

Addition and Subtraction

The addition or subtraction of numbers in any base is carried out using the same approach as that for decimal numbers. For example, the addition of 89_{10} and 7_{10} using a binary representation yields:

$$\begin{array}{r} 89_{10} \quad 1011001_2 \quad + \\ 7_{10} \quad \quad 111_2 \\ \hline \text{carry } 10_{10} \quad 0111110_2 \\ \hline 96_{10} \quad 1100000_2 \end{array}$$

The subtraction of 7_{10} from 89_{10} yields:

$$\begin{array}{r} 89_{10} \quad 1011001_2 \quad - \\ 7_{10} \quad \quad 111_2 \\ \hline \text{carry } 00_{10} \quad 0001100_2 \\ \hline 82_{10} \quad 1010010_2 \end{array}$$

Finite Representation

A computer may interpret and manipulate a set of bits as a numerical value. The representation of a number in a computer uses a fixed number of bits, even though there are usually a few hardware dependent sized representations used in a computer. In any finite representation the range of values that can be represented is limited.

If we consider the representation of non-negative (i.e. unsigned) number using n bits, then the minimum value is represented by n zeroes ($0 \times 2^{n-1} + \dots + 0 \times 2^1 + 0 \times 2^0 = 0$) and the maximum value is represented by n ones ($1 \times 2^{n-1} + \dots + 1 \times 2^1 + 1 \times 2^0 = 2^n - 1$).

Bits	Min	Max
4	0	15
8	0	255
16	0	65535
32	0	4294967295
64	0	18446744073709551615

Multiplication and Division

The multiplication and division of base two numbers can be performed in the same way as that of decimal numbers, unlike decimal you do not need to know your times tables.

The evaluation of $v \times u$ can be performed by calculating the impact each digit of u has on the result. Each digit of u is multiplied by v and the result *shifted left* by the power of that digit. For example (in decimal) $2 \times 23 = 2 \times 2 \times 10^1 + 2 \times 3 \times 10^0$. Multiplication follows the same pattern in any base, including binary.

$$\begin{array}{r} 10011_2 \quad \times \\ 110_2 \\ \hline 100110_2 \quad + \\ 1001100_2 \\ \hline 0011000_2 \quad \text{carry} \\ \hline 1110010_2 \end{array}$$

Long division is easier in binary than decimal, since for each power the dividend will only divide into the quotient zero or one times. The process of long division is identical, starting with the largest power (left most position) dividing the quotient by the dividend and calculating the remainder. The remainder becomes the new quotient and the process repeated.

$$\begin{array}{r|l} & 10011_2 \\ \hline 110_2 & 1110011_2 \\ & 1100000_2 \\ \hline & 10011_2 \\ & 1100_2 \\ \hline & 111_2 \\ & 110_2 \\ \hline \text{remainder} & 1_2 \end{array}$$

Representation of Signed Numbers

In order to represent signed numbers the computer must also encode the sign of the number in the available bits. At first glance a simple method would be to use a bit to indicate whether the number was positive or negative. If the representation of a number used 4 bits, then the numbers 5_{10} and -2_{10} could be represented by:

$$\begin{aligned} +5_{10} &= 0101_2 \\ -2_{10} &= 1010_2 \end{aligned}$$

Adding 0101_2 and 1010_2 using standard binary addition would yield 1111_2 (-7_{10}) where as the correct answer would be 0011_2 . As such this encoding requires the computer to know that the numbers are signed and alter its behaviour according to the signs of the numbers.

