Computers, for the Confused

Justin Paul Skycak, *November 2015.*

Computers are complex machines. However, the big picture behind computers quite simple. Complexity arises because commercial computers are optimized for performance, so they include many technological bells and whistles to speed them up and make them more easy to use. But the main ideas behind computers can be understood by anyone.

In this article, we will learn - in simple terms - how computers work. Rather than focusing on the nitty-gritty details and countless acronyms, we'll take a bird's-eye view as we soar from circuitry to the internet. We'll also structure our journey in a problem/solution approach so that we understand why things are the way they are in the world of computers.

So, what do computers do?

Computers exist because we can't always figure out solutions to problems just by thinking about them. Sometimes, such as when attempting to break a code, the only way we can solve a problem is by checking each of a large number of possibilities. Other times, we just tire of thinking.

Either way, rather than thinking about what would happen if we were to do X, it is often easier to just go ahead and do X and observe what happens. Computers let us simulate situations and observe outcomes.

How do we build a device to simulate situations?

To simulate a situation, we need to do things to objects. Hence, our device should include memory (the objects) and a processor (something that changes the objects). We also need to tell, or program, the processor how it should change what's stored in memory.

For example, we could construct a computer out of a person, some rocks, and a hole. The person would be the processor, and the rocks/hole would be the memory. Programming the person would be easy - we could verbally tell the person how to move the rocks in and out of the hole.

A computer can be created with a person moving rocks in and out of a hole. So why isn't this what comes to mind when we think of computers?

Although the person + rocks/hole system can technically pass as a computer, there are several practical problems which make the system hard to use.

First of all, if we want to represent different numbers, we're going to need a lot of rocks. If we we want to represent the quantity sixty-four, we need sixty-four rocks. But we don't want to gather that many rocks, and we don't want to count them up individually every time we need to find how many rocks are in a pile.

Second of all, it's slow and tiring to manually move and count rocks, and they take up a lot of space. Modern-day computers can execute roughly a million

operations per second, whereas a human/rocks/hole computer would peak out around one operation per second. Furthermore, rocks take up a lot of space, so we can't take our rock computer with us if we want to relocate.

How can we use fewer rocks?

We can use fewer rocks if we use multiple holes. Think about the way we count - we can represent any number using only ten digits (0, 1, 2, ..., 9). The key is in the way we order the digits. If we want to represent the quantity sixty-four, we can put six rocks in the left hole and four rocks in the right hole: 64. That's only ten rocks!

We can do even better using two digits (0 and 1) instead of all ten. Using ten digits, we count like this: 0, 1, 2, ..., 9, 10, 11, 12, ..., 19, 20, 21, 22, ..., 99, 100, and so on. Each time we increase, we change the rightmost digit to the next highest digit – unless the rightmost digit was 9, in which case there is not a next highest digit, so we increase the digit immediately to the left of the 9 and replace 9 with 0. Using only two digits but keeping the same rule for increasing a quantity, we count like this: 0, 1, 10, 11, 100, 101, 110, 111, 1000, ... In this binary system, the quantity sixty-four is represented as 100000000. Only a single rock is needed!

How can we make computers fast and small?

Most modern computers use electrical circuits. So that the circuits cannot be easily messed up, they are usually soldered onto chips called integrated circuits.

But although most of today's computers are built from electrical circuits, computers do not have to be built from electrical circuits. Electrical circuits are just the best known solution as of now. Perhaps quantum computing or organic computing may become a better option in the future.

How can we adapt the computer to multiple purposes, if the circuit arrangement is fixed?

It's easy enough to build a an electrical circuit to accomplish a single task. However, we want to build a computer that can do any sort of operation we tell it. We want to build a programmable computer.

Integrated circuits are fixed, so we cannot rearrange circuits to perform different operations as needed. Besides, even if we could rearrange the circuits, it would take a while, and we could very easily make a mistake.

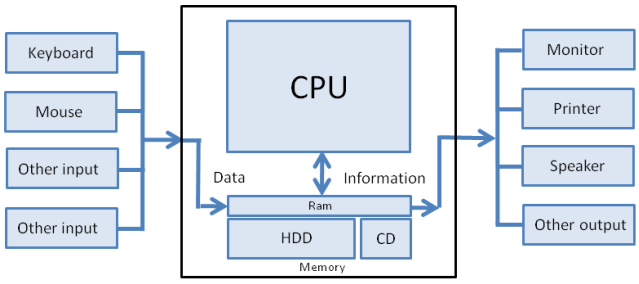
The solution is to store programs in the computer's memory. Rather than building a circuit to execute a fixed operation, we build a circuit that can execute multiple operations, and can execute those operations in any sequence that is stored in its memory. We allocate a memory locations to each operation, so that an operation will be completed whenever its memory location is filled. Then, we can change a computer's program by simply modifying its memory.

How do we modify and view a computer's memory?

We still need to figure out how to physically modify a computer's memory. We could manually turn on and off different memory locations, but this would be tedious and difficult. The solution is to use an input device, such as a keyboard.

For each memory location that we allocate to a particular operation, we define a binary number called an operation code ("opcode" for short). Whenever the computer sees the opcode, it will know to execute the corresponding operation. This way, we can program a computer by giving it a list of binary numbers, often via keyboard.

Because the results of a program are stored in a computer's memory, we also need some way to view computer memory. The solution is to use an output device, such as a monitor. The monitor can also help us see what we're doing as we're programming the computer.



Inputs modify computer memory so that the processor (CPU) knows how to operate on the memory. Then, output devices are used to display the resulting memory.[1]

This is where we make the jump from physical hardware to abstract software. We no longer have to worry about designing the physical parts of a computer, since we now have a set of opcodes, called an instruction set, which define all the individual operations that a computer can execute. We just

[1] https://upload.wikimedia.org/wikipedia/commons/6/60/Computer2.png

need to figure out what opcodes to give the computer, and in what order to give them, so that the computer completes our desired task. We need to create a language by which we can easily communicate with the computer.

Suppose we create a language that allows us to communicate with our computer. How can we be sure that ALL computers will understand us?

Often, when we write a program for a particular computer, we also want to be able to run it on another computer. The two computers might not have the same instruction sets, so an instruction on one computer might not mean the same thing on the other computer. However, we can solve this problem with standards.

We design instruction sets according to certain standards, and when we're writing code that should be transferable from one computer to another, we write the code in terms of the standard instructions. Different computers can have different instruction sets, but as long as the instruction sets adhere to standards, code written in terms of the standards can be used on any computer.

Thanks to standards, pieces of code that accomplish particular tasks can be used across computer platforms. These pieces of code can be gathered into libraries, and rather than writing programs from scratch, we can piece together code from a library.

How can we write programs more easily?

We have a way to put programs into our computer's memory, we can write our instructions in terms of standards so that our programs can be read by any computer, and we can write programs by piecing together code from libraries. However, writing programs in binary is tedious and error-inviting for humans (even when using letters, we make many typos!). We think semantically; hence, we would rather use a language which assigns mnemonics (words and abbreviations that are indicative of an operation's function) to the binary machine code. We call this language assembly language.

If we write a program in the mnemonics of assembly language, though, we still need the computer to receive binary machine code. The computer can't read the assembly language, so we need to translate it. We do this with an assembler, a computer program that takes assembly code as input and gives the corresponding machine code as output.

Even still, working with assembly language can be tedious at times. It would be nice if we could write programs in a language that flows better and is more like the ones we speak with other humans. This way, we would make fewer errors when writing programs, and programs would make more sense to us when we try to read them. Furthermore, although there are standards in place, different machine and assembly languages are often used for different kinds of computers. How can we make an easy one-size-fits-all programming language?

This is where we make the jump from low-level languages (the machine and assembly languages) to high-level languages, such as C, Python, MATLAB, Java, etc. Just as we used a program called an assembler to translate assembly language to machine language, we use another program called a compiler to translate high-level languages to low-level languages.

Using an input device, we can easily program a computer to do any task that can be accomplished by the computer's instruction set, and our program can work on any computer that is designed in accordance with particular standards. Humans can program computers, so why not let computers program other computers? Instead of using a keyboard as an input device, why not use another computer? All we need to do is transmit information from one computer to another.

How can we transmit information from one computer to another?

One solution is to connect a memory storage device to a computer, write part of the computer's memory to that device, physically transport that device to another computer, and write the device memory to this computer. However, using this method, information could travel only as fast as humans could manually transport it.

A better solution is to use wires or optical fibers to transmit information from one computer to other computers, without any manual labor involved. We can also designate specific computers in the network, called servers, to do specific functions for all the other computers. This arrangement of communicating computers is called a computer network.