1 Plus 1 Equals 10

Paul S. Wang, Sofpower.com

May 15, 2023

Modern computers store and process digital data represented by *bits* which can represent either a 0 or a 1. All data—numbers, text, images, sound, and video—must be turned into bits and stored in memory before a computer can process them. Among all data, the representation of numbers is the most fundamental.

A focus here is the representation of numbers inside computers. We'll see how a sequence of bits is used to represent numbers in what's known as the *binary notation*. And we'll see how the *place value* system works.

Concepts discussed here are critical to understanding the digital nature of modern computers, and therefore important for any computational thinker. For many, concepts explained in this article can be eye-opening mental breakthroughs.

This post is part of our *Computational Thinking* (CT) blog where you can find many other interesting and useful articles.

Digital Computers

Modern computers are *digital* because they 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. Discrete values have gaps in them.

- Continuous data—Data are *continuous* when all values in a continuous range, finite or infinite, are allowed. Length, weight, volume, temperature, pressure, speed, brightness, and so on are examples. Continuous

values have no gaps separating them. Thus, even in a small range, there are an infinite number of continuous values.

In the past, analog computers processed 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 (Figure 1). With enough sampling points, the continuous sound wave

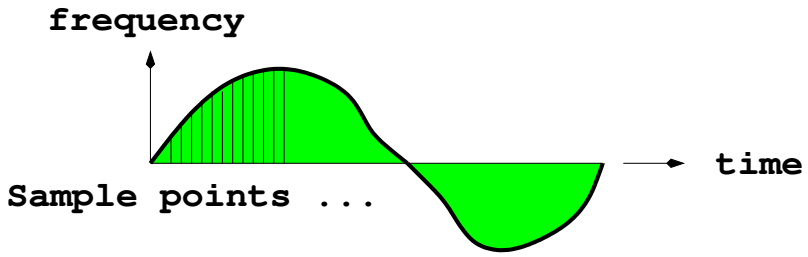

Figure 1: Sampling a Continuous Wave

can be recreated. Information, represented in digital form, must be stored in computer memory to be processed. The most basic memory unit is a bit, which can represent either a `1` or a `0`. A *byte* is a group of 8 bits. A *word* consists of several (usually 4 or 8) bytes (Figure 2). Note in computing, as shown in Figure 2, we count starting from zero.

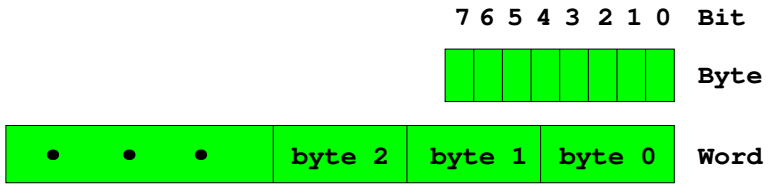

Figure 2: Bit, Byte, and Word

A computer's *Central Processing Unit* (CPU) performs logic computations on data. The CPU loads data from memory in order to process them, then stores the results back to memory. A *word* is normally the smallest integral data entity a CPU is designed to manipulate. The word size is measured in bits and is one of the most important hardware architecture features

of a computer. For most modern computers, the word size is either 32 bits (4 bytes) or 64 bits (8 bytes). Of course the word size is determined by the size of CPU registers. A CPU register is able to load from and store to cache/RAM (*Random Access Memory*) all its bits (the whole word) as a unit at once.

For modern general purpose computers, the entire memory is an array of bytes (Figure 3), each of which can be *addressed* directly (called byte addressing), hence the term random access memory (RAM). A computer with word size n bits can address (reach directly) at most 2^n bytes in its RAM. Often the actual allowable size of RAM for a computer is well below this upper bound due to other hardware limitations.

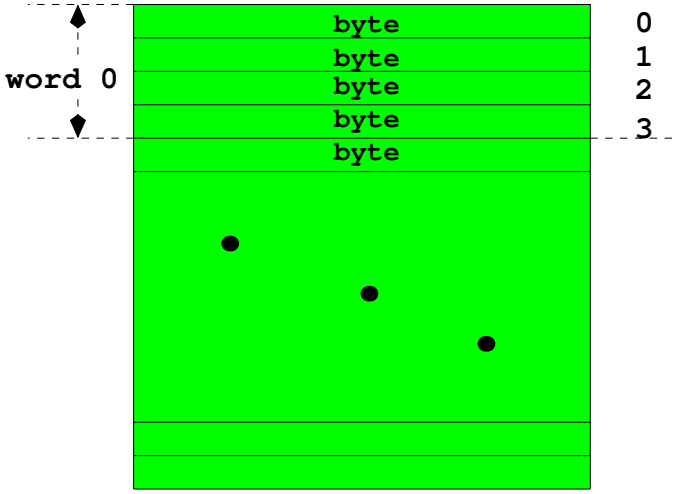

Figure 3: Memory Array of Bytes

Digital information must be encoded by zeros and ones and stored in memory in order to be processed. Information stored in RAM is *volatile* and will disappear when the system is turned off. Nonvolatile or *persistent* data storage is provided by one or more storage drives, including hard-disk drives (HHD) or solid-state drives (SSD).

In computer product specifications, available memory in cache, RAM, or data storage is given in byte units:

- kilobyte (KB) = 1024 bytes
- megabyte (MB) = 1024 KB

- gigabyte (GB) = 1024 MB
- terabyte (TB) = 1024 GB
- petabyte (PB) = 1024 TB

Typical desktop computer RAM sizes range from 4GB to 32GB.

Ordinarily, K (kilo) is a prefix for 1000 (in the metric system), but digitally (in the binary system), K is 1024. Similarly, as far as memory or data sizes go, M (Mega) is KK, Giga is KM, Tera is KG, and Peta is KT, keeping in mind always that K is 1024[1]

Bit Patterns

The basic way to represent data in computer memory is to use *bit patterns*. A bit pattern is a particular sequence of zeros and ones presented in a fixed number of bits. Table 1 shows all possible patterns made up of three bits. Each bit has two variations, 0 and 1. For each value of bit 1, there are two

Table 1: The Eight 3-Bit Patterns

Bit 2	Bit 1	Bit 0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

values for bit 0, resulting in $2 \times 2 = 2^2$ two-bit patterns. Similarly, for each value of bit 2, there are 4 patterns for bits 0 and 1, giving $2 \times 2^2 = 2^3$ three-bit patterns.

In general, the total number of different patterns with n bits is 2^n. Therefore, a byte can give you $2^8 = 256$ bit patterns, a 32-bit word $2^{32} = 4294967296$ bit patterns, and a 64-bit word $2^{64} = 18446744073709551616$ bit patterns.

[1] In actual usage, there is often confusion between the metric and binary interpretations.

In a digital computer, bit patterns are the only way to represent data. And, the same bit patterns can be used to represent different types of data, such as a number, a character, or a network address. We will see how numbers are represented next.

Number Representations

To compute with numbers, we need to represent them in memory. For integers (whole numbers), the *binary number* system uses only zeros and ones and is therefore particularly suited to be stored as bit patterns.

Numerals

Numerals are symbols we use to write down numbers. The numerals we use today (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) are *Arabic numerals* derived from the Hindu-Arabic numeral system. Different civilizations have invented and used their own symbols for numbers. Figure 4 shows the Roman and Egyptian numerals.

Figure 4: Roman and Egyptian Numerals

And Figure 5 shows ancient Chinese numerals. Simplified modern versions of these are still in use today.

With the digits, the Arabic numerals, we can write down numbers from zero to nine. We still need a way to represent larger numbers. We can invent new symbols, of course. But, how will we create new symbols to keep up with ever larger numbers? A systematic way must be found. An ingenuous solution is the ***place value system***. For example, with three digits side by side, we can write down numbers up to, but not including, a thousand. Thus, the notation 379 means three hundreds, seven tens, and nine ones, or

$$379 = 3 \times 10^2 + 7 \times 10 + 9 \times 10^0$$

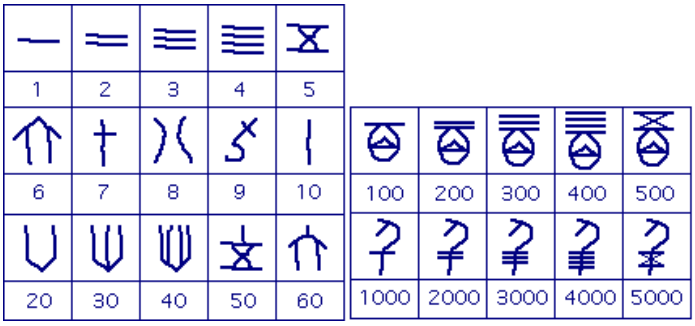

Figure 5: Ancient Chinese Numerals

With the place value system, larger numbers simply require more places. Thus, we have a system for creating new symbols for numbers by combining the numerals. Such numbers, where each place represents a power of 10, are known as *base-10 numbers* or *decimal numbers*.

Base-2 Numbers

Binary numbers also use a place value system, just like decimal numbers, except the base is 2, not 10. We use only two numerals, 0 and 1, for binary numbers. Using one place, only two numbers can be represented, namely 0 and 1. To represent two in binary, we need to go to the next place. Thus, **the binary number** 10 **means two**

$1 \times 2 + 0 \times 1$

Thus, in binary we can say $1 + 1 = 10$.

And the binary number 11 means three

$1 \times 2 + 1 \times 1$

Similarly, 101 means five

$1 \times 2^2 + 0 \times 2 + 1 \times 1$

With four bits, we can represent numbers from zero to fifteen (Table 2). Keep in mind that, just as in decimal numbers, the least significant digit (bit) has the rightmost position and the most significant bit has the leftmost position. And the bit and byte positions are always counted from the right.

Table 2: Four-Bit Binary Numbers

Dec	Binary	Dec	Binary	Dec	Binary	Dec	Binary
0	0000	1	0001	2	0010	3	0011
4	0100	5	0101	6	0110	7	0111
8	1000	9	1001	10	1010	11	1011
12	1100	13	1101	14	1110	15	1111

To get familiar with binary notation, take any binary number in Table 2 and add 1 to it, carrying to the next higher position as you would in doing addition, you will get the bit pattern for the next binary number.

It is not surprising if you find binary numbers confusing at first. After all, as far as numbers, the decimal system is our mother tongue and is firmly ingrained in our thinking. Look at Figure 6 and see how you feel yourself.

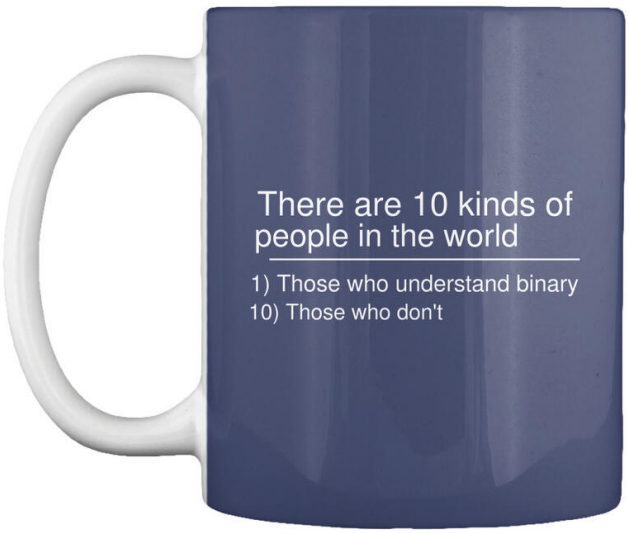

Figure 6: A Binary Cup

Binary numbers would become easier if we memorize the powers of 2 as well as we do powers of 10. Here goes: 2 (bit 2), 4 (bit 3), 8 (bit 4), 16 (bit 5), 32 (bit 6), 64 (bit 7), 128 (bit 8), 256 (bit 9), 512 (bit 10), 1024 (bit 11), 2048 (bit 12), and so on.

Inside a digital computer, numbers are naturally in binary, and hardware

support is provided for their operations, as long as they fit in a single word. A 32- or 64-bit word can represent numbers from 0 to $2^{32}-1$ or $2^{64}-1$. That's a lot of numbers. Still, it is far from covering all numbers. To handle larger numbers, we can use multiple words and write software for manipulating them.

Here is a principle for computational thinkers:

> *Anyone can invent a symbol and assign it a meaning. The same symbol may have another meaning in a different context. Do not guess a symbol's meaning by its appearance. Always refer to its definition within the intended context.*

Numerals and numbers from different cultures, as well as the place value system, are examples of this principle.

Ancient Chinese Binary Symbols

I-Ching (易經), a Chinese text that traces back to the 3rd to the 2nd millennium BCE, introduced two symbols, yin (⚋) and yang (⚊). Combinations of three or six yin-yang symbols form the 8 trigrams (八卦 Figure 7) or the 64 hexagrams (六十四卦). Thus, the concept of repeating two symbols to

Figure 7: I-Ching Trigrams (八卦)

form increasingly more symbols is an ancient one.

Numbers in Other Bases

While we are at it, why don't we look at numbers in other bases? Octal numbers are base 8 and use digits 0–7. Hexadecimal (hex) numbers are base 16 and use digits 0–9 followed by A (ten), B (eleven), C (twelve), D (thirteen), E (fourteen), and F (fifteen). The symbol 10 stands for eight in octal and sixteen in hexadecimal. The symbol 25 means

$2 \times 8 + 5$ (21 in octal)
$2 \times 16 + 5$ (37 in hex)

For interactive demos, visit our companion website (**Demo:** `OctalCounter` and **Demo:** `HexCounter`).

Table 3: Numbers in Different Bases

Decimal	11	17	23	29
Octal	13	21	27	35
Hex	B	11	17	1D
Binary	01011	10001	10111	11101

Three bits are needed for each octal digit and four bits for each hexadecimal digit. A byte of all 1s can represent the hex number FF (two hundred fifty-five). In general, we can use numbers in any base we desire, and we are not, limited by having two hands with ten digits. A principle for computational thinkers:

> *No feature in a system should be treated as rigid or sacred. Ask "what if it is changed?" and you will begin to think flexibly, acquire a deeper understanding, and, perhaps, even make a breakthrough.*

Octal and hex numbers are often used as a shorthand for the bit patterns representing them. Each octal digit specifies a 3-bit pattern, and each hex digit specifies a 4-bit pattern. Also, distinct prefixes, `0x` for hex, `0o` for octal, and `0b` for binary are used to denote such numbers. This is especially true in programming languages. Table 4 shows the bits for each digit of `0o1357` or `0x2EF`.

Table 4: Octal and Hex Bit Patterns

Octal	1			3			5			7		
Binary	0	0	1	0	1	1	1	0	1	1	1	1
Hex	2			E			F					

Mixed-Base Numbers

Decimal numbers use powers of ten as place values; binary numbers powers of two; octal numbers powers of eight; and hex numbers powers of sixteen. Talking about flexible thinking, what about asking the question why the place values must always be a power of some fixed number?

Well, there is no reason for that at all. In fact, we can use any desired value for each place. Such numbers are called *mixed-radix* or *mixed-base* numbers. For instance, how about numbers with place values 1, 60, 60, 24, 7? Would that be crazy?

Actually, we use such numbers every day, literally. We have 60 seconds in a minute, 60 minutes in an hour, 24 hours in a day, and 7 days in a week, don't we? And we have 12 inches in a foot, 3 feet in a yard, and 1760 yards in a mile. Granted, such length measurements should have long ago been replaced by the metric system.

Computational thinks, let's apply abstraction here. The intrinsic nature of a place-value system is "assign a value for each place," and nothing more.

Meaning of Symbols

The binary number system may be eye opening for many. But what is more important is the realization of the fact that "*a symbol, any symbol, has no inherent meaning and we can use it to stand for anything we wish.*"

Thus, the symbol 10 can stand for the number ten, or the number two, or smothing else, for example "a closed eye and an open eye".

The meaning of a symbol (Figure 8) must be obtained from its definition within the context of its usage. For example, 911 can be a count, a phone number, or a disaster event. The symbol 03/04 may be March 4th or April 3rd.

Consider the statement: $1+1$ equals zero and $1-1$ equals two. Here, the

Figure 8: What Is That?

symbol + is used for subtraction and − for addition. And why not! After all, + and − are not born with any meaning at all.

Human beings use symbols, in written, verbal and other forms, to represent ideas in order to communicate with others. The process involves three steps:

1. You use a sequence of symbols to represent the idea, concept or meaning which resides in your mind.

2. You communicate these symbols to someone else.

3. The receiver decodes the symbols back into a meaning.

Do you see where the problem lies? **There is no guarantee that the meaning received is the same as the original one**.

The situation is made worse by the fact that the same sequence of symbols may mean different things and often does. For example, the symbol O may be the number zero or an English alphabet. Similarly for the symbol l.

Consider the 4-letter word "chef." In English, it refers to a professional cook or the head cook in a restaurant. In Spanish, it is used to refer to a boss or leader. In French, it means chief or leader in a general sense, and can refer to someone in a position of authority or a person who is skilled or accomplished in a particular field.

From a computational thinking perspective, such symbol reuse is unavoidable. Because the number of symbols in use are limited whereas ideas, concepts, and meanings are infinite. We can invent a few new symbols or sequence of symbols, but we still cannot avoid reuse.

Figure 9: Texting

Consider another 4-letter word "text". We know what it means, do we? It means "a short message" sent or received on a cellphone! Or the act of sending such a message (Figure 9). It is a perfect example of symbols acquiring new meaning with the changing times.

Computational thinkers must keep all this in mind when initiating a communication or when receiving one, especially with texting! By being precise and careful, we can make sure that any message conveyed/received is correctly understood. Thus, we can avoid confusion, mistakes, and even save lives.