How a computer stores numbers

Integers in base 2

We normally write numbers as sums of small multiples of powers of 10, but the base 10 is somewhat arbitrary, an ancient cultural artefact of the number of fingers we possess. Inside computers, the simplest way to record data is in terms of two states, 'on' or 'off'. This leads to a representation of numbers with respect to base 2.

The basic fact is that every positive integer can be expressed as a sum

$$b_0 + 2b_1 + 4b_2 + \cdots$$

where each $b_i = 0$ or 1.

Here are some examples:

$$\begin{aligned}
2 &= 2^1 \\
&= \text{`}10\text{'} \\
4 &= 2^2 \\
&= \text{`}100\text{'} \\
8 &= 2^3 \\
&= \text{`}1000\text{'} \\
7 &= 4+2+1 \\
&= 2^2 + 2^1 + 2^0 \\
&= \text{`}111\text{'} \\
13 &= 8+4+1 \\
&= 2^3 + 2^2 + 2^0 \\
&= \text{`}1101\text{'} \\
15 &= 8+4+2+1 \\
&= 2^3 + 2^2 + 2^1 + 2^0 \\
&= \text{`}1111\text{'}
\end{aligned}$$

Real numbers in base 2

We can also have negative powers of 2:

$$\begin{aligned}
1/2 &= 2^{-1} \\
&= \text{`}0.1\text{'} \\
1/4 &= 2^{-2} \\
&= \text{`}0.01\text{'} \\
1/8 &= 2^{-3} \\
&= \text{`}0.001\text{'} \\
13/16 &= 1/2 + 1/4 + 1/16 \\
&= 2^{-1} + 2^{-2} + 2^{-4} \\
&= \text{`}0.1101\text{'}
\end{aligned}$$

For example, to find how to write $19/32$, we write

$$19 = 16 + 2 + 1 = 2^4 + 2^1 + 2^0, \quad 19/32 = 19/2^5 = 2^{-1} + 2^{-4} + 2^{-5} = \text{`}0.10011\,.$$

We can also see that $19/32$ is greater than $1/2$, so we subtract $1/2$ giving us $3/32$; $3/32$ is just larger than $1/16$; $3/32 - 1/16 = 1/32$.

How about fractions whose denominators are not powers of 2, such as $2/3$? We could start off by noting that $1/2 \leq 2/3 < 1$, so the expression starts off with '0.1'; then subtract $2/3 - 1/2 = 1/6$; $1/8 < 1/6 < 1/4$ so it continues '0.101'; etc. But this involves sooner or later a lot of large denominators, and calculations get messy Another way is to carry out long division directly in base 2, which is pretty simple. But there is yet another method which is best for machine computation.

Since $2/3 < 1$ we can write

$$2/3 = 0.b_{-1}b_{-2}b_{-3}b_{-4}\ldots$$

How doe we find b_{-1}? If we multiply the expression by 2 we get

$$4/3 = b_{-1}.b_{-2}b_{-3}b_{-4}\ldots$$

and since $1 \leq 4/3 < 2$ this means that $b_{-1} = 1$. We now subtract off b_{-1} and get

$$\begin{aligned} 4/3 - 1 &= 1/3 \\ &= 0.b_{-2}b_{-3}b_{-4}\ldots \\ 2/3 &= b_{-2}.b_{-3}b_{-4}\ldots \end{aligned}$$

which gives us $b_{-2} = 0$.

$$\begin{aligned} 2/3 - 0 &= 2/3 \\ &= 0.b_{-3}b_{-4}b_{-5}\ldots \\ 4/3 &= b_{-3}.b_{-4}b_{-5}\ldots \end{aligned}$$

so $b_{-3} = 1$. Etc. At any stage we are looking at some

$$x_n = 0.b_{-n}b_{-n-1}b_{-n-2}\ldots$$

and we multiply by 2 to see

$$2x_n = b_{-n}.b_{-n-1}b_{-n-2}\ldots$$

which means that $b_{-n} = 0$ if $2x_n < 1$ and $b_{-n} = 1$ if $2x_n \geq 1$. In other words, b_{-n} is the **floor** of $2x_n$, the integer just below or equal to it, which in these circumstances is always 0 or 1. This calculation can be set up very nicely on a spreadsheet.

Machine numbers

A computer stores most real numbers with a set of 64 zeroes or ones—i.e. **bits**. How it does this is in terms of base 2 scientific notation. For example, $16 =$ '10000' but in this notation is $2^4 \cdot 1.0000$. And

$$2/3 = 2^{-1} \cdot 1.010101\ldots$$

In general, a number is expressed as $\pm 2^e \cdot x$ where $1 \leq x < 2$ so $x = 1.\ldots$. The machine allows 52 bits for x, the **mantissa**, one bit for the sign, and 11 for the **exponent** e, making 64 bits in all. The mantissa by definition always starts off with 1, so it uses up 52 bits but has 53 bits in its expression since the initial 1 is redundant. The exponent is allowed to be negative, roughly in a range $[-1024, 1023]$.

The exponent of y is the e such that $|y| = 2^e \cdot x$ with $1 \leq x < 2$, so

$$\begin{aligned} &1 \leq 2^{-e}|y| < 2 \\ &0 \leq -e + \log_2 |y| < 1 \\ &e \leq \log_2 |y| < e + 1 \end{aligned}$$

or in other words e is the floor of $\log_2 |y|$.