**Assembly Language**

**introduction**

## 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.

**Definition :**

 **What does *Assembly Language* mean?**

An assembly language is a low-level programming language for microprocessors and other programmable devices. It is not just a single language, but rather a group of languages. An assembly language implements a symbolic representation of the machine code needed to program a given CPU architecture.

Assembly language is also known as assembly code. The term is often also used synonymously with 2GL.

## 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.

### Disadvantages Of Assembly Language

1. Long programs written in such languages cannot be executed on small sized computers.
2. It takes lot of time to code or write the program, as it is more complex in nature.
3. Difficult to remember the syntax.
4. Lack of portability of program between computers of different makes.
5. No SDKs (System Development Kit).

# Conclusion:

You never really learn assembly language.

You can improve your skills over time, by reading good books on the subject, by reading good code that others have written, and, most of all, by writing lots and lots of code yourself. But at no point will you be able to stand up and say, "I know it."

You shouldn't feel bad about this. In fact, I take some encouragement from occasionally hearing that Michael Abrash, author of Zen of Assembly Language, Zen of Code Optimization, and his giant compendium Michael Abrash's Graphics Programming Black Book, has learned something new about assembly language. Michael has been writing high-performance assembly code for almost 30 years, and has become one of the two or three best assembly language programmers in the Western hemisphere.

If Michael is still learning, is there hope for the rest of us?

Wrong question. Silly question. If Michael is still learning, it means that all of us are students and will always be students. It means that the journey is the goal, and as long as we continue to probe and hack and fiddle and try things that we never tried before, over time we will advance the state of the art and create programs that would have made the pioneers in our field catch their breath in 1977.