**Low Level Programming**

A low-level programming language is a programming language that provides little or no abstraction from a computer's instruction set architecture—commands or functions in the language map closely to processor instructions. Generally this refers to either machine code or assembly language.

**Assembly Language**

Assembly language is a low-level programming language for a computer or other programmable device specific to a particular computer architecture in contrast to most high-level programming languages, which are generally portable across multiple systems. Assembly language is converted into executable machine code by a utility program referred to as an assembler like NASM, MASM, etc.

**What is Assembly Language?**  
Each personal computer has a microprocessor that manages the computer's arithmetical, logical, and control activities.  
  
Each family of processors has its own set of instructions for handling various operations such as getting input from keyboard, displaying information on screen and performing various other jobs. These set of instructions are called 'machine language instructions'.  
  
A processor understands only machine language instructions, which are strings of 1's and 0's. However, machine language is too obscure and complex for using in software development. So, the low-level assembly language is designed for a specific family of processors that represents various instructions in symbolic code and a more understandable form.  
  
**Advantages of Assembly Language**  
Having an understanding of assembly language makes one aware of −

* How programs interface with OS, processor, and BIOS;
* How data is represented in memory and other external devices;
* How the processor accesses and executes instruction;
* How instructions access and process data;
* How a program accesses external devices.

**Other advantages of using assembly language are −**

* It requires less memory and execution time;
* It allows hardware-specific complex jobs in an easier way;
* It is suitable for time-critical jobs;
* It is most suitable for writing interrupt service routines and other memory resident programs.

**There are many good assembler programs, such as −**

* Microsoft Assembler (MASM)
* Borland Turbo Assembler (TASM)
* The GNU assembler (GAS)

**Assembly language statement examples**

* INC COUNT ; Increment the memory variable COUNT
* MOV TOTAL, 48 ; Transfer the value 48 in the
* ; memory variable TOTAL
* ADD AH, BH ; Add the content of the
* ; BH register into the AH register
* AND MASK1, 128 ; Perform AND operation on the
* ; variable MASK1 and 128
* ADD MARKS, 10 ; Add 10 to the variable MARKS
* MOV AL, 10 ; Transfer the value 10 to the AL register

**Hello World Program Example**

The following assembly language code displays the string 'Hello World' on the screen −  
  
section .text  
 global \_start ;must be declared for linker (ld)  
   
\_start: ;tells linker entry point  
 mov edx,len ;message length  
 mov ecx,msg ;message to write  
 mov ebx,1 ;file descriptor (stdout)  
 mov eax,4 ;system call number (sys\_write)  
 int 0x80 ;call kernel  
   
 mov eax,1 ;system call number (sys\_exit)  
 int 0x80 ;call kernel  
  
section .data  
msg db 'Hello, world!', 0xa ;string to be printed  
len equ $ - msg ;length of the string