UNIT-1

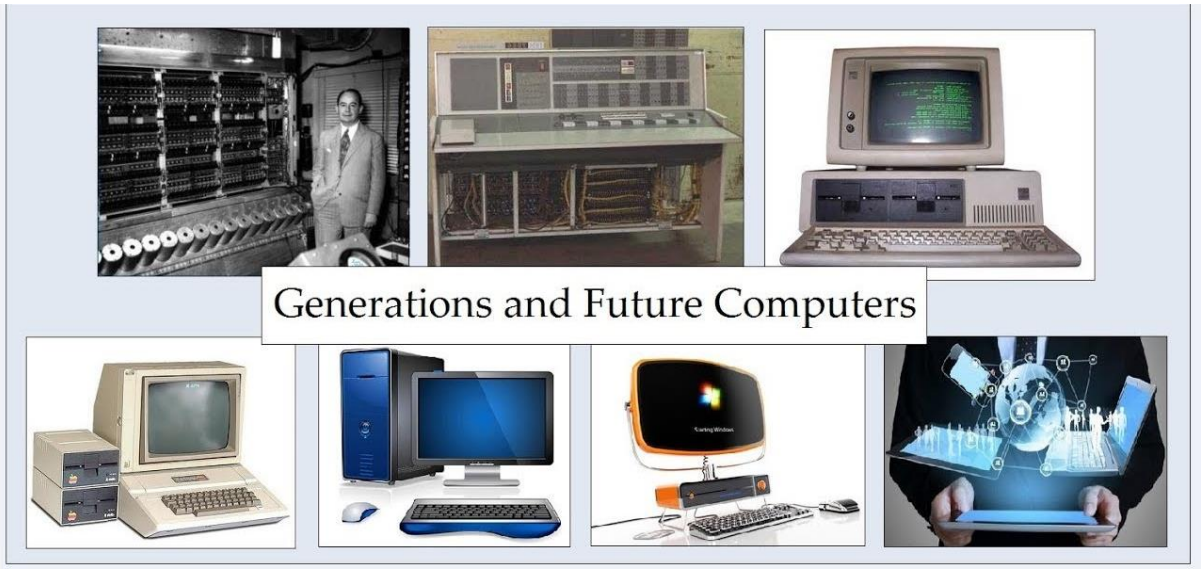
BASIC STRUCTURE OF A COMPUTER AND MACHINE INSTRUCTIONS

1. History of computer generations

Computers are electronic devices which are developed to assist the users in performing difficult calculations as well as other activities. Development of powerful computing devices, led to the invention of several other devices. It however took many years to develop the advanced computer.

1.1 Generations of computers

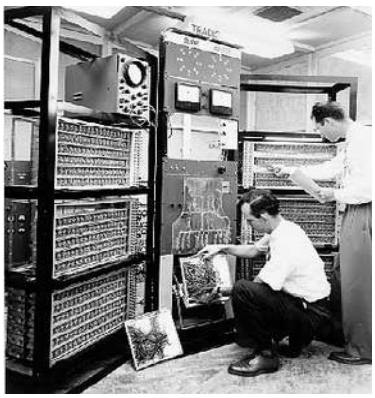
The generation of computers refers to the stages of innovation or development done in the field of computers. This innovation resulted in much smaller, less expensive, more powerful and reliable devices each time. There are five generations of computers developing on the type of processors installed in a machine.



1.2 First generation computers (1940-1956)

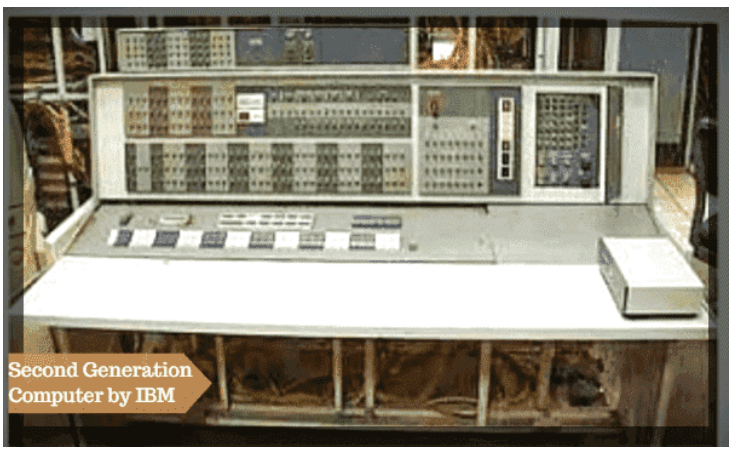
- First generation computers were based on vacuum tubes technology.
- These were larger in size and used more space for installing
- They generate lot of heat as thousands of vacuum tubes were used. Because of this, air conditioning was essential for these machines.
- The computational speed was fast i.e., in milliseconds.
- These machines were non-portable.
- They were not reliable and were highly susceptible to hardware failures.
- These machines were very expensive and consumed large amount of electricity.
- These machines lacked in versatility and speed.

- ❖ Every component of this machine was manually assembled. Therefore commercial production of the first generation machines were very poor.
- ❖ They were machine language dependent and hence it is very difficult to program and use them.
- ❖ Examples: ENIAC (Electronic Numeric Integrator and Computer), EDVAC (Electronic Discrete Variable Automatic Computer) and UNIVAC (Universal Automatic Computer)



1.3 Second generation computers (1956-1963)

- ❖ These were based on transistors technology
- ❖ Comparatively smaller in size than the first generation machines.
- ❖ Generate less amount of heat, but still air conditioning was required.
- ❖ Computational speed was very high (from milliseconds to microseconds)
- ❖ Machines were portable.
- ❖ Machines were reliable and were not susceptible to the hardware failures.
- ❖ Every individual component was manually assembled to form a functional unit.
- ❖ They are assembly language dependent used to program computers.
- ❖ Examples: PDP-8 (Programmed Data Processor), IBM-1401 (International Business Machines), IBM-7090.



1.4 Third generation computers (1964-Early 1970's)

- These were based on Integrated Circuit (IC) technology
- The speed of computation was very fast (from microseconds to nanoseconds)
- These were smaller in size when compared to other machines.
- Machines were portable as well as reliable.
- Generate less amount of heat. Air conditioning was not mandatory bur required in some cases.
- The cost of maintenance was very less.
- They are high-level language dependent.
- They didn't require manual assembling of individual components.
- The commercial production became easy and less expensive.
- Examples: NCR395 (National Cash Register) and B6500

1.5 Fourth generation computers (Early 1970's-Till date)

- These were based on microprocessor's technology
- These machines were very small in size, when compared to other machines.
- These machines were very much cheaper.
- They were portable and reliable.
- They didn't generate much heat, therefore air conditioning was not required.
- Maintenance cost was very less, since didn't prone to any sort of hardware failure.
- The cost of production is very low.
- These machines were easier to use because of GUI and pointing devices.
- Interconnection among different computers resulted in better communication and resource sharing.
- Examples: Apple 11, Altair 8800 and CRAY-1

1.6 Fifth generation computers (Present and beyond)

- ✓ **Mega Chips:** These computers will user Super Large Scale Integrated (SLSI) chips. The usage of these chips will result in the generation of microprocessors. These microprocessors consist of hundreds and thousands of electronic components mounted on a single chip. Fifth generation computers required large amount of storage space for storing information as well as instructions. The usage of Mega Chip allows the memory capacity of computer to be approximately equal to the capacity of human mind.
- ✓ **Parallel Processing:** In contrast to serial processing, parallel processing executed several instructions simultaneously as a single operation.
- ✓ **Artificial Intelligence:** Artificial Intelligence (AI) refers to sequence interrelated technologies that generally simulate and reproduce human behaviour which include thinking, speaking and reasoning. The different technologies present in artificial intelligence are, expert systems, natural language processing, speed and voice recognition and robotics.

After fourth generation, the computer exhibits high organizational or application driven features. It includes computer showing artificial intelligence, large parallel machines, and extensively distributed systems. The development of the computer industry is mainly caused by increasing the powerful desktops and also increasing the usage of information resources present in the internet. There was a significant increase in speed when the usage of mechanical and electromechanical devices was stopped and vacuum tubes were used. The increased speed was 100 to 1000 fold from seconds to milliseconds. Then tubes were later replaced by transistors boosting the speed up to 1000 fold. As a result, the basic operations were completed within few microseconds. When the density of the integrated chips were increased, the microprocessor chips were manufactured which performed the basic operations within nanoseconds with an increase in speed of 1000-fold. Apart from this, caches and pipelining were introduced to improve the architecture.

2. Digital computer:

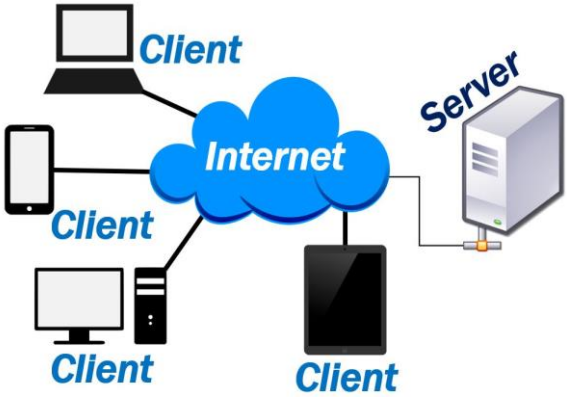
A digital computer is a fast electronic data processing device, which takes the input in the form of instructions (often referred as programs), process it by referring to its various memory elements and finally produces the result.

Though there is no unique distinction defined which refers to differentiation of various types of computers, but we can have the following classification based on size, speed, power consumption requirements, etc.

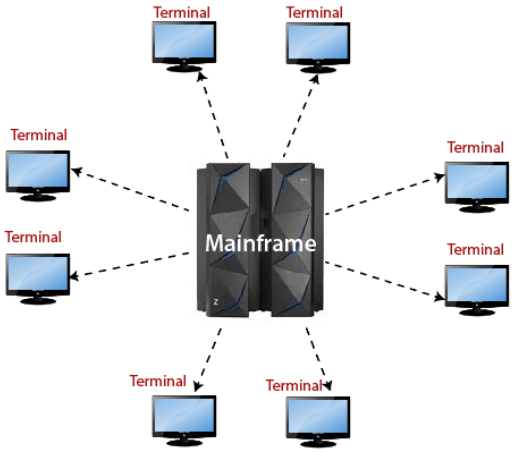
1. **Personal computers:** These are also called as desktop computers. These are used in homes, banks, small scale institutions, etc. These can have several input units in the form of mouse, joysticks, keyboard, etc. through which it fetches data and memory units such as CD-ROMs, hard disk, etc., which are used as a storage media. Apart from these peripherals, a personal computer can also have monitors, speakers, etc. as output units, using which it supplies data to the outside world.

2. **Workstations**: While compromising in terms of size, a workstation is another type of computer which has an edge on personal computers. The difference can be very well calibrated in terms of its computational speed, high degree of visual effects, the level of designing capabilities, etc. Due to this versatile behaviour, workstations are now groomed to satisfy almost all the engineering requirements.

3. **Servers:** These are large computational handling devices which can adhere to thousands of requests made by several clients over Internet. For this purpose, these computers maintain heavy storage locations often referred as databases which are query dependent and act according to the directions proposed by servers. As these types of computers are satisfying or serving various client requests and hence, they are names as “Servers”.



4. **Mainframes:** In essence to workstations, mainframes are the computers which are devised to compromise with the large-scale needs of business sector. Hence, they are far better than workstations in terms of speed and other computational capabilities.



5. **Super computers:** When there is high level of complex calculations at the same time there is a necessity of handling of extremely large volumes of data, super computers serve best for this purpose. It not only manages large volumes of data, but also performs several billions of calculations per second. These computers generally occupy large rooms and are operated by several professional engineers. They can be used in designing dynamics of machines, weapons, supersonic aircrafts, etc. Globally, there are only few countries which possess super computers.

3. Functional units

These are fundamental parts that form a computer. Every computer is made up of five independent functional units. Those are input unit, output unit, memory unit, arithmetic and logic unit and control unit.

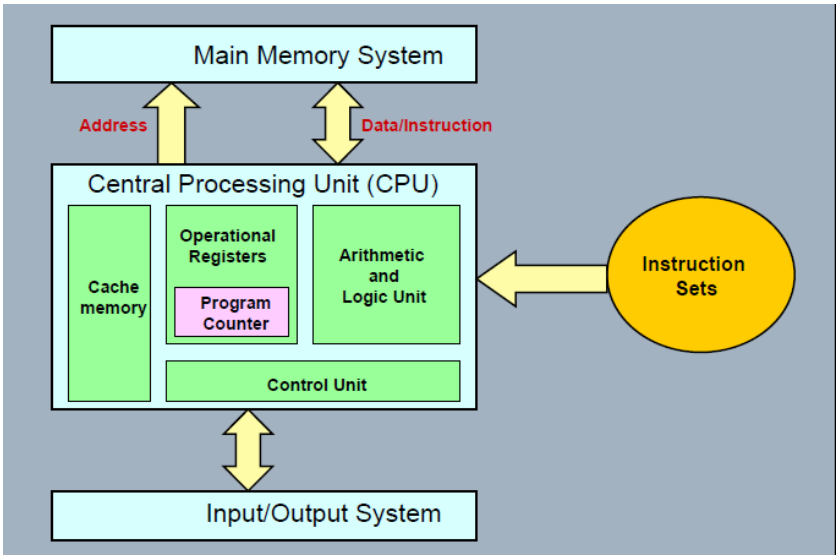
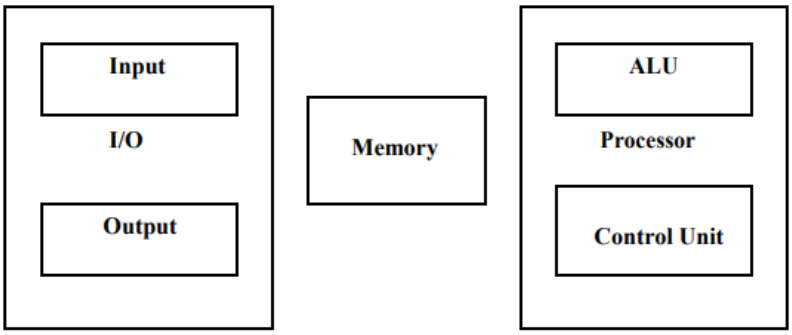


Fig.1 Organization of computer



(a) Basic Units

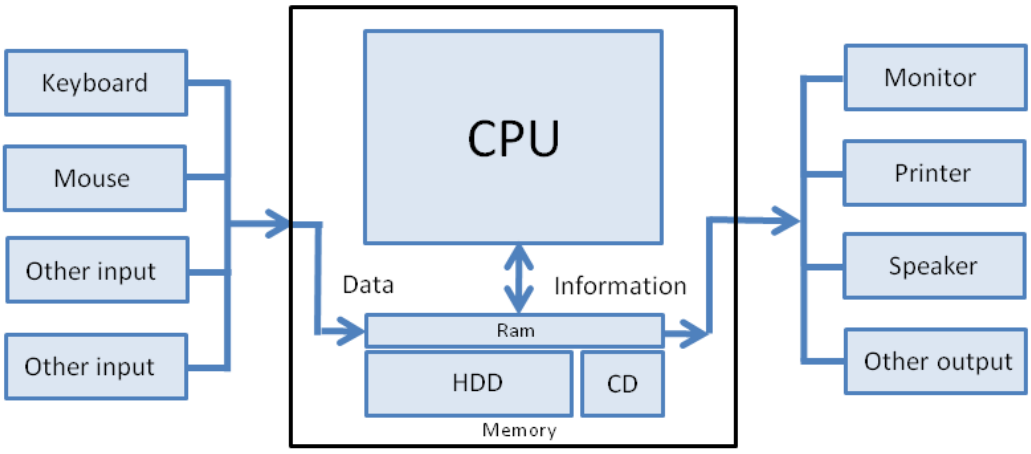
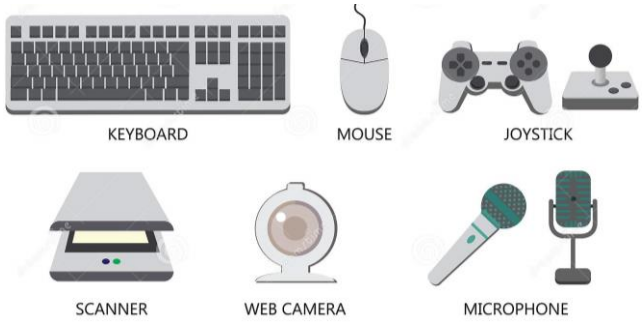


Fig.2 Functional units of a computer

i. **I/O subsystems:** I/O subsystems are necessary for the process of communication between the computer and external environment. The data and instructions are stored on the computers by using some input devices. After the processing of data, the output generated will be stored on the output devices. These devices are useful for various applications. Buses are used to connect the components of a computer system. They are used for the transmission of data from one component to the other.

a) **Input unit:** These are the units which accepts the data from the outside world. The most commonly used input unit is keyboard. It is mechanical device through which we supply data to the computer. Other input units include mouse, scanners, microphones, track balls, joysticks, etc. These input units get connected to the computers by using an interface. The mouse is a device, which acts on graphical displays and proves fast accessing of different entities. Scanners are responsible for scanning various images, texts, etc. Microphones take voice signals and manipulate them accordingly. Track balls act like mouse, but are placed or gorm a part of keyboard. Joysticks are the devices which are used for controlling the motion of fast-moving images, etc.

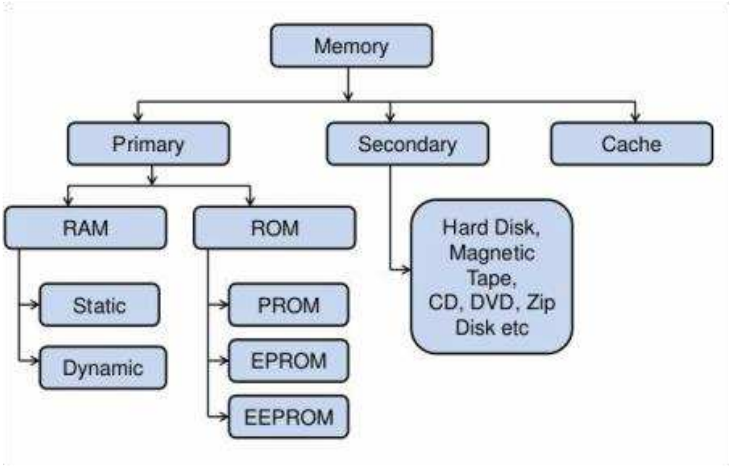


b) **Output unit:** The devices through which data is supplied to the outside world. The basic output device is a monitor. It displays text, images, etc. Few other output devices include speaker, printer, etc. The speakers are the devices through which sound is heard. It also acts as an amplifier. Printer is a device used for printing data in the form of text or images, it is also most widely used output device.

ii. **Memory unit:** It is a unit which is responsible for storing large volumes of computer data. Basically, whenever a program is under execution, the processor utilizes these memories to extract required data for execution of the given program. Every computer maintains two sets of memory, the primary and the secondary memory. Combining two or more memory chips in order for enhancing the capacity of a memory unit is called memory subsystem.

Primary memories are extremely fast memories. An example includes RAMs. These are nothing but semiconductor elements capable of storing single bit of information. These elements together store certain length of data called words. A small-scale memory stores about few thousands of words where as a medium scale memory can store about few millions of words. An extremely fast memory which is provided to support the speed of processor is referred to as cache memory. A computer consists of a new set of memory referred to as main memory. These types of memories are slow, but can store large volumes of data, i.e., data extending in Giga bytes.

Secondary memories have less cost and at the same time provide large storage capacity. Few examples of secondary memories are magnetic tapes, CDROMS, etc.



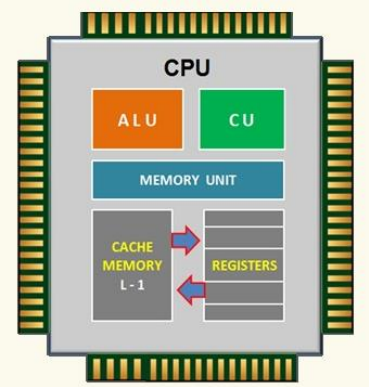
iii. **Central Processing Unit (CPU):** It is the brain of the computer that is responsible for controlling it, by interpreting and executing most of the commands from the hardware and software. It fetches, decodes and executes the instructions in order to perform the basic arithmetical, logical and input, output operations of the system. The three internal sections that are generally present in the CPU are:

a) **Register section:** It is a collection of a bus and registers that are special types of memory units used by the computer in order to transfer the data with high speed. These registers are not considered as the part of main memory but they do store the

data or information temporarily and then pass it based on the directions given by the control unit.

b) **Arithmetic and Logic unit (ALU):** This is responsible for performing arithmetic related operations. When user supplies data for either addition, subtraction, multiplication or division, etc. the processor initiates the arithmetic and logic unit for further manipulations. For example, if a user wants to perform multiplication of two numbers, the processor initially fetches the required data i.e., operands and initiates the arithmetic and logic unit. This unit, examine the data and accordingly performs the required computation and then returns the output.

c) **Control unit (CU):** This looks quite simple, but forms the major component of the entire computer. Its main function is to govern the activities of other functional units. This is done by transmitting the required control signals to various other units. These control signals inhibit the activities of these units and instruct them to perform their operations in the prescribed timing. Hence not granting full independence to other functional units which may lead to degradation of systems performance.



4. Basic operational concepts

The instructions are responsible for carrying out any activity in a computer. When a task is to be performed, its associated program and data operands are stored in the memory. Then each instruction is moved into the processor one-by-one for executing the specified operations.

Example, a task of adding two numbers is to be carried out. Let the operands used in this addition be stored at memory locations LOC1 and LOC2. Then the code for this addition operation will be

```
LOAD LOC1, R0
LOAD LOC2, R1
ADD R0, R1
```

In the first instruction, the operand at memory location LOC1 is loaded into the processor register R0. In the second instruction, the operand at memory location LOC2 is loaded into the processor register R1. In the third instruction, the operand in the register R0 is added to the operand in the register R1 and the sum is stored in the register R1.

The data transfer between memory and processor is carried out by sending the address of the required memory location to the memory unit and issuing the relevant control signals. When this is done, data can be transferred to or from the memory.

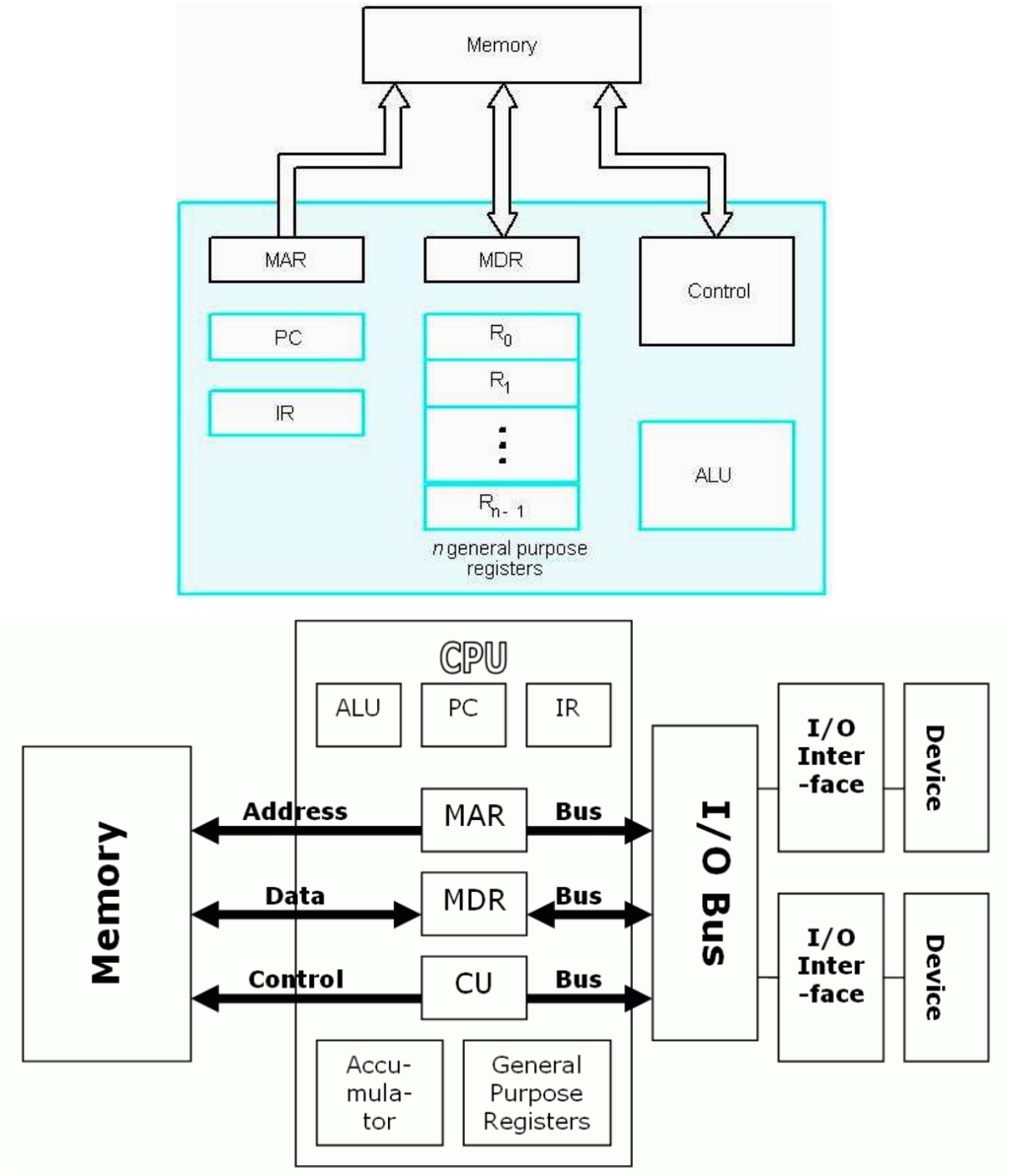


Fig.3 Interconnection between processor and main memory

The Fig.3 shows the interconnection structure of processor and main memory. Basically, the processor structure internally consists of many complex circuitries i.e., ALU, CU, registers. These registers get further divided into general purpose registers and special purpose registers. Special purpose registers include MAR, MDR, PC, IR, etc. Among these registers only MAR and MDR possess direct contact with the main memory. Apart from these registers, only CU can have its direct contact with the main memory.

The components of the processor are

- ➔ Memory unit is an essential unit of every computer which is responsible for storing all the essential data.
- ➔ Control unit issues various control signals which are essential in controlling the activities of all other co-units.
- ➔ ALU is responsible for performing various arithmetic and logical operations.
- ➔ MAR (Memory Address Register) is responsible for addressing various locations in the memory.
- ➔ MDR (Memory Data Register) is responsible for reading/writing data from/to memory.
- ➔ PC (Program Counter) stores the address of the next instruction to be executed.
- ➔ IR (Instruction Register) stores the instruction which is under execution.
- ➔ General purpose registers are responsible for storing frequently required data related to program execution temporarily.

Consider, the execution of a given program through this circuitry. Initially, the data which is input through a keyboard gets stored in the main memory. The program counter points to the first instruction. The address of this instruction gets stored in MAR and MDR extracts the required instruction and stored it in the IR. This entire process gets initiated only after receiving the control signals from the control unit. If the instruction contains certain arithmetic and logical operations, then ALU gets initiated which fetches the required operators and operands while considering MAR and MDR registers. After termination of execution of first instruction, the result is transmitted to MDR, which places it into the main memory and its address gets placed into MAR. Now, the program counter points to next instruction and the same process repeats until all the instruction of the program get executed. It may happen that, while the program is under execution, the processor may get interrupted by any input/output routines. In this case, the processor initially checks the priority of the interrupt. If it is of high priority, then the processor places the contents of PC, IR, the general-

purpose registers and few control information in the memory and switches to respond to that interrupt routine. Once the interrupt task gets completed, the processor acquires stored contents from the memory and continues with its process. These interrupt mechanisms are supported by modern computers so as to improve the overall performance of the system and also to utilize the processor optimally.

5. Bus structures

A transmitting media which is a combination of two or more wires and capable of driving data from one unit to other unit is used and is referred to as a bus. These are used to exchange information/messages between different functional units of a computer.

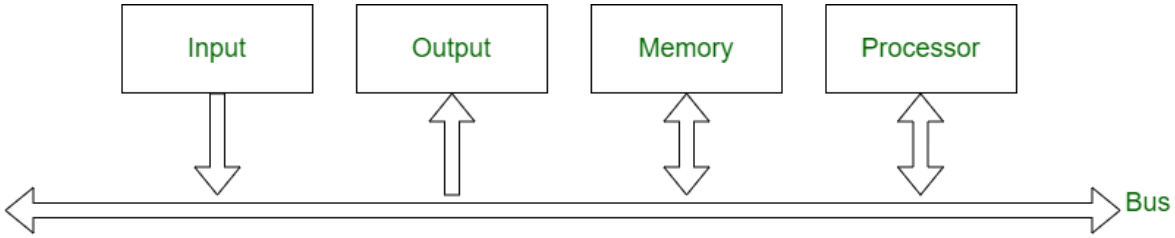


Fig.4 Single bus structure

A bus that carries data in the form of signals, with one signal per wire to given functional units. Apart from data, a bus also includes wires which allows address and certain control signals to be transmitted. As there is parallel transmission of signals, a single word can be transmitted in each attempt, at the same time increasing the speed of the system. The Fig.4. represents single bus architecture. Since, there is one bus connecting various functional devices. In this, two units can use the bus at a time.

However, there are many systems which came into existence, supporting more than one bus. In those systems, parallel transmission of data is possible as well as more than one device can interact with each other. But as the number of buses increases, the effective cost and the complexity of the architecture increases. The architecture no longer remains simple in implementation and flexible as it was the case with single bus architecture. When entire system is considered, there are many units with different speeds. For example, the processor of the system is always of high speed when compared to slow devices like input/output units, etc.

Hence, the architecture selected should provide compatibility among such devices. In order to compromise with the relative speeds of these units, several mechanisms are implemented. Once of such mechanism include, the introduction of special registers referred to as “buffered registers”. These registers store the data temporarily till a given task is completed. For example, a processor needs to print 300 bytes of data. For this purpose, the

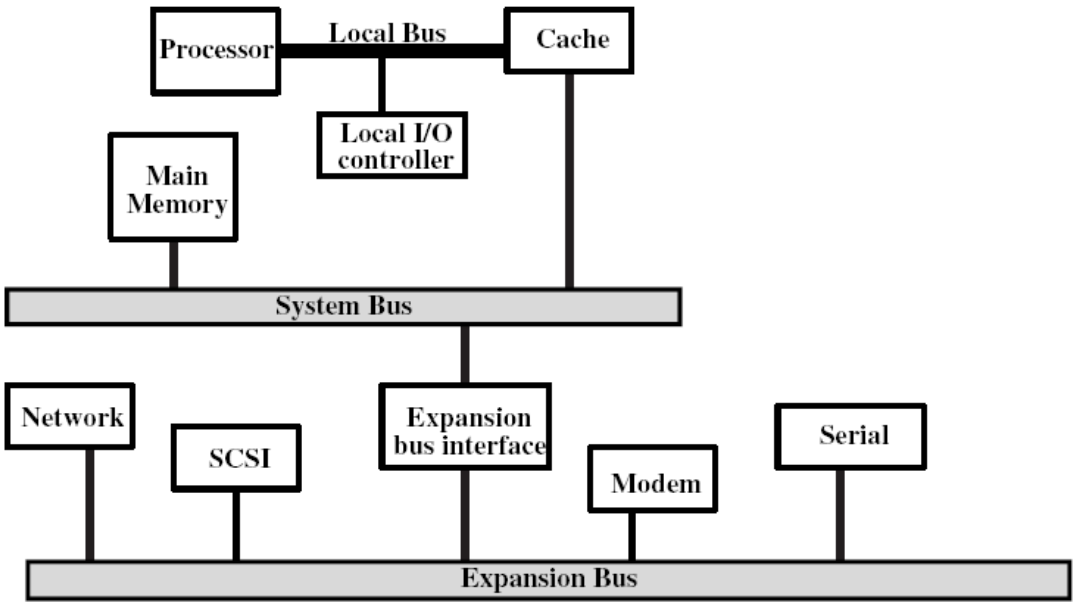
processor simply extracts the required data from the memory and transmits it to the buffer registers and switches to next task. The printer sequentially extracts the data from these registers and prints them accordingly. Hence, the processor will no longer remain idle till the completion of printing process and can easily execute other tasks. This mechanism not only increases the speed of the system, but also improves the system performance.

→Multiple bus structures (Traditional)

Figure shows some typical example of I/O devices that might be attached to expansion devices. The traditional bus connection uses three buses local bus, system bus and expansion bus

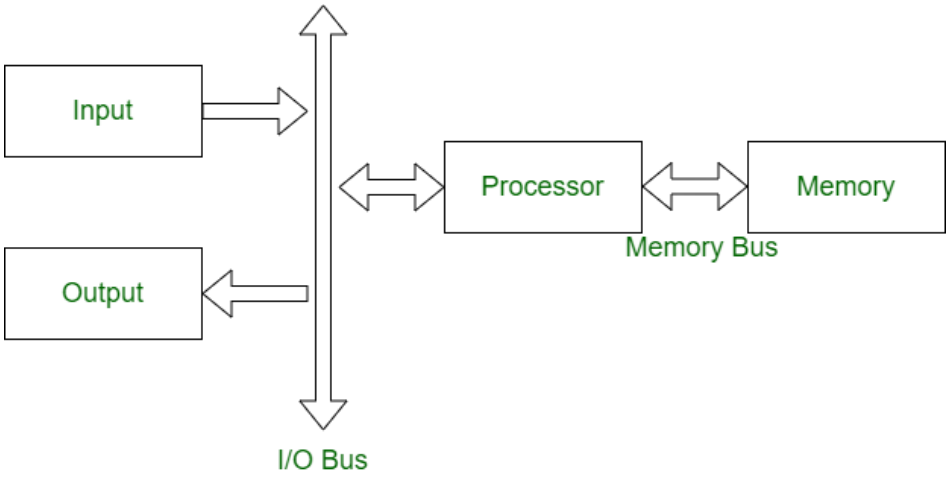
i. Local bus connects the processor to cache memory and may support one or more local devices
ii. The cache memory controller connects the cache to local bus and to the system bus.
iii. System bus also connects main memory module
iv. Input /output transfer to and from the main memory across the system bus do not interface with the processor activity because process accesses cache memory.
v. It is possible to connect I/O controllers directly on to the system bus. A more efficient solution is to make use of one or more expansion buses for this purpose. An expansion bus interface buffers data transfer between system bus and i/o controller on the expansion bus.

This arrangement allows the system to support a wide variety of i/o devices and at the same time insulate memory to process or traffic from i/o traffic.



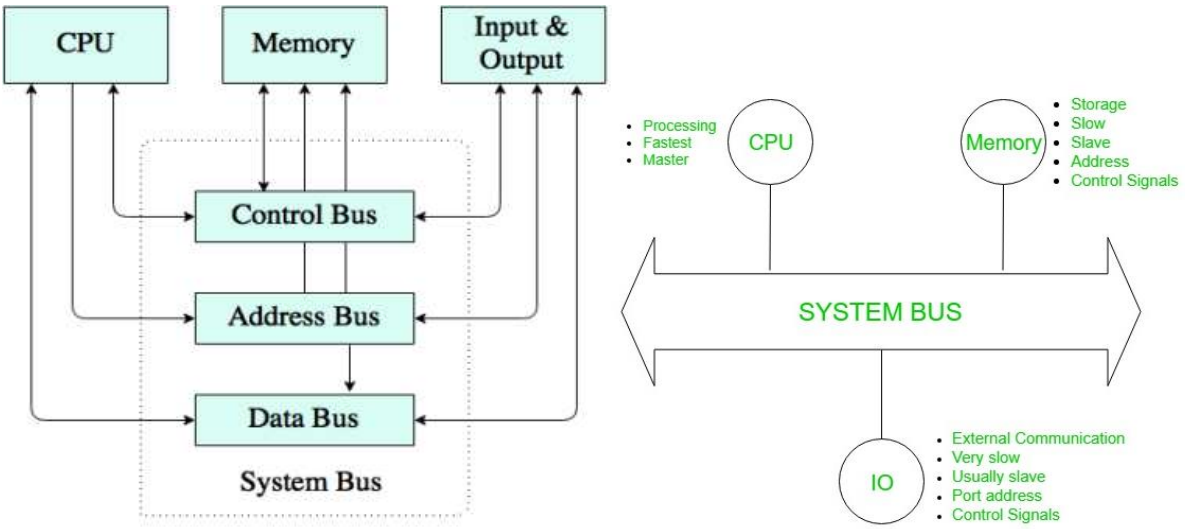
→Double bus structure

In double bus structure, one bus is used to fetch instruction while other is used to fetch data, required for execution. It is to overcome the bottleneck of single bus structure. One common bus is used for communication between peripherals and processor.



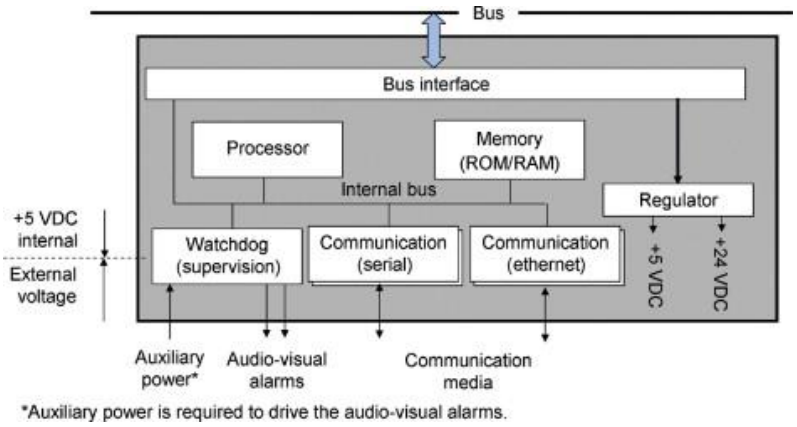
→System bus structure

The system bus is a pathway composed of cables and connectors used to carry data between a computer microprocessor and the main memory. The bus provides a communication path for the data and control signals moving between the major components of the computer system

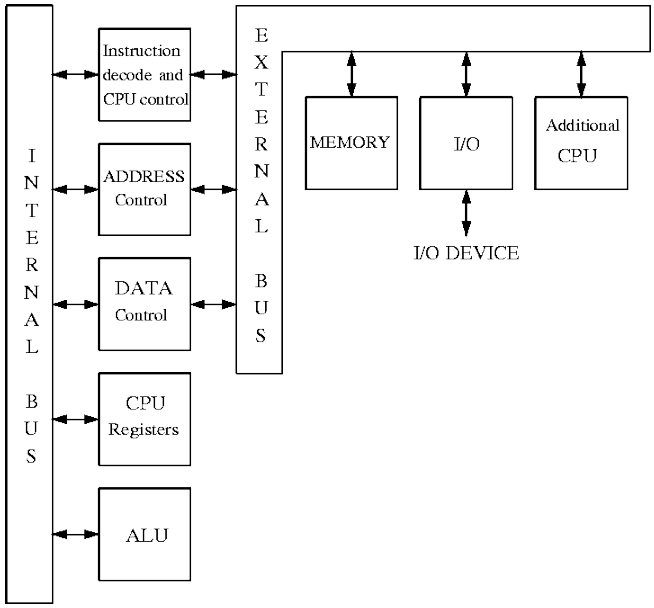


5.1 Types of buses

i. **Internal bus:** The bus that operates only within the internal circuitry of the CPU, communicating among the internal caches of memory is referred to as internal bus. This is quick and remains independent of the rest of the computers operations.

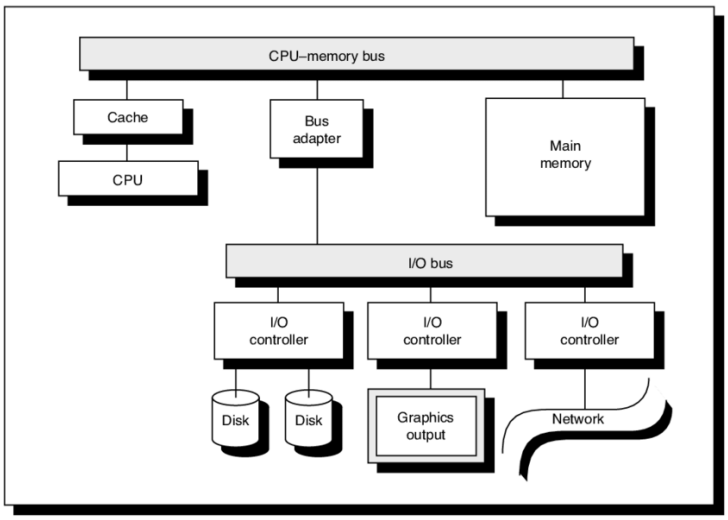


ii. **External bus:** This is a bus which connects computer to external devices is called an external bus. For example, USB, IEEE 1394.



iii. **Backplane:** This bus includes a row of connectors into which system modules can be plugged in. It also supports a logical protocol through which electrical conflicts are avoided to greater extent.

iv. **I/O bus:** The bus used by I/O devices to communicate with the CPU is usually referred to as I/O bus. These buses have wide applications in a given computer. The major function of this bus is to transfer the data to/from CPU to I/O devices.



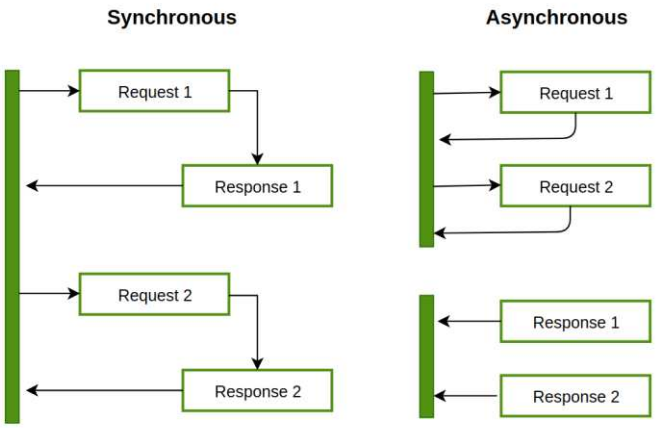
v. **System bus:** This bus is usually a combination of address bus, data bus and control bus. This forms one of the important buses in the entire computer system architecture.

vi. **Data bus:** This bus is capable of transferring data to/from the memory or any other peripheral devices or between memory and peripheral devices is usually referred as a data bus. These are bidirectional buses.

vii. **Address bus:** This bus is used by the CPU for transmission of address and which depicts a particular memory location or a given input/output port is referred to as an address bus. These are unidirectional buses.

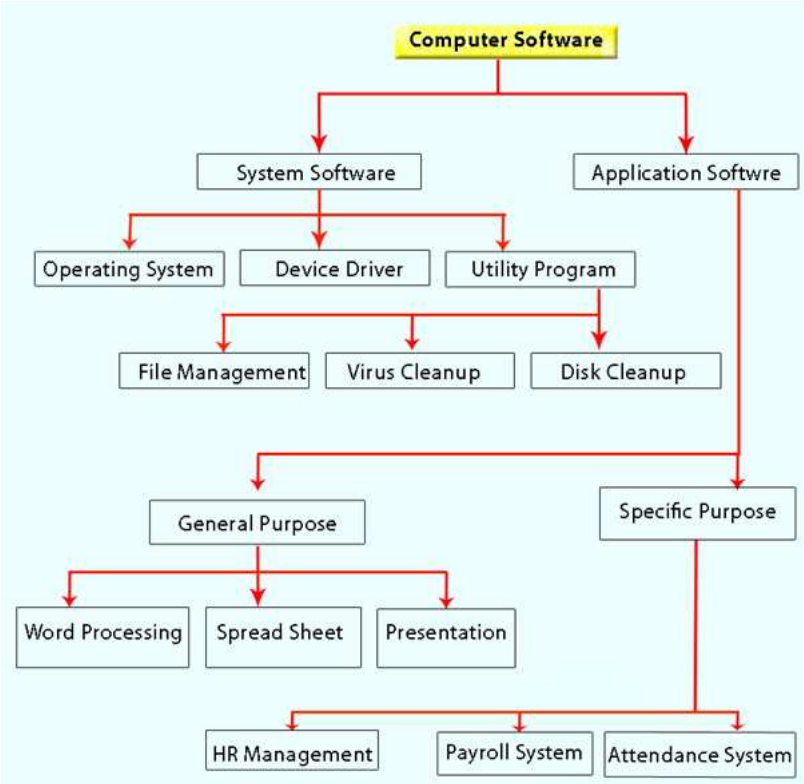
viii. **Synchronous bus:** While using synchronous bus, data transmission between source and destination units takes place in a given time slot which is already known to these units. This can be done by maintaining single clock unit which derives both of these units.

ix. **Asynchronous bus:** The data transmission is governed by a special concept i.e., handshaking control signals. Using this concept the time of transmit/receive can be easily maintained.



6. System software

System software is usually a program residing in main memory which is capable of organizing various components of a computer. It can also be defined as a collection of various programs, providing a platform to the user through which one can easily interact with the system. Examples are operating systems, text editors, compilers, etc.



6.1 Functions of system software

- ➔ Facilitates execution of user developed programs by including various libraries and packages etc.
- ➔ Controls various input and output activities.
- ➔ Accepts and processes various user typed commands.

- ➔ Performs memory management functions.
- ➔ Manages all the application programs and accordingly stores and retrieves them
- ➔ Converts high level language programs into machine level language.
- ➔ Controls various programs such as video games, where there is strong user intervention. It also executes user written application programs.
- ➔ Schedules various processes in accordance to their priorities.
- ➔ Organizes all the peripheral devices attached to a given system.
- ➔ Prevents system failures and unauthorized accessing of the system by providing authorization codes such as user name and passwords.

6.2 Operating system performance

An operating system (OS) is a program which acts as an interface between a user and a computer. It provides user friendly environment to the operator, so that one can directly interacts with the system. The functions are

- ➔ Input/output management
- ➔ Memory or disk management
- ➔ Organization of various peripheral devices.
- ➔ Scheduling various processes by providing them with priorities.
- ➔ Granting access to various system resources.
- ➔ Inhibiting unauthorized users from accessing different system resources.

In order to understand the services of an operating system, consider a simple example. Assume that certain data provided as input to the system, which gets stored in the main memory, which is later extracted and executed and finally the result is displayed. In all the activities mentioned, an operating system plays a vital role. While data is to be provided to the system, it is the operating system which enables the keyboard operation through which a user enters the required data. When execution of the program begins, assume that this program requires certain file which is currently residing on the disk. To manage this situation,

the control switches from the program to the operating system. Where OS extracts the corresponding file and stores it in the main memory and later return the control to the program. As soon as the execution of program completes, it is again the responsibility of the operating system to gain the control and display the results on an output device. Hence in this way operating system and user program interacts with each other by sharing the processor. This can be diagrammatically represented as

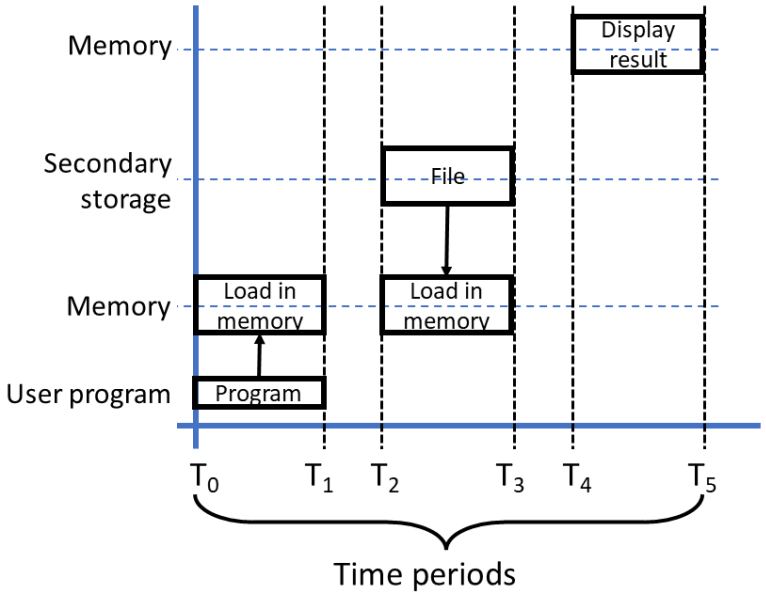
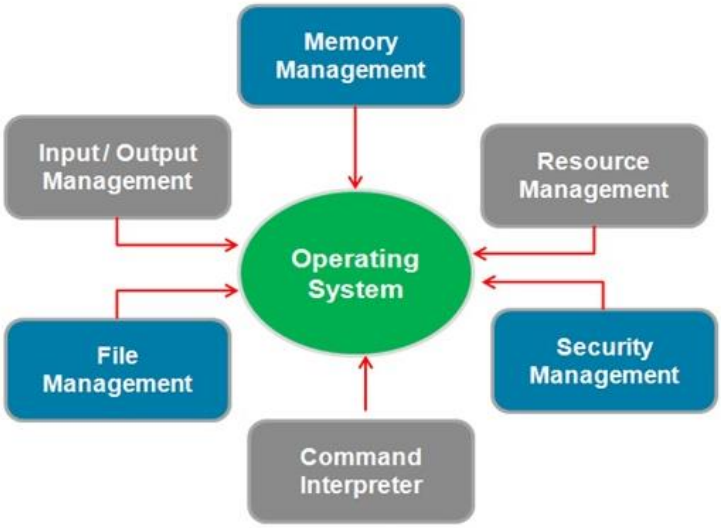


Fig.5 Various activities handled by the OS

It can be observed from above diagram the way different activities are handled by the operating system. During the time period T0 and T1, the user program is loaded into the main memory. During time period T2 and T3, the required file is extracted from the secondary storage and loaded into the memory. After end of entire processing the results are passed on to the monitor where they are displayed.

In this entire scenario, it is the processor responsible for switching control between various components. The processor schedules these operations by maintaining time slots. Specified component gets activated only when the processor authorizes it to use its time slots. In this way, the operating system routines and a user program share the processor.

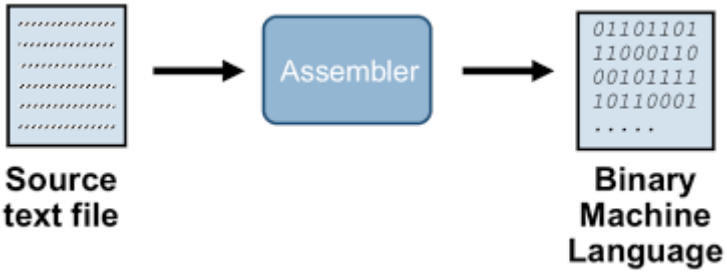


6.3 Text editor

Text editors are system software's which provide feasibility to users while typing data belonging to several application programs. These editors receive several commands entered by the user and accordingly execute them. Several new editors provide facilities such as, highlighting the inappropriate sentences, changing colour of misspelled words, underlining the sentences which are not obeying the syntax, etc. Apart from these feature, the editors consider the data entered by the users into a file and accordingly store them into memory or other locations. These editors sometimes display informative messages if the user is not following the exact format etc.

6.4 Assembler

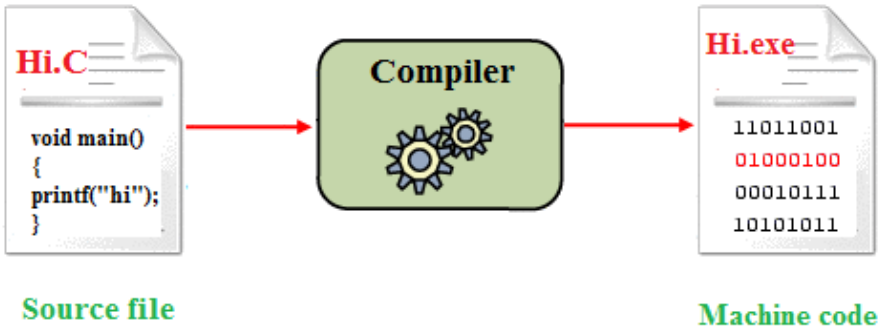
An assembler translates assembly language programs into machine code. The output of assembler is called an object file, which contains a combination of machine instruction as well as the data required to place these instructions in memory.



6.5 Compiler

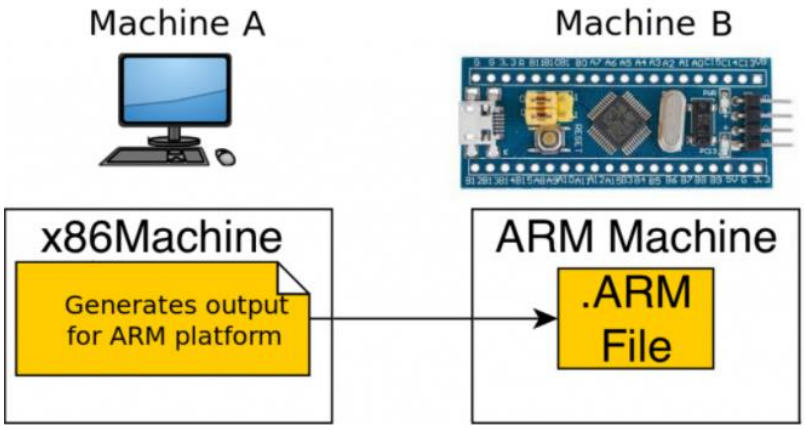
A compiler is a computer program (or a set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language).

Typically, from high level source code to low level machine code or object code.



6.6 Cross compiler

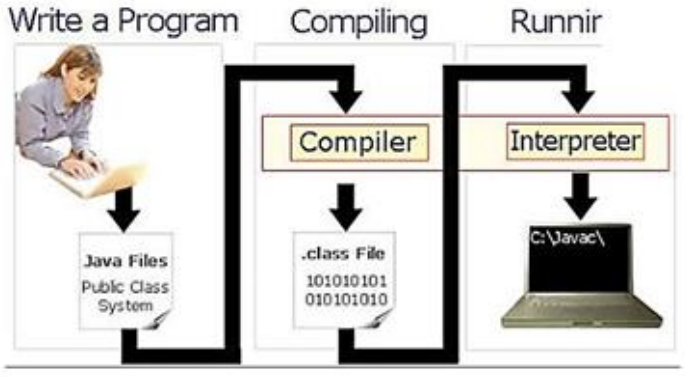
A **cross compiler** is a **compiler** capable of creating executable code for a platform other than the one on which the **compiler** is running. For example, a **compiler** that runs on a Windows 7 PC but generates code that runs on Android smartphone is a **cross compiler**.



6.7 Interpreter

An interpreter is a common kind of language processor. Instead of producing target program as a translation, an interpreter appears to directly execute the operations specified in the source program on inputs supplied by the user.

In contrast, an interpreter reads a statement from the input, converts it to an intermediate code, execute it, then takes the next statement in sequence. If an error occurs, an interpreter stops execution and reports it.

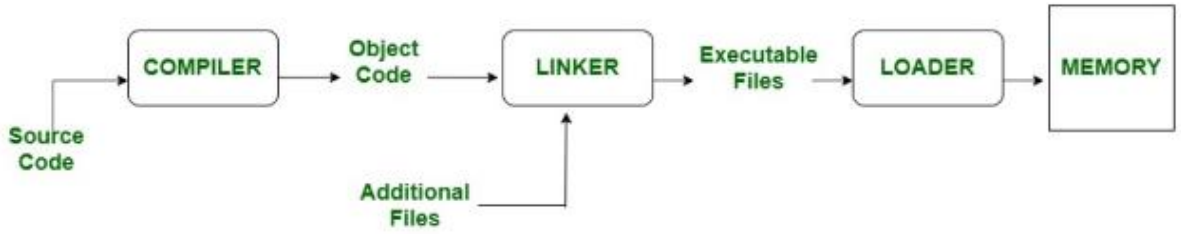


Topic:- Write a program Compiling Running

6.8 Linker

Linker is a computer program that links and merges various object files together in order to make an executable file. All these files might have been compiled by separate assembler.

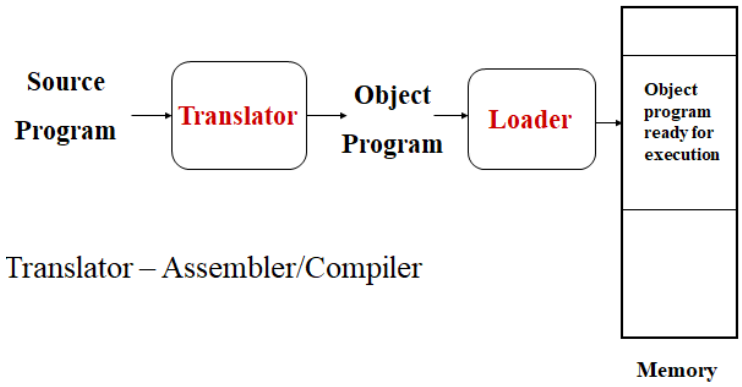
The major task of a linker is to search and locate referenced module/routines in a program and to determine the memory location where these codes will be loaded making the program instruction to have absolute reference.



6.9 Loader

Loader is a part of operating system and is responsible for loading executable files into memory and execute them.

It calculates the size of a program (instructions and data) and create memory space for it. It initializes various registers to initiate execution



7. Performance of a computer

For any computer, performance is measured in terms of its accessing capability. Hence, performance is nothing but how fast a computer executes programs. The accessing speed of a computer basically depends on three entities. They are

i. The hardware circuitry through which the computer is made.

ii. The design of various instruction sets, supported by a computer.

iii. The design of different compilers, which are used in compiling a set of programs.

7.1 Processor clock

In order to organize or schedule the processor's activities, a clock is embedded on its circuit board which is referred to as a processor clock. Hence, a processor performs its activities depending on the clock cycles. For example, assume that a processor has to store certain data, display and print certain text. In order to manage these activities, the processor clock, ascertain clock cycles. Say clock cycle T1 to store the data, T2 to display the data and T3 for printing the text. Hence, the processor completes these tasks in the prescribed clock cycles. The length of a clock cycle is an important measure that affects the system's performance.

7.2 Basic performance equation

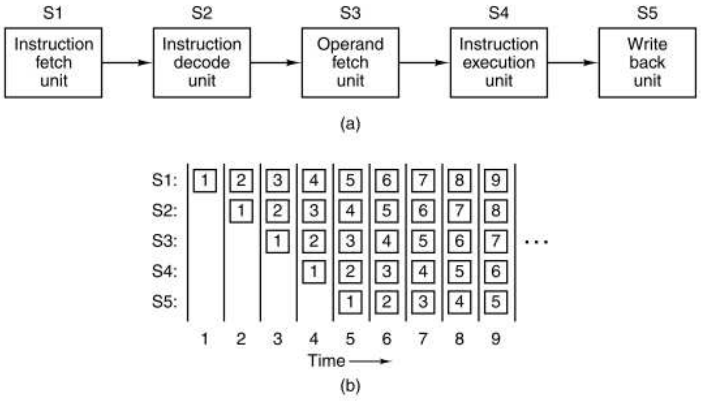
The basic performance equation gives relationship among various parameters through which system's performance can be enhanced to greater extent. To achieve high performance, the value of 'T' must be low.

$$T = \frac{N \times S}{R}$$

Where, 'T' is program execution time, 'N' is the number of instruction in a given high level language program, 'S' is the average number of processing steps essential in executing a given machine instruction, 'R' is the clock rate in terms of cycles per second.

7.3 Pipelining and superscalar operation

In olden days most of the computers support only sequential execution of instructions i.e., the processor switches to instruction-two only after completion of execution of instruction-one. This mechanism would utilize large number of clock cycles. An improvement in this field lead to development of new concept referred to as pipelining. In this case, the processor need not wait till the execution of instruction-one is complete, in order to execute instruction-two, rather the processor directly starts executing instruction-two, while the instruction-one is under execution. Pipelining can be achieved by dividing the processor activities into time slots and assigning each of these slots to various processes.



As pipelining gives rise to overlapping or parallel execution of many instructions, in order to attain the behaviour most of the circuitry has to be redesigned. Hence, as execution of several instructions is possible in a given time slot, this activity is also referred to as superscalar operation.

Instruction Fetch, Instruction Decode, Execution, Memory access, Register write back

7.4 Clock rate

The clock rate must be increased to achieve the high performance. This is achieved in two ways. First, by improving Integrated Circuit (IC) technology that makes logic circuits fastest. Second, by reducing the amount of processing done in a machine instruction.

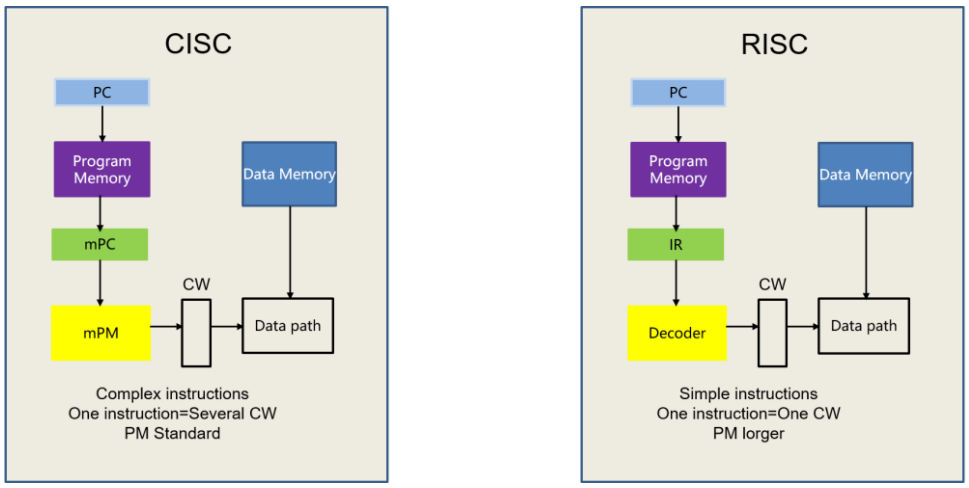
7.5 CISC and RISC

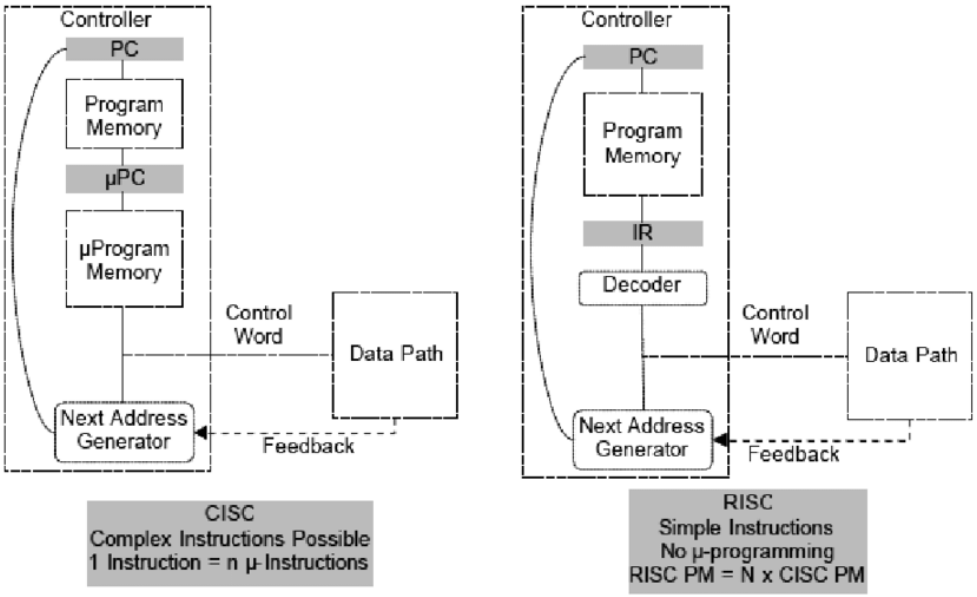
According to the architecture design, the CPU is divided into **Complex Instruction Set Computing** (CISC) and **Reduced Instruction Set Computing** (RISC).

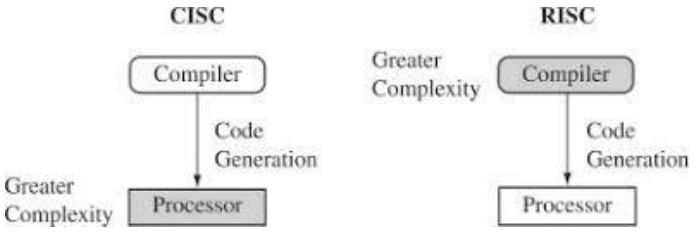
Complex Instruction Set Computer (CISC) refers to processors that can implement certain complex instruction. These complex instructions may have their resemblance with the program written in high level programming languages.

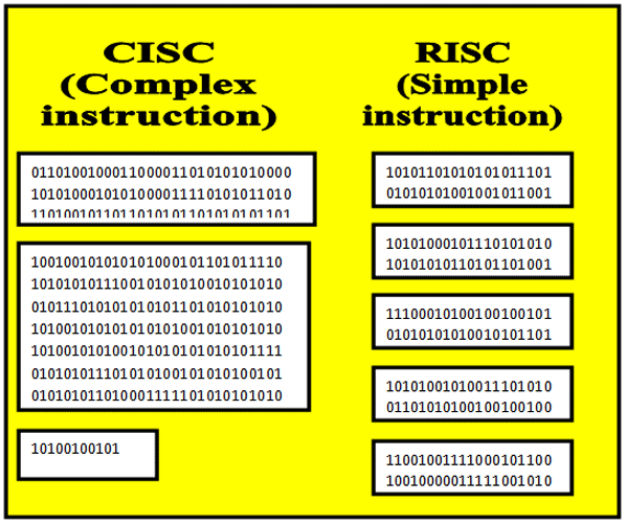
The CISC is a microprocessor that executes a set of computer instructions. The core of each microprocessor is the circuit that runs the instructions. An instruction consists of several steps to complete a task, transferring values into a register, or performing an addition operation. Intel processors are used as examples.

Reduced Instruction Set Computer (RISC) refers to another variety of processors which implements a concept that, every instruction is valid to acquire only single word. The RISC design idea reduces the number of instructions and an addressing manner so that implementation is easier, an instruction execution degree is better, and the efficiency of a compiler is higher.









7.6 Performance measurement

In order to demonstrate or measure the performance of the system, a new organization was formed under the name SPEC (System Performance Evaluation Corporation). This organization framed certain demonstrating program and executed them on different systems. The compilation of programs was initially organized on the system which was under test and followed compilation on the standard systems. Finally, they came out with a ratio known as SPEC rating which is defined as "running time of a given program on the standard system to the running time of the same program on the under observation system"

$$\text{SPEC rating} = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

8. Number Systems

Human beings use decimal (base 10) and duodecimal (base 12) number systems for counting and measurements (probably because we have 10 fingers and two big toes). Computers use binary (base 2) number system, as they are made from binary digital components (known as transistors) operating in two states - on and off. In computing, we also use hexadecimal (base 16) or octal (base 8) number systems, as a compact form for representing binary numbers.

8.1 Decimal (Base 10) Number System Decimal number system has ten symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, called digits. It uses positional notation. That is, the least-significant digit (right-most digit) is of the order of 10^0 (units or ones), the second right-most digit is of the order of 10^1 (tens), the third right-most digit is of the order of 10^2 (hundreds), and so on, where ^ denotes exponent. For example,

735 = 700 + 30 + 5 = 7×10^2 + 3×10^1 + 5×10^0

We shall denote a decimal number with an optional suffix D if ambiguity arises.

8.2 Binary (Base 2) Number System

Binary number system has two symbols: 0 and 1, called bits. It is also a positional notation, for example,

10110B = 10000B + 0000B + 100B + 10B + 0B = 1×2^4 + 0×2^3 + 1×2^2 + 1×2^1 + 0×2^0

We shall denote a binary number with a suffix B. Some programming languages denote binary numbers with prefix 0b or 0B (e.g., 0b1001000), or prefix b with the bits quoted (e.g., b'10001111').

A binary digit is called a bit. Eight bits is called a byte.

8.3 Hexadecimal (Base 16) Number System

Hexadecimal number system uses 16 symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F, called hex digits. It uses positional notation, for example,

A3EH = A00H + 30H + EH = 10×16^2 + 3×16^1 + 14×16^0

We shall denote a hexadecimal number (in short, hex) with a suffix H. Some programming languages denote hex numbers with prefix 0x or 0X (e.g., 0x1A3C5F), or prefix x with hex digits quoted (e.g., x'C3A4D98B').

Each hexadecimal digit is also called a hex digit. Most programming languages accept lowercase 'a' to 'f' as well as uppercase 'A' to 'F'.

Computers use binary system in their internal operations, as they are built from binary digital electronic components with 2 states - on and off. However, writing or reading a long sequence of binary bits is cumbersome and error-prone (try to read this binary string: 1011 0011 0100 0011 0001 1101 0001 1000B, which is the same as hexadecimal B343 1D18H). Hexadecimal system is used as a compact form or shorthand for binary bits. Each hex digit is equivalent to 4 binary bits, i.e., shorthand for 4 bits, as follows:

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

Hexadecimal	Binary	Decimal
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13

Hexadecimal	Binary	Decimal		Hexadecimal	Binary	Decimal
E	1110	14		F	1111	15

8.4 Conversion from Hexadecimal to Binary

Replace each hex digit by the 4 equivalent bits (as listed in the above table), for examples,

A3C5H = 1010 0011 1100 0101B

102AH = 0001 0000 0010 1010B

8.5 Conversion from Binary to Hexadecimal

Starting from the right-most bit (least-significant bit), replace each group of 4 bits by the equivalent hex digit (pad the left-most bits with zero if necessary), for examples,

1001001010B = 0010 0100 1010B = 24AH

10001011001011B = 0010 0010 1100 1011B = 22CBH

It is important to note that hexadecimal number provides a compact form or shorthand for representing binary bits.

8.6 Conversion from Base r to Decimal (Base 10)

Given a n-digit base r number: $d_{n-1}d_{n-2}d_{n-3}...d_2d_1d_0$ (base r), the decimal equivalent is given by:

$$d_{n-1}\times r^{n-1} + d_{n-2}\times r^{n-2} + ... + d_1\times r^1 + d_0\times r^0$$

For examples,

A1C2H = 10×16^3 + 1×16^2 + 12×16^1 + 2 = 41410 (base 10)
10110B = 1×2^4 + 1×2^2 + 1×2^1 = 22 (base 10)

8.7 Conversion from Decimal (Base 10) to Base r

Use repeated division/remainder. For example,

To convert 261(base 10) to hexadecimal:
261/16 => quotient=16 remainder=5
16/16 => quotient=1 remainder=0
1/16 => quotient=0 remainder=1 (quotient=0 stop)
Hence, 261D = 105H (Collect the hex digits from the remainder in reverse order)

The above procedure is actually applicable to conversion between any 2 base systems. For example,

To convert 1023(base 4) to base 3:
1023(base 4)/3 => quotient=25D remainder=0
25D/3 => quotient=8D remainder=1
8D/3 => quotient=2D remainder=2
2D/3 => quotient=0 remainder=2 (quotient=0 stop)
Hence, 1023(base 4) = 2210(base 3)

8.8 Conversion between Two Number Systems with Fractional Part

1. Separate the integral and the fractional parts.
2. For the integral part, divide by the target radix repeatedly, and collect the remainder in reverse order.
3. For the fractional part, multiply the fractional part by the target radix repeatedly, and collect the integral part in the same order.

Example 1: Decimal to Binary

Convert 18.6875D to binary
Integral Part = 18D
18/2 => quotient=9 remainder=0
9/2 => quotient=4 remainder=1
4/2 => quotient=2 remainder=0
2/2 => quotient=1 remainder=0
1/2 => quotient=0 remainder=1 (quotient=0 stop)
Hence, 18D = 10010B
Fractional Part = .6875D
.6875*2=1.375 => whole number is 1
.375*2=0.75 => whole number is 0
.75*2=1.5 => whole number is 1
.5*2=1.0 => whole number is 1
Hence .6875D = .1011B
Combine, 18.6875D = 10010.1011B

Example 2: Decimal to Hexadecimal

Convert 18.6875D to hexadecimal
Integral Part = 18D
18/16 => quotient=1 remainder=2
1/16 => quotient=0 remainder=1 (quotient=0 stop)
Hence, 18D = 12H
Fractional Part = .6875D
.6875*16=11.0 => whole number is 11D (BH)
Hence .6875D = .BH
Combine, 18.6875D = 12.BH

9. Integer Representation

Integers are whole numbers or fixed-point numbers with the radix point fixed after the least-significant bit. They are contrast to real numbers or floating-point numbers, where the position of the radix point varies. It is important to take note that integers and floating-point numbers are treated differently in computers. They have different representation and are processed differently (e.g., floating-point numbers are processed in a so-called floating-point processor).

Computers use a fixed number of bits to represent an integer. The commonly-used bit-lengths for integers are 8-bit, 16-bit, 32-bit or 64-bit. Besides bit-lengths, there are two representation schemes for integers:

1. Unsigned Integers: can represent zero and positive integers.
2. Signed Integers: can represent zero, positive and negative integers. Three representation schemes had been proposed for signed integers:
    a. Sign-Magnitude representation
    b. 1's Complement representation
    c. 2's Complement representation

Suppose that you need a counter for counting a small quantity from 0 up to 200, you might choose the 8-bit unsigned integer scheme as there is no negative numbers involved.

9.1 n-bit Unsigned Integers

Unsigned integers can represent zero and positive integers, but not negative integers. The value of an unsigned integer is interpreted as "the magnitude of its underlying binary pattern".

Example 1: Suppose that n=8 and the binary pattern is 0100 0001B, the value of this unsigned integer is 1×2^0 + 1×2^6 = 65D.

Example 2: Suppose that n=16 and the binary pattern is 0001 0000 0000 1000B, the value of this unsigned integer is 1×2^3 + 1×2^12 = 4104D.

Example 3: Suppose that n=16 and the binary pattern is 0000 0000 0000 0000B, the value of this unsigned integer is 0.

An n-bit pattern can represent 2^n distinct integers. An n-bit unsigned integer can represent integers from 0 to (2^n)-1, as tabulated below:

n	Minimum	Maximum
8	0	(2^8)-1 (=255)

16	0	(2^16)-1 (=65,535)
32	0	(2^32)-1 (=4,294,967,295) (9+ digits)
64	0	(2^64)-1 (=18,446,744,073,709,551,615) (19+ digits)

9.2 Signed Integers

Signed integers can represent zero, positive integers, as well as negative integers. Three representation schemes are available for signed integers:

1. Sign-Magnitude representation
2. 1's Complement representation
3. 2's Complement representation

In all the above three schemes, the most-significant bit (msb) is called the sign bit. The sign bit is used to represent the sign of the integer with '0' for positive integers and '1' for negative integers. The magnitude of the integer, however, is interpreted differently in different schemes.

9.3 n-bit Sign Integers in Sign-Magnitude Representation

In sign-magnitude representation:

- The most-significant bit (msb) is the sign bit, with value of 0 representing positive integer and 1 representing negative integer.
- The remaining n-1 bits represent the magnitude (absolute value) of the integer. The absolute value of the integer is interpreted as "the magnitude of the (n-1)-bit binary pattern".

Example 1: Suppose that n=8 and the binary representation is 0 100 0001B.

Sign bit is 0 ⇒ positive

Absolute value is 100 0001B = 65D

Hence, the integer is +65D

Example 2: Suppose that n=8 and the binary representation is 1 000 0001B.

Sign bit is 1 ⇒ negative

Absolute value is 000 0001B = 1D

Hence, the integer is -1D

Example 3: Suppose that n=8 and the binary representation is 0 000 0000B.

Sign bit is 0 ⇒ positive

Absolute value is 000 0000B = 0D

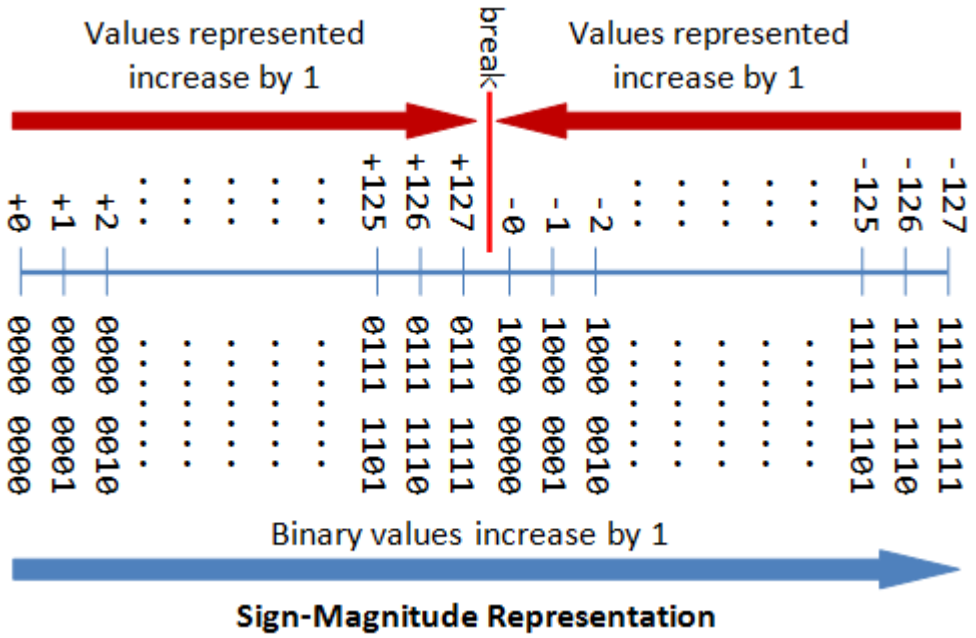
Hence, the integer is +0D

Example 4: Suppose that n=8 and the binary representation is 1 000 0000B.

Sign bit is 1 ⇒ negative

Absolute value is 000 0000B = 0D

Hence, the integer is -0D



The drawbacks of sign-magnitude representation are:

1. There are two representations (0000 0000B and 1000 0000B) for the number zero, which could lead to inefficiency and confusion.
2. Positive and negative integers need to be processed separately.

9.4 n-bit Sign Integers in 1's Complement Representation

In 1's complement representation:

- Again, the most significant bit (msb) is the sign bit, with value of 0 representing positive integers and 1 representing negative integers.
- The remaining n-1 bits represents the magnitude of the integer, as follows:
  - for positive integers, the absolute value of the integer is equal to "the magnitude of the (n-1)-bit binary pattern".
  - for negative integers, the absolute value of the integer is equal to "the magnitude of the complement (inverse) of the (n-1)-bit binary pattern" (hence called 1's complement).

Example 1: Suppose that n=8 and the binary representation 0 100 0001B.

Sign bit is 0 ⇒ positive

Absolute value is 100 0001B = 65D

Hence, the integer is +65D

Example 2: Suppose that n=8 and the binary representation 1 000 0001B.

Sign bit is 1 ⇒ negative

Absolute value is the complement of 000 0001B, i.e., 111 1110B = 126D

Hence, the integer is -126D

Example 3: Suppose that n=8 and the binary representation 0 000 0000B.

Sign bit is 0 ⇒ positive

Absolute value is 000 0000B = 0D

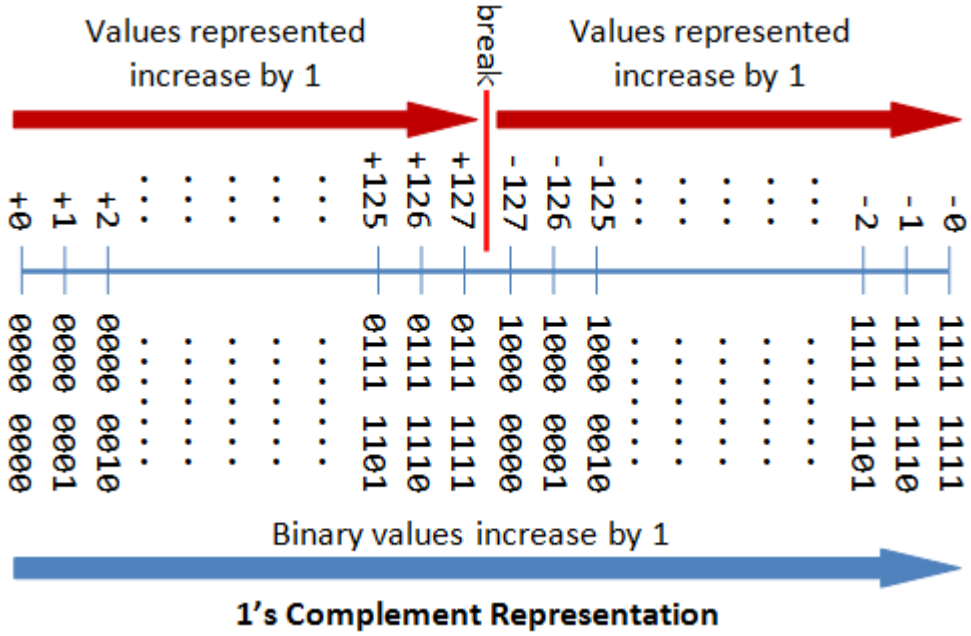
Hence, the integer is +0D

Example 4: Suppose that n=8 and the binary representation 1 111 1111B.

Sign bit is 1 ⇒ negative

Absolute value is the complement of 111 1111B, i.e., 000 0000B = 0D

Hence, the integer is -0D



Again, the drawbacks are:

1. There are two representations (0000 0000B and 1111 1111B) for zero.
2. The positive integers and negative integers need to be processed separately.

9.5 n-bit Sign Integers in 2's Complement Representation

In 2's complement representation:

- Again, the most significant bit (msb) is the sign bit, with value of 0 representing positive integers and 1 representing negative integers.
- The remaining n-1 bits represents the magnitude of the integer, as follows:

- for positive integers, the absolute value of the integer is equal to "the magnitude of the (n-1)-bit binary pattern".
- for negative integers, the absolute value of the integer is equal to "the magnitude of the complement of the (n-1)-bit binary pattern plus one" (hence called 2's complement).

Example 1: Suppose that n=8 and the binary representation 0 100 0001B.

Sign bit is 0 ⇒ positive

Absolute value is 100 0001B = 65D

Hence, the integer is +65D

Example 2: Suppose that n=8 and the binary representation 1 000 0001B.

Sign bit is 1 ⇒ negative

Absolute value is the complement of 000 0001B plus 1, i.e., 111 1110B + 1B = 127D

Hence, the integer is -127D

Example 3: Suppose that n=8 and the binary representation 0 000 0000B.

Sign bit is 0 ⇒ positive

Absolute value is 000 0000B = 0D

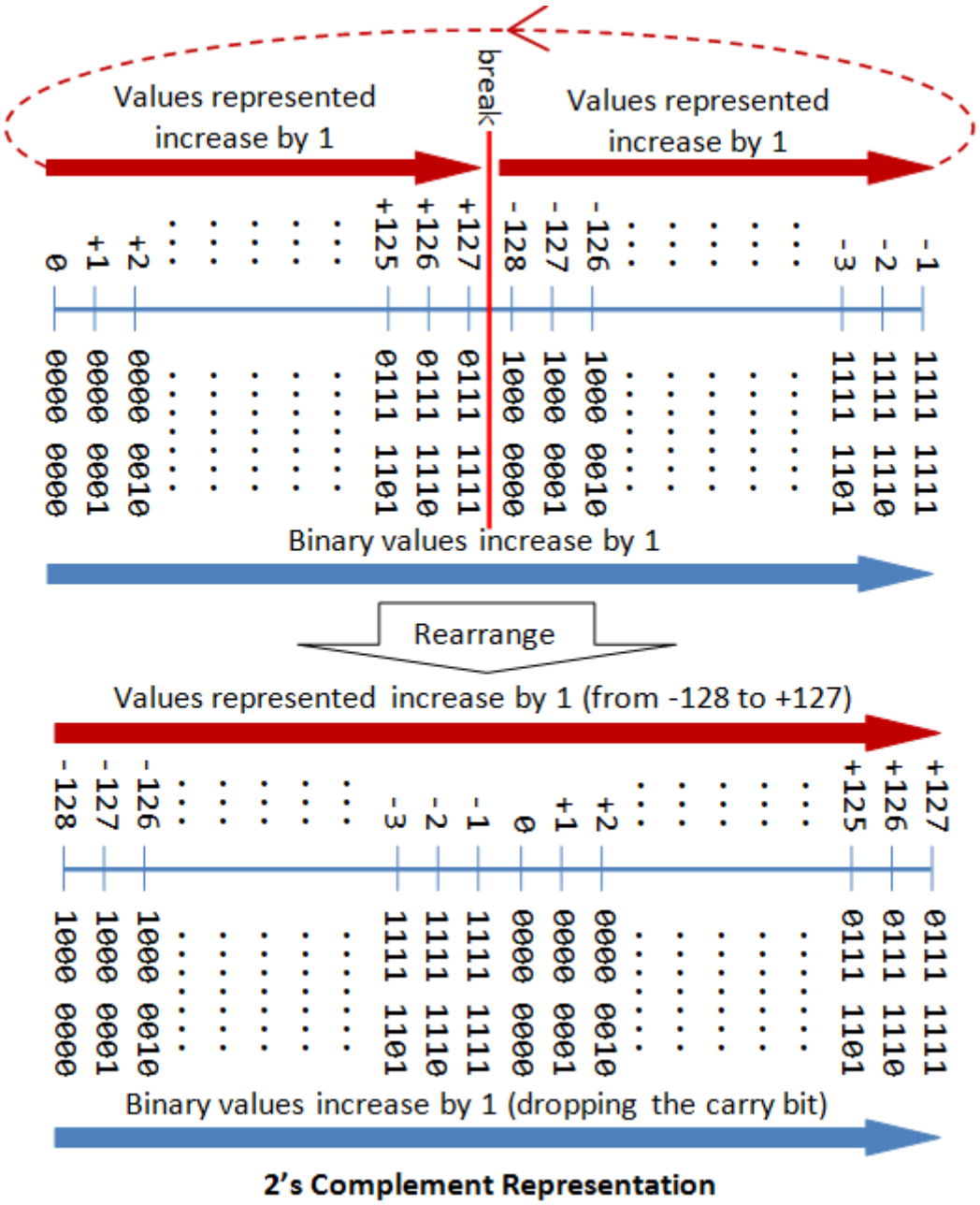
Hence, the integer is +0D

Example 4: Suppose that n=8 and the binary representation 1 111 1111B.

Sign bit is 1 ⇒ negative

Absolute value is the complement of 111 1111B plus 1, i.e., 000 0000B + 1B = 1D

Hence, the integer is -1D



2's Complement Representation

9.6 Computers use 2's Complement Representation for Signed Integers

We have discussed three representations for signed integers: signed-magnitude, 1's complement and 2's complement. Computers use 2's complement in representing signed integers. This is because:

1. There is only one representation for the number zero in 2's complement, instead of two representations in sign-magnitude and 1's complement.
2. Positive and negative integers can be treated together in addition and subtraction. Subtraction can be carried out using the "addition logic".

Example 1: Addition of Two Positive Integers: Suppose that n=8, 65D + 5D = 70D

65D → 0100 0001B

```
 5D →   0000 0101B(+
        0100 0110B   → 70D (OK)
```

Example 2: Subtraction is treated as Addition of a Positive and a Negative Integers: Suppose that n=8, 5D - 5D = 65D + (-5D) = 60D

```
65D →   0100 0001B
-5D →   1111 1011B(+
        0011 1100B   → 60D (discard carry - OK)
```

Example 3: Addition of Two Negative Integers: Suppose that n=8, -65D - 5D = (-65D) + (-5D) = -70D

```
-65D →   1011 1111B
 -5D →   1111 1011B(+
         1011 1010B   → -70D (discard carry - OK)
```

Because of the fixed precision (i.e., fixed number of bits), an n-bit 2's complement signed integer has a certain range. For example, for n=8, the range of 2's complement signed integers is -128 to +127. During addition (and subtraction), it is important to check whether the result exceeds this range, in other words, whether overflow or underflow has occurred.

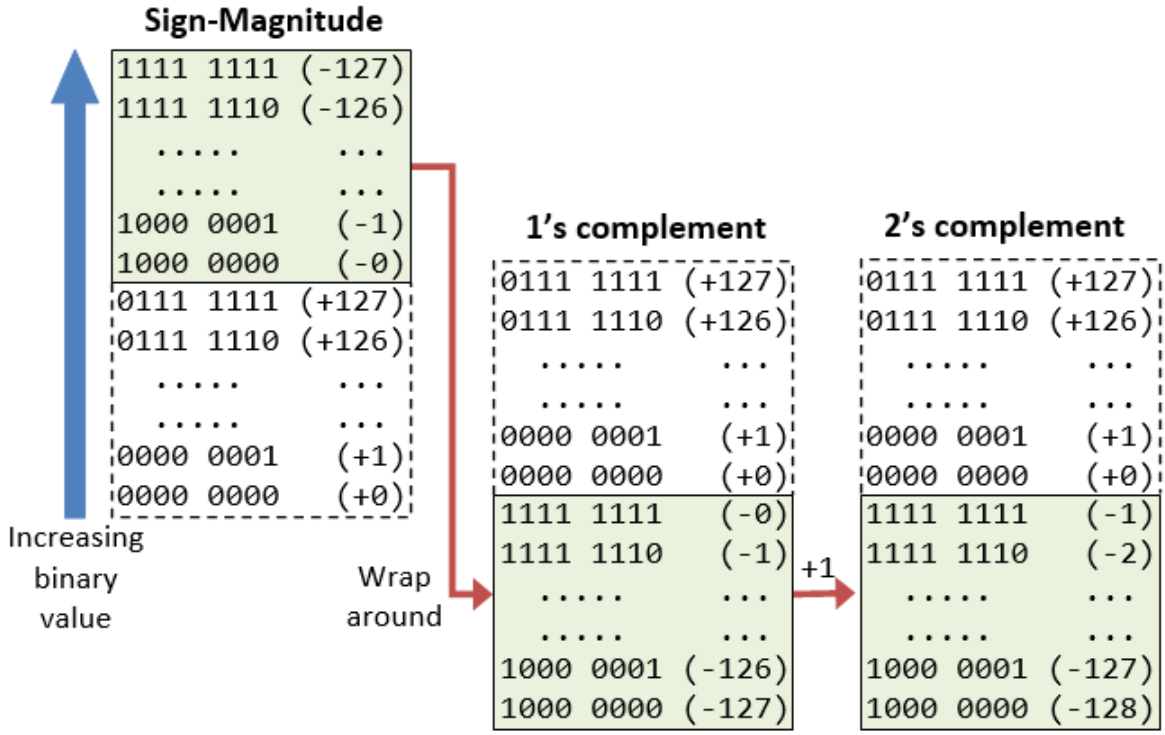
Example 4: Overflow: Suppose that n=8, 127D + 2D = 129D (overflow - beyond the range)

```
127D →   0111 1111B
  2D →   0000 0010B(+
         1000 0001B   → -127D (wrong)
```

Example 5: Underflow: Suppose that n=8, -125D - 5D = -130D (underflow - below the range)

```
-125D →   1000 0011B
  -5D →   1111 1011B(+
          0111 1110B   → +126D (wrong)
```

The following diagram explains how the 2's complement works. By re-arranging the number line, values from -128 to +127 are represented contiguously by ignoring the carry bit.



9.7 Range of n-bit 2's Complement Signed Integers

An n-bit 2's complement signed integer can represent integers from -2^(n-1) to +2^(n-1)-1, as tabulated. Take note that the scheme can represent all the integers within the range, without any gap. In other words, there is no missing integers within the supported range.

n	minimum	maximum
8	-(2^7) (=-128)	+(2^7)-1 (=+127)
16	-(2^15) (=-32,768)	+(2^15)-1 (=+32,767)
32	-(2^31) (=-2,147,483,648)	+(2^31)-1 (=+2,147,483,647)(9+ digits)
64	-(2^63) (=-9,223,372,036,854,775,808)	+(2^63)-1 (=+9,223,372,036,854,775,807)(18+ digits)

9.8 Decoding 2's Complement Numbers

1. Check the sign bit (denoted as S).
2. If S=0, the number is positive and its absolute value is the binary value of the remaining n-1 bits.
3. If S=1, the number is negative. you could "invert the n-1 bits and plus 1" to get the absolute value of negative number.

Alternatively, you could scan the remaining n-1 bits from the right (least-significant bit). Look for the first occurrence of 1. Flip all the bits to the left of that first occurrence of 1. The flipped pattern gives the absolute value. For example,

n = 8, bit pattern = 1 100 0100B

S = 1 → negative

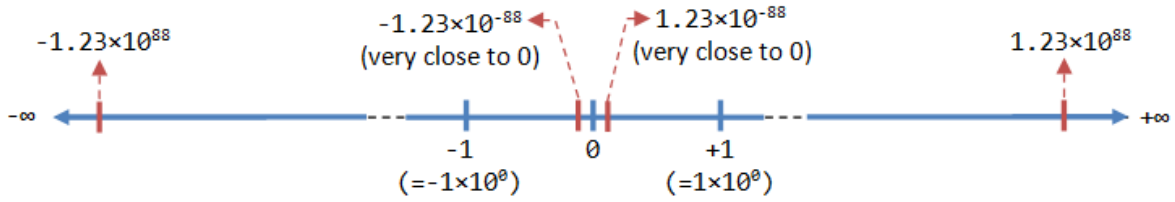
Scanning from the right and flip all the bits to the left of the first occurrence of 1 ⇒ 011

1100B = 60D

Hence, the value is -60D

10. Floating-Point Number Representation

A floating-point number (or real number) can represent a very large (1.23×10^88) or a very small (1.23×10^-88) value. It could also represent very large negative number (-1.23×10^88) and very small negative number (-1.23×10^88), as well as zero, as illustrated:



Floating-point Numbers (Decimal)

A floating-point number is typically expressed in the scientific notation, with a fraction (F), and an exponent (E) of a certain radix (r), in the form of F×r^E. Decimal numbers use radix of 10 (F×10^E); while binary numbers use radix of 2 (F×2^E).

Representation of floating point number is not unique. For example, the number 55.66 can be represented as 5.566×10^1, 0.5566×10^2, 0.05566×10^3, and so on. The fractional part can be normalized. In the normalized form, there is only a single non-zero digit before the radix point. For example, decimal number 123.4567 can be normalized as 1.234567×10^2; binary number 1010.1011B can be normalized as 1.0101011B×2^3.

It is important to note that floating-point numbers suffer from loss of precision when represented with a fixed number of bits (e.g., 32-bit or 64-bit). This is because there are infinite number of real numbers (even within a small range of says 0.0 to 0.1). On the other hand, a n-bit binary pattern can represent a finite 2^n distinct numbers. Hence, not all the real numbers can be represented. The nearest approximation will be used instead, resulted in loss of accuracy.

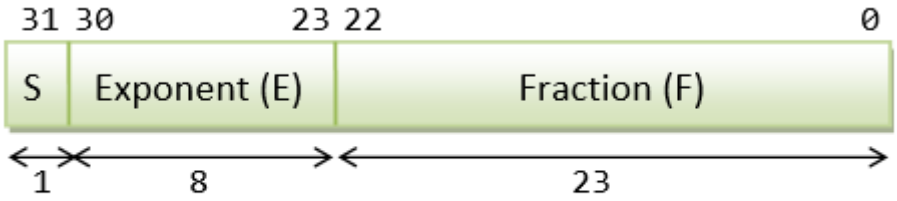
It is also important to note that floating number arithmetic is very much less efficient than integer arithmetic. It could be speed up with a so-called dedicated floating-point co-processor. Hence, use integers if your application does not require floating-point numbers.

In computers, floating-point numbers are represented in scientific notation of fraction (F) and exponent (E) with a radix of 2, in the form of F×2^E. Both E and F can be positive as well as negative. Modern computers adopt IEEE 754 standard for representing floating-point numbers. There are two representation schemes: 32-bit single-precision and 64-bit double-precision.

11. IEEE-754 32-bit Single-Precision Floating-Point Numbers

In 32-bit single-precision floating-point representation:

- The most significant bit is the sign bit (S), with 0 for positive numbers and 1 for negative numbers.
- The following 8 bits represent exponent (E).
- The remaining 23 bits represents fraction (F).



32-bit Single-Precision Floating-point Number

11.1 Normalized Form

Let's illustrate with an example, suppose that the 32-bit pattern is 1 1000 0001 011 0000 0000 0000 0000 0000, with:

- S = 1
- E = 1000 0001
- F = 011 0000 0000 0000 0000 0000

In the normalized form, the actual fraction is normalized with an implicit leading 1 in the form of 1.F. In this example, the actual fraction is 1.011 0000 0000 0000 0000 0000 = 1 + 1×2^-2 + 1×2^-3 = 1.375D.

The sign bit represents the sign of the number, with S=0 for positive and S=1 for negative number. In this example with S=1, this is a negative number, i.e., -1.375D.

In normalized form, the actual exponent is E-127 (so-called excess-127 or bias-127). This is because we need to represent both positive and negative exponent. With an 8-bit E,

ranging from 0 to 255, the excess-127 scheme could provide actual exponent of -127 to 128. In this example, E-127=129-127=2D.

Hence, the number represented is -1.375×2^2=-5.5D.

11.2 De-Normalized Form

Normalized form has a serious problem, with an implicit leading 1 for the fraction, it cannot represent the number zero! Convince yourself on this!

De-normalized form was devised to represent zero and other numbers.

For E=0, the numbers are in the de-normalized form. An implicit leading 0 (instead of 1) is used for the fraction; and the actual exponent is always -126. Hence, the number zero can be represented with E=0 and F=0 (because 0.0×2^-126=0).

We can also represent very small positive and negative numbers in de-normalized form with E=0. For example, if S=1, E=0, and F=011 0000 0000 0000 0000 0000. The actual fraction is 0.011=1×2^-2+1×2^-3=0.375D. Since S=1, it is a negative number. With E=0, the actual exponent is -126. Hence the number is -0.375×2^-126 = -4.4×10^-39, which is an extremely small negative number (close to zero).

Example 1: Suppose that IEEE-754 32-bit floating-point representation pattern is 0 10000000 110 0000 0000 0000 0000 0000.

Sign bit S = 0 ⇒ positive number

E = 1000 0000B = 128D (in normalized form)

Fraction is 1.11B (with an implicit leading 1) = 1 + 1×2^-1 + 1×2^-2 = 1.75D

The number is +1.75 × 2^(128-127) = +3.5D

Example 2: Suppose that IEEE-754 32-bit floating-point representation pattern is 1 01111110 100 0000 0000 0000 0000 0000.

Sign bit S = 1 ⇒ negative number

E = 0111 1110B = 126D (in normalized form)

Fraction is 1.1B (with an implicit leading 1) = 1 + 2^-1 = 1.5D

The number is -1.5 × 2^(126-127) = -0.75D

Example 3: Suppose that IEEE-754 32-bit floating-point representation pattern is 1 01111110 000 0000 0000 0000 0000 0001.

Sign bit S = 1 ⇒ negative number

E = 0111 1110B = 126D (in normalized form)

Fraction is 1.000 0000 0000 0000 0000 0001B (with an implicit leading 1) = 1 + 2^-23

The number is -(1 + 2^-23) × 2^(126-127) = -0.5000000596046447753906 25 (may not be exact in decimal!)

Example 4 (De-Normalized Form): Suppose that IEEE-754 32-bit floating-point representation pattern is 1 00000000 000 0000 0000 0000 0000 0001.

Sign bit S = 1 ⇒ negative number

E = 0 (in de-normalized form)

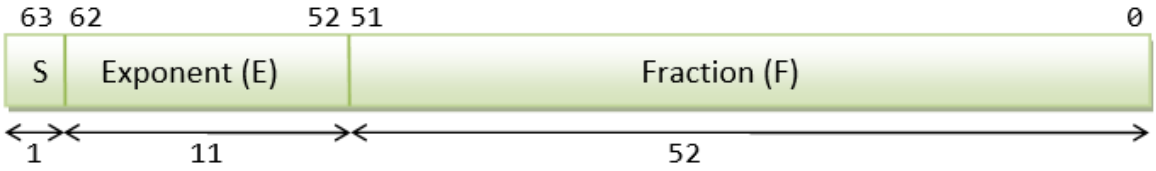
Fraction is 0.000 0000 0000 0000 0000 0001B (with an implicit leading 0) = 1×2^-23

The number is -2^-23 × 2^(-126) = -2×(-149) ≈ -1.4×10^-45

11.3 IEEE-754 64-bit Double-Precision Floating-Point Numbers

The representation scheme for 64-bit double-precision is similar to the 32-bit single-precision:

- The most significant bit is the sign bit (S), with 0 for positive numbers and 1 for negative numbers.
- The following 11 bits represent exponent (E).
- The remaining 52 bits represents fraction (F).



64-bit Double-Precision Floating-point Number

The value (N) is calculated as follows:

- Normalized form: For 1 ≤ E ≤ 2046, N = (-1)^S × 1.F × 2^(E-1023).
- Denormalized form: For E = 0, N = (-1)^S × 0.F × 2^(-1022). These are in the denormalized form.
- For E = 2047, N represents special values, such as ±INF (infinity), NaN (not a number).

11.4 More on Floating-Point Representation

There are three parts in the floating-point representation:

- The sign bit (S) is self-explanatory (0 for positive numbers and 1 for negative numbers).
- For the exponent (E), a so-called bias (or excess) is applied so as to represent both positive and negative exponent. The bias is set at half of the range. For single precision with an 8-bit exponent, the bias is 127 (or excess-127). For double precision with a 11-bit exponent, the bias is 1023 (or excess-1023).

- The fraction (F) (also called the mantissa or significand) is composed of an implicit leading bit (before the radix point) and the fractional bits (after the radix point). The leading bit for normalized numbers is 1; while the leading bit for denormalized numbers is 0.

11.5 Normalized Floating-Point Numbers

In normalized form, the radix point is placed after the first non-zero digit, e,g., 9.8765D×10^-23D, 1.001011B×2^11B. For binary number, the leading bit is always 1, and need not be represented explicitly - this saves 1 bit of storage.

In IEEE 754's normalized form:

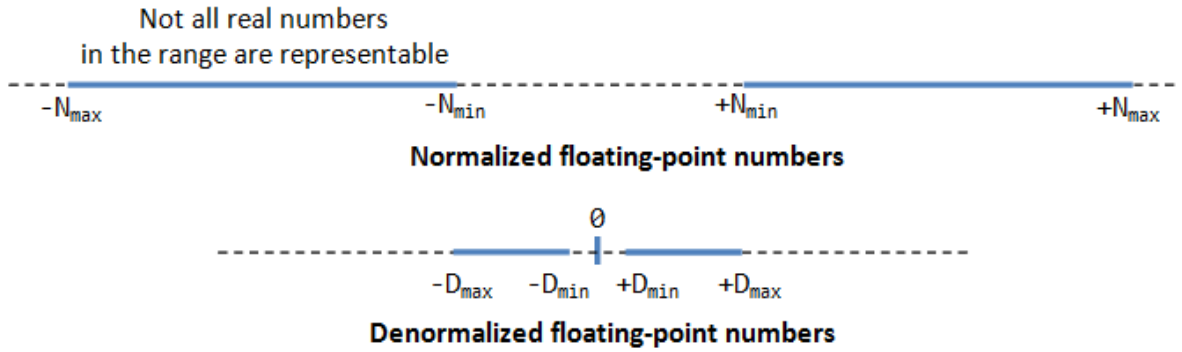
- For single-precision, 1 ≤ E ≤ 254 with excess of 127. Hence, the actual exponent is from -126 to +127. Negative exponents are used to represent small numbers (< 1.0); while positive exponents are used to represent large numbers (> 1.0).
  N = (-1)^S × 1.F × 2^(E-127)
- For double-precision, 1 ≤ E ≤ 2046 with excess of 1023. The actual exponent is from -1022 to +1023, and
  N = (-1)^S × 1.F × 2^(E-1023)

Take note that n-bit pattern has a finite number of combinations (=2^n), which could represent finite distinct numbers. It is not possible to represent the infinite numbers in the real axis (even a small range says 0.0 to 1.0 has infinite numbers). That is, not all floating-point numbers can be accurately represented. Instead, the closest approximation is used, which leads to loss of accuracy.

The minimum and maximum normalized floating-point numbers are:

Precision	Normalized N(min)	Normalized N(max)
Single	0080 0000H 0 00000001 00000000000000000000000B E = 1, F = 0 N(min) = 1.0B × 2^-126 (≈1.17549435 × 10^-38)	7F7F FFFFH 0 11111110 00000000000000000000000B E = 254, F = 0 N(max) = 1.1...1B × 2^127 = (2 - 2^-23) × 2^127 (≈3.4028235 × 10^38)
Double	0010 0000 0000 0000H N(min) = 1.0B × 2^-1022	7FEF FFFF FFFF FFFFH N(max) = 1.1...1B × 2^1023 = (2 - 2^-

	(≈2.2250738585072014 × 10^-308)	52) × 2^1023 (≈1.7976931348623157 × 10^308)



11.6 Denormalized Floating-Point Numbers

If E = 0, but the fraction is non-zero, then the value is in denormalized form, and a leading bit of 0 is assumed, as follows:

- For single-precision, E = 0,

  N = (-1)^S × 0.F × 2^(-126)

- For double-precision, E = 0,

  N = (-1)^S × 0.F × 2^(-1022)

Denormalized form can represent very small numbers closed to zero, and zero, which cannot be represented in normalized form, as shown in the above figure.

The minimum and maximum of denormalized floating-point numbers are:

Precision	Denormalized D(min)	Denormalized D(max)
Single	0000 0001H 0 00000000 00000000000000000000001B E = 0, F = 00000000000000000000001B D(min) = 0.0...1 × 2^-126 = 1 × 2^-23 × 2^-126 = 2^-149 (≈1.4 × 10^-45)	007F FFFFH 0 00000000 11111111111111111111111B E = 0, F = 11111111111111111111111B D(max) = 0.1...1 × 2^-126 = (1-2^-23)×2^-126 (≈1.1754942 × 10^-38)
Double	0000 0000 0000 0001H D(min) = 0.0...1 × 2^-1022 = 1 × 2^-52 × 2^-1022 = 2^-1074 (≈4.9 × 10^-324)	001F FFFF FFFF FFFFH D(max) = 0.1...1 × 2^-1022 = (1-2^-52)×2^-1022 (≈4.4501477170144023 × 10^-308)

11.7 Special Values

Zero: Zero cannot be represented in the normalized form, and must be represented in denormalized form with E=0 and F=0. There are two representations for zero: +0 with S=0 and -0 with S=1.

Infinity: The value of +infinity (e.g., 1/0) and -infinity (e.g., -1/0) are represented with an exponent of all 1's (E = 255 for single-precision and E = 2047 for double-precision), F=0, and S=0 (for +INF) and S=1 (for -INF).

Not a Number (NaN): NaN denotes a value that cannot be represented as real number (e.g. 0/0). NaN is represented with Exponent of all 1's (E = 255 for single-precision and E = 2047 for double-precision) and any non-zero fraction.

12. Character Encoding

In computer memory, character are "encoded" (or "represented") using a chosen "character encoding schemes" (aka "character set", "charset", "character map", or "code page").

For example, in ASCII (as well as Latin1, Unicode, and many other character sets):

- code numbers 65D (41H) to 90D (5AH) represents 'A' to 'Z', respectively.
- code numbers 97D (61H) to 122D (7AH) represents 'a' to 'z', respectively.
- code numbers 48D (30H) to 57D (39H) represents '0' to '9', respectively.

It is important to note that the representation scheme must be known before a binary pattern can be interpreted. E.g., the 8-bit pattern "0100 0010B" could represent anything under the sun known only to the person encoded it.

The most commonly-used character encoding schemes are: 7-bit ASCII (ISO/IEC 646) and 8-bit Latin-x (ISO/IEC 8859-x) for western european characters, and Unicode (ISO/IEC 10646) for internationalization (i18n).

A 7-bit encoding scheme (such as ASCII) can represent 128 characters and symbols. An 8-bit character encoding scheme (such as Latin-x) can represent 256 characters and symbols; whereas a 16-bit encoding scheme (such as Unicode UCS-2) can represents 65,536 characters and symbols.

12.1 7-bit ASCII Code (aka US-ASCII, ISO/IEC 646, ITU-T T.50)

- ASCII (American Standard Code for Information Interchange) is one of the earlier character coding schemes.

- ASCII is originally a 7-bit code. It has been extended to 8-bit to better utilize the 8-bit computer memory organization. (The 8th-bit was originally used for parity check in the early computers.)
- Code numbers 32D (20H) to 126D (7EH) are printable (displayable) characters as tabulated (arranged in hexadecimal and decimal) as follows:

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Dec	0	1	2	3	4	5	6	7	8	9
3			SP	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~			

  - Code number 32D (20H) is the blank or space character.
  - '0' to '9': 30H-39H (0011 0001B to 0011 1001B) or (0011 xxxxB where xxxx is the equivalent integer value)

- 'A' to 'Z': 41H-5AH (0101 0001B to 0101 1010B) or (010x xxxxB). 'A' to 'Z' are continuous without gap.
- 'a' to 'z': 61H-7AH (0110 0001B to 0111 1010B) or (011x xxxxB). 'A' to 'Z' are also continuous without gap. However, there is a gap between uppercase and lowercase letters. To convert between upper and lowercase, flip the value of bit-5.

- Code numbers 0D (00H) to 31D (1FH), and 127D (7FH) are special control characters, which are non-printable (non-displayable), as tabulated below. Many of these characters were used in the early days for transmission control (e.g., STX, ETX) and printer control (e.g., Form-Feed), which are now obsolete. The remaining meaningful codes today are:
  - 09H for Tab ('\t').
  - 0AH for Line-Feed or newline (LF or '\n') and 0DH for Carriage-Return (CR or 'r'), which are used as line delimiter (aka line separator, end-of-line) for text files. There is unfortunately no standard for line delimiter: Unixes and Mac use 0AH (LF or "\n"), Windows use 0D0AH (CR+LF or "\r\n"). Programming languages such as C/C++/Java (which was created on Unix) use 0AH (LF or "\n").
  - In programming languages such as C/C++/Java, line-feed (0AH) is denoted as '\n', carriage-return (0DH) as '\r', tab (09H) as '\t'.

DEC	HEX	Meaning		DEC	HEX	Meaning	
0	00	NUL	Null	17	11	DC1	Device Control 1
1	01	SOH	Start of Heading	18	12	DC2	Device Control 2
2	02	STX	Start of Text	19	13	DC3	Device Control 3
3	03	ETX	End of Text	20	14	DC4	Device Control 4
4	04	EOT	End of Transmission	21	15	NAK	Negative Ack.
5	05	ENQ	Enquiry	22	16	SYN	Sync. Idle
6	06	ACK	Acknowledgment	23	17	ETB	End of Transmission
7	07	BEL	Bell	24	18	CAN	Cancel
8	08	BS	Back Space '\b'	25	19	EM	End of Medium
9	**09**	**HT**	**Horizontal Tab '\t'**	26	1A	SUB	Substitute
10	**0A**	**LF**	**Line Feed '\n'**	27	1B	ESC	Escape

11	0B	VT	Vertical Feed	28	1C	IS4	File Separator
12	0C	FF	Form Feed 'f'	29	1D	IS3	Group Separator
13	**0D**	**CR**	**Carriage Return '\r'**	30	1E	IS2	Record Separator
14	0E	SO	Shift Out	31	1F	IS1	Unit Separator
15	0F	SI	Shift In				
16	10	DLE	Datalink Escape	127	7F	DEL	Delete

12.2 8-bit Latin-1 (aka ISO/IEC 8859-1)

ISO/IEC-8859 is a collection of 8-bit character encoding standards for the western languages.

ISO/IEC 8859-1, aka Latin alphabet No. 1, or Latin-1 in short, is the most commonly-used encoding scheme for western european languages. It has 191 printable characters from the latin script, which covers languages like English, German, Italian, Portuguese and Spanish. Latin-1 is backward compatible with the 7-bit US-ASCII code. That is, the first 128 characters in Latin-1 (code numbers 0 to 127 (7FH)), is the same as US-ASCII. Code numbers 128 (80H) to 159 (9FH) are not assigned. Code numbers 160 (A0H) to 255 (FFH) are given as follows:

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	SHY	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

ISO/IEC-8859 has 16 parts. Besides the most commonly-used Part 1, Part 2 is meant for Central European (Polish, Czech, Hungarian, etc), Part 3 for South European (Turkish, etc), Part 4 for North European (Estonian, Latvian, etc), Part 5 for Cyrillic, Part 6 for Arabic, Part 7 for Greek, Part 8 for Hebrew, Part 9 for Turkish, Part 10 for Nordic, Part 11 for Thai, Part 12 was abandon, Part 13 for Baltic Rim, Part 14 for Celtic, Part 15 for French, Finnish, etc. Part 16 for South-Eastern European.

12.3 Other 8-bit Extension of US-ASCII (ASCII Extensions)

Beside the standardized ISO-8859-x, there are many 8-bit ASCII extensions, which are not compatible with each other's.

ANSI (American National Standards Institute) (aka **Windows-1252**, or Windows Codepage 1252): for Latin alphabets used in the legacy DOS/Windows systems. It is a superset of ISO-8859-1 with code numbers 128 (80H) to 159 (9FH) assigned to displayable characters, such as "smart" single-quotes and double-quotes. A common problem in web browsers is that all the quotes and apostrophes (produced by "smart quotes" in some Microsoft software) were replaced with question marks or some strange symbols. It is because the document is labeled as ISO-8859-1 (instead of Windows-1252), where these code numbers are undefined. Most modern browsers and e-mail clients treat charset ISO-8859-1 as Windows-1252 in order to accommodate such mis-labeling.

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	€		‚	*ƒ*	„	…	†	‡	ˆ	‰	Š	‹	Œ		Ž	
9		‘	’	“	”	•	–	—		™	š	›	œ		ž	Ÿ

EBCDIC (Extended Binary Coded Decimal Interchange Code): Used in the early IBM computers.

13. Machine Instructions

Machine Instructions are commands or programs written in machine code of a machine (computer) that it can recognize and execute. A machine instruction consists of several bytes in memory that tells the processor to perform one machine.

A computer program consist of a sequence of small steps, such as adding two numbers, testing for a particular condition, reading a character from the keyboard, or sending a character to be displayed on a display screen. A computer must have the instruction capable of performing four types of operations

1. Data transfers between the memory and the registers (MOV, PUSH, POP, XCHG).
2. Arithmetic and logic operations on data (ADD, SUB, MUL, DIV, AND, OR, NOT).
3. Program sequencing and control (CALL, RET, LOOP, INT).
4. I/0 transfers (IN, OUT).

13.1 Register Transfer Notation (RTN)

This is used to describe the transfer of information from one location to another location inside the computer. In this, the source which is always a value, is specified on the

right hand side of '←' whereas the destination, which is always a processor register, is specified on the left-hand side. Its syntax is

Register ← Source

Where the **Source** can be a processor register, an I/O register, a memory location a combination of any of these locations. However, **Register** is always a processor register.

RTN uses square brackets to indicate the contents of a location. These brackets are placed only around the source.

Examples:

Consider the instruction R5 ← [R0] + [R1]

This instruction adds the contents of 'R0' and 'R1' and moves the result in 'R5'. Actually, it overwrites the old values of 'R5' with the sum of 'R0' and 'R1'.

Consider another instruction R0 ← [MEM]

This instruction simply stores, the contents of the memory location 'MEM' into 'R0'. Again, it overwrites the old value of 'R0' by the contents of 'MEM'.

13.2 Assembly Language Notation (ALN)

The assembly language notation uses assembly language format to describe the transfer of information. Its syntax is,

Opcode Source, Register

Where, **Opcode** is the operation code and specified the operation to be performed such as Add, Move and Call. The **Source** operand can be a memory location or a register, whereas the destination is always a register.

Examples:

Consider the instruction, Move MEM, R0

This instruction performs the same operation performed by the instruction R0←[MEM] represented in RTN. It can be observed that, in assembly language notation the source comes first and the destination comes next in the instruction (this is completely opposite in RTN). Moreover, no square brackets are used to explicitly to indicate the contents of the memory location 'MEM'.

Consider another instruction, Add R0, R1, R5

This instruction performs the same operation as performed by the instruction R5 ← [R0] + [R1] represented in RTN.

13.3 Instruction execution

Execution of an instruction is a 2 step process. The first step is called “instruction fetch”. In this step, the instruction is fetched from the memory location specified in a register called “Program Counter (PC)”. The fetched instruction is stored in the register called “Instruction Register (IR)”. The next step is called “Instruction Execute (IE)”. In this step, the operation code and operands are fetched from IR, the operation is being performed and the result is stored in the destination. During this step, PC is updated.

13.4 Straight line sequencing

When PC is updated based on the instruction length then the program is said to have straight line sequencing. Because, the addresses of the instructions are successive address resulting in execution of instructions in a straight line. For example, if PC=2000, word-length=32 bits and the memory is byte-addressable then the length of one instruction is 4-bytes. The successive addresses would be 2000, 2004, 2008, etc.

13.5 Branching

Most of the time, PC is updated sequentially. However, sometimes it may be necessary to update PC in a different way. Consider a program to add first 10 natural numbers. The program should have 10 Add instructions as shown

Location	Instruction
--	--
2000	Add N1, R0
2004	Add N2, R0
--	--
2034	Add N10, R0
2038	Move R0, SUM

Assuming that the memory locations N1, N2,…., N10 hold the numbers 1,2,….10. And initially R0=0. In this program PC is updated sequentially.

Branching is a basic concept in computer science. It means an instruction that tells a computer to begin executing a different part of a program rather than executing statements one-by-one. Branching is implemented as a series of control flow statements in high-level programming languages.

13.6 Condition code flags

The conditional branch instructions use few flags called “condition code flags” in order to test the condition. Generally, there are four conditional code flags. They are N (Negative) flag, Z (Zero) flag, V (Overflow) flag and C (Carry) flag. These flags are together

stored in a register called “condition code register” or “status register”, using one bit for each flag. They are set/reset by the processor depending upon the result of an operation.

13.6.1 N Flag: This indicates whether the result of an arithmetic or logic operation is negative. It is set to when the result is negative. Otherwise, it is cleared to ‘0 (Zero)’. Instructions such as Move, Load and Store also affect this flag. Because, the value that is moved, loaded or stored could be negative.

13.6.2 Z Flag: This indicates whether the result of arithmetic, logic or even a data transfer operation is zero. It is set to one (1) when result is zero. Otherwise, it is cleared to ‘0’. Using the N and Z flags, the conditional branch instruction can trigger a branch even when the previously moved, loaded, or stored operand was negative or zero. For this purpose, some computers also provide a special test instruction.

13.6.3 V Flag: This indicates whether the size of the result of an arithmetic operation is more than the operand size used by the system. It is set to ‘1’ when the arithmetic overflow occurs. Otherwise, it is cleared to ‘0’. A special branch instruction called “Branch If Overflow” is used to branch to a location when the V flag is 1. Moreover, an interrupt is automatically generated when V=1.

13.6.4 C Flag: This indicates whether the leftmost bit of the result generates a carry. It is set to 1 when a carry is generated during arithmetic operations. Otherwise, it is cleared to ‘0’.

14. Basic instruction types

There are mainly four types of instructions available which are explained as follows.

14.1 Zero-address instructions: These instructions do not use operands.

Operation

The source operand is pushed into the stack using the push instruction. After the operation is performed, the result is popped from the stack and stored in destination using the pop instruction.

Example: Consider the instruction M← [x] + [y]. This instruction can be implemented using the one zero address instruction and three stack instructions, which are one-address instruction as follows,

Push x

Push y

Add zero-address instruction

Pop M

In the first instruction, ‘x’ is pushed into stack. Then, in next instruction ‘y’ is pushed into stack. The third instruction, which is a zero-address instruction, adds the contents of the stack i.e., ‘x’ and ‘y’ and pushes the result back on the stack. The last instruction pops the result from stack and store it in M.

14.2 One-address instructions: These instructions have only the source operand explicitly specified.

Operation Source

Where, the **Source** is memory location. The destination of the operation is usually the processor register “Accumulator (AC)”. The two-address instructions do not completely solve the problem of addressing can be solved by using only one memory operand in each instruction.

Example: Consider the instruction M←[x]+[y]. This instruction can be implemented using the following sequence of one-address instructions,

```
Load x
Load y
Store M
```

The first instruction loads the contents of ‘x’ into AC. The next instruction adds ‘y’ to AC. Finally, the **Store** instruction stores the contents of AC in M. Here ‘M’ is the destination location for store instruction.

14.3 Two-address instructions: These instructions have one source and one destination.

Operation Source, Destination

The source can be a memory location or a processor register. But, the destination is always a processor register.

Two-address instructions overcome the problem of addressing in three-address instructions. They use only 2K-bits for addressing. However, the same addition operation performed by a single three-address instruction needs two-address instructions.

Example: Consider the same instruction RTN,

$$M \leftarrow [x] + [y]$$

Using two-address instructions, the above instruction is executed as,

```
Add    x, y
Move   y, M
```

It is equivalent to,

```
Move   y, M
Add    x, M
```

14.4 Three-address instructions: These instructions have two Sources and one destination

Operation Source1, Source2, Destination

Where, the two sources, source1 and source2 can be memory locations or processor registers. The destination is always a processor register.

The three-address instructions can complete an operation just in one step. However, they need 3K bits for addressing, if each operand is specified using k-bits. This could be a problem when the instruction can't fit in one word of a machine having a reasonable word-length.

Example: Consider the instruction in RTN,

$$M \leftarrow [x] + [y]$$

In this instruction, the contents 'x' and 'y' are added and the result is stored in M.

Comparison of all instructions

Zero-address	One-address	Two-address	Three-address
It doesn't contain any operand address	It contains only one operand address in the instruction	It contains two operand addresses in the instruction	It contains two addresses for two operands
It stores its operands on stack in memory	It stores its operands in the accumulator	It stores the operands by overwriting the previously stored contents in the memory location	It stores its variable names on distinct locations in memory
It is also called as stack organization	It is also called as single accumulator organization		It is also called as general register organization
Operation	Operation Source	Operation Source, Destination	Operation Source1, Source2, Destination
Example: Instruction->> Push P Push 8 Add Pop P Computes->> Increment P by 8	Example: Instruction->> ADD P Computes->> AC←AC+M[P]	Example: Instruction->> ADD 5,X Computes->> X←X+5	Example: Instruction->> ADD A,B,C Computes->> C←A+B