

Number Representations

1 Storing an Integer using Two's Complement

An integer is stored in binary, also known as base 2. For example, the integer 23 is 10111_2 : $23 = 2^4 + 2^2 + 2^1 + 2^0$. On a typical machine, an integer is stored with a fixed number of bits. For example, consider an 8-bit machine. So here, 23 would be stored as 00010111.

The next question is: how to store negative numbers such as -23 . One idea is to reserve the first bit to indicate the sign (0 for positive and 1 for negative). Another idea is to store -23 by taking the representation for 23 and flipping every bit. The most common way is called *two's complement*:

To get the representation of $-x$, take the representation of x , flip every bit, and then add 1.

For example, -23 is stored as 11101001.

The advantage of two's complement—it turns out—is that one does not have to treat negative and positive numbers differently. Indeed, for internal arithmetic purposes, whether 11101001 represents -23 or 233 is irrelevant. And one can add, subtract, multiply and divide numbers, just like in decimal. For example, $(-23)+30$ is calculated as:

```

11101001
+00011110
-----
00000111

```

where the leftmost carry is discarded.

2 Storing a Floating-Point Number

You might have seen scientific notation: for example, Avogadro's number is written as 6.02214×10^{23} . This is the basic idea for *floating-point numbers*. Every real number can be written as $m \times 2^e$. The m part is called the *mantissa* or significand, and the e part the *exponent*. One can always adjust e and m such that $|m|$ is within some small range, say at least 1 and less than 2.

So, floating-point numbers are stored with a fixed number of bits for the mantissa and a fixed number of bits for the exponent. For example, a 32-bit machine might use

24 bits for the mantissa and 8 bits for the exponent. (Since our mantissa always starts $1.\dots$, the first 1 does not need to be stored, and so we get one bit more of precision.)

The fixed number of bits for the mantissa means that most real numbers (such as $1/6$ or π) cannot be stored perfectly. The mantissa can be chopped or rounded, depending on the implementation. For example, $1/6$ in binary is $0.0010101010\dots$; so the mantissa is $1.01010101\dots$ and the exponent -3 . If we have 7 bits for the mantissa, it is stored as **0101010** or **0101011**.

Negative numbers are stored with a negative mantissa, using a suitable scheme (usually not two's complement). See an example at http://en.wikipedia.org/wiki/Single_precision_floating-point_format.