



Representing Numbers and Letters with Binary: Crash Course Computer Science #4

Crash Course: Computer Science

<https://youtube.com/watch?v=1GSjbWt0c9M>

<https://nerdfighteria.info/v/1GSjbWt0c9M>

Hi, I'm Carrie Anne, and this is Crash Course Computer Science. Today we're going to talk about how computers store and represent numerical data, which means we've got to talk about math. But don't worry – every single one of you already knows exactly what you need to know to follow along.

So last episode, we talked about how transistors can be used to build logic gates which can evaluate Boolean statements, and in Boolean algebra there are only two binary values: true and false. But if we only have two values, how in the world do we represent information beyond just these two values? That's where the math comes in.

[Crash Course intro]

So as we mentioned last episode, a single binary value can be used to represent a number. Instead of 'true' and 'false', we can call these two states 'one' and 'zero', which is actually incredibly useful. And if we want to represent larger things, we just need to add more binary digits. This works exactly the same way as the decimal numbers that we're all familiar with. With decimal numbers, there are only ten possible values a single digit can be: 0 through 9. To get numbers larger than 9, we just start adding more digits to the front.

We can do the same with binary. For example, let's take the number 263. What does this number actually represent? Well, it means we've got two 100s, six 10s, and three 1s. If you add those all together, we've got 263. Notice how each column has a different multiplier; in this case, it's 100, 10, and 1. Each multiplier is 10 times larger than the one to the right. That's because each column has 10 possible digits to work with, 0 through 9, after which you have to carry one to the next column. For this reason, it's called base 10 notation, also called decimal since 'deci-' means 10.

Binary works exactly the same way, it's just base 2. That's because there are only two possible digits in binary, 1 and 0. This means that each multiplier has to be two times larger than the column to its right. Instead of 100s, 10s, and 1s, we now have 4s, 2s, and 1s. Take for example the binary number 101. This means we have one 4, zero 2s, and one 1. Add those all together and we've got the number 5 in base 10.

But to represent larger numbers, binary needs a lot more digits. Take this number in binary: 10110111. We can convert it to decimal in the same way. We have $1 \times 128 + 0 \times 64 + 1 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = 183$.

Math with binary numbers isn't hard either. Take for example decimal addition of $183 + 19$. First we add $3 + 9 = 12$ so we put 2 as the sum and carry 1 to the 10s column. Now we add $8 + 1 + 1$ (that we carried) = 10, so the sum is 0 carry 1. Finally we add $1 + 1$ (that we carried) = 2, so the total sum is 202.

Here's the same sum but in binary. Just as before, we start with the 1s column. Adding $1 + 1 = 2$ but there is no symbol 2 so we use 10 and put 0 as our sum and carry the 1, just like in our decimal example. $1 + 1 + 1$ (that we carried) = 3, or 11 in binary, so we put the sum as 1 and carry 1 again and so on. We end up with this number 11001010, which is the same as the number 202 in base 10.

Each of these binary digits 1 or 0 is called a bit. So in these last few examples we were using 8-bit numbers with their lowest value of 0 and highest value of 255, which requires all eight bits to be set to 1. That's 256 different values, or 2^8 . You might have heard of 8-bit computers or 8-bit graphics or audio. These were computers that did most of their operations in chunks of 8 bits, but 256 different

values isn't a lot to work with so it meant things like 8-bit games were limited to just 256 different colours for their graphics.

8 bits is such a common size in computing it has a special word: a 'byte'. A byte is 8 bits. If you've got 10 bytes, it means you've really got 80 bits. You've heard of kilobytes, megabytes, gigabytes, and so on. These prefixes denote different scales of data. Just like one kilogram is 1000 grams one kilobyte is 1000 bytes, or really 8000 bits. Mega is a million bytes and giga is a billion bytes. Today, you might even have a hard drive that has one terabyte of storage. That's eight trillion 1s and 0s.

But hold on, that's not always true. In binary, a kilobyte has 2^{10} bytes, or 1024. 1000 is also right when talking about kilobytes, but we should acknowledge it isn't the only correct definition. You've probably also heard the term 32-bit or 64-bit computers. You're almost certainly using one right now. What this means is that they operate in chunks of 32 or 64 bits. That's a lot of bits!

The largest number you can represent with 32 bits is just under 4.3 billion, which is 2^{32} in binary. This is why our Instagram photos are so smooth and pretty; they are composed of millions of colours, because computers today use 32-bit graphics.

Of course, not everything is a positive number, like my bank account in college, so we need a way to represent positive and negative numbers. Most computers use the first bit for the sign, 1 for negative and 0 for positive numbers, and then use the remaining 31 bits for the number itself. That gives us a range of roughly plus or minus two billion. While this is a pretty big range of numbers, it's not enough for many tasks. There are seven billion people on the Earth, and the US national debt is almost 20 trillion dollars after all. This is why 64-bit numbers are useful. The largest value a 64-bit number can represent is around 9.2 quintillion. That's a lot of possible numbers and will hopefully stay above the US national debt for a while.

Most importantly, as we'll discuss in a later episode, computers must label locations in their memory known as 'addresses' in order to store and retrieve values. As computer memory has grown to gigabytes and terabytes – that's trillions of bytes – it was necessary to have 64-bit memory addresses as well.

In addition to negative and positive numbers, computers must deal with numbers that are not whole numbers, like 12.7 and 3.14, or maybe even Stardate 43989.1. These are called floating-point numbers because the decimal point can float around in the middle of a number. Several methods have been developed to represent floating-point numbers, the most common of which is the IEEE 754 standard. And you thought historians were the only people bad at naming things! In essence, this standard stores decimal values sort of like scientific notation. For example, 625.9 can be written as 0.6259×10^3 . There are two important numbers here. The .6259 is called the significand and 3 is the exponent.

In a 32-bit floating-point number, the first bit is used for the sign of the number, positive or negative. The next eight bits are used to store the exponent, and the remaining 23 bits are used to store the significand.

Okay, we've talked a lot about numbers, but your name is probably composed of letters, so it's really useful for computers to also have a way to represent text. However, rather than have a special form of storage for letters, computers simply use numbers to represent letters. The most straightforward approach might be to simply number the letters of the alphabet, A being 1, B being 2, C being 3, and so on. In fact, Francis Bacon, the famous English writer, used 5-bit sequences to encode all 26 letters of the English alphabet to send secret messages back in the 1600s.



Representing Numbers and Letters with Binary: Crash Course Computer Science #4

Crash Course: Computer Science

<https://youtube.com/watch?v=1GSjbWt0c9M>

<https://nerdfighteria.info/v/1GSjbWt0c9M>

5 bits can store 32 possible values, so that's enough for the 26 possible letters but not enough for punctuation, digits, and upper and lowercase letters. Enter ASCII, the American Standard Code for Information Interchange. Invented in 1963, ASCII was a 7-bit code, enough to store 128 different values. With this expanded range, it could encode capital letters, lowercase letters, digits 0 through 9, and symbols like the @ sign and punctuation marks. For example, a lowercase A is represented by the number 97, while a capital A is 65, a colon is 58, and a close parenthesis is 41. ASCII even had a selection of special command codes, such as a newline character to tell a computer where to wrap a line to the next row. In older computers, the line of text would literally continue off the edge of the screen if you didn't include a newline character.

Because ASCII was such an early standard, it became widely used, and critically allowed different computers built by different companies to exchange data. This ability to universally exchange information is called interoperability. However, it did have a major limitation: it was really only designed for English.

Fortunately, there are 8 bits in a byte not 7, and it soon became popular to use codes 128-255, previously unused, for national characters. In the US, those extra numbers were largely used to encode additional symbols like mathematical notation, graphical elements, and common accentuated characters. On the other hand, while the Latin characters were used universally, Russian computers used the extra codes to encode Cyrillic characters and Greek computers used Greek letters and so on. And national character codes worked pretty well for most countries.

The problem was if you opened an email written in Latvian on a Turkish computer, the result was completely incomprehensible, and things totally broke with the rise of computing in Asia, as languages like Chinese and Japanese have thousands of characters. There was no way to encode all those characters in 8 bits.

In response, each country invented multibyte encoding schemes, all of which were mutually incompatible. The Japanese were so familiar with this encoding problem that they even had a special name for it, *mojibake*, which means 'scrambled text'.

And so it was born: Unicode, one format to rule them all. Devised in 1992 to finally do away with all of the different international schemes, it replaced them with one universal encoding schemes. The most common version of Unicode uses 16 bits with space for over a million codes, enough for every single character from every language ever used – more than 120,000 of them in over 100 types of script, plus space for mathematical symbols and even graphical characters like emoji.

And in the same way that ASCII defines a scheme for encoding letters as binary numbers, other file formats like MP3s or GIFs use binary numbers to encode sounds or colours of a pixel in our photos, movies, and music. Most importantly, under the hood it all comes down to long sequences of bits. Text messages, this YouTube video, every webpage on the internet, and even your computer's operating system are nothing but long sequences of 1s and 0s.

So next week, we'll start talking about how your computer is manipulating those binary sequences for our first true taste of computation. Thanks for watching; see you next week.

Crash Course Computer Science is produced in association with PBS Digital Studios. At their channel, you can check out a playlist of shows like Physics Girl, Deep Look, and PBS Spacetime. This episode was filmed at the Chad & Stacey Emigholz Studio in Indianapolis, Indiana, and it was made with the help of all these nice people and our wonderful graphics team Thought Café.

That's where we're going to have to halt and catch fire. See you next week.