Lesson Plan

Classification of Programming Languages

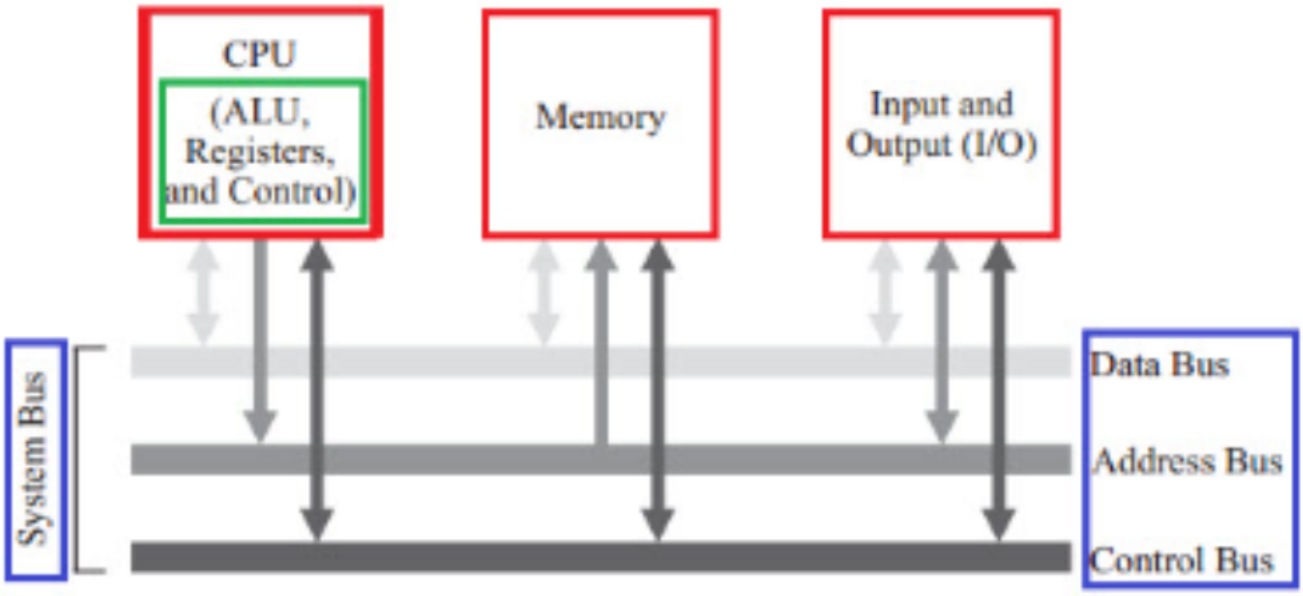
PW SKILLS

List of Concepts Involved:

- Fundamentals of Microprocessor
- MLL Vs ALL Vs HLL
- Assembler, Compiler & Interpreter
- Programming Paradigms
- History of Popular HLLs
- Features of Java
- Versions of Java

Fundamentals of Microprocessor:

The processor on a single chip is called a Microprocessor which can process micro-instructions. Instructions in the form of 0s and 1s are called micro-instructions. The microprocessor is the CPU part of a microcomputer, and it is also available as a single integrated circuit. Thus as main components, the microprocessor will have the Control Unit (CU) and the Arithmetic Logic Unit (ALU) of a microcomputer. An example is Intel 8085 microprocessor.

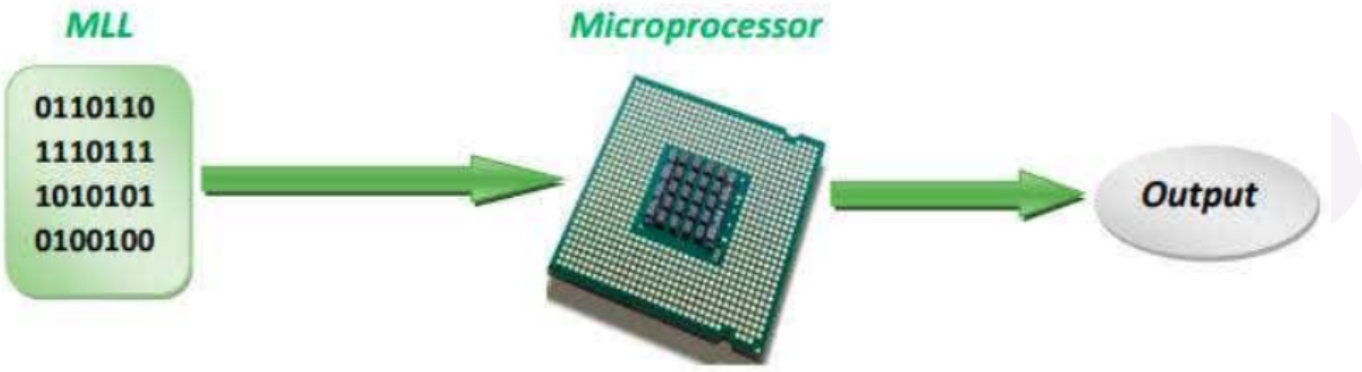


The microprocessor follows a sequence: Fetch, Decode, and then Execute. Initially, the instructions are stored in the memory in sequential order. The microprocessor fetches those instructions from the memory, then decodes it and executes those instructions till STOP instruction is reached. Later, it sends the result in binary to the output port. Between these processes, the register stores the temporary data and ALU performs the computing functions.

MLL Vs ALL Vs HLL:

- **Machine Level Language (MLL):**

- Machine language is only understood by computers.
- In machine language data is only represented with the help of binary format(0s and 1s), hexadecimal and octal decimal.
- Machine language is very difficult to understand by human beings.
- Modifications and error fixing cannot be done in machine language.
- Machine language is very difficult to memorize so it is not possible to learn the machine language.
- Execution is fast in machine language because all data is already present in binary format.
- There is no need for a translator.The machine understandable form is the machine language.
- Machine language is hardware dependent.

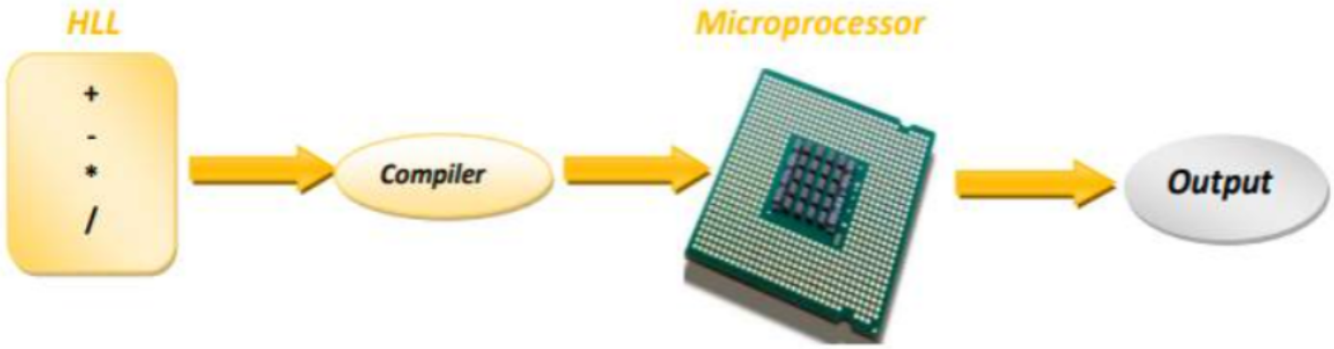


- **Assembly Level Language (ALL) & Assembler:**

- Assembly language is only understood by human beings, not by computers.
- In assembly language data can be represented with the help of mnemonics such as Mov, Add, Sub, End etc.
- Assembly language is easy to understand by human being as compared to machine language.
- Modifications and error fixing can be done in assembly language.
- Easy to memorize the assembly language because some alphabets and mnemonics are used.
- Execution is slow as compared to machine language.
- Assembler is used as a translator to convert mnemonics into machine understandable form.
- Assembly language is machine dependent and it is not portable.

- **High Level Language (HLL):**

- It needs a compiler/interpreter for conversion
- In this, we convert a high-level language to Assembly level language to machine level language
- It is machine-independent
- In this English statement is used
- In this, it is difficult to access hardware component

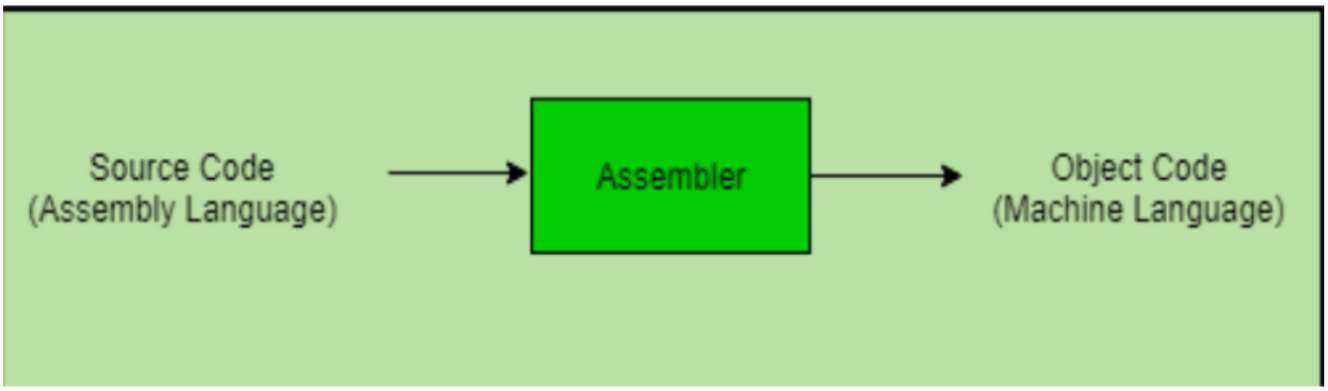


- **Assembler, Compiler & Interpreter Assembler:**

- An Assembler is a special tool that translates programs written in Assembly language into machine code, which the computer can understand.
- In the source program, we write instructions using simple mnemonics like ADD, MUL, SUB, etc.
- The Assembler acts as the first link between humans and machines, helping them communicate.
- It takes the program written in Assembly language as input and produces the machine code as output.
- The machine code consists of 0s and 1s, and it depends on the specific computer's architecture.
- The Assembler converts these mnemonics (instructions) into the corresponding binary code, bridging the gap between human-readable instructions and computer-executable instructions.

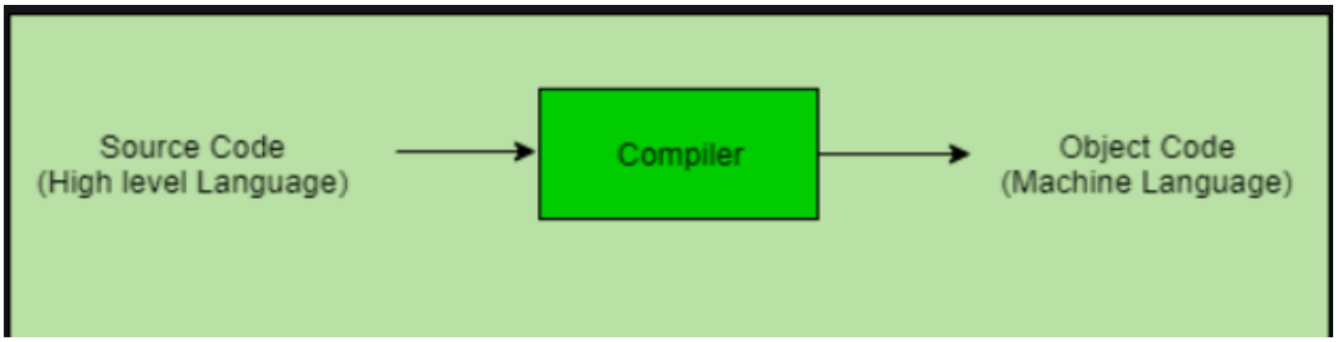
- **High Level Language (HLL):**

- It needs a compiler/interpreter for conversion
- In this, we convert a high-level language to Assembly level language to machine level language
- It is machine-independent
- In this English statement is used
- In this, it is difficult to access hardware component



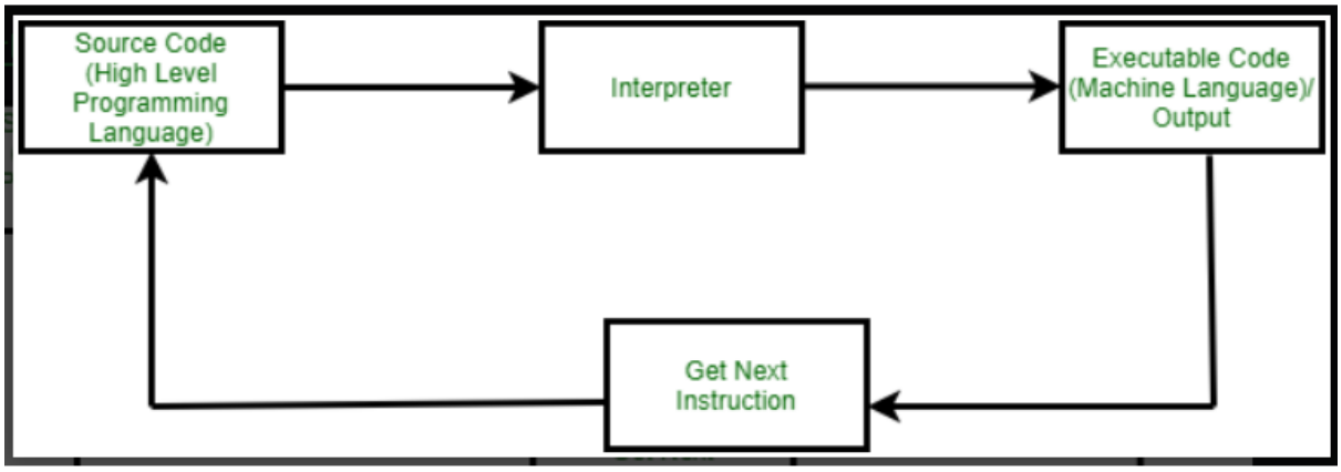
- **Compiler:**

- A Compiler is a type of language processor that takes the entire program written in a high-level language (like C, C++, Java, C#) and translates it all at once into machine language.
- It's like converting the whole code into a language that the computer can understand.
- If the source code is free of errors, the compiler can successfully translate it into an object code.
- However, if there are any errors in the source code, the compiler will point them out at the end of the process, indicating the line numbers where the errors occur.
- To get the program to run smoothly, these errors must be fixed before recompiling.
- Once the code is error-free and successfully compiled into object code, it can be executed multiple times without needing to translate it again.



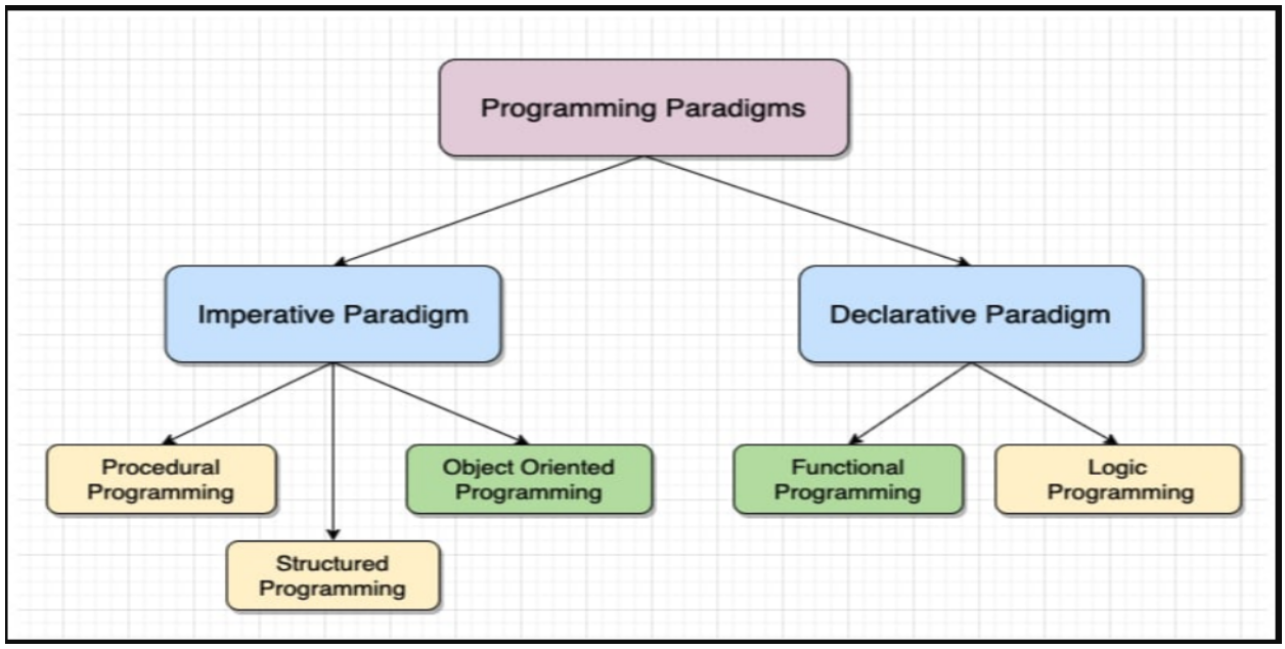
- **Interpreter:**

- An Interpreter will run line by line and further translate each line accordingly after taking the source program
- In this Processor, debugging is much easy since it translates until the error is found
- It needs less memory than the compiler
- No object code is generated.
- A compiler is more useful for the security purpose.
- Examples: Python, Perl, JavaScript and Ruby



- **Programming Paradigms**

A paradigm is also a way of approaching a problem or carrying out a task.
A programming paradigm is a way of approaching problems with programming languages, or alternatively, It is a way of approaching problems with the tools and techniques at our disposal while adhering to a certain approach.
There are many well-known programming languages, but when they are used, they all need to adhere to a technique or strategy, and this approach is known as paradigms.
In addition to several programming languages, there are numerous paradigms to meet every need.



Imperative programming paradigm: It is one of the oldest programming paradigm. It features close relation to machine architecture. It is based on Von Neumann architecture. It works by changing the program state through assignment statements. It performs step by step tasks by changing state. The main focus is on how to achieve the goal. The paradigm consists of several statements and after execution all the results are stored.

Examples of Imperative programming paradigm:
C : developed by Dennis Ritchie and Ken Thompson
Fortran : developed by John Backus for IBM
Basic : developed by John G Kemeny and Thomas E Kurtz

Procedural programming paradigm –
This paradigm emphasizes procedure in terms of the underlying machine model. There is no difference in between procedural and imperative approaches. It has the ability to reuse the code and it was boon at that time when it was in use because of its reusability.

Examples of Procedural programming paradigm:
C : developed by Dennis Ritchie and Ken Thompson
Pascal : developed by Niklaus Wirth

Examples of Object Oriented programming paradigm:
Simula : first OOP language
Java : developed by James Gosling at Sun Microsystems
C++ : developed by Bjarne Stroustrup
Objective-C : designed by Brad Cox
Visual Basic .NET : developed by Microsoft
Python : developed by Guido van Rossum
Ruby : developed by Yukihiro Matsumoto
Smalltalk : developed by Alan Kay, Dan Ingalls, Adele Goldber

History of Popular HLLs

High-level languages are programming languages that are designed to allow humans to write computer programs and interact with a computer system without having to have specific knowledge of the processor or hardware that the program will run on.
High-level languages use command words and Syntax which reflects everyday language, which makes them easier to learn and use. High-level languages also offer the programmer development tools such as libraries and built-in functions.

Many types of high-level language exist and are in common use today, including:

- **C**
- **Java**
- **Python**
- **C++**
- **C#**
- **JavaScript, etc.**

Features of Java:

- ## Object Oriented:

  In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

- ## Platform Independent

  Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into a platform specific machine, rather into platform-independent bytecode. This byte code is distributed over the web and interpreted by the Java Virtual Machine (JVM) on whichever platform it is being run on.

- ## Simple

  Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

- **Secure**

  With Java's secure feature it enables the development of virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

- **Architecture-neutral**

  Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

- **Portable**

  Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. The compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.

- **Robust**

  Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.

- **Multithreaded**

  With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

- **Interpreted**

  Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.

- **High Performance**

  With the use of Just-In-Time compilers, Java enables high performance.

- **Distributed**

  Java is designed for the distributed environment of the internet.

- **Dynamic**

  Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects at run-time.

Versions of Java:

There are many java versions that have been released.

- JDK Alpha and Beta (1995)
- JDK 1.0 (23rd Jan, 1996)
- JDK 1.1 (19th Feb, 1997)
- J2SE 1.2 (8th Dec, 1998)
- J2SE 1.3 (8th May, 2000)
- J2SE 1.4 (6th Feb, 2002)
- J2SE 5.0 (30th Sep, 2004)
- Java SE 6 (11th Dec, 2006)
- Java SE 7 (28th July, 2011)
- .....