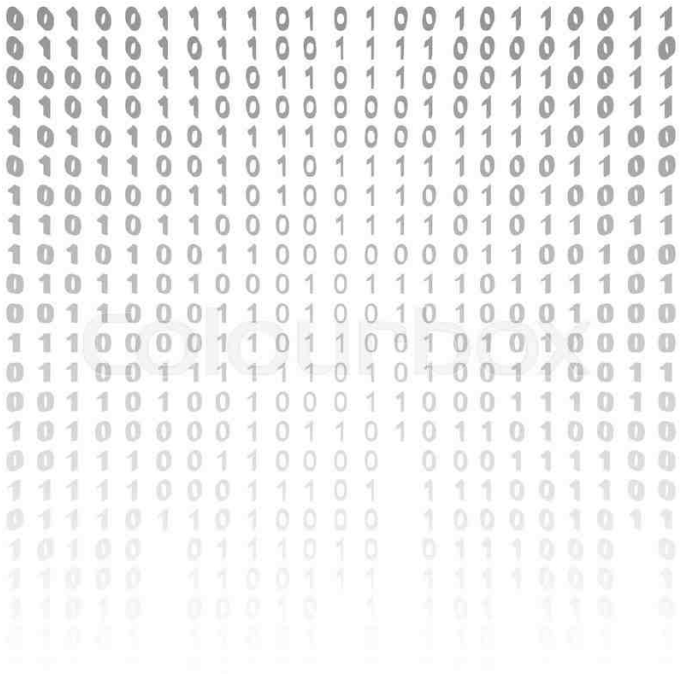
October, 2016

# Signed Magnitude, One's Complement & Two's Complement



Justin “Redhart” A. E.
profile.darkstack.org/redhart

# PREFACE

A modern digital computer represents data using the binary numeral system. Text, numbers, pictures, audio, and nearly any other form of information can be converted into a string of bits, or binary digits, each of which has a value of 1 or 0.

So what happens when the computer is expected to store a negative number or perform a subtraction? This is where complementation is required. The ideal notation of representing signed numbers is the 2's complement notation.

Addition is one of the basic operations that a computer can accomplish using basic logic gates. All other operations are just variations or manipulations of bits involving addition.

In base 2, subtraction is achieved usually by using 2's complement addition (adding one number to the 2's complement of the other number).

Multiplication and division are achieved using a shift and add/subtract operations repeated a certain number of times.

Shifting is achieved by multiplication or division by multiples of 2, which takes us back to addition.

This paper discusses the ways in which we can represent signed numbers in the binary numeral system. On that note, we will look at signed magnitude, 1's complement and 2's complement.

# SIGNED MAGNITUDE

Signed magnitude is a term associated with flipping the leftmost bit, known as the Most Significant Bit (MSB) of a system of bits to represent a negative number.

0 represents a positive number

1 represents a negative number

To negate the decimal 3 (011 in binary), we simply invert the MSB.

**0**11 -> **1**11

But is it really okay?

Let's check this by performing a subtraction operation.

Don't forget that computers only have an addition unit, so in order to perform a subtraction operation, we actually have to negate the operand on the RHS and add it to the operand on the LHS.

Say we want to subtract 2 (010) from 3 (011), the steps are:

1. Invert the MSB of 2 in binary, turning 010 to 110.
2. Add 110 to 011 in binary form and obtain your result.

The result of that expression would be 1010 (010 with a carry bit 1) which is wrong because 1 which is the correct answer in binary form is 001. This led to the introduction of 1's Complement.

# ONE'S COMPLEMENT

The difference between signed magnitude and 1's complement is that instead of inverting only the most significant value, all the bits involved are inverted. For example, 010 becomes 101. Also, if we have a carry bit, we add it to the final result.
$2^n$ gives the total number of decimal values a bit system of size *n* can represent.

For simplicity, we will still work with 3 bits. So $2^3 = 8$.

| DECIMAL | BINARY | 1's COMPLEMENT |
|---|---|---|
| 0 | 000 | +0 |
| 1 | 001 | +1 |
| 2 | 010 | +2 |
| 3 | 011 | +3 |
| 4 | 100 | -3 |
| 5 | 101 | -2 |
| 6 | 110 | -1 |
| 7 | 111 | -0 |

*Table 1: One's Complement*

From the table above, it can be seen that one can only represent signed numbers from -3 to +3 for n = 3 in 1's complement. The reason being that after +3, the number that follows (+4) already has 1 as its MSB.

Also, notice the nonsensical negative zero (-0) matching 7 (111) which we obtain by inverting 000.

**Example 1.1:**

Find the value of 3 - 2 in binary form of 3 bits.

**Solution 1.1:**

3 -> 011

2 -> 010

-2 -> 101 (invert all bits: 1's complement)

011 + 101 = 1 000

Adding 101 to 011 becomes 1000. Then we add 1 which is the carry bit to 000 to obtain 001 which is correct in binary form. This is okay until when tested with 3 - 3 in binary, the addition of 011 and 100 which gives us 111 which is negative zero (-0) in *Table 1*.

**Example 1.2:**

Find the value of 3 - 3 in binary form of 3 bits.

**Solution 1.2:**

3 -> 011

-3 -> 100 (invert all bits: 1's complement)

011 + 100 = 111

Goes as planned, but negative zero… really? We call on 2's complement to save the day.

# TWO'S COMPLEMENT

2's complement is just like 1's complement, except after the inverting the bits, 1 is added to it. Also, if we have a carry bit, we simply discard it.

| DECIMAL | BINARY | 2's COMPLEMENT |
|---|---|---|
| 0 | 000 | +0 |
| 1 | 001 | +1 |
| 2 | 010 | +2 |
| 3 | 011 | +3 |
| 4 | 100 | +4 |
| 5 | 101 | -3 |
| 6 | 110 | -2 |
| 7 | 111 | -1 |

*Table 2: Two's Complement*

Here, inverting 0 (000) still gives 0 (00). Let's see how:

0 -> 000

0 -> 111 (inverting all bits: 1's complement)

0 -> 1000 (adding 1 to it)

0 -> 000 (discard the carry bit: 2's complement)

**Example 2.1:**

Find the value of 3 - 3 in binary form of 3 bits.

**Solution 2.1:**

3 -> 011

-3 -> 100 (invert all bits: 1's complement)

-3 -> 101 (add 1 to it: 2's complement)

011 + 101 = 1 000

Then add 101 to 011 and the result is 1000. We have a carry bit 1, so we discard it and we have 000. No more negative zero. Problem solved.

**Example 2.2:**

Find the value of 3 - 3 in binary form of a byte (8 bits).

**Solution 2.2:**

3 -> 0000 0011

-3 -> 1111 1100 (invert all bits: 1's complement)

-3 -> 1111 1101 (add 1 to it: 2's complement)

0000 0011 + 1111 1101 = 1 0000 0000

Discarding the carry bit, we have 0000 0000 which is correct.

# GLOSSARY

- **LHS**: Left Hand Side. Used in this context to refer to the operand on the left of the binary operator.

- **MSB**: Most Significant Bit. The leftmost bit in a given binary number.

- **RHS**: Right Hand Side. Used in this context to refer to the operand on the right of the binary operator.