Computer Numerical Representation

Andy Chong Sam

November 24, 2022

1 Introduction

Math is most commonly performed using a decimal system. We call this a decimal system because ten symbols are used: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Any number can be decomposed using a member from the set of symbols and a power of ten, take the number 1421 for example, which can be written as:

$$1421 = (1)(10)^3 + (4)(10)^2 + (2)(10)^1 + (1)(10)^0$$

Computers use a binary system, where the set of usable symbols has been reduced to just two: $\{0, 1\}$. This is referred to as a binary system. This document explains the algorithms to convert between decimal and binary systems. It will also explain how negative and fractional amounts are represented in binary.

2 Decimal to Binary

The algorithm to translate decimal to binary involves repeated divisions by 2, and recording the remainders. The stopping point is when the dividend becomes 1. The list of remainders is then reversed. We will use the same number as before, 1421 to illustrate this.

$$\begin{array}{r|l} 2 & 1421 \\ \hline 710 & \\ 14 & \\ \hline 02 & \\ 2 & \\ \hline 01 & \end{array} \quad \begin{array}{r|l} 2 & 710 \\ \hline 355 & \\ 6 & \\ \hline 11 & \\ 10 & \\ \hline 10 & \\ 10 & \\ \hline 0 & \end{array} \quad \begin{array}{r|l} 2 & 355 \\ \hline 177 & \\ 2 & \\ \hline 15 & \\ 14 & \\ \hline 15 & \\ 14 & \\ \hline 1 & \end{array} \quad \begin{array}{r|l} 2 & 177 \\ \hline 88 & \\ 16 & \\ \hline 17 & \\ 16 & \\ \hline 1 & \end{array} \quad \begin{array}{r|l} 2 & 88 \\ \hline 44 & \\ 8 & \\ \hline 08 & \\ 8 & \\ \hline 0 & \end{array} \quad \begin{array}{r|l} 2 & 44 \\ \hline 22 & \\ 4 & \\ \hline 04 & \\ 4 & \\ \hline 0 & \end{array} \quad \begin{array}{r|l} 2 & 22 \\ \hline 11 & \\ 2 & \\ \hline 02 & \\ 2 & \\ \hline 0 & \end{array} \quad \begin{array}{r|l} 2 & 11 \\ \hline 5 & \\ 10 & \\ \hline 1 & \end{array} \quad \begin{array}{r|l} 2 & 5 \\ \hline 2 & \\ 4 & \\ \hline 1 & \end{array} \quad \begin{array}{r|l} 2 & 2 \\ \hline 1 & \\ 2 & \\ \hline 0 & \end{array} \quad \begin{array}{r|l} 2 & 1 \\ \hline 0 & \end{array}$$

The recorded list of remainders is 10110001101. Reversing the list gets us the binary representation of 1421: **10110001101**.

Let's try the reverse process now. Suppose that I have 10110001101 and I want to transform that into a decimal number. Similar to the example in the introduction, we can assign place values to all the binary digits:

Place Value	10	9	8	7	6	5	4	3	2	1	0
Binary Digit	1	0	1	1	0	0	0	1	1	0	1

We then proceed to multiply each binary digit by two raised to the place value. All these terms will then be added together:

$$2^{10} + 2^8 + 2^7 + 2^3 + 2^2 + 1 = 1421$$

3 Negative Integers

The binary equivalent of a negative number is formulated using the two's complement technique. First, we take the one's complement which involves flipping each zero to a one, and each one to a zero. Finally we add a binary 1. Suppose we wanted to represent -1421:

Binary Digit	1	0	1	1	0	0	0	1	1	0	1
One's Complement	0	1	0	0	1	1	1	0	0	1	0

One's Complement	0	1	0	0	1	1	1	0	0	1	0
+ 1	0	0	0	0	0	0	0	0	0	0	1
Answer	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**1**

We are left with the result **01001110011**. This result occupies 10 positions (or 11 bits). In an actual computer, the most significant bit is known as the **sign bit**, and for negative numbers it will be set to 1. The left most bit is the most significant bit.

If we above result was processed through a 12 bit computer, -1421 would be stored as **101001110011**.

4 Fractional Amounts

For amounts less than one, we will multiply the number by 2 and observe whether or not the result is less than one or not. We will then take the fractional part of the result and repeat the process. The algorithm ends when we obtain a fractional result of 0.

We will evaluate 0.625 as an example:

Binary Digit	Value	Operation
	0.625	0.625 x 2
1	1.25	0.25 x 2
0	0.5	0.5 * 2
1	1.0	STOP

We can validate these results. Here are the digits with labeled placed values. Notice that as we are now dealing with values less than 1, the place values are now negative:

Place Value	0	-1	-2	-3
Binary Digit	0	1	0	1

We perform the following calculation:

$$2^{-1} + 2^{-3} = 0.625$$

Some fractional points cannot be represent with a finite number of bits. Instead we get a never ending but sometimes predictable sequence of bits. Before we see an actual example of the conversion, consider that this result is not unusual even in the decimal system. For example, it is not possible to represent the fraction $\frac{1}{3}$ in a finite sequence, instead we get the unending sequence: 0.33333...

Consider the case of converting 0.2 into binary:

Binary Digit	Value	Operation
	0.2	0.2 x 2
0	0.4	0.4 x 2
0	0.8	0.8 x 2
1	1.6	0.6 x 2
1	1.2	0.2 x 2
0	0.4	0.4 x 2
...		

The stopping condition of the algorithm is never encountered, and we end up with an endless sequence of **001100110011**...