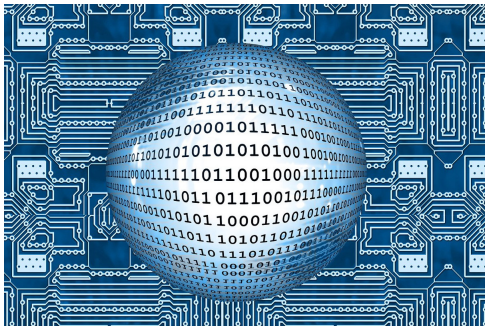
A World of 1's and 0's

Paul S. Wang, Sofpower.com

February 20, 2019

Introduction

We live in a world dominated by digital technologies. They are everywhere and enrich our lives in countless ways. The term *digital* comes from "digital computers", the very basis of such technologies. 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. How incredible? How fascinating?



Inside a computer, it is a world of 1's and 0's. But how could this work? How can 1's and 0's play music or stream movies? Understand speech? Land men on the Moon? Operate driver-less cars?

All this may sound complicated. But the basics are really quite simple to understand, as well as informative for *computational thinking* (CT). We'll try to shine a light on the hidden world of 1's and 0's in this 8th article of our CT series (past articles in `aroundkent.net` Vol. 13 to 19).

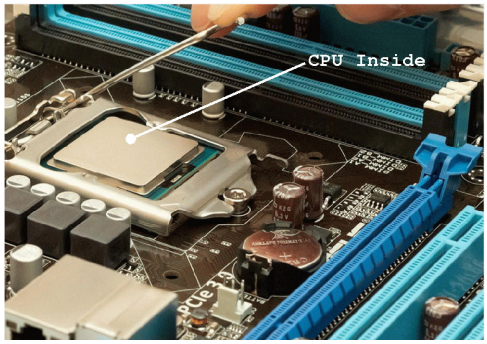
Digital Hardware

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 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 it's 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).

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 "white dot" and 0 as "black dot" 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. If we have three bits, how many numbers can we cover? Here are all the 8 different patterns three bits can form:

```
0 0 0,  0 0 1,  0 1 0,  0 1 1,  1 0 0,  1 0 1,  1 1 0,  1 1 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,

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

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

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

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

Characters

Numbers are fundamental, 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.

The world does not revolve around English. Unicode is an international standard for encoding, representing, and handling 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 Unicode.

Data Context

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 do we know which one. The answer is "context". *The same bit pattern can be interpreted differently depending on the context where it is used.* This happens often in natural languages as well. For example, the word "bra" means "good" in Swedish but a specific garment in English.

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 > 0`, we know the value of `x` needs to be interpreted as a number. In a computer program, the *data type* of each and every constant and variable quantity must be declared implicitly or explicitly. The type informs a program how to interpret the data representation associated with any given quantity.

Be careful to always interpret information in its proper context. That 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.

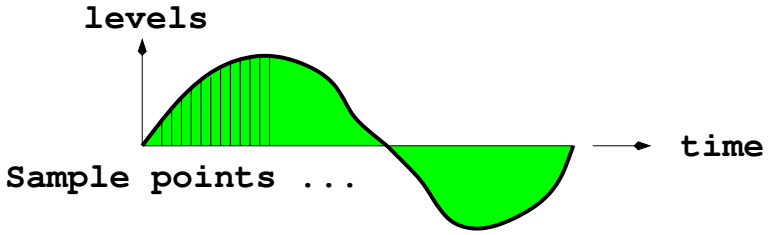
Music to My Ears

Numbers and characters are relatively easy to represent using bit patterns. But what about more complicated data such as music, picture, or video?

Digital computers store and process *discrete* rather than *continuous* information.

- Discrete data: Data are *discrete* when only certain distinct separate values are allowed. The number of chickens, letter grades, basketball scores, age, income, integers, fractions, and so on are examples.
- Continuous data: Data are *continuous* when all values in a finite or infinite range are allowed. Length, weight, volume, temperature, pressure, speed, brightness, and so on are examples.

In the past, analog computers process continuous electronic waves. Such analog signals are hard to store, transmit, or reproduce precisely. In contrast, digital computers use integers to represent information and therefore avoid these critical problems. A continuous value, that of a sound wave for example, can be *digitized* by sampling values at a number of discrete points— the more sampling points the more precise the representation.



Thus, a continuous wave is represented by a series of levels taken at the sampling points. Each level is rounded up or down to the nearest discrete value representable by a prescribed bit length. Using a larger number of bits for each sample point also means a more precise representation. The digitized sound wave, now represented by 1's and 0's, can then be stored, transmitted, retrieved and reproduced by speakers with almost no loss of fidelity—not any the human ear can detect. 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. In the RGB (red, green, blue) color system, a color is represented by the level of intensity of its red, green, and blue components. With 256 levels (0 to 255), level 255 red is full-bright red while level 0 red has no brightness and can't be seen.

The widely used RGB system represents a color by the triple (r, g, b) where each number r, g, or b ranges from 0 to 256. 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, bit patterns are used to represent the rgb numbers which in turn can represent 256^3 different colors.

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 is nothing but a point in a rectangular grid over the picture. The finer the gird the better the represented picture. In *vector graphics*, x-y coordinates, lines and other geometrical elements are used to represent the 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, slow to render on a display, hard to store, and slow to transmit. Luckily, many highly efficient video compression algorithms have been developed to reduce their size and increase transmission speed while striving to preserve picture quality. Also, added graphics hardware such as GPUs (graphics processing units) and graphics cards can greatly speed up display of images.

Programs

By now we can clearly see that ingenious ways have been used to represent all kinds of data. But computers also store and process programs. How are programs represented?

Well, first of all, each instruction in the *CPU instruction set* consists of a *command* coded by a number and zero or more *arguments*. Each argument is data to be acted upon by the command and indicated simply by a memory address, again a number, where the data can be found.

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

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?

In the End

The nature of digital hardware gives rise to a strange 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!