Representation of real numbers and integers

Representing a number in a computer

Representing (or encoding) a number means to express it in binary form. Representing numbers in a computer is necessary in order for it to be able to store and manipulate them. However, the problem is that a mathematical number can be infinite (as great as desired), but the representation of a number in a computer must occupy a predefined number of bits. The key, then, is being able to predefine a number of bits, and how to intepret them, so that they can represent the figure as efficiently as possible. For this reason, it would be foolish to use 16 bits to encode a character (65536 possibilities) when less than 256 bits are typically used.

Representation of a natural number

A natural number is a positive integer or zero. The choice of how many bits to use depends on the range of numbers to be used. To encode the natural numbers between 0 and 255, all that is needed is 8 bits (a byte) as $2^8=256$. Generally speaking, *n*-bit encoding can be used for representing natural numbers between 0 and 2^n-1.
To represent a natural number, having defined how many bits will be used to code it, arrange the bits into a binary cell, with each bit placed according to its binary weight from right to left, then "fill" the unused bits with zeroes.

Representation of an integer

An integer is a whole number which may be negative. The number must therefore be encoded in such a way as to tell if it is positive or negative, and to follow the rules of addition. The trick involves using an encoding method called *twos complement*.

- **A positive integer or zero** will be represented in binary (base 2) as a natural number, except that the highest-weighted bit (the bit on the far left) represents the plus or minus sign. So for a positive integer or zero, this bit must be set to 0 (which corresponds to a plus sign, as 1 is a minus sign). Thus, if a natural number is encoded using 4 bits, the largest number possible will be 0111 (or 7 in decimal). Generally, the largest positive integer encoded using *n* bits will be $2^{n-1}-1$.
- **A negative integer** is encoded using twos complement. The principle of *twos complement*: Choose a negative number.
  - Take its absolute value (its positive equivalent)
  - It is represented in base 2 using n-1 bits

- Each bit is switched with its complement (i.e. the zeroes are all replaced by ones and vice versa)
- Add 1

Note that by adding a number and its twos complement the result is 0

Let's see this demonstrated in an example:
We want to encode the value -5 using 8 bits. To do so:

- write 5 in binary: 00000101
- switch it to its complement: 11111010
- add 1: 11111011
- the 8-bit binary representation of -5 is 11111011

Comments:

The highest-weighted bit is 1, so it is indeed a negative number.
If you add 5 and -5 (00000101 and 11111011) the sum is 0 (with remainder 1).

Representation of a real number

The goal is to represent a number with a decimal point in binary (for example, *101.01*, which is not read *one hundred one point zero one* because it is in fact a binary number, i.e. *5.25* in decimal) using the form 1.XXXXX... * 2^n (in our example, $1.0101*2^2$). IEEE standard 754 defines how to encode a real number.
This standard offers a way to code a number using 32 bits, and defines three components:

- the plus/minus sign is represented by one bit, the highest-weighted bit (furthest to the left)
- the exponent is encoded using 8 bits immediately after the sign
- the mantissa (the bits after the decimal point) with the remaining 23 bits

Thus, the coding follows the form:
seeeeeeeemmmmmmmmmmmmmmmmmmmmmmm

- the **s** represents the sign bit.
- each **e** represents an exponent bit
- each **m** represents a mantissa bit

However, there are some restrictions for exponents:

- the exponent 00000000 is forbidden
- the exponent 11111111 is forbidden However, they are sometimes used to report errors. This numeric configuration is called *NaN*, for *Not a number.*
- 127 (01111111) must be added to the exponent in order to convert the decimal to a real number in binary. The exponents, therefore, can range from -254 to 255

Thus, the formula for expressing real numbers is:

(-1)^S * 2^(E - 127) * (1 + F)

where:

- S is the sign bit and so 0 is understood as positive (-1^0=1).
- E is the exponent to which 127 must be added to obtain the encoded equivalent.
- F is the fraction part, the only one which is expressed, and which is added to 1 to perform the calculation.

Here is an example:
The value 525.5 is to be encoded.

- 525.5 is positive, so the first bit will be 0.
- Its representation in base 2 is: 1000001101.1
- By normalising it, we get: $1.0000011011*2^{9}$
- Adding 127 to the exponent, which is 9, gives 136, or in base 2: 10001000
- The mantissa is composed of the decimal part of 525.5 in normalised base 2, which is 0000011011.
- As the mantissa must take up 23 bits, zeroes must be added to complete it: 00000110110000000000000
- The binary representation of 525.5 under IEEE standard 754 is therefore: 0 1000 1000 00000110110000000000000 0100 0100 0000 0011 0110 0000 0000 0000 (4403600 in hexadecimal)

Here is another example, this time using a negative real number :
The value -0.625 is to be encoded.

- The s bit is 1, as 0.625 is negative.
- 0.625 is written in base 2 as follows: 0.101
- We want to write it in the form 1.01 x 2-1
- Consequently, the exponent is worth 1111110 as 127 - 1 = 126 (or 1111110 in binary)
- The mantissa is 01000000000000000000000 (only the digits after the decimal point are represented, as the whole number is always equal to 1)
- The binary representation of the number 0.625 under IEEE standard 754 is: 1 1111 1110 01000000000000000000000 1111 1111 0010 0000 0000 0000 0000 0000 (FF 20 00 00 in hexadecimal)

Source: http://en.kioskea.net/contents/62-representation-of-real-numbers-and-integers