Second Generation Languages 2GLs (Assembly Language)

Assembly language is a low-level programming language for a computer or other programmable device specific to 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.

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 like getting input from keyboard, displaying information on screen and performing various other jobs. These set of instructions are called 'machine language instructions'.

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

An understanding of assembly language provides knowledge of:

- Interface of programs with OS, processor and BIOS;
- Representation of data in memory and other external devices;
- How 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.

Assembly Language Statements

Assembly language programs consist of three types of statements:

- Executable instructions or instructions
- Assembler directives or pseudo-ops
- Macros

The **executable instructions** or simply **instructions** tell the processor what to do. Each instruction consists of an operation code (opcode). Each executable instruction generates one machine language instruction.

The **assembler directives** or **pseudo-ops** tell the assembler about the various aspects of the assembly process. These are non-executable and do not generate machine language instructions.

Macros are basically a text substitution mechanism.

Syntax of Assembly Language Statements

Assembly language statements are entered one statement per line. Each statement follows the following format:

[label] mnemonic [operands] [;comment]

The fields in the square brackets are optional. A basic instruction has two parts, the first one is the name of the instruction (or the mnemonic), which is to be executed, and the second are the operands or the parameters of the command.

Following are some examples of typical assembly language statements:

```
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
```