কোনো তথ্য অসামঞ্জস্য মনে হলে নিজের দক্ষতা দিয়ে সমাধান খুঁজুন।
যেকোনো প্রয়োজনে সরাসরি যোগাযোগ করুন।
ধন্যবাদ।

NiRoB BarMan
Find me on Facebook

Embedded System Programming

1.Introduction to Embedded System

- **What is an embedded system? Applications of embedded systems.**

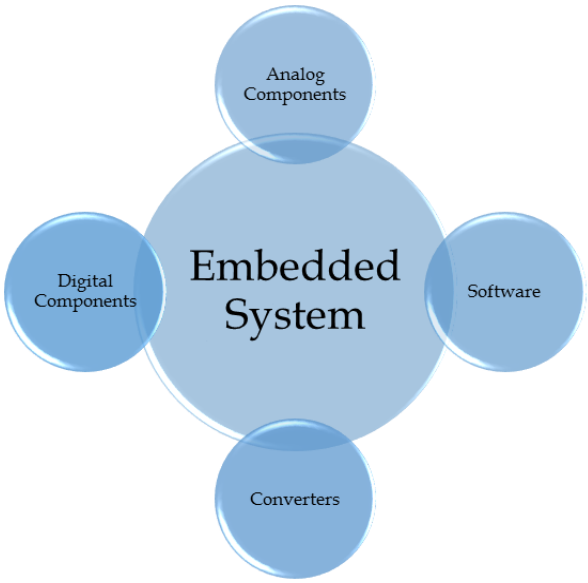
An embedded system is one that has computer hardware with software embedded in it as one of its components.

Or We can define an embedded system as “A microprocessor based system that does not look like a computer”.

Or we can say that it is “A combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function. In some cases, embedded systems are part of a larger system or product, as is the case of an antilock braking system in a car ”.

An embedded system is a special-purpose computer system designed to perform certain dedicated functions. It is usually embedded as part of a complete device including hardware and mechanical parts.

The term “embedded system” refers to a system that combines computer hardware and software. This system, like any other electronic system, requires a hardware platform, which consists of a microprocessor or microcontroller.



Embedded systems everywhere?

Embedded systems span all aspects of modern life and there are many examples of their use.

a) **Biomedical Instrumentation** – ECG Recorder, Blood cell recorder, patient monitor system

b) **Communication systems** – pagers, cellular phones, cable TV terminals, fax and transreceivers, video games and so on.

c) **Peripheral controllers of a computer** – Keyboard controller, DRAM controller, DMA controller, Printer controller, LAN controller, disk drive controller.

d) **Industrial Instrumentation** – Process controller, DC motor controller, robotic systems, CNC machine controller, closed loop engine controller, industrial moisture recorder and controller.

e) **Scientific** – digital storage system, CRT display controller, spectrum analyzer.

- ❖ What is inside an embedded system?

What is inside an embedded system?

Every embedded system consists of custom-built hardware built around a Central Processing Unit (CPU). This hardware also contains memory chips onto which the software is loaded. The software residing on the memory chip is also called the 'firmware'.

The operating system runs above the hardware, and the application software runs above the operating system. The same architecture is applicable to any computer including a desktop

computer. However, there are significant differences. It is not compulsory to have an operating system in every embedded system.

For small appliances such as remote control units, air- conditioners, toys etc., there is no need for an operating system and we can write only the software specific to that application. For applications involving complex processing, it is advisable to have an operating system.

In such a case, you need to integrate the application software with the operating system and then transfer the entire software onto the memory chip. Once the software is transferred to the memory chip, the software will continue to run for a long time and you don't need to reload new software.

❖ Networked Information Applications. Languages for Programming Embedded Systems

Networked Information Appliances

Embedded systems that are provided with network interfaces and accessed by networks such as Local Area Network or the Internet are called networked information appliances. Such embedded systems are connected to a network, typically a network running TCP/IP (Transmission Control Protocol/Internet Protocol) protocol suite, such as the Internet or a company's Intranet.

These systems have emerged in recent years. These systems run the protocol TCP/IP stack and get connected through PPP or Ethernet to an network and communicate with other nodes in the network

Example

The door lock of your home can be a small embedded system with TCP/IP and HTTP server software running on it. When your children stand in front of the door lock after they return from school, the web camera in the door-lock will send an alert to your desktop over the Internet and then you can open the door-lock through a click of the mouse.

Languages for Programming Embedded Systems

Assembly language was the pioneer for programming embedded systems till recently. Nowadays there are many more languages to program these systems. Some of the languages are C, C++, Ada, Forth, and Java together with its new enhancement J2ME.

The presence of tools to model the software in UML, SDL is sufficient to indicate the maturity of embedded software programming

The majority of software for embedded systems is still done in C language. Recent survey indicates that approximately 45% of the embedded software is still being done in C language.

C++ is also increasing its presence in embedded systems. As C++ is based on C language, thus providing programmers the object oriented methodologies to reap the benefits of such an approach.

C is very close to assembly programming and it allows very easy access to underlying hardware. A huge number of high quality compilers and debugging tools are available for the C language.

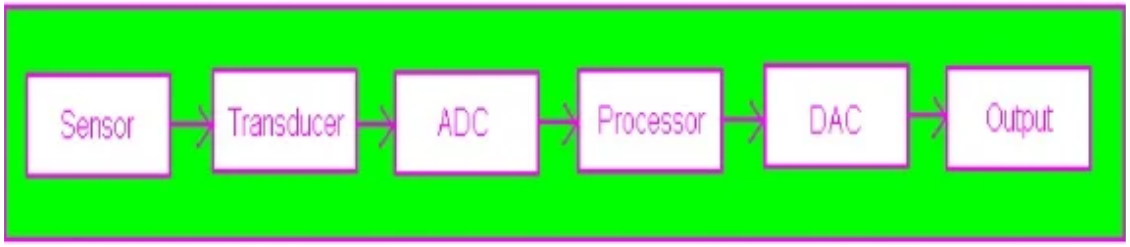
Though C++ is theoretically more efficient than C, some of its compilers have bugs due to the huge size of the language. These compilers may cause a buggy execution.

The C language can definitely claim to have more mature compilers C++. Now in order to avail the extra benefits of C++ and plus to avoid buggy execution, experts are doing efforts to identify a subset of C++ that can be used in embedded systems and this subset is called Embedded C++.

- **Design of embedded systems - a case study.**

Design of an embedded system – A Case study

To understand the design of a simple embedded system let us first consider the idea of a data acquisition system.



Data acquisition system

For example let me consider a simple case of temperature measurement embedded systems. First we must select a temperature sensor like a thermistor or AD590 or LM35 or LM335 or LM75 etc. After this the analog data is converted into digital data and at the same time proper signal conditioning is done.

2.Concept of Visual Programming

- **Define Visual Programming** and describe its key branches.

Visual programming is a sort of programming language and Any language that uses the graphics or blocks that are already defined with the code and you just need to use those blocks without worrying about the lines of code is known as a visual programming language.

In essence, visualization provides a means to better understand how a program works after it has been coded.

This is in direct contrast with visual programming where a program is actually designed by manipulating graphical representations(icons) or by a combination of icons and textual information.

Two key branches -

1. Graphical interaction systems
2. Visual programming systems

In graphical interaction systems, the user interaction with the system

Graphical interaction systems

Systems where the user guides or instructs the system in order to create the program are classified as graphical interaction systems.

Visual programming systems

Visual programming systems consist of a system in which icons, symbols, charts or forms are used to specify the program

- Define Visual Computing and Visualization.

Visual computing is a generic term for all computer science disciplines dealing with images and 3D models, such as computer graphics, image processing, **visualization**, computer vision, virtual and augmented reality and video processing. Visual computing also includes aspects of pattern recognition, human computer interaction, machine learning and digital libraries. The core challenges are the acquisition, processing, analysis and rendering of visual information (mainly images and video). Application areas include industrial quality control, medical image processing and visualization, surveying, robotics, multimedia systems, virtual heritage, special effects in movies and television, and computer games.

Visualization is **the process of representing data graphically and interacting with these representations in order to gain insight into the data**. Traditionally, computer graphics has provided a powerful mechanism for creating, manipulating, and interacting with these representations.

- **Define Visual Programming with examples.**

Introduction to **Visual Programming Language**

Any language that uses the graphics or blocks that are already defined with the code and you just need to use those blocks without worrying about the lines of code is known as a visual programming language. In today's era the majority of the programming languages are text-based i.e. we have to write the lines of code to perform the specific task like in C or C++. programming if you want to print a table of 2 then you have to write the complete text using syntax and functions of that language but in visual programming language this task is replaced by graphics or blocks like components then can be joined logically to perform the task.
Visual Programming language lets the user think in a logical manner unlike in regular programming language the user has to think about that how he/she can explain the program to the computer, to do this let's take one small analogy like if you have to code multiplication table of 2 then in regular programming language what you will do is you will take the loop and with the help of it you can print the multiplication table but in the visual basic language you just have to add the block which has the inbuilt code in it of loop and you just specify the value and you just have to think logically and your work is done without worrying about the semicolon, syntax, functions, etc.

Examples of Visual Programming Language:

There are n numbers of visual programming languages and the few which are in the top list is given below

SB MBM-vpe

- **Scratch**: With the help of this language users can create stories, games, and animations without writing any lines of code. In this you just have to create the logic and assemble the blocks.
- **Blockly**: Used to create block-based programming language and editors, and also to generate code from blocks to javascript lua dart python and PHP, etc.
- **mBlock language**: It is used in programming robots.
- **Bubble language**: It is used to create web applications.
- **Minibloq language**: It is used as a graphical programming environment for Arduino.

❖ Applications of Visual Programming Language.

Applications of Visual Programming language :

VPL can be used in multiple domains like multimedia, educational purposes, video games, automation. Let's see them in brief:

❖ **Multimedia:**- VPL helps users to create multimedia without worrying about the real code or other complex features. It narrows down to specific functions and with the help of those functions, multimedia is created.
❖ **Educational Purpose**:-Scratch VPL, etc are used to help students in their projects and make them familiar with the coding.
❖ **VideoGames**:-VPL helps to create the video games without writing lines of codes Ex-Scratch VPL is used to make video games.

❖ Advantages and disadvantages of visual programming language.

Advantages of visual programming language:

- Easy to convert **ideas** into reality. For example you don't know how to code so you can start with VPL(Visual Programming Language). and then switch to actual coding
- Visuals are easy to **grasp** i.e. to develop something in visual programming language requires less **efforts**
- It includes a variety of built in **objects** that are required while creating something using VPL.
- It is a **beginner**-friendly also anyone will be able to derive the logic without worrying about writing lines of code
- Adding a user-specific code is also available and simple as it allows to create of blocks as per the convenience of the user

Disadvantages of visual programming language:

- These languages require more **memory** as they use graphics, as a result, their execution is also slow and a large amount of memory is occupied by them.
- They can only work in an **operating** system like windows, mac, or any other operating system which supports graphics.
- As the **inbuilt** functions are not sufficient so you have to add your custom code as a result it is cumbersome.
- Only **limited** functions are present in these languages.
- Adding our custom code as a block requires coding knowledge or else you have to work with limited functions which are provided with the language.
- As a computer engineer, it is not a good idea to **use** VPL as most of the tech giants like FAANG or other tech companies work on textual languages like JAVA, HTML, etc, rather than VPL.
- For the long run VPL might not be that much **useful** as in a regular language you can explore more in it but in VPL at one point you will get bored by using the same language.

❖ Difference between regular and visual programming languages.

Difference between regular programming languages and visual programming languages:

Sr. No	Regular Languages	Visual Programming Language
1.	It is a programming language that only **uses** text.	It is a programming language that uses graphics or blocks in place of text.

2.	It is not a **beginner**-friendly language	It is a beginner-friendly language
3.	Customization and flexible applications can be created using regular languages	There is not that much customizable as the blocks or graphics that contain the codes are limited and after that, we need to add our custom code as a block.
4.	These are quite **fast** and **efficient**	This is not fast and efficient as every block has some code with it so it takes time and also it has graphics with it.
5.	The **interface** is not good i.e. only text and syntax of language we have to get familiar with.	The interface is great as the user has to just join the blocks and frame the logic without writing the code
6	requires **time** to learn as the user has to get familiar with the language syntax then code in it	any school student will be able to grasp the VPL and create the applications
7.	Requires a lot of **efforts** as a beginner to start with the language	Doesn't require a lot of effort and also the main targeted user of VPL is school level students so that they can love the coding
8.	These are quite **fast** as compared to VPL as they don't have graphics.	These are slow as compared to regular languages as they have graphics.
9.	These require less **memory** as compared to VPL	This requires more memory as it has graphics so to store them more memory is used.
10.	**Examples**: Javascript, C, C++, Java, Python Etc.	Examples: Mblock, Blockly, Scratch Etc.

Although both Regular and Visual Programming Language is in the trend in their respective categories the only difference that lies between them is that regular languages are written first and with the help of visual programming languages the parts of regular language are taken in according to its functions and the blocks and with the help of block programming is made simpler.

3.System Programming

- Define System, Programming,System Programme, **System Programming**.

What is System?

System is the collection of various components.

Ex:- College is a system.

What is Programming?

Art of designing and implementing the programs.

System Program:

These(System programs) are programs which are required for the effective execution of general user programs on computer systems.

System Programming

It(System Programming) is an art of designing and implementing system programs.

- What is Software? Types of Software.

What is Software?

Software is a collection of many programs.

Two types of software:

System software- These programs assist general use application programs. Ex:- Operating System, Assembler etc.

Application software- These are the software developed for the specific goal.

- **Diff bet application and system software**.

What is Software?

Software is a collection of many programs.

Two types of software:

System software- These programs assist general use application programs. Ex:- Operating System, Assembler etc.

Application software- These are the software developed for the specific goal.

System software	Application software
General-purpose software use to run system	**Specific**-purpose software
written in **low** level language	written in **high** level language
Fast in speed	Slow in speed as compared to system software
Small in size	Large in size
Pre-installed on system	May install at any time according to need
Does Not interact with the user	Interacts with the user directly

Complex to design	Easy to design
Difficult to write code	Easy to write code
Run independently	Does not run alone

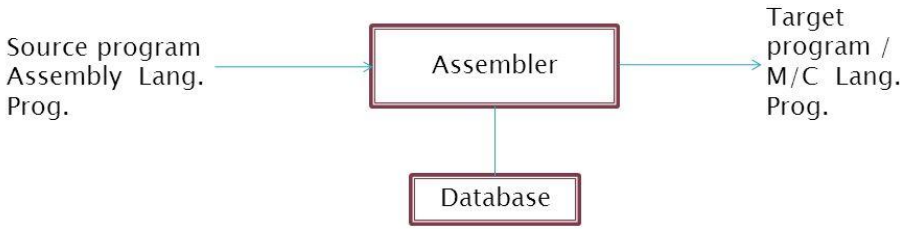
- ❖ Define system software. **Components,** goal and evaluation of it.

System software- These programs assist general use application programs. Ex:- Operating System, Assembler etc.

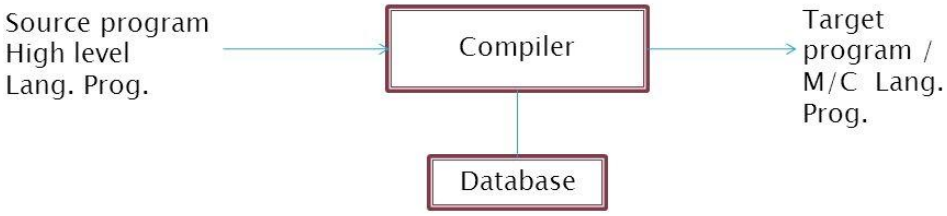
Components of System software

- Assembler
- Compiler
- Macros and Microprocessors
- Linkers And Loader

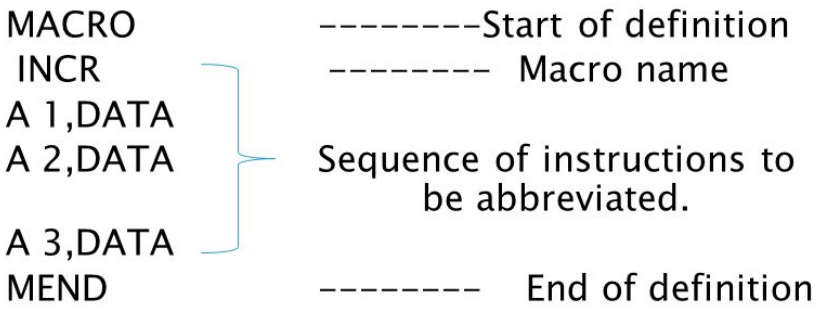
Assembler:- These are the **system programs** which will automatically **translate** the assembly language program into the machine language program.



Compiler:- These are the **system programs** which will automatically **translate** the High level language program into the machine language program.



Macro : Macro is a single line abbreviation for a group of instructions.



Linking

The Process of merging many object modules to form a single object program is called linking.

Linker

The Linker is the software program which binds(merges) many object modules to make(form) a single object program.

Loader: After the program is translated then the object program needs to be loaded on to the memory which is accomplished by another system program called loader.

Goal of System Software

The basic need of system software is to achieve the following goals :-

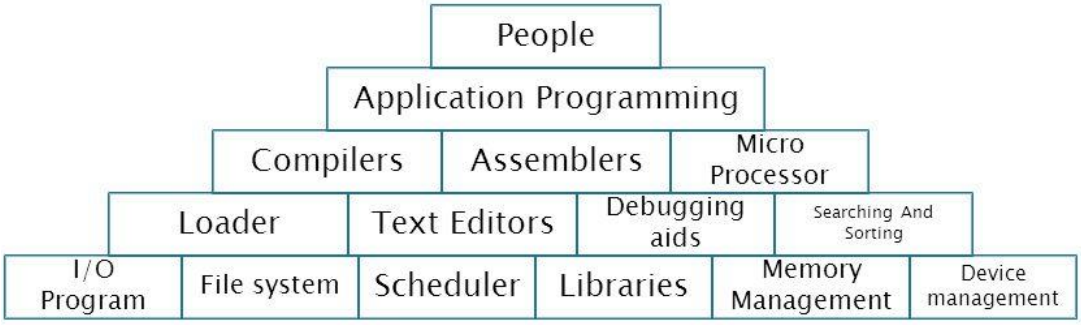
- To achieve efficient performance of the system.
- To make effective execution of a general user program.
- To make effective utilization of human resources.
- To make available new, better facilities.

Evolution of System Software and Operating System

- It is the collection of system programs which acts as an interface between the user and the computer and computer hardware.
- The purpose of an operating system is to provide an environment in which A user can execute programs in a convenient manner.

❖ Describe foundations of system programming.

PAP CAMP
LTDS
IFS LMD



Foundations of System Programming

People: People or users who **utilize and interact** with the system. System programs allow interface between the user and then computer hardware.

Application Programming: Application Programming provides services to the user directly, such as world processor , games, Excel etc.
Compilers: Compiler translates source code (high-level language) into a machine language.
Assemblers: Assembler is a software that converts assembly language to machine language.
Macro Processor: It substitutes the definition for all occurrences of the abbreviation in the program. single line abbreviation for a group of instructions.

Macros : Macro is a single line abbreviation for a group of instructions.

Loaders : The assembler produces an object program. Loadere assures that object programs are placed in memory in an executable form.

Loader: After the program is translated then the object program needs to be loaded on to the memory which is accomplished by another system program called loader.

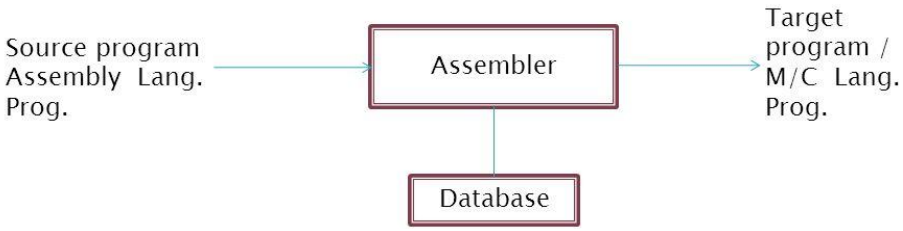
Texteditores : Texteditor allows users to edit , enter , store the text, such as Notepad.
Debugging aids: It is a technique to find and remove the errors or defects in a program.
Searching & Sorting: Searching involves **retrieving** data Sorting involves **arranging** the data in a specific order.
I/O programs : It describes any program or device that transfers data to or from a computer.
File systems: File system is a method to control how data is stored and retrieved.
Scheduler : Scheduler arranges the computer operation.
Libraries: It refers to a collection of files, functions in the system.
Memory Management : It is responsible for allocating and deallocating memory space.
Device Management: It is responsible for keeping tracks of all devices and the program.

- **Describe components of SP.**

EVOLUTION OF THE COMPLEMENTS OF A PROGRAMMING SYSTEM

Assembler: At one time a person used to write the program in machine level codes (interns of 0 and 1) and used to up the same in the memory and press a button to start the program. Programmer found it difficult to write the program in machine language and started to make use of mnemonic which they would subsequently translate into machine language. Such a mnemonic machine language is now called the assembly language. Programs known as assemblers were written to automate the translation of assembly language to machine language. The input to an assembler is called as source program and the output is a machine language translation (object program).

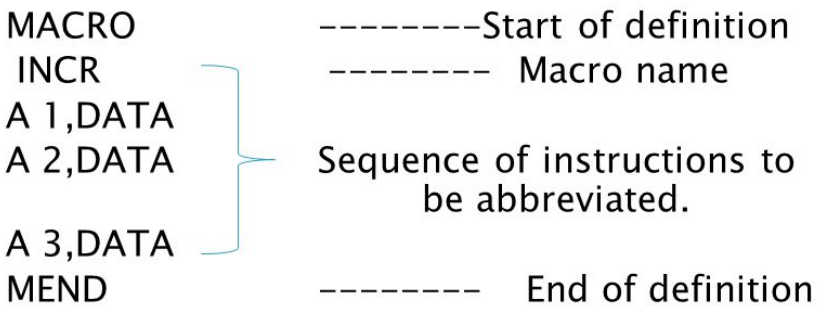
Assembler : These are the **system programs** which will automatically **translate** the assembly language program into the machine language program.The input to an assembler is called a source program and the output is a machine language translation (object program).



Loader: After the program is translated then the object program needs to be loaded on to the memory which is accomplished by another system program called loader.

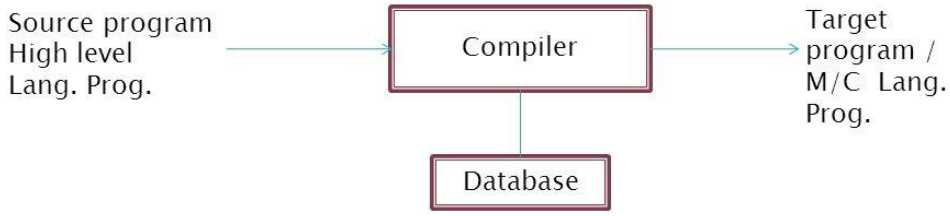
Macros and microprocessor : In case of lengthy programs the repeated identical parts use the macro processing facility of the operating system. Here the identical part of the program is abbreviated and this is treated as a macro definition and saves definition. The macro processor substitutes the definition for all occurrences of the abbreviation (macro call) in the program. Basically it is used in the design of operating systems where a number of macro calls are written.

Macros : Macro is a single line abbreviation for a group of instructions.



Compiler: A compiler is a program that accepts a program in high level language and produces an object program where an interpreter is a program that executes a source program as if it were in machine language. Basically the name of the language and the compiler are same i.e. C, C++, FORTRAN

Compiler:- These are the **system programs** which will automatically **translate** the High level language program into the machine language program.



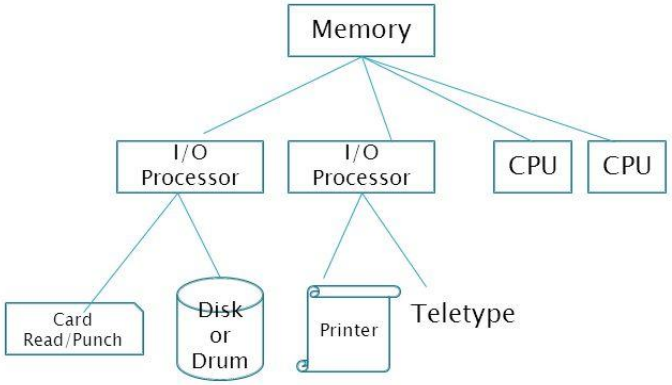
Operating system

It is the collection of system programs which acts as an interface between the user and the computer and computer hardware.

The purpose of an operating system is to provide an environment in which A user can execute programs in a convenient manner.

- **Write down the general hardware organization of computer systems.**

The general hardware organization of a computer system can be listed as below.



General Hardware Organization Of A Computer System

Memory is the device where information is stored

Bits are the unit of storage that may be either one or zero. Bits are grouped as character, bytes and words. Memory locations are specified by addresses, where each address identifies a specific character, byte and word.

Data or instructions are the content of words. A processor performs these instructions that are stored in memory. Thus instruction and data share the same storage medium.

Programs are a sequence of instructions.

Codes is a set of rules for interpreted groups of bits i.e. codes for representation of decimal digits (BCD), for characters (EBCDIC, or ASCII), or for instructions (specific processor operation codes).

Processors are the devices that operate on this information. There are two types of processors i.e. Input/Output (I/O) processors that are concerned with transfer of data between memory and peripheral devices such as disks, drums, printers and typewriters whereas the Central processing unit (CPU) is concerned with the manipulation of data stored in the memory. The I/O instructions are stored in the memory that is activated by the command from the CPU. Typically this consists of an “execute I/O” instruction whose argument is the address of an I/O

program. The CPU interprets this instruction and passes the argument to the I/O processor (commonly called I/O channels).

❖ How to get knowledge of Embedded System Programming

Every beginner needs to know about embedded system programming before starting their projects.

1. Choose Programming Language

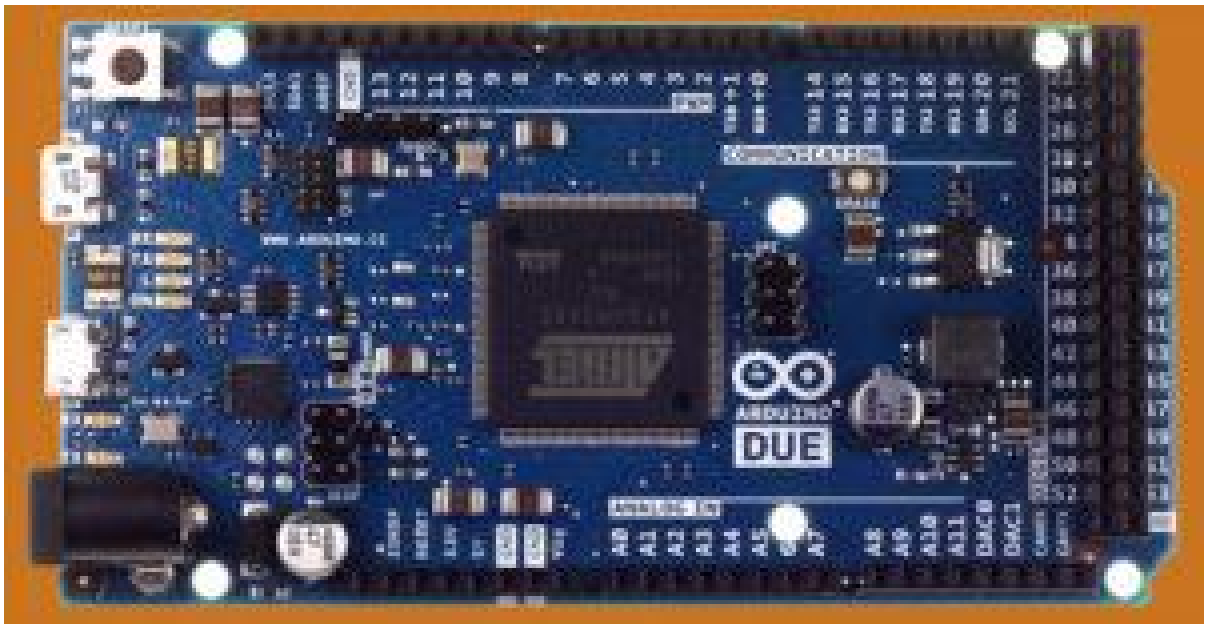
- The first step of learning Embedded System Programming is to select a programming language. Your first stage is to know all the options available and then choosing one from them according to your requirement and application.
- Embedded Systems can be programmed using both low level programming languages and high level programming languages.
- Both languages have their own advantages and uses. Assembly language is used for low level programming. For high level programming, C is used mostly.
- Complex and sophisticated systems make use of low level programming languages. The user has more control on the hardware and memory with a low level programming language.
- Small systems use high level programming languages. Such languages are easily understandable and the programs are readable and compact.
- As a beginner, you can start developing your embedded system using C language.

2. Learn C/ C++

- The second thing to do for embedded system programming is to learn the programming language. The most commonly used language is C or C++.
- Most of the embedded system products are designed such that they support C language.
- This language is easy to learn and a good start for beginners. C is preferred for embedded software development.
- Programs written in C language are readable and easy to debug. This language is efficient and provides support for Input and Output devices.
- Therefore it is necessary to learn this language if you want to go for embedded system programming.
- You should get an idea about the basics of C. know about variables, conditionals, loops, structures and functions etc.
- You can check this great C# Tutorial on our blog, if you wanna learn C#.

3. Know your Microcontroller

- Microcontroller or microprocessor is the very important part of embedded system. So the third thing that you should know is your microcontroller.
- A microcontroller will act as a CPU of your embedded system. It has RAM, ROM and some other peripherals such as timers, counters etc.
- Most commonly used microcontrollers are Arduino, PIC Microcontroller or 8051 Microcontroller etc.
- Depending upon the application, first choose a microcontroller. It depends on the hardware that you want to connect with. Also the software requirements should be kept in mind while choosing a microcontroller.
- It is the third step of learning embedded system programming. Now let's move towards fourth step of embedded system programming.

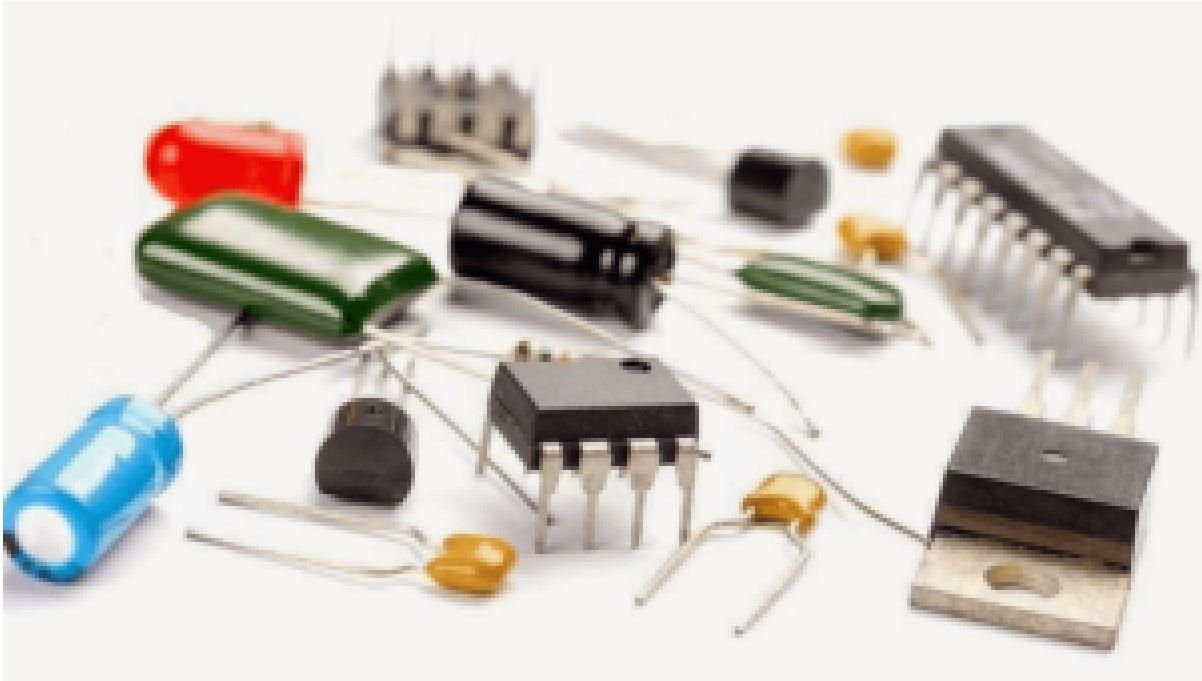


4. Know Basic Electronics

- For learning embedded system programming, you need to know about electronic devices. If you have no idea about basic electronics, it is almost impossible to design an embedded system program.
- Embedded system is not just writing a program. Your purpose is to make a complete system that can effect physical things. So it is necessary to learn hardware with software.
- For learning basic electronics, you can start with small projects instead of studying the details of all types of devices available.
- Also you should get an idea of basic terms like voltage, power, current, resistance etc.
- You can read tutorials about the concepts and also you can try starting with small circuits.
- Once you know these basics, you can work with your components and embedded system programming more efficiently.

5. Get your Equipment and Tools

- For learning embedded system programming, you will need some equipment other than microcontroller and hardware.
- You can start by having
    - jumper wires,
    - circuit boards,
    - batteries,
    - resistors
    - leds
    - soldering iron and
    - Some testing devices like DMM (digital multimeter).
- These equipment is necessary just as other parts of the system.



6. Select Components

- Selecting components is also an important part of learning embedded system programming. For each component you have a variety of types to choose from. It is important to select the one best suited for your application.
- Datasheets are available for each component. First you should learn how to read a datasheet.
- All information regarding a component can be found from its datasheet.
- You can find how to use a component, what the specifications are and what are the power requirements from a datasheet.

7. Start with Mini Projects

- The best method to learn is to practice. Before you start working on some real projects, try to work on small projects.

- Working on some mini projects will help you in getting familiar with the microcontroller and your components.
- Also you can start with microcontroller kits, it is also helpful for beginners.
- After working on such projects, you can move to your actual project.

8**. Use Simulation Tools**

- Using simulation for your embedded system programming is also very useful.
- It is impractical to design a system and develop it into hardware form before checking if it is working as required or not.
- Simulation allows the programmer to check for various conditions and to control different parameters. It is even more useful when working on large projects where you have minimum resources and also trying things after developing a project is expensive.

❖ What is OS? Write down the Importance and Functions of the operating system.

Operating System

An operating system (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs.

For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between application programs and the computer hardware. It provides common services for the execution of application programs. Examples of popular modern operating systems are: Linux, Mac OS, Microsoft Windows and UNIX.

It is the collection of system programs which acts as an interface between the user and the computer and computer hardware.

The purpose of an operating system is to provide an environment in which A user can execute programs in a convenient manner.

The main objectives / goals of an operating system are:

CEE-ogos

Convenience: It makes the computer more convenient to use

Efficiency: It allows computer system resources to be used in an efficient manner

Ability to evolve: Permit effective development, testing, and introduction of new system functions without interfering with service

Operating System Services/function

PIPM FCE RAP

Program Execution - It provides the capability to load a program into memory and to run it.

I/O Operations - Since user programs cannot execute I/O operations directly, the

operating systems must provide some means to perform I/O operations.

Process Management - A process is a program in execution. The operating system is responsible for - process creation and deletion, process suspension and resumption, process synchronization and communication.

Main-Memory Management - Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices. The operating system is responsible for the following activities in connections with memory management:

- Keep track of which parts of memory are currently being used and by whom.
- Decide which processes to load when memory space becomes available.
- Allocate and deallocate memory space as needed.

File Management - It provides the capability to read, write, create, and delete files.

Communications - Exchange of information between processes executing either on the same computer or on different systems tied together by a network.

Error Detection - It ensures correct computing by detecting errors in the CPU, memory, I/O devices, or in user programs.

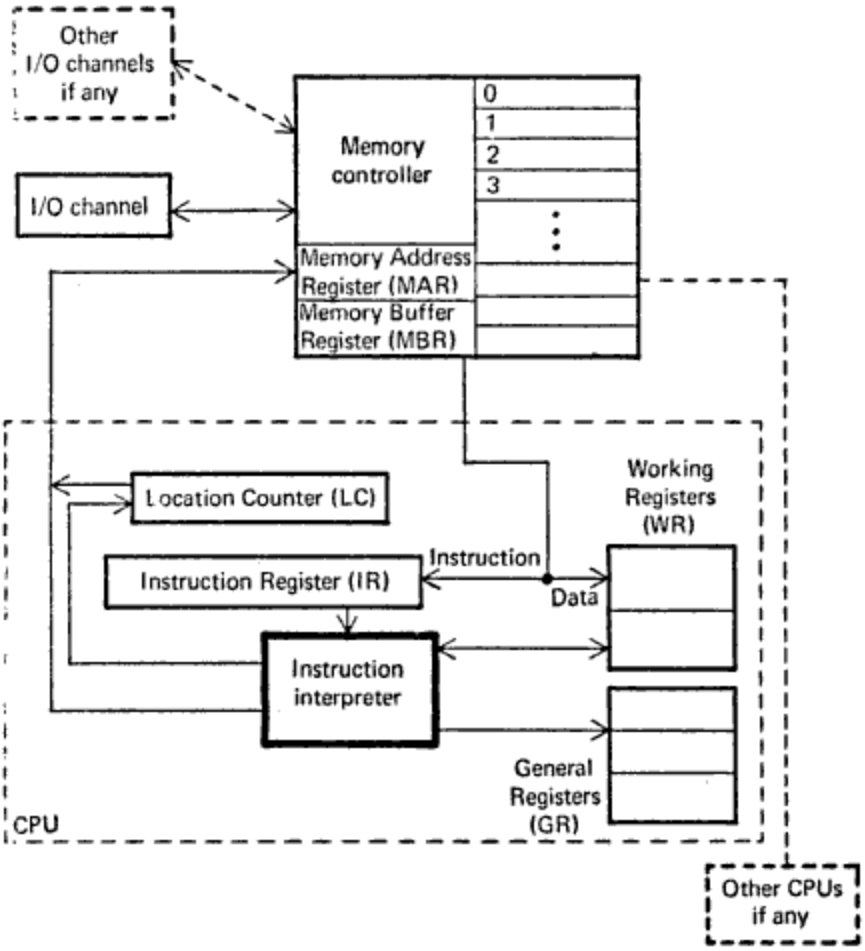
Resource Allocation - It allocates resources to multiple users or multiple jobs running at the same time.

Accounting - It keeps track of records which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics.

Protection-It ensures that all access to system resources is controlled.

4.Machine Structure

- **Draw and describe general machine structure.**



The structure above consists of -

ILI WG-gms
MMM IO

1. Instruction Interpreter
2. Location Counter
3. Instruction Register
4. Working Registers
5. General Register

Instruction interpreter:

A group of electronic circuits performs the intent of instruction fetched from memory.

Location counter:

LC, otherwise called a program counter PC or instruction counter IC, is a hardware memory device which denotes the location of the current instruction being executed.

Instruction register:

A copy of the content of the LC is stored in IR.

Working register:

are the memory devices that serve as “scratch pad” for the instruction interpreter.

General register:
are used by programmers as storage locations and for special functions.

Memory address register (MAR):
contains the address of the memory location that is to read from or stored into.

MAR (Memory Address Register) - contains address of memory location (to be read from or stored into)

Memory buffer register (MBR):
contain a copy of the content of the memory location whose address is stored in MAR. The primary interface between the memory and the CPU is through the memory buffer register.

MBR (Memory Buffer Register) - contains copy of address specified by MAR

Memory controller:
is a hardware device whose work is to transfer the content of the MBR to the core memory location whose address is stored in MAR.

Memory controller is used to transfer data between MBR & the memory location specified by MAR

I/O channels:
may be thought of as separate computers which interpret special instructions for inputting and outputting information from the memory.

The role of I/O Channels is to input or output information from memory

- **Describe step by step micro flowchart for ADD instruction**.

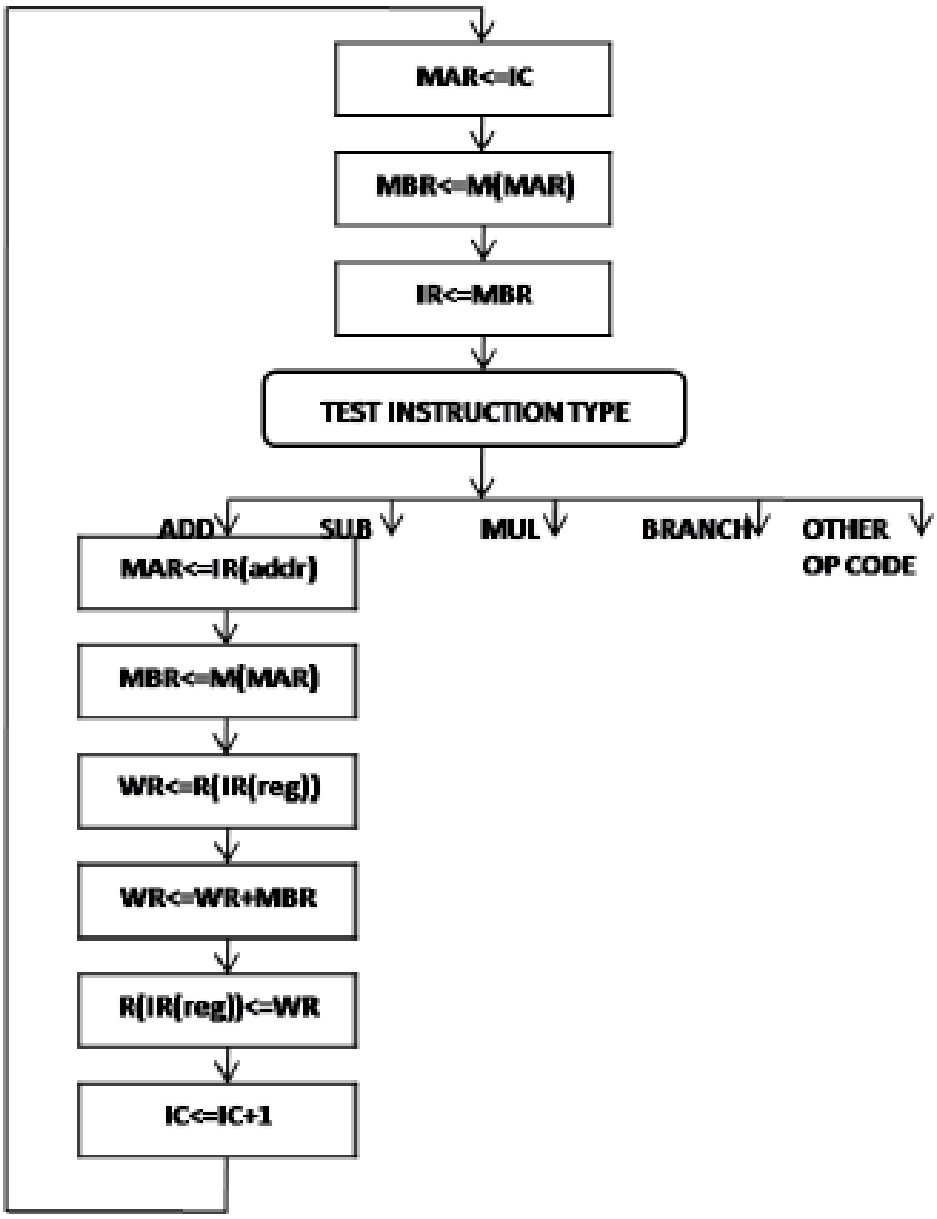


FIG: Example of micro flow chart for ADD instruction.

To illustrate how these components of machine structure interact, let us take a command ADD 2,176.

This instruction has three parts first the opcode i.e. ADD, second is the number of the register that contain the first operator, third is the memory location address that contains the second operand.

- At first, the address from the IC is **copied** to the MAR.
- Then the instruction is **fetched** to the MBR.
- The instruction is then **transferred** to the IR.
- Then the operand of the instruction is **checked** and the corresponding branch is taken, here ADD branch is chosen.
- Then the memory location of the second operand is **placed** in the MAR.
- Then the content is **placed** in MBR.
- Now the first operand is **placed** in the WR.
- Finally the sum of WR and MBR is **calculated** and then **store**d in WR.
- The content of WR is **stored** in the register that contains the first operand.
- And then IC is i**ncremented** to point to the next instruction.

❖ **Describe a general approach to a new machine.**

GENERAL APPROACHES TO A NEW MACHINES

In order to know a new machine we have a number of questions in mind. These questions can be categorized as follows.

- **Memory**: Basic unit, size and addressing scheme.
- **Registers**: Number of registers, and size, functions, interrelation of each register.
- **Data:** Types of data and their storing scheme.
- **Instruction**: Classes of instructions, allowable operations and their storing scheme.
- **Special Features**: Additional features like interrupt and protections.

❖ **Describe machine structure - 360/370(Memory**, Register, Data and **Instructions**).

All the parameters defined above will be discussed for IBM 360 and 370 machines.

Memory

The basic unit of memory in 360 and 370 is a byte (eight bits of information). It means, each addressable position in memory can contain a byte of information. There are facilities to operate on contiguous bytes in basic units. The basic units are as follows:

Unit of memory	Bytes	Length in bits
Byte	1	8
Half word	2	16
Word	4	32
Double word	8	64

A smaller unit of memory of size 4 bits is called a nibble. The size of a 360 memory is upto 224 bytes. The addressing of a 360 memory consists of three parts. Specifically the value of an

address equals the value of the offset, plus the content of the base register, plus the content of the index register.

Registers

There are a total of 16 general purpose registers of 32 bits each. In addition there are 4 floating point registers of 64 bits each. It also has a 64 bits program status word (PSW) that contains the value of the location counter, protection information and interrupt status.

The general purpose registers are basically used in arithmetic and logical operations as base registers and helps in address formation. The general purpose registers are also used as scratch pads for the programmers. Let us take instruction A 1,901(2,15).

A(opcode)1(operand in register 1),901(offset) (2(index register),15(base register))

This is how the memory locations are addressed in case of 360 and 370. The use of base registers in addressing is twofold.

First it helps the loader in the process of relocation (changing the content of base register causes the code to be relocated to the specified location).

Secondly it decreases the size of instruction as follows: since the memory of 360 is of 224 hence a total of 24 bits are required to specify a particular location of memory. This increases the size of instruction as opcode takes 8 bits and the registers require 4 bits each and the address requires a 24 bits hence the size of the instruction is 8+4+4+24=40 bits (without base register)

Fig. Addressing without the use of a base register.

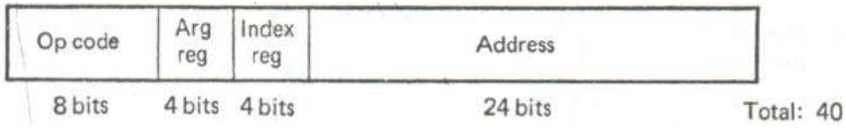
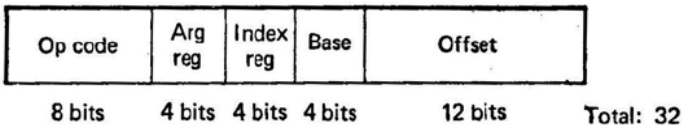


Fig. Addressing using base register.

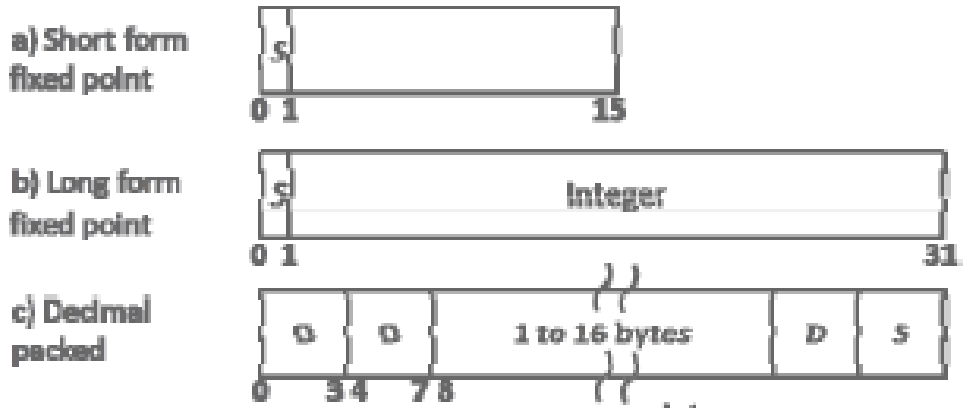


In the second scheme when we use the base register then a total of 8+4+4+4+12(offset)=32 bits are used, saving a total of 40-32=8 bits.

The only disadvantage with this method is that the offset can take value from 0 to 4,095 locations away from the core location to which the base register is pointing.

Data

The 360 may store several different types of data as is depicted in the figure.



The groups of bits stored in memory are interpreted by the 360 processor in several ways. The list of different interpretations shown in the figure are as follows.

Short form fixed integer	Long form fixed integer
Decimal packed	Unpacked
Short form floating point	Long form floating point

Logical

E.g. +541 is represented as 0000 0010 0001 1101, the first bit represents the sign i.e. + and the rest 15 bits represents the integer 15. Where -021 can be represented as 0000 0010 0001 1101 i.e. 0 2 1 and -, in decimal packed form representation. Where + 300 is represented as 0000 0001 0010 1100.

It is more convenient to represent the numbers in hexadecimal as every hexadecimal digit represents four binary digits and vice versa. Thus +300 which is B'0000 0001 0010 1100' in binary can be written as X'012C' in hexadecimal. The prefix B and X represents binary and hexadecimal respectively

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

FIGURE Hexadecimal-binary-decimal conversion

Fixed point numbers can be represented by a halfword or full word as in figure a, b. Decimal numbers can be represented in a form similar to that of binary as 12 can be represented as 0001 0010 as shown in figure c, d. Floating point numbers and logical data can be stored as in figure e, f, g.

Instruction

The 360/370 has five instruction formats. Each format is associated with a single addressing mode and has a set of operations defined for that format. While some operations are defined in multiple formats, most are not. The 360/370 has five instruction formats. Each format is associated with a single addressing mode and has a set of operations defined for that format. While some operations are defined in multiple formats, most are not.

- RR (register-register)
- RX (register-indexed)
- RS (register-storage)
- SI (storage-immediate)
- SS (storage-storage)

❖ Define assembly language. Write down its advantages.

ASSEMBLY LANGUAGE

The era of programming languages starts with machine language and end in English language, thereby making a move from the language that are best for machines that are best for programmers.

An assembly language is a type of low-level programming language that is intended to communicate directly with a computer's hardware.

So far we have machine language and mnemonic machine language. Now we will be using assembly language for the following reasons (advantages).

1. It is mnemonics, e.g. we use ST instead of the bit stream 0101000 for the store instruction.
2. Addresses are symbolic, not absolute.
3. Reading is easier.
4. Introduction of data to a program is easier.
    - Memory efficient
    - Faster in speed
    - Easy to make insertions and deletions
    - Hardware oriented
    - Require few instructions
    - Better control on hardware
    - Simple execution

The main drawback of assembly language is that it makes use of an assembler to translate a source program to object code. The assembly language for 360 is much similar to assembly languages that are meant for other machines

- ❖ Write down an assembly language program.

Ans: TITLE PROB 01.asm:DISPLAY ONE CHARACTER

```
dosseg
.model small
.stack
.code
main proc
  mov ah,01h
  int 21h

  mov ah,02h
  mov dl,al
  int 21h

  mov ah,4ch
  int 21h

 main endp
end main
```

- Define machine language with examples.

Machine code, also known as machine language, is the elemental language of computers. It is read by the computer's central processing unit (CPU), is composed of digital binary numbers and looks like a very long sequence of zeros and ones.

Each CPU has its own specific machine language. The processor reads and handles instructions, which tell the CPU to perform a simple task.

For example, the ASCII code 01000001 represents the letter "A" in machine language, yet it is shown on the screen as "A".

Different machine code is used by different processor architectures; however, machine code includes 1s and 0s.

5.MDI

- **What is MDI?**

A multiple-document interface (**MDI**) is a graphical user interface in which multiple windows reside under a single parent window. Such systems often allow child windows to embed other windows inside them as well, creating complex nested hierarchies. This contrasts with single-document interfaces (SDI) where all windows are independent of each other.

- **Describing the following terms**
  - **Frame, Client and Child Windows**

Frame, Client, and Child Windows

An MDI application has three kinds of windows: a frame window, an MDI client window, as well as a number of child windows.

The **frame window** is like the **main window** of the application: it has a sizing border, a title bar, a window menu, a minimize button, and a maximize button. The application must register a window class for the frame window and provide a window procedure to support it. An MDI application does not display output in the client area of the frame window. Instead, it displays the MDI client window.

An **MDI client window** is a special type of child window belonging to the pre registered window class MDICLIENT. The client window is a child of the frame window; it serves as the background for child windows. It also provides support for creating and manipulating child windows. For example, an MDI application can create, activate, or maximize child windows by sending messages to the MDI client window.

When the user opens or creates a document, the **client window creates a child window** for the document. The client window is the parent window of all MDI child windows in

the application. Each child window has a sizing border, a title bar, a window menu, a minimize button, and a maximize button. Because a child window is clipped, it is confined to the client window and cannot appear outside it.
An MDI application can support more than one kind of document. For example, a typical spreadsheet application enables the user to work with both charts and spreadsheets. For each type of document that it supports, an MDI application must register a child window class and provide a window procedure to support the windows belonging to that class. For more information about window classes, see Window Classes. For more information about window procedures, see Window Procedures.
Following is a typical MDI application. It is named Multipad.

➢ **Child Window Creation**

Child Window Creation

To create a child window, an MDI application either calls the CreateMDIWindow function or sends the WM_MDICREATE message to the MDI client window. A more efficient way to create an MDI child window is to call the CreateWindowEx function, specifying the WS_EX_MDICHILD extended style.
To destroy a child window, an MDI application sends a WM_MDIDESTROY message to the MDI client window.

$$MDI, APP \rightarrow CreateMDIWindow\ Function$$
$$(message)Send \rightarrow WM\_MDICREATE \rightarrow MDI\ Client\ Window$$
$$Extended\ style \rightarrow Create\ WindowEX/WS\_EX\_MDICHILD$$
$$Destroy \rightarrow WM\_MDIDESTROY$$

➢ Child Window Activation

Child Window Activation

Any number of child windows can appear in the client window at any one time, but only one can be active. The active child window is positioned in front of all other child windows, and its border is highlighted.

The user can activate an inactive child window by clicking it. An MDI application activates a child window by sending a WM_MDIACTIVATE message to the MDI client window. As the client window processes this message, it sends a WM_MDIACTIVATE message to the window procedure of the child window to be activated and to the window procedure of the child window being deactivated.

To prevent a child window from activating, handle the WM_NCACTIVATE message to the child window by returning FALSE.
The system keeps track of each child window's position in the stack of overlapping windows. This stacking is known as the Z-Order. The user can activate the next child window in the Z

order by clicking Next from the window menu in the active window. An application activates the next (or previous) child window in the Z order by sending a WM_MDINEXT message to the client window.

To retrieve the handle to the active child window, the MDI application sends a WM_MDIGETACTIVE message to the client window.

$$Active \rightarrow WM\_MDIACTIVATE(message)$$
$$Deactivated \rightarrow WM\_NCACTIVATE$$
$$Next \rightarrow WM\_MDINEXT$$
$$Retrieve \rightarrow WM\_MDIGETACTIVE$$

➢ **Multiple Document Menus**

Multiple Document Menus

The frame window of an MDI application should include a menu bar with a window menu. The window menu should include items that arrange the child windows within the client window or that close all child windows. The window menu of a typical MDI application might include the items in the following table.

Menu item	Purpose
Tile	Arranges child windows in a tile format so that each appears in its entirety in the client window.
Cascade	Arranges child windows in a cascade format. The child windows overlap one another, but the title bar of each is visible.
Arrange Icons	Arranges the icons of minimized child windows along the bottom of the client window.
Close All	Closes all child windows.

Whenever a child window is created, the system automatically appends a new menu item to the window menu. The text of the menu item is the same as the text on the menu bar of the new child window. By clicking the menu item, the user can activate the corresponding child window. When a child window is destroyed, the system automatically removes the corresponding menu item from the window menu.

The system can add up to ten menu items to the window menu. When the tenth child window is created, the system adds the More Windows item to the window menu. Clicking this

item displays the Select Window dialog box. The dialog box contains a list box with the titles of all MDI child windows currently available. The user can activate a child window by clicking its title in the list box.

If your MDI application supports several types of child windows, tailor the menu bar to reflect the operations associated with the active window. To do this, provide separate menu resources for each type of child window the application supports. When a new type of child window is activated, the application should send a WM_MDISETMENU message to the client window, passing the handle to the corresponding menu.
When no child window exists, the menu bar should contain only items used to create or open a document.

When the user is navigating through an MDI application's menus by using cursor keys, the keys behave differently than when the user is navigating through a typical application's menus. In an MDI application, control passes from the application's window menu to the window menu of the active child window, and then to the first item on the menu bar.

$$Tile \rightarrow Each\ appears\ in\ its\ entirely\ the\ client\ window$$
$$Cascade \rightarrow Overlap\ one\ another$$
$$Arrange\ Icons$$
$$Close\ All$$

➢ **Multiple Document Accelerators**

Multiple Document Accelerators

To receive and process accelerator keys for its child windows, an MDI application must include the **TranslateMDISysAccel** function in its message loop. The loop must call TranslateMDISysAccel before calling the TranslateAccelerator or DispatchMessage function. Accelerator keys on the window menu for an MDI child window are different from those for a non-MDI child window. In an MDI child window, the **ALT+ – (minus)** key combination opens the window menu, the **CTRL+F4** key combination closes the active child window, and the **CTRL+F6** key combination activates the next child window.

➢ Child window size and Arrangement

Child Window Size and Arrangement

An MDI application **controls the size and position** of its child windows by sending messages to the MDI client window.

To **maximize** the active child window, the application sends the **WM_MDIMAXIMIZE** message to the client window. When a child window is maximized, its client area completely fills the MDI client window. In addition, the system automatically hides the child window's title bar, and adds the child window's window menu icon and Restore button to the MDI application's menu bar. The application can **restore** the client window to its original (premaximized) size and position by sending the client window a **WM_MDIRESTORE** message.

An MDI application can arrange its child windows in either a cascade or tile format. When the child windows are cascaded, the windows appear in a stack. The window on the bottom of the stack occupies the upper left corner of the screen, and the remaining windows are offset vertically and horizontally so that the left border and title bar of each child window is visible. To arrange child windows in the **cascade format**, an MDI application sends the **WM_MDICASCADE** message. Typically, the application sends this message when the user clicks Cascade on the window menu.

When the child windows are tiled, the system displays each child window in its entirety - overlapping none of the windows. All of the windows are sized, as necessary, to fit within the client window. To arrange child windows in the **tile format**, an MDI application sends a **WM_MDITILE** message to the client window. Typically, the application sends this message when the user clicks Tile on the window menu.

An MDI application should provide a different icon for each type of child window it supports. The application specifies an icon when registering the child window class. The system automatically displays a child window's icon in the lower portion of the client window when the child window is minimized. An MDI application directs the system to arrange child window icons by sending a WM_MDIICONARRANGE message to the client window. Typically, the application sends this message when the user clicks Arrange Icons on the window menu.

To **arrange** child window icons, the application sends the **WM_MDIICONARRANGE** message to the client window.

$$Maximize \rightarrow WM\_MDIMAXIMIZE$$

$$Restore \rightarrow WM\_MDIRESTORE$$

$$Cascade \rightarrow WM\_MDICASCADE$$

$$Tile \rightarrow WM\_MDITILE$$

$$Icon\ arrange \rightarrow WM\_MDIICONARRANGE$$

➢ **Icon Title Windows**

Icon Title Windows

Because MDI child windows may be minimized, an MDI application must avoid manipulating icon title windows as if they were normal MDI child windows. Icon title windows appear when the application enumerates child windows of the MDI client window. Icon title windows differ from other child windows, however, in that they are owned by an MDI child window.

To determine whether a child window is an icon title window, use the **GetWindow** function with the **GW_OWNER** index. Non-title windows return NULL. Note that this test is insufficient for top-level windows, because menus and dialog boxes are owned windows.

➢ Child Window Data

Child Window Data

Because the number of child windows varies depending on how many documents the user opens, an MDI application must be able to associate data (for example, the name of the current file) with each child window. There are two ways to do this :

- Store **child window data** in the window structure.
- Use window properties.

6.ActiveX

❖ What is ACtiveX?

ActiveX

ActiveX is a software framework from Microsoft (MSFT) that allows applications to share functionality and data with one another through web browsers, regardless of what programming language they're written in. ActiveX add-ons allowed early web browsers to embed multimedia files or deliver software updates to users.

❖ Explain ActiveX control and activeX components

ActiveX controls

ActiveX controls are mostly talked about in reference to Internet Explorer, the default Web browser for the Windows operating system. Let's say you open a Web page with Internet Explorer that contains video clips encoded as Windows Media files (.wmv). Internet Explorer **comes pre-loaded with an ActiveX control** that allows for Windows Media files to be played directly in the Web page

ActiveX Components

There are several different types of ActiveX objects:

ActiveX **component**. An ActiveX object that can be used by other programs.

ActiveX **control**. A special type of ActiveX component that functions as a control window.

ActiveX **container**. A program that can accept **ActiveX controls** or document objects.

ActiveX **server**. A DLL or other executable program that provides the services of one or more

ActiveX **client**. A program that uses an ActiveX component.

ActiveX **document object**. A document that can be linked or embedded into an ActiveX container.

❖ What are the differences between ActiveX & ActiveX Components

Not finished yet

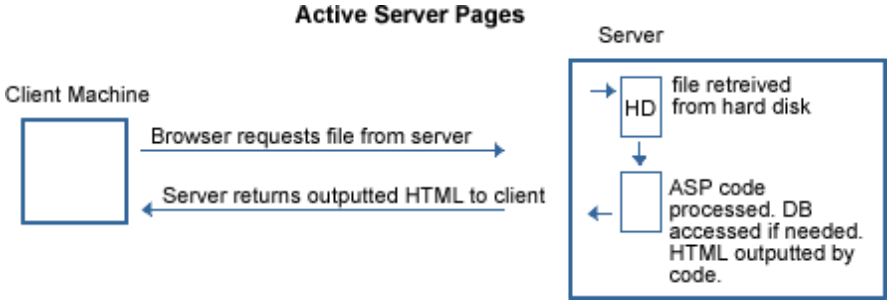
- Explain Active server page

Active Server Pages (also known as ASP or classic ASP) is Microsoft's first server-side script engine that enables dynamically-generated web pages. While the initial release was an add-on to the Internet Information Services (IIS) component of Windows NT 4.0, it was later incorporated into the Windows Server operating system.

ASP employs server-side scripting to dynamically **produce web pages** based on a specific request from the client. The result is a HTML webpage sent back to the client for display. VBScript is the default scripting language used for writing ASP, although other scripting languages can be used.

ASP was Microsoft' alternative to Common Gateway Interface (CGI) scripts and Java Server Pages (JSPs), both intended to allow clients to interact with server-side databases and enterprise services. ASP has gone through three major releases: ASP 1.0 in 1996 (included with IIS 3.0), ASP 2.0 in 1997 (IIS 4.0) and ASP 3.0 in 2000 (IIS 5.0). ASP 3.0 becomes part of IIS 6.0 on Windows Server 2003 and part of IIS 7.0 on Windows Server 2008.

ASP is now obsolete and replaced with ASP.NET. Though, ASP.NET is not strictly an enhanced version of ASP; the two technologies have completely different underlying implementations. ASP.NET is a compiled language and relies on the .NET Framework, while ASP is strictly an interpreted language. As with any older technology, you can certainly find ASP in production, but you'd be hard-pressed to make the case to use it for a new project.



Active Server Structure

- What is plug in?

A plug-in is an element of a software program that can be added to provide support for specific features or functionality.

Plug-ins are commonly used in Internet browsers but also can be utilized in numerous other types of applications.

In general, plug-ins are part of an array of software components known as add-ons. Programs may be changed by different kinds of add-ons in different ways.

In popular technologies, like Internet browsers and audio/video applications, the ability to utilize plug-ins makes products more versatile and allows transparent and convenient customization

according to the user's desired features. Plug-ins also can enable easier software upgrades or patches or additions by project collaborators. Plug-ins also can be a strategy for dealing with complex software licensing.

One plug-in example is the range of customizable options common with browsers like Mozilla Firefox. Users can download individual plug-ins for this free Web browser tool to promote different results on devices.

7.Server

- What is an Apache web server? Why are apache web servers needed?

Apache HTTP Server is a free and open-source web server that delivers web content through the internet. It is commonly referred to as Apache and after development, it quickly became the most popular HTTP client on the web. It's widely thought that Apache gets its name from its development history and process of improvement through applied patches and modules but that was corrected back in 2000. It was revealed that the name originated from the respect of the Native American tribe for its resiliency and durability.

Why Apache Web Servers?

Apache is considered open source software, which means the original source code is freely available for viewing and collaboration. Being open source has made Apache very popular with developers who have built and configured their own modules to apply specific functionality and improve on its core features. Apache has been around since 1995 and is responsible as a core technology that helped spur the initial growth of the internet in its infancy.

One of the pros of Apache is its ability to handle large amounts of traffic with minimal configuration. It scales with ease and with its modular functionality at its core, you can configure Apache to do what you want, how you want it. You can also remove unwanted modules to make Apache more lightweight and efficient.

Some of the most popular modules that can be added are SSL, Server Side Programming Support (PHP), and Load Balancing configs to handle large amounts of traffic. Apache can also be deployed on Linux, MacOS, and Windows. If you learn how to configure Apache on Linux, you can administer Apache on Windows and Mac. The only difference would be directory paths and installation processes.

- Pros and Cons of Apache web server.

Pros of Apache Server:

- Apache is open-source, and anyone can get it for free
- Customizable code can be adjusted to the needs
- Ability to add more features and modules to improve functions
- Highly reliable and excellent performance

- Apache is straightforward to install
- Immediate recording of changes
- Can be run on every operating system
- Actively maintained and upgraded by a community
- Highly flexible Web Server
- Impressive Documentation that is quite extensive and helpful

Cons of Apache Server

- ❖ Ability to modify the configuration offered an invitation to various threats when you meddled with code, insecure gates open.
- ❖ Again, the customization means new bugs and errors. Debugging means time and resources consumption
- ❖ Strict updating policy is necessary that needs to be done at regular intervals
- ❖ Recognizing and disabling unwanted services and modules
- ❖ Performance issues on extremely traffic-heavy websites.

- ❖ Explain the Apache web server architecture.

Apache Web Application Architecture

Apache is just one component that is needed in a web application stack to deliver web content. One of the most common web application stacks involves LAMP, or Linux, Apache, MySQL, and PHP.

Linux is the operating system that handles the operations of the application. Apache is the web server that processes requests and serves web assets and content via HTTP. MySQL is the database that stores all your information in an easily queried format. PHP is the programming language that works with apache to help create dynamic web content.

While actual statistics may vary, it's fair to say a large portion of web applications run on some form of the LAMP stack because it is easy to build and also free to use. For the most part, web applications tend to generally have similar architecture and structure even though they serve many different functions and purposes. Most web applications also benefit from Firewalls, Load Balancers, Web Servers, Content Delivery Networks, and Database Servers.

Firewalls help protect the web application from both external threats and internal vulnerabilities depending on where the firewalls are configured. Load Balancers help distribute traffic across the web servers which handle the HTTP(S) requests (this is where Apache comes in) and application servers (servers that handle the functionality and workload of the web app.) We also have Database Servers, which handle asset storage and backups. Depending on your infrastructure, your database and application can both live on the same server although it's recommended to keep those separate.

- ❖ List the features of apache.

Features of Apache Web Server

- Handling of static files
- Loadable dynamic modules

- Auto-indexing
- .htaccess
- Compatible with IPv6
- Supports HTTP/2
- FTP connections
- Gzip compression and decompression
- Bandwidth throttling
- Perl, PHP, Lua scripts
- Load balancing
- Session tracking
- URL rewriting
- Geolocation based on IP address

❖ How does Apache web server work?

Apache functions as a way to communicate over networks from client to server using the TCP/IP protocol. Apache can be used for a wide variety of protocols, but the most common is HTTP/S. HTTP/S or HyperText Transfer Protocol (S stands for Secure) is one of the main protocols on the web, and the one protocol Apache is most known for.

HTTP/S is used to define how messages are formatted and transmitted across the web, with instructions for browsers and servers on how to respond to various requests and commands. Hypertext Transfer Protocol Secure is usually through port 443 with the unsecured protocol being through port 80.

The Apache server is configured via config files in which modules are used to control its behavior. By default, Apache listens to the IP addresses configured in its config files that are being requested. This is where one of Apaches' many strengths come into play.

With the Listen directive, Apache can accept and route specific traffic to certain ports and domains based on specific address-port combination requests. By default, Listen runs on port 80 but Apache can be bound to different ports for different domains, allowing for many different websites and domains to be hosted and a single server. You can have domain1.com listening on port 80, domain2.com on port 8080 and domain3.com on port 443 using HTTPS all on Apache.

Once a message reaches its destination or recipient, it sends a notice, or ACK message, basically giving acknowledgment to the original sender that their data has successfully arrived. If there's an error in receiving data, or some packets were lost in transit, the destination host or client sends a Not Acknowledged, or NAK message, to inform the sender that the data needs to be retransmitted.

❖ **What is a server ? Explain the types of servers**.

Server

A server is a computer program or device that provides a service to another computer program and its user, also known as the client. In a data center, the physical computer that a server program runs on is also frequently referred to as a server. That machine might be a dedicated server or it might be used for other purposes.

In the client/server programming model, a server program awaits and fulfills requests from client programs, which might be running in the same, or other computers. A given application in a computer might function as a client with requests for services from other programs and as a server of requests from other programs.

Types of Server

Servers are often categorized in terms of their purpose. A few examples of the types of servers available are as follows:

Web server: a computer program that serves requested HTML pages or files. In this case, a web browser acts as the client.

Application server: a program in a computer in a distributed network that provides the business logic for an application program.

Proxy server: software that acts as an intermediary between an endpoint device, such as a computer, and another server from which a user or client is requesting a service.

Mail server: an application that receives incoming emails from local users -- people within the same domain -- and remote senders and forwards outgoing emails for delivery.

Virtual server: a program running on a shared server that is configured in such a way that it seems to each user that they have complete control of a server.

File server: a computer responsible for the central storage and management of data files so that other computers on the same network can access them.

Policy server: a security component of a policy-based network that provides authorization services and facilitates tracking and control of files.

Database server: this server is responsible for hosting one or more databases. Client applications perform database queries that retrieve data from or write data to the database that is hosted on the server.

Print server: this server provides users with access to one or more network-attached printers -- or print devices as some server vendors call them. The print server acts as a queue for the print jobs that users submit. Some print servers can prioritize the jobs in the print queue based on the job type or on who submitted the print job.

❖ How do servers connect to the internet?

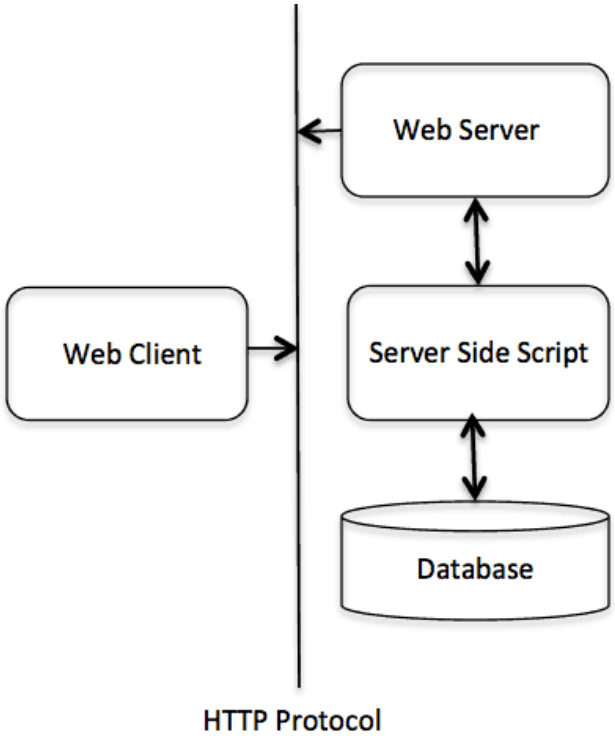
In the case of an Internet server, the device is connected to the Web, so that any computer with a Web connection can access the files stored on the server. Servers store and process data just like a computer, and are connected to the Internet through wired or wireless connections.

So basically, a server is directly connected to the Internet via routers that transmit all the information between the Internet and the server.

❖ What is HTTP? Draw the basic architecture of HTTP.

Hypertext transfer protocol or HTTP is a fundamental protocol used on the Internet in order to control data transfer to and from a hosting server, in communication with a web browser.

Architecture of HTTP protocol



Architecture of HTTP protocol

The HTTP protocol is a request/response protocol based on the client/server based architecture where web browsers, robots and search engines, etc. act like HTTP clients, and the Web server acts as a server.

Client

The **HTTP client** sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a TCP/IP connection.

Server

The **HTTP server** responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta information, and possible entity-body content.

- Explain the request method of HTTP.

GET: The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.

HEAD: Same as GET, but it transfers the status line and the header section only.

POST: A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.

PUT: Replaces all the current representations of the target resource with the uploaded content.

DELETE: Removes all the current representations of the target resource given by URI.

CONNECT: Establishes a tunnel to the server identified by a given URI.

OPTIONS: Describe the communication options for the target resource.

TRACE: Performs a message loop back test along with the path to the target resource.

- **Explain how OLE automation works?**

In Microsoft Windows applications programming, OLE Automation (later renamed to simply Automation) is an inter-process communication mechanism by Microsoft. It is based on a subset of Component Object Model (COM) that was created intended for use by scripting languages - originally Visual Basic - but now is used by several languages on Windows. All automation objects are required to implement the Dispatch interface. It provides an infrastructure whereby applications called automation controllers can access and manipulate (i.e. set properties of or call methods on) shared automation objects that are exported by other applications. It supersedes Dynamic Data Exchange (DDE), an older mechanism for applications to control one another. As with DDE, in OLE Automation **the automation controller** is the "**client**" and the application exporting **the automation objects** is the "**server**".

8.API

- What is API and API testing?

Application Programming Interface (API)

Application programming interfaces, or APIs, simplify software development and innovation by enabling applications to exchange data and functionality easily and securely.

An application programming interface, or API, enables companies to open up their applications' data and functionality to external third-party developers, business partners, and internal departments within their companies. This allows services and products to communicate with each other and leverage each other's data and functionality through a documented interface. Developers don't need to know how an API is implemented; they simply use the interface to communicate with other products and services. API use has surged over the past decade, to the degree that many of the most popular web applications today would not be possible without APIs.

API testing

- Injection consists of sending an API malicious commands through a user input field, whether a text input, file upload, or other means. This attack vector allows malicious actors to send code or other executable commands to the API's interpreter, which can be used to bypass security, change permissions, access information, damage, or disable the API. Common injections include cross-site scripting (XSS), SQL injections, and template injections.
- A malicious application can inject input to the user interface to mimic user interaction through the abuse of Android's accessibility APIs.
- Input Injection can be achieved using any of the following methods:
- Mimicking user clicks on the screen, for example to steal money from a user's PayPal account.
- Injecting global actions, such as GLOBAL_ACTION_BACK (programatically mimicking a physical back button press), to trigger actions on behalf of the user.
- Inserting input into text fields on behalf of the user. This method is used legitimately to auto-fill text fields by applications such as password managers.

- How does an API work?

How an API works

An API is a set of defined rules that explain how computers or applications communicate with one another. APIs sit between an application and the web server, acting as an intermediary layer that processes data transfer between systems.

Here's how an API works:

1. A client application initiates an API call to retrieve information—also known as a request. This request is processed from an application to the web server via the API's Uniform

Resource Identifier (URI) and includes a request verb, headers, and sometimes, a request body.

2. After receiving a valid request, the API makes a call to the external program or web server.
3. The server sends a response to the API with the requested information.
4. The API transfers the data to the initial requesting application.

While the data transfer will differ depending on the web service being used, this process of requests and response all happens through an API. Whereas a user interface is designed for use by humans, APIs are designed for use by a computer or application.

APIs offer security by design because their position as middleman facilitates the abstraction of functionality between two systems—the API endpoint decouples the consuming application from the infrastructure providing the service. API calls usually include authorization credentials to reduce the risk of attacks on the server, and an API gateway can limit access to minimize security threats. Also, during the exchange, HTTP headers, cookies, or query string parameters provide additional security layers to the data.

For example, consider an API offered by a payment processing service. Customers can enter their card details on the frontend of an application for an ecommerce store. The payment processor doesn't require access to the user's bank account; the API creates a unique token for this transaction and includes it in the API call to the server. This ensures a higher level of security against potential hacking threats.

- Why do we need APIs?

Why we need APIs

Whether you're managing existing tools or designing new ones, you can use an application programming interface to simplify the process. Some of the main benefits of APIs include the following:

- **Improved collaboration**: The average enterprise uses almost 1,200 cloud applications (link resides outside of IBM), many of which are disconnected. APIs enable integration so that these platforms and apps can seamlessly communicate with one another. Through this integration, companies can automate workflows and improve workplace collaboration. Without APIs, many enterprises would lack connectivity and would suffer from informational silos that compromise productivity and performance.

- **Easier innovation**: APIs offer flexibility, allowing companies to make connections with new business partners, offer new services to their existing market, and, ultimately, access new markets that can generate massive returns and drive digital transformation. For example, the company Stripe began as an API with just seven lines of code. The company has since partnered with many of the biggest enterprises in the world, diversified to offer loans and corporate cards, and was recently valued at USD 36 billion (link resides outside of IBM).

- **Data monetization**: Many companies choose to offer APIs for free, at least initially, so that they can build an audience of developers around their brand and forge relationships with potential business partners. However, if the API grants access to valuable digital assets, you can monetize it by selling access (this is referred to as the API economy). When AccuWeather (link resides outside of IBM) launched its self-service developer portal to sell a wide range of API packages, it took just 10 months to attract 24,000 developers, selling 11,000 API keys and building a thriving community in the process.

- **Added security**: As noted above, APIs create an added layer of protection between your data and a server. Developers can further strengthen API security by using tokens, signatures, and Transport Layer Security (TLS) encryption; by implementing API gateways to manage and authenticate traffic; and by practicing effective API management.

- Give some common examples on API.

Common API examples

Because APIs allow companies to open up access to their resources while maintaining security and control, they have become a valuable aspect of modern business. Here are some popular examples of application programming interfaces you may encounter:

- Universal logins: A popular API example is the function that enables people to log in to websites by using their Facebook, Twitter, or Google profile login details. This convenient feature allows any website to leverage an API from one of the more popular services to quickly authenticate the user, saving them the time and hassle of setting up a new profile for every website service or new membership.
- Third-party payment processing: For example, the now-ubiquitous "Pay with PayPal" function you see on ecommerce websites works through an API. This allows people to pay for products online without exposing any sensitive data or granting access to unauthorized individuals.
- Travel booking comparisons: Travel booking sites aggregate thousands of flights, showcasing the cheapest options for every date and destination. This service is made possible through APIs that provide application users with access to the latest information about availability from hotels and airlines. With an autonomous exchange of data and requests, APIs dramatically reduce the time and effort involved in checking for available flights or accommodation.
- Google Maps: One of the most common examples of a good API is the Google Maps service. In addition to the core APIs that display static or interactive maps, the app utilizes other APIs and features to provide users with directions or points of interest. Through geolocation and multiple data layers, you can communicate with the Maps API when plotting travel routes or tracking items on the move, such as a delivery vehicle.
- Twitter: Each Tweet contains descriptive core attributes, including an author, a unique ID, a message, a timestamp when it was posted, and geolocation metadata. Twitter makes

public Tweets and replies available to developers and allows developers to post Tweets via the company's API.

- **Explain different types of APIs and API protocols**.

Types of APIs

Nowadays, most application programming interfaces(APIs) are web APIs that expose an application's data and functionality over the internet. Here are the **four** main types of web API:

- **Open APIs** are open source application programming interfaces you can access with (can be accessed with) the HTTP protocol. Also known as public APIs, they have defined API endpoints and request and response formats.
- **Partner APIs** are application programming interfaces exposed to or by strategic business partners. Typically, developers can access these APIs in self-service mode through a public API developer portal. Still, they will need to complete an onboarding process and get login credentials to access partner APIs.
- **Internal APIs** are application programming interfaces that remain hidden from external users. These private APIs aren't available for users outside of the company and are instead intended to improve productivity and communication across different internal development teams.
- **Composite APIs** combine multiple data or service APIs. These services allow developers to access several endpoints in a single call. Composite APIs are useful in microservices architecture where performing a single task may require information from several sources.

Types of API protocols

As the use of web APIs has increased, certain protocols have been developed to provide users with a set of defined rules that specifies the accepted data types and commands. In effect, these API protocols facilitate standardized information exchange:

- **SOAP (Simple Object Access Protocol)** is an API protocol built with XML, enabling users to send and receive data through SMTP and HTTP. With SOAP APIs, it is easier to share information between apps or software components that are running in different environments or written in different languages.

- **XML-RPC** is a protocol that relies on a specific format of XML to transfer data, whereas SOAP uses a proprietary XML format. XML-RPC is older than SOAP, but much simpler, and relatively lightweight in that it uses minimum bandwidth.

- **JSON-RPC** is a protocol similar to XML-RPC, as they are both remote procedure calls (RPCs), but this one uses JSON instead of XML format to transfer data. Both protocols are simple. While calls may contain multiple parameters, they only expect one result.

- **REST** (**Representational State Transfer**) is a set of web API architecture principles, which means there are no official standards (unlike those with a protocol). To be a REST API (also known as a RESTful API), the interface must adhere to certain architectural constraints. It's possible to build RESTful APIs with SOAP protocols, but the two standards are usually viewed as competing specifications.

- What is input injection in API?

Not finished yet

- Explain rest with examples.

REST (**Representational State Transfer**) is a set of web API architecture principles, which means there are no official standards (unlike those with a protocol). To be a REST API (also known as a RESTful API), the interface must adhere to certain architectural constraints. It's possible to build RESTful APIs with SOAP protocols, but the two standards are usually viewed as competing specifications.

9.WEB

- What is a web application?

Not finished yet

- **Explain web based application development.**

A web application is an interactive program that runs on a web server and is accessed through a web browser. A web app is built so that the user interface provides data back to the development team that designed it. This data offers insights into customer interests, usage, and preferences that can prove invaluable to product and marketing strategies. The data can also inform optimization and other client-centered aspects of the mobile app or desktop applications

How does a web application work?
All you need to access a web app is an internet connection. You use a web browser like Safari, Mozilla Firefox or Google Chrome to connect to your app. There are three elements the web application requires to function: **a web server** to handle requests from the client, **an application server** to execute the tasks requested and **a database** to store the information.

Developers code web applications in two types of languages. A web application generally uses a combination of s**erver-side script** and c**lient-side scrip**t to function. The server-side script deals with s**toring and retrieving** the information and requires languages like Python or Java. Developers program server-side to create scripts the web app will use. The client-side script requires languages like JavaScript, Cascading Style Sheets (CSS) and HTML5. These languages rely on the browser to execute the program. They are browser-supported languages. The client-side script deals with **the presentation** of the information to the user.

Most web apps have short development cycles and can be created by small teams. Some of the apps require server-side processing. They are called "dynamic." Some don't need processing at the server-side and are static.
Here is how a web application works:

1. The user creates a request to the web server over the Internet through the application's user interface.
2. The web server sends this request to the web application server.
3. The web application server executes the requested task, then generates the results of the required data.
4. The web application server sends those results back to the web server (requested information or processed data).
5. The web server carries the requested information to the client (tablet, mobile device or desktop).
6. The requested information appears on the user's display.

- Explain how database connected to web application

Not finished yet

- What are the differences between traditional client-server applications & web based applications?

Tradition Client-Server Applications	Web-Based Applications
Platform-dependent	Platform-independent
Client is natively compiled and therefore has fast execution speed. fast **execution** speed.	Client is an interpreter (eg, HTML, Java, Java Script) and therefore is slower. Slow execution speed
Installation necessary.	No need for installation.
Complex client, high maintenance cost.	Simple client, minimum maintenance
high maintenance **cost**	minimum maintenance
New, unfamiliar **interface** for users	One common, familiar interface across applications
Rich, custom GUI constructs possible	Limited set of GUI constructs, custom ones add to download time
Difficult to **integrate** with existing applications	Easy to integrate
Difficult to add **multimedia**.	Easy to add multimedia

Persistent connection to database	Non-persistent connection.

10.Database

- What is database programming? Explain it.
- What is the data object?
- Explain how OLE automation works?
- Describe the layers of web database architecture.

Web database architectures

Components of a database application Web database applications may be created using various approaches. However, there are a number of components that will form essential building blocks for such applications. In other words, a Web database application should comprise the following four layers (i.e. components):

1. Browser layer
2. Application logic layer
3. Database connection layer
4. Database layer

Browser layer

The browser is the client of a Web database application, and it has two major functions. First, it handles the layout and display of HTML documents. Second, it executes the client-side extension functionality such as Java, JavaScript, and ActiveX (a method to extend a browser's capabilities). The three most popular browsers at the present are Mozilla Firefox (Firefox for short), Google Chrome and Microsoft Internet Explorer (IE). All three browsers are graphical browsers. During the early days of the Web, a text-based browser, called Lynx, was popular. As loading graphics over the Internet can be a slow and time-consuming process, database performance may be affected. If an application requires a speedy client and does not need to display graphics, then the use of Lynx may be considered. All browsers implement the HTML standard. Browsers are also responsible for providing forms for the collection of user input, packaging the input, and sending it to the appropriate server for processing. For example, input can include registration for site access, guest books and requests for information. HTML, Java, JavaScript or ActiveX (for IE) may be used to implement forms.

Application logic layer

The application logic layer is the part of a Web database application with which a developer will spend the most time. It is responsible for:

- Collecting data for a query (e.g. a SQL statement).
- Preparing and sending the query to the database via the database connection layer.
- Retrieving the results from the connection layer.
- Formatting the data for display.

- Most of the application's business rules and functionality will reside in this layer. Whereas the browser client displays data as well as forms for user input, the application logic component compiles the data to be displayed and processes user input as required. In other words, the application logic generates HTML that the browser renders. Also it receives, processes and stores user input that the browser sends. Depending on the implementation methods used for the database application, the application logic layer may have different security responsibilities. If the application uses HTML for the front end, the browser and server can handle data encryption (i.e. a security measure to ensure that data will not be able to be intercepted by unauthorized parties). If the application is a Java applet and uses Java for the front end, then it itself must be responsible for adopting transmission encryption.

Database connection layer :

This is the component which actually links a database to the Web server. Because manual Web database programming can be a daunting task, many current Web database building tools offer database connectivity solutions, and they are used to simplify the connection process. The database connection layer provides a link between the application logic layer and the DBMS. Connection solutions come in many forms, such as DBMS net protocols, API (Application Programming Interface)or class libraries, and programs that are themselves database clients. Some of these solutions resulted in tools being specifically designed for developing Web database applications. In Oracle, for example, there are native API libraries for connection and a number of tools, such as Web Publishing Assistant, for developing Oracle applications on the Web. The connection layer within a Web database application must accomplish a number of goals. It has to provide access to the underlying database, and also needs to be easy to use, efficient, flexible, robust, reliable and secure. Different tools and methods fulfill these goals to different extents.

Database layer:

This is the place where the underlying database resides within the Web database application. As we have already learned, the database is responsible for storing, retrieving and updating data based on user requirements, and the DBMS can provide efficiency and security measures. In many cases, when developing a Web database application, the underlying database has already been in existence. A major task, therefore, is to link the database to the Web (the connection layer) and to develop the application logic layer.

❖ What is the Database gateway?

A **database gateway** allows clients using one communication protocol to connect with data servers that use a different protocol

Database Gateway is a database connection service for remote access to on-premises databases or databases that are hosted on third-party clouds. Database Gateway allows you to connect an on-premises database or a database that is hosted on a third-party cloud to Alibaba Cloud in a secure manner at low costs. You can also integrate Database Gateway with other Alibaba Cloud services, such as Data Transmission Service (DTS), Database Backup (DBS),

and Data Management (DMS)

- **Write the steps for a CGI program to execute successfully**.

The following steps need to be taken in order for a CGI program to execute successfully:

- The user (Web client) **calls** the CGI program by clicking on a link or by pressing a button. The program can also be invoked when the browser loads an HTML document (hence being able to create a dynamic Web page).
- The browser **contacts** the Web server, asking for permission to run the CGI program.
- The server **checks** the configuration and access files to ensure that the program exists and the client has access authorization to the program.
- The server **prepares** the environment variables and launches the program.
- The program **executes and reads** the environment variables and STDIN.
- The program **sends** the appropriate MIME headers to STDOUT, followed by the remainder of the output, and terminates.
- The s**erver sends** the data in STDOUT (i.e. the output from the program's execution) to the browser and closes the connection.
- The browser displays the information received from the server.

- How do CGI programs connect to the server?

Not finished yet

- Describe the advantages & disadvantages of CGI.

11.Multithreading

- **Multithreading**

Multithreading is a technique that allows for concurrent (simultaneous) execution of two or more parts of a program for maximum utilization of a CPU. As a really basic example, multithreading allows you to write code in one program and listen to music in another. Programs are made up of processes and threads. You can think of it like this:

- A program is an executable file like chrome.exe
- A process is an executing instance of a program. When you double click on the Google Chrome icon on your computer, you start a process which will run the Google Chrome program.
- Thread is the smallest executable unit of a process. A process can have multiple threads with one main thread. In the example, a single thread could be displaying the current tab you're in, and a different thread could be another tab.

- **What is multithreading**? Give example, why use it?
- Define the basic concept of multithreading.

Operating systems today can run multiple programs at the same time. For example, you're reading this article in your browser (a program) but you can also listen to music on your media player (another program).

Processes are what actually execute the program. Each process is able to run concurrent subtasks called threads.

Threads are sub-tasks of processes and if synchronized correctly can give the illusion that your application is performing everything at once. Without threads you would have to write one program per task, run them as processes and synchronize them through the operating system.

- **What are the issues involved with multiple threads**? How to avoid issues with multiple threads?

Issues Involved with Multiple Threads

- **Deadlock**

  Deadlocks happen when two or more threads aren't able to make any progress because the resource required by the first thread is held by the second and the resource required by the second thread is held by the first.

- **Race conditions**

  Critical section is any piece of code that has the possibility of being executed concurrently by more than one thread of the application and exposes any shared data or resources used by the application for access.

Race conditions happen when threads run through critical sections without thread synchronization. The threads "race" through the critical section to write or read shared resources and depending on the order in which threads finish the "race", the program output changes.

In a race condition, threads access shared resources or program variables that might be worked on by other threads at the same time causing the application data to be inconsistent.

How to avoid issues with multiple threads

How to avoid deadlocks?

- **Avoid Nested Locks**: This is the main reason for deadlock. Deadlock mainly happens when we give locks to multiple threads. Avoid giving locks to multiple threads if you already have given to one.
- **Avoid Unnecessary Locks**: You should lock only those members which are required. Having unnecessary locks can lead to a deadlock. As a best practice, try to reduce the need to lock things as much as you can.

How to avoid race conditions?

- Race conditions occur within the critical section of your code. These can be avoided with **proper thread synchronization** within critical sections by using techniques like locks, atomic variables, and message passing.

How to avoid starvation?

- The best way to avoid starvation is to use a lock such as ReentrantLock or a mutex. This introduces a “fair” lock which favors granting access to the thread that has been waiting longest. If you wanted to have multiple threads run at once while preventing starvation, you can use a semaphore.

How to avoid livelocks?

- Livelocks can be avoided by making use of ReentrantLock as a way to determine which thread has been waiting longer so that you can assign it a lock. As a best practice, don’t block locks; if a thread can’t acquire a lock, it should release previously acquired locks to try again later.

❖ What is concurrency? What are the problems in concurrency?

Concurrency

Concurrency is the execution of the **multiple instruction sequences** at the same time. It happens in the operating system when there are several process threads running in parallel. The running process threads always communicate with each other through shared memory or message passing. Concurrency results in sharing of resources resulting in problems like deadlocks and resource starvation.

It helps in techniques like coordinating execution of processes, memory allocation and execution scheduling for maximizing throughput.

❖ **Explain the Advantages & disadvantages of concurrency.**

Advantages of Concurrency :

- **Running of multiple application**s –
  It enables to run multiple applications at the same time.

- **Better resource utilization** –
  It enables that the resources that are unused by one application can be used for other applications.

- **Better average response time** –
  Without concurrency, each application has to be run to completion before the next one can be run.

- **Better performance** –
  It enables better performance by the operating system. When one application uses only the processor and another application uses only the disk drive then the time to

run both applications concurrently to completion will be shorter than the time to run each application consecutively.

Drawbacks of Concurrency :

- It is required to protect multiple applications from one another.
- It is required to coordinate multiple applications through additional mechanisms.
- Additional performance overheads and complexities in operating systems are required for switching among applications.
- Sometimes running too many applications concurrently leads to severely degraded performance.

12.XML

❖ **What is XML** ? Explain it with examples. How does XML work?

XML stands for Extensible Markup Language. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

There are three important characteristics of XML that make it useful in a variety of systems and solutions −

- XML is extensible − XML allows you to create your own self-descriptive tags, or language, that suits your application.
- XML carries the data, does not present it − XML allows you to store the data irrespective of how it will be presented.
- XML is a public standard − XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

Examples

XML is very widely used today. It is the basis of a great many standards such as the Universal Business Language (UBL); of Universal Plug and Play (UPnP) used for home electronics; word processing formats such as ODF and OOXML; graphics formats such as SVG; it is used for communication with XMLRPC and Web Services, it is supported directly by computer programming languages and databases, from giant servers all the way down to mobile telephones.

If you double-click an icon on your computer desktop (the icon may well have been drawn with SVG), chances are that an XML message is sent from one component of the desktop to another. If you take your car to be repaired, the engine's computer sends XML to the mechanic's diagnostic systems. It is the age of XML: it is everywhere.

How does XML work?

XML works by providing a predictable data format. XML is strict on formatting; if the formatting is off, programs that process or display the encoded data will return an error.

For an XML document to be considered well-formed -- that is, conforming to XML syntax and able to be read and understood by an XML parser -- it must be valid XML code. All XML documents consist of elements; an element acts as a container for data. The beginning and end of an element are identified by opening and closing tags, with other elements or plain data within.

XML works by providing properly formatted data that can be reliably processed by programs designed to handle XML inputs. For example, technical documentation may include a <warning> element similar to that shown in the following snippet of XML code:

```
<warning>
    <para>
        <emphasis type="bold">May cause serious injury</emphasis>
        Exercise extreme caution as this procedure could result in serious injury or death if precautions are not taken.
    </para>
</warning>
```

In this example, this data is interpreted and displayed in different ways, depending on the form factor of the technical documentation. On a webpage, this element could be displayed in the following way:

WARNING: Exercise extreme caution as this procedure could result in serious injury or death if precautions are not taken.

The same XML code is rendered differently on an appliance user interface (UI) or in print. This element could be interpreted to display the text tagged as emphasis differently, such as having it appear in red and with flashing highlights. In printed form, the content might be provided in a different font and format.

XML documents do not define presentation, and there are no default XML tags. Most XML applications use predefined sets of tags that differ, depending on the XML format. Most users rely on predefined XML formats to compose their documents, but users may also define additional XML elements as needed.

- ❖ Write a short note on XML elements and entities.

XML elements

The logical structure of an XML file requires that all data in the file be encapsulated within an XML element called the root element or document element. This element identifies the type of data contained in the file; in the example above, the root element is <library>.

The root element contains other elements that define the different parts of the XML document; in the example above, the root element contains <book> elements, which, in turn, consist of the two elements <title> and <author>.

All XML elements must be properly terminated for an XML file to be considered well-formed. This means that a tag must be properly terminated with an opening and closing tag, like this paragraph element that would be a part of a document:
<para>This is an example of an XML tag for a paragraph.</para>
A tag can also be empty, in which case it is terminated with a forward slash. In this example, an empty self-terminating paragraph tag is used to insert an extra space in a document:
<para />
XML enables users to define their own additional elements if needed. In the preceding example, an XML author might define new elements for publisher, date of publication, International Standard Book Number and any other relevant data. The elements can also be defined to enforce rules regarding the contents of the elements.

XML entities
XML elements can also contain predefined entities, which are used for special reserved XML characters. Custom entities can be defined to insert a predefined string of characters for inclusion in an XML file.
The five standard predefined XML entities are the following:

1. < -- The less than symbol (<), also known as the open angle bracket, is normally used in XML to indicate the start of an XML tag. This entity is used when the open angle bracket is part of the content of the XML file.
2. > -- The greater than symbol (>), also known as the close angle bracket, is normally used in XML to indicate the end of an XML tag. This entity is used when the close angle bracket is part of the content of the XML file.
3. & -- The ASCII ampersand symbol (&) is reserved in XML for indicating the start of an XML entity. This entity is used when an ampersand occurs within an XML element.
4. " -- The ASCII double quote character (") is used in XML element tags to identify optional attribute values of the element. For example, an <emphasis> tag might include options for emphasizing some text, such as bold, italic or underline. This entity is used when a double quote character appears in the contents of an XML element.
5. ' -- The ASCII single quote character ('), also known as an apostrophe, is used in XML element tags to identify option attributes of the element. For example, an <emphasis> tag might include options for emphasizing some text, such as bold, italic or underline. This entity is used when a single quote or apostrophe appears in the contents of an XML element.

XML entities take the form of &name; where the entity name begins with the ampersand symbol and ends with a semicolon. Custom entities can be single characters or complex XML elements. For example, boilerplate language for technical documentation or legal contracts can be reduced to a single entity. However, when using entities, the XML author must ensure that inserting the entity into an XML file will produce well-formed XML data.

- ❖ Advantages and disadvantages of XML

Advantages of XML

1. XML is platform independent and programming language independent, thus it can be used on any system and supports the technology change when that happens.
2. XML supports Unicode. Unicode is an international encoding standard for use with different languages and scripts, by which each letter, digit, or symbol is assigned a unique numeric value that applies across different platforms and programs. This feature allows XML to transmit any information written in any human language.
3. The data stored and transported using XML can be changed at any point of time without affecting the data presentation. Generally other markup language such as HTML is used for data presentation, HTML gets the data from XML and display it on the GUI (graphical user interface), once data is updated in XML, it does reflect in HTML without making any change in HTML GUI.
4. XML allows validation using DTD and Schema. This validation ensures that the XML document is free from any syntax error.
5. XML simplifies data sharing between various systems because of its platform independent nature. XML data doesn't require any conversion when transferred between different systems.
Disadvantages of XML
1. XML syntax is verbose and redundant compared to other text-based data transmission formats such as JSON.
2. The redundancy in syntax of XML causes higher storage and transportation cost when the volume of data is large.
3. XML document is less readable compared to other text-based data transmission formats such as JSON.
4. XML doesn't support array.
5. XML file sizes are usually very large due to its verbose nature, it is totally dependent on who is writing it.

- **What are the differences between XML and HTML?**

XML	HTML
XML stands for extensible Markup Language.	HTML stands for Hyper Text Markup Language.
XML is **dynamic** in nature.	HTML is **static** in nature.
XML provides a framework to define markup languages.	HTML is a markup language.
XML does not allow **errors**.	HTML can ignore small errors.
XML is **Case** sensitive.	HTML is not Case sensitive.
XML tags are user **defined** tags.	HTML tags are **predefined** tags.

XML tags are extensible.	There are a limited number of tags in HTML.
White space can be preserved in XML.	HTML does not preserve white spaces.
XML tags are used for describing the data not for displaying.	HTML tags are used for displaying the data.
In XML, **closing** tags are necessary.	In HTML, closing tags are not necessary.
XML is used to **store** data.	HTML is used to **display** the data.
XML carries the data to and from the database.	HTML does not carry data, it just displays it.
IN XML , the objects are expressed by conventions using attributes.	HTML offers native object support.
XML document size is relatively large as the approach of formatting and the codes are both lengthy.	HTML document size is relatively small.
DOM(Document Object Model) is required for parsing JavaScript codes and mapping of text.	Additional application is not required for parsing of JavaScript code into the HTML document.

The main differences from HTML are:

1. All elements must be closed or marked as empty.
2. Empty elements can be closed as normal, <happiness></happiness> or you can use a special short-form, <happiness /> instead.
3. In HTML, you only need to quote an attribute value under certain circumstances (it contains a space, or a character not allowed in a name), but the rules are hard to remember. In XML, attribute values must always be quoted:
   <happiness type="joy" />
4. In HTML there is a built-in set of element names (along with their attributes). In XML, there are no built-in names (although names starting with xml have special meanings).
5. In HTML, there is a list of some built-in character names like é for é but XML does not have this. In XML, there are only five built-in character entities: <, >, &, " and ' for <, >, &, " and ' respectively. You can define your own entities in a Document Type Definition, or you can use any Unicode character (see next item).
6. In HTML, there are also numeric character references, such as & for &. You can refer to any Unicode character, but the number is decimal, whereas in the Unicode tables the number is usually in hexadecimal. XML also allows hexadecimal references: & for example.

❖ **What is SOAP**?Describe it.

SOAP

SOAP (formerly an acronym for Simple Object Access Protocol) is a messaging protocol specification for exchanging structured information in the implementation of web services in computer networks. It uses XML Information Set for its message format, and relies on application layer protocols, most often Hypertext Transfer Protocol (HTTP), although some legacy systems communicate over Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

SOAP allows developers to invoke processes running on disparate operating systems (such as Windows, macOS, and Linux) to authenticate, authorize, and communicate using Extensible Markup Language (XML). Since Web protocols like HTTP are installed and running on all operating systems, SOAP allows clients to invoke web services and receive responses independent of language and platforms.

SOAP (Simple Object Access Protocol) is an API protocol built with XML, enabling users to send and receive data through SMTP and HTTP. With SOAP APIs, it is easier to share information between apps or software components that are running in different environments or written in different languages.

SOAP provides the Messaging Protocol layer of a web services protocol stack for web services. It is an XML-based protocol consisting of three parts:

- an envelope, which defines the message structure and how to process it
- a set of encoding rules for expressing instances of application-defined datatypes
- a convention for representing procedure calls and responses

SOAP has three major characteristics:

1. **extensibility** (security and WS-Addressing are among the extensions under development)
2. **neutrality** (SOAP can operate over any protocol such as HTTP, SMTP, TCP, UDP)
3. **independence** (SOAP allows for any programming model)

As an example of what SOAP procedures can do, an application can send a SOAP request to a server that has web services enabled—such as a real-estate price database—with the parameters for a search. The server then returns a SOAP response (an XML-formatted document with the resulting data), e.g., prices, location, features. Since the generated data comes in a standardized machine-parsable format, the requesting application can then integrate it directly.

The SOAP architecture consists of several layers of specifications for:

- message format
- Message Exchange Patterns (MEP)
- underlying transport protocol bindings

- message processing models
- protocol extensibility

SOAP evolved as a successor of XML-RPC, though it borrows its transport and interaction neutrality from Web Service Addressing and the envelope/header/body from elsewhere (probably from WDDX).

❖ Explain the structure of a SOAP message with example

The structure of a SOAP message

A SOAP message is encoded as an XML document, consisting of an <Envelope> element, which contains an **optional <Header> element**, and **a mandatory <Body> element**. **The <Fault> element**, contained in <Body>, is used for reporting errors.

The SOAP envelope

<Envelope> is the **root element** in every SOAP message, and contains two child elements, an optional <Header> element, and a mandatory <Body> element.

The SOAP header

<Header> is an **optional subelement** of the SOAP envelope, and is used to pass application-related information that is to be processed by SOAP nodes along the message path;

see The SOAP header.

The SOAP body

<Body> is a **mandatory subelement** of the SOAP envelope, which contains information intended for the ultimate recipient of the message;

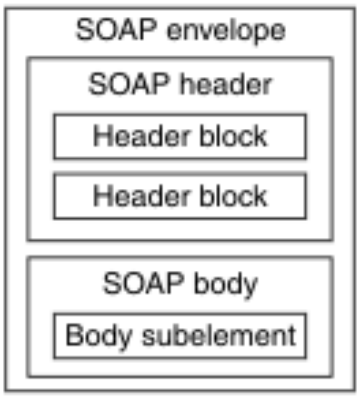
see The SOAP body.

The SOAP fault

<Fault> is a **subelement** of the SOAP body, which is used for reporting errors;

see The SOAP fault.

XML elements in <Header> and <Body> are defined by the applications that make use of them, although the SOAP specification imposes some constraints on their structure. The following diagram shows the structure of a SOAP message.



Skeleton SOAP Message

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

        <soap:Header>
                …
        </soap:Header>

        <soap:Body>
                …
                        <soap:Fault>
                                …
                        </soap:Fault>
        </soap:Body>

</soap:Envelope>
```

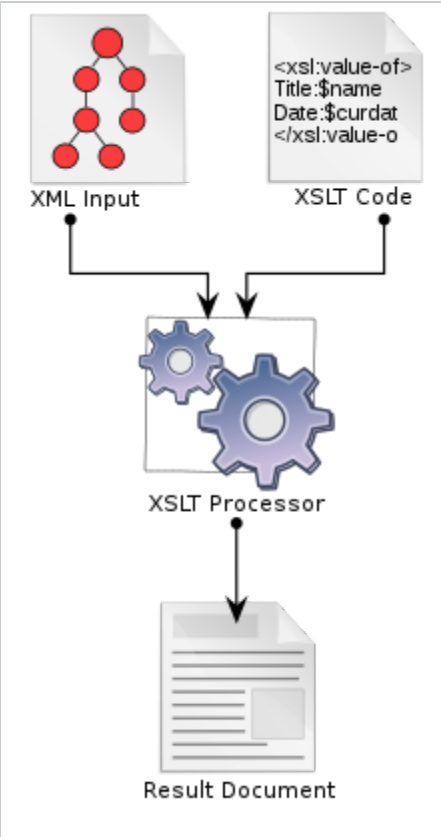
- What is XSLT?

XSLT (Extensible Stylesheet Language Transformations)

XSLT (Extensible Stylesheet Language Transformations) is a language for transforming XML documents into other XML documents, or other formats such as HTML for web pages, plain text or XSL Formatting Objects, which may subsequently be converted to other formats, such as PDF, PostScript and PNG. XSLT 1.0 is widely supported in modern web browsers.

The original document is not changed; rather, a new document is created based on the content of an existing one.Typically, input documents are XML files, but anything from which the processor can build an XQuery and XPath Data Model can be used, such as relational database tables or geographical information systems.

Although XSLT is designed as a special-purpose language for XML transformation, the language is Turing-complete, making it theoretically capable of arbitrary computations.



❖ **What are the differences between XML & XSLT?**

Difference between XML & XSLT written down below

XML	XSLT
XML is used for storing data in a structured format.	XSLT is used for transforming and also for formatting XML files to display the content or to process the content.
XML does not perform transformation of data.	XSLT is a specialized language that performs transformation of one XML document into a different XML document. XSLT can also perform transformation of XML documents into HTML documents and XHTML documents.
XPath is a specialized language that is used to address portions of the XML file.	XSLT will be using XPath for transformation and formatting of XML files.

- How to use XSLT in XML? Explain it.
- **What is Xpath? Discuss the versions of Xpath**.

XPath (XML Path Language) is an expression language designed to support the query or transformation of XML documents. It was defined by the World Wide Web Consortium (W3C). and can be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML document. Support for XPath exists in applications that support XML, such as web browsers, and many programming languages.

- XSLT uses XPath to identify subsets of the source document tree and perform calculations. XPath also provides a range of functions, which XSLT itself further augments.
- XSLT 1.0 uses XPath 1.0, while XSLT 2.0 uses XPath 2.0. XSLT 3.0 will work with either XPath 3.0 or 3.1. In the case of 1.0 and 2.0, the XSLT and XPath specifications were published on the same date. With 3.0, however, they were no longer synchronized; XPath 3.0 became a Recommendation in April 2014, followed by XPath 3.1 in February 2017; XSLT 3.0 followed in June 2017.

Overview

The XPath language is based on a tree representation of the XML document, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria. In popular use (though not in the official specification), an XPath expression is often referred to simply as "an XPath".

Originally motivated by a desire to provide a common syntax and behavior model between XPointer and XSLT, subsets of the XPath query language are used in other W3C specifications such as XML Schema, XForms and the Internationalization Tag Set (ITS). XPath has been adopted by a number of XML processing libraries and tools, many of which also offer CSS Selectors, another W3C standard, as a simpler alternative to XPath.

Versions

There are several versions of XPath in use. **XPath 1.0** was published in 1999, **XPath 2.0** in 2007 (with a second edition in 2010), **XPath 3.0** in 2014, and **XPath 3.1** in 2017. However, XPath 1.0 is still the version that is most widely available.

- XPath 1.0 became a Recommendation on 16 November 1999 and is widely implemented and used, either on its own (called via an API from languages such as Java, C#, Python or JavaScript), or embedded in languages such as XSLT, XProc, XML Schema or XForms.
- XPath 2.0 became a Recommendation on 23 January 2007, with a second edition published on 14 December 2010. A number of implementations exist but are not as widely used as XPath 1.0. The XPath 2.0 language specification is much larger than XPath 1.0 and changes some of the fundamental concepts of the language such as the type system.

  The most notable change is that XPath 2.0 is built around the XQuery and XPath Data Model (XDM) that has a much richer type system. Every value is now a sequence (a single atomic value or node is regarded as a sequence of length one). XPath 1.0

node-sets are replaced by node sequences, which may be in any order.
To support richer type sets, XPath 2.0 offers a greatly expanded set of functions and operators.
XPath 2.0 is in fact a subset of XQuery 1.0. They share the same data model (XDM). It offers a for expression that is a cut-down version of the "FLWOR" expressions in XQuery. It is possible to describe the language by listing the parts of XQuery that it leaves out: the main examples are the query prolog, element and attribute constructors, the remainder of the "FLWOR" syntax, and the typeswitch expression.

- XPath 3.0 became a Recommendation on 8 April 2014. The most significant new feature is support for functions as first-class values. XPath 3.0 is a subset of XQuery 3.0, and most current implementations (April 2014) exist as part of an XQuery 3.0 engine.
- XPath 3.1 became a Recommendation on 21 March 2017. This version adds new data types: maps and arrays, largely to underpin support for JSON.

❖ What is DOM? Describe it.

The **Document Object Model** (DOM) is the data representation of the objects that comprise the structure and content of a document on the web. This guide will introduce the DOM, look at how the DOM represents an HTML document in memory and how to use APIs to create web content and applications.

The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.

A web page is a document that can be either displayed in the browser window or as the HTML source. In both cases, it is the same document but the Document Object Model (DOM) representation allows it to be manipulated. As an object-oriented representation of the web page, it can be modified with a scripting language such as JavaScript.

For example, the DOM specifies that the querySelectorAll method in this code snippet must return a list of all the <p> elements in the document:

```
const paragraphs = document.querySelectorAll("p");
// paragraphs[0] is the first <p> element
// paragraphs[1] is the second <p> element, etc.
alert(paragraphs[0].nodeName);
```

All of the properties, methods, and events available for manipulating and creating web pages are organized into objects. For example, the document object that represents the document itself, any table objects that implement the HTMLTableElement DOM interface for accessing HTML tables, and so forth, are all objects.

The DOM is built using multiple APIs that work together. The core DOM defines the entities describing any document and the objects within it. This is expanded upon as needed by other APIs that add new features and capabilities to the DOM. For example, the HTML DOM API adds support for representing HTML documents to the core DOM, and the SVG API adds support for representing SVG documents.

Accessing the DOM
You don't have to do anything special to begin using the DOM. You use the API directly in JavaScript from within what is called a script, a program run by a browser.

When you create a script, whether inline in a <script> element or included in the web page, you can immediately begin using the API for the document or window objects to manipulate the document itself, or any of the various elements in the web page (the descendant elements of the document). Your DOM programming may be something as simple as the following example, which displays a message on the console by using the console.log() function:
<body onload="console.log('Welcome to my home page!');">

As it is generally not recommended to mix the structure of the page (written in HTML) and manipulation of the DOM (written in JavaScript), the JavaScript parts will be grouped together here, and separated from the HTML.

For example, the following function creates a new <h1> element, adds text to that element, and then adds it to the tree for the document:

```
<html>
  <head>
    <script>
      // run this function when the document is loaded
      window.onload = function() {

        // create a couple of elements in an otherwise empty HTML page
        const heading = document.createElement("h1");
        const heading_text = document.createTextNode("Big Head!");
        heading.appendChild(heading_text);
        document.body.appendChild(heading);
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

13.Kernel

❖ What is kernel? What are the functions of the kernel?

A Kernel is a computer program that is the heart and core of an Operating System. Since the Operating System has control over the system, the Kernel also has control over everything in the system. It is the most important part of an Operating System. Whenever a system starts, the Kernel is the first program that is loaded after the bootloader because the Kernel has to handle the rest of the thing of the system for the Operating System. The Kernel remains in the memory until the Operating System is shut-down.
The Kernel is responsible for low-level tasks such as disk management, memory management, task management, etc. It provides an interface between the user and the hardware components of the system. When a process makes a request to the Kernel, then it is called System Call.
A Kernel is provided with a protected Kernel Space which is a separate area of memory and this area is not accessible by other application programs. So, the code of the Kernel is loaded into this protected Kernel Space. Apart from this, the memory used by other applications is called the User Space. As these are two different spaces in the memory, so communication between them is a bit slower.

Functions of a Kernel
Following are the functions of a Kernel:

- Access Computer resource: A Kernel can access various computer resources like the CPU, I/O devices and other resources. It acts as a bridge between the user and the resources of the system.
- Resource Management: It is the duty of a Kernel to share the resources between various processes in such a way that there is uniform access to the resources by every process.
- Memory Management: Every process needs some memory space. So, memory must be allocated and deallocated for its execution. All these memory management is done by a Kernel.
- Device Management: The peripheral devices connected in the system are used by the processes. So, the allocation of these devices is managed by the Kernel.

❖ Explain kernel module programming.

Kernel modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system.
Custom codes can be added to Linux kernels via two methods.

- The basic way is to add the code to the kernel source tree and recompile the kernel.
- A more efficient way to do this is by adding code to the kernel while it is running. This process is called loading the module, where module refers to the code that we want to add to the kernel.

Since we are loading these codes at runtime and they are not part of the official Linux kernel, these are called loadable kernel modules(LKM), which is different from the “base kernel”. Base kernel is located in the /boot directory and is always loaded when we boot our machine whereas

LKMs are loaded after the base kernel is already loaded. Nonetheless, these LKM are very much part of our kernel and they communicate with the base kernel to complete their functions.

LKMs can perform a variety of task, but basically they come under three main categories,

- device driver,
- filesystem driver and
- System calls.

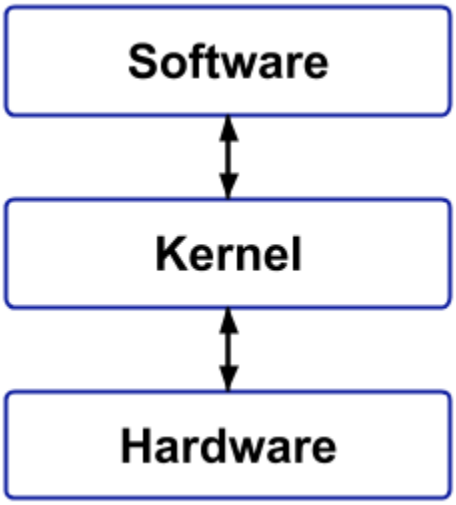
❖ What are the types of kernel?

Types of Kernel

In general, there are five types of Kernel. They are:

1. **Monolithic Kernels**

Monolithic Kernels are those Kernels where the user services and the kernel services are implemented in the same memory space i.e. different memory for user services and kernel services are not used in this case. By doing so, the size of the Kernel is increased and this, in turn, increases the size of the Operating System. As there is no separate User Space and Kernel Space, so the execution of the process will be faster in Monolithic Kernels.



Advantages:

- It provides CPU scheduling, memory scheduling, file management through System calls only.
- Execution of the process is fast because there is no separate memory space for user and kernel.

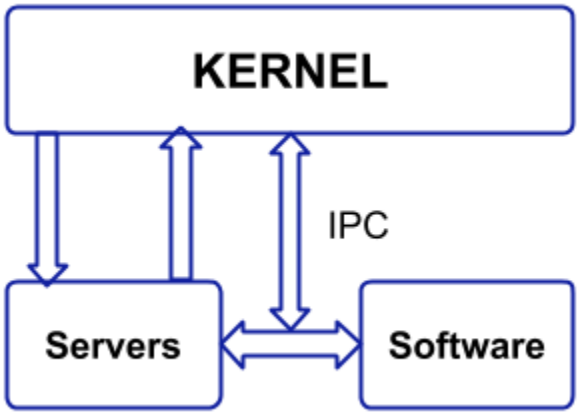
Disadvantages:

- If any service fails, then it leads to system failure.
- If new services are to be added then the entire Operating System needs to be modified.

2. **Microkernel**

A Microkernel is different from a Monolithic kernel because in a Microkernel, the user services and kernel services are implemented into different spaces i.e. we use User Space and Kernel

Space in case of Microkernels. As we are using User Space and Kernel Space separately, it reduces the size of the Kernel and this, in turn, reduces the size of the Operating System.



As we are using different spaces for user services and kernel service, so the communication between application and services is done with the help of message parsing and this, in turn, reduces the speed of execution.
Advantages:

- If new services are to be added then it can be easily added.

Disadvantages:

- Since we are using User Space and Kernel Space separately, so the communication between these can reduce the overall execution time.

3. **Hybrid Kernel**

A Hybrid Kernel is a combination of both Monolithic Kernel and Microkernel. It makes use of the speed of the Monolithic Kernel and the modularity of Microkernel.
Hybrid kernels are micro kernels that have some "non-essential" code in kernel-space in order for the code to run more quickly than it would be in user-space. So, some services such as network stack or filesystem are run in Kernel space to reduce the performance overhead, but still, it runs kernel code as servers in the user-space.

4. **Nanokernel**

In a Nanokernel, as the name suggests, the whole code of the kernel is very small i.e. the code executing in the privileged mode of the hardware is very small. The term nanokernel is used to describe a kernel that supports a nanosecond clock resolution.

5. **Exokernel**

Exokernel is an Operating System kernel that is developed by the MIT parallel and the Distributed Operating Systems group. Here in this type of kernel, the resource protection is separated from the management and this, in turn, results in allowing us to perform application-specific customization.
In the Exokernel, the idea is not to implement all the abstractions. But the idea is to impose as few abstractions as possible and by doing so the abstraction should be used only when needed. So, no force abstraction will be there in Exokernel and this is the feature that makes it different

from a Monolithic Kernel and Microkernel. But the drawback of this is the complex design. The design of the Exokernel is very complex.

- **Describe kernel log level string.**

The Linux kernel log levels / Kernel Log Level String

There are basically eight log levels which a message sent by the linux kernel can adopt, starting from level 0 and decreasing in severity 'till level 7: the lowest log level identifier, the most critical context.

When a log level is set as the default for the console, either persistently or temporarily, it acts as a filter, so that only messages with a log level lower than it, (therefore messages with an higher severity) are displayed. Let's see, briefly, how log levels are organized:

The first log level is 0, identified by the **KERN_EMERG** string. This is the highest level in order of severity: it's adopted by messages about system instability or imminent crashes.

Loglevel 1, or KERN_ALERT it's what comes immediately after. This level is used in situations where the user's attention is immediately required.

The next log level in order of severity is **KERN_CRIT, or loglevel 2**. This level of severity is used to inform about critical errors, both hardware or software related.

Loglevel 3, also identified by the **KERN_ERR string**, it's the next in the scale. Messages adopting this level are often used to notify the user about non-critical errors, as for example a failed or problematic device recognition, or more generally driver-related problems.

KERN_WARNING, or loglevel 4 it's the log level usually used as the default in the majority of linux distributions. This level it's used to display warnings or messages about non imminent errors.

Loglevel 5 it's KERN_NOTICE. Messages which use this level of severity are about events which may be worth noting.

Normal situations worthy of note

Loglevel 6 it's KERN_INFO: this is the log level used for informational messages about the action performed by the kernel.

Finally, we have **KERN_DEBUG, or loglevel 7**, which is mainly used for debugging.

14.Linux

❖ What are the basic components of Linux?

First you need to understand what a computer operating system or OS is. An OS is the computer code which manages the hardware in a physical computer. It exists as a layer between your software and your hardware. Most people writing software do not want to know how to address a CPU in assembler, or how to communicate with a graphics card. An OS such as Linux or Windows acts as a middleman.

Linux is one of the most commonly used operating systems, underpinning everything from PCs through to servers and mobile phones. In fact, Linux has been around since the 90s and is in use around the world, and in every application and field imaginable.

Every OS has component parts, and the Linux OS also has the following components parts:

- Bootloader. Your computer needs to go through a startup sequence called booting. This boot process needs guidance, and your OS is the software in control throughout the boot process. When you start your computer the bootloader for your operating system kickstarts the process.
- OS Kernel. You can call the kernel the part of the operating system which is the “closest” to your computing hardware as it is the part which controls the CPU, access to memory and any peripheral devices. It is the “lowest” level at which your operating system works.
- Background services. Called “daemons” in Linux, these small applications act as servants in the background, ensuring that key functions such as scheduling, printing and multimedia function correctly. They load after you have booted up, or when you have logged into your computer.
- OS Shell. You need to be able to tell our operating system what to do, and this is the goal of the shell. Also known as the command line, it is a facility which lets you instruct your OS using text. However few people nowadays are familiar with command line code, and it once used to put people off using Linux. This changed because a modern distribution of Linux will use a desktop shell just like Windows.
- Graphics server. This provides a graphical subsystem that renders images and shapes on your computer monitor. Linux uses a graphical server called “X” or “X-server”.
- Desktop environment. You can’t interact with the graphical server directly. Instead you need software that can drive the server. This is called a desktop environment in Linux and there are plenty of options including KDE, Unity and Cinnamon. A desktop environment is usually bundled with a number of applications including file and web browsers plus a couple of games.
- Applications. Obviously, the desktop environment which is bundled with your Linux OS or which you choose to install cannot cater for every application need, there are too many. Individual applications, however, can and there are thousands for Linux just like Windows and Apple’s OS X has thousands of applications. Most Linux distros have app stores which help you find and install apps, for example Ubuntu Software which comes with Ubuntu.

It's worth noting that Ubuntu's application repository, the Ubuntu software center, is a great place to look around for Linux applications, both free to use and paid to use.

- ❖ **What is Linux shell? What types of shells are there in linux?**

Shell is a UNIX term for the interactive user interface with an operating system. The shell is the layer of programming that understands and executes the commands a user enters. In some systems, the shell is called a command interpreter.

Different Types of Shells in Linux

If you now understand what a kernel is, what a shell is, and why a shell is so important for Linux systems, let's move on to learning about the different types of shells that are available. Each of these shells has properties that make them highly efficient for a specific type of use over other shells. So let us discuss the different types of shells in Linux along with their properties and features.

1. **The Bourne Shell (sh)**

Developed at AT&T Bell Labs by Steve Bourne, the Bourne shell is regarded as the first UNIX shell ever. It is denoted as sh. It gained popularity due to its compact nature and high speeds of operation.

This is what made it the default shell for Solaris OS. It is also used as the default shell for all Solaris system administration scripts. Start reading about shell scripting here.
However, the Bourne shell has some major drawbacks.

- It doesn't have in-built functionality to handle logical and arithmetic operations.
- Also, unlike most different types of shells in Linux, the Bourne shell cannot recall previously used commands.
- It also lacks comprehensive features to offer a proper interactive use.

The complete path-name for the Bourne shell is /bin/sh and /sbin/sh. By default, it uses the prompt # for the root user and $ for the non-root users.

2. **The GNU Bourne-Again Shell (bash)**

More popularly known as the Bash shell, the GNU Bourne-Again shell was designed to be compatible with the Bourne shell. It incorporates useful features from different types of shells in Linux such as Korn shell and C shell.

It allows us to automatically recall previously used commands and edit them with the help of arrow keys, unlike the Bourne shell.

The complete path-name for the GNU Bourne-Again shell is /bin/bash. By default, it uses the prompt bash-VersionNumber# for the root user and bash-VersionNumber$ for the non-root users.

3. **The C Shell (csh)**

The C shell was created at the University of California by Bill Joy. It is denoted as csh. It was developed to include useful programming features like in-built support for arithmetic operations and a syntax similar to the C programming language.

Further, it incorporated command history which was missing in different types of shells in Linux like the Bourne shell. Another prominent feature of a C shell is “aliases”.
The complete path-name for the C shell is /bin/csh. By default, it uses the prompt hostname# for the root user and hostname% for the non-root users.

4. **The Korn Shell (ksh)**

The Korn shell was developed at AT&T Bell Labs by David Korn, to improve the Bourne shell. It is denoted as ksh. The Korn shell is essentially a superset of the Bourne shell.

Besides supporting everything that would be supported by the Bourne shell, it provides users with new functionalities. It allows in-built support for arithmetic operations while offering interactive features which are similar to the C shell.

The Korn shell runs scripts made for the Bourne shell, while offering string, array and function manipulation similar to the C programming language. It also supports scripts which were written for the C shell. Further, it is faster than most different types of shells in Linux, including the C shell.

The complete path-name for the Korn shell is /bin/ksh. By default, it uses the prompt # for the root user and $ for the non-root users.

5. **The Z Shell (zsh)**

The Z Shell or zsh is a sh shell extension with tons of improvements for customization. If you want a modern shell that has all the features and much more, the zsh shell is what you’re looking for.
Some noteworthy features of the z shell include:

- Generate filenames based on given conditions
- Plugins and theming support
- Index of built-in functions
- Command completion
- and many more…

Let us summarize the different shells in Linux which we discussed in this tutorial in the table below.

Shell	Complete path-name	Prompt for root user	Prompt for non root user
Bourne shell (sh)	/bin/sh and /sbin/sh	#	$
GNU Bourne-Again shell (bash)	/bin/bash	bash-VersionNumber#	bash-VersionNumber$
C shell (csh)	/bin/csh	#	%
Korn shell (ksh)	/bin/ksh	#	$

Z Shell (zsh)	/bin/zsh	<hostname>#	<hostname>%

15.Assembler & Compiler

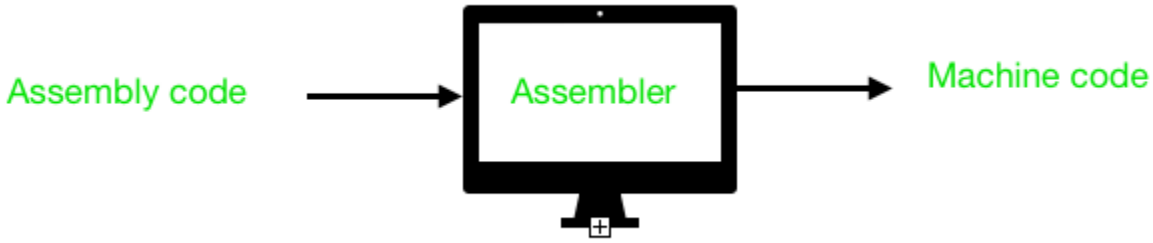
- **What is an assembler**? Explain how assemblers work?

An assembler is a program that converts assembly language into machine code. It takes the basic commands and operations from assembly code and converts them into binary code that can be recognized by a specific type of processor.

Assemblers are similar to compilers in that they produce executable code. However, assemblers are more simplistic since they only convert low-level code (assembly language) to machine code. Since each assembly language is designed for a specific processor, assembling a program is performed using a simple one-to-one mapping from assembly code to machine code. Compilers, on the other hand, must convert generic high-level source code into machine code for a specific processor.

Most programs are written in high-level programming languages and are compiled directly to machine code using a compiler. However, in some cases, assembly code may be used to customize functions and ensure they perform in a specific way. Therefore, IDEs often include assemblers so they can build programs from both high and low-level languages.

Assembler is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader.



- What are the functions of a basic assembler?

Functions of a Basic Assembler

- Convert mnemonic operation codes to their machine language equivalents

E.g. STL -> 14

- Convert symbolic operands to their equivalent machine addresses

E.g. RETADR -> 1033

- Build the machine instructions in the proper format
- Convert the data constants to internal machine representations

E.g. EOF -> 454F46

- **Write the features of machine dependent assembler & machine independent assembler.**

Machine-dependent Assembler Features

Instruction format
Addressing mode
Program relocation

Machine independent assembler features

Literals
Symbol defining statements
Expressions
Program blocks and control sections

- What is a base register?

Not finished yet

- **Briefly explain the single and multi pass compiler.**

There are a number of stages in the compilation process.

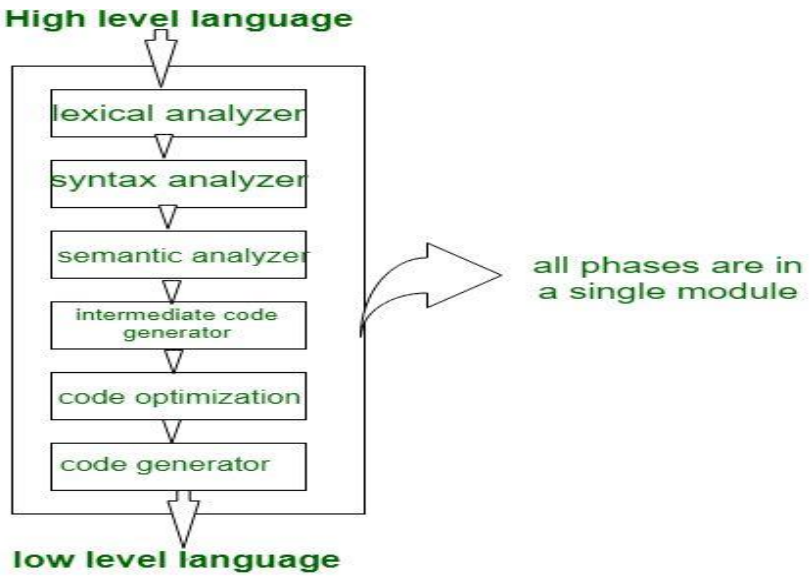
1. Single pass compiler
2. Multi pass compiler

Single Pass Compiler

In computer programming, a one-pass compiler is a compiler that passes through the parts of each compilation unit only once, immediately translating each part into its final machine code. This is in contrast to a multi-pass compiler which converts the program into one or more intermediate representations in steps between source code and machine code, and which reprocesses the entire compilation unit in each sequential pass.
This refers to the logical functioning of the compiler, not to the actual reading of the source file once only. For instance, the source file could be read once into temporary storage but that copy could then be scanned many times. The IBM 1130 Fortran compiler stored the source in memory and used many passes; by contrast the assembler, on systems lacking a disc storage unit, required that the source deck of cards be presented twice to the card reader/punch.

Main stages of single pass compiler are lexical analysis, syntactical analysis and code generator. First, the lexical analysis scans the source code and divides it into tokens. Every programming language has a grammar. It represents the syntax and legal statements of the

language. Then, the syntactical analysis determines the language constructs described by the grammar. Finally, the code generator generates the target code. Overall, single pass compiler does not optimize the code. Moreover, there is no intermediate code generation.



Multi pass compiler

A multi pass compiler makes the source code go through parsing, analyzing, generating, etc. multiple times while generating intermediate code after each stage. It converts the program into one or more intermediate representations in steps between source code and machine code. It reprocesses the entire compilation unit in each sequential pass.
Each pass takes the result of the previous pass as the input and creates an intermediate output. Likewise, in each pass, the code improves until the final pass generates the final code. A multi pass compiler performs additional tasks such as intermediate code generation, machine dependent code optimization and machine independent code optimization.

- What are the differences between single and multi pass compilers?

16.Linker & Loader

- Define linkerk, dynamic linker, static linker and linking editor

Not finished yet

- What is a loader? Explain the responsibilities of it.

In computer systems a loader is the part of an operating system that is responsible for loading programs and libraries. It is one of the essential stages in the process of starting a program, as it places programs into memory and prepares them for execution. Loading a program involves reading the contents of the executable file containing the program instructions into memory, and then carrying out other required preparatory tasks to prepare the executable for running. Once loading is complete, the operating system starts the program by passing control to the loaded program code.

All operating systems that support program loading have loaders, apart from highly specialized computer systems that only have a fixed set of specialized programs. Embedded systems typically do not have loaders, and instead, the code executes directly from ROM or similar. In order to load the operating system itself, as part of booting, a specialized boot loader is used. In many operating systems, the loader resides permanently in memory, though some operating systems that support virtual memory may allow the loader to be located in a region of memory that is pageable.

In the case of operating systems that support virtual memory, the loader may not actually copy the contents of executable files into memory, but rather may simply declare to the virtual memory subsystem that there is a mapping between a region of memory allocated to contain the running program's code and the contents of the associated executable file. The virtual memory subsystem is then made aware that pages with that region of memory need to be filled on demand if and when program execution actually hits those areas of unfilled memory. This may mean parts of a program's code are not actually copied into memory until they are actually used, and unused code may never be loaded into memory at all.

Responsibilities
In Unix, the loader is the handler for the system call execve(). The Unix loader's tasks include:

validation (permissions, memory requirements etc.);
memory-mapping the executable object from the disk into main memory;
copying the command-line arguments on the stack;
initializing registers (e.g., the stack pointer);
jumping to the program entry point (_start).

- ❖ Write the features of **machine dependent loader** and **machine independent loader**

Machine-Independent Loader Features
Loading and linking are often thought of as operating system service functions.

Machine independent loader features:
1. Automatic Library Search
2. Loader Options

Automatic Library Search for handling external references

Allows programmers to use standard subroutines without explicitly including them in the program to be loaded.
The routines are automatically retrieved from a library as they are needed during linking.

Loader Options
Common options that can be selected at the time of loading and linking
Automatic Library Search
Standard system library, subprogram library
Automatic library search (Automatic library call)
The programmer does not need to take any action beyond mentioning the subroutine names as external references

Linking loaders that support automatic library search:
Must keep track of external symbols that are referred to, but not defined, in the primary input to the loader

Loader Options
Many loaders have a special command language that is used to specify options n a separate input file to the loader that contain control statements
Control statements embedded in the primary input stream between object programs
Control statements are included in the source program, and the assembler or compiler retains these commands as a part of the object program
Machine-dependent Loader Features

1. Relocation
2. Program Linking
3. Algorithm and Data Structures for a Linking Loader

❖ Explain absolute and bootstrap loader.

Absolute loader
The absolute loader is a kind of loader in which relocated object files are created, loader accepts these files and places them at a specified location in the memory.
This type of loader is called absolute loader because no relocating information is needed, rather it is obtained from the programmer or assembler.
The starting address of every module is known to the programmer, this corresponding starting address is stored in the object file then the task of loader becomes very simple, that is to simply place the executable form of the machine instructions at the locations mentioned in the object file.

In this scheme, the programmer or assembler should have knowledge of memory management. The programmer should take care of two things:
Specification of starting address of each module to be used. If some modification is done in some module then the length of that module may vary. This causes a change in the

starting address of immediate next modules, it's then the programmer's duty to make necessary changes in the starting address of respective modules.
While branching from one segment to another the absolute starting address of respective module is to be known by the programmer so that such address can be specified at respective JMP instruction.

Bootstrap Loader
A Bootstrap Loader (BSL) is a small program which can be activated immediately after a microcontroller has been powered up, in order to load and execute another program in a well defined manner. In the next step those routines are used to upload a new firmware and save it in the program memory.

- What are the differences between linker and loader?

LINKER	LOADER
1. The main function of Linker is to **generate** executable files.	1. Whereas main objective of Loader is to **load** executable files to main memory.
2. The linker takes input of object code generated by compiler/assembler.	2. And the loader takes input of executable files generated by linker.
3. Linking can be defined as process of combining various pieces of codes and source code to obtain executable code.	3. Loading can be defined as process of loading executable codes to main memory for further execution.
4. Linkers are of 2 types: Linkage Editor and Dynamic Linker.	4. Loaders are of 4 types: Absolute, Relocating, Direct Linking, Bootstrap.
5. Another use of linker is to **combine** all object modules.	5.It helps in **allocating** the address to executable codes/files.
6. Linker is also responsible for arranging objects in program's address space.	6. Loader is also responsible for adjusting references which are used within the program.

17.IOT

- **Explain IOT in embedded system programming with example**

An embedded system in IoT

An embedded system in IoT is still an embedded system. What makes it different is that it happens to also connect to the internet, or another network like home network, in order to

perform functions that go beyond what's happening on the system itself. This expands the range of functions that otherwise wouldn't have been possible without being connected.

Take a smart refrigerator.It may have sensors inside the fridge that can detect levels of produce. Through its connection to the internet (and likely a smartphone and/or web application), it can order produce from a store when levels of a certain item fall by a certain amount.

ARM, whose designs are behind many of the microprocessors running IoT devices, define embedded systems in IoT like this:

> "IoT devices are pieces of hardware, such as sensors, actuators, gadgets, appliances, or machines, that are programmed for certain applications and can transmit data over the internet or other networks. They can be embedded into other mobile devices, industrial equipment, environmental sensors, medical devices, and more."
>
> When combined with the ability to transmit data over the internet, it's all these sensors, basically making them an IoT device. An embedded system in IoT is an embedded device that has that IoT functionality added to it. (Note that an Industrial Internet of Things (IIoT) device doesn't necessarily use the internet to transmit data.)
>
> In transmitting data, these devices can be connected to one another and potentially make "smart" decisions themselves or help end-users make smart decisions.
>
> Consider a personal fitness tracker that connects with an application on a smartphone. The tracker records activity while an app on the smartphone may track this against calories consumed, so the user can decide whether they can treat themselves to a doughnut.
>
> Or it could be a "smart" insulin pump (technically, an Internet of Medical Things (IoMT) device) that keeps track of a user's sugar levels and helps them monitor their sugar and insulin levels through a phone app, and deliver insulin as needed.
>
> What's important to remember is that:
>
> All IoT devices have embedded systems, but not all embedded systems are part of the IoT.

- The significance of embedded system in IOT

The significance of embedded systems in IoT

> Embedded systems are a vital part of the foundations behind the IoT ecosystem. The significance of an embedded system in IoT is that much of the IoT wouldn't exist as we know it without the embedded systems that help devices function.
>
> While the internet is essential in transmitting data to and from IoT devices to online (cloud) services, it's embedded systems that will enable this data to be sent and often interpreted at a local level.
>
> The data that embedded systems are interpreting can come from applications on smartphones, cloud services and other nearby computers. Still, the critical source of information is the host of sensors that input real-world data in real-time. Embedded systems will then decide whether this data is significant enough to be transmitted through whatever connectivity means an IoT device has.

- Challenges of IOT embedded software

Challenges of IoT embedded software

Developing embedded software in IoT faces many similar challenges to developing software for embedded systems that are not connected to the internet. Similarly, some software development difficulties here become an even greater focus when the software is for an IoT product.
Here are three top challenges facing IoT embedded software development:

1. Security

   The device's security is possibly the biggest and most important challenge faced in coding for an IoT product. While much of the challenge here comes from the hardware decisions that enable devices to connect, the software is also responsible for mitigating security risk. Connected devices have a larger attack surface area than a device that isn't connected to a wider network. How big is this issue? In 2020, Nokia found that IoT devices accounted for 32.72% of malware infections observed by its endpoint security software . And that's just what Nokia saw. Other security software partners have recorded similar infection rates in IoT.

   An unsecured device presents a threat to the privacy of any data transmitted by the device. The device could also become part of a botnet, used to launch other cyberattacks. But the worst-case scenario? An IoT enabled device could become compromised and lead to loss of life. For IoT in automotive, IoMT and IIoT, the risk here is immense when security isn't accounted for. Still, even a consumer-level IoT product such as a smart fridge, kettle, or television is a safety risk with inadequate security.

2. Limited memory

   Embedded systems generally do not have the same vast amounts of memory and processing speeds that a personal computer does (for various reasons, including cost). Due to these hardware limitations, embedded software needs to be as efficient and lean as possible.

   Every line of code must have a reason to be there; every feature must have a purpose. An embedded system won't have gigabytes of memory directly available to it. It likely won't even have megabytes.

   Our fitness band example—that wearable IoT device?

   The embedded system within would likely only have 256 kilobytes (kB) of flash memory (along with 32kB SRAM and 8kB EEPROM). A floppy disk from the 1990s would have around 1.41 megabytes (MB) of memory—that's almost six times more memory than the fitness band.

   But unlike that floppy disk, the band needs to be able to monitor its sensors, record data and communicate with its paired smartphone through a Bluetooth beacon. The Bluetooth controller stack, the software that will enable Bluetooth to function, could take up around 192 kB in flash memory (and 24 kB of RAM). And that's just to get the Bluetooth to work.

3. Updates

An IoT device's significant advantage is its connectivity, which means embedded software in an IoT product can benefit from over-the-air updates as can the applications on any connecting smartphones or cloud services.

Updates to software can fix flaws that weren't previously uncovered in testing or fixing security holes found in software libraries used in coding the device software. Device updates can also add new features or enhance existing ones for end-users and extend the product's marketability and relevancy.

But to keep up with the pace of development needed—embedded software development teams need to be coding and testing to support this and have processes that synch up with application developers and other specialists in a product team. These processes need to be capable of:

Understanding when updates are needed.

Developing timely updates efficiently and effectively.

Fully testing the updates to make sure nothing breaks on the device and its application.

Carrying out usability testing before rolling out.

Making any needed changes found through software testing and usability testing.

Securing updates and validating they're coming from a trusted source.

Figuring out when it's best to push out updates.

Providing rollbacks or suspension of updates if issues are found after deployment.

Working with the limitations of device hardware and connectivity.

18.Memory Management

- **What is memory management?** Explain the steps of memory management.

Memory management is the process of controlling and coordinating computer memory, assigning portions called blocks to various running programs to optimize overall system performance. Memory management resides in hardware, in the OS (operating system), and in programs and applications.

In hardware, memory management involves components that physically store data, such as RAM (random access memory) chips, memory caches, and flash-based SSDs (solid-state drives). In the OS, memory management involves the allocation (and constant reallocation) of specific memory blocks to individual programs as user demands change. At the application level, memory management ensures the availability of adequate memory for the objects and data structures of each running program at all times. Application memory management combines two related tasks, known as allocation and recycling.

- When the program requests a block of memory, a part of the memory manager called the allocator assigns that block to the program.
- When a program no longer needs the data in previously allocated memory blocks, those blocks become available for reassignment. This task can be done manually (by the programmer) or automatically (by the memory manager).

Steps of Memory Management
Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.
This tutorial will teach you basic concepts related to Memory Management.

- **Process address space**

Process Address Space
The process address space is the set of logical addresses that a process references in its code. For example, when 32-bit addressing is in use, addresses can range from 0 to 0x7fffffff; that is, 2^31 possible numbers, for a total theoretical size of 2 gigabytes.
The operating system takes care of mapping the logical addresses to physical addresses at the time of memory allocation to the program. There are three types of addresses used in a program before and after memory is allocated −

S.N.	Memory Addresses & Description
1	Symbolic addresses The addresses used in a source code. The variable names, constants, and instruction labels are the basic elements of the symbolic address space.
2	Relative addresses At the time of compilation, a compiler converts symbolic addresses into relative addresses.
3	Physical addresses The loader generates these addresses at the time when a program is loaded into main memory.

Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.

The set of all logical addresses generated by a program is referred to as a logical address space. The set of all physical addresses corresponding to these logical addresses is referred to as a physical address space.
The runtime mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address.

- The value in the base register is added to every address generated by a user process, which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.
- The user program deals with virtual addresses; it never sees the real physical addresses.

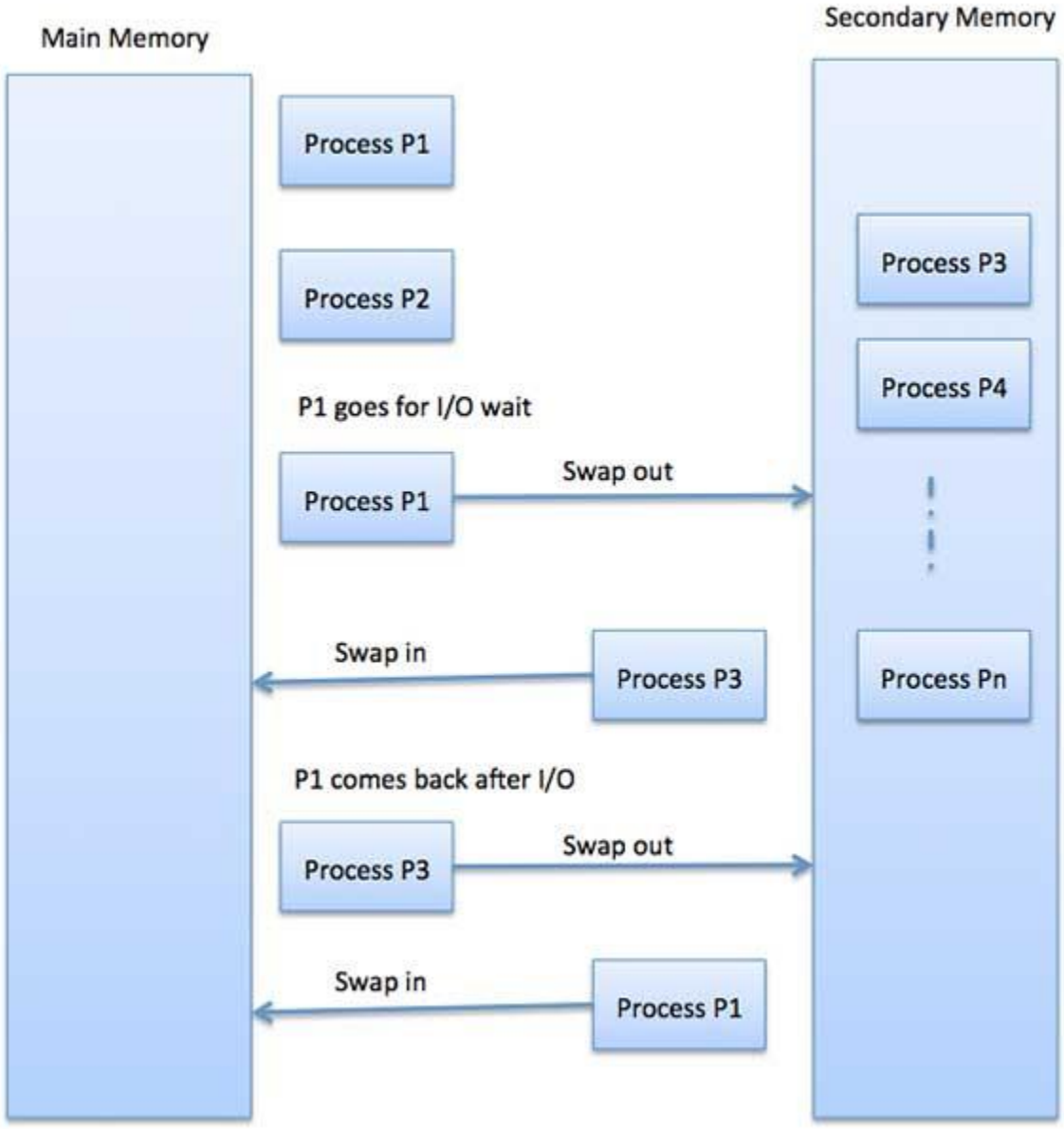
➢ Static vs dynamic loading

Static vs Dynamic Loading
The choice between Static or Dynamic Loading is to be made at the time of computer program being developed. If you have to load your program statically, then at the time of compilation, the complete programs will be compiled and linked without leaving any external program or module dependency. The linker combines the object program with other necessary object modules into an absolute program, which also includes logical addresses.
If you are writing a Dynamically loaded program, then your compiler will compile the program and for all the modules which you want to include dynamically, only references will be provided and rest of the work will be done at the time of execution.
At the time of loading, with static loading, the absolute program (and data) is loaded into memory in order for execution to start.
If you are using dynamic loading, dynamic routines of the library are stored on a disk in relocatable form and are loaded into memory only when they are needed by the program.
Static vs Dynamic Linking
As explained above, when static linking is used, the linker combines all other modules needed by a program into a single executable program to avoid any runtime dependency.
When dynamic linking is used, it is not required to link the actual module or library with the program, rather a reference to the dynamic module is provided at the time of compilation and linking. Dynamic Link Libraries (DLL) in Windows and Shared Objects in Unix are good examples of dynamic libraries.

➢ **Swapping**

Swapping
Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason Swapping is also known as a technique for memory compaction.



The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.
Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take
2048 KB / 1024KB per second
= 2 seconds
= 2000 milliseconds

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

➢ Memory allocation

Memory Allocation

Main memory usually has two partitions −

- Low Memory − Operating system resides in this memory.
- High Memory − User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

S.N.	Memory Allocation & Description
1	Single-partition allocation In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.
2	Multiple-partition allocation In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

➢ **Fragmentation**

Fragmentation

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after that processes cannot be allocated to memory blocks considering their small size and memory blocks remain unused. This problem is known as Fragmentation.

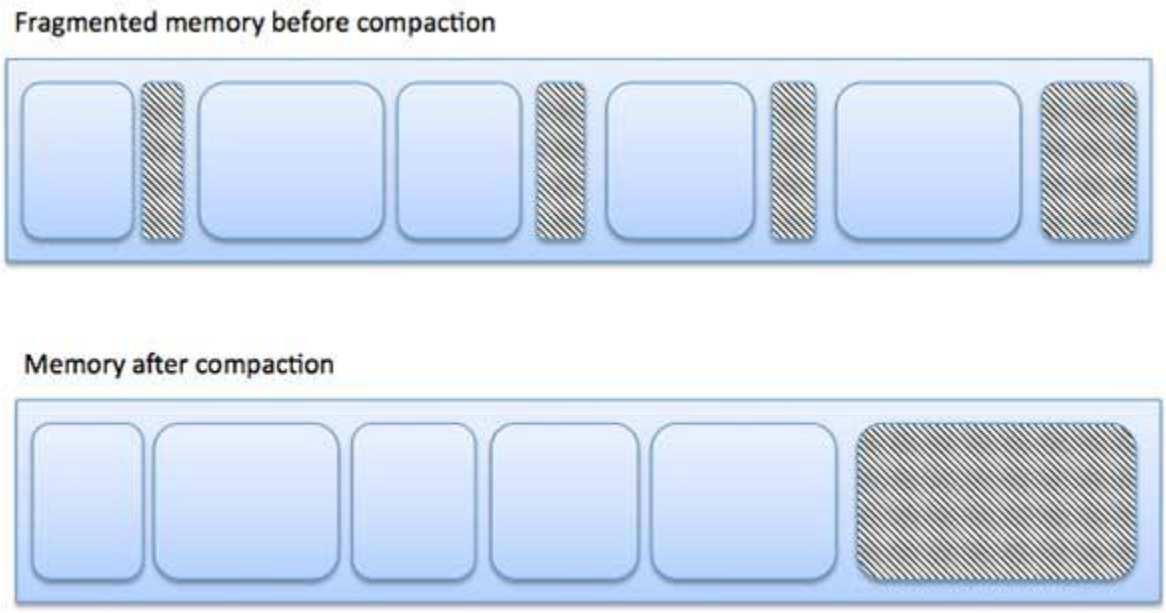
Fragmentation is an unwanted problem where the memory blocks cannot be allocated to the processes due to their small size and the blocks remain unused. It can also be understood as when the processes are loaded and removed from the memory they create free space or holes in the memory and these small blocks cannot be allocated to new upcoming processes and results in inefficient use of memory. Basically, there are two types of fragmentation

Fragmentation is of two types −

S.N.	Fragmentation & Description
1	**External fragmentation** Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.

	In this fragmentation, although we have total space available that is needed by a process still we are not able to put that process in the memory because that space is not contiguous. This is called external fragmentation.
2	**Internal fragmentation** Memory block assigned to the process is bigger. Some portion of memory is left unused, as it cannot be used by another process. In this fragmentation, the process is allocated a memory block of size more than the size of that process. Due to this some part of the memory is left unused and this causes internal fragmentation.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory −



External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

External fragmentation can be reduced by merging all the free memory together in one large block. This technique is also called **defragmentation**.

The internal fragmentation can be reduced by **effectively assigning the smallest partition but large enough for the process**.

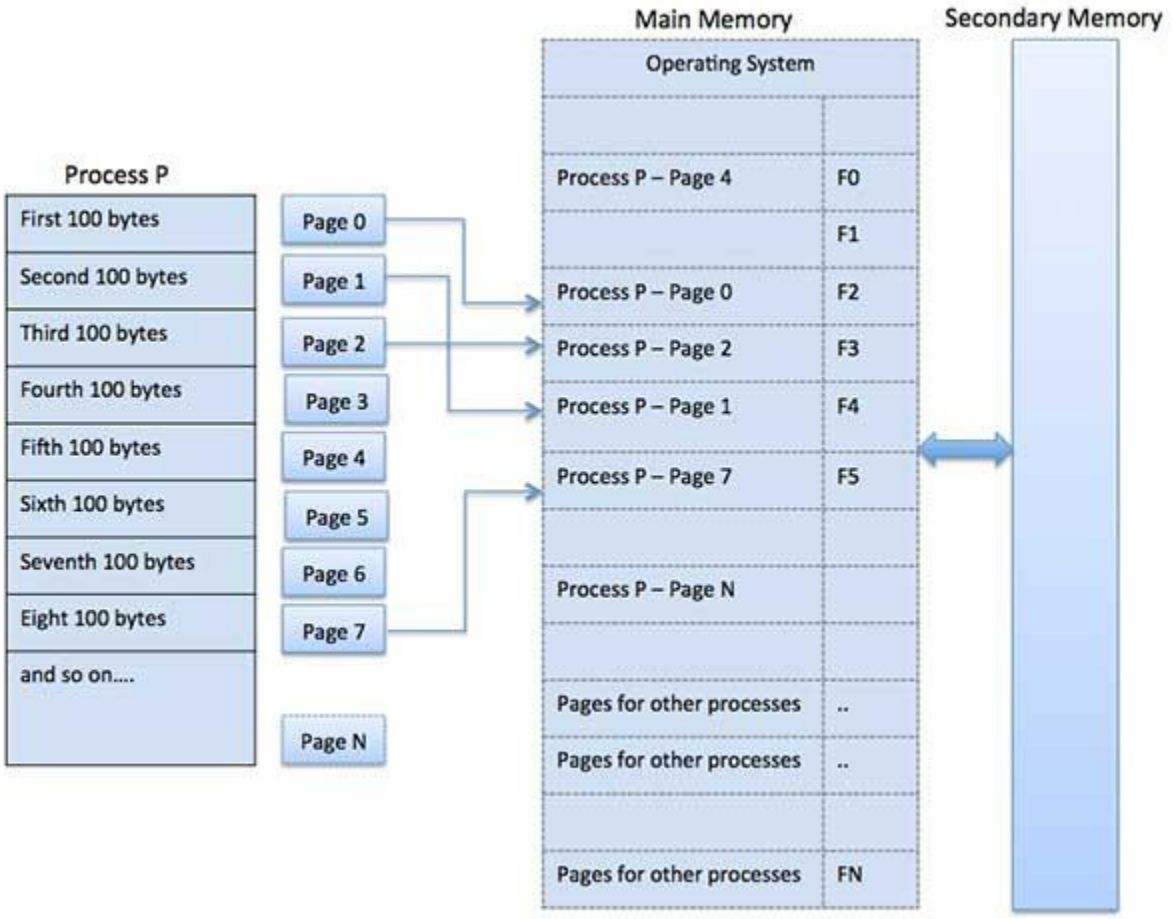
➢ **Paging**

Paging

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called pages (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called frames and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.



➢ Address translation

Address Translation

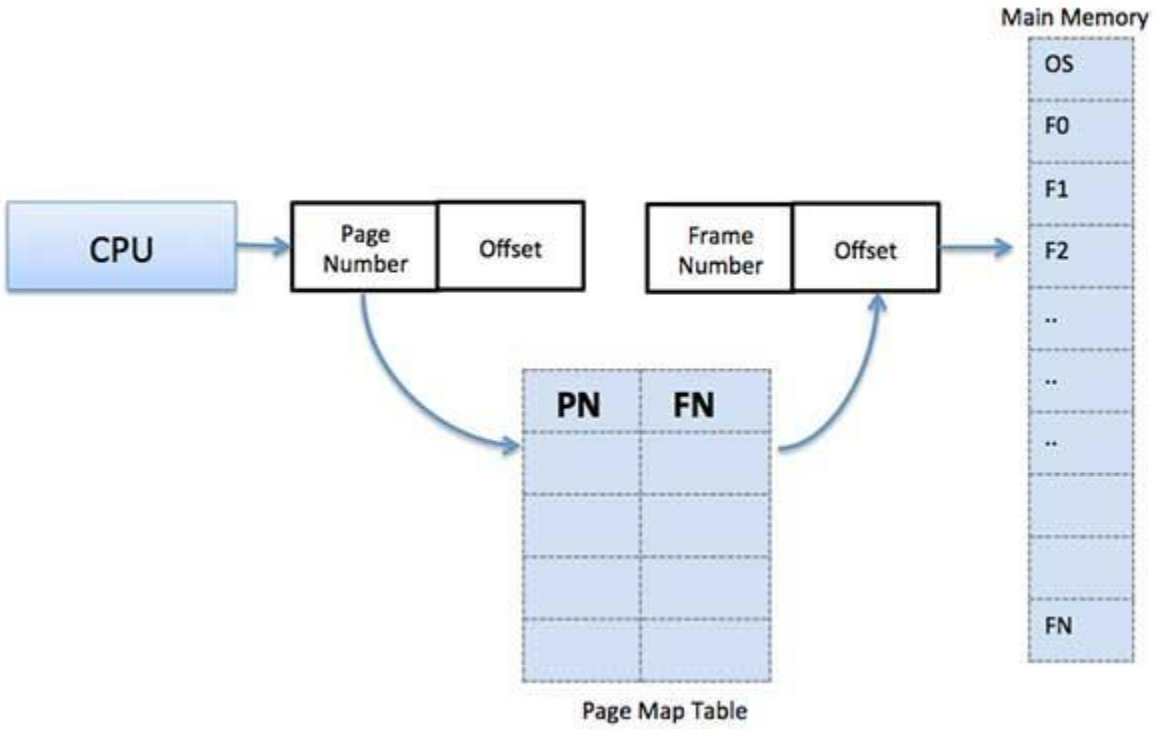
Page address is called logical address and represented by page number and the offset.

Logical Address = Page number + page offset

Frame address is called physical address and represented by a frame number and the offset.

Physical Address = Frame number + page offset

A data structure called page map table is used to keep track of the relation between a page of a process to a frame in physical memory.



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program. When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging −

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

➢ Segmentation

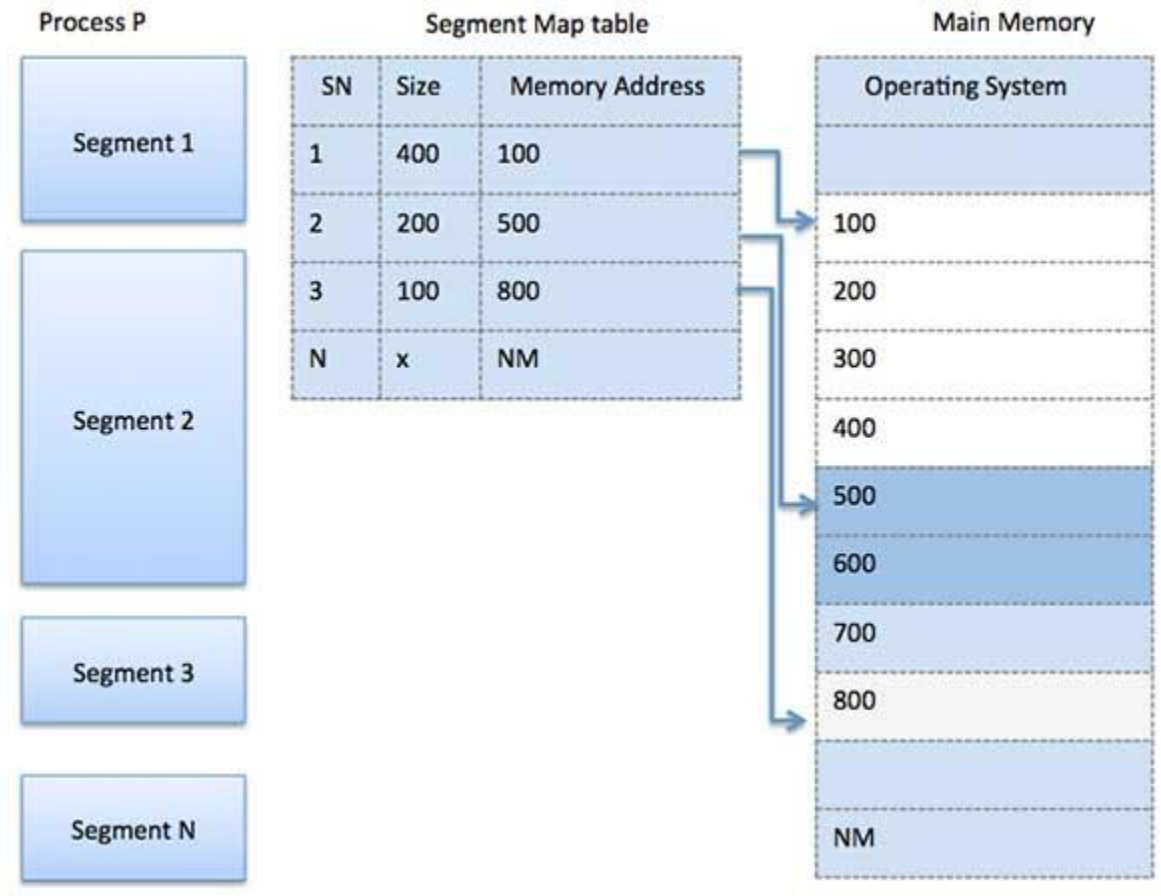
Segmentation

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a segment map table for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.



- Explain interrupt handler.

In computer systems programming, an interrupt handler, also known as an interrupt service routine or ISR, is a special block of code associated with a specific interrupt condition. Interrupt handlers are initiated by hardware interrupts, software interrupt instructions, or software exceptions, and are used for implementing device drivers or transitions between protected modes of operation, such as system calls.

The traditional form of interrupt handler is the hardware interrupt handler. Hardware interrupts arise from electrical conditions or low-level protocols implemented in digital logic, are usually dispatched via a hard-coded table of interrupt vectors, asynchronously to the normal execution stream (as interrupt masking levels permit), often using a separate stack, and automatically entering into a different execution context (privilege level) for the duration of the interrupt handler's execution. In general, hardware interrupts and their handlers are used to handle high-priority conditions that require the interruption of the current code the processor is executing.

Later it was found convenient for software to be able to trigger the same mechanism by means of a software interrupt (a form of synchronous interrupt). Rather than using a hard-coded interrupt dispatch table at the hardware level, software interrupts are often implemented at the operating system level as a form of callback function.

Interrupt handlers have a multitude of functions, which vary based on what triggered the interrupt and the speed at which the interrupt handler completes its task. For example, pressing a key on a computer keyboard, or moving the mouse, triggers interrupts that call interrupt handlers which read the key, or the mouse's position, and copy the associated information into the computer's memory.

An interrupt handler is a low-level counterpart of event handlers. However, interrupt handlers have an unusual execution context, many harsh constraints in time and space, and their intrinsically asynchronous nature makes them notoriously difficult to debug by standard practice (reproducible test cases generally don't exist), thus demanding a specialized skillset—an important subset of system programming—of software engineers who engage at the hardware interrupt layer.