

A World of 1's and 0's

Paul S. Wang, Sofpower.com

May 16, 2023

We live in a world dominated by information technology (IT). IT is everywhere and enriches our lives in countless ways. At the very heart of IT is the digital computer. Digital computers use 1's and 0's, nothing else, to represent and store information. There are no exceptions—all data and all programming are coded in 1's and 0's. In other words, computers do everything using only 1's and 0's (Figure 1). How incredible? How fascinating? Inside a computer, it is a world of 1's and 0's. But how could this work?

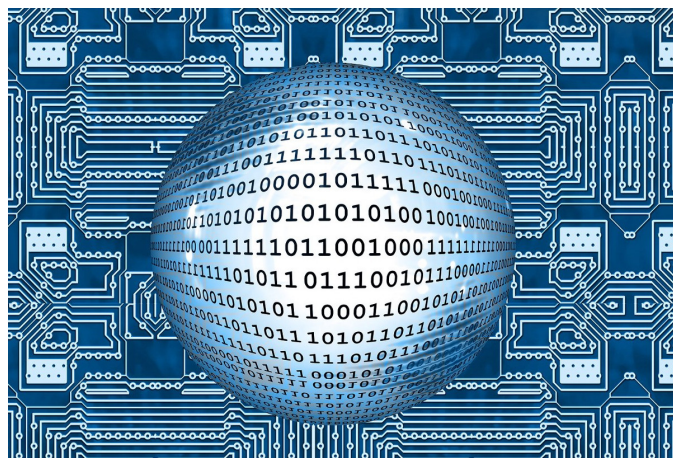


Figure 1: A World of Zeros and Ones

How can 1's and 0's play music or stream movies? Understand speech? Land men on the Moon? Operate driver-less cars?

At this point, some readers may think “this is way over my head” and tune out. Please don't. Just continue and you'll find it very interesting and rewarding.

We use a two-step approach:

1. Simply explain why computers use only 1's and 0's.
2. In common terms, show how 1's and 0's make everything work.

Digital Hardware

Modern computers process digital signals represented by the presence or absence of an electric current or voltage. Such a signal is the smallest unit of data inside a computer and is known as a *bit* (binary digit). A bit can represent one of two opposing states: on/off, yes/no, yin/yang, up/down, left/right, true/false, and, of course, 1/0. There are many other possibilities but we conventionally refer to these two states as 1 and 0, the two binary digits. A group of 8 bits is called a *byte* and a group of bytes, usually 4 or 8 bytes depending on the computer, is called a *word*.

The *central processing unit* (CPU) is the brain of a computer where information gets processed. A modern CPU, on a fingernail sized silicon chip, may contain billions of transistors—fast, tiny (about 70 silicon atoms, certainly invisible to the naked eye), and cheap devices to store and process electronic signals.

Typically, the CPU (Figure 2) performs operations by retrieving and storing data in the main memory, a place to readily access information to be processed. The simplest type of main memory is DRAM (Dynamic Random Access Memory). A DRAM bit may be formed with a single transistor and a single capacitor—using the charged and discharged status of the capacitor to represent the two states. A CPU comes with an *instruction set* for a well-designed group of built-in operations. Instructions take input and produce results in 4/8-byte words. Modern CPUs are extremely fast, executing over 100 billion instructions per second.

DRAM holds information for CPU processing and its memory cells are *volatile*, losing their contents if power goes off. This is in contrast to hard drives, USB drives, and CD/DVD discs used for long-term data storage. It explains why every time a computer is turned on, it needs to bring the operating system back from a disk into main memory, a process known as *booting*.

In today's computers, main memories are usually 4 to 16 Gigabytes (GB= 10^9 bytes) while hard drives are much larger, approaching several Terabytes (TB= 10^{12} bytes).

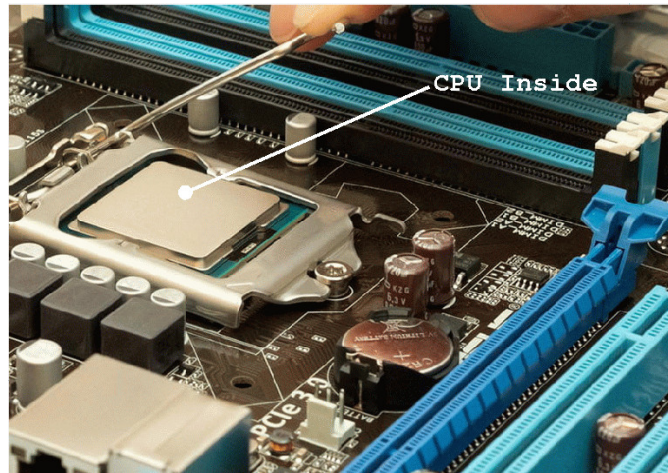


Figure 2:

Such is the nature of digital computer hardware and it dictates that, inside the computer, information be represented and processed exclusively in the form of 1's and 0's. How interesting and challenging?

Integers

Before we immerse ourselves in a world of 1's and 0's, let's first look at our own world. In English, we represent information using words and numbers composed of a set of alphabets (upper and lower case A-Z) and digits (0 through 9). Other languages may use different alphabets.

Inside the computer the alphabet has just two symbols, namely 1 and 0. In this strange world, everything must be stated in 1's and 0's. It is important to realize the 1 and 0 are just convenient symbols to indicate the two states of a bit. **Be sure to disassociate 0 and 1 as bit values from their day-to-day meaning as numbers.** Why not think of 1 as "presence" and 0 as "absence" if you wish. Now, the trick is to use bits to represent other information in systematic ways.

Let's first consider using bits to represent whole numbers zero, one, two, three, and so on. Using three bits, how many numbers can we cover? Here are all the 8 different patterns:

0 0 0, 0 0 1, 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0, 1 1 1

These are all the different 3-letter words using the two letters 0 and 1. We can use them to represent integers 0 through 7, a total of 8 numbers.

The representation is not arbitrary. They are *base-two* or *binary* numbers. Numbers we use everyday are *base-ten* (decimal) where each place value is a power of ten. For example,

$$\textit{Decimal} \quad 209 = 2 \times 10^2 + 0 \times 10 + 9 \times 10^0$$

Similarly for base-2 (binary) numbers, each place value is a power of two. For example,

$$\textit{Binary} \quad 101 = 1 \times 2^2 + 0 \times 2 + 1 \times 2^0$$

Increasingly larger numbers require more bits to represent. With 32 bits we can represent 2^{32} different numbers, enough to cover integers in the range positive and negative $2^{31} - 1$. With 64 bits, we can cover far far more. Arithmetic rules for binary numbers are entirely similar to decimal numbers and are easily performed in a computer.

Characters

Numbers are the most basic, but computers need to handle other types of data among which perhaps the most important is text or character data. Again, bit patterns are used to represent individual characters.

Basically, each character can be assigned a different binary number whose bit pattern represents that character. For example, the American Standard Code for Information Interchange (US-ASCII) uses 7 bits in a byte (covering 0 to 127) to represent 128 characters on a typical keyboard: 0-9, A-Z, a-z, punctuation marks, symbols, and control characters. We have, for example, these character representations:

'0'	00110000	(48)	'9'	00111001	(57)
'A'	01000001	(65)	'Z'	01011010	(90)
'a'	01100001	(97)	'z'	01111010	(122)

Note that the bit pattern for the character 'A' can also represent the integer 65. Note also that, counterintuitively, the bit pattern for the character '9' is different from that for the number 9. Thus, 9 as a character is fundamentally a different symbol than it as a number. With character encoding, textual files are just a sequence of characters.

The world has many languages. Unicode is an international standard for encoding text data from most of the world's writing systems. It now contains more than 110,000 characters from 100 languages/scripts. The Unicode Consortium, an international collaboration, publishes and updates the Unicode standard.

Unicode allows mixing, in a single document file, characters from practically all known languages. This is very advantageous especially in a world increasingly interconnected by the Internet and the Web. Most webpages are written in HTML using UTF-8, a particularly efficient form of Unicode.

Data Context

A symbol, word, or phrase may have a very different meaning depending on the context where it is used. For example, consider the symbol 'like': we like the puppy; it looks like a cat; and I was like "crazy!". Bit patterns are no exception.

You must have realized that a given bit pattern may represent a binary number or a character. For example, the bit pattern 01000001 represents 65 or the character 'A'. Question is how to tell which one. The answer is "context". *The same bit pattern can be interpreted differently depending on the context where it is used.* We must provide a context for any given bit pattern to indicate if it is a number, a character, or something else. The context can be given explicitly or deduced from where the pattern is used. For example, in evaluating the expression $x + 5$, we know the value of x needs to be interpreted as a number. In a computer program, the *data type* of each quantity must be declared implicitly or explicitly. The type informs a program how to interpret the data representation associated with any given quantity.

In everyday life, we must always **interpret data in their proper contexts and avoid information being separated from its context**. For example, we ought to avoid such terms as "today" or "next week" in emails. The awareness is an important part of computational thinking. Not doing so can have serious consequences. In 1999, NASA's Mars Climate Orbiter burned up in the Martian atmosphere because engineers failed to convert units from English to metric.

Music to My Ears

Numbers and characters are relatively easy to represent using bit patterns. But what about more complicated data such as music, picture, or video? Let's look at music first. Sound travels through air as a continuous wave. Sound pitch levels vary from low to high through an infinite number of values.

In the past, analog computers can process electronic wave, produced by a microphone from sound, with ease. But, such analog signals are hard to store, transmit, or reproduce and have loss of fidelity problems. Digital computers can not process continuous data, such as sound, directly. Such data must first be *digitized* and become a series of numbers. It is not hard to digitize. A continuous value, that of a sound wave for example, can be digitized by taking values at a number of sampling points—the more sampling points the more precise the representation (Figure ??). Thus, continuous data become

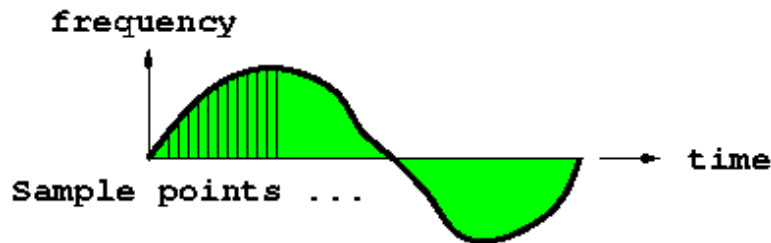








Figure 3:

a series of numbers that can be represented by 1's and 0's and be stored, transmitted, and received without change or loss of fidelity. The digital sound data can further be processed by clever data compression algorithms, such as mp3, to reduce data size and increase transmission speed.

A Picture Is Worth a Thousand Words

A picture is basically different colors on a surface. Thus, representing colors digitally is the first requirement. The widely used RGB (red, green, blue) color system represents a color by the triple (r, g, b) where each number r , g , or b ranges from 0 to 255. For example $(255,0,0)$ is full red, $(0,255,0)$ full green, $(0,0,255)$ full blue, $(0,0,0)$ pure black, and $(255, 255, 255)$ pure white.

Thus, in the RGB system, a byte is used to represent each rgb number and, in turn, a total of 256^3 different colors can be represented .

Red		(255,0,0)	Cyan		(0,255,255)
Green		(0,255,0)	Yellow		(255,255,0)
Blue		(0,0,255)	Magenta		(255,0,255)

The CYMK (cyan, yellow, magenta, black) system is similarly represented. RGB is used for screen displaying while CYMK is used for printing.

In *raster graphics*, a picture is represented by listing the color of all its *pixels*. Each pixel (picture element) is nothing but a point in a rectangular grid over the picture. The finer the grid the better the represented picture. In *vector graphics*, x-y coordinates, lines and other geometrical elements are used to represent a picture. Raster graphics is well-suited for photographs while vector graphics is better for drawings, logos and icons.

With digital pictures at hand, videos can then be represented by a timed sequence of pictures, together with sound data. As you can imagine, high resolution video data can be huge, compute-intensive to render on a display. Often, added graphics hardware such as GPUs (graphics processing units) and graphics cards are used to greatly speed up display of images. Also, many highly efficient image/video data compression algorithms have been developed to reduce their size and increase transmission speed while striving to preserve picture quality.

Programs As Data

By now we can clearly see that ingenious ways have been used to represent all kinds of data. But computers need programs, operating systems and apps, to run. How are programs represented?

Machine language programs, written in CPU instructions, can be run directly, but they are hard for programmers to use and understand. High-level programs such as C++ and Java are invented to make programming much easier by allowing mathematical and English-like expressions. High-level language programs are textual and must first be processed by another program, known as a compiler or an interpreter, and translated into machine language before being executed. Thus, all programs are also data represented by 1's and 0's. In essence, **programs are data**. This means, a computer

can transform and manipulate programs, it can even change and modify its own programming. Programs that can learn or be trained are examples.

Modern computers are *general purpose* machines because they uniquely can store and load different programs—apps and even operating systems. When you run a different program, the computer literally becomes a different machine. A computer, without modification or intervention at the hardware level, can simply load into memory a new program and perform a new task. Therefore a computer can perform any arbitrary task that can be programmed. Can any other machine do this?

Because a program can be manipulated in a computer as data, it can be altered, modified, repaired, and improved by another program, or even by itself. The latter case is like performing brain surgery on oneself. Isn't that fascinating?

In the End

The nature of digital hardware gives rise to a wondrous world of bits, bytes and words—a world that has only two letters in its alphabet, a world where any and all information consists of 1's and 0's, a world in which instructions are given in 1's and 0's, so are input and output, all in head-spinning speed.

Dealing exclusively with 1's and 0's may seem idiotic, yet that simplicity is the basis for reducing size, decreasing cost, and increasing speed. Think today's computers powerful? Just wait, we haven't seen anything yet!

PS: If a world with only two symbols is so nice, why not go one step further to a world of ONE SYMBOL? Wouldn't that be even better? Think about it a bit and see the answer next.

It is not possible to have only one symbol. Its presence necessarily imply its absence (another symbol). Thus, the world of 1's and 0's is it.