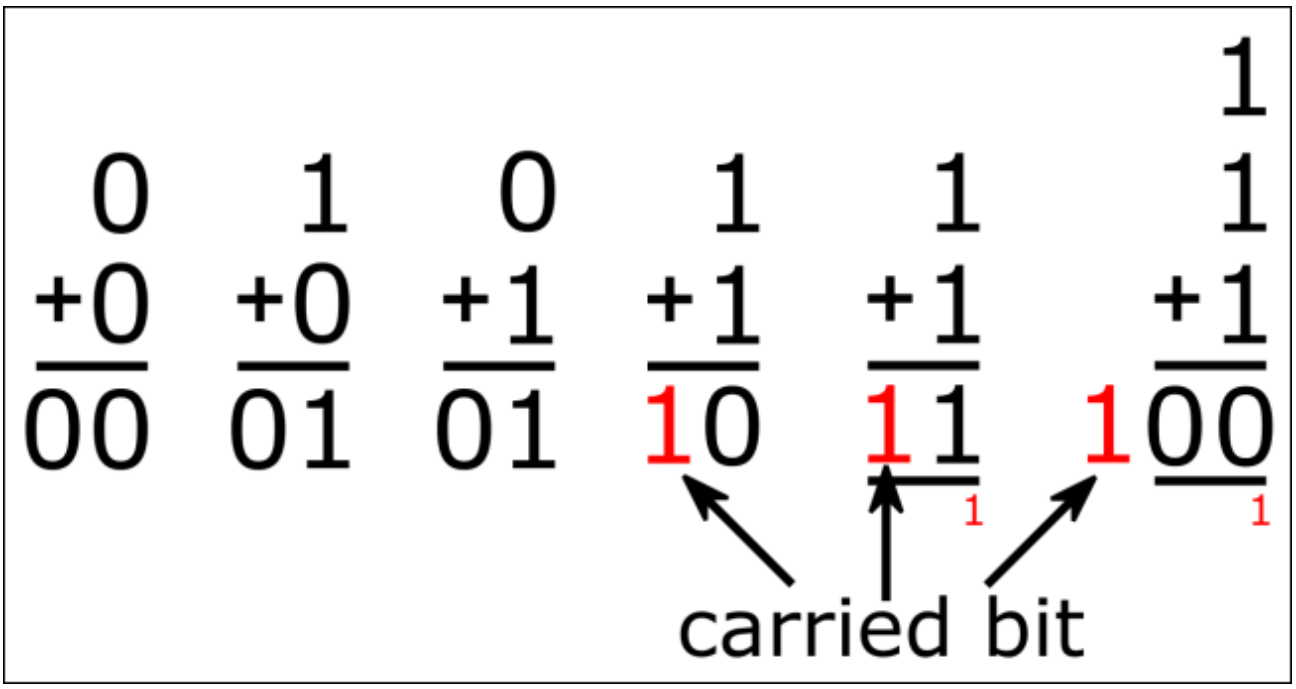
What is Binary, and Why Do Computers Use It?

Computers don't understand words or numbers the way humans do. Modern software allows the end user to ignore this, but at the lowest levels of your computer, everything is represented by a binary electrical signal that registers in one of two states: on or off. To make sense of complicated data, your computer has to encode it in binary.

Binary is a base 2 number system. Base 2 means there are only two digits—1 and 0—which correspond to the on and off states your computer can understand. You're probably familiar with base 10—the decimal system. Decimal makes use of ten digits that range from 0 to 9, and then wraps around to form two-digit numbers, with each digit being worth ten times more than the last (1, 10, 100, etc.). Binary is similar, with each digit being worth two times more than the last.

Counting in Binary



In binary, the first digit is worth 1 in decimal. The second digit is worth 2, the third worth 4, the fourth worth 8, and so on—doubling each time. Adding these all up gives you the number in decimal. So,

```
1111 (in binary)  =  8 + 4 + 2 + 1  =  15 (in decimal)
```

Accounting for 0, this gives us 16 possible values for four binary bits. Move to 8 bits, and you have 256 possible values. This takes up a lot more space to represent, as four digits in decimal give us 10,000 possible values. It may seem like we're going through all this trouble of reinventing our counting system just to make it clunkier, but computers understand binary much better than they understand decimal. Sure, binary takes up more space, but we're held back by the hardware. And for some things, like logic processing, binary is better than decimal.

There's another base system that's also used in programming: hexadecimal. Although computers don't run on hexadecimal, programmers use it to represent binary addresses in a human-readable format when writing code. This is because two digits of hexadecimal can represent a whole byte, eight digits in binary. Hexadecimal uses 0-9 like decimal, and also the letters A through F to represent the additional six digits.

So Why Do Computers Use Binary?

The short answer: hardware and the laws of physics. Every number in your computer is an electrical signal, and in the early days of computing, electrical signals were much harder to measure and control very precisely. It made more sense to only distinguish between an "on" state—represented by negative charge—and an "off" state—represented by a positive charge.

For those unsure of why the "off" is represented by a positive charge, it's because electrons have a negative charge—more electrons mean more current with a negative charge.

So, the early room-sized computers used binary to build their systems, and even though they used much older, bulkier hardware, we've kept the same fundamental principles. Modern computers use what's known as a transistor to perform calculations with binary. Here's a diagram of what a field-effect transistor (FET) looks like:

Essentially, it only allows current to flow from the source to the drain if there is a current in the gate. This forms a binary switch. Manufacturers can build these transistors incredibly small—all the way down to 5 nanometers, or about the size of two strands of DNA. This is how modern CPUs operate, and even they can suffer from problems differentiating between on and off states (though that's mostly due to their unreal molecular size, being subject to the weirdness of quantum mechanics[1]).

But Why Only Base 2?

So you may be thinking, "why only 0 and 1? Couldn't you just add another digit?" While some of it comes down to tradition in how computers are built, to add another digit would mean we'd have to distinguish between different levels of current—not just "off" and "on," but also states like "on a little bit" and "on a lot."

The problem here is if you wanted to use multiple levels of voltage, you'd need a way to easily perform calculations with them, and the hardware for that isn't viable as a replacement for binary computing. It indeed does exist; it's called a ternary computer[2], and it's been around since the 1950s, but that's pretty much where development on it stopped. Ternary logic is way more efficient than binary, but as of yet, nobody has an effective replacement for the binary transistor, or at the very least, no work's been done on developing them at the same tiny scales as binary.

The reason we can't use ternary logic comes down to the way transistors are stacked in a computer—something called "gates"—and how they're used to perform math. Gates take two inputs, perform an operation on them, and return one output.

This brings us to the long answer: binary math is way easier for a computer than anything else. Boolean logic maps easily to binary systems, with True and False being represented by on and off. Gates in your computer operate on boolean logic: they take two inputs and perform an

operation on them like AND, OR, XOR, and so on. Two inputs are easy to manage. If you were to graph the answers for each possible input, you would have what's known as a truth table:

A binary truth table operating on boolean logic will have four possible outputs for each fundamental operation. But because ternary gates take three inputs, a ternary truth table would have 9 or more. While a binary system has 16 possible operators (2^2^2), a ternary system would have 19,683 (3^3^3). Scaling becomes an issue because while ternary is more efficient, it's also exponentially more complex.

Who knows? In the future, we could begin to see ternary computers become a thing, as we push the limits of binary down to a molecular level. For now, though, the world will continue to run on binary.

Image credits: spainter_vfx[3]/Shutterstock, Wikipedia[4], Wikipedia[5], Wikipedia[6], Wikipedia[7]

Links

1. https://en.wikipedia.org/wiki/Quantum_tunnelling
2. https://en.wikipedia.org/wiki/Ternary_computer
3. https://www.shutterstock.com/image-illustration/3d-illustration-blue-bytes-binary-code-610018388?src=ExNjRgOV4Z3nRobKZgz8gg-1-28
4. https://upload.wikimedia.org/wikipedia/commons/thumb/a/a6/CPT-Binary-Addition.svg/1280px-CPT-Binary-Addition.svg.png
5. https://commons.wikimedia.org/wiki/File:Truth_table_for_AND,_OR,_and_NOT.png
6. https://fr.m.wikipedia.org/wiki/Fichier:TFET_lateral_structure.png
7. https://en.wikipedia.org/wiki/File:Full-Adder_Propagation_Delay.svg