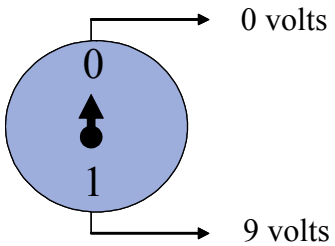
CS101
Binary Storage Devices and Boolean Logic

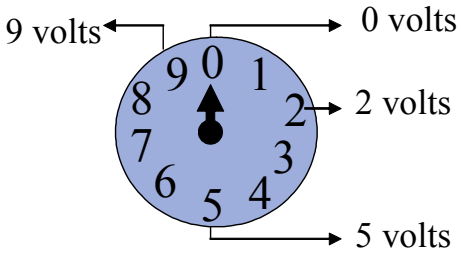
Now that we have discussed number representation, why do computers use the binary representation and not something we are more familiar with, like decimal?

The main reason is that binary is cheap and it is reliable. A binary 0 or 1 will generally be stored as some sort of electrical charge. For example, say we have a binary device that represents a 0 by storing an electrical charge of 0 volts, and a value of 1 is stored with an electrical charge of 9 volts:



There is a good chance of a little error; let's say that the device is really charged to 8 volts instead of 9 volts. That's ok, because a binary 1 is still closer to 8 volts than to a binary 0. We wouldn't confuse a 0 and a 1 until we dropped all the way down to 4.5 volts.

Instead, let's say that our device is storing 10 digits instead of 2 digits. We could do this by storing varying amounts of charge; a charge of 0 volts represents digit 0, a charge of 1 volts represents digit 1, a charge of 2 volts represents digit 2, etc. :

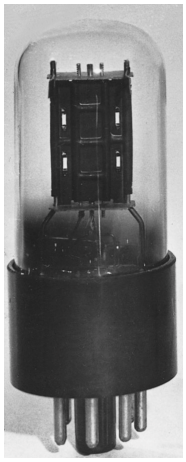


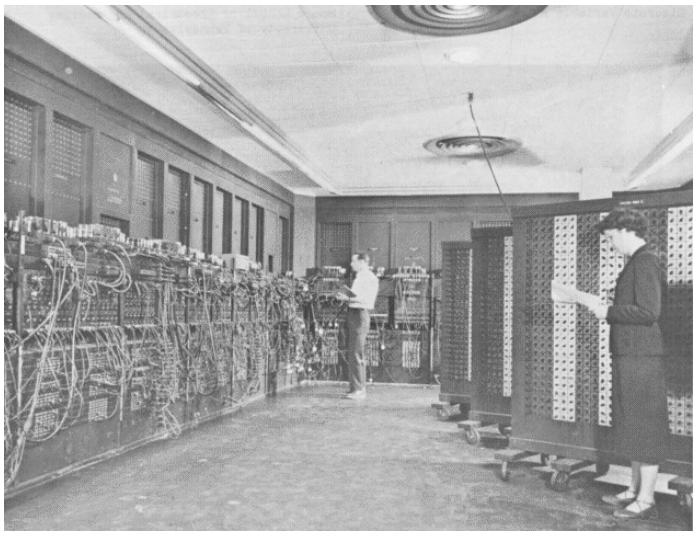
Although we are storing more digits on the device than with the binary system, there is less room for error. If we meant to store the digit "9" but somehow our device is not charged correctly and stores a voltage of 8.3 instead, then when we go to read the value stored in the device we would get the digit "8" instead. In contrast, there is more room for error on the binary device since we are only storing two digits with our voltage range. This makes binary less susceptible to error and noise than analog storage devices (e.g., clear digital audio or music vs. analog audio).

How is the binary charge actually stored on the computer?

In the 40's and 50's, computers used **vacuum** tubes. A vacuum tube operates in a manner similar to a lightbulb. In a lightbulb, electrical current flows across the bulb giving out light in the process. In contrast, the vacuum tube has a plate in the center that provides a mechanism to control the current. If the plate is charged one way, the current flow will stop. If the plate is charged oppositely, the current continues. This in effect is an electronically controlled switch, where the switch can represent a binary 0 or 1.

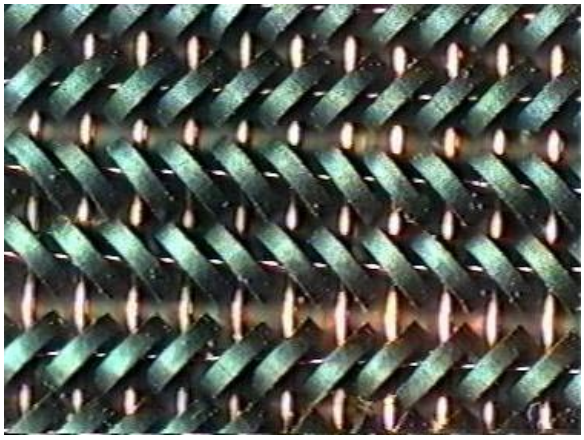
A major problem of vacuum tubes is that they take up a lot of space and give out a ton of heat. The ENIAC computer shown below from 1947 required over 18,000 vacuum tubes and occupied an entire room.





In the 60's and 70's, computers used **core** memory. Core memory stored data magnetically, much as you would store data on a cassette recorder. Each bit was stored on a magnetized "doughnut". If the magnetic field was clockwise, the core represented a zero. If the magnetic field was counter-clockwise, the core stored a one. Wires through the center of the core could set the magnetic field (i.e. store a one or a zero) or detect the orientation of the magnetic field (i.e. read whether a one or a zero was stored).

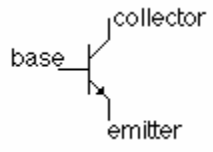
Each doughnut of the core memory was organized in a big two-dimensional grid that allowed direct access to each cell in the memory:



One problem of core memory is that it was slow and also occupied a large amount of physical space. In contrast, today's computer memories are based on **transistors**. The transistor was invented in 1947 at Bell Labs by Bardeen, Brattain, and Shockley and is essentially a very tiny switch created in silicon.

The transistor was a transforming invention! Ultimately it would allow devices to be constructed millions of times smaller and millions of times cheaper than before. The first transistor was about the size of a thumb, with a paperclip, gold foil, germanium, and coiled wire. Germanium is, like silicon, a semiconductor – it only conducts a trickle of electricity. This can be used to control the amplification of signals by changing the material from a conductor to an insulator. The discovery of the transistor was actually an accident – there is some evidence that Shockley was annoyed and was looking for something else. It took until 1948 to see the real value and in 1951 you could license transistor technology for $25K. Then Shockley went on to start the seeds of Silicon Valley.

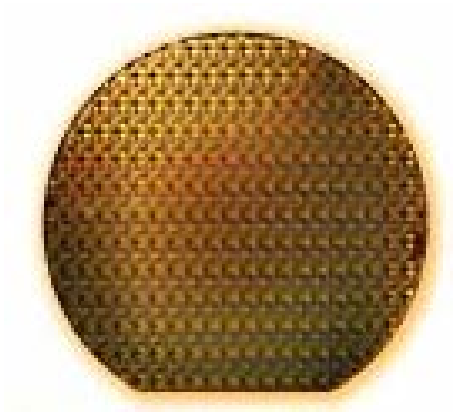
A schematic of a transistor is shown below. Electrons flow from the emitter to the collector. By varying the current on the base, the amount of current flowing between the collector and emitter can be regulated (e.g. stopped to represent a zero, or flowing to represent a one).



The next improvement in memory technology came with the **integrated circuit**, or IC, invented by Jack Kilby and Bob Noyce at Texas Instruments in 1964. The IC is a way to package together multiple transistors on the same piece of silicon. This allows more transistors to be placed in the same amount of space, requires less energy, outputs less heat, and runs faster since there is a shorter distance for electrons to travel.

The creation of an IC begins with an ingot of purified silicon:

This ingot is sliced with a diamond saw into thin wafers. The transistors are then "etched" into the wafer using a lithography process similar to photography. Masks are laid over the wafers, ultraviolet light is shone through the mask, and the chip is built up in layers on the silicon wafer. The entire process takes about three months and the end result is multiple chips etched onto a single wafer. Here is a cross-section of a sample wafer:



Finally, each chip is cut out of the wafer and packaged into a plastic case with metal connectors that you are probably familiar with:

As process technology improves, engineers have been able to put more and more transistors onto a single chip. We are currently at the point where approximately 10 million transistors can be placed into a square centimeter! This technology is called VLSI, or Very Large Scale Integration. Thanks to integration improvements, our

microprocessors have been running faster and faster. The more transistors we can pack together, the shorter distance electrons have to travel and the more performance we get, resulting in faster computers.

The degree of integration is often measured in microns, where a micron is one millionth of a meter. The number of microns is used to denote how far apart we can place transistors on the chip, so a smaller number means we can pack more transistors in the same chip. The micron process technology has been dropping over the years:

- Human Hair: 100 microns wide
- Bacterium: 5 microns
- Virus: 0.8 microns
- Early microprocessors: 10-15 micron technology
- 1997: 0.35 Micron
- 1998: 0.25 Micron
- 1999: 0.18 Micron
- 2001: 0.13 Micron
- 2002: 0.11 Micron

The physical limits are believed to be around 0.06 Microns, so we still have a ways to go!

Boolean Logic

The construction of computer circuits, and to a large degree, of software programs, is based on a branch of symbolic logic called Boolean logic, named after English mathematician George Boole. Boolean logic manipulates only two values: true and false.

We have actually already used Boolean logic in describing algorithms.

For example, let's say that we store the value 5 into variable *x*. We can now make Boolean expressions regarding *x*:

The Boolean expression: x equals 4
is false, because *x* equals 1, not 4.

The Boolean expression: $x < 4$
is true, because 1 is less than 4.

The Boolean expression: $x > 4$
is false, because 1 is not greater than 4.

Boolean expressions become more interesting when we combine multiple expressions. The basic operators to combine Boolean expressions are AND, OR, and NOT.

AND is a binary Boolean operator, meaning it requires two Boolean values (operands). If both operands are true, the result is true. Otherwise, the result is false.

	True	**False**
True	True	False
False	False	False

Example: (1<4) AND (4<3) is false because we are AND'ing true and false which results in false.

(1<4) AND (3<4) is true because we are AND'ing true and true, which results in true.

OR is a binary Boolean operator. If at least one of the operands is true, the result is true. Otherwise, the result is false.

	True	**False**
True	True	True
False	True	False

Example: (1<4) OR (4<3) is true because we are AND'ing true and false which results in true.

NOT is a unary Boolean operator – it requires only one Boolean value to operate on. NOT changes the value of its operand: If the operand is true, the result is false; if the operand is false, the result is true.

	Not Value
True	False
False	True

Example: NOT (1 < 4) is false, because (1 < 4) is true, and by NOT'ing it, we get the opposite of true, which is false.

Examples: Assume that x = 1 and y = 2:

1. (x equals 1) AND (y equals 3)

   evaluates:
   (true) AND (false) → FALSE

   All operands must be true for the AND to become true.

2. (x < y) OR (x > 1) OR (y < 1)

   evaluates:
   (true) OR (false) OR (false) → TRUE

   Only one operand must be true for the OR to become true.

3. NOT ((x equals 1) AND (y equals 3))

   evaluates:
   NOT ((true) AND (false))
   NOT (false) → TRUE

4. ((x equals 1) AND ((y equals 3) OR (x < y))

   evaluates:
   ((true) AND ((false) OR (true)))
   ((true) AND (true)) → TRUE