Computer Systems

Alex Sverdlov
alex@theparticle.com

1 Introduction

A computer is a device that computes some function. They can be built out of essentially anything.

2 A bit of history

Early computers were mechanical devices with gears that needed to be turned by hand. For example, Stepped reckoner, invented by Gottfried Wilhelm Leibniz around 1690s was a mechanical calculator capable of performing the four arithmetic operations.

It was later realized that many useful computations can be accomplished with specially interconnected switches that are either on or off.

The computers of WWII-era were made out of interconnected electronic relays. A relay is an electronic switch that opens and closes a mechanical switch. Because of the mechanical motion to open and close the switch, the operating speeds of such devices were limited (maximum of around tens of switches per second). The mechanical parts often rattled, had electrical arcing, and were susceptible to dirt and actual bugs getting jammed in the parts (it is popularly thought that the term software "bug" originated from a moth that was crushed by a relay).

Around the same time (WWII-era), folks realized that a vacuum tube can serve the same function as a relay, and a vacuum based computer was constructed. Because the operation was fully electronic (no mechanical moving parts), the switch could be operated thousands of times faster than a relay. Vacuum tubes often burned out, requiring redundancies built into the machine to enable successful operation.

The transistor, invented around 1947, is a solid state (no moving parts) electronic switch. Unlike a vacuum tube, it was very reliable, and could operate at much faster speeds. It was also very expensive—but the price dropped very quickly.

Around 1960, Integrated Circuits (ICs) were invented. These are essentially dozens (and subsequently thousands, then millions, then billions, etc.) of transistors etched into a piece of silicon. The success of modern technology is essentially based on successful mass production of integrated circuits.

3 Stored Programs

Early computers were purpose built for a particular task, and needed an expensive and error prone re-wiring & rebuilding for different tasks.

Around 1930s, Alan Turing published a paper describing what is now known as a Turing Machine. He reasoned that any problem can be solved by following a fixed set of instructions, plus the ability to read from and write an infinite paper tape, one character at a time. One particular program was the program for the program for the Turing machine itself—in other words, a Turing machine can run a Turing machine... or any other program specially coded on a piece of paper. In other words, a properly built "universal" machine can run any program that can be written.

Turing machine model was not practical, and the model we currently use is: von Neumann architecture (described in 1945). It has:

- A CPU with registers
- An instruction register (pointing to the current instruction in memory).
- Memory that stores data and instructions.
- External storage
- Input and Output mechanism

Once working instances of this architecture were built, it was possible to re-program computers to perform any task using only software. This allowed hardware and software to evolve at their own respective paces.

Recently, this 'stored program' concept has been taken to yet another level with vitalization: entire networks and clusters of computers are now software defined—in a sense, we're defining the hardware we want using instructions.

Just a note about the Turing machine: it is an ideal machine that is more powerful than any machine we can actually build—it has an infinite tape which gives it capabilities that physical devices won't ever have.

4 Types of Computers

Computers come in all shapes and sizes. The ones everyone is familiar with are desktops and laptops. Perhaps not as obvious are phones, tablets, TVs and lots of other smart devices we have all around.

Then there are servers that are constructed with some use in mind. For example, a machine with 32 CPUs and 100 connected hard-drives, etc.

Recently, the cloud made the type of computer less relevant. It is trivial to allocate whatever computer we need with whatever specs we need. The computer is defined in software, and the 'machine' often runs along-side many other machines on the same physical hardware.

5 Software

Software is everything that runs on hardware. The variety is hard to describe: essentially there are two bulk categories: operating system (or similar) and everything else (applications).

5.1 Operating Systems

With different kinds of hardware, users and application programs need a way to use hardware in a consistent and abstract way. For example, a file is a file, irrelevant of whether it's stored in a disk, DVD, or network.

Operating systems abstract away CPU, RAM, and storage. Unify access to resources such as files, sockets, etc., via an API, etc.

Often software that is not "explicitly" operating system-ish gains some of the same attributes. For example, an operating system manages resources on a single machine. A cluster based operating system, such as Hadoop, or AWS environment, is essentially an operating system to use computer clusters—they abstract away concepts such as files and other resources—things that are potentially stored on hundreds of computers are just 'files' in Hadoop File System (HDFS).

5.2 Applications

Applications is everything that is not operating system. This categorization puts things like database servers, web-servers, word processors, and spreadsheets in the same category.

Often application software is custom written for use at a particular company, or franchise.

6 Programming Languages

The basic idea of computer programming is that we need to get the computer to do what we want it to do. How we express those wishes to the machine is what makes a programming language.

6.1 Procedural Languages

A procedural language is essentially a sequence of precise instructions of how to perform a particular task—the computer just follows these instructions to the letter.

C is an example of a procedural language.

6.2 Object Oriented Languages

Object oriented languages are essentially procedural languages, with an abstraction at the level of objects. Instead of working with arrays or structures, we may operate on customer objects, which have a state that the objects maintains.

In some languages, object orientation is just syntactic sugar on top of a procedural language.

An example of an object oriented language is C++ and Java, though most languages these days have a bit of object-oriented styling.

6.3 Functional Languages

Like object oriented languages where the primary abstraction was an object, in function languages the primary abstraction is a function. The programmer creates functions by composing and creating new functions, etc.

Functional languages (or language concepts) come in many varieties. For example, JavaScript is a kind of functional language, and so is Perl and Python. Recently, Java got some functional constructs, such as the ability to pass a lambda function.

An example of a semi-pure modern functional language is Scala. It is often used for ETL and data analysis as part of Spark.

One useful feature of some functional languages is the lack of side-effects: meaning that when a function is called, there is no state that gets modified—just the return value. This allows for results caching, and very reliable testing—as the function will always return the same thing given the same inputs.

6.4 Declarative Languages

Declarative languages are what are known as 4th generation languages (assuming everything that came before is 3rd or less generation). What makes declarative languages special is that normally, when we want to solve a problem, we tell the computer *how* to solve it. In declarative languages, we tell the computer *what* we want, and let the computer figure out the *how* part on its own.

An example of such a language is: SQL (structured query language).

6.5 Markup Languages

Alongside programming languages which store instructions for computers, there are markup languages, which store either data or text, for computer or human consumption. A few examples are HTML, XML, and json.

These languages are very important in standardizing input and output of programs, etc.