Intro to Computing

Richard K. Herz, ReactorLab.net

What is a computer?
"A device that processes information"

INPUTS INFORMATION:

types of information:

numbers, text, images, sound, electrical voltages, position, motion

from:

keyboard, mouse; graphic tablet; touch screen; magnetic disks, "hard" and "floppy", other computers over networks; CD-ROM and DVD; video, "frame grabber", digital cameras; audio. sound digitizer, MIDI; sensors, chemical analyzers, pressure sensors, motion sensors, etc.; tape; ID cards, credit cards

PROCESSES INFORMATION:

calculate with numbers; format text, search text (and maybe image in future); analyze and transform graphic images; analyze and transform sound; make "decisions"

OUTPUTS INFORMATION:

types of info:

formatted text; tables of numbers; plots; images; modified and synthesized sound; electrical signals

to:

display screen; printers; speakers, sound synthesizers, MIDI controllers; video tape; read-write CDs and DVDs; control commands to actuators, robots, valves, switches, etc.

COMMUNICATES, *via* :

computer network:, ethernet, fiber-optic; computer - **Internet** - computer; radio - cellular data networks

---

Examples of Computers:

common "digital" computers:

Apple Macintosh; Intel and AMD PC's; Sun and HP workstations; supercomputers

other digital computers:

"morphing" and graphics in movies; video games; air-traffic control

"**Embedded computers**" or "**embedded systems**" are found inside other devices. They don't have the usual screen and keyboard but they are still computers. Devices which contain them are increasing in number and include: cell phones; iPods; sound synthesizers; air bag in car; air-fuel ratio in car

biological "neural network" computer:

Your brain! Roughly equivalent to one dozen Cray supercomputers (though an "apple and orange" comparison)!! Your brain is a "neural network." A neural network computer has a very different "architecture" than a digital computer. A neural network is composed of a pattern of connections between cells called neurons. There is no "program" - information is stored and processed by the **pattern of connections** in the network.

Architecture of a digital computer

Hardware:

CPU, "central processing unit," contains a small amount of memory locations (RAM) and logic units which perform the operations.

RAM, "random access memory," goes blank when the power is off

ROM, "read-only memory," info stays there when the power is off

storage memory (disk, CD-ROM, tape, etc), info stays there when the power is off

input devices: keyboard, disk, microphone, etc.

output devices: video screen, printer, disk, speaker, etc.

communication devices: ethernet, phone modems, etc.

Firmware:

Programs in ROM that are always accessible to the CPU. For example, when you turn on the computer, the CPU reads firmware programs from ROM that tell it how to get more programs from storage memory and load it into RAM.

Software:

Software, or "programs", lists of "instructions" and "data" that get information (data and other instructions) from memory, process it, and store the results in memory. Also instructions to get input, display and communicate output.

Software programs are recorded in storage memory and brought into RAM as needed.

Types of Software:

Application:

Software that you write or you buy that does something useful.
Examples: programs ***you*** will write, word processors (Microsoft Word, Wordperfect, etc.), spreadsheets, databases, games, programming languages (Basic, C, Fortran), etc.

Operating System or OS:

The inteface between the hardware and applications. Examples: Windows on Intel PCs and compatibles, Macintosh OS, Linux, Unix, etc.

Data:

lists of information encoded as **binary numbers**, this is information to be processed or that has been processed.

What are computer programs?

Set of instructions that tell the computer what to do.

Whenever you use anything that is a computer or has a computer in it, realize that someone wrote a program that tells that device what to do.

They probably sat at a computer **terminal** and "wrote" a **program**, a set of instructions, and stored the program on a **disk**.

First programmer: **Ada Lovelace**, who helped **Charles Babbage** design the first **mechanical** computers in the 1830s, and who devised the basis of what would later be called "programming." Many programs written for the U.S. military are written in a computer language called **Ada**, named after her.

Why learn to write computer programs?

Be able to do more - get more access to computer power

- get more information about a company's financial health
- analyze data from an opinion poll
- present results of a psychological study in a different way

Modify existing programs to get them to do what you want better

Be a producer instead of just a consumer - get paid!

Understand programmers in case you need to hire them and manage them.

Types or levels of computer programs

Low Level <-> High Level:

As you go up in level, get more type of commands and a single command does a more complex task. Since a single command can do a complex task, the higher the language, usually, the easier it is to write programs in.

Since we want to do fairly complex things, we will use a relatively high-level language - BASIC.

The levels:

Digital computers only operate on strings of 0s and 1s, so at the lowest level are "**machine language**" programs - strings of 0s and 1s. What all digital computers actually run. Hard to read and write: 0111 1011 1000 0100, etc.

Next up - "**assembly language**" - only a few simple commands, and often numbers in "hexadecimal" - 0 through 1 and ABCDEF (base 16). Easier to read than machine language but still not easy: STO 8E, AND 95, etc. "Assemblers" are programs that translate assembly language programs into machine language programs.

Next - "**high-level**" languages such as **BASIC**, Fortran, C, Pascal, Lisp, Forth, Ada, etc.

PRINT "INPUT A", INPUT A, C = A + B, etc.
"Compilers" are programs that translate high level languages into machine language.

At the **highest level** are programs that let you design software graphically, then translate your designs into a program that the computer can run:

LabView from National Instruments allows graphical programming of laboratory interfaces. There are several "integrated development environments (IDE)" which allow you to design graphical user interfaces by graphically, e.g., for languages such as Visual Basic, Java, Runtime Revolution.

Computers process encoded information

The types of information are diverse and a computer can only do its processing on one type - so the other types have to be translated into the appropriate type as they come into the computer.

In this class, we will talk mainly about one type of computer, which we will refer to as a **digital computer**. This is the most common type of computer that people think of when they hear the word "computer." Macintoshes, IBMs, PCs, Crays, are all digital computers.

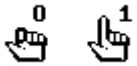
In digital computers, all information - "real world" quantities, images, etc and abstract mathematical information - is encoded in a hierarchy:

Characters (a, b, c, .., @, #) and decimal numbers
decimal or hexidecimal numbers
finally, binary numbers (0011 0100, 0010 0011)

Binary numbers

All operations done in computer on "0's and 1's"
The decimal number 0 in binary is 0000 0000
The decimal number 1 in binary is 0000 0001
The decimal number 2 in binary is 0000 0010
Etc.

Each "digit" or number is called a "**bit**". Latin *digitus* , finger



Eight bits are called a "**byte**". What are four bits called? (a "nibble")

Each bit represents either:

0 when its value is 0, **or a power of two** (2^n) when its value is 1.

The **right-most bit** in a byte is either 0 or $2^0 = 1$. The next one to the left is either 0 or $2^1 = 2$. The next one is either 0 or $2^2 = 4$, and so on.

The **power of two** represented by a digit can be obtained by starting at the far right of the byte and counting from zero.

The **value of a byte** is obtained by summing the values of its bits.

The letter or character "A" is represented first as the decimal number 65, and then in computer operations as the binary number 0100 0001. A special byte preceding this byte identifies this byte as representing a character rather than a number.

Why binary numbers?

Digital computers are made up of millions of transistors which function as "on - off" switches.

Each transistor can only represent the numbers 0 and 1.
The number 0 can be represented as the "off" state of a transistor.
The number 1 can be represented as the "on" state of the transistor.

Advantages of storing and processing binary numbers - relative to "analog" storage of information, e.g., in audio "records" or cassette tapes (see p. 178):

Insensitive to noise

Digital circuits only have to detect a relatively low or high voltage state in the presence of noise to detect a 0 or 1. Analog outputs are directly proportional to the voltage signal carrying the

information, so noise in the voltage signal becomes part of the output and can only be partially removed by complex filtering circuits.

Error detection and correction possible

Since information is stored as numbers, mathematical algorithms can be used to detect errors and correct them.

High density of information storage and communication

Since the physical devices which store 0s and 1s are very simple, they can be made very small and packed together to get a high density of 0s and 1s in storage media.

Also, since information is stored as numbers, mathematical algorithms can be used to "compress" information so less storage media is required and communication media can carry more info.

Examples of 0 and 1s:

in RAM: capacitor is charged or uncharged

on magnetic disk: direction of N-S poles of domain of chromium or iron oxide

on CD-ROM or optical disk: pits in shiny film that scatter laser beam

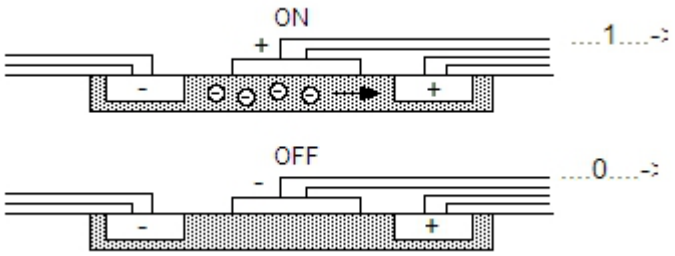
fiber optic cables: light pulses

Physical representations of BITS

Value of a **BIT**, **0** or **1**, can be represented **physically** as:

Switch: OFF or ON

transistor in CPU and RAM:



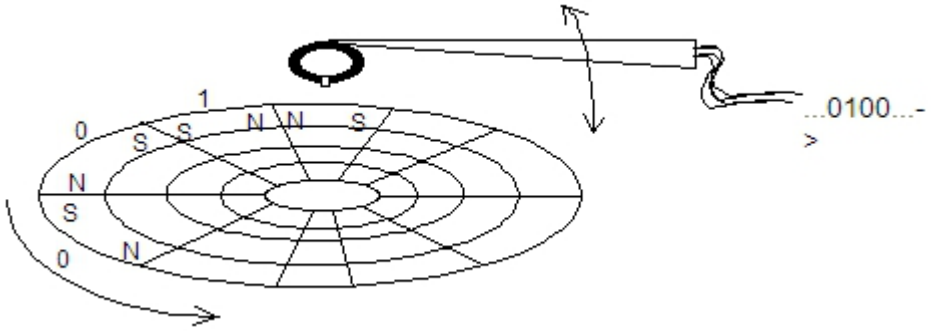
The circles with minus signs in them in the top figure represent electrons flowing in the silicon material on top of which the transistor is formed. The other plus and minus signs represent electrical voltages. Notice especially the plus sign near "ON" in the top figure and the minus sign near "OFF" in the lower figure.

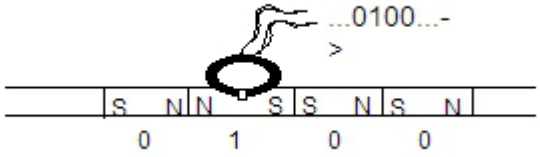
These transistors are very small, a few millionths of a meter in size. Many thousands of these transistors are formed to make electrical circuits on the surface of a silicon crystal. They are formed using a process called "photolithography".

Photolithography is also used in printing magazines except the size scale is much smaller with the electrical circuits and the materials are different: silicon instead of paper, and chemical elements such as phosphorus, boron, silicon and aluminum instead of different ink colors.

Direction of Pole of Magnetic "Domain": NORTH or SOUTH

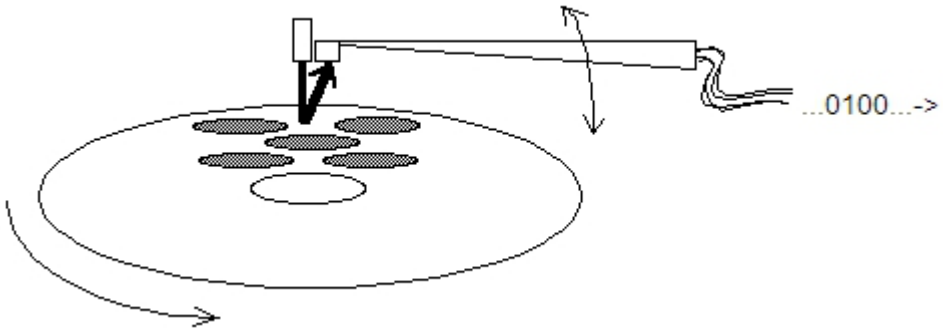
hard disk, magnetic tape:





Surface Coating: NON-REFLECTIVE or REFLECTIVE

CD-ROM:



Light Pulse: OFF or ON

fiber optic networks
your TV and VCR controllers

Radio Frequency: LOW or HIGH

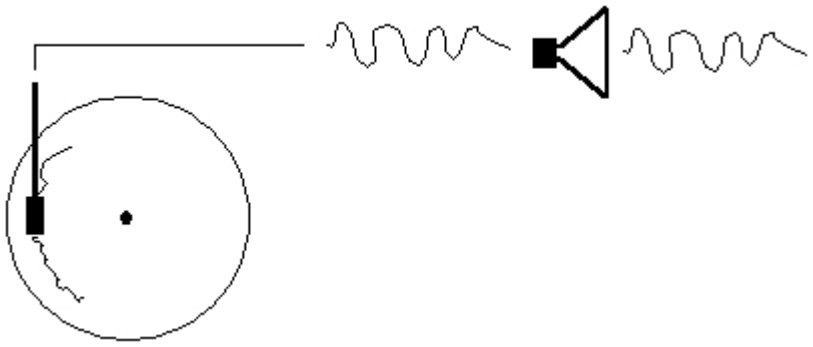
digital cellular radio

Analog storage and processing of information

Greek *analogos* , proportionate. Examples:

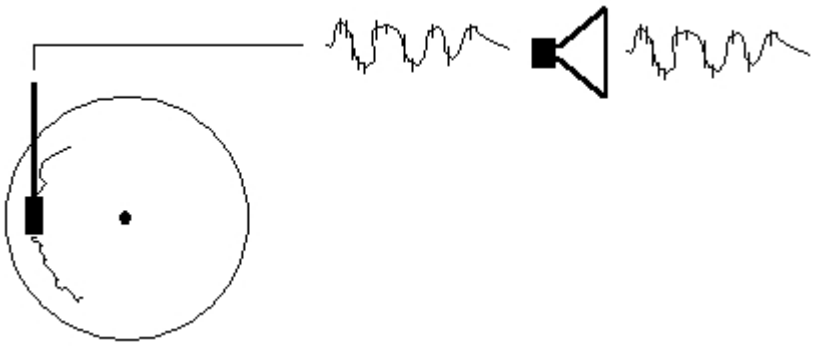
Images recorded in photographic film.

Sound waves in voice and music stored as "proportionate" wiggly grooves in audio "records" (the old black plastic ones)

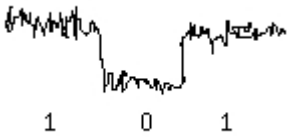


"Noise" is a bigger problem with analog storage than digital

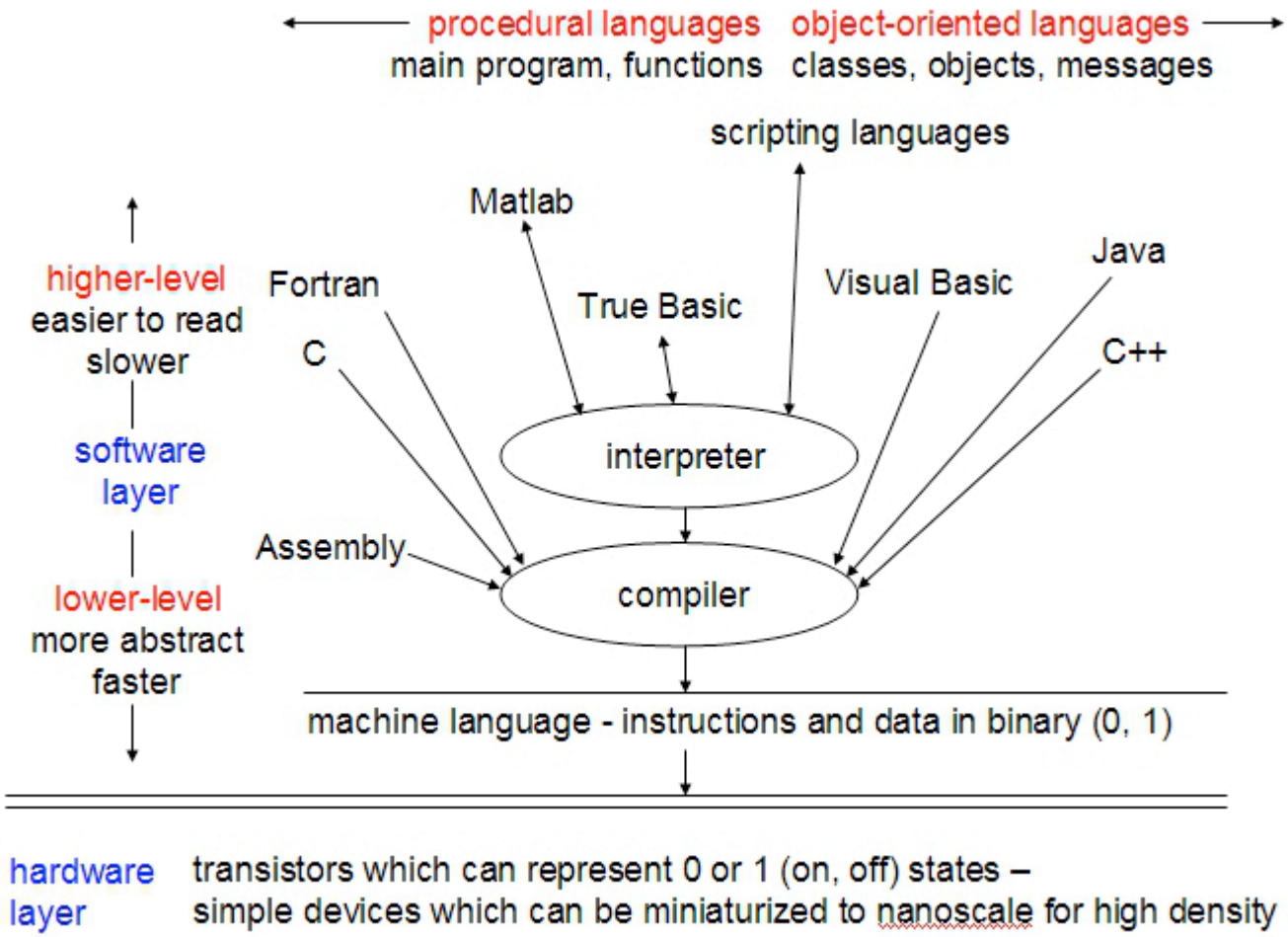
Noise can be reduced but not eliminated from analog signals



Digital 0's and 1's are easy to detect in the presence of noise: only care whether voltage is "high" or "low". Once the signals are converted to numbers, mathematics can be used to detect and correct any errors



Here is a sketch of the relationships between computer hardware and various computer languages:



The compiler is a program that takes instructions written in the programming language and converts it into machine language (in binary 0's and 1's) that can interact directly with the hardware layer. A machine language program will only run on the specific type of hardware platform it was compiled on, e.g., Intel Pentium PC, IBM Power PC, Sun Solaris, etc. If you have a lot of time and are really patient, you can write programs directly in machine language, as in the flight navigation computer for the first F14 fighter plane, which used an early form of integrated circuit computer. Some early computers had to be programmed in machine language by literally flipping a line of switches up or down (1 or 0) to enter each binary instruction or data element separately, e.g., Digital Equipment Corp PDP-1 circa 1960. Even earlier in the 1940s, ENIAC, the first large-scale electronic computer, used vacuum tubes, not transistors, and "programming" consisted of rewiring the computers electrical circuits!

Java and the scripting language LiveCode are compiled in a two-step process to provide for cross-platform deployment, where cross-platform means the intermediate file produced by the first step will run on PCs, Macs, or Unix computers, etc. In the first step, the source code is compiled to cross-platform byte-code. A byte-code file can be transferred to any computer platform. In step 2, the byte-code is compiled on a "virtual machine" to the machine language of the computer hardware being used.

Note the double-headed arrows between True Basic and the Interpreter and between Matlab and the Interpreter. The Matlab interpreter is a compiled program - the Matlab development environment - which interacts with the programmer during program development, running each line as it as entered or read from a source code file. This speeds program development but slows program execution. On the homework programs you write for

other classes, the slowdown won't be noticed on a 2+ GHz machine! Plus, Matlab has a compiler option that can compile a Matlab program into C when you are through with development of a really big project.

Procedural languages are developed by writing code to represent the actions to be carried out in the project. They consist of a main program section and other modules such as functions (all languages) and subroutines (Fortran, True Basic) which allow a project to be constructed from reusable modules. Procedural languages are useful in engineering for doing "number crunching" such as calculating stresses on a mechanical part.

Object-oriented languages are developed by writing code to describe classes of objects and handle messages which are received from the outside (user, Internet) and which are passed between objects. Software objects consist of (a) data and (b) methods which operate on the data and which interact with other objects. Object-based languages are similar to object-oriented languages but have pre-defined classes of objects such as user-interface controls (buttons, fields) which are sufficient for many projects. Object-oriented languages, are used to use to write projects with a close mapping to real objects and processes. They are also used to write Graphical User Interfaces (GUIs) in projects with a lot of interaction with the computer mouse.

A scripting language is a language that does not require variable-type declaration and which is compiled to a byte code or is interpreted rather than compiled. Scripting languages usually provide less or no control over bit- and byte-level processes than do programming languages but have the advantage that many types of high-level processes can be accomplished in many fewer lines of code than would be required in a programming language. JavaScript, PHP, Perl, Tcl, and Revolution are examples of the scripting languages. Different scripting languages have differing levels of object-orientation and ease of writing GUIs.