Understanding the Compilation and Execution Process in Computers

Sarwar Nazrul

October 30, 2023

Audience and Scope:

This guide is designed for a wide range of readers, from those just starting out in computer science to experienced developers looking for a quick refresher. It’s perfect for students, curious learners, and teachers. The document explains in simple terms how computers read and execute the code that developers write. It starts with the basics, showing how code is turned into a language that computers can understand.

As the guide progresses, it goes into how computers run this code, making sure to cover everything in a way that’s easy to grasp. Even when tackling more complex topics like linking and the environment where the code runs, the guide uses everyday examples to help make these difficult ideas easier to understand. In this way, the guide aims to provide clear and helpful information to all its readers, regardless of their previous knowledge or experience.

Introduction:

In the world of software, developers write detailed instructions using a specific programming language. This creation process is integral to making applications or tools that serve various purposes. These instructions are shown in Figure 1 as "hello_world.c". While these sets of directions are clear and structured for developers, they're written in a way that our modern computers can't directly understand or execute.

Figure 1: *Compilation Process of a C Program*

Expanding on this process, the 'Compiler,' a specialized program designed to translate high-level programming language into machine code, becomes essential. Its primary function is not just to act as a translator but also to ensure that there's no loss of information or intent. By taking the written instructions, it meticulously converts them into a streamlined language that the computer can seamlessly understand, which is purely made up of 1s and 0s. Once transformed into this new version, termed "hello_world.o", computers can then easily execute the instructions, allowing the desired software or application to come to life and function as the developer envisioned.

Compilation Process:

Compilation is like translating a story from one language to another so a computer can understand it. In essence, it bridges the gap between human ingenuity and machine capability. Programmers, equipped with their expertise and creativity, script their ideas, and it's the compiler's job to make these scripts understandable for the computer. It's a vital step where the ideas and instructions written by programmers are turned into a format that the computer's brain, the CPU, can work with. The CPU is a powerhouse that performs billions of tasks in a fraction of a second, and it requires instructions in a specific format to perform at its best [1].

Just as a translator ensures that the essence of a story remains intact across languages, the compiler ensures that the programmer's intentions are accurately represented in the computer's language. It's not just about converting words but about preserving the essence, the logic, and the flow of the original code. This process has several steps to ensure the translation is accurate and efficient: breaking down complex instructions, optimizing for performance, and finally, producing the binary code, which is the computer's native tongue. These meticulous steps guarantee that the software runs smoothly and efficiently on any machine [2].

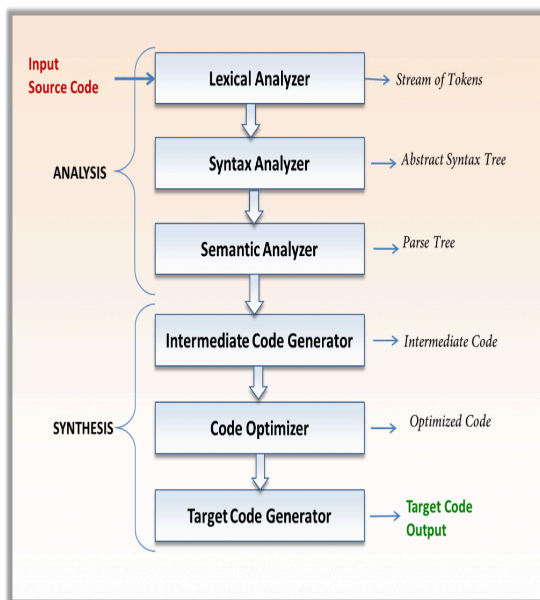

Figure 2: *Phases of Compiler Design from Source Code to Target Code Output*

❖ ***Lexical Analysis****:* The compiler reads the code and picks out important parts like names, actions, and symbols.

❖ ***Syntax Analysis****:* It checks if the code is written correctly like making sure sentences in a story have the right structure.

❖ ***Semantic Analysis****:* The compiler makes sure the code makes sense. It's like checking if the story's events and characters fit together logically.

❖ ***Code Generation***: Here, the code is turned into a language the computer speaks, made up of 1s and 0s.

❖ ***Optimization****:* The compiler tries to make the code run faster and smoother, like editing a story to make it more exciting.

Execution Process:

After the code is translated into a language the computer understands, it's time for the computer to act on it. The execution process is where the computer brings the code to life, turning written instructions into actions. It's a systematic and organized sequence, ensuring that every piece of code gets its turn and that tasks are performed accurately and efficiently [1]. Here's a more detailed look at how the computer runs the code:

- ***Loading***: The computer first places the code in its memory, readying it for action. This is like setting up the stage before a play begins, ensuring all props and actors are in place.
- ***Fetching***: The computer reads the instructions one by one, similar to how we read sentences in a book. Each instruction is carefully retrieved, ensuring no detail is missed.
- ***Decoding***: The computer figures out what each instruction means, ensuring it knows the exact action to take. It's like understanding the steps of a dance routine before performing it.

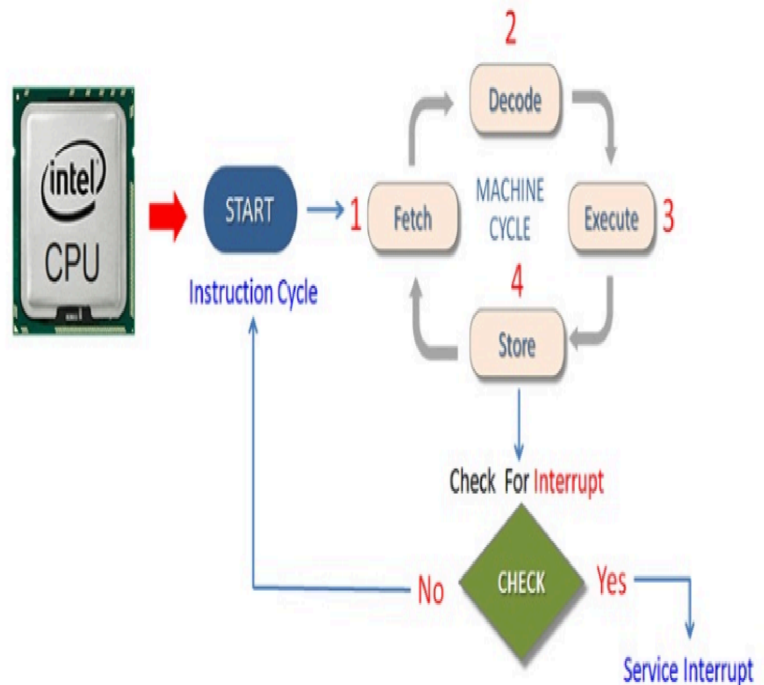

Figure 3: *CPU Instruction Cycle*.

- ***Executing***: The computer carries out the instructions, like following a step in a recipe. It uses its internal components to perform the required actions, ensuring accuracy.
- ***Storing Results***: After performing the task, the computer saves the outcome for future reference. This is crucial as results might be needed for subsequent instructions or tasks.
- ***Handling Interrupts***: Sometimes, the computer encounters urgent tasks or errors. It stops to address them before continuing with the main tasks. Think of it as a detour on a road trip, where you briefly divert from the main path to address something important.
- ***Termination***: When all tasks are completed, the computer stops running the code.

Computers are incredibly fast and can go through these steps billions of times in just a second, allowing them to perform a wide range of functions we rely on daily.

Linking:

Programs often consist of multiple code files or use external libraries. A linker combines these separate pieces into a single executable file, resolving any references between them [2]. Linking is like piecing together a puzzle in software creation. After we've prepared individual pieces of code, we need to join them to make a complete, working program. The linker does this by connecting different code parts and making sure they work well together. There are two main ways to link: static and dynamic. With static linking, everything the program needs is packed into one big file. Dynamic linking, however, uses common parts shared by other programs. It's like multiple toys using the same batteries. The linker also adjusts where things are stored in the computer's memory and keeps track of all the program's parts, making sure everything runs smoothly.

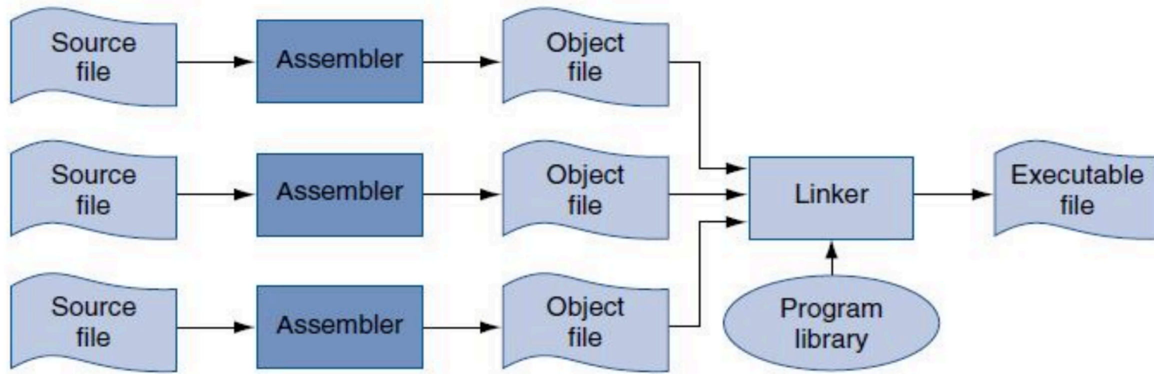

Figure 4: *Multi-file Compilation and Linking Process in Software Development*

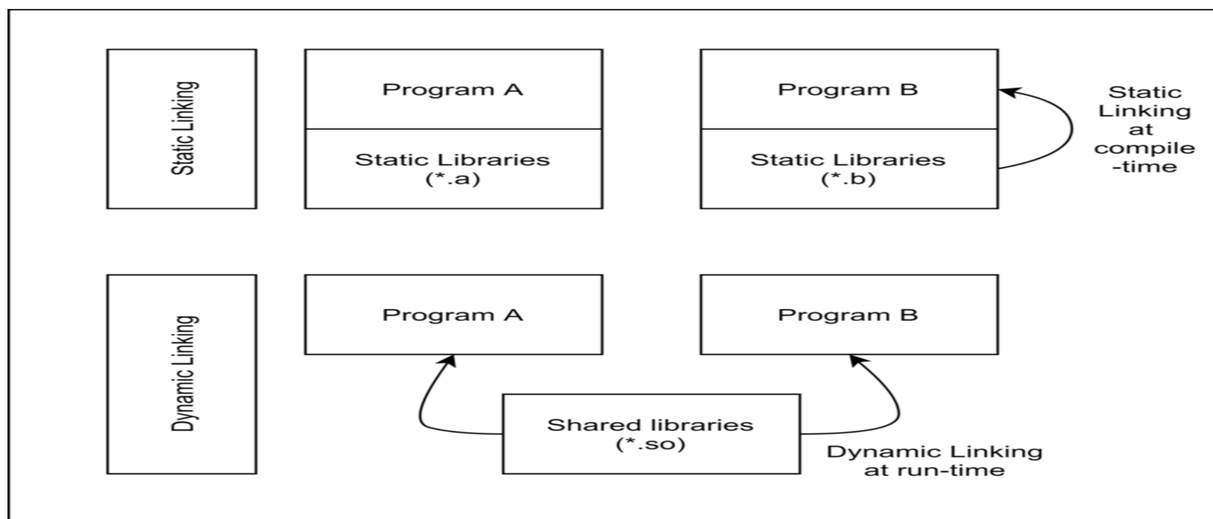

Figure 5: *Static vs. Dynamic Linking in software programs.*

Runtime Environment:

The runtime environment is a special system that helps software run correctly and efficiently. It serves as a bridge between the program and the underlying computer system [3]. After a program is created, this environment ensures it operates smoothly, providing a stable foundation. It manages memory, making sure the software uses it properly and doesn't waste any. It also guarantees the consistency of operations, maintaining the delicate balance between the software and hardware.

The environment plays a pivotal role in how software interacts with devices. It ensures that the software works on various computers and systems, adapting to different hardware configurations. If there are unexpected issues or errors, the environment handles them, offering solutions or bypassing them to ensure the software's continuity. It keeps everything secure, setting up protective barriers against potential threats [3]. Additionally, it allows the software to access computer files, connect to the internet, and interface with other software. The environment facilitates multitasking, letting different parts of a program run simultaneously. Some advanced runtime environments even possess capabilities to automatically clean up unused memory or optimize performance. Essentially, the runtime environment is the unsung hero, working behind the scenes, making sure software functions seamlessly and efficiently.

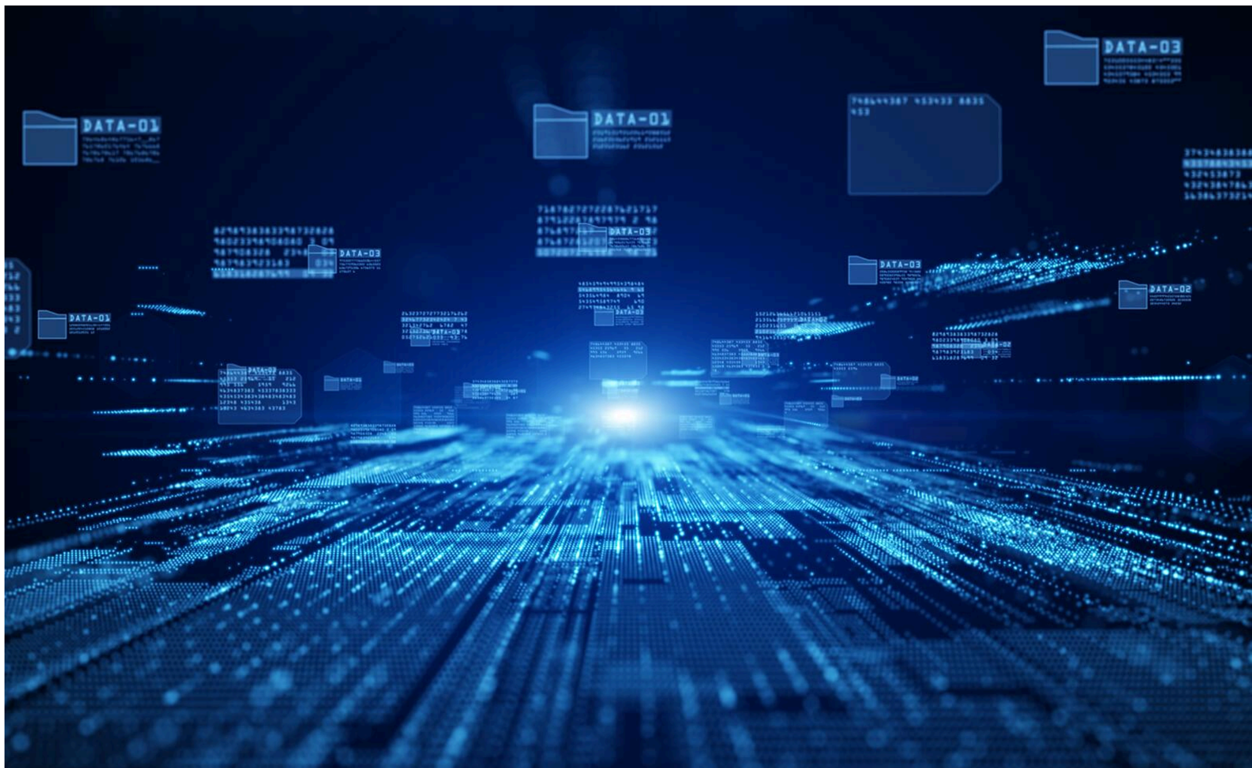

Figure 6: *Digital Data Landscape.*

Conclusion:

Software development is like building a complex puzzle where every piece has a unique place. The processes of compilation and execution are crucial in this journey. Compilation is about transforming the code we write into a language that computers can grasp. Execution, on the other hand, is when the computer acts on that language. This means it starts doing what the code tells it to, like opening an app or saving a file. Linking is like the puzzle's connectors, tying different parts of code together.

Another vital piece of this puzzle is the runtime environment. It ensures that everything runs without hiccups. While most of us just enjoy using our favorite apps, there's a lot happening in the background. Every click, every sound, and every movement on the screen happens because of these intricate processes. They make sure our apps work well and are safe from glitches or threats. Each step, from the start to the finish, showcases the wonders of technology. So, the next time you use a software or play a game, remember the amazing processes behind it. As we move forward, these processes will only evolve, making our tech experiences even more seamless and engaging.

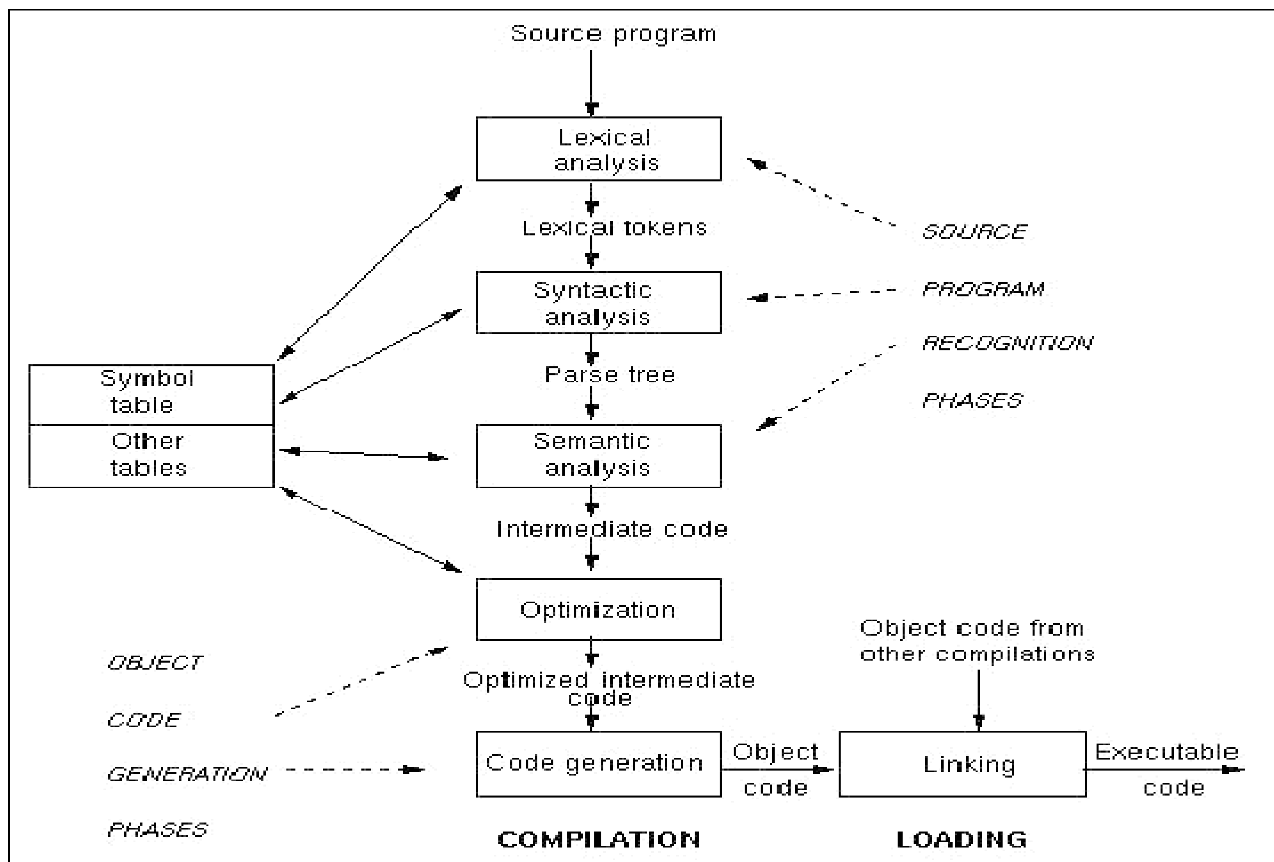

Figure 7: *Flowchart depicting the stages of program compilation and execution.*

Works Cited:

---

1. A. Aho, *Instruction Cycle Explained: Fetch, Decode, Execute Cycle Step-by-Step*: Learn Computer Science, 2021. [online]. Available: www.learncomputerscienceonline.com/instruction-cycle/. Accessed: Oct. 6, 2023.

2. A. Stec, *How Compilers Work:* Baeldung, 2023. [online]. Available: www.baeldung.com/cs/how-compilers-work. Accessed: Oct. 17, 2023.

3. T. Truong, *What Is Runtime Environment?:* Under The Hood Learning, 2021. [online]. Available: www.underthehoodlearning.com/what-is-runtime-environment/. Accessed: Oct. 17, 2023.

Figure 1
Compiler Shematic. (2018, March 21). HPCWIKI.
https://hpc-wiki.info/hpc/File:Compiler_Shematic.png

Figure 2
Six Phases of the Compilation Process. (2017, February 9). KTTPRO.
https://www.kttpro.com/wp-content/uploads/2017/02/Compiler-1024x792.png

Figure 3
CPU instruction cycle. (2021, August 21). Learn Computer Science.
https://en.wikipedia.org/wiki/Multi-pass_compiler

Figure 4
Linker and Loader. (2019, October 17). Digital Learning.
https://1.bp.blogspot.com/-n2U0rzZzoOI/XahJ5Bd_W4I/AAAAAAAAAfw/OT4-40Ch-TwQsHQ9OPqeeWsJj1CkDJI3wCLcBGAsYHQ/s1600/linker-and-loader-5-638.jpg

Figure 5
Static vs. Dynamic linking in software programs. (2023, March 20). Baeldung.
https://www.baeldung.com/cs/dynamic-linking-vs-dynamic-loading

Figure 6
Digital Data Landscape. (2021, June 15). Docusnap.
https://www.docusnap.com/it-documentation/no-unwanted-software-in-the-company/#

Figure 7
Multi-pass compiler. (2023, August 28). Wikipedia.
https://en.wikipedia.org/wiki/Multi-pass_compiler