Computer Organization

Simple Levels

Modern computers are organized in levels. The most fundamental level is the hardware or digital logic level. Each new level on top of this represents a new level of abstraction. Each level of abstraction hides a few more details from view. Consider a few simple levels:

Operating System Level Machine Language Level Digital Logic Level	The digital logic level is concerned with electrical current and lack of current. Each instruction that the computer executes sets the lines that control the processor, either by allowing power to run through a line, or by preventing power from runnning through a line. Each instruction has a different configuration for which lines get power and which ones don't.

The machine language is an abstract representation of which lines get power and which ones don't. It is a simple abstraction: a one represents power, and a zero represents no power. We can write a machine instruction, such as 00000011001010, and the processor will set and clear the appropriate lines to perform the indicated operation. This level of abstraction moves the programmer away from the hardware and into the software. The programmer writes a program that the computer interprets into electrical currents.

The operating system is just a collection of machine language subroutines. After people programmed in machine language for a while, they noticed that they were performing the same instructions every time they read input, or wrote output. So they created subroutines that could be called to simplify these tasks. Now, instead of writing the 10 lines necessary to read some input, the machine language programmer could just call a subroutine.

Here is an analogy for these three levels. Think about the lights used to illumine a theatrical stage. Imagine that a play is being performed on this stage. Before each scene of the play, it is necessary to set the lights to produce different effects. For example, one scene may be set in broad daylight, so all the lights would be turned on. Another scene might be set on a moonlit night, so only a few lights would be on. A third scene might have a soliloquy, so a few bright lights would be aimed at one spot.

Hardware level. In its simplest form, imagine that each light has a power cord that must be plugged into an outlet in order to be turned on. Then before each scene, the stage manager has to plug in the appropriate cords into outlets to achieve the correct lighting.

Machine Language Level: Now suppose that there is a control panel with a set of toggle switches, one for each light. Each light can be turned on or off by flicking the appropriate switch. Before each scene, the stage manager sits at the control panel and flicks the switches for the lights.

: The control panel has been upgraded with some new buttons that represent each scene. It is possible to set these buttons to control many lights. By pressing the button for scene one, all the lights for scene one would be lit. Before each scene, the stage manager pushes one button to set all the lights for the scene.

Complex Levels

Here is a more realistic view of the levels in a modern computer:

Application Program
High Level Language
Assembly Language
Operating System Level
Machine Language Level
Microprogram Level
Hardware Level

Microprogram Level: The Intel chips have all used a microprogram to interpret machine language instructions into hardware instructions. Some of the instructions in the Intel instruction set are too complicated for the hardware, so they must be interpreted into a series of actual hardware instructions. In this case, the microprogram contains actual hardware instructions, and each machine language instruction is executed by calling a subroutine in the microprogram. The microprogram technique allows the hardware to be modified, while remaining backwards compatible with the old machine language. The change from machine language to microcode done by interpretation.

Assembly Language Level: This level has a one-to-one correspondence with the machine language. Instead of using ones and zeros to indicate an instruction, symbolic names are used. So the machine language instruction 00000011001010 can be written as ADD CX,DX. The change from assembly language to machine language is done by translation: the assembly file is used to create a new machine language file.

High Level Language: This would be a language like C++, where one statement would be translated into several machine language instructions. For instance

```
        if (A > B) C = 1;
```

would translate into these instructions (represented in assembler instead of machine)

```
        mov ax,A
        cmp ax,B
        jle next
        mov C,1
next:
```

Application Program: This would be like a word processor that is written in a high level language. Each action in the word processor would result in entire subroutines in the high-level language being executed.

Changing from one level to another

When an instruction from one level is changed to a lower level, two things can happen

1. There is a change from a more readable format to a less readable format. For example, changing assembly to machine.
2. The one instruction is changed into several instructions at the new level. For example, changing C++ to machine. Imagine how many hardware instructions are executed for one instruction in an application program.

There are two ways that the change can take place when changing from a higher level to a lower level.

1. Translation
2. Interpretation

Translation

Each high instruction is translated into the low level and the new low instructions are saved. After all the high instructions have been changed to the low level, then all the new low instructions are executed.

Interpretation

Each high instruction is translated and executed in the low level before continuing with the next high instruction. The new low level instructions are not saved.

Virtual Machines

When we use computers, we do not think in terms of levels. When we work on a word processor, we think the computer directly understands the commands we give it. When we work in C++, we think the computer understands what a *for* loop is. When we go use an ATM machine, we imagine there is a tiny teller inside that is giving us money! In reality, the computer does not understand any of these high-level abstractions. The computer know how to add, subtract, and move data. It has no idea what **bold** means, nor what a loop is, nor what a bank account is. These are all abstractions that we use.

Virtual Machine is the phrase (abstraction???) we use to talk about the different levels of computer organization. Imagine a machine whose hardware language is the language for a word processor. It could be built. Then when you click the **bold** button, the action is carried out in hardware. No translation nor interpretation would be needed. The machine would be very fast, but it could only do word processing. Similarly, a machine whose hardware language was C++ could be written. Then it would be a very good C++ machine, but wouldn't be able to do word processing. So a virtual machine is just a phrase to represent how we think about computers. When I am writing a C++ program, I am working on a virtual C++ machine. When I am typing a letter, I am working on a virtual word processing machine. In reality, I am working on the same processor for each task, but in my mind I think of the computer as understanding two completely different languages.