

Bits and Bytes: The Language of Computers





How do computers process data into information?

Unlike humans, computers work exclusively with numbers (not words). To process data into information, computers need to work in a language they understand. This language, called binary language, consists of just two digits: 0 and 1. Everything a computer does, such as processing data, printing a report, or editing a photo, is broken down into a series of 0s and 1s. Each 0 and 1 is a binary digit, or bit for short. Eight binary digits (or bits) combine to create one byte. In computers, each letter of the alphabet, each number, and each special character (such as @, pronounced “at”) consists of a unique combination of eight bits, or a string of eight 0s and 1s. So, for example, in binary language, the letter *K* is represented as 01001011. This is eight bits, or one byte.

What else can bits and bytes be used for?

Bits and bytes not only are used as the language that tells the computer what to do, they are also used to represent the *quantity* of data and information that the computer inputs and outputs. Word processing files, digital pictures, and even software are represented inside computing devices as a series of bits and bytes. These files and applications can be quite large, containing thousands or millions of bytes.

To make it easier to measure the size of such files, we need units of measure larger than a byte. Kilobytes, megabytes, and gigabytes are therefore simply larger amounts of bytes.

How Much Is a Byte?		
NAME	NUMBER OF BYTES	RELATIVE SIZE
Byte (B)	1 byte	One character of data (8 bits, or binary digits)
Kilobyte (KB)	1,024 bytes (2^{10} bytes)	1,024 characters, or about 1 page of plain text
Megabyte (MB)	1,048,576 bytes (2^{20} bytes)	About 4 books (200 pages, 240,000 characters) 
Gigabyte (GB)	1,073,741,824 bytes (2^{30} bytes)	About 4,500 books, or over twice the size of Sir Isaac Newton's library (considered very large for the time) 
Terabyte (TB)	1,099,511,627,776 bytes (2^{40} bytes)	About 4.6 million books, or about the number of volumes in the Rutgers University Library
Petabyte (PB)	1,125,899,906,842,624 bytes (2^{50} bytes)	About 4.7 billion books, which would fill the Library of Congress (the United States' largest library) 140 times! 
Exabyte (EB)	1,152,921,504,606,846,976 bytes (2^{60} bytes)	About 4.8 trillion books, which, if stored as they are in the Library of Congress, would occupy about 11,000 square miles, or an area almost the size of the state of Maryland
Zettabyte (ZB)	1,180,591,620,717,411,303,424 bytes (2^{70} bytes)	The library required to house the 4.9 quadrillion books equal to a ZB of data would occupy about 11.3 million square miles, or an area about 1 million square miles larger than all of North America 

As shown in Figure 2.2, a kilobyte (KB) is approximately 1,000 bytes, a megabyte (MB) is about 1 million bytes, and a gigabyte (GB) is around 1 billion bytes. Today, personal computers are capable of storing terabytes (TB) of data (around 1 trillion bytes), and many business computers can store up to a petabyte (PB) (1,000 terabytes) of data. The Google search engine processes more than 1 PB of user-generated data per *hour*!

How does your computer process bits and bytes?

Your computer uses hardware and software to process data into information that lets you complete tasks such as writing a letter or playing a game. Hardware is any part of the computer you can physically touch. However, a computer needs more than just hardware to work. Software is the set of computer programs that enables the hardware to perform different tasks. There are two broad categories of software: *application software* and *system software*. Application software is the set of programs you use on a computer to help you carry out tasks such as writing a research paper. If you've ever typed a document, created a spreadsheet, or edited a digital photo, for example, you've used application software.

System software is the set of programs that enables your computer's hardware devices and application software to work together. The most common type of system software is the operating system (OS)—the program that controls how your computer system functions. It manages the hardware, such as the monitor and printer, and provides a means by which users can interact with the computer. Most likely, the computer you own or use at school runs a version of Windows as the system software. However, if you're working on an Apple computer, you're probably running OS X.

Switches

How does a computer process the data you input? A computer system can be viewed as an enormous collection of on/off switches. These simple on/off switches are combined in different ways to perform addition and subtraction and to move data around the system.

Electrical Switches

To process data into information, computers need to work in a language they understand. Computers understand only two states of existence: on and off. Inside a computer, these two possibilities, or states, are defined using the two numbers *1* and *0*; the language represented by these numbers is called binary language because just two numbers are used. Everything a computer does, such as processing data or printing a report, is broken down into a series of *0*s and *1*s. Electrical switches are the devices inside the computer that are flipped between the two states of *1* and *0*, signifying “on” and “off.”

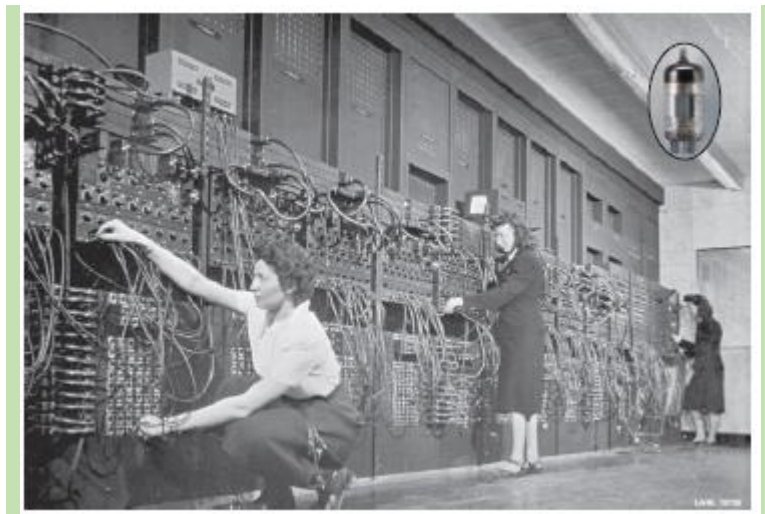
You use various forms of switches every day. The light switch in your kitchen either is ON, allowing current to flow to the light bulb, or OFF. Another switch you use each day is a water faucet. As shown in Figure 1, shutting off the faucet so that no water flows could represent the value *0*, whereas turning it on could represent the value *1*.



Computers are built from a huge collection of electrical switches. The history of computers is really a story about creating smaller and faster sets of electrical switches so that more data can be stored and manipulated quickly.

Vacuum Tubes. The earliest generation of electronic computers used vacuum tubes as switches. Vacuum tubes act as switches by allowing or blocking the flow of electrical current.

The problem with vacuum tubes is that they take up a lot of space, as shown in Figure 2.



The first high-speed digital computer, the Electronic Numerical Integrator and Computer (ENIAC), was deployed in 1945. It used nearly 18,000 vacuum tubes as switches and was about half the size of a basketball court! In addition to being large, the vacuum tubes produced a lot of heat and burned out frequently.

Since the introduction of ENIAC's vacuum tubes, two major revolutions have occurred in the design of switches, and consequently computers, to make them smaller and faster:

1. The invention of the *transistor*
2. The fabrication of *integrated circuits*

Transistors. Transistors are electrical switches built out of layers of a special type of material called a semiconductor. A semiconductor is any material that can be controlled either to conduct electricity or to act as an insulator (to prohibit electricity from passing through). Silicon, which is found in common sand, is the semiconductor material used to make transistors (see Figure 3).



By itself, silicon doesn't conduct electricity particularly well, but if specific chemicals are added in a controlled way to the silicon, it begins to behave like a switch. The silicon allows electrical current to flow easily when a certain voltage is applied; otherwise, it prevents electrical current from flowing, thus behaving as an on/off switch. This kind of behavior is exactly what's needed to store digital information—the 0s (off) and 1s (on) of binary language.

The first transistors were much smaller than vacuum tubes, produced little heat, and could quickly be switched from on to off, thereby allowing or blocking electrical current. They also were less expensive than vacuum tubes.

It wasn't long, however, before transistors reached their limits. Continuing advances in technology began to require more transistors than circuit boards could reasonably handle. Something was needed to pack more transistor capacity into a smaller space. Thus, integrated circuits, the next technical revolution in switches, were developed.

Integrated Circuits. Integrated circuits (or chips) are tiny regions of semiconductor material that support a huge number of transistors (see Figure 4). Most integrated circuits are no more than a quarter inch in size yet can hold billions of transistors.

This advancement has enabled computer designers to create small yet powerful microprocessors, which are the chips that contain a central processing unit (CPU). The Intel 4004, the first complete microprocessor to be located on a single integrated circuit, was released in 1971, marking the beginning of the true miniaturization of computers. The Intel 4004 contained slightly more than 2,300 transistors.

Today, more than 2 billion transistors can be manufactured in a space as tiny as the nail of your little finger!

Number Systems

How can simple switches be organized so that they let you use a computer to pay your bills online or write an essay? How can a set of switches describe a number or a word or give a computer the command to perform addition?

Recall that to manipulate the on/off switches, the computer works in binary language, which uses only two digits, *0* and *1*. Let's look at how numbering systems work so that we can begin to understand this more deeply.

The Base-10 Number System

A number system is an organized plan for representing a number. Although you may not realize it, you're already familiar with one number system. The base-10 number system, also known as decimal notation, is the system you use to represent all of the numeric values you use each day. It's called base 10 because it uses 10 digits—0 through 9—to represent any value.

To represent a number in base 10, you break the number down into groups of ones, tens, hundreds, thousands, and so on. Each digit has a place value depending on where it appears in the number. For example, using base 10, in the whole number 6,954 there are 6 sets of thousands, 9 sets of hundreds, 5 sets of tens, and 4 sets of ones. Working from right to left, each place in a number represents an increasing power of 10, as follows:

$$\begin{aligned} 6,954 &= (6 * 1,000) + (9 * 100) + (5 * 10) + (4 * 1) \\ &= (6 * 10^3) + (9 * 10^2) + (5 * 10^1) + (4 * 10^0) \end{aligned}$$

Note that in this equation, the final digit 4 is represented as $4 * 100$ because any number raised to the zero power is equal to 1.

The Base-2 (or Binary) Number System

Anthropologists theorize that humans developed a base-10 number system because we have 10 fingers. However, computer systems are not well suited to thinking about numbers in groups of 10. Instead, computers describe a number in powers of 2 because each switch can be in one of two positions: on or off. This numbering system is referred to as the binary number system.

The binary number system is also referred to as the base-2 number system. Even with just two digits, the binary number system can still represent all the values that a base-10 number system can. Instead of breaking the number down into sets of ones, tens, hundreds, and thousands, as is done in base-10 notation, the binary number system describes a number as the sum of powers of 2—ones, twos, fours, eights, and sixteens. Binary numbers are used to represent every piece of data stored in a computer: all of the numbers, all of the letters, and all of the instructions that the computer uses to execute work.

Representing Integers. In the base-10 number system, a whole number is represented as the sum of *1s*, *10s*, *100s*, and *1,000s*—that is, sums of powers of 10. The binary system works in the same way, but describes a value as the sum of groups of *1s*, *2s*, *4s*, *8s*, *16s*, *32s*, *64s*, etc.—that is, powers of 2: 1, 2, 4, 8, 16, 32, 64, and so on.

Let's look at the number 67. In base 10, the number 67 would be six sets of *10s* and seven sets of *1s*, as follows:

$$\text{Base 10: } 67 = (6 * 10^1) + (7 * 10^0)$$

One way to figure out how 67 is represented in base 2 is to find the largest possible power of 2 that could be in the number 67. Two to the eighth power is 256, and there are no groups of 256 in the number 67. Two to the seventh power is 128, but that is bigger than 67. Two to the sixth power is 64, and there is a group of 64 inside a group of 67.

67 has	1	group of	64	That leaves 3 and
3 has	0	groups of	32	
	0	groups of	16	
	0	groups of	8	
	0	groups of	4	
	1	group of	2	That leaves 1 and
1 has	1	group of	1	Now nothing is left

So, the binary number for 67 is written as 1000011 in base 2:

$$\begin{aligned} \text{Base 2: } 67 &= 64 + 0 + 0 + 0 + 0 + 2 + 1 \\ &= (1 * 2^6) + (0 * 2^5) + (0 * 2^4) + (0 * 2^3) + \\ &\quad (0 * 2^2) + (1 * 2^1) + (1 * 2^0) \\ &= (1000011)_{\text{base 2}} \end{aligned}$$

It's easier to have a calculator do this for you! Some calculators have a button labeled DEC (for decimal) and another labeled BIN (for binary). Using Windows 8, you can access the Scientific Calculator that supports conversion between decimal (base 10) and binary (base 2) by accessing Search from the Charms bar and typing "calc" on the Start screen. From the App search results, choose the first Calculator on the search results.

From the Calculator menu, select View and then Programmer. You can enter your calculation in decimal and instantly see the binary representation in 64 bits.

Hexadecimal Notation. A large integer value becomes a very long string of *1*s and *0*s in binary! For convenience, programmers often use hexadecimal notation to make these expressions easier to use. Hexadecimal is a base-16 number system, meaning it uses 16 digits to represent numbers instead of the 10 digits used in base 10 or the 2 digits used in base 2. The 16 digits it uses are the 10 numeric digits, 0 to 9, plus six extra symbols: A, B, C, D, E, and F. Each of the letters A through F corresponds to a numeric value, so that A equals 10, B equals 11, and so on (see Figure 7). Therefore, the value 67 in decimal notation is 1000011 in binary or 43 in hexadecimal notation.

It is much easier for computer scientists to use the 2-digit 43 than the 7-digit string 1000011. The Windows Calculator in Programmer view also can perform conversions to hexadecimal notation. (You can watch a video that shows you how to perform conversions between bases using the Windows Calculator in the Sound Byte titled, “Where Does Binary Show Up?”)

Representing Characters: ASCII. We’ve just been converting integers from base 10, which *we* understand, to base 2 (binary state), which the computer understands. Similarly, we need a system that converts letters and other symbols that *we* understand to a binary state the computer understands. To provide a consistent means for representing letters and other characters, certain codes dictate how to represent characters in binary format. Most of today’s personal computers use the American National Standards Institute (ANSI, pronounced “AN-see”) standard code, called the American Standard Code for Information Interchange (ASCII, pronounced “AS-key”), to represent each letter or character as an 8-bit (or 1-byte) binary code.

Each binary digit is called a bit for short. Eight binary digits (or bits) combine to create one byte. We’ve been converting base-10 numbers to a binary format. In such cases, the binary format has no standard length. For example, the binary format for the number 2 is two digits (10), whereas the binary format for the number 10 is four digits (1010). Although binary numbers can have more or fewer than 8 bits, each single alphabetic or special character is 1 byte (or 8 bits) of data and consists of a unique combination of a total of eight *0*s and *1*s.

The ASCII code represents the 26 uppercase letters and 26 lowercase letters used in the English language, along with many punctuation symbols and other special characters, using 8 bits. Figure 8 shows several examples of the ASCII code representation of printable letters and characters.

Representing Characters: Unicode. Because it represents letters and characters using only 8 bits, the ASCII code can assign only 256 (or 2 power 8) different codes for unique characters and letters. Although this is enough to represent English and many other characters found in the world’s languages, ASCII code can’t represent all languages and symbols, because some languages require more than 256 characters and letters. Thus, a new encoding scheme, called Unicode, was created. By using 16 bits instead of the 8 bits used in ASCII, Unicode can represent nearly 1,115,000 code points and currently assigns more than 96,000 unique character symbols.

The first 128 characters of Unicode are identical to ASCII, but because of its depth, Unicode is also able to represent the alphabets of all modern and historic languages and notational systems, including such languages and writing systems as Tibetan, Tagalog, and Canadian Aboriginal

syllabics. As we continue to become a more global society, it's anticipated that Unicode will replace ASCII as the standard character formatting code.

Representing Decimal Numbers. The binary number system can also represent a decimal number. How can a string of *1*s and *0*s capture the information in a value such as 99.368? Because every computer must store such numbers in the same way, the Institute of Electrical and Electronics Engineers (IEEE) has established a standard called the *floating-point standard* that describes how numbers with fractional parts should be represented in the binary number system. Using a 32-bit system, we can represent an incredibly wide range of numbers.

The method dictated by the IEEE standard works the same for any number with a decimal point, such as the number -0.75 .

The first digit, or bit (the sign bit), is used to indicate whether the number is positive or negative. The next 8 bits store the magnitude of the number, indicating whether the number is in the hundreds or millions, for example. The standard says to use the next 23 bits to store the value of the number.

Sample Hexadecimal Values		
DECIMAL NUMBER	BINARY VALUE	HEXADECIMAL VALUE
00	0000	00
01	0001	01
02	0010	02
03	0011	03
04	0100	04
05	0101	05
06	0110	06
07	0111	07
08	1000	08
09	1001	09
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Interpretation. *All* data inside the computer is stored as bits:

- Positive and negative integers can be stored using signed integer notation, with the first bit (the sign bit) indicating the sign and the rest of the bits indicating the value of the number.
- Decimal numbers are stored according to the IEEE floating-point standard.
- Letters and symbols are stored according to the ASCII code or Unicode.

These number systems and codes exist so that computers can store different types of information in their on/off switches. No matter what kind of data you input in a computer—a color, a musical note, or a street address—that data will be stored as a pattern of *1*s and *0*s. The important lesson is that the interpretation of *0*s and *1*s is what matters. The same binary pattern could represent a positive number, a negative number, a fraction, or a letter.

How does the computer know which interpretation to use for the *1*s and *0*s? When your mind processes language, it takes the sounds you hear and uses the rules of English, along with other clues, to build an interpretation of the sound as a word. If you're in New York City and hear someone shout, "Hey, Lori!" you expect that someone is saying hello to a friend. If you're in London and hear the same sound—"Hey! Lorry!"—you jump out of the way because a truck is coming at you! You knew which interpretation to apply to the sound because you had some other information—the fact that you were in England.

Likewise, the CPU is designed to understand a specific language or set of instructions. Certain instructions tell the CPU to expect a negative number next or to interpret the following bit pattern as a character. Because of this extra information, the CPU always knows which interpretation to use for a series of bits.

How the CPU Works

Any program you run on your computer is actually a long series of binary code describing a specific set of commands the CPU must perform. These commands may be coming from a user's actions or may be instructions fed from a program while it executes.

Each CPU is somewhat different in the exact steps it follows to perform its tasks, but all CPUs must perform a series of similar general steps. These steps are referred to as a CPU machine cycle (or processing cycle):

1. **Fetch:** When any program begins to run, the *1*s and *0*s that make up the program's binary code must be "fetched" from their temporary storage location in random access memory (RAM) and moved to the CPU before they can be executed.
2. **Decode:** Once the program's binary code is in the CPU, it is decoded into commands the CPU understands.
3. **Execute:** Next, the CPU actually performs the work described in the commands. Specialized hardware on the CPU performs addition, subtraction, multiplication, division, and other mathematical and logical operations.
4. **Store:** The result is stored in one of the registers, special memory storage areas built into the CPU, which are the most expensive, fastest memory in your computer. The CPU is then ready to fetch the next set of bits encoding the next instruction.

No matter what program you're running and no matter how many programs you're using at one time, the CPU performs these four steps over and over at incredibly high speeds. Shortly, we'll look at each stage in more detail. But first, let's examine components that help the CPU perform its tasks.

The Control Unit

The CPU, like any part of the computer system, is designed from a collection of switches. How can the simple on/off switches of the CPU “remember” the fetch-decode-execute store sequence of the machine cycle?

The control unit of the CPU manages the switches inside the CPU. It is programmed by CPU designers to remember the sequence of processing stages for that CPU and how each switch in the CPU should be set (i.e., on or off) for each stage.

With each beat of the system clock, the control unit moves each switch to the correct on or off setting and then performs the work of that stage.

References