Compiler, Linker, Assembler, and Loader

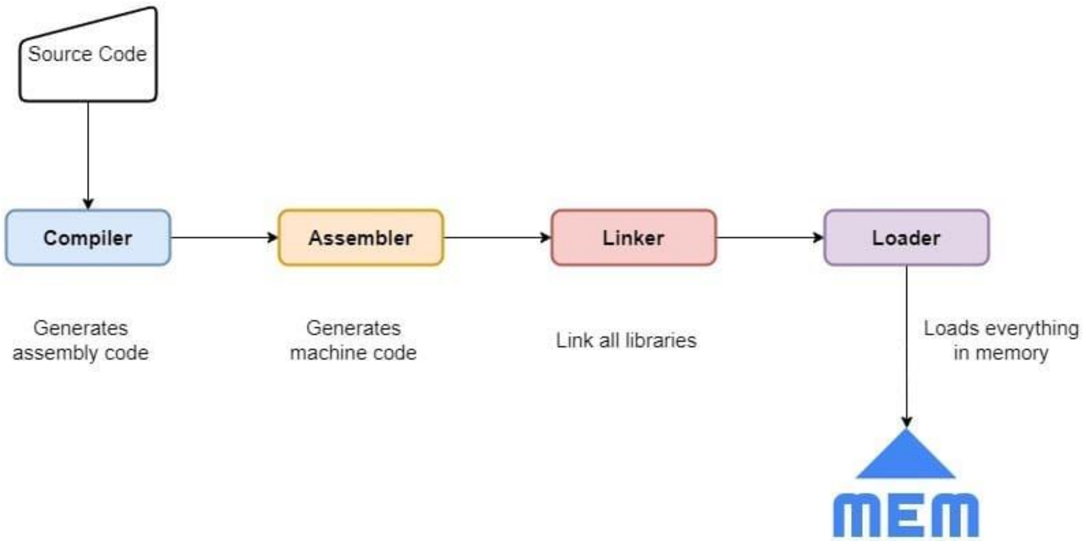
Introduction

In this tutorial, we'll study the roles of the compiler, linker, assembler, and loader modules in a typical process of generating an executable.

1. Executable Generation

A computer program is a sequence of statements in a programming language that instructs the CPU to achieve a particular result.

To execute our program**,** we convert the source code to machine code. So, first, we compile the code to an intermediate level and then convert it to assembly-level code. Then, we link this assembly code with other external libraries or components it uses. Finally, we load it in memory and execute the code:



Let's now dive into each component in the process.

2. Compiler

A compiler is a specialized system tool that translates a program written in a specific programming language into the assembly language code. The assembly language code is specific to each machine and is governed by the CPU of the machine.

The compiler takes a source-code file as input and carries out various transformations on it to output the corresponding assembly code file. Internally, the compiler reads the whole source code in a single pass before starting its work. Then, it creates language tokens from each line of code and verifies the program conforms to the semantic rules of the programming language's grammar. Afterward, it generates assembly code.

Most compilers perform multiple iterations before producing an output file.

3. Assembler

The assembler enters the arena after the compiler has played its part. The assembler translates our assembly code to the machine code and then stores the result in an object file. This file contains the binary representation of our program.

Moving further, the assembler gives a memory location to each object and instruction in our code. The memory location can be physical as well as virtual. A virtual location is an offset that is relative to the base address of the first instruction.

4. Linker

Next, we move to the linker module. The linker spawns to action after the assembler has done its job. The linker combines all external programs (such as libraries and other shared components) with our program to create a final executable. At the end of the linking step, we get the executable for our program.

So, the linker takes all object files as input, resolves all memory references, and finally merges these object files to make an executable file.

Thus, there are two prime tasks of the linker. The first is to probe and find referenced modules or methods, or variables in our program. And the second is to determine and resolve the absolute memory location where these codes need to be loaded.

5. Loader

The loader is a specialized operating system module that comes last in the picture. It loads the final executable code into memory.

Afterward, it creates the program and data stack. Then, it initializes various registers and finally gives control to the CPU so that it can start executing the code.

Conclusion

In this article, we studied the role of the compiler, assembler, linker, and loader modules in program execution.

Each component plays its part. A compiler takes our source code and generates the corresponding assembly code. An assembler converts the assembly code to the machine code. A linker merges all the machine-code modules referenced in our code, whereas a loader moves the executable to RAM and lets it be executed by a CPU.