Assignment No :-1

Computer Fundamentals & Architecture.

Q1] What is Computer?

→ Computer is an electronic device used to perform simple as well as complex operations with the help of an input output devices as well as CPU.

Q2] What are different components of Computer?

→ Components of Computer are:

1] Microprocessor (CPU) :-

It is considered as a decision taking device which accept the instruction from the program and it will execute that program.

-It only understand binary Language, binary Language represent as only 0 and 1.

2] Math- Coprocessor :-

It consider as a Coprocessor of Macroprocessor.

It responsible to execute mathematical problems.

It is a unit of macroprocessor.

3] Graphical Processing Unit (GPU) :-

It is a dedicated processor for graphical processing.

4] Storage Device :-

Storage devices are responsible to shared data.

It can be used to stored temperary or permanant data. It stored any kind of data.

Types of storage device.

- Primary Storage device
- Secondary Storage device.

5] Input Devices :-

Input devices are used to provide input to the

computer.
e.g. keyboard & Mouse.
keyboard is a standard input device
6] Output Devices :-
Output devices are used to generate output.
e.g. Console, Printer.
Console is a standard output device.
7] Motherboard :-
Motherboard also known as Printed Circuit Board
It used to Connect all components to each other.
If any component wants to communicate with each other then communication done by motherboard.
8] System Bus :-
It considered as a bunch of wire and used to connect the devices to mother board and depend on the uses.
Types of System Bus:-
- Address Bus
- Data Bus
- Control Bus

Q3] What is storage device and types of storage devices.
→ - Storage devices is hardware component. Used to stored data.
- It keep information either temporarily or permanently.
- Storage devices are essensial for saving files, applications.
- Types of storage devices can broadly categorized into two main types :

Primary storage , Secondary storage.

• Primary Storage Devices :-

Devices which are directly accessible to the microprocessor.

- It also known as main memory or volatile memory
- It used to stored data that the CPU needs quickly.
- It is temporary storage , it means the data is lost when the device is powered off.

- RAM (Random Access Memory) :-

- It is primary electrical storage device which is used to stored data temporary.
- It used for active processes , running programs , and data that the CPU needs to access quickly.
- Every program get loaded into the RAM for the execution purpose.

- ROM (Read Only Memory) :-

- ROM is primary electrical storage device.
- When data is permanantly written then it is known as ROM.
- Every ROM before becoming a ROM is considered as a ~~RM~~ RAM.

• Secondary Storage Device.

- Secondary storage also known as non-volatile storage.
- It is used for storage of data on a long-term basis.
- Unlike primary storage, the data is keep even when the device is powered off.
- Before ~~get~~ data get loaded into the secondary device

it initially loaded in a primary. It is compulsary and mandatory to shift the data for primary to secondary.

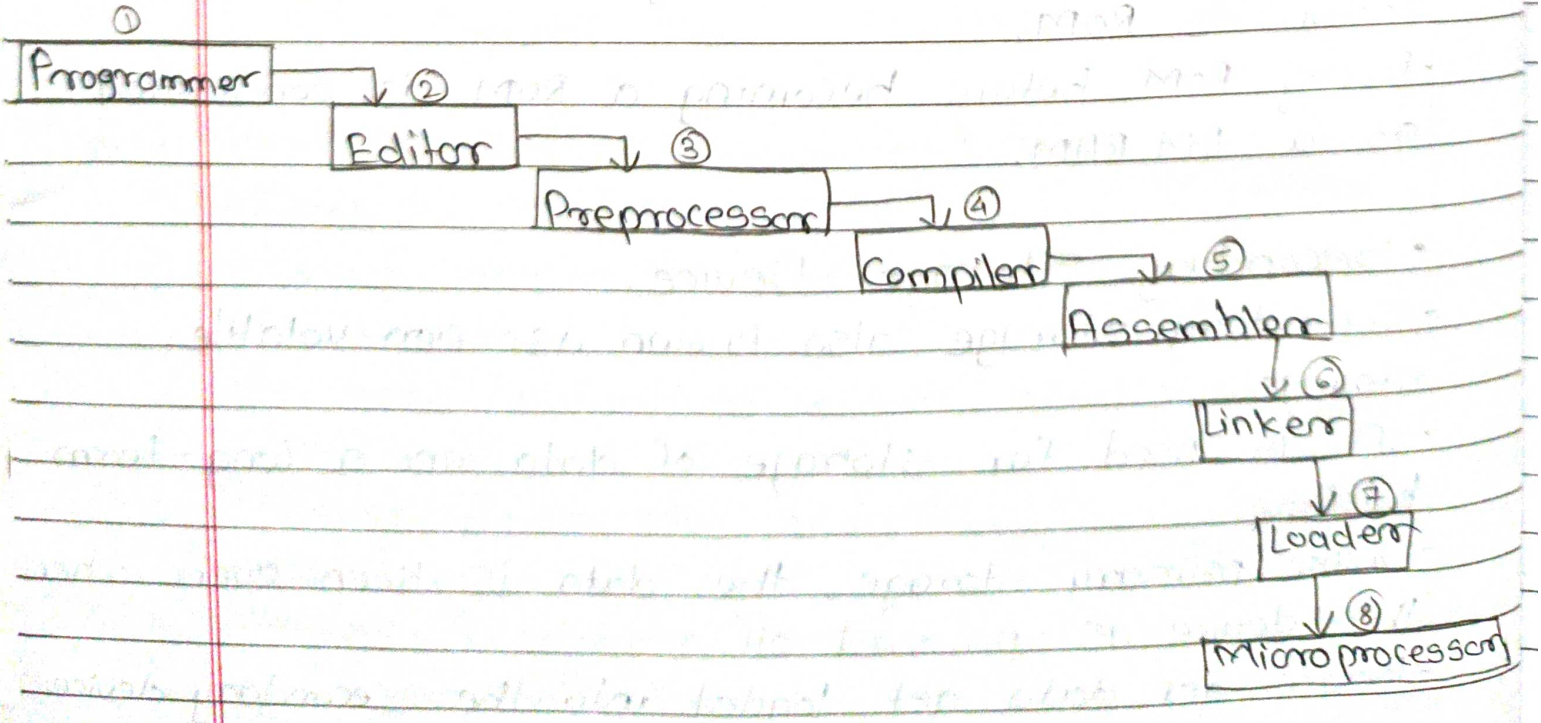
- Hard Disk Drive :-

It is a secondary storage device.

- Used to stored the data using magnetism.
- when any file is created and save it then file stored in hard disk.
- other examples of secondary storage devices are - Solid State Drive (SSD)
  - Optical Discs (CDs, DVDs)
  - USB Flash Drives
  - Memory Cards
  - Cloud Storage

Q.4] What is mean by Toolchain or x86 Toolchain?

→ Toolchain is considered as a set of software which are used to convert human understandable program into machine understandable program



• Step 1:-
- Programmer uses editor to write a program.
- After writting gets completed the file Demo.c get save inside the hdd Demo.c
- The content in Demo.c is human readable and understandable.

• Step 2 :-
- Preprocesser accept the input in Demo.c format and it generate the file which is the expanded version of Demo.c file.
- File created by preprocessor is Demo.i
- Content of Demo.i file are human readable and understandable.

• Step 3 :-
- Output of the processor is get input of compiler.
- Compiler is software which convert program one language to another language.
- In this case compiler convert program human understandable to machine dependent language which is Assembly language.
- File created by compiler having extension ~~of~~ as a .asm/.s
- Newly created file by compiler is Demo.s

• Step 4 :-
- Output of compiler get pass as a input of Assembler
- Assembler is a software of used to convert machine dependent to binary file, which binary file is not executable.
- Output of assembler is .obj file.
- .obj content the code in binary format but it is not directly executable.

• Step 5 :-
- Linker responsible to Link .obj file into executable format.
- Linker generate output with .exe extention.
- In this case .exe is output of Linker.
• Step 6 :-
- Demo.exe file is currently stored in hard disk drive.
- To execute any application or program which ~~is~~ has to be present in RAM.
Loader is responsible to load Demo.exe file to Hard disk drive to RAM.
- After loading Demo.exe file into RAM it consider as process. And it get executed with the help of OS.

Q5] What is CPU register? What are types of it and explain use of each CPU register.

→ - CPU Registers are small, fast storage locations within the central Processing Unit that hold data, instructions and addresses that are being used or processed by the CPU.
- Registers are crucial for the CPU's operation as they provide the faster access to data, enabling the CPU to perform calculations and execute instructions efficiently.
• Types of CPU Register
1] General Purpose Registers
These registers are used to store data temporarily during the execution of instructions.
- They can hold operands for operations, intermediate

results, and other data

ex - AH (Arithmatic, Auxilary Register)
BH (Base Register)
CH (Count Register)
DH (Data Register)

2] Segment Registers :-

These registers are used to store segment addresses in memory.

- They are crucial in memory segmentation, particularly in older x86 architectures, where the memory is divided into segments.

ex - CS (code Segment)
DS (Data Segment)
SS (Stack Segment)
ES (Extra Segment)

3] Special - Purpose Registers

- Instruction Pointer - Holds the address of the next instruction to be executed.
- Flags Register (Status Register)

Contains individual bits that are set or cleared based on the results of operations.

ex - ZF (zero flag)
CF (Carry Flag)
SF (Sign Flag)
OF (Overflow Flag)

4] Control Registers

- These register control operation of the CPU, managing tasks like memory management, interrupt handling, and switching between modes of operation

ex - CR0 to CR4
MRSRs (Model - Specific Registers)

5] Index and Pointer Registers
These registers are used to hold memory addresses or offsets for data in memory.
They are often used in operations involving arrays loops and string manipulation.
ex - SI (Source Index)
DI (Destination Index)
BP (Base Pointer)
SP (Stack Pointer)

Q6] What is Operating System & different tasks of O.S.?

→ - An operating system is a software layer that manages computer hardware and software resources and provides services for computer programs.
- It acts as an intermediary between users and the computer hardware.
The OS is essential for the functioning of a computer, as it manages the system's resources and ensures that different applications can run simultaneously without interfering with each other.
operating system perform five tasks.
1] File Management:
- The OS manages files on different storage devices by organizing them into directories.
- It ensures that files are accessed only by authorized users.
- OS handles operations such as reading, writing, moving files.

2] Memory Management:
- The OS allocate memory to different applications and processes, ensuring they do not interfere with each other.
- It frees up memory that is no longer in use, making it available for other tasks.
- OS can use disk space to simulate additional RAM, allowing more programs to run simultaneously than would otherwise be possible.

3] Process Management:
- OS is responsible for creating or terminating processes. The process is an instance of a running program.
- It determines the order in which processes access the CPU, ensuring efficient use of resources
- OS allows the multiple processes to run simultaneously by quickly switching between them giving the illusion ~~by~~ of parellelism.

4] CPU Scheduling:
- OS decides which process gets to use the CPU and for how long, based on scheduling algorithms.
- When the CPU switches from one process to another, the OS saves the states of current process and loads the state of the next process.

5] Hardware Abstraction:
- OS provides a layer of abstraction between the hardware and software through device drivers, which allow software to interact with hardware without needing to know the hardware's specifics.

Q7] Explain working of each tool from Toolchain i.e. Editor, Preprocessor, Compiler, Assembler, Linker, Loader.

→ Toolchain is considered as a set of software (tools) which are used to convert humanunderstandable program into machine understandable program (binary language).

1] Editor:

Editor is software application used by programmer to write and edit source data.

- Editor provides a text-based interface where the programmer can write code in high level programming language.
- The output of editor is a plain text file, containing source code, usually with a specific file extension (Demo.c).

2] Preprocessor:

- Preprocessor processes the source code before it is compiled.
- Preprocessor handles directives in the source code that begin with # such as #include, #define.
- It includes the content of header files into the source code.
- File created by preprocessor is demo.i

3] Compiler:

Compiler converts the preprocessed source code into machine code.

- Compiler generates machine code specific to the target CPU architecture. The output is an object file (Demo.obj / Demo.o) which contains the

machine code but may be incomplete for execution.

4] Assembler:

-The assembler converts assembly code (generated by compiler) into machine code.

-Assembly code is low-level code that closely represents the machine instructions.

-Assembler translate the assembly code into binary machine code that the CPU can execute.

5] Linker:

Linker combines various object files and libraries into a single executable file.

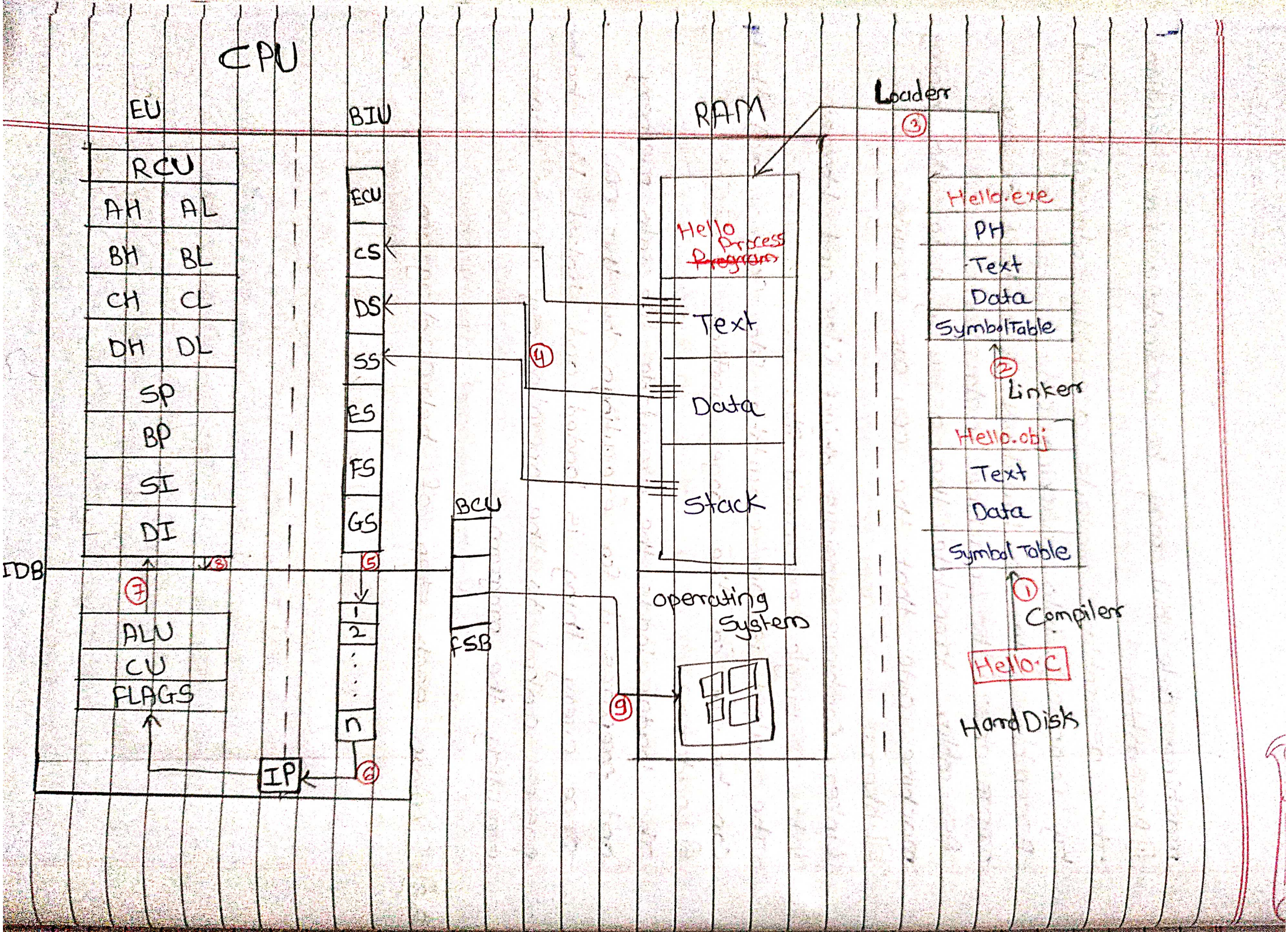
-The output is an executable file (Demo.exe) that can be run on the target machine.

6] Loader:

The loader loads the executable file into memory and prepares it for execution by the CPU

-Once every thing is loaded and linked the loader passes control to the operating system to start executing the program.

Q8] Explain each step of below diagram.


CPU
EU
RCU
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
IDB
ALU
CU
FLAGS
BIU
ECU
CS
DS
SS
ES
FS
GS
1
2
n
IP
BCU
FSB
RAM
Hello Process
Text
Data
Stack
operating System
Loader
Hello.exe
PH
Text
Data
SymbolTable
Linker
Hello.obj
Text
Data
Symbol Table
Compiler
Hello.C
Hard Disk
1
2
3
4
5
6
7
8
9

The diagram provides illustrates the process of converting a high-level source code file into a running process in memory.

Step I:
The process begins with the source code file 'Hello.c', stored on the hard disk.
- The source code file (Hello.c) is fed into the compiler.
- The compiler translates the high-level language into an object file (Hello.obj), which contains machine code, a symbol table and other necessary information.

Step II:
The object file is passed to the linker.
- The linker combines this object file with other obj file and libraries to create an executable file.
- The executable file also contains the machine code. But it is not complete and ready to be loaded into memory.

Step III:
The loader loads the executable file (Hello.exe) into the system's memory (RAM).
- The loader sets up the process, including its code, initialized data and stack segments in memory.

Step IV:
Dia shows process layout in RAM:

• Text Segment:
Contains the executable code (instructions).
• Data Segment:
Contains initialized global and static variables.
• Stack Segment:
Used for function call management, including local variables, return addresses and function parameters.

• CPU (central Processing Unit):
CPU is divided into two main units:
- Execution Unit (EU): Responsible for execution instructions.
- RCU (Register Control Unit): Holds general-purpose registers like AX, Bx, Cx, etc.
Used for arithmatic, logic operations and data movement.
- ALU (Arithmatic Logic Unit): Perform arithmatic and logic operations.
- CU (control Unit): Directs the operation of the processor.
- FLAGS: Indicates the status of the CPU and the results of operations.
- Bus Interface Unit (BIU): Manages the communication between the CPU and the memory or I/o devices.
- ECU (Execution Control Unit): Manages segment registers (DS, CS, SS, etc) which are used for addressing Unit.
- IP (Instruction Pointers): Points to the next instruction to be executed.

Step V:
The BIU fetches instructions from memory and passes them to the FU.

Step VI:
The IP is Updated as the instructions are fetched and executed.

Step VII:
The FU uses the RCU registers to perform operations, and the results

Step VIII:
Written back to memory or used within the CPU.
- Communication between the CPU and memory is facilitated through the Bus Control Unit (BCU) and Front side Bus (FSB).

Step IX:
operating system is responsible for managing the execution of the process, allocation memory, and handling system calls and I/O operations.

39] what are the contents of Physical Header?
→ The physical Header is a part of the executable file format, such as the executable and Linkable format (ELF) used in Unix-like operating systems, or the Portable Executable (PE) format used in Windows.
- The physical header provides essensial information that the os and loader need to load and run the program.

• Contents of Physical Header:

1] Magic Number:
A unique identifier that indicates the file type such as '0x7F ELF' for ELF files.

2] File Type:
Specifies the type of file, such as executable, shared object or relocatable.

3] Entry Point:
The memory address where the CPU should start executing the program.

4] Program header table:
Contains information about the segments in the executable, including their locations in the file and where they should loaded in memory.

5] Section Header Table:
A table that describes the sections of the executable file, such as .text, .data, .bss etc.

6] Segment Descriptors:
Segment Descriptors includes-
- Segment Type
- Segment offset
- Segment Virtual Address
- Segment Physical Address
- Segment Size in File
- Segment size in Memory
- Flags.
- Alignment.

7] Flags:
Various flags that provide additional information, such as whether the file is dynamically linked or position-independent.

8] Loadable Segments:
The program header contains information about the segments that need to be loaded into memory including their file offset, memory size and permissions.

9] Interpreter Path:
Specifies the path to the interpreter that should be used to load the program.

10] Auxiliary Data:
-Additional metadata that might be required for the proper loading and execution of the program. Such as info on dynamic linking threading.

20] What is mean by Text, Data and stack section?
→ The terms 'text', 'Data, 'stack' sections typically refer to segments of memory in the context of a computer program or process.
-These sections are part of how a program's memory is organized when it is executed.

1] Text Section (Code Segment):
It contains the compiled code of a program, meaning the actual machine instructions that the CPU executes.
-This section is usually read-only to prevent accidental or malicious modification of the code during execution.

2] Data Section:
The data section holds the global and static variables that are explicitly initialized in the source code.

3) Stack Section:

The stack section is used for managing function calls and local variables.

When a function is called, a new stack frame is created to store the function's local variables, return address, and other control information.

- The stack grows and shrinks as functions are called and return.

This section is essential for managing program execution flow and local variable storage.