Basics of Programming

A computer’s CPU can only understand instructions that are written in

machine language. Because people find it very difficult to write entire

programs in machine language, other programming languages have been

invented.

The CPU is the most important component in a computer because it

is the part of the computer that runs programs. Sometimes the CPU is called the “computer’s

brain,” and is described as being “smart.” Although these are common metaphors, you

should understand that the CPU is not a brain, and it is not smart. The CPU is an electronic

device that is designed to do specific things. In particular, the CPU is designed to perform

operations such as the following:

• Reading a piece of data from main memory

• Adding two numbers

• Subtracting one number from another number

• Multiplying two numbers

• Dividing one number by another number

• Moving a piece of data from one memory location to another

• Determining whether one value is equal to another value

As you can see from this list, the CPU performs simple operations on pieces of data. The

CPU does nothing on its own, however. It has to be told what to do, and that’s the purpose

of a program. A program is nothing more than a list of instructions that cause the CPU to

perform operations.

Each instruction in a program is a command that tells the CPU to perform a specific operation. Here’s an example of an instruction that might appear in a program:

10110000

To you and me, this is only a series of 0s and 1s. To a CPU, however, this is an instruction

to perform an operation.

It is written in 0s and 1s because CPUs only understand instructions that are written in machine language, and machine language instructions always have an underlying binary structure.

A machine language instruction exists for each operation that a CPU is capable of performing. For example, there is an instruction for adding numbers, there is an instruction for subtracting one number from another, and so forth. The entire set of instructions that a CPU

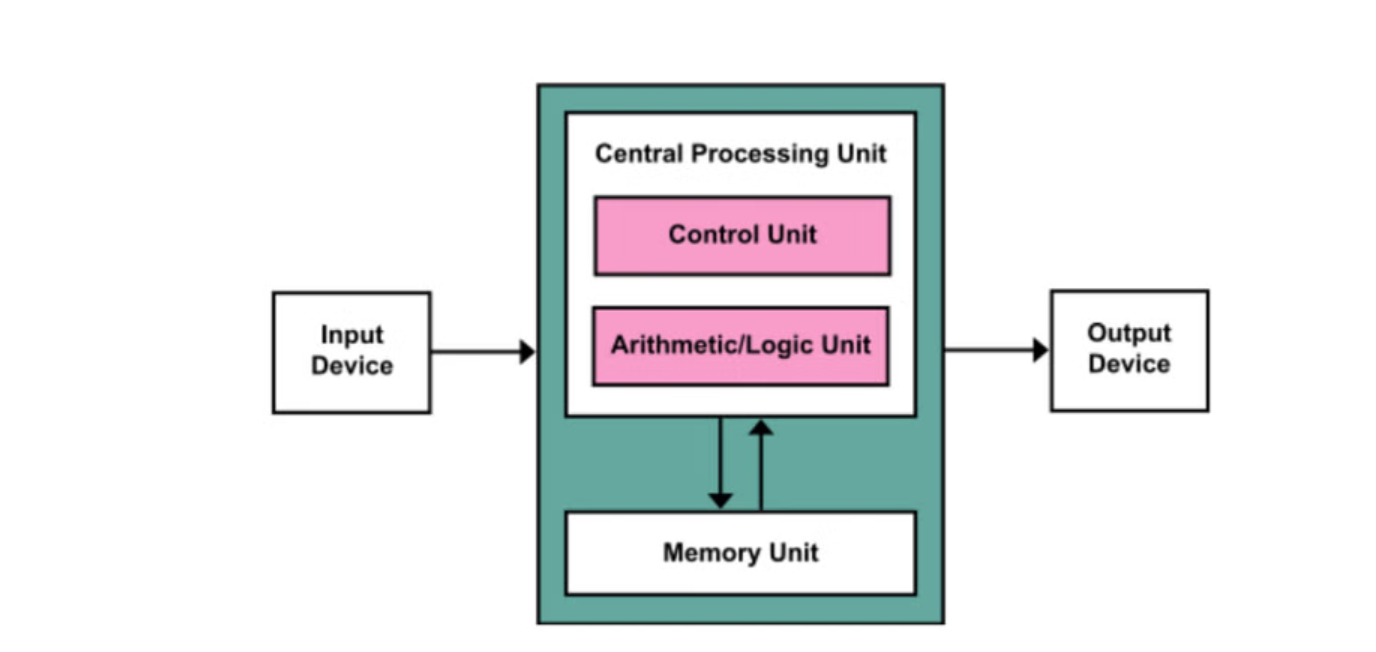
can execute is known as the CPU’s instruction set.

For example, if you want your computer to calculate the amount of interest that you will earn from your savings account this year, the CPU will have to

perform a large number of instructions, carried out in the proper sequence. It is not

unusual for a program to contain thousands or even millions of machine language

instructions.



Programs are usually stored on a secondary storage device such as a disk drive. When you

install a program on your computer, the program is typically copied to your computer’s disk

drive from a CD-ROM, or perhaps downloaded from a website.

Although a program can be stored on a secondary storage device such as a disk drive,

it has to be copied into main memory, or RAM, each time the CPU executes it. For

example, suppose you have a word processing program on your computer’s disk. To

execute the program you use the mouse to double-click the program’s icon. This causes

the program to be copied from the disk into main memory. Then, the computer’s CPU

executes the copy of the program that is in main memory.

How a Program Works

When a CPU executes the instructions in a program, it is engaged in a process that is known

as the fetch-decode-execute cycle. This cycle, which consists of three steps, is repeated for

each instruction in the program. The steps are:

1. Fetch A program is a long sequence of machine language instructions. The first step of the

cycle is to fetch, or read, the next instruction from memory into the CPU.

2. Decode A machine language instruction is a binary number that represents a command that tells the CPU to perform an operation.

In this step the CPU decodes the instruction that was just fetched from memory, to determine which operation it should perform.

3. Execute The last step in the cycle is to execute, or perform, the operation.

Although a computer’s CPU only understands machine language, it is impractical for people

to write programs in machine language. For this reason, assembly language was created in the

early days of computing as an alternative to machine language. Instead of using binary numbers for instructions, assembly language uses short words that are known as mnemonics. For

example, in assembly language, the mnemonic add typically means to add numbers, mul typically means to multiply numbers, and mov typically means to move a value to a location in

memory. When a programmer uses assembly language to write a program, he or she can write

short mnemonics instead of binary numbers.

High-Level Languages

Although assembly language makes it unnecessary to write binary machine language

instructions, it is not without difficulties. Assembly language is primarily a direct substitute

for machine language, and like machine language, it requires that you know a lot about the

CPU. Assembly language also requires that you write a large number of instructions for

even the simplest program. Because assembly language is so close in nature to machine language, it is referred to as a low-level language.

In the 1950s, a new generation of programming languages known as high-level languages

began to appear. A high-level language allows you to create powerful and complex programs

without knowing how the CPU works, and without writing large numbers of low-level

instructions.

Doing the same thing in assembly language would require several instructions, and an intimate

knowledge of how the CPU interacts with the computer’s output device.

high-level languages allow programmers to concentrate on the tasks they want to perform with their programs rather than the details of how the CPU will execute those programs.

Each high-level language has its own set of predefined words that the programmer must

use to write a program. The words that make up a high-level programming language are

known as key words or reserved words. Each key word has a specific meaning, and cannot be used for any other purpose.

In addition to key words, programming languages have operators that perform various

operations on data. For example, all programming languages have math operators that perform arithmetic

The following adds 12 and 75:

12 + 75

In addition to key words and operators, each language also has its own syntax, which is a

set of rules that must be strictly followed when writing a program. The syntax rules dictate

how key words, operators, and various punctuation characters must be used in a program.

When you are learning a programming language, you must learn the syntax rules for that

particular language.

The individual instructions that you use to write a program in a high-level programming

language are called statements. A programming statement can consist of key words, operators, punctuation, and other allowable programming elements, arranged in the proper

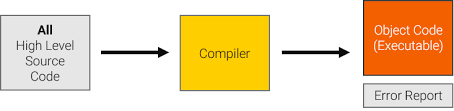
sequence to perform an operation.

Compilers and Interpreters

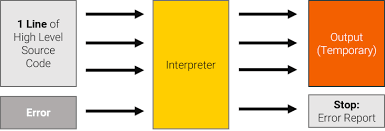
Because the CPU understands only machine language instructions, programs that are written in a high-level language must be translated into machine language. Depending on the

language that a program has been written in, the programmer will use either a compiler or

an interpreter to make the translation.



Create one extra file that is executable file and whole file at once



Statement by statement

A compiler is a program that translates a high-level language program into a separate

machine language program. The machine language program can then be executed any time

it is needed. The interpreter translates each high-level instruction to

its equivalent machine language instructions and immediately executes them.

Because interpreters combine translation and execution, they typically do not create separate machine language programs.

The statements that a programmer writes in a high-level language are called source code,

or simply code. Typically, the programmer types a program’s code into a text editor and

then saves the code in a file on the computer’s disk. Next, the programmer uses a compiler

to translate the code into a machine language program, or an interpreter to translate and

execute the code. If the code contains a syntax error, however, it cannot be translated. A

syntax error is a mistake such as a misspelled key word, a missing punctuation character,

or the incorrect use of an operator. When this happens the compiler or interpreter displays

an error message indicating that the program contains a syntax error. The programmer corrects the error and then attempts once again to translate the program.