowledgement (HLDA), DMA controller acknowledges I/O device **(DACK)** that the data transfer can be performed and DMA controller takes the charge of the system bus and transfers the data to or from memory.

4. When the data transfer is accomplished, the DMA raise an **interrupt** to let know the processor that the task of data transfer is finished and the processor can take control over the bus again and start processing where it has left.

Now the DMA controller can be a separate unit that is shared by various I/O devices, or it can also be a part of the I/O device interface.

Direct Memory Access Diagram

After exploring the working of DMA controller, let us discuss the block diagram of the DMA controller. Below we have a block diagram of DMA controller.

Whenever a processor is requested to read or write a block of data, i.e. transfer a block of data, it instructs the DMA controller by sending the following information.

1. The first information is whether the data has to be read from memory or the data has to be written to the memory. It passes this information via read or write control lines that is between the processor and DMA controllers control logic unit**.**

2. The processor also provides the startingaddress of/ for the data block in the memory, from where the data block in memory has to be read or where the data block has to be written in memory. DMA controller stores this in its address register. It is also called the starting address register**.**

3. The processor also sends the word count, i.e. how many words are to be read or written. It stores this information in the data count or **the** word count register.

4. The most important is the address of I/O device that wants to read or write data. This information is stored in the data register**.**

Direct Memory Access Advantages and Disadvantages

Advantages:

1. Transferring the data without the involvement of the processor will speed up the read-write task.

2. DMA reduces the clock cycle requires to read or write a block of data.

3. Implementing DMA also reduces the overhead of the processor.

Disadvantages

1. As it is a hardware unit, it would cost to implement a DMA controller in the system.
2. Cache coherence problem can occur while using DMA controller.

Key Takeaways

- DMA is an abbreviation of direct memory access.

- DMA is **a** method of data transfer between main memory and peripheral devices.

- The hardware unit that controls the DMA transfer is a DMA controller.
- DMA controller transfers the data to and from memory without the participation of the processor.
- The processor provides the start address and the word count of the data block which is transferred to or from memory to the DMA controller and frees the bus for DMA controller to transfer the block of data.
- DMA controller transfers the data block at the faster rate as data is directly accessed by I/O devices and is not required to pass through the processor which save the clock cycles.
- DMA controller transfers the block of data to and from memory in three modes burstmode, cyclestealmode and transparentmode.
- DMA can be configured in various ways it can be a part of individual I/O devices, or all the peripherals attached to the system may share the same DMA controller.

Thus the DMA controller is a convenient mode of data transfer. It is preferred over the programmed I/O and Interrupt-driven I/O mode of data transfer.

Input-Output Processor

An input-output processor (IOP) is a processor with direct memory access capability. In this, the computer system is divided into a memory unit and number of processors.

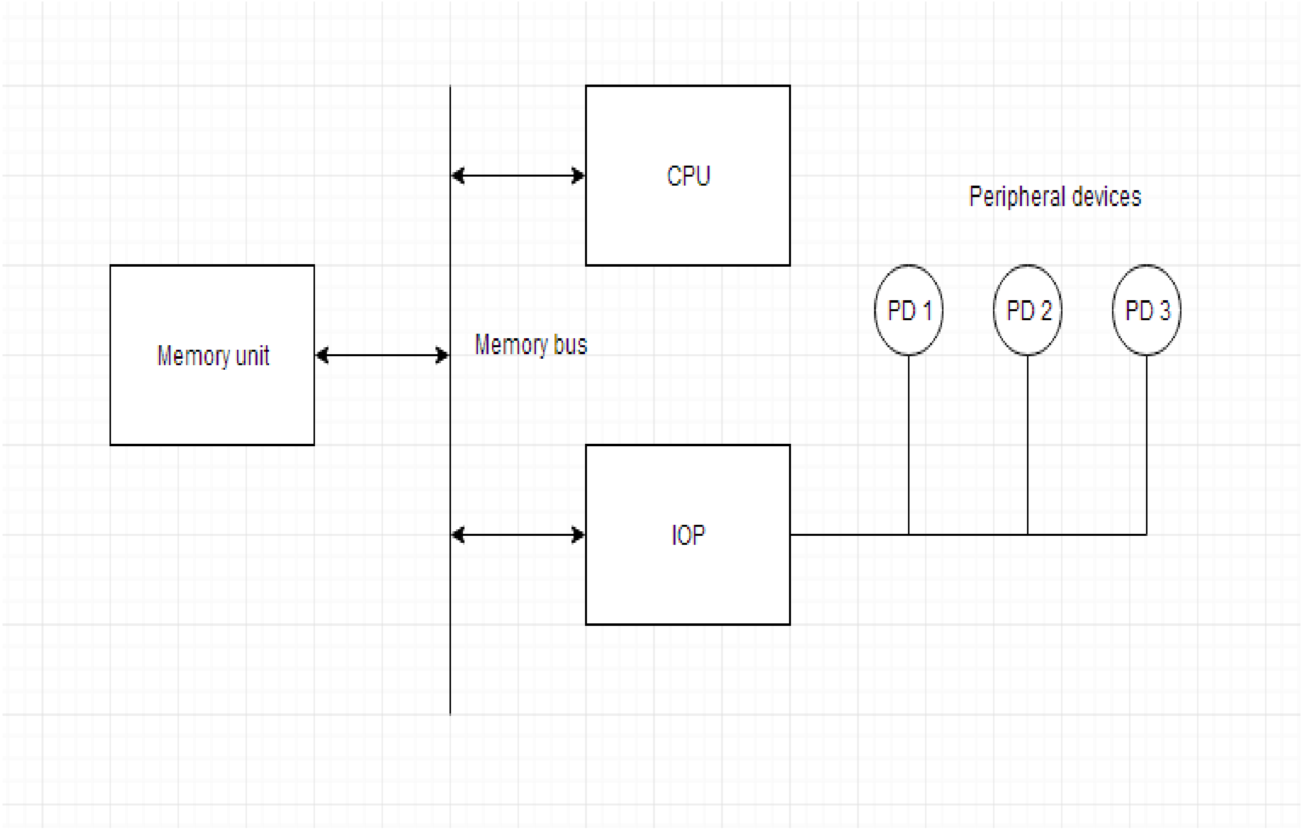
Each IOP controls and manage the input-output tasks. The IOP is similar to CPU except that it handles only the details of I/O processing. The IOP can fetch and execute its own instructions. These IOP instructions are designed to manage I/O transfers only.

Block Diagram Of I/O Processor

Below is a block diagram of a computer along with various I/O Processors. The memory unit occupies the central position and can communicate with each processor.

The CPU processes the data required for solving the computational tasks. The IOP provides a path for transfer of data between peripherals and memory. The CPU assigns the task of initiating the I/O program.

The IOP operates independent from CPU and transfer data between peripherals and memory.



The communication between the IOP and the devices is similar to the program control method of transfer. And the communication with the memory is similar to the direct memory access method.

In large scale computers, each processor is independent of other processors and any processor can initiate the operation.

The CPU can act as master and the IOP act as slave processor. The CPU assigns the task of initiating operations but it is the IOP, who executes the instructions, and not the CPU. CPU instructions provide operations to start an I/O transfer. The IOP asks for CPU through interrupt.

Instructions that are read from memory by an IOP are also called *commands* to distinguish them from instructions that are read by CPU. Commands are prepared by programmers and are stored in memory. Command words make the program for IOP. CPU informs the IOP where to find the commands in memory.

Serial Communication

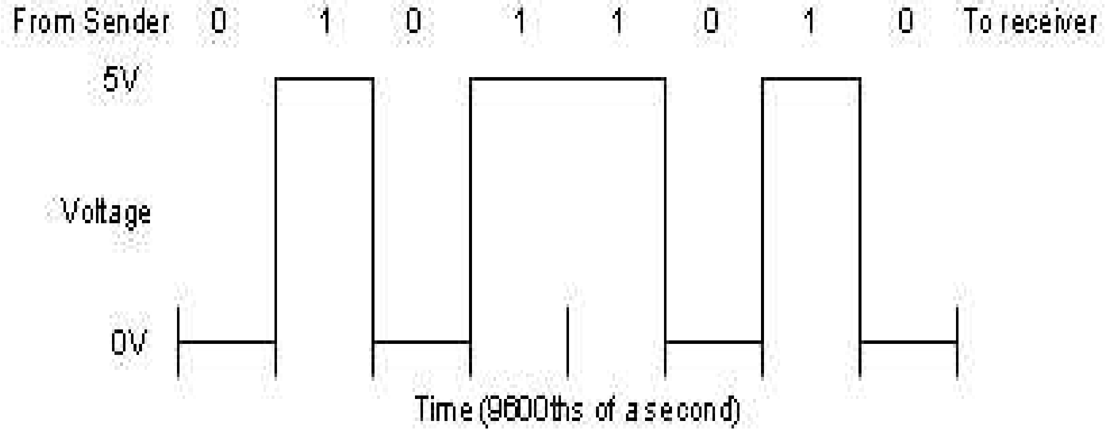
In order to make two devices communicate, whether they are desktop computers, microcontrollers, or any other form of integrated circuit, we need a method of communication and an agreed-upon language. The most common form of communication between electronic devices is *serial communication*. Communicating serially involves sending a series of digital pulses back and forth between devices at a mutually agreed-upon rate. The sender sends pulses representing the data to be sent at the agreed-upon *data rate*, and the receiver listens for pulses at that same rate. This is what's known as *asynchronous serial communication*. There isn't one common clock in asynchronous serial communication; instead, both devices have their own clock and agree on a rate to which to set their clocks.

For example, let's say two devices are to exchange data at a rate of 9600 bits per second. First, we would make three connections between the two devices:

- a common ground connection, so both devices have a common reference point to measure voltage by;
- one wire for the sender to send data to the receiver on (transmit line for the sender);
- one wire for the receiver to send date to the sender on (receive line for the sender).

Now, since the data rate is 9600 bits per second (sometimes called 9600 *baud*), the receiver will continually read the voltage that the sender is putting out, and every 1/9600th of a second, it will interpret that voltage as a new bit of data. If the voltage is high (+5V in the case of Wiring/Arduino, the PIC, and BX-24), it will interpret that bit of data as a 1. If it is low (0V in the case of Wiring/Arduino, the PIC, and BX-24), it will interpret that bit of data as a 0. By interpreting several bits of data over time, the receiver can get a detailed message from the sender. at 9600 baud, for example, 1200 bytes of data can be exchanged in one second. If you have a home computer and a modem, you've seen serial communication in action. Your computer's modem exchanges information with your service provider's modem serially.

Let's look at a byte of data being exchanged. Imagine I want to send the number 90 from one device to another. First, I have to convert the number from the decimal representation 90 to a binary representation. in binary, 90 is 01011010. So my sending device will pulse its transmit line as follows:



As you can tell from this diagram, both devices also have to agree on the order of the bits. Usually the sender sends the highest bit (or most significant bit) first in time, and the lowest (or least significant bit) last in time. As long as we have an agreed upon voltage, data rate, and order of interpretation of bits, we can exchange any data we want serially.

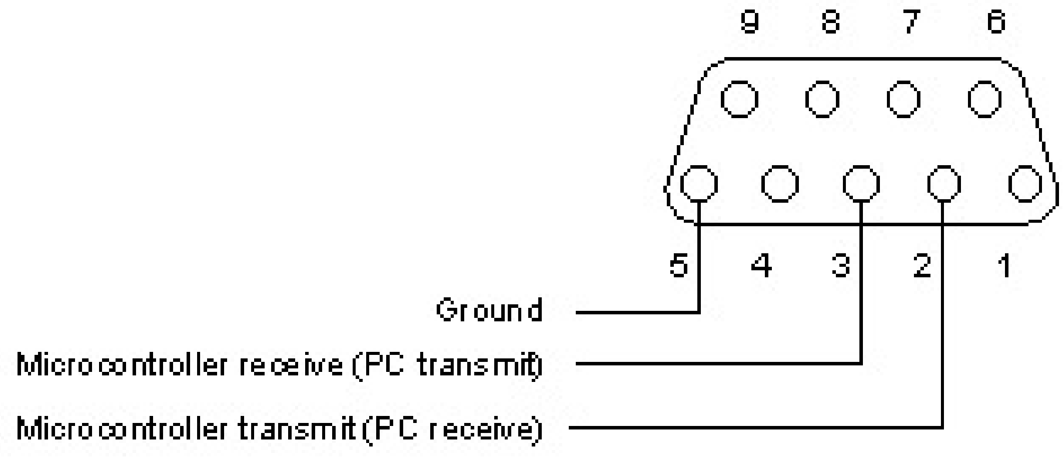
For the data transmission above, a high voltage indicates a bit value of 1, and a low voltage indicates a voltage of 0. This is known as true logic. Many serial protocols use inverted logic, meaning that a nigh voltage indicates a logic 0, and a low voltage indicates a logic 1. It's important to know whether your protocol is true or inverted. For example, RS-232, described below, uses inverted logic.

For most of our work, we'll be using a particular serial protocol called RS-232. The RS-232 standard defines voltages and general baud rate ranges for serial communications between devices using it. We won't be getting the voltages exactly right, but for most applications, we'll be close enough. Until recently, most desktop computers had an RS-232 or similar serial port. Now, many desktop computers are shifting to other forms of serial communication such as USB, or Universal Serial Bus, and Firewire, which allow for more flexible configurations and faster data rates. the RS-232 standard is still very common in other devices, though, as it is cheaper to use than USB, simpler to implement, consumes less power, and provides more than adequate speeds for exchanging control data (i.e. data that allows one device to control another).

Serial Communication to a PC or Mac

The RS-232 serial ports on Windows-based PC's looks like this:

PC serial cable (facing the soldering lugs of a female connector)

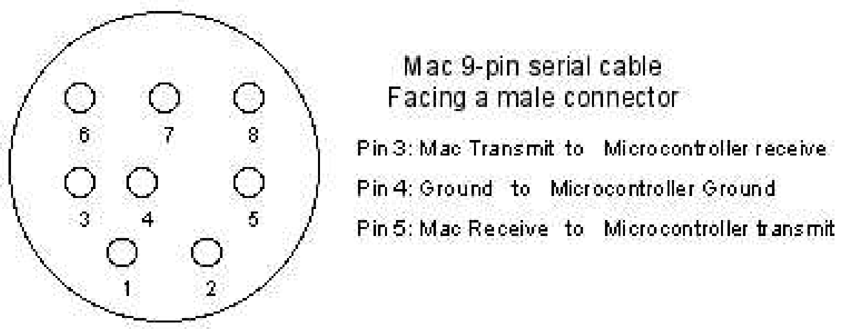

9
8
7
6
5
4
3
2
1
Ground
Microcontroller receive (PC transmit)
Microcontroller transmit (PC receive)

The connections for this are the same as the connector used to program the BX-24, and to debug from both the PIC and the BX-24, so you can use the same connector, or make a second one just like it. From there, you will need a 9-pin serial cable. Use any serial port you want, but be sure that all other programs that might be trying to use the serial port are turned off.

Wiring and Arduino boards have a built-in USB-to-serial converter, so they communicate serially using the USB port.

Macintoses used to have two serial ports, one for the printer, one for the modem. The current Mac models (and increaslingly, many PCs) do not come with a built-in serial port, and one must be installed. An RS-232 to USB serial port adaptor such as the Keyspan USB USA19HS Adaptor will do fine.

Classic Mac serial cable (Facing the male pins of a cable)


Mac 9-pin serial cable
Facing a male connector
Pin 3: Mac Transmit to Microcontroller receive
Pin 4: Ground to Microcontroller Ground
Pin 5: Mac Receive to Microcontroller transmit
6
7
8
3
4
5
1
2

Serial Output from the Microcontroller

Once you’ve got the computer and themicrocontroller connected, you’ll need to write a program to address the serial ports. The process is slightly different on the different microcontrollers, but there are some elements common to all of them.

Desktop computers have a place they store incoming data from the serial port, called a serial buffer. It's a little area of memory to store whatever comes in the serial port. Because of this, they can do other tasks while waiting for data to come in, and act on the data from the buffer. Microcontrollers usually don't have a serial buffer, but the BX-24 can have a small one, as do Wiring and Arduino. Details on how to set it up on the BX-24 follow. It's automatically set up for you on a Wiring or Arduino module. On the PIC, you don'thave a serial buffer, so you don't have to set it up.

Multimedia computers have one or more serial ports, and each port can be controlled by only one program at a time. To use a port, you have to open it, set its parameters, and then look for data. On the PIC, there are no serial ports. Each serial command defines the parameters of communication within the command. On the BX-24, you have to define the port, open it, then look for data, as on the PC.

On Wiring or Arduino boards, the serial pins are fixed, and you can't change them. You can, however, use the software serial library for Arduino if you need to use different pins, or if you need more than one serial port.

See the PIC serial notes for the software details of serial output using PicBasic Pro. See the BX-24 serial notes for the software details using a BX-24. See the Arduino serial lab for more on serial on Arduino.

If you're sending a single byte out serially, it's easy for the receiving device to know what's coming. It will read a byte and know the value. Sometimes, however, you have to send multiple bytes to get a message across. Those bytes have to be reassembled on the receiving side. More on that in the serial data interpretation notes.

I/O Controllers

Alternatively referred to as an input/output interface, IOC, or PIOC for Peripheral input/output controller. The input/output controller is a device that interfaces between an input or output device and the computer or hardware device. The input/output controller on a computer is commonly on the motherboard.

Additionally, an I/O controller can allow for additional input or output devices for the computer. It can also be an internal add-on component that replaces a malfunctioning I/O controller on the motherboard.

Asynchronous data transfer

The internal operations of a CPU are synchronized by a common clock.

Generally, it is not possible, or at least not practical to synchronize I/O devices with the CPU's internal clock.

I/O devices operate at different speeds than the CPU (note that the same I/O device may be connected to computers with vastly different CPU speeds).

Data transfer links such as USB, Firewire, and Ethernet, operate on a clock that is separate from the CPU's internal clock, but common to devices at both ends of the link.

The operations of an I/O interface must somehow be synchronized with the CPU's activities. This is often done using *handshaking* signals. The I/O device and CPU set wires or flip-flops to 0 or one to indicate their current state. (Recall the Mano FGI and FGO flags.)

The sender on an RS-232 serial interface sets a wire known as RTS (ready to send) to indicate that it has data to send. The receiver responds by by setting the CTS (clear to send) wire to indicate that it's OK to begin sending the next byte.

Newer interfaces such as USB, Ethernet, etc. use more complex setups, but basic 2-wire handshaking is still present in many modern interfaces.

Strobe Control

In computer or memory technology, a strobe is a signal that is sent that validates data or other signals on adjacent parallel lines. In memory technology, the CAS (column address strobe) and RAS (row address strobe) signals are used to tell a dynamic RAM that an address is a column or row address.

Handshaking

Term used to describe the process of one computer establishing a connection with another computer or device. The handshake is often the steps of verifying the connection, the speed, or the authorization of a computer connection. An example of handshaking is when a modem connects to another Modem. The tones heard after the dialing are the handshake, indicating that the computers are greeting each other.

If you did not grow up on a dial-up Modem or you're nostalgic, you can listen to a Modem connecting to the Internet in the below sound file. Each time someone wanted to connect to the Internet this is the noise their Modem would make.

In this audio file, you hear the Modem dialing a phone number and then communicating with the other Modem over the phone line. The squealing noise heard after the phone number is the Modem establishing a connection (handshaking). Once the connection is established, the Modem goes silent.

In the business world you may make a couple of handshakes every day, but your computer makes hundreds if not thousands in the same time frame. The computer version of the handshake is a greeting, just like the human one. While your computer won't notice things like a limp hand or a sweaty palm during its handshakes, the quick burst of data it receives can be just as important as it learns how to communicate with various devices and other networks.

Handshaking Defined

Handshaking is an automated process that sets parameters for communication between two different devices before normal communication begins. Much like the way a human handshake sets the stage for the communication to follow, the computing handshake provides both devices with the basic rules for the way data is to be shared between them. These rules can include transfer rate, coding alphabet, parity, interrupt procedure and more.

When Handshaking Takes Place

In order to establish a connection between a computer and a device like a modem, printer, or server, the handshake process begins the connection by telling both devices how to communicate with each other. A classic example is the noises made when two dial-up modems connect to each other. That squealing noise is actually the handshaking procedure. A handshake can also be used between a computer and a printer before printing takes place to tell the printer how to receive and output the data it receives from the computer.

Why Handshaking Is Used

In addition to exchanging protocol information, handshaking is often used to verify the quality or speed of the connection, as well as any authority that may be required to complete the connection between devices. This latter purpose is a common part of establishing a connection between your computer and a remote server.

Unit-V

(Process Organization) :-

Basic Concept of 8-bit micro Processor (8085) and 16-bit Micro Processor (8086)

Both 8085 and 8086 are two major microprocessors designed by **Intel**. However, the crucial difference between 8085 and 8086 microprocessor is that an 8085 microprocessor is an 8-bit microprocessor i.e., can operate on 8-bit data at a time. As against 8086 is a 16-bit microprocessor, that can perform operation on 16-bit data in one cycle.

There exist various other factors that create significant differences between 8085 and 8086 microprocessor. In this section, we will discuss the other differences between 8085 and 8086 microprocessor using a comparison chart.

Content: 8085 Vs 8086

1. Comparison Chart
2. Definition
3. Key Differences
4. Conclusion

Comparison Chart

Basis for Comparison	**8085**	**8086**
Microprocessor type	8-bit	16-bit
Size of data bus	8-bit	16-bit
Size of address bus	16-bit	20-bit
Supportable memory capacity	64 KB	1 MB

Basis for Comparison	8085	8086
Operating frequency	3 MHz	5 MHz
Number of flags present	5	9
Number of transistors	Less (around 6500)	More (around 29000)
Operating mode	Only one	Two (minimum and maximum mode)
Pipelining	Unsupportable	Supportable
Cost	Low	Comparitively high
Memory segmentation	Unsupportable	Supportable
Instruction queue	Absent	Present
Addressing mode	5	9

Definition of 8085 Microprocessor

8085 is an **8-bit microprocessor** that is able to perform an operation on 8-bit data in a single cycle. Basically, it is called so because the ALU size is of 8-bit. It offers a data bus size of 8-bit with an address bus of 16-bit. Thus the permissible accessible memory space is 64KB.

More specifically we can say that as one data byte is stored in one memory location, therefore overall 64 kilobytes of data can be stored by 8085 microprocessor. The arithmetic and logic unit of 8085 microprocessor is able to perform operations like add, subtract, compare, complement, increment, decrement, shift, AND, OR, X-OR.

It is an accumulator based processor. This means that the data during operation reside in the accumulator and temporary registers. Also, the output of the operation is stored in the accumulator and in accordance with the outcome generated, the flags get set and reset. It performs program execution in 3 stages, which
are fetching, decoding and executing.

Firstly, fetching of instruction from the memory is done and then is stored in the instruction register. After that, the instruction is decoded by the decoder and the respective control signal is produced by the timing and control unit. According to the signal received the ALU performs the desired operation and stores the result in the accumulator and accordingly sets the flag register.

Definition of 8086 Microprocessor

8086 is a 16-bit microprocessor that is designed to perform execution over 16-bit data in one cycle. The reason behind it being a 16-bit microprocessor is the size of its ALU. The data bus size in case of 8086 microprocessor is 16-bit and that of the address bus is 20-bit. Therefore, the permissible memory location offered by the 8086 microprocessor is **1** MB.

More simply we can say that it can store 1 megabyte of data inside it. Along with the operation performed by the ALU of the 8085 microprocessor, the arithmetic and logic unit of 8086 microprocessor can also perform multiplication and division operation.

8086 has two separate operating units that work independently:

- Bus Interface Unit (BIU) and
- Execution Unit (EU)

The BIU is responsible for fetching the instructions from the memory or I/O device. And the EU executes the perfected instructions present in the instruction queue.The presence of an instruction queue in 8086 acts as an advantageous factor when compared with the 8085 microprocessor.

This is so because, by the help of instruction queue, BIU can prefetch the instructions and can store them in 6-byte instruction pre-fetch queue. This somehow reduces the instruction execution time and resultantly the overall efficiency of the system.

8086 microprocessor can operate as both single processors as well as a multi-processor system. Thus has two operating modes: **minimum mode** and **maximum mode**. In case of a single processor system, it operates in minimum mode. While for multiple processors, 8086 operates in maximum mode.

Key differences between 8085 and 8086 Microprocessor

1. The **size of the data bus** specifies the amount of data that can be fetched by the data bus in one cycle. The size of the data bus in the case of 8085 microprocessor is 8-bit while in case of 8086 microprocessor, it is 16-bit.

2. The **address bus size** in case of 8085 microprocessor is 16-bit whereas in case of 8086 microprocessor it is of 20-bit.

3. The **memory addressing capacity** of 8085 is 2^{16} i.e., 64 KB. On the contrary, the memory addressing the capacity of 8086 is 2^{20} i.e., 1 MB.

4. 8085 operates at a frequency of about 3 MHZ. While the **operating frequency** of 8086 microprocessor is 5 MHz, also the advanced version of 8086 microprocessor operates at a frequency about 8 and 10 MHz also.

5. 8085 microprocessor consists of less number of transistors in its structure. Whereas 8086 comparatively holds a very large number of processors in it.

6. 8085 supports a single **mode of operation**, while 8086 supports two operating modes, minimum and maximum mode.

7. There exist total 5 **flags** (i.e., sign, zero, auxiliary carry, parity and carry flag) in 8085 microprocessor. As against overall 9 flags (i.e., overflow, direction, interrupt, trap and rest other of 8085) are present in 8086 microprocessor.

8. 8085 is a single processor configuration microprocessor. On the contrary 8086 is a multi-processor configuration microprocessor.

9. **Pipelining** is unsupportable by 8085. Whereas pipelining is supported by the 8086 microprocessor.

10. **Instruction queue** is absent in 8085 microprocessor. While as queuing is supported by 8086 it has instruction queue.

11. **Memory segmentation** is not supported by 8085 while it is supported by the 8086 microprocessor.

12. 8085 is an accumulator based processor. On the contrary 8086 is a general-purpose register type microprocessor.

Conclusion

Due to advanced architecture and more providable features, 8086 is more expensive than the 8085 microprocessor.

Assembly Instruction Set

The processor instruction set provides the instructions AND, OR, XOR, TEST, and NOT Boolean logic, which tests, sets, and clears the bits according to the need of the program.

The format for these instructions −

Sr.No.	Instruction	Format
1	AND	AND operand1, operand2
2	OR	OR operand1, operand2
3	XOR	XOR operand1, operand2
4	TEST	TEST operand1, operand2
5	NOT	NOT operand1

The first operand in all the cases could be either in register or in memory. The second operand could be either in register/memory or an immediate (constant) value. However, memory-to-memory operations are not possible. These instructions compare or match bits of the operands and set the CF, OF, PF, SF and ZF flags.

The AND Instruction

The AND instruction is used for supporting logical expressions by performing bitwise AND operation. The bitwise AND operation returns 1, if the matching bits from both the operands are 1, otherwise it returns 0. For example −

```
           Operand1:        0101
           Operand2:        0011
----------------------------
After AND -> Operand1:   0001
```

The AND operation can be used for clearing one or more bits. For example, say the BL register contains 0011 1010. If you need to clear the high-order bits to zero, you AND it with 0FH.

```
AND      BL,0FH;This sets BL to 00001010
```

Let's take up another example. If you want to check whether a given number is odd or even, a simple test would be to check the least significant bit of the number. If this is 1, the number is odd, else the number is even.

Assuming the number is in AL register, we can write –

```
AND      AL,01H;ANDingwith00000001
JZ    EVEN_NUMBER
```

The following program illustrates this −

Example

```
section .text
global _start           ;must be declared forusing gcc

_start:;tell linker entry point
   mov   ax,8h;getting 8in the ax
and   ax,1;and ax with1
   jz    evnn
   mov   eax,4;system call number (sys_write)
   mov   ebx,1;file descriptor (stdout)
   mov   ecx, odd_msg       ;message to write
   mov   edx, len2          ;length of message
int0x80;call kernel
   jmp   outprog
```

```
evnn:

   mov   ah,09h
   mov   eax,4;system call number (sys_write)
   mov   ebx,1;file descriptor (stdout)
   mov   ecx, even_msg      ;message to write
   mov   edx, len1          ;length of message
int0x80;call kernel

outprog:

   mov   eax,1;system call number (sys_exit)
int0x80;call kernel

section   .data
even_msg  db  'Even Number!';message showing even number
len1  equ  $ - even_msg

odd_msg db  'Odd Number!';message showing odd number
len2  equ  $ - odd_msg
```

When the above code is compiled and executed, it produces the following result −

```
Even Number!
```

Change the value in the ax register with an odd digit, like –

```
mov  ax,9h; getting 9in the ax
```

The program would display:

```
Odd Number!
```

Similarly to clear the entire register you can AND it with 00H.

The OR Instruction

The OR instruction is used for supporting logical expression by performing bitwise OR operation. The bitwise OR operator returns 1, if the matching bits from either or both operands are one. It returns 0, if both the bits are zero.

For example,

```
Operand1:     0101
Operand2:     0011
```

```
----------------------------
After OR -> Operand1:    0111
```

The OR operation can be used for setting one or more bits. For example, let us assume the AL register contains 0011 1010, you need to set the four low-order bits, you can OR it with a value 0000 1111, i.e., FH.

```
OR BL,0FH;This sets BL to  00111111
```

Example

The following example demonstrates the OR instruction. Let us store the value 5 and 3 in the AL and the BL registers, respectively, then the instruction,

```
OR AL, BL
```

should store 7 in the AL register –

```
section .text
global _start           ;must be declared forusing gcc

_start:;tell linker entry point
   mov    al,5;getting 5in the al
   mov    bl,3;getting 3in the bl
or     al, bl           ;or al and bl registers, result should be 7
add    al,byte'0';converting decimal to ascii

   mov    [result],  al
   mov    eax,4
   mov    ebx,1
   mov    ecx, result
   mov    edx,1
int0x80

outprog:
   mov    eax,1;system call number (sys_exit)
int0x80;call kernel

section    .bss
result resb 1
```

When the above code is compiled and executed, it produces the following result −

```
7
```

The XOR Instruction

The XOR instruction implements the bitwise XOR operation. The XOR operation sets the resultant bit to 1, if and only if the bits from the operands are different. If the bits from the operands are same (both 0 or both 1), the resultant bit is cleared to 0.

For example,

```
          Operand1:    0101
          Operand2:    0011
----------------------------

After XOR -> Operand1:    0110
```

XORing an operand with itself changes the operand to **0**. This is used to clear a register.

```
XOR     EAX, EAX
```

The TEST Instruction

The TEST instruction works same as the AND operation, but unlike AND instruction, it does not change the first operand. So, if we need to check whether a number in a register is even or odd, we can also do this using the TEST instruction without changing the original number.

```
TEST    AL, 01H
JZ      EVEN_NUMBER
```

The NOT Instruction

The NOT instruction implements the bitwise NOT operation. NOT operation reverses the bits in an operand. The operand could be either in a register or in the memory.

For example,

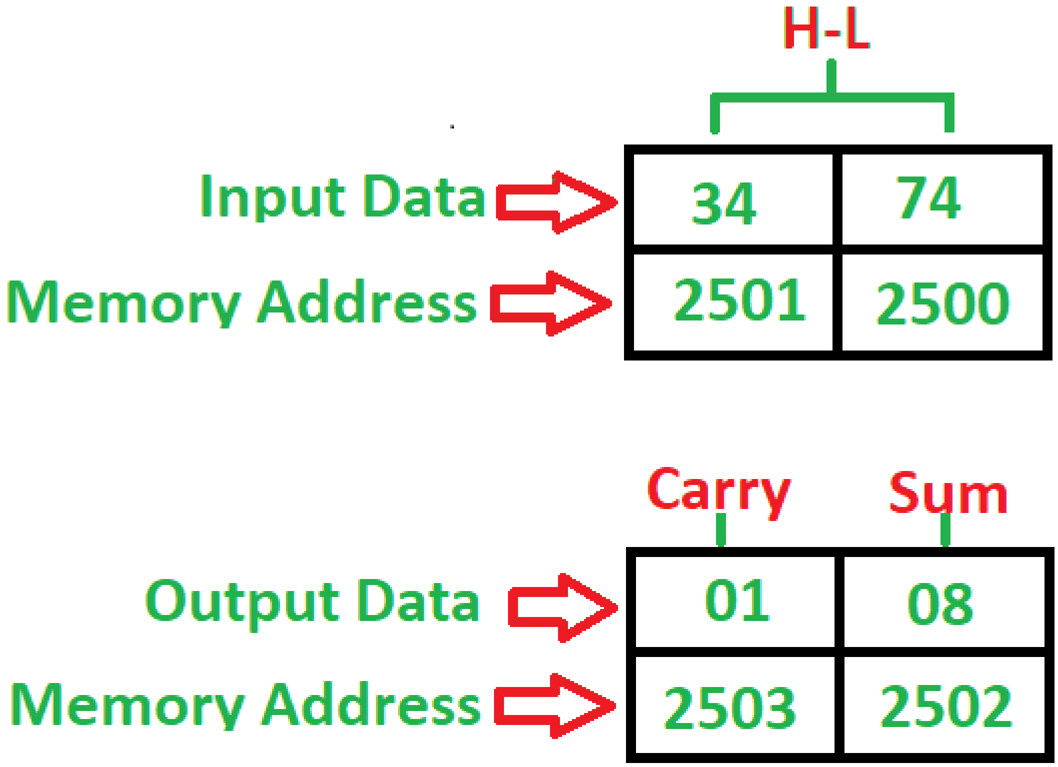
```
          Operand1:    0101 0011
After NOT -> Operand1:    1010 1100
```

Assembly language program of (8085):

Addition of two numbers

Problem – Write a program to add 2-BCD numbers where starting address is 2000 and the numbers is stored at 2500 and 2501 memory addresses and store sum into 2502 and carry into 2503 memory address.

Example –



Algorithm –

1. Load 00H in a register (for carry)
2. Load content from memory into register pair
3. Move content from L register to accumulator
4. Add content of H register with accumulator
5. Add 06H if sum is greater than 9 or Auxiliary Carry is not zero
6. If carry flag is not equal to 1, go to step 8
7. Increment carry register by 1
8. Store content of accumulator into memory
9. Move content from carry register to accumulator
10. Store content of accumulator into memory

11. Stop

Program –

MEMORY	MNEMONICS	OPERANDS	COMMENT
2000	MVI	C, 00H	[C] <- 00H, carry
2002	LHLD	[2500]	[H-L] <- [2500]
2005	MOV	A, L	[A] <- [L]
2006	ADD	H	[A] <- [A] + [H]
2007	DAA		Add 06 if sum > 9 or AC = 1
2008	JNC	200C	Jump if no carry
200B	INR	C	[C] <- [C] + 1
200C	STA	[2502]	[A] -> [2502], sum
200F	MOV	A, C	[A] <- [C]
2010	STA	[2503]	[A] -> [2503], carry
2013	HLT		Stop

Explanation – Registers A, C, H, L are used for general purpose

1. MVI is used to move data immediately into any of registers (2 Byte)
2. LHLD is used to load register pair direct using 16-bit address (3 Byte instruction)
3. MOV is used to transfer the data from memory to accumulator (1 Byte)

4. ADD is used to add accumulator with any of register (1 Byte instruction)

5. STA is used to store data from accumulator into memory address (3 Byte instruction)

6. DAA is used to check if sum > 9 or AC = 1 add 06 (1 Byte instruction)

7. JNC is used jump if no carry to given memory location (3 Byte instruction)

8. INR is used to increase given register by 1 (1 Byte instruction)

9. HLT is used to halt the program

Block Transfer

One definition of a block transfer is a transfer of multiple bytes (or words or registers) of data under the control of a single software instruction. The instruction needs to know how many bytes of data to transfer, the address of the first byte of the source and the destination, and it has enough "guts" to transfer the whole block of data without additional instructions. (This sort of comes from the idea of a block transfer within a computer, from one location in memory to another, or from one device to another (a hard drive to a destination in memory). Sometimes such block transfer instructions are "really" one logical instruction in the firmware (but consisting of multiple subinstructions). Other times, a block transfer routine may be created at a higher level, by writing a subroutine in a programming language. In either case, the block transfer may be (is usually) smart enough to call for more help or at least indicate an error if something goes wrong.

Then there is the idea of (I think) a block transfer from one computer to another. Oops, maybe I'm about to confuse, for example, data transfer over a serial link vs. data transfer over a parallel link with block transfer -- I guess a modem (or its driving software) can be programmed to do a block transfer (of many bytes) even though each bit is transmitted (and received) separately. Likewise, multiple bytes of data can be transmitted over a parallel link, even though one byte (or group of bytes, or maybe better, one group of bits) is transmitted at a time. I guess I won't go any further in this discussion -- this may be appropriate for some other discussion.

Maybe a good way of looking at a block transfer is something like a transfer that uses software (or hardware) to automatically transfer a block of data larger than the interface being discussed can normally transfer at once, by breaking a block of data down to the size that can be handled at once and then continuing to transmit that amount of data time after time until the entire block is transferred.

(I think my definition demonstrates that I have a handle on the meaning (thus making me feel good) and can get the idea across to others, but probably should be simplified to use some more common terminology that is more compact.)

Aside: Is synchronous and asynchronous data transfer relevant here? Not quite, I don't think, that's a different subject, although perhaps block transfers may tend to be synchronous (??). Maybe not, that's not a good distinction. A block tranfer can be done over a modem, and modems / serial lines are ususally (I think) considered to be asyncronous. (Ignoring things like SDLCs(??), I think.)

find greatest number

In this tutorial, we have shared a program that compares three input integer numbers and returns the greatest number as output. To do this comparison, we are using a simple if-elseif-else block.

Program to find largest of three input numbers

The program will prompt user to input three integer numbers and based on the input, it would compare and display the greatest number as output. In this program num1, num2 & num3 are three int variables that represents number1, number2 and number3 consecutively.

```
#include<stdio.h>
int main()
{
   int num1,num2,num3;

   //Ask user to input any three integer numbers
   printf("\nEnter value of num1, num2 and num3:");
   //Store input values in variables for comparsion
   scanf("%d %d %d",&num1,&num2,&num3);

   if((num1>num2)&&(num1>num3))
      printf("\n Number1 is greatest");
   else if((num2>num3)&&(num2>num1))
      printf("\n Number2 is greatest");
   else
      printf("\n Number3 is greatest");
   return 0;
}
```

Output

:

```
Enter value of num1, num2 and num3: 15 200 101
 Number2 is greatest
```

Table search

A table is an arrangement of information in rows and columns containing cells that make comparing and contrasting information easier. As you can see in the following example, the data is easier to read in a table format.

Example table in HTML

Name	Date of Birth	Phone
Bob Smith	01-05-65	555-123-4567
Joe Smith	09-10-79	555-801-9876
Jane Doe	07-20-70	555-232-1818

Example of the same data in a list

Name,Date of Birth,Phone

Bob Smith,01-05-65,555-123-4567

Joe Smith, 09-10-79,555-801-9876

Jane Doe,07-20-70,555-232-1818

Tables in a database

In a database, a table consists of columns and rows of data, much like an Excel spreadsheet. It is often referenced by software programs and web pages, to store and retrieve data for users. There are multiple types of databases, but the structure of a table in each database type is mostly the same.

Numeric Manipulation

Much of any computer language concerns manipulation and comparison of strings and numbers. Perl has the usual complement of functions and operators for both. Note that much Perl string manipulation is performed with regular expressions, especially if performance is not an issue. Regular expressions are covered in their own document. This document details non-regex string handling.

Numeric Operators

Numeric manipulation is arithmetic. I won't give you Perl's order of precidence -- I don't know it myself. I use parentheses liberally, ensuring the intended calculation. Generally speaking, if you understand C or Java arithmetic, you understand Perl's:

Operator	**Functionality**	**Example**	**$var contains**
=	Assignment	$var = 8;	8
+	Addition	$var = 8 + 3;	11
-	Subtraction	$var = 8 - 3;	5
*	Multiplication	$var = 8 * 3;	24
/	Fractional Division (5/2 is 2.50, not 2)	$var = 8 / 3;	2.66666666666667
%	Modulus	$var = 8 % 3	2
	Whole number division	$var = (8-(8%3))/3	2

Perl's numeric relational operators are similar to C's:

Operator	**Functionality**	**Example**
>	Numeric greater than	if($var > 8)

>=	Numeric greater than or equal	if($var >= 8)
<	Numeric less than	if($var < 8)
<=	Numeric less than or equal	if($var <= 8)
==	Numeric equality	if($var == 8)

String Manipulation: dot, length(), substr(), index() and rindex()

Use the dot operator to concatinate 2 strings:

```
$var = "George" . " " . "W." . " " . "Bush";
```

In the preceding, $var now contains "George W. Bush".

You often need the length of a string. Use the length() function to get that.

The opposite of concatination is truncation. Personally, I do most truncation with regular expressions, which are discussed in a different document. However, you can also use the substr().substr() is faster than regular expressions, so if you're truncating in a tight loop, you might want to usesubstr(). Its syntax is as follows:

```
substr EXPR,OFFSET,LENGTH,REPLACEMENT
```

LENGTH and REPLACEMENT are optional. Personally, I seldom use REPLACEMENT because it's clearer to replace text with a concatenation after truncation. Here are some ways you can use substr() to truncate from the right and from the left of a string. In the following examples, imagine that:

```
$string = "Perl Programmer":
```

Task	**Statement**	**Value of $var**
Remove 3 characters from the front of the string	$var = substr($string, 3);	"l Programmer"

Remove all but the 3 final characters from the string	$var = substr($string", -3);	"mer"
Remove the 3 final characters from the string	$var = substr($string, 0, -3);	"Perl Program"
Remove all but the first 3 chaaracters from the string	$var = substr($string, 0, 3);	"Per"

To remove specific text, you can use the index()and rindex() functions to find the offset of the start of a substring, and then plug that number into the OFFSET or LENGTH arguments of substr(). Information on length(), substr(), index() and rindex()are contained in their respective man pages.

String Comparison Operators

Perl's string relational operators are distinct from its numeric relational operators:

Operator	**Functionality**	**Example**
gt	String greater than	if($var gt "cccc")
ge	String greater than or equal	if($var ge "cccc")
lt	String less than	if($var lt "cccc")
le	String less than or equal	if($var le "cccc")
eq	String equality	if($var eq "cccc")

Introductory Concept of pipeline

Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as **pipeline processing**.

Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end.

Pipelining increases the overall instruction throughput.

In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of combinational circuit is applied to the input register of the next segment.

Pipeline system is like the modern day assembly line setup in factories. For example in a car manufacturing industry, huge assembly lines are setup and at each point, there are robotic arms to perform a certain task, and then the car moves on ahead to the next arm.

Types of Pipeline

It is divided into 2 categories:

1. Arithmetic Pipeline
2. Instruction Pipeline

Arithmetic Pipeline

Arithmetic pipelines are usually found in most of the computers. They are used for floating point operations, multiplication of fixed point numbers etc. For example: The input to the Floating Point Adder pipeline is:

```
X = A*2^a

Y = B*2^b
```

Here A and B are mantissas (significant digit of floating point numbers), while **a** and **b** are exponents.

The floating point addition and subtraction is done in 4 parts:

1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract mantissas
4. Produce the result.

Registers are used for storing the intermediate results between the above operations.

Instruction Pipeline

In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system.

An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.

Pipeline Conflicts

There are some factors that cause the pipeline to deviate its normal performance. Some of these factors are given below:

1. Timing Variations

All stages cannot take same amount of time. This problem generally occurs in instruction processing where different instructions have different operand requirements and thus different processing time.

2. Data Hazards

When several instructions are in partial execution, and if they reference same data then the problem arises. We must ensure that next instruction does not attempt to access data before the current instruction, because this will lead to incorrect results.

3. Branching

In order to fetch and execute the next instruction, we must know what that instruction is. If the present instruction is a conditional branch, and its result will lead us to the next instruction, then the next instruction may not be known until the current one is processed.

4. Interrupts

Interrupts set unwanted instruction into the instruction stream. Interrupts effect the execution of instruction.

5. Data Dependency

It arises when an instruction depends upon the result of a previous instruction but this result is not yet available.

Advantages of Pipelining

1. The cycle time of the processor is reduced.
2. It increases the throughput of the system
3. It makes the system reliable.

Disadvantages of Pipelining

1. The design of pipelined processor is complex and costly to manufacture.
2. The instruction latency is more.

Flynn’s and Feng’s Classification

In 1966, Michael Flynn proposed a classification for computer architectures based on the number of instruction steams and data streams (Flynn’s Taxonomy).

- Flynn uses the stream concept for describing a machine's structure.
- A stream simply means a sequence of items (data or instructions).

- The classification of computer architectures based on the number of instruction steams and data streams (Flynn’s Taxonomy).

Flynn’s Taxonomy

- SISD: Single instruction single data
  - Classical von Neumann architecture

- SIMD: Single instruction multiple data
- MISD: Multiple instructions single data
  - Non existent, just listed for completeness

- MIMD: Multiple instructions multiple data
  - Most common and general parallel machine

SISD

- SISD (Singe-Instruction stream, Singe-Data stream)

- SISD corresponds to the traditional mono-processor (von Neumann computer). A single data stream is being processed by one instruction stream

- A single-processor computer (uni-processor) in which a single stream of instructions is generated from the program.

SIMD

- SIMD (Single-Instruction stream, Multiple-Data streams)

- Each instruction is executed on a different set of data by different processors i.e multiple processing units of the same type process on multiple-data streams.

- This group is dedicated to array processing machines.
- Sometimes, vector processors can also be seen as a part of this group.

MISD

- MISD (Multiple-Instruction streams, Singe-Data stream)
- Each processor executes a different sequence of instructions.

- In case of MISD computers, multiple processing units operate on one single-data stream .
- In practice, this kind of organization has never been used

MIMD

- MIMD (Multiple-Instruction streams, Multiple-Data streams)
- Each processor has a separate program.
- An instruction stream is generated from each program.
- Each instruction operates on different data.

- This last machine type builds the group for the traditional multi-processors. Several processing units operate on multiple-data streams

Parallel Architectural classification

Parallel processing has been developed as an effective technology in modern computers to meet the demand for higher performance, lower cost and accurate results in real-life applications. Concurrent events are common in today's computers due to the practice of multiprogramming, multiprocessing, or multicomputing.

Modern computers have powerful and extensive software packages. To analyze the development of the performance of computers, first we have to understand the basic development of hardware and software.

- **Computer Development Milestones** − There is two major stages of development of computer - **mechanical** or **electromechanical** parts. Modern computers evolved after the introduction of electronic components. High mobility electrons in electronic computers replaced the operational parts in mechanical computers. For information transmission, electric signal which travels almost at the speed of a light replaced mechanical gears or levers.

- **Elements of Modern computers** − A modern computer system consists of computer hardware, instruction sets, application programs, system software and user interface.

The computing problems are categorized as numerical computing, logical reasoning, and transaction processing. Some complex problems may need the combination of all the three processing modes.

- **Evolution of Computer Architecture** − In last four decades, computer architecture has gone through revolutionary changes. We started with Von Neumann architecture and now we have multicomputers and multiprocessors.
- **Performance of a computer system** − Performance of a computer system depends both on machine capability and program behavior. Machine capability can be improved with better hardware technology, advanced architectural features and efficient resource management. Program behavior is unpredictable as it is dependent on application and run-time conditions

Multiprocessors and Multicomputers

In this section, we will discuss two types of parallel computers −

- Multiprocessors
- Multicomputers

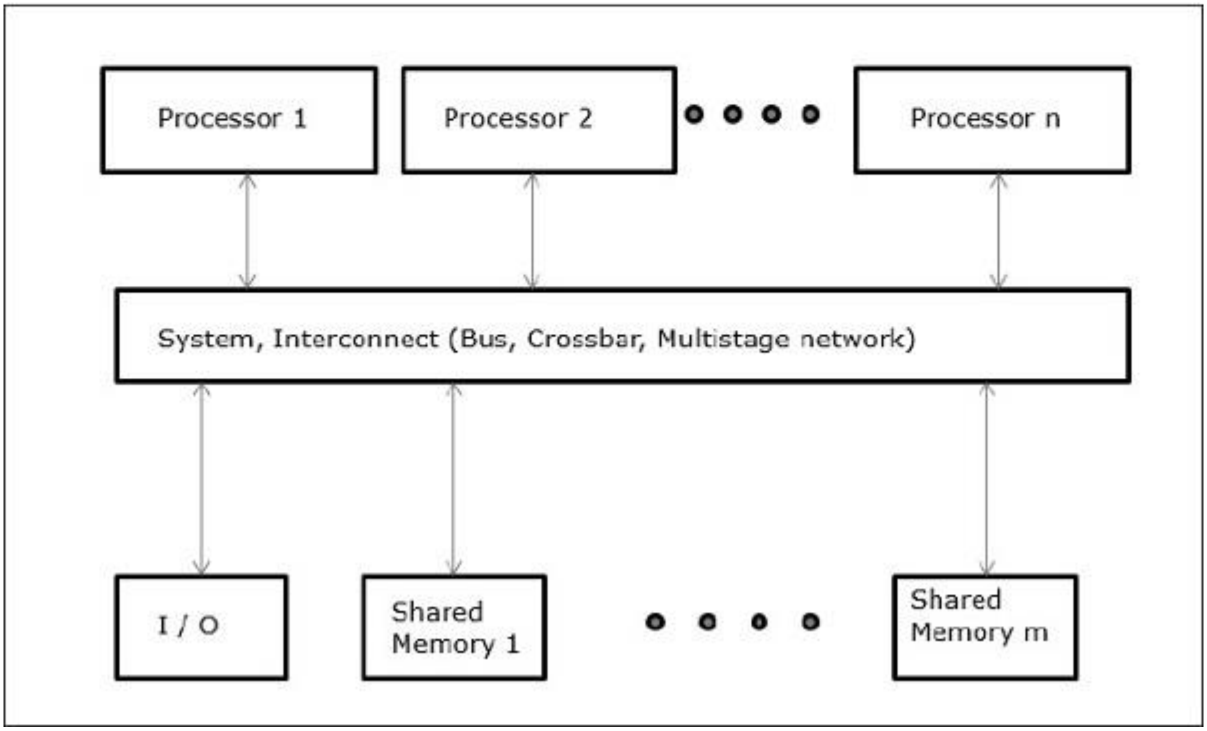
Shared-Memory Multicomputers

Three most common shared memory multiprocessors models are −

Uniform Memory Access (UMA)

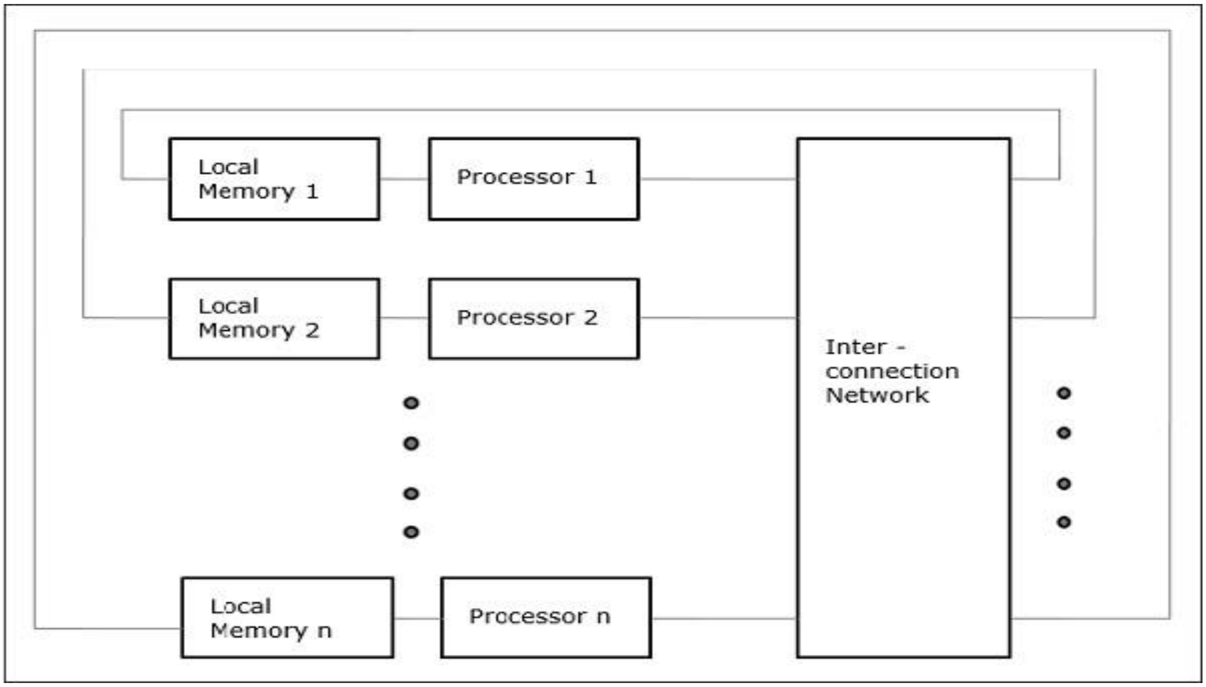
In this model, all the processors share the physical memory uniformly. All the processors have equal access time to all the memory words. Each processor may have a private cache memory. Same rule is followed for peripheral devices.

When all the processors have equal access to all the peripheral devices, the system is called a **symmetric multiprocessor**. When only one or a few processors can access the peripheral devices, the system is called an **asymmetric multiprocessor**.



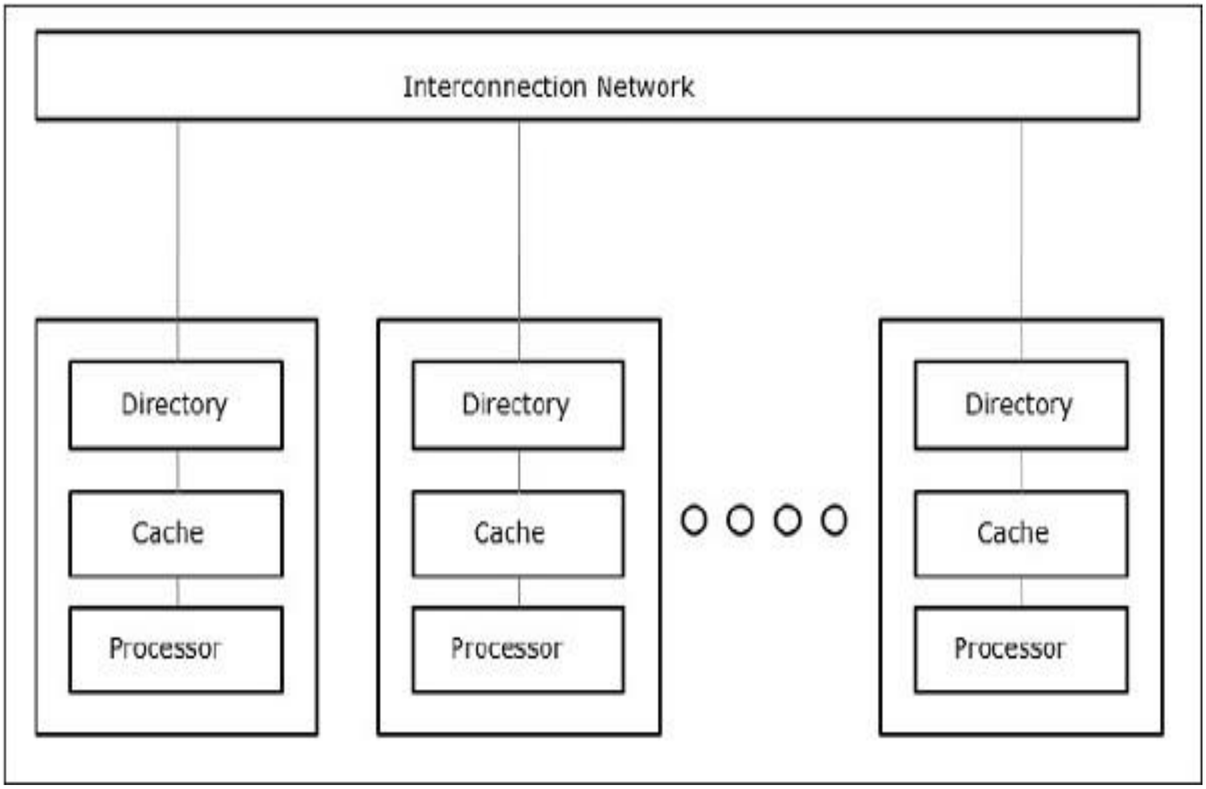
Non-uniform Memory Access (NUMA)

In NUMA multiprocessor model, the access time varies with the location of the memory word. Here, the shared memory is physically distributed among all the processors, called local memories. The collection of all local memories forms a global address space which can be accessed by all the processors.

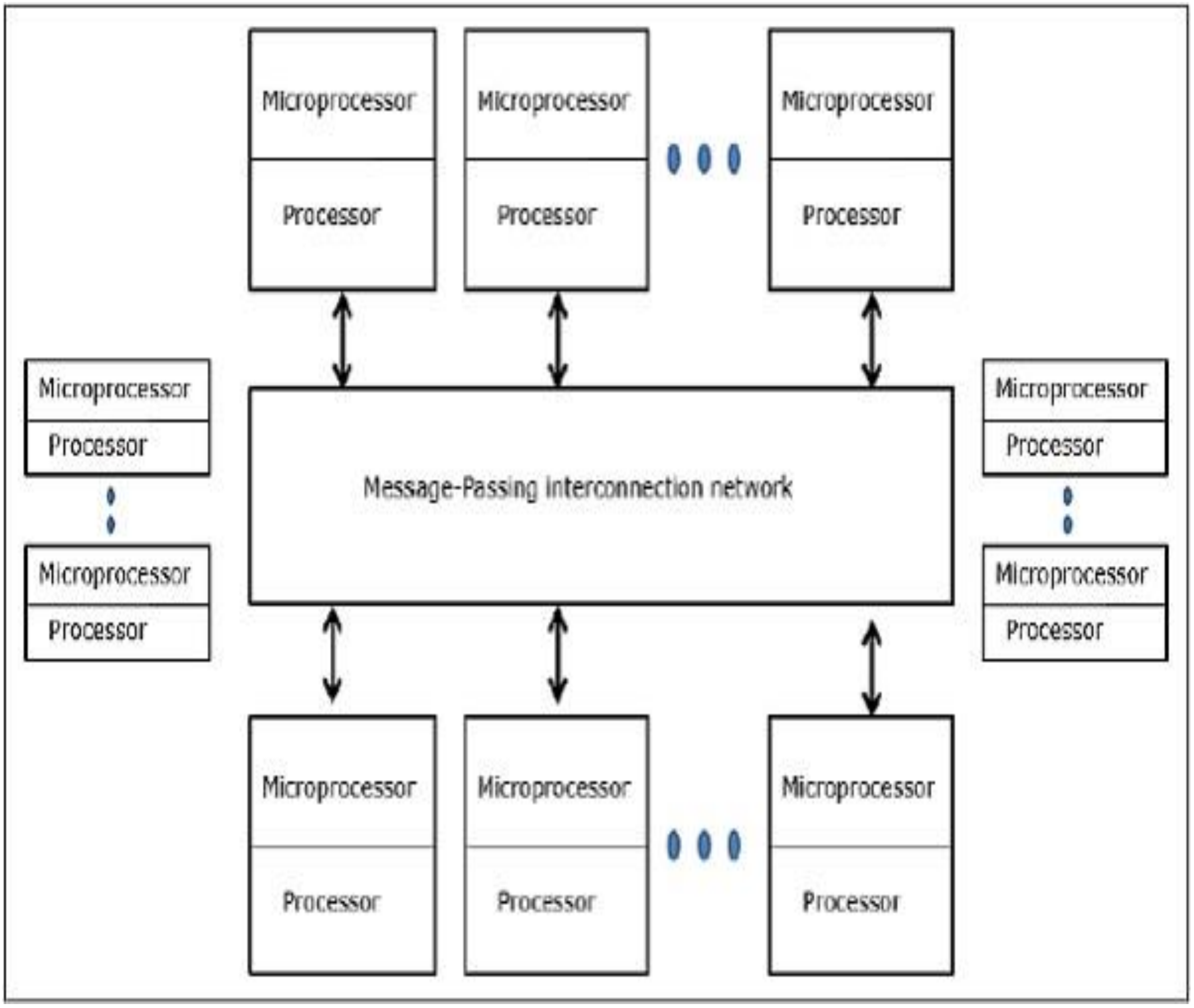


Cache Only Memory Architecture (COMA)

The COMA model is a special case of the NUMA model. Here, all the distributed main memories are converted to cache memories.



- **Distributed - Memory Multicomputers** − A distributed memory multicomputer system consists of multiple computers, known as nodes, inter-connected by message passing network. Each node acts as an autonomous computer having a processor, a local memory and sometimes I/O devices. In this case, all local memories are private and are accessible only to the local processors. This is why, the traditional machines are called **no-remote-memory-access (NORMA)** machines.



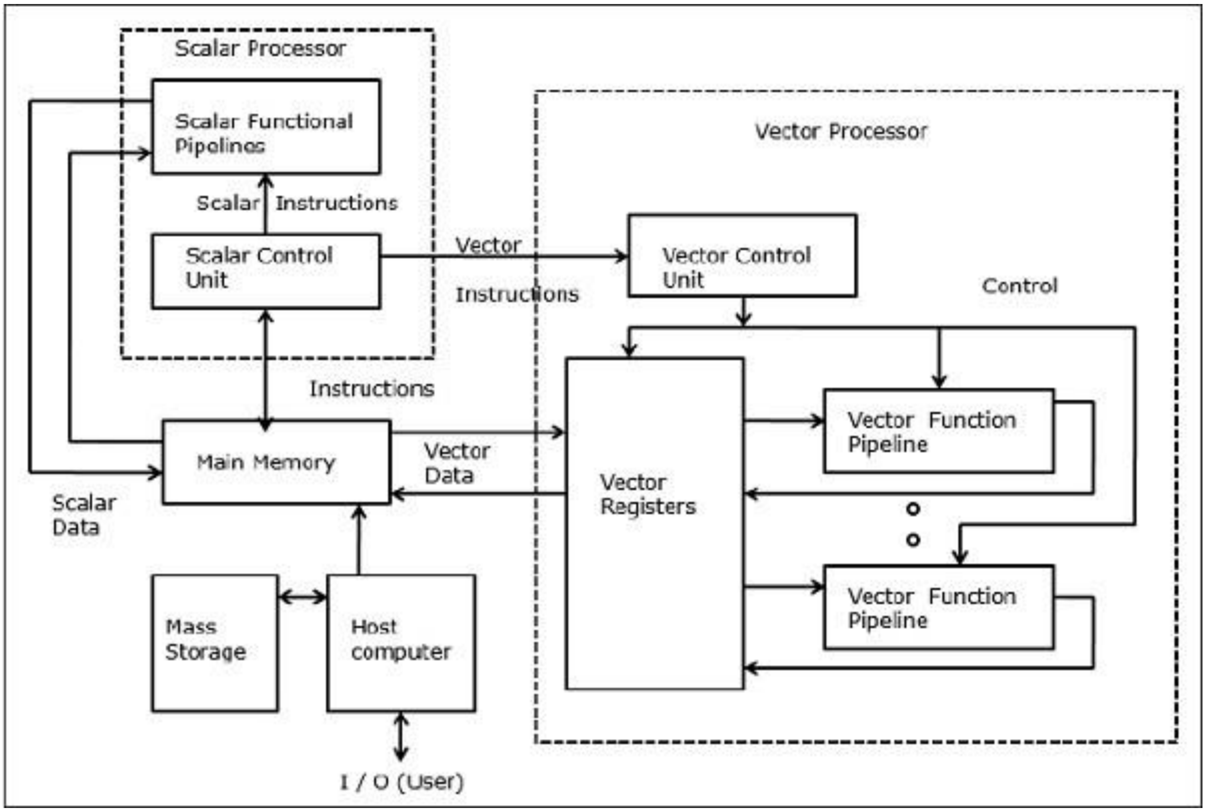
Multivector and SIMD Computers

In this section, we will discuss supercomputers and parallel processors for vector processing and data parallelism.

Vector Supercomputers

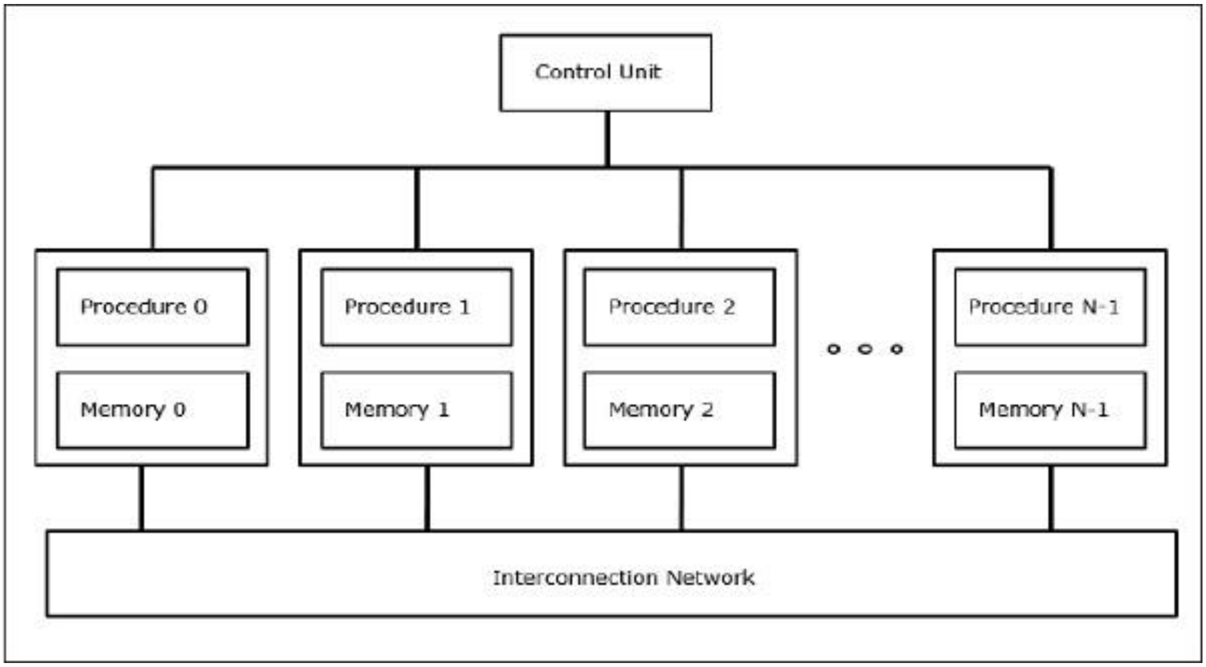
In a vector computer, a vector processor is attached to the scalar processor as an optional feature. The host computer first loads program and data to the main memory. Then the scalar control unit decodes all the instructions. If the decoded instructions are scalar operations or program operations, the scalar processor executes those operations using scalar functional pipelines.

On the other hand, if the decoded instructions are vector operations then the instructions will be sent to vector control unit.



SIMD Supercomputers

In SIMD computers, 'N' number of processors are connected to a control unit and all the processors have their individual memory units. All the processors are connected by an interconnection network.



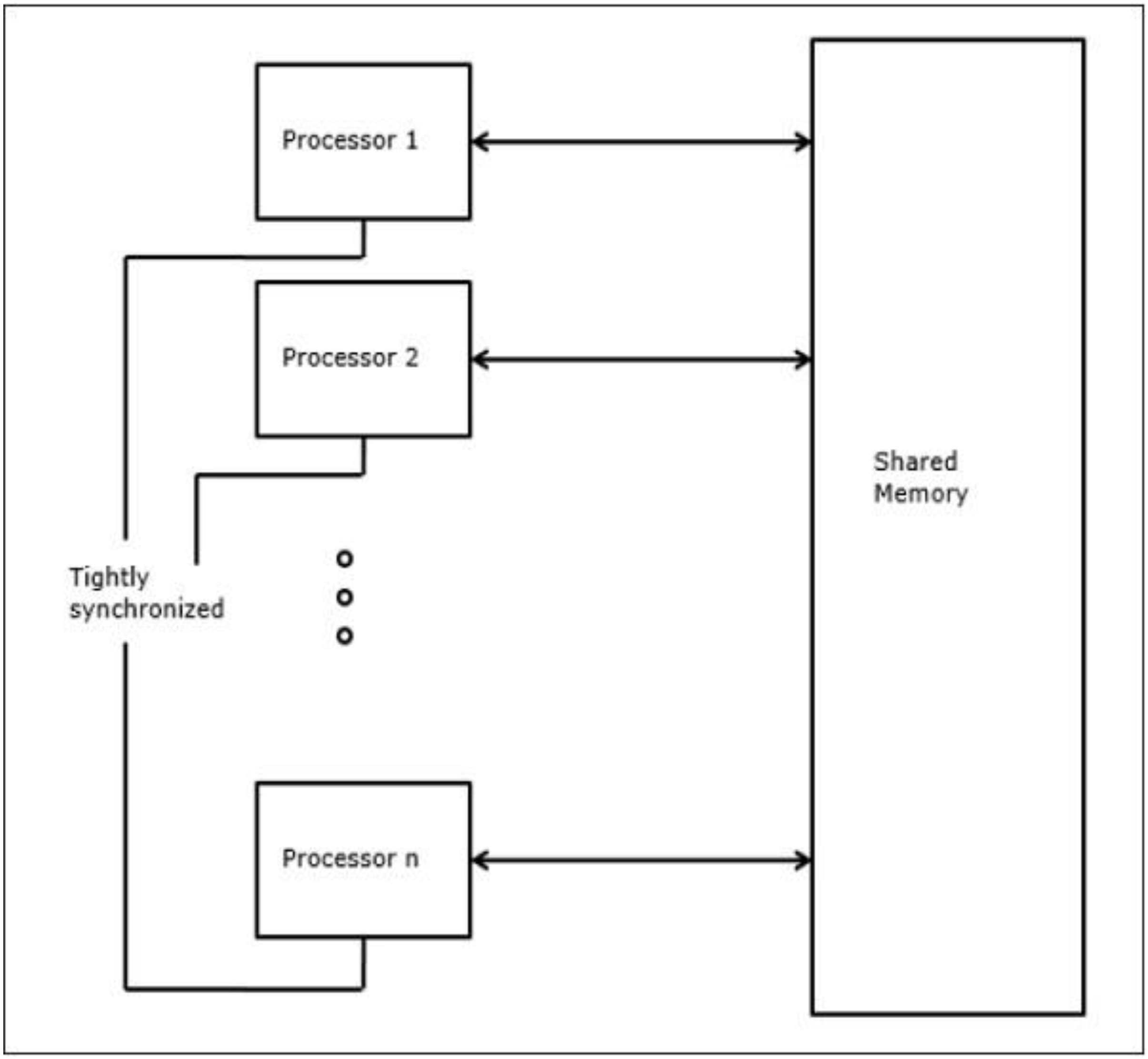
PRAM and VLSI Models

The ideal model gives a suitable framework for developing parallel algorithms without considering the physical constraints or implementation details.

The models can be enforced to obtain theoretical performance bounds on parallel computers or to evaluate VLSI complexity on chip area and operational time before the chip is fabricated.

Parallel Random-Access Machines

Sheperdson and Sturgis (1963) modeled the conventional Uniprocessor computers as random-access-machines (RAM). Fortune and Wyllie (1978) developed a parallel random-access-machine (PRAM) model for modeling an idealized parallel computer with zero memory access overhead and synchronization.



An N-processor PRAM has a shared memory unit. This shared memory can be centralized or distributed among the processors. These processors operate on a synchronized read-memory, write-memory and compute cycle. So, these models specify how concurrent read and write operations are handled.

Following are the possible memory update operations −

- **Exclusive read (ER)** − In this method, in each cycle only one processor is allowed to read from any memory location.
- **Exclusive write (EW)** − In this method, at least one processor is allowed to write into a memory location at a time.
- **Concurrent read (CR)** − It allows multiple processors to read the same information from the same memory location in the same cycle.
- **Concurrent write (CW)** − It allows simultaneous write operations to the same memory location. To avoid write conflict some policies are set up.

VLSI Complexity Model

Parallel computers use VLSI chips to fabricate processor arrays, memory arrays and large-scale switching networks.

Nowadays, VLSI technologies are 2-dimensional. The size of a VLSI chip is proportional to the amount of storage (memory) space available in that chip.

We can calculate the space complexity of an algorithm by the chip area (A) of the VLSI chip implementation of that algorithm. If T is the time (latency) needed to execute the algorithm, then A.T gives an upper bound on the total number of bits processed through the chip (or I/O). For certain computing, there exists a lower bound, f(s), such that

$$A.T^2 >= O\ (f(s))$$

Where A=chip area and T=time

Architectural Development Tracks

The evolution of parallel computers I spread along the following tracks −

- Multiple Processor Tracks
  - Multiprocessor track
  - Multicomputer track
- Multiple data track
  - Vector track
  - SIMD track
- Multiple threads track
  - Multithreaded track

- Dataflow track

In **multiple processor track**, it is assumed that different threads execute concurrently on different processors and communicate through shared memory (multiprocessor track) or message passing (multicomputer track) system.

In **multiple data track**, it is assumed that the same code is executed on the massive amount of data. It is done by executing same instructions on a sequence of data elements (vector track) or through the execution of same sequence of instructions on a similar set of data (SIMD track).

In **multiple threads track**, it is assumed that the interleaved execution of various threads on the same processor to hide synchronization delays among threads executing on different processors. Thread interleaving can be coarse (multithreaded track) or fine (dataflow track).

In the 80's, a special purpose processor was popular for making multicomputers called **Transputer**. A transputer consisted of one core processor, a small SRAM memory, a DRAM main memory interface and four communication channels, all on a single chip. To make a parallel computer communication, channels were connected to form a network of Transputers. But it has a lack of computational power and hence couldn't meet the increasing demand of parallel applications. This problem was solved by the development of RISC processors and it was cheap also.

Modern parallel computer uses microprocessors which use parallelism at several levels like instruction-level parallelism and data level parallelism.

High Performance Processors

RISC and RISCy processors dominate today's parallel computers market.

Characteristics of traditional RISC are −

- Has few addressing modes.
- Has a fixed format for instructions, usually 32 or 64 bits.
- Has dedicated load/store instructions to load data from memory to register and store data from register to memory.
- Arithmetic operations are always performed on registers.
- Uses pipelining.

Most of the microprocessors these days are superscalar, i.e. in a parallel computer multiple instruction pipelines are used. Therefore, superscalar processors can execute more than one instruction at the same time. Effectiveness of superscalar processors is dependent on the amount of instruction-level parallelism (ILP) available in the applications. To keep the pipelines filled, the instructions at the hardware level are executed in a different order than the program order.

Many modern microprocessors use super pipelining approach. In super pipelining, to increase the clock frequency, the work done within a pipeline stage is reduced and the number of pipeline stages is increased.

Very Large Instruction Word (VLIW) Processors

These are derived from horizontal microprogramming and superscalar processing. Instructions in VLIW processors are very large. The operations within a single instruction are executed in parallel and are forwarded to the appropriate functional units for execution. So, after fetching a VLIW instruction, its operations are decoded. Then the operations are dispatched to the functional units in which they are executed in parallel.

Vector Processors

Vector processors are co-processor to general-purpose microprocessor. Vector processors are generally register-register or memory-memory. A vector instruction is fetched and decoded and then a certain operation is performed for each element of the operand vectors, whereas in a normal processor a vector operation needs a loop structure in the code. To make it more efficient, vector processors chain several vector operations together, i.e., the result from one vector operation are forwarded to another as operand.

Caching

Caches are important element of high-performance microprocessors. After every 18 months, speed of microprocessors become twice, but DRAM chips for main memory cannot compete with this speed. So, caches are introduced to bridge the speed gap between the processor and memory. A cache is a fast and small SRAM memory. Many more caches are applied in modern processors like Translation Look-aside Buffers (TLBs) caches, instruction and data caches, etc.

Direct Mapped Cache

In direct mapped caches, a ‘modulo’ function is used for one-to-one mapping of addresses in the main memory to cache locations. As same cache entry can have multiple main memory blocks mapped to it, the processor must be able to determine whether a data block in the cache is the data block that is actually needed. This identification is done by storing a tag together with a cache block.

Fully Associative Cache

A fully associative mapping allows for placing a cache block anywhere in the cache. By using some replacement policy, the cache determines a cache entry in which it stores a cache block. Fully associative caches have flexible mapping, which minimizes the number of cache-entry conflicts. Since a fully associative implementation is expensive, these are never used large scale.

Set-associative Cache

A set-associative mapping is a combination of a direct mapping and a fully associative mapping. In this case, the cache entries are subdivided into cache sets. As in direct mapping, there is a fixed mapping of memory blocks to a set in the cache. But inside a cache set, a memory block is mapped in a fully associative manner.

Cache strategies

Other than mapping mechanism, caches also need a range of strategies that specify what should happen in the case of certain events. In case of (set-) associative caches, the cache must determine which cache block is to be replaced by a new block entering the cache.

Some well-known replacement strategies are −

- First-In First Out (FIFO)
- Least Recently Used (LRU)