The Magic Pixie

How the Magic Pixie in your Computer makes it work

Computer

By 'computer' I mean any programmable digital device, so that includes desktop PCs, laptops, tablets, smartphones, smartTVs and so on.

Level of abstraction

This uses one level of abstraction - discussed at the end

Processor and instruction set

The device contains a 'processor' - such as a 64 bit Intel Pentium.

This might be multi-core multi-processor, with several levels of cache, a separate graphics processor, and many other optimizations. Here I just describe the basic process (and show that in fact the magic pixie has nothing to do).

The processor can execute instructions. These are very simple steps, such as add two numbers, move a value from one place in memory to another, compare two values, jump to a new instruction (not just the next one) and so on. These are called machine code instructions, or native code - not program steps in Java or C.

A processor will have a set of maybe around 100 different instructions. Different processors have different instruction sets.

Each instruction has a different 'opcode', operation code. The opcodes are numbers, so can be written in binary. A program is a sequence of instructions.

RAM and bus

The instructions are held in memory, usually RAM. This is connected to the processor by a 'bus'. This is a set of wires, each of which can carry a voltage, 'on' or 'off'. IOW the bus can carry a binary pattern, from memory to processor and the other way as well.

Fetch execute

So long as power is supplied to the processor, it does the following (why? Because of its electronic design):

1. Fetch a program instruction from memory into the processor

2. Decode it

3. Execute it (do the instruction)

4. Work out where the next instruction is (usually the next one in memory)

5. Go to step 1

Step 2, instruction decoding, is done by a set of logic gates in the processor. Each instruction has a different opcode, a binary pattern which the logic gates can 'recognise'.

Step 3 might be something like add the number in address 9 into address 11. This means the system must fetch the values from address 9 and 11, add them, and write the result back to 11. IOW step 3 might require several more memory read and writes.

That is it. No magic pixie.

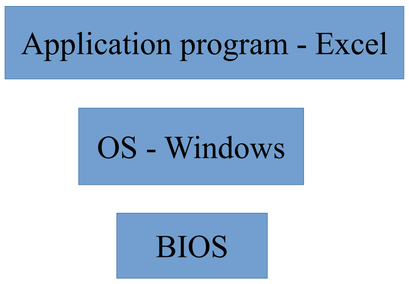
Layers of software

In fact that is just instruction execution, with very simple basic instructions. We also need something to make the device usable. For example we want multi-processing, so we can run say a web browser and a media player at the same time. Something is then needed to make sure these processes run in separate memory areas, and cannot read or write into other memory areas (or we have no security).

The 'something' is an operating system, like Windows or Linux. An OS is just software, and it executes the same way as any other software.

Usually an OS is stored on disc, so there is also a need to have something to load it into memory on power start - to 'boot' the device. Again this is native code software, stored in ROM - so it is there already on pwer switch-on. This (and other facilities like memory self-test) is held in the BIOS (or EUFI).

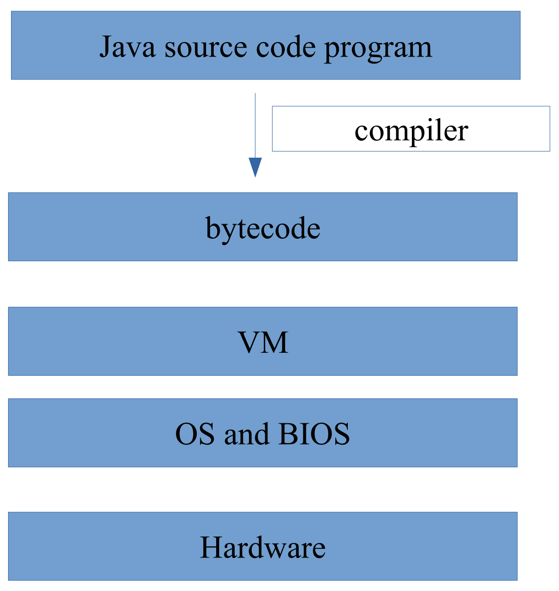
So we have at least 3 levels of software:



Source code and virtual machines

Native code is very difficult to program in directly. Usually we use a language like C or Java, and write source code. A compiler translates that into native code.

An issue is that each different processor as a different native code instruction set, so source code has to be re-compiled for a different platform (OS and processor combination). This is how C works. Java uses a virtual machine. This is a piece of software which can execute programs in 'bytecode'. There is a different VM for each platform, written so that the same bytecode will work the same on each platform. So we have:



Levels of abstraction

I have not talked about how processors work - transistors and TTL and CMOS and so on, or how VLSI chips are made. That would be a very concrete level, the least abstract.

The most abstract is a Turing machine, or lambda calculus. These are mathematical models of what a 'computer' is, and they say nothing about what it is made of.

An example of an alternative to current computer design based on binary is quantum computing - currently still in its infancy.