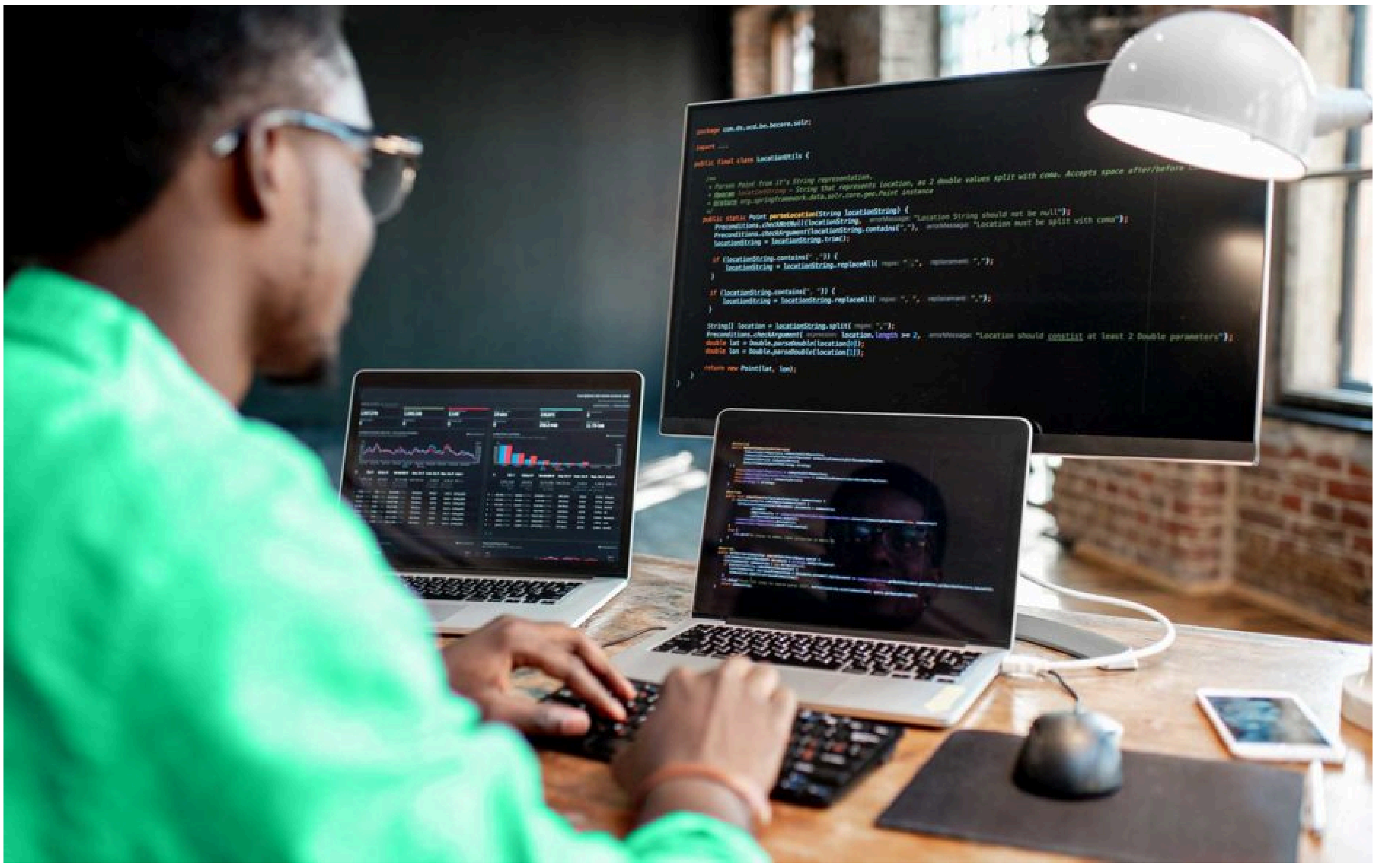


May 3, 2022 0

What Is Assembly Language? A Quick Overview

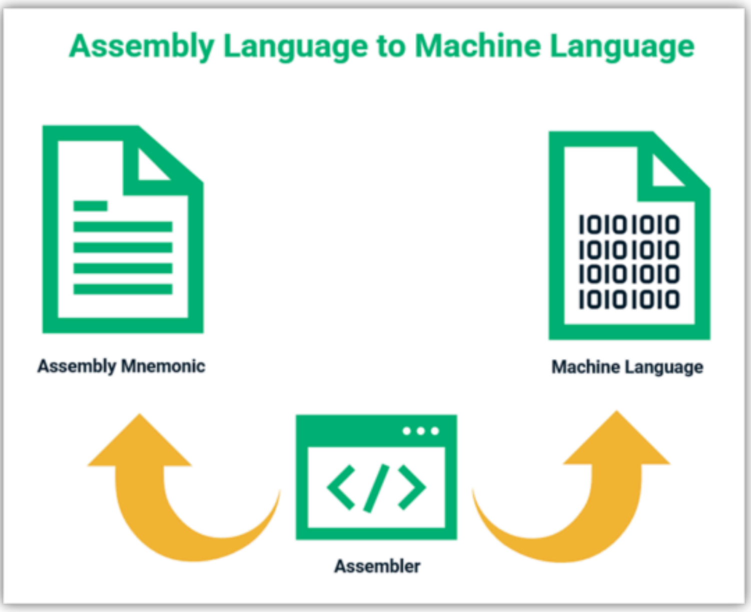
in **OTHER**

(**14** votes, average: **5.00** out of 5)

If you want to know how computers work and avoid a few programming pitfalls, it’s helpful to learn the basics. So, let’s take a quick look at assembly language and a few of the benefits of understanding it.

If you're a software developer, you're probably aware that there are multiple types of computer languages that developers and systems use. Computers understand machine-level language, while humans use higher-level languages like JavaScript, C++, etc. But there's a language that falls in the middle that helps bridge the divide between these two entities called assembly language. But what is an assembly language and where does it fit into the programming hierarchy?

What Is Assembly Language? A Look at Machine-Specific Programming



A basic graphic that illustrates how assemblers convert assembly language into machine language.

Assembly language, also known as assembler language, is a low-level programming language that's designed to communicate instructions with specific computer hardware and direct the flow of information. It does this using human-readable mnemonics (consisting of mnemonics like "LDA" to represent *load accumulator*) to form short code that makes it easier for the person trying to complete the work. These short codes are converted into machine learning language (binary, i.e., 1s and 0s) through the use of programs called assemblers.

In a nutshell, machine language uses binary code, which is almost impossible for humans to decipher, whereas assembly language uses mnemonic codes to write a program. Mnemonic codes make it simpler for humans to understand or remember something, and so make the language a bit easier for humans to use than machine code.

Assembly is language is nothing new; it's been around almost as long as computers themselves. The first assembly language was invented in 1947 by mathematician Kathleen Booth at Birkbeck College, London, with assistance from Andrew Booth (her husband), Jon von Neumann, and Herman Goldstein.

Is Assembly Commonly Used by Developers?

No way. The truth is that most developers don't use assembly language anymore because it's often regarded as being cumbersome and challenging to use. This is so much the case that assembly language has basically gone the way of Javan Rhinoceroses — it's virtually extinct as it's something most professionals nowadays never learn. (In fact, many colleges only offer it as an elective course [if they offer it at all]!)

There are other languages that are easier to use — namely high-level languages, which we'll speak more about momentarily. And many of these high-level languages bypass the need for knowing assembly language altogether because they convert a developer's written code to a standard intermediate language on the backend.

Occasionally, you'll find the rare hardcore developer who still uses assembly language. But you might want to snap a picture when you see it because it's truly a rare occurrence.

Assembly Language Is One of Multiple Types of Computer Languages

To understand the concept of assembly language, it helps to take a holistic approach and learn about all three types of computer language. It's important to note that some people say there are three main groups of languages, some say four, some say even more than that. The answer varies depending on whom you ask and how they categorize these languages.

1. Machine Language

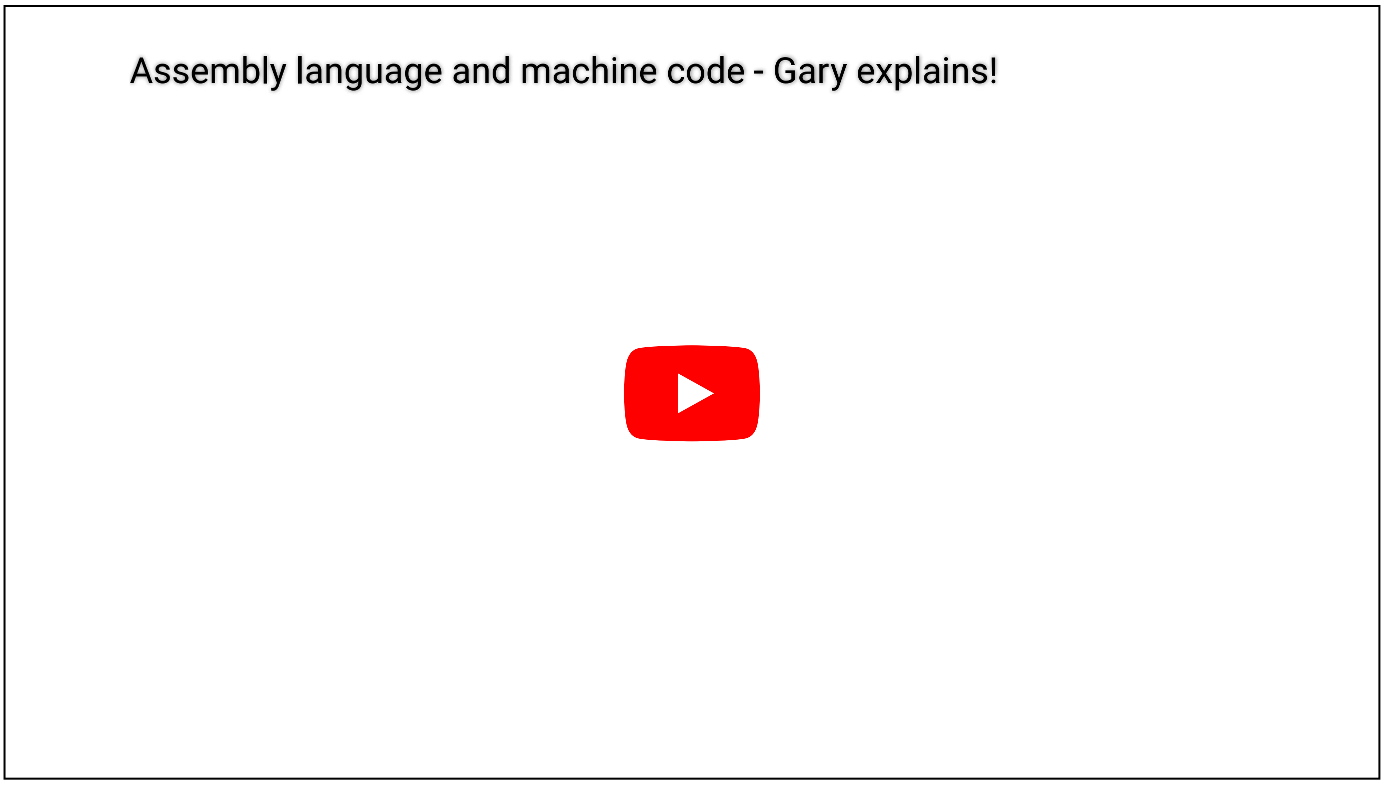
Every device has a processor responsible for all the functions performed by that device. These processors use machine language to "talk" to the different parts of the device, for example, giving and receiving instructions to/from the keyboard and mouse. As we touched on earlier, machine language is written in binary form (0s and 1s) in a hexadecimal format and each family of processors has its own version. Although processors speak machine language fluently, it's extremely challenging for a human to read or use — and virtually impossible for someone to do so quickly. That's where assembly language comes in.

2. Assembly Languages

In general, assembly language is a bit more user-friendly than machine-level language but more difficult than high-level language. It uses short codes to instruct the machine to perform certain operations. Whereas machine language uses 0s and 1s to instruct the computer, assembly language uses mnemonics that are easier for humans to work with.

As we already know, processors only speak machine language. So, to translate the machine code into assembly language and vice versa, a program called an assembler is required. The translation process is known as assembling, and the time required to translate the language is called assembling time. As different families of processors use different machine codes, the assembly language for each family is also different. Some assembly languages work across different operating systems, whereas others are specific to one OS or platform.

Here’s a quick look at what assembly language looks and how it translates to machine code via the assembler:



3. Middle-Level Languages (i.e., Common Intermediate Language)

As we mentioned earlier, developers typically don’t use assembly languages anymore. Often, developers use high-level languages that are then compiled into intermediate language (IL) that gets translated into machine language code. Basically, this means that it bypasses the need for assembly altogether.

4. High-level Languages

So, what are higher-level languages? Assembly language requires multi-line, detailed instructions to carry out simple functions. Higher-level computer languages are processor agnostic and are designed to give instructions in a human-readable manner. Programs written in higher-level languages can be read by humans (as long as they know the language) and are concise and easy to operate.

For example, what could take eight lines worth of code in assembly might take three short commands in a high-level language. This information would then use a compiler tool to convert it into machine code.

However, the execution time is greater than with lower-level languages. Although the difference is minimal, some programmers prefer to use assembly language when a short program is needed, and time is of the essence.

Machine Language	Assembly Languages	Middle-Level Language	High-Level Language
Machine language is the most basic language to interact with the computer.	Assembly language is a low-level language that relies on codes to interact with the computer. It's a little easier to work with than machine language but not as easy to use as high-level languages.	A middle-level language, often called common intermediate language (CIL) or bytecode, is a more universal language that compilers will use to run code cross-platform.	A high-level language is a computer language that uses commands easily understood by humans. This is often a developer's go-to language because it's easy to use.
Machine language is understood by the processor. It's a binary language consisting of 0s and 1s	Assembly language contains slightly more human-friendly short codes. It relies on assemblers to convert the code into machine language.	It's the easiest low-level programming language for people to read (easier than assembly).	High-level language has very user-friendly commands and syntax. In some cases, it uses a compiler to translate these commands into assembly, which then translates into machine language for processors to understand. But isn't a technical requirement; many compilers instead translate to CIL instead (which then converts to machine language).
The execution speed of machine language is very high as the processor doesn't have to translate it.	The execution speed of assembly language is not as fast as machine language. The commands need to be translated to be understood by the processor.	The execution speed of a compiler is slower than that of an assembler.	The execution speed of a high-level language is the slowest in comparison to the other types of languages.

It is humanly impossible to write executable programs in machine language.	By and large, this language is no longer used. It's rare to see in the wild anymore,	This is something that's done on the backend via a compiler that doesn't require the developer to do anything.	The syntax of high-level language can be easily read by humans and hence, programming is faster and easier.
Example(s) of machine languages: Binary language	**Examples of components that may use assembly:** ARM, MIPS, x86 (each has its own assembly language)	**Example of a CIL:** Microsoft intermediate language (MSIL), which later became standardized as CIL	**Examples of high-level languages:** Java, JavaScript, C++, C#, F#

Assembly Language Short Code Consists of Opcode and Operands

Assembly language instructions use mnemonic code and meta statements to give instructions to the computer (once translated into machine language). A computer can perform arithmetic functions like addition and subtraction, and logical functions like comparison and conditional functions. To instruct the computer, the assembly language uses one operation code (opcode), followed by two arguments (operands).

Opcode: The operation code (opcode) instructs the machine about the type of operation to be performed.

Operands: Operands can represent the data to be processed, or the constants on which the operation is to be performed. Operands can either be a constant value or an address pointing to where the data is stored (e.g., a specific register value or memory location). They can define the address of the data using:

- **Register values –** Registers are a small part of a computer processor used to store data for immediate, temporary use. They are built to enhance efficiency in program and operation executions.
- **Stack values –** A stack is an abstract data type (ADT) that is linear and uses the last-in-first-out (LIFO) method for following operations. Homogenous data are stored in a stack where the last input becomes the first output.

- **Memory values –** Random access memory (RAM) is the place where the computer stores temporary data like the variables in a program. Every byte has a unique memory location. So, if the program wants to retrieve the variable during the execution, the memory address would be used.
- **Input/output ports –** Input/output ports are the sockets on a computer to attach external drives or networks. If the data is stored on any drives attached to the computer, you can point to that data in operands.

The following table shows the commands in different assembly architecture and assembler program languages:

Operation	Syntax in ARM	Syntax in x86	Syntax in MIPS
Addition	ADD	Add	Add
Subtraction	SUB	Sub	Sub
Multiplication	MUL	Mul	Mult

A Breakdown of the Two Types of Assemblers

Assembly languages need an assembler in some cases to translate high-level code into machine code. Assemblers are categorized based on the number of cycles (passes) they perform to generate the object code. We might use a one-pass or a multi-pass assembler depending on the situation and requirements.

Let's take a look at the two types of assemblers.

- **One-pass assemblers (single-pass assemblers):** A one-pass assembler passes the code through the assembler only once to generate the object file. It's faster than a two-pass assembler but is also more complex. This type reads the source files one time before working on data structures in memory.
- **Two-pass assemblers (multi-pass assemblers):** A multi-pass assembler is an older style of assembler that uses more than one pass to generate the object file. This involved the assembler reading files twice, each separate pass handling specific tasks.

Of course, there are other differences between one-pass and two-pass assemblers, but we're not going to get into all of that here. Check out this Quora post for a more granular comparison.

And just remember: in many cases, assemblers are no longer used. This is because many high-level languages compile into CIL and then directly into machine language; they bypass the need for assembly altogether.

Pros and Cons of Learning and Using Assembly Languages

Assembly language is typically used in boot codes and operating system kernels but can also be used at higher levels to create many different kinds of programs. Let’s look at some of the pros and cons of learning and using assembly language:

Pros of Learning/Using Assembly Language	Cons of Learning/Using Assembly Language
Assembly language makes it faster to run complex code.	It is incredibly difficult to write code in assembly language compared to high-level languages.
Simpler for humans to understand than machine-level language.	The syntax is hard to learn and can be difficult to remember or use quickly.
Convenient to use in testing, debugging, optimization, and development.	The code is longer and more complex than with a high-level language.
Requires little memory and works well with devices with low random access memory (RAM).	It lacks portability between devices with different processors.
Assembly language can be used to reverse engineer software where the source code is not available.	It’s often bypassed by compilers that convert high-level languages straight to machine language.

Final Words on Assembly Language

Assembly language fits between high-level and machine-level languages, helping to facilitate communication between users and computers that use different languages. While high-level languages are easier for a programmer to use, machines don’t understand them. On the other hand, machine code is easier for devices but very difficult for humans to understand. So, assembly language serves as a way to bridge that gap historically.

Although assembly language is easier for us to use than machine language, that doesn't mean it's *easy* to use. This is why most high-level programming languages and programs nowadays bypass the need for assembly and go straight to CIL and machine language after that.

#ASSEMBLY LANGUAGE

About the author

Megha Thakkar

Megha can usually be found reading, writing, or watching documentaries, guaranteed to bore her family. She is a techno-freak with interests ranging from cooking to travel. A regular contributor to various web security blogs, she has earned her diploma in network-centric computing. Being a mother has taught her to speak less and write more (coz who listens to moms, right?).

You might also like

What You Need to Know About PCI DSS 4.0 (and Version 4.0.1)
June 24, 2024

Cyber Security and Digital Forensics: What's the Difference?
March 22, 2022

GDPR Data Breach Reporting Requirements
Here's what to know about when and how to report a GDPR personal data breach.

What Are the GDPR Breach Reporting Requirements?
May 6, 2021

SECTIGO Store
OPERATED BY THE SSL STORE

Shop SSL

Search InfoSec Insights

Search here

Latest Articles

5 SMB Takeaways from the NIST Cybersecurity Framework 2.0

Up Your Game with This Small Business Cyber Security Plan Template

What You Need to Know About PCI DSS 4.0 (and Version 4.0.1)

How to Perform a Website Security Check

What Is Transport Layer Security? A Breakdown of the Secure TLS Encryption Protocol

Recommended Posts

DevSecOps: A Definition, Explanation & Exploration of DevOps Security