When we talk about programming languages the first thing that comes to our mind is languages like C,

C++, Java, Python etc. But those languages hide the actual working i.e., that abstracts many things from

users. But there is a language that really lies on basic concepts behind the programming or interaction

between computer hardware.

What is Assembly Language?

[Assembly language](https://www.geeksforgeeks.org/difference-between-machine-language-and-assembly-language/) is a low-level language that helps to communicate directly with computer hardware.

It uses mnemonics to represent the operations that a processor has to do. Which is an intermediate

language between high-level languages like [C++](https://www.geeksforgeeks.org/introduction-to-c-programming-language/) and the binary language. It uses hexadecimal and binary

values, and it is readable by humans.

How Assembly Language Works?

Assembly languages contain mnemonic codes that specify what the processor should do. The mnemonic

code that was written by the programmer was converted into machine language (binary language) for

execution. An assembler is used to convert assembly code into machine language. That machine code is

stored in an executable file for the sake of execution.

It enables the programmer to communicate directly with the hardware such as registers, memory

locations, [input/output devices](https://www.geeksforgeeks.org/input-and-output-devices/) or any other [hardware](https://www.geeksforgeeks.org/computer-hardware/)components. Which could help the programmer

to directly control hardware components and to manage the resources in an efficient manner.

How to execute Assembly Language?

**Write assembly code**: Open any text editor in device and write the mnemonic codes in it and save the

file with a proper extension according to your assembler. Extension can be *.asm*,*.s*, *.asm*x.

**Assembling the code**: Convert your code to machine language using an [assembler](https://www.geeksforgeeks.org/introduction-of-assembler/).

**Generating object file**: It will generate an object file corresponding to your code. It will have an

extension .*obj*.

**Linking and creating executables**: Our assembly language might contain multiple source codes. And we

have to link them to libraries to make it executable. We can use a linker like lk for this purpose.

**Running program**: After creating an executable file we can run it as usual. It will depend on the software

that how to run the program.

Components of Assembly Language

**Registers:** Registers are the fast memory locations situated inside the processor. Which helps [ALU](https://www.geeksforgeeks.org/introduction-of-alu-and-data-path/) to

perform arithmetic operations and temporary storing of data. Example: Ax (Accumulator), Bx, Cx.

**Command:** An instruction in assembly code known as a command informs the assembler what to do.

Assembly language instructions typically employ self-descriptive abbreviations to make the vocabulary

simple, as “ADD” for addition and “MOV” for data movement.

**Instructions:** Instructions are the mnemonic codes that we give to the processor to perform specific

tasks like LOAD, ADDITION, MOVE. Example: ADD

**Labels:** It is a symbolic name/identifier given to indicate a particular location or address in the assembly

code. Example: FIRST to indicate starting of execution part of code.

**Mnemonic:** A mnemonic is an acronym for an assembly language instruction or a name given to a

machine function. Each mnemonic in assembly corresponds to a specific machine instruction. Add is an

illustration of one of these machine commands. CMP, Mul, and Lea are among further instances.

**Macro:** Macros are the program codes that can be used anywhere in the program through calling it once

we define it. And it is often embedded with assemblers and compilers. We should define it using a

directive %macro. Example: %macro ADD\_TWO\_NUMBERS 2

add eax, %1

add eax, %2

%endmacro

**Operands:** These are the data or values that we are given through instruction to perform some

operation on it. Example: In ADD R1,R2 ; R1 and R2 are operands.

**Opcode:** These are the mnemonic codes that specify to the processor which operation has to be done.

Example: ADD means Addition.

