Table of Contents

Programming Language

Assembly

Assembly *(programminglanguage)*	
Full Name	Assembly
Short Name	Assembly
Description	low-level programming language
Company	Unkown
Web	No
Mobile	No
Enterprise	No
Embedded	Yes

What is Assembly language?

Assembly language is a low-level programming language that uses a series of mnemonics to represent instructions that can be executed directly by the processor of a computer. It is one step above machine language, which is a binary code that can be executed directly by the computer.

Why use Assembly language?

Assembly language can be used to write code that is very fast and efficient, as it can directly access hardware resources and memory addresses. It is often used in applications that require real-time processing, such as operating systems, device drivers, and embedded systems.

What are some common Assembly languages?

Some common Assembly languages include x86 (used by Intel and AMD processors), ARM (used in many mobile devices), and MIPS (used in many embedded systems).

Is Assembly language difficult to learn?

Assembly language can be difficult to learn, as it requires a good understanding of computer architecture and low-level programming concepts. However, with practice and patience, it can be a powerful tool for writing efficient code.

Can Assembly language be used with modern programming languages?

Yes, Assembly language can be used in conjunction with higher-level programming languages, such as C and C++. This is often done to optimize certain portions of code that require fast execution or direct access to hardware resources.

Are there any drawbacks to using Assembly language?

One of the main drawbacks of Assembly language is that it can be time-consuming to write and debug compared to higher-level programming languages. Additionally, code written in Assembly language can be less portable between different hardware architectures.

*Snippet from Wikipedia: **Assembly language***

> In computer programming, **assembly language** (alternatively **assembler language** or **symbolic machine code**), often referred to simply as **assembly** and commonly abbreviated as **ASM** or **asm**, is any low-level programming language with a very strong correspondence between the instructions in the language and the architecture's machine code instructions. Assembly language usually has one statement per machine instruction (1:1), but constants, comments, assembler directives, symbolic labels of, e.g., memory locations, registers, and macros are generally also supported.
>
> The first assembly code in which a language is used to represent machine code instructions is found in Kathleen and Andrew Donald Booth's 1947 work, *Coding for A.R.C.*. Assembly code is converted into executable machine code by a utility program referred to as an *assembler*. The term "assembler" is generally attributed to Wilkes, Wheeler and Gill in their 1951 book *The Preparation of Programs for an Electronic Digital Computer*, who, however, used the term to mean "a program that assembles another program consisting of several sections into a single program". The conversion process is referred to as *assembly*, as in *assembling* the source code. The computational step when an assembler is processing a program is called *assembly time*.
>
> Because assembly depends on the machine code instructions, each assembly language is specific to a particular computer architecture.
>
> Sometimes there is more than one assembler for the same architecture, and sometimes an assembler is specific to an operating system or to particular operating systems. Most assembly languages do not provide specific syntax for operating system calls, and most assembly languages can be used universally with any operating system, as the language provides access to all the real capabilities of the processor, upon which all system call mechanisms ultimately rest. In contrast to assembly languages, most high-level programming languages are generally portable across multiple architectures but require interpreting or compiling, much more complicated tasks than assembling.
>
> In the first decades of computing, it was commonplace for both systems programming and application programming to take place entirely in assembly language. While still irreplaceable for some purposes, the majority of programming is now conducted in higher-level interpreted and compiled languages. In "No Silver Bullet", Fred Brooks summarised the effects of the switch away from assembly language programming: "Surely the most powerful stroke for software productivity, reliability, and simplicity has been the progressive use of high-level languages for programming. Most observers credit that development with at least a factor of five in productivity, and with concomitant gains in reliability, simplicity, and comprehensibility."
>
> Today, it is typical to use small amounts of assembly language code within larger systems

> implemented in a higher-level language, for performance reasons or to interact directly with hardware in ways unsupported by the higher-level language. For instance, just under 2% of version 4.9 of the Linux kernel source code is written in assembly; more than 97% is written in C.

Hello World

```
bdos    equ    0005H    ; BDOS entry point
start:  mvi    c,9      ; BDOS function: output string
        lxi    d,msg$   ; address of msg
        call   bdos
        ret             ; return to CCP

msg$:   db    'Hello, world!$'
end     start
```

ToDo

- Registers
- Memory addressing modes
- Interrupts
- Input/output operations
- Arithmetic instructions
- Logic instructions
- Control flow instructions
- Macros
- Subroutines
- Stack operations
- Bit manipulation
- Assembly directives
- Conditional branching
- Assembly language syntax
- Assembly language programming principles
- CPU architecture
- Assembly language optimization
- Debugging assembly code
- Inline assembly
- Assembly language frameworks

See also: Programming Languages

programminglanguage