UNIT – I

Computer Fundamentals: What is a Computer, Evolution of Computers, Generations of Computers, Classification of Computers, Anatomy of a Computer, Memory revisited, Introduction to Operating systems, Operational overview of a CPU. Introduction to Programming, Algorithms and Flowcharts: Programs and Programming, Programming languages, Compiler, Interpreter, Loader, Linker, Program execution, Fourth generation languages, Fifth generation languages, Classification of Programming languages, Structured programming concept, Algorithms, Pseudo-code, Flowcharts, Strategy for designing algorithms, Tracing an algorithm to depict logic, Specification for converting algorithms into programs.

1.1 Computer Fundamentals:

Introduction to a computer:

Computers are electronic devices that receives input, stores or processes the input as per user instructions and provides output in desired format.

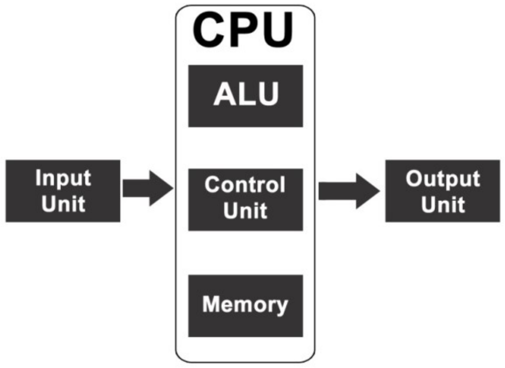
Computers have become an integral part of our lives.

Computers can perform easy and complex tasks repeatedly without committing errors.

Input-Process-Output Model :

Simple workflow in a computer is Input ---> Process ---> Output

Block Diagram of a Computer System:



Input Unit:

The set of devices used to give input to the computer is called an input unit. This unit makes a link between user and computer.

Examples: Keyboard, Mouse, Joystick, Track Ball, Scanner, Stylus, Microphone, etc.

The most commonly used input devices are

Keyboard: Keyboard allows the user to enter inputs by typing the keys present on the keyboard. Most common keyboard contains 104 keys.

Mouse: Mouse allows the user to select elements by pointing and clicking them and to draw and point on the screen of the computer. The mouse is also known as a pointing device because it helps to change the position of the pointer or cursor on the screen.

Scanner: Scanner is an input device that converts documents and images as the digitized images understandable by the computer system.

Joystick: Joystick is an input device consisting of a stick that pivots on a base and reports its angle or direction to the device it is controlling. Joysticks are often used to control video games, and usually have one or more push-buttons whose state can also be read by the computer.

Central Processing Unit (CPU):

It is the brain of the computer that is responsible for controlling and executing program instructions. It is also simply called as the processor. 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 has the following three components:

Memory Unit

Control Unit

ALU(Arithmetic Logic Unit)

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

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

Primary memory/main memory includes RAM and Cache memory. Secondary memory includes hard disks or floppy disks.

Primary memory (RAM): It is a read and write memory. It is a temporary storage device data is erased as soon as the power supply is turned off. It is used as the main memory of a computer system because it is used by the CPU to load instructions temporarily

ROM (Read Only Memory): It is a read-only memory. It is a permanent storage device i.e. the data remains stored even after the power supply has been turned off.It is used to store Basic Input-Output System(BIOS).

Cache memory: Cache memory is very small but very fast memory used to store frequently used data and instructions. Normally it is on CPU or closer to CPU.

Secondary memory: Secondary memory represents the external storage devices that are connected to the computer. They provide a temporary memory source used to store information that is not in use currently. A storage device is either located in the CPU casing of the computer or is connected externally to the computer. The secondary storage devices can be classified as:

Magnetic storage devices: Floppy disk, Hard disk and Magnetic tapes.

Optical storage devices: CD-ROM and DVD-ROM.

Universal serial bus (USB) devices.

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 compute

It communicates with Input/Output devices for transfer of data or results from storage.

ALU (Arithmetic Logic Unit): This unit consists of two sub sections

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

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

Output Unit:

The set of devices that are used to take the output from the computer are called an output unit. This unit is a link between computer and users. The most commonly used output devices are

Monitor: It is a display screen, which shows information in visual form.

Printer: Used to produce a hard copy of the electronic text displayed on the screen in the form of paper sheets that can be used by the end-user.

Types of printers: Dot matrix printer, Inkjet printer, and Laser printer.

Speaker: The speaker is an electro-mechanical transducer that converts an electrical signal into sound.

Plotter: Used to print large documents such as engineering or construction drawings.

Characteristics of a computer:

- Speed: Computers can perform operations at a great speed.
- Accuracy: Computers exhibit a very high degree of accuracy. There is a very less chance of computing error. Errors that may occur are usually due to human errors.
- Reliability: Computers can carry out same type of work repeatedly without throwing up errors.
- Versatility: Computers can carry out a wide range of work ranging from simple data storage to complex continuous astronomical calculations. With the necessary input data and process instructions, computers will be able to generate the accurate output.
- Storage Capacity: Computers can store a very large amount of data compared to traditional papers and books.

Booting:

Starting a computer or a computer-embedded device is called booting. Booting includes the following steps:

- Switching on power supply to power on the computer.

- Loading the operating system into computer's main memory.
- Keeping all applications ready to launch when needed by the user.

The first program(set of instructions) that runs when the computer is switched on is called BIOS or Basic Input Output System. BIOS is a firmware, i.e., it is permanently programmed into the hardware.

There are two types of booting −

- **Cold Booting:** When the system is started by switching on the power supply it is called cold booting. The next step in cold booting is loading of BIOS.
- **Warm Booting:** When the system is already running and needs to be restarted or rebooted, it is called warm booting. Warm booting is faster than cold booting because BIOS is not reloaded.

If a system is already running but needs to be restarted, it is called rebooting.

1.2 Evolution of Computers:

The whole picture of the computer goes back to decades. The first fully electronic computers, introduced in the 1940s, were voluminous devices that required teams of people to handle. In comparison to those machines, today's computers are astounding. They are not only thousands of times more faster, but are also compact.

There are five generations of computers. Each generation is defined by a paramount technological development that changes necessarily how computers operate leading to more compressed, inexpensive, dynamic and efficient machines.

First Generation – Vacuum Tubes (1940 – 1956): These ancient computers utilized vacuum tubes as circuitry and magnetic drums for recollection. As a result they were huge, actually taking up entire rooms and costing resources to run. These were ineffective materials which produce a huge amount of heat, sucked enormous electricity and subsequently generated lot of heat resulting in perpetual breakdowns.

These first generation computers relied on '**machine language**' (which is the most fundamental programming language that can be directly understood by computers). These computers were limited to solving one problem at a time. Input was predicated on punched cards and paper tape. Output emerged on print-outs. The two eminent machines of this era were the **UNIVAC** and **ENIAC** machines – the **UNIVAC** is the first ever commercial computer which was purchased in 1951 by a business named as the US Census Bureau.

Second Generation – Transistors (1956 – 1963): The supersession of vacuum tubes by transistors, visualized the onset of the second generation of computing. Although first invented in 1947, transistors weren't used considerably in computers until the cessation of the 1950s.

They were a huge development over the vacuum tubes.

They were extremely superior to the vacuum tubes, making computers smaller, more expeditious, inexpensive. The electricity consumption reduced to a great extent with the use of transistors but still the heat generated was huge.

They still worked with punched card for input/printouts.

The language emerged from strange binary language to symbolic ('assembly') languages. This meant that the programmers could discover instructions in words. Meanwhile during the same time high caliber programming languages were being developed (early versions of COBOL and FORTRAN). Transistor-driven machines were the first computers to store instructions into their recollections, peregrinating from magnetic drum to magnetic core 'technology'. The anticipatory versions of these machines were created for the atomic energy industry.

Third Generation – Integrated Circuits (1964 – 1971): By this phase, transistors were minimized and were put on silicon chips. This led to a huge improvement in speed and effectiveness of these machines. These were the first computers where users interacted utilizing keyboards and monitors which interfaced with an operating system, a consequential leap up from the punch cards and printouts. This facilitated these machines to run various applications at once utilizing a central program which functioned to monitor memory.

As a result of these advances which again made machines more reasonable and more tiny, a brand new group of users emerged during the '60s.

Fourth Generation – Microprocessors (1972 – 2010): This innovation can be defined in one word: Intel. The chip-maker accomplished the Intel 4004 chip in 1971, which located all components of computer such as CPU, recollection, input/output controls onto a single chip. What overcrowded a room in the 1940s now gets fit in the palm of the hand. The Intel chip contained thousands of unified circuits. The year 1981 saw the first ever computer (IBM) categorically designed for home use and 1984 saw the Macintosh introduced by Apple. Microprocessors even transformed beyond the realm of computers and into an incrementing number of everyday products.

The incremented power of these small computers denoted that they could be linked, establishing networks. Which eventually led to the expansion, birth and rapid evolution of the Internet.

Fifth Generation – Artificial Intelligence (2010 Onwards): Computer devices with artificial potentiality are still in development. AI is an authenticity, made possible by adopting parallel processing and superconductors. Inclining to the future, computers will be thoroughly revolutionized again by quantum computation, molecular and nanotechnology. The essence of fifth generation will be utilizing these technologies to ultimately engender machines which can process and acknowledge natural language, and have efficiency to determine and organise themselves.

1.3 Classification of Computers:

Historically computers were classified according to processor types because development in processor and processing speeds were the developmental benchmarks. Earliest computers used vacuum tubes for processing, were huge and broke down frequently. However, as vacuum tubes were replaced by transistors and then by chips, their size decreased and processing speeds increased manifold.

All modern computers and computing devices use microprocessors whose speeds and storage capacities are skyrocketing day by day. The developmental benchmark for computers is now their size. Computers are now classified on the basis of their use or size −

- Desktop
- Laptop

- Tablet
- Server
- Mainframe
- Supercomputer

Desktop: Desktop computers are personal computers (PCs) designed for use by an individual at a fixed location. IBM was the first to introduce and popularize the use of desktops. A desktop unit typically has a CPU (Central Processing Unit), monitor, keyboard and mouse. Introduction of desktops popularized use of computers among common people as it was compact and affordable.

Riding on the wave of desktop's popularity many software and hardware devices were developed specially for the home or office user. The foremost design consideration here was user friendliness.

Laptop: Despite its huge popularity, desktops gave way to a more compact and portable personal computer called laptop in 2000s. Laptops are also called notebook computers or simply notebooks. Laptops run using batteries and connect to networks using Wi-Fi (Wireless Fidelity) chips. They also have chips for energy efficiency so that they can conserve power whenever possible and have a longer life.

Modern laptops have enough processing power and storage capacity to be used for all office work, website designing, software development and even audio/video editing.

Tablet: After laptops computers were further miniaturized to develop machines that have processing power of a desktop but are small enough to be held in one's palm. Tablets have touch sensitive screen of typically 5 to 10 inches where one finger is used to touch icons and invoke applications.

Keyboard is also displayed virtually whenever required and can be used with touch strokes. Applications that run on tablets are called apps.

Server: Servers are computers with high processing speeds that provide one or more services to other systems on the network. They may or may not have screens attached to them. A group of computers or digital devices connected together to share resources is called a network.

Servers have high processing powers and can handle multiple requests simultaneously. Most commonly found servers on networks include −

- File or storage server
- Game server
- Application server
- Database server
- Mail server
- Print server

Mainframe: Mainframes are computers used by organizations like banks, airlines and railways to handle millions and trillions of online transactions per second. Important features of mainframes are −

- Big in size
- Hundreds times faster than servers.

- Very expensive
- Use proprietary OS provided by the manufacturers
- In-built hardware, software and firmware security features

Supercomputer: Supercomputers are the fastest computers on Earth. They are used for carrying out complex, fast and time intensive calculations for scientific and engineering applications. Supercomputer speed or performance is measured in teraflops, i.e. 1012 floating point operations per second.

Chinese supercomputer Sunway TaihuLight is the world's fastest supercomputer with a rating of 93 petaflops per second, i.e. 93 quadrillion floating point operations per second.

Most common uses of supercomputers include:

- Molecular mapping and research
- Weather forecasting
- Environmental research
- Oil and gas exploration

1.4 Anatomy of Computers:

The following are the parts of a desktop computer.

1. Power Supply: When you plug your power cable into your computer, you are actually plugging into a socket in the power supply unit that has been fitted inside your case. This component is responsible for converting the 240 volt AC mains power to low voltage DC power needed by computer components. The power supply generates +3.3V, +5V, +12V, -5V. These voltages must be constant, right up to the maximum current your system will draw under load.
2. Monitor: Commonly known as a "screen," the monitor gives you a visual display of what your computer is up to. Monitor displays are divided into pixels. The higher the pixel count, the higher the "resolution." Resolutions are measured in Rows x Columns. Common resolution settings are 640 x 480, 800 x 600, 1024 x 768, 1280 x 1024, etc.
3. Motherboard: The motherboard (sometimes called the mainboard) is the main circuit board inside a personal computer. Other vital system components like the central processing unit (CPU) and random access memory (RAM) modules are connected directly to the motherboard via slots or sockets designed specifically for those components. The motherboard also provides a number of expansion slots designed to accommodate add-on cards such as video graphics adapter (VGA) cards and network interface cards (NICs).
4. Central Processing Unit: The Central Processing Unit (CPU) is usually called either a CPU or just a Processor. The CPU is the brain of the system. It executes all the program code from the operating system and the applications the user runs and processing of data.
5. Main Memory or Random Access Memory (RAM): RAM is the short term memory that the computer uses to keep track of what it's doing. If the computer loses power, anything stored in RAM is lost.
6. Storage Device: Computer storage device is any type of hardware that stores data. The most common type of storage device, which nearly all computers have, is a hard drive.

7. Input and Output devices:

   Input Devices: Input device is a hardware device that sends information to the computer.

   Mouse: Mouse is a pointer device.The mouse allows an individual to control a pointer in a graphical user interface (GUI). Using a mouse a user has the ability to perform various functions such as opening a program or file and does not require the user to memorize commands.

   Web Cam: A camera connected to a computer that allows anyone connected to the Internet to view still pictures or motion video of a user.

   Joystick: A computer joystick allows an individual to easily navigate an object in a game such as navigating a plane in a flight simulator.

   Keyboard: One of the main input devices used on a computer, a computer keyboard looks very similar to the keyboards of electric typewriters, with some additional keys.

   Microphone: Sometimes abbreviated as mic, a microphone is a hardware peripheral that allows computer users to input audio into their computers.

   Scanner: Input device that allows a user to take an image and/or text and convert it into a digital file, allowing the computer to read and/or display the scanned object

   Output Devices: Any peripheral that receives and/or displays output from a computer. Below are some examples of different types of output devices commonly found on a computer.

   Monitor: A monitor is a video display screen. Monitor is also called as Visual Display Unit (VDU) or Video Display Terminal (VDT). CRT (Cathode Ray Tube) Monitors are built very similarly to older (tube) television sets. They are heavy, bulky, take up a lot of desk space, and emit radiation. LCD (Liquid Crystal Display) Monitors are thin and flat. They are light, compact, take up very little desk space emit no radiation.

   Speakers: Speakers are the devices used for sending the audio output.

   Printer: A printer is an output device responsible for taking computer data and generating a hard copy of that data.

1.5 Primary Memory:

Memory is required in computers to store data and instructions. Memory is physically organized as a large number of cells that are capable of storing one bit each. Logically they are organized as groups of bits called words that are assigned an address. Data and instructions are accessed through these memory address. The speed with which these memory addresses can be accessed determines the cost of the memory. Faster the memory speed, higher the price.

Memory is of two types – primary and secondary.

The main features of primary memory, which distinguish it from secondary memory are −

- It is accessed directly by the processor
- It is the fastest memory available
- It is volatile, i.e. its contents are lost once power is switched off

As primary memory is expensive, several technologies are developed to optimize its use. The following types of primary memories are available:

RAM: RAM stands for Random Access Memory. The processor accesses all memory addresses directly, irrespective of word length, making storage and retrieval fast. RAM is the fastest memory available and hence most expensive. These two factors imply that RAM is available in very small quantities of up to 1GB. RAM is volatile but my be of any of these two types

1. DRAM (Dynamic RAM): Each memory cell in a DRAM is made of one transistor and one capacitor, which store one bit of data. However, this cell starts losing its charge and hence data stored in less than thousandth of a second. So it needs to be refreshed thousand times a second, which takes up processor time. However, due to small size of each cell, one DRAM can have large number of cells. Primary memory of most of the personal computers is made of DRAM.
2. SRAM (Static RAM): Each cell in SRAM is made of a flip flop that stores one bit. It retains its bit till the power supply is on and doesn't need to be refreshed like DRAM. It also has shorter read-write cycles as compared to DRAM. SRAM is used in specialized applications.

ROM: ROM stands for Read Only Memory. As the name suggests, ROM can only be read by the processor. New data cannot be written into ROM. Data to be stored into ROM is written during the manufacturing phase itself. They contain data that does not need to be altered, like booting sequence of a computer or algorithmic tables for mathematical applications. ROM is slower and hence cheaper than RAM. It retains its data even when power is switched off, i.e. it is non-volatile. ROM cannot be altered the way RAM can be but technologies are available to program these types of ROMs −

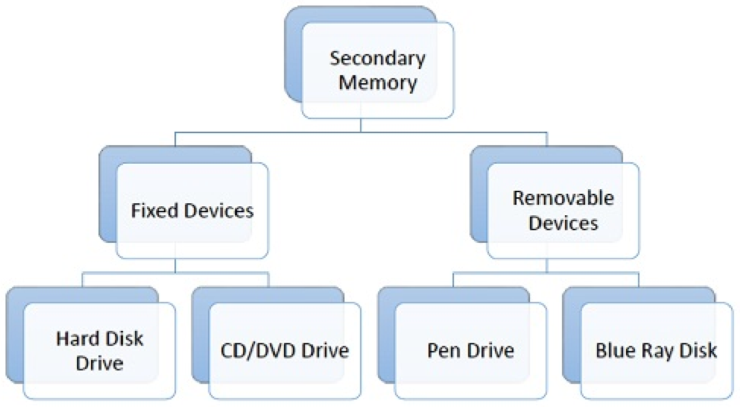
1. PROM (Programmable ROM): PROM can be programmed using a special hardware device called PROM programmer or PROM burner.
2. EPROM (Erasable Programmable ROM): EPROM can be erased and then programmed using special electrical signals or UV rays. EPROMs that can be erased using UV rays are called UVEPROM (Ultra-Violet Erasable Programmable Read Only Memory) and those that can be erased using electrical signals are called EEPROM(Electrically Erasable Programmable Read Only Memory). Handling electric signals is easier and safer than UV rays.

Cache Memory: Cache memory is a small piece of high speed volatile memory available to the processor for fast processing. Cache may be a reserved portion of main memory, another chip on CPU or an independent high speed storage device. Cache memory is usually made of fast speed

SRAMs. The process of keeping some data and instructions in cache memory for faster access is called caching. Caching is done when a set of data or instructions is accessed again and again.

Whenever the processor needs any piece of data or instructions, it checks the cache first. If it is unavailable there, then the main memory and finally secondary memory is accessed. As cache has very high speed, time spent in accessing it every time is negligible compared to time saved if data indeed is in the cache. Finding data or instruction in cache is called cache hit.

1.6 Secondary Memory:



Primary memory also known as processor memory is expensive as well as limited. The faster primary memory is also volatile.

To store large amount of data or programs permanently, a cheaper and permanent memory is needed. Such memory is called secondary memory.

Characteristics of Secondary Memory

These are some characteristics of secondary memory, which distinguish it from primary memory –

- It is non-volatile, i.e. it retains data even when power is switched off
- It is available in large capacities to the tune of terabytes
- It is cheaper as compared to primary memory

Depending on whether secondary memory device is part of CPU or not, there are two types of secondary memory – fixed and removable.

Let us look at some of the secondary memory devices available.

Hard Disk Drive: Hard disk drive is made up of a series of circular disks called platters arranged one over the other almost ½ inches apart around a spindle. Disks are made of non-magnetic material like aluminum alloy and coated with 10-20 nm of magnetic material.

Standard diameter of these disks is 14 inches and they rotate with speeds varying from 4200 rpm (rotations per minute) for personal computers to 15000 rpm for servers. Data is stored by magnetizing or demagnetizing the magnetic coating. A magnetic reader arm is used to read data from and write data to the disks. A typical modern HDD has capacity in terabytes (TB).

CD Drive: CD stands for Compact Disk. CDs are circular disks that use optical rays, usually lasers, to read and write data. They are very cheap as you can get 700 MB of storage space for less than 10 rupees. CDs are inserted in CD drives which are generally built into CPU cabinet. They are portable as you can eject the drive, remove the CD and carry it with you. There are three types of CDs −

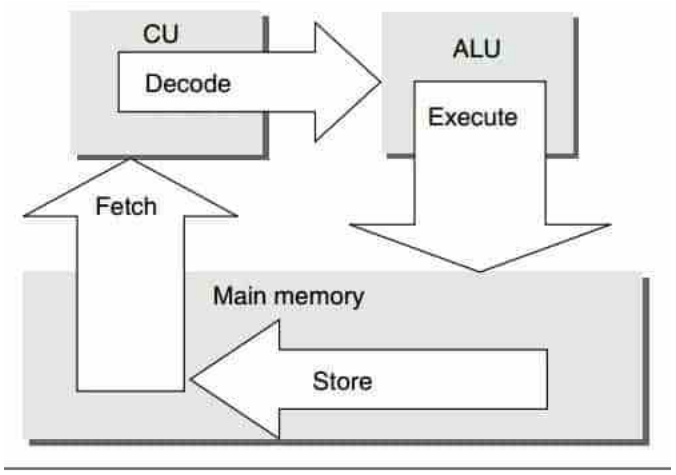
- CD-ROM (Compact Disk – Read Only Memory): The data on these CDs are recorded by the manufacturer. Proprietary Software, audio or video are released on CD-ROMs.
- CD-R (Compact Disk – Recordable): Data can be written by the user once on the CD-R. It cannot be deleted or modified later.
- CD-RW (Compact Disk – Rewritable): Data can be written and deleted on these optical disks again and again.

DVD Drive: DVD stands for Digital Video Display. DVD are optical devices that can store 15 times the data held by CDs. They are usually used to store rich multimedia files that need high storage capacity. DVDs also come in three varieties – read only, recordable and rewritable.

Pen Drive: Pen drive is a portable memory device that uses solid state memory rather than magnetic fields or lasers to record data. It uses a technology similar to RAM, except that it is nonvolatile. It is also called USB drive, key drive or flash memory.

Blu Ray Disk: Blu Ray Disk (BD) is an optical storage media used to store high definition (HD) video and other multimedia filed. BD uses shorter wavelength laser as compared to CD/DVD. This enables writing arm to focus more tightly on the disk and hence pack in more data. BDs can store up to 128 GB data.

1.7 Operational Overview of CPU:



Any processing performed by central processing unit is directed by the instructions. The processing involved in executing a single instruction is called an instruction cycle. The four steps which the CPU carries out for each machine language instruction are **fetch**, **decode**, **execute**, and **store**.

The steps involved in the instruction cycle while executing a program:

1. The Program Counter (PC) is the register that keeps track of what instruction has to be executed next. At the first step, the instruction is fetched from main memory and loaded

into Instruction Register (IR), whose address is specified by PC register. Immediately the PC is incremented so that it points to the next instruction in the program.
2. Once in IR, the instruction is decoded to determine the actions needed for its execution.
3. The control unit then issues the sequence of control signals that enables execution of the instruction. Data needed to be processed by the instructions are either fetched from a register from RAM through an address register.
4. The result of the instruction is stored (written) to either a register or a memory location.

Every instruction of a program will follow the same steps. This will continue until there is no more instruction in the program or the computer is turned off or some sort of unrecoverable error occurs.

A register is a single, permanent storage location within the CPU used for a particular, defined purpose. CPU contains several important registers such as

1. The program counter(PC) register holds the address of the instruction to be executed next.
2. The instruction register (IR) holds the actual instruction being executed currently by the computer. To access data in memory, CPU makes use of two internal registers:
3. The memory address register (MAR) holds the address of a memory location.
4. The memory data register (MDR), sometimes known as the memory buffer register, will hold a data value that is being stored to or retrieved from the memory location currently addressed by the memory address register.

1.8 Introduction to Operating System:

An operating system may be defined as a system software which acts as an intermediary between the user and the hardware, an interface which isolates the user from the details of the hardware implementation.

It consists of a set of specialized software modules that makes computing resources (hardware and software) available to users. Thus, the computer system is easier to use with the operating system in place than without it.

Some of the popular operating systems used are: Mac, MS Windows, Linux, Solaris, etc.

The common functions of an operating system includes:

1. Process(or) management The process abstraction is a fundamental mechanism implemented by the operating system for management of the execution of programs. A process is basically a program in execution. The operating system decides which process gets to run, for how long and perhaps at what priority or level of importance.
2. Memory management Operating system is responsible for keeping track of which parts of the memory are currently being used and by whom. It organizes and addresses memory handle requests to allocate memory, frees up memory no longer being used, and rearranges memory to maximize the useful amount. Often several programs may be in

memory at the same time. The operating system selects processes that are to be placed in memory, where they are to be placed, and how much memory is to be given to each.

3. Device management The operating system allocates the various devices to the processes and initiates the I/O operation. It also controls and schedules accesses to the input/output devices among the processes.
4. File management A file is just a sequence of bytes. Files are storage areas for programs, source codes, data, documents etc. The operating system keeps track of every file in the system, including data files, program files, compilers, and applications. The file system is an operating system module that allows users and programs to create, delete, modify, open, close, and apply other operations to various types of files. It also allows users to give names to files, to organize the files hierarchically into directories, to protect files, and to access those files using the various file operations.

When a computer boots up, it goes through some initialization functions, such as checking the memory.

It then loads the kernel and switches control to it. The kernel then starts up all the processes needed to communicate with the user and the rest of the environment. The user interface is the portion of the operating system that users interact with directly. Operating systems such as MS-DOS and early versions of UNIX accepted only typed-in text commands. Now most operating systems provide users a graphical user interface for their interactions with the system. Operating systems such as Microsoft Windows, Solaris and Linux allow the user to interact with the operating system through icons, menus, keyboard and mouse movements. The user interface and way of interactions vary widely from one operating system to another.

Loading an Operating System

In some digital devices like controllers of small appliances, hand-held devices and video game console, the operating system is relatively simple and small and is stored in ROM (Read-Only Memory). The operating system is also present in a ROM for systems such as industrial controllers and petrol-filling equipment. In such a system, it gains immediate control of the processor, the moment it is turned on.

In personal computer, the operating system is usually stored on hard disk. Because size of the operating system is large enough, it cannot be placed entirely in RAM. The kernel, the core part of the operating system, is loaded into RAM at start-up and is always present in memory. Other parts of the operating system are loaded into RAM as and when required. Generally no operating system is resident in a new computer. The operating system is usually sold on a CD or DVD media and has to be permanently transferred from a CD or DVD media to the hard disk by expanding compressed files and initializing the whole system for use.

Booting is the general term for the process that a computer or other digital device follows from the instant it is turned on until the operating system is finally loaded and ready for use.

The Basic Input Output System (BIOS) is a small set of instructions stored on a ROM that is executed when the computer is turned on. When the computer is switched on, the ROM circuitry receives power and begins the boot process. At first, an address is automatically loaded into the Program Counter (PC) register. This is done by hardware circuitry. The address given is the location of the first executable instruction of the BIOS.

The code in the BIOS runs a series of tests called the POST (Power On Self Test) to make sure that system devices such as main memory, monitor, keyboard, the input/output devices are connected and functional. During POST, the BIOS compares the system configuration data obtained from POST with the system information stored on a Complementary Metal-Oxide Semiconductor (CMOS) memory chip located on the motherboard.

The BIOS also sets various parameters such as the organization of the disk drive, using information stored in a CMOS chip. This CMOS chip gets updated whenever new system components are added and contains the latest information about system components.

The BIOS then loads only one block of data, called the Master Boot Record, from a specific and fixed place (the very first sector at cylinder 0, head 0, and sector 1) of the bootable device and is placed at a specific and fixed place of main memory. The master boot record is of 512 bytes in size and contains machine code instructions, called a bootstrap loader. Then the boot loader program starts the process of loading the OS and transfers control to the OS itself which completes the process.

Cold boot describes the process of starting the computer and loading its operating system by turning the power on. If the computer is running, one can carry out cold boot by first switching it off and then back on.

Warm boot describes the process of restarting the computer and loading its operating system again without switching it off after it has already been running.

1.9 Introduction to Programming:

Programs and Programming:

A computer can neither think nor make a decision on its own.

It is not possible for any computer to independently analyze a given data and find a solution on its own.

Computer needs a program which will convey what is to be done.

A program is a set of logically related instructions that is arranged in a sequence that directs the computer in solving a problem.

The process of writing the program is called programming.

Programming is a necessary and critical step in data processing. An incorrect program delivers results that cannot be used.

There are two ways by which one can acquire a program:

- Purchase an existing program (packaged software).
- Prepare a new program from scratch (customized software).

Computer Software:

Computer software can be broadly classified into two categories:

System software.

Application software.

System Software: System software is a collection of programs that interfaces with the hardware. Some common categories of system software are described as follows:

Language translator: It is a system software that transforms a computer program written by a user into a form that can be understood by the machine.

Operating system (OS): This is the most important system software that is required to operate a computer system. An operating system manages the computer's resources effectively, takes care of scheduling multiple jobs for execution, and manages the flow of data and instructions between the input/output units and the main memory. An operating system has become a part of computer software with the advent of the third generation computers. Since then a number of operating systems have been developed and some have undergone several revisions and modifications to achieve better utilization of computer resources. Advances in computer hardware have helped in the development of more efficient operating systems.

Application Software: Application software is written to enable the computer to solve a specific data processing task. There are two categories of application software:

Pre-written software packages.

User application programs.

A number of powerful application software packages that do not require significant programming knowledge have been developed. These are easy to learn and use compared to . Although these packages can perform many general and special functions, there are applications where these packages are found to be inadequate. In such cases, user application programs are written to meet the exact requirements.

A user application program may be written using one of these packages or a programming language. The most important categories of software packages available are

Database management software

Spreadsheet software

Word processing, Desktop Publishing (DTP), and presentation software

Multimedia software

Data communication software

Statistical and operational research software

1.10 Programming Languages:

To write a computer program, a standard programming language is used.

A **programming language** is composed of a set of instructions in a language understandable to the programmer and recognizable by a computer. Programming languages can be classified as **high-level**, **middle-level**, and **low-level**.

High-level languages such as BASIC, COBOL (Common Business Oriented Programming Language), and FORTRAN (Formula Translation Language) are used to write application programs.

A **middle-level** language such as C is used for writing application and system programs.

A **low-level** language such as the assembly language is mostly used to write system programs.

Low-level programming languages were the first category of programming languages to evolve. Gradually, high-level and middle-level programming languages were developed and put to use.

System Programming Languages:

System programs or softwares are designed to make the computers easier to use. An example of system software is an operating system consisting of many other programs that control input/output devices, memory, processor, schedule the execution of multiple tasks, etc.

To write an operating system program, the programmer needs instructions to control the computer's circuitry as well as manage the resources of the computer.

For example, instructions that move data from one location of storage to a register of the processor are required. Assembly language, which has a one-to-one correspondence with machine code, was the normal choice for writing system programs like operating systems.But, today C is widely used to develop system software.

Application Programming Languages:

There are two main categories of application programs:

1. Business programs
2. Scientific application programs.

Application programs are designed for specific computer applications, such as payroll processing and inventory control. To write programs for payroll processing or other such applications, the programmer does not need to control the basic circuitry of a computer. Instead, the programmer needs instructions that make it easy to input data, produce output, perform calculations, and store and retrieve data.

Programming languages suitable for such application programs have the appropriate instructions. Most programming languages are designed to be good for one category of applications but not necessarily for the other, although there are some general-purpose languages that support both types.

Business applications are characterized by processing of large inputs and high-volume data storage and retrieval but call for simple calculations. Languages which are suitable for business program development must support high-volume input, output, and storage but do not need to support complex calculations. On the other hand, programming languages designed for writing scientific programs contain very powerful instructions for calculations but have poor instructions for input, output, etc.

Among the traditionally used programming languages, COBOL is more suitable for business applications whereas FORTRAN is more suitable for scientific applications.

Low Level Languages:

A low-level computer programming language is one that is closer to the native language of the computer, which is 1's and 0's. Machine languages and assembly languages are referred to as low-level languages.

Machine language: This is a sequence of instructions written in the form of binary numbers consisting of 1's and 0's to which the computer responds directly.The first part is the command or operation code (OPCODE) that conveys to the computer what function has to be performed by the instruction. All computers have operation codes for functions such as adding, subtracting and moving. The second part of the instruction either specifies that the operand contains data on which the operation has to be performed or it specifies that the operand contains a location, the contents of which have to be subjected to the operation.

Computer programming languages also have a generation classification based on the level of interaction with the machine. Machine language is considered to be the first generation language (1GL).

Advantage of machine language: The CPU directly understands machine instructions, and hence no translation is required. Therefore, the computer directly starts executing the machine language instructions, and it takes less execution time.

Disadvantages of machine language:

1. Difficult to use: It is difficult to understand and develop a program using machine language.
2. Machine dependent: The programmer has to remember machine characteristics while preparing a program. As the internal design of the computer is different across types, which in turn is determined by the actual design or construction of the ALU, CU, and size of the word length of the memory unit, the machine language also varies from one type of computer to another.
3. Error prone: It is hard to understand and remember the various combinations of 1's and 0's representing data and instructions.
4. Difficult to debug and modify: Checking machine instructions to locate errors are as tedious as writing the instructions. Further, modifying such a program is highly problematic.

Assembly language: When symbols such as letters, digits, or special characters are employed for the operation, operand, and other parts of the instruction code, the representation is called an assembly language instruction. Such representations are known as mnemonic codes; they are used instead of binary codes. Assembly language is second generation language (2GL).Each computer has its own assembly language that is dependent upon the internal architecture of the processor.

An assembly language program cannot be executed by a machine directly as it is not in a binary machine language form. An assembler is needed to translate an assembly language program into the object code, which can then be executed by the machine.

Advantage of assembly language: Writing a program in assembly language is more convenient than writing one in machine language. Instead of binary sequence, as in machine language, a program in assembly language is written in the form of symbolic instructions. This gives the assembly language program improved readability.

Disadvantages of assembly language:

1. Assembly language is specific to a particular machine architecture, i.e., machine dependent and not portable. This means that assembly language programs written for one processor will not work on a different processor if it is architecturally different.
2. Programming is difficult and time consuming.
3. The programmer should know all about the logical structure of the computer.

High Level Languages:

The time and cost of creating machine and assembly language programs is quite high. This motivated the development of high-level languages.

High-level programming languages such as COBOL, FORTRAN, and BASIC have instructions that are similar to human languages and have a set grammar that makes it easy for a programmer to write programs and identify and correct errors in them.

Advantages of high-level programming languages:

1. Readability: Programs written in these languages are more readable than those written in assembly and machine languages.
2. Portability: High-level programming languages can be run on different machines with little or no change.
3. Easy debugging: Errors can be easily detected and removed.
4. Ease of development: Since the commands of these programming languages are closer to the English language, software can be developed with ease.

High-level languages are also called third generation languages (3GLs).

1.11 Compiler:

Compiler translates the program written in a high-level language to machine level language.

The compiler takes the high-level language program as input and produces the machine language code as output for the machine to execute the program .

During the process of translation, the compiler reads the source program statement- wise and checks for syntax errors. In case of any error, the computer generates a printout of the same. This action is known as diagnostics. There is another type of software that also does translation. This is called an interpreter.

The compiler and interpreter have different approaches to translation.

Compiler	Interpreter
Compiler takes entire program as input	Interpreter takes single instruction as input
Intermediate object code is generated	No Intermediate object code is generated
Conditional control statements execute faster	Conditional control statements are slower in execution
Memory Requirement is More(Since Object code is generated)	Memory Requirement is Less
Program need not be compiled every time	Every time higher level program is converted into lower level program
Errors are displayed after entire program is checked	Errors are displayed for every instruction interpreted (if any)
Example : C compiler	Example : Python

Compiling and Executing High Level Language Programs:

The compiling process consists of the following phases

Lexical analysis phase.

Syntax analysis phase.

Semantic analysis phase.

Code generation phase.

In the lexical analysis phase, successive lines of a program are broken into individual lexical items namely, identifier, operator delimiter, etc. and attaches a type tag to each of these. A symbol table is constructed for each identifier and the symbol table is used later to allocate memory to each variable. The lexical analysis phase uses the precise description of the source programming language. A source language is described using lexical rules, syntax rules, and semantic rules which defines all the valid statements of the language. Lexical rules specify the valid syntactic elements or words of the language generally using a notation known as BNF (Backus Naur Form) grammar.

In the syntax analysis phase (also called as syntax analysis or parsing), expressions, declarations, and other statements are identified by using the results of lexical analysis. Syntax analysis is done by using techniques based on formal grammar of the programming language.

In the semantic analysis phase, the syntactic units recognized by the syntax analyzer are processed. An intermediate representation of the final machine language code is produced.

In the code generation phase, optimizations to reduce the length of machine language program is carried out. The output of the code generator is a machine level language program for the specified computer.

Linking and Loading: If a subprogram library is used or if some subroutines are separately translated and compiled, a final linking and loading step is needed to produce the complete machine language program in an executable form.

If subroutines were compiled separately, then the address allocation of the resulting machine language instructions would not be final. When all routines are connected and placed together in the main memory, suitable memory addresses are allocated. The linker's job is to find the correct main memory locations of the final executable program. The loader then places the executable program in memory at its correct address.

Execution of a program written in high-level language involves the following steps:

1. Translation of the program resulting in the object program.
2. Linking of the translated program with other object programs needed for execution, thereby resulting in a binary program.
3. Relocation of the program to execute from the specific memory area allocated to it.
4. Loading of the program in the memory for the purpose of execution.

1.12 Loader:

Loading means physically placing the machine instructions and data into main memory, also known as primary storage area.

A loader is a system program that accepts object programs and prepares them for execution and initiates the execution.

The functions performed by the loader are :

1. Assignment of load-time storage area to the program
2. Loading of program into assigned area
3. Relocation of program to execute properly from its load time storage area
4. Linking of programs with one another

An absolute loader places these items into the precise locations indicated in the machine language program.

A relocating loader may load a program at various places in primary storage depending on the availability of primary storage area at the time of loading.

A program may be relocated dynamically with the help of a relocating register. The base address of the program in primary storage is placed in the relocating register. The contents of the relocation register are added to each address developed by a running program. The user is able to execute the program as if it begins at location zero. At execution time, as the program runs, all address references involve the relocation register. This allows the program to reside in memory locations other than those for which it was translated to occupy.

Linking Loader and Linkage Editor:

Loading means physically placing the machine instructions and data into main memory, also known as primary storage area.

A loader is a system program that accepts object programs and prepares them for execution and initiates the execution.

The functions performed by the loader are :

1. Assignment of load-time storage area to the program
2. Loading of program into assigned area
3. Relocation of program to execute properly from its load time storage area
4. Linking of programs with one another

An absolute loader places these items into the precise locations indicated in the machine language program.

A relocating loader may load a program at various places in primary storage depending on the availability of primary storage area at the time of loading.

A program may be relocated dynamically with the help of a relocating register. The base address of the program in primary storage is placed in the relocating register. The contents of the relocation register are added to each address developed by a running program. The user is able to execute the program as if it begins at location zero. At execution time, as the program runs, all address references involve the relocation register. This allows the program to reside in memory locations other than those for which it was translated to occupy.

1.13 Program Execution:

The primary memory of a computer, also called the Random Access Memory, is divided into units known as **words**.

Depending on the processor architecture, a word of memory may be two, four, or even eight bytes in size.

Each word of the primary memory is identified using an unique number called as **address**.

CPU can randomly access any of these words using its address.

When a program is compiled and linked, **each instruction and each item of data is assigned an address**. At execution time, the CPU **fetches** instructions and data from these addresses.

The **program counter (PC)**, is a CPU register that holds the address of the next instruction to be executed in a program. In the beginning, the PC holds the address of the **zeroth instruction** of the program. The CPU fetches and then executes the instruction found at this address. The PC is meanwhile incremented to the address of the next instruction in the program. Having executed one instruction, the CPU goes back to look up the PC where it finds the address of the next instruction in the program. Note this instruction may not necessarily be in the next memory location. It could be at quite a different address. Program execution proceeds in this way **until the CPU has processed the last instruction**

1.14 Fourth Generation Programming Languages:

The Fourth Generation Language is a non-procedural language that allows the user to simply specify what the output should be without describing how data should be processed to produce the result.

Most of the third generation languages are procedural languages. That is, the programmer must specify the steps of the procedure the computer has to follow in a program. By contrast, most of the fourth generation languages are nonprocedural languages. The programmer does not have to give the details of the procedure in the program, but specify, instead, what is wanted.

Major fourth generation languages are used to get information from files and databases. These fourth generation languages contain a query language, which is used to answer queries or questions with data from a database.

Other fourth generation languages are used to design screens for data input and output and for menus. These languages contain certain types of programs called screen painters. The programmer designs the screen to look as desired and, therefore, it can be said that the programmer paints the screen using the screen painter program.

Fourth generation languages are mostly machine independent. Usually they can be used on more than one type of computers. They are mostly used for office automation or business applications, and not for scientific programs.

Advantages of 4GLs:

1. Programming productivity is increased. One line of a 4GL code is equivalent to several lines of a 3GL code.
2. Software development is faster.
3. Program maintenance is easier.
4. End users can often develop their own applications.
5. Programs developed in 4GLs are more portable than those developed in other generation languages.
6. Documentation is of improved order because most 4GLs are self-documenting.

3GL vs. 4GL

3GL	4GL
Meant for use by professional programmers	May be used by nonprofessional programmers as well as by professional programmers.
Requires specifications of how to perform a task	Requires specifications of what task to perform.
All alternatives must be specified	System determines how to perform the task. Default alternatives are built in. User need not specify these alternatives.
Memory requirement is more (since object code is generated)	Memory requirement is less
Execution time is less	Execution time is even lesser as many bultin

3GL	4GL
	optimizations are done.
Requires large number of procedural instructions code may be difficult to read, understand, and maintain by the user	Requires fewer instructions.

1.15 Fifth Generation Languages:

Natural languages represent the next step in the development of programming languages belonging to fifth generation languages.

Natural language is similar to query language(4GL), with one difference: it eliminates the need for the user or programmer to learn a specific vocabulary, grammar, or syntax.

The text of a natural language statement resembles human speech closely.

Natural language takes the user one step further away from having to deal directly and in detail with computer hardware and software.

These languages are also designed to make the computer smarter—that is, to simulate the human learning process.

A lot of research and development is still happening in creating the best 5th generation languages.

1.16 Classification of Programming Languages:

Programming languages can be classified into the following categories:

Procedural languages.

Problem-Oriented languages.

Non-procedural languages.

Procedural Languages:

Procedural languages are further classified as the following:

Algorithmic languages.

Object oriented languages.

Scripting languages.

Algorithmic languages: These are high-level languages designed for forming convenient expression of procedures, used in the solution of a wide class of problems. In this language, the programmer must specify the steps the computer has to follow while executing a program. Some of languages that fall in the category are C, COBOL, and FORTRAN.

Object-oriented language: The basic philosophy of object oriented programming is to deal with objects rather than functions or subroutines as in strictly algorithmic languages. In a conventional programming language, data and subroutines or functions are separate. In object oriented programming, subroutines as well as data are locally defined in objects.

The most important object-oriented programming features are abstraction

encapsulation and data hiding

polymorphism

inheritance

reusable code

C++, JAVA, SMALLTALK, etc. are examples of object oriented languages.

Scripting languages: These languages assume that a collection of useful programs, each performing a task, already exists. A scripting language has facilities to combine these components to perform a complex task. One of the earliest scripting languages is the UNIX shell scripting language. VB Script, Perl, Action Script, PHP, etc are few examples of scripting languages.

Problem-oriented Languages: These are high-level languages designed for developing a convenient expression of a given class of problems.

Non-procedural languages:

Non-procedural languages are further classified as the following:

Functional (applicative) languages.

Logic based programming languages.

Functional (applicative) languages: These functional languages solve a problem by applying a set of functions to the initial variables in specific ways to get the answer. The functional programming style relies on the idea of function application rather than on the notion of variables and assignments. A program written in a functional language consists of function calls together with arguments to functions. LISP, ML, etc. are examples of functional languages.

Logic-based programming language: A logic program is expressed as a set of atomic sentences, known as fact, and horn clauses, such as if-then rules. A query is then posed. The execution of the program begins and the system tries to find out if the answer to the query is true or false for the given facts and rules. PROLOG is a classic example for logic-based programming languages.

1.17 Introduction to Structured Programming:

In 1968, computer scientist Edsger Dijkstra of Netherlands published a letter to the editor in the journal of the Association of Computing Machinery with the title 'Go To statement considered harmful'. goto is a command available in most programming languages to transfer a control to a particular statement.

A better way of programming and a systematic way to organize programs is called as structured programming. There is no standard definition of structured programs available but it is often thought to be programming without the use of a goto statement. Indeed, structured programming does discourage the frequent use of goto but there is more to it than that.

Structured programming is:

1. concerned with improving the programming process through better organization of programs and better programming notation to facilitate correct and clear description of data and control structure.
2. concerned with improved programming languages and organized programming techniques which should be understandable and therefore, more easily modifiable and suitable for documentation.
3. more economical to run because good organization and notation make it easier for an optimizing compiler to understand the program logic.
4. more correct and therefore more easily debugged, because general correctness theorems dealing with structures can be applied to proving the correctness of programs.

Structured programming can be defined as a

- top–down analysis for program solving
- modularization for program structure and organization
- structured code for individual modules

Top-Down Analysis:

A program is a collection of instructions in a particular language that is prepared to solve a specific problem. For larger programs, developing a solution can be very complicated.

Top-down analysis is a method of problem solving and problem analysis. The idea of top-down analysis is to subdivide a large problem into several smaller tasks or parts for ease of analysis. Top-down analysis, therefore, simplifies or reduces the complexity of the process of problem solving.

It is not limited by the type of program. Top-down analysis is a general method for attending to any problem.

There are two essential ideas in top-down analysis:

1. Subdivision of a problem
2. Hierarchy of tasks

Subdivision of a problem means breaking a big problem into two or more smaller problems. Top-down analysis does not simply divide a problem into two or more smaller problems. Further each of these smaller problems is further subdivided creating a hierarchy of tasks, from one level to the next, until no further break up is possible.

The four basic steps to top-down analysis are as follows:

Step 1: Define the complete scope of the problem to determine the basic requirement for its solution. Three factors must be considered in the definition of a programming problem.

1. Input: What data is required to be processed by the program?
2. Process: What must be done with the input data? What type of processing is required?
3. Output: What information should the program produce? In what form should it be presented?

Step 2: Based on the definition of the problem, divide the problem into two or more separate parts.

Step 3: Carefully define the scope of each of these separate tasks and subdivide them further, if necessary, into two or more smaller tasks.

Step 4: Repeat step 3. Every step at the lowest level describes a simple task, which cannot be broken further.

Modular Programming:

Modular program is a program that is divided into logically independent smaller sections, which can be written separately. These sections, being separate and independent units, are called modules.

1. A module consists of a series of program instructions or statements in some programming language.
2. A module is clearly terminated by some special markers required by the syntax of the language. For example, a BASIC language subroutine is terminated by the return statement.
3. A module as a whole has a unique name.
4. A module has only one entry point to which control is transferred from the outside and only one exit point from which control is returned to the calling module.

Advantages of modular programming:

1. Complex programs may be divided into simpler and more manageable elements.
2. Simultaneous coding of different modules by several programmers is possible.
3. A library of modules may be created, and these modules may be used in other programs as and when needed.
4. The location of program errors may be traced to a particular module; thus, debugging and maintenance may be simplified.

Structured Code:

After the top-down analysis and design of the modular structure, the third and final phase of structured programming involves the use of structured code. Structured programming is a style of coding, i.e., writing a program that produces a well-organized module. A high-level language supports several control statements, also called structured control statements or structured code, to produce a well-organized structured module.

These control statements represent a conditional and repetitive type of executions. Each programming language has a different syntax for these statements. The unconditional branching statements like goto should be avoided while creating a structured programming code.

Process of Programming:

Programmer has to do the following for creating a working program :

1. Understand the problem to be solved
2. Think and design the solution logic
3. Write the program in the chosen programming language
4. Translate the program to machine code
5. Test the program with sample data (Rectify errors if any)
6. Put the program into operation

The first job of the programmer is to understand the problem. To do that the requirements of the problem should be clearly defined. And for this, the programmer may have to interact with the user to know the needs of the user. Thus this phase of the job determines the 'what to' of the task. The next job is to develop the logic of solving the problem. Different solution logics are designed and the order in which these are to be used in the program are defined. Hence, this phase of the job specifies the 'how to' of the task.

Once the logics are developed, the third phase of the job is to write the program using a chosen programming language. The rules of the programming language have to be observed while writing the program instructions. The computer recognizes and works with 1's and 0's. Hence program instructions have to be converted to 1's and 0's for the computer to execute it. Thus, after the program is written, it is translated to the machine code, which is in 1's and 0's with the help of a translating program. Now, the program is tested with dummy data. Errors in the programming logic are detected during this phase and are removed by making necessary changes in either the logic or the program instructions. The last phase is to make the program operational. This means the program is put to actual use. Errors occurring in this phase are rectified to finally make the program work to the user's satisfaction.

1.18 Introduction to Algorithms and Flowcharts:

Computer scientist Niklaus Wirth stated that

Program = Algorithms + Data.

An algorithm is a part of the plan for the computer program. An algorithm is 'an effective procedure for solving a problem in a finite number of steps'. It is effective, which means that an answer is found and it has a finite number of steps. A well- designed algorithm is also guaranteed to terminate.

An Algorithm is independent of the programming language. An Algorithm is the core logic to solve a given problem.

Algorithm can be defined as "a sequence of steps to be performed for getting the desired output for a given input."

The sequence of steps of an algorithm can be stated in human readable English statements or pseudocode (meaning - a notation resembling a simplified programming language).

There may be more than one way to solve a problem, hence there may be more than one algorithms for a given problem.

Before attempting to write an algorithm, one should find out what the expected inputs and outputs are for the given problem.

The properties of an algorithm are:

1. Input: Algorithm should be accepting 0 or more inputs supplied externally.
2. Output: Algorithm should be generating at least one output.
3. Definiteness: Each step of an algorithm must be precisely defined. Meaning the step should perform a clearly defined task without much complication.
4. Finiteness: An algorithm must always terminate after a finite number of steps.
5. Effectiveness: The efficiency of the steps and the accuracy of the output determine the effectiveness of the algorithm.
6. Correctness: Each step of the algorithm must generate a correct output.

Different Ways of Stating Algorithms:

Algorithms may be represented in various ways. There are four ways of stating algorithms. These are as follows:

1. Step-form
2. Pseudo-code
3. Flowchart
4. Nassi-Schneiderman

In the step form representation, the procedure of solving a problem is stated with written statements. Each statement solves a part of the problem and these together complete the solution. The step-form uses just normal language to define each procedure. Every statement, that defines an action, is logically related to the preceding statement. This algorithm has been discussed in the following section with the help of an example.

The pseudocode is a written form representation of the algorithm. However, it differs from the step form as it uses a restricted vocabulary to define its action of solving the problem. One problem with human language is that it can seem to be imprecise. But the pseudo-code, which is in human language, tends toward more precision by using a limited vocabulary. It is similar to a 3GL, and for many programmers and program designers it is the preferred way to state algorithms and program specifications. Although there is no standard for pseudo-code, it is generally quite easy to read and use. For instance, a sample pseudo-code is written as follows:

```
dowhile kettle_empty
        Add_Water_To_Kettle
end dowhile
```

As can be seen, it is a precise statement of a while loop.

Flowchart and Nassi-Schneiderman are graphically oriented representation forms. They use symbols and language to represent sequence, decision, and repetition actions.

Step-Form:

An algorithm shows these three features:

1. Sequence (also known as process)
2. Decision (also known as selection)
3. Repetition (also known as iteration or looping)

Therefore, an algorithm can be stated using three basic constructs: sequence, decision, and repetition.

Sequence: Sequence means that each step or process in the algorithm is executed in the specified order.

Decision: The decision constructs— if ... then, if ... then ... else ... In algorithms the outcome of a decision is either true or false; there is no state in between. The outcome of the decision is based on some condition that can only result in a true or false value.

Repetition: The repetition constructs— repeat and while. Repetition can be implemented using constructs like the repeat..until loop, while loop, and if.. then .. goto .. loop. The Repeat loop is used to iterate or repeat a process or sequence of processes until some condition becomes true. Repeat loop does some processing before testing the state of the proposition.

Termination: Algorithms might also include procedures that could run forever without stopping. Such a procedure has been called a computational method by Knuth or calculation procedure by Kleene. However, Kleene notes that such a method must eventually exhibit 'some object.' Minsky (1967) makes the observation that, if an algorithm has not terminated, then how can the following question be answered: "Will it terminate with the correct answer?" Thus the answer is: undecidable. It can never be known, nor can the designer do an analysis beforehand to find it out. The analysis of algorithms for their likelihood of termination is called termination analysis.

Correctness: The prepared algorithm needs to be verified for its correctness. Correctness means how easily its logic can be argued to meet the algorithm's primary goal. This requires the algorithm to be made in such a way that all the elements in it are traceable to the requirements. Correctness requires that all the components like the data structures, modules, external interfaces, and module interconnections are completely specified. In other words, correctness is the degree to which an algorithm performs its specified function. The most common measure of correctness is defects per Kilo Lines of Code (KLOC) that implements the algorithm, where the defect is defined as the verified lack of conformance to requirements.

Introduction to Flow-Charts:

A flowchart provides appropriate steps to be followed in order to arrive at the solution to a problem. It is a program design tool which is used before writing the actual program. Flowcharts are generally developed in the early stages of formulating computer solutions. A flowchart comprises a set of various standard shaped boxes that are interconnected by flow lines. Flow lines have arrows to indicate the direction of the flow of control between the boxes.

The activity to be performed is written within the boxes in English. In addition, there are connector symbols that are used to indicate that the flow of control continues elsewhere, for example, the next page.

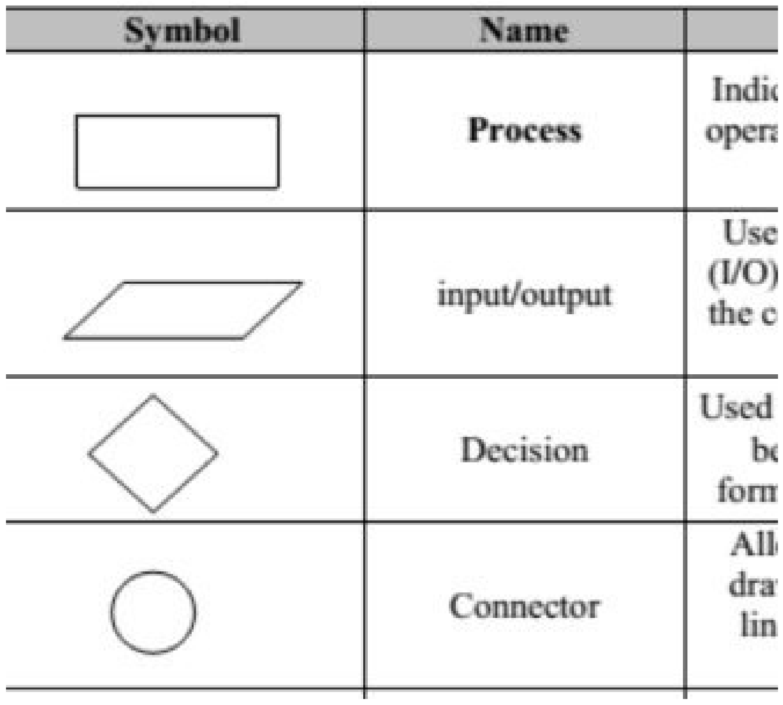
Flowcharts facilitate communication between programmers and business persons. These flowcharts play a vital role in the programming of a problem and are quite helpful in understanding the logic of complicated and lengthy problems. Once the flowchart is drawn, it becomes easy to write the program in any high-level language. Often flowcharts are helpful in explaining the program to others. Hence, a flowchart is a must for better documentation of a complex program.

Standards for flowcharts:

The following standards should be adhered to while drawing flow charts.

1. Flowcharts must be drawn on white, unlined 81/2 × 11 paper, on one side only.
2. Flowcharts start on the top of the page and flow down and to the right.
3. Only standard flowcharting symbols should be used. However, some special symbols can also be developed when required.
4. A template to draw the final version of the flowchart should be used.
5. The contents of each symbol should be printed legibly.
6. English should be used in flowcharts, not programming language.
7. The flowchart for each subroutine, if any, must appear on a separate page. Each subroutine begins with a terminal symbol with the subroutine name and a terminal symbol labeled return at the end.
8. Draw arrows between symbols with a straight edge and use arrowheads to indicate the direction of the logic flow.

Standard Flow Chart Symbols:



Symbol	Name	
▭	**Process**	Indi opera
▱	input/output	Use (I/O) the c
◇	Decision	Used be form
○	Connector	All dra lin

Guideliness for Flowchart:

The following are some guidelines in flowcharting

- In drawing a proper flowchart, all necessary requirements should be listed out in a logical order.
- There should be a logical start and stop to the flowchart.
- The flowchart should be clear, neat, and easy to follow. There should be no ambiguity in understanding the flowchart.
- The usual direction of the flow of a procedure or system is from left to right or top to bottom.
- Only one flow line should emerge from a process symbol.
- Only one flow line should enter a decision symbol, but two or three flow lines, one for each possible answer, can leave the decision symbol.
- Only one flow line is used in conjunction with a terminal symbol.
- The writing within standard symbols should be brief. If necessary, the annotation symbol can be used to describe data or computational steps more clearly.
- If the flowchart becomes complex, connector symbols should be used to reduce the number of flow lines. The intersection of flow lines should be avoided to make the flowchart a more effective and better way of communication.
- The validity of the flowchart should be tested by passing simple test data through it.

Advantages of Flow charts:

Advantages of using flowcharts

1. Communication: Flowcharts are a better way of communicating the logic of a system to all concerned.
2. Effective analysis: With the help of flowcharts, problems can be analyzed more effectively.
3. Proper documentation: Program flowcharts serve as a good program documentation needed for various purposes.
4. Efficient coding: Flowcharts act as a guide or blueprint during the systems analysis and program development phase.
5. Proper debugging: Flowcharts help in the debugging process.
6. Efficient program maintenance: The maintenance of an operating program becomes easy with the help of a flowchart.

Limitations of Flowchart:

Limitations of using flowcharts:

1. Complex logic: Sometimes, the program logic is quite complicated. In such a case, a flowchart becomes complex and clumsy.

2. Alterations and modifications: If alterations are required, the flowchart may need to be redrawn completely.
3. Reproduction: Since the flowchart symbols cannot be typed in, the reproduction of a flowchart becomes a problem.
4. Loss of objective: The essentials of what has to be done can easily be lost in the technical details of how it is to be done.

Converting Algorithms to Programs:

1.19 Strategy for Designing Algorithms

Useful strategy for designing algorithms is as follows:

Investigation step:

Identify the outputs needed. This includes the form in which the outputs have to be presented. At the same time, it has to be determined at what intervals and with what precision the output data needs to be given to the user.

Identify the input variables available. This activity considers the specific inputs available for this program, the form in which the input variables would be available, the availability of inputs at different intervals, the ways in which the input would be fed to the transforming process.

Identify the major decisions and conditions. This activity looks into the conditions imposed by the need identified and the limitations of the environment in which the algorithm has to be implemented.

Identify the processes required to transform inputs into required outputs. This activity identifies the various types of procedures needed to manipulate the inputs, within the bounding conditions and the limitations mentioned in step 3, to produce the needed outputs.

Identify the environment available. This activity determines the kind of users and the type of computing machines and software available for implementing the solution through the processes considered in steps.

Top-down development step:

Devise the overall problem solution by identifying the major components of the system.The goal is to divide the problem solution into manageable small pieces that can be solved separately.

Verify the feasibility of breaking up the overall problem solution. The basic idea here is to check that though each small piece of solution procedure is independent, they are not entirely independent of each other, as they together form the whole solution to the problem. In fact, the different pieces of solution procedures have to cooperate and communicate in order to solve the larger problem.

Stepwise refinement:

Work out each and every detail for each small piece of manageable solution procedure. Every input and output dealt with and the transformation algorithms implemented in each small piece of solution procedure, which is also known as a process, is detailed. Even the interfacing details between each small procedure are worked out.

Decompose any solution procedure into further smaller pieces and iterate until the desired level of detail is achieved. Every small piece of solution procedure detailed in step 1 is checked once again. If necessary any of these may be further broken up into still smaller pieces of solution procedure till it can no more be divided into the meaningful procedure.

Group processes together which have some commonality. Some small processes may have to interface with a common upper-level process. Such processes may be grouped together if required.

Group variables together which have some appropriate commonality. Certain variables of the same type may be dealt with elements of a group.

Test each small procedure for its detail and correctness and its interfacing with the other small procedures. Walk through each of the small procedures to determine whether it satisfies the primary requirements and would deliver the appropriate outputs. Also, suitable tests have to be carried out to verify the interfacing between various procedures. Hence, the top-down approach starts with a big and hazy goal. It breaks the big goal into smaller components.

1.20 Tracing an Algorithm to Depict Logic:

An algorithm is a collection of some procedural steps that have some precedence relation between them. Certain procedures may have to be performed before some others are performed.

Decision procedures may also be involved to choose whether some procedures arranged one after other are to be executed in the given order or skipped or implemented repetitively on fulfillment of conditions arising out of some preceding manipulations. Hence, an algorithm is a collection of procedures that results in providing a solution to a problem.

Tracing an algorithm primarily involves tracking the outcome of every procedure in the order they are placed. Tracking in turn means verifying every procedure one by one to determine and confirm the corresponding result that is to be obtained. This in turn can be traced to offer an overall output from the implementation of the algorithm as a whole.

Algorithm to Convert Miles into Kilometers:

Consider the following example for developing an algorithm and the corresponding program in C for converting the distance given in miles to kilometers.

Note: As C programming language is not yet introduced, understand the following important points related to C programming language.

A C program starts the execution from the main() function.

printf() function is used to print out data to the user.

scanf() function is used to read the input from the user.

Further information about C programming language will be given in upcoming sections.

The following data is gathered after the analysis of the above given problem statement:

Input Data : miles	/* the distance in miles*/
Required Output: kilometers	/* the distance in kilometers */

Relevant Formula: 1 mile = 1.609 kilometers

Given below is a sample algorithm for the given problem:

Step-1: Start

Step-2: Read the distance in miles.

Step-3: Convert the distance from miles to kilometers as kilometers ←miles * 1.609.

Step-4: Display the distance in kilometers.

Step-5: Stop

The steps given below show how to transform the algorithm into statements in C:

Step-1: Start → corresponds to void main() {

Step-2: Reading miles → can be done using scanf("%f", &miles);

Step-3: kilometers miles * 1.609 →is converted to kilometers = miles * 1.609;

Step-4: Displaying the output →can be done using

printf("Distance in Kilometers = %7.2f\n", kilometers);

Step-5: Stop → corresponds to }

Algorithm to Convert Fahrenheit to celcius:

Consider the following example for developing an algorithm and the corresponding program in C for converting temperature given in Fahrenheit to Celsius.

The following data is gathered after the analysis of the above given problem statement:

Input Data : fahrenheit /* the temperature in fahrenheit */

Required Output: celsius /* the temperature in celsius */

Relevant Formula: celsius = (5 / 9) * (F - 32)

Given below is a sample algorithm for the given problem:

Step-1: Start

Step-2: Read the temperature in fahrenheit.

Step-3: Convert the temperature to celsius as celsius ←(5 / 9) * (fahrenheit - 32).

Step-4: Display the temperature in celsius.

Step-5: Stop

The steps given below show how to transform the algorithm into statements in C:

Step-1: Start corresponds to void main() {

Step-2: Reading fahrenheit can be done using scanf("%f", &fahrenheit);

Step-3: celsius (5 / 9) * (fahrenheit - 32) is converted to celsius = (5.0 / 9.0) * (fahrenheit - 32);

Step-4: Displaying the output can be done using

printf("Temperature in celsius = %f\n", celsius);

Step-5: Stop corresponds to }