Two’s Complement Representation: Theory and Examples

November 16, 2017 by Steve Arar (/author/steve-arar)

The two’s complement representation is a basic technique in digital arithmetic which allows us to replace a subtraction operation with an addition.

This article will first review the theory of the two’s complement representation along with some examples. Then, we will briefly discuss the block diagram of an adder/subtractor.

Two’s complement representation is a way to represent the signed numbers in a digital computer. The main goal is to develop a technique which replaces a subtraction operation with an addition. In this way, we will be able to use the same circuit to perform both addition and subtraction. This will reduce the number of gates and, consequently, the size, power, and cost of the system. In fact, without the complement concept, we would have to use a method similar to the paper-and-pencil approach that we use when working with the decimal numbers. This would require use of two different blocks to perform addition and subtraction. Moreover, we would have to make several decisions before being able to apply the inputs to our adder/subtractor blocks (to see the problems of this hypothetical computer, read section 2.4.1 of this book (https://www.amazon.com/Essentials-Computer-Organization-Architecture/dp/1449600069)).

Background: Playing with Unsigned Numbers

Assume that we have an adder which takes two four-bit numbers, $a = a_3a_2a_1a_0$

and $b = b_3b_2b_1b_0$, along with an input carry, c_{in}, and calculates the sum $a + b + c_{in}$. How can we use this adder to perform a subtraction, i.e. $S = a - b$? Adding a constant, such as M, to S, and, then, subtracting the same constant from S will not change the result:

$$S = a + M - b - M$$

Equation 1

For a sufficiently large M, we have

$$B = M - b > 0$$

Equation 2

and Equation 1 can be rewritten as

$$S = a + B - M$$

Equation 3

Equation 3 requires one addition and one subtraction. Besides, to use Equation 3, we need to calculate another subtraction, i.e. $B = M - b$. It seems that we are making things more complicated than before because $S = a - b$ requires only one subtraction but Equation 3 requires one addition and two subtractions! However, we need to note that the two subtractions required for Equation 3 has one thing in common: both of these subtractions involve a common operand, M. This leads us to the idea that maybe we can find a suitable M which allows us to simplify the subtractions of Equations 2 and 3. If possible, this would allow us to use Equation 3 to replace a subtraction with an addition. So the question remains: what is the suitable value for the constant, M? As we will see in the following sections, $M = 2^k$ proves to be the appropriate value for a k-bit number.

Let's assume that b is a four-bit number and examine the subtraction $M - b$. We can use $M = 10000_{(2)} = 16_{(10)}$ to simplify the subtraction. This simplification is possible, because we can write M=(M-1)+1and obtain

$$B = 10000_{(2)} - b = (01111_{(2)} - b) + 00001_{(2)}$$

We can easily calculate $01111_{(2)} - b$ because it is, in fact, the bitwise complement of b. For example, if $b = 0011_{(2)}$, then

$$B = (01111_{(2)} - 0011_{(2)}) + 00001_{(2)} = 01100_{(2)} + 00001_{(2)}$$

As you can see, the subtraction inside the parentheses is the bitwise complement of b. Therefore, to perform the subtraction $M - b$, we only need to find the bitwise complement of b and then add $00001_{(2)}$ to the result. We will see later in this article that, from an implementation point of view, it's an easy task to add $00001_{(2)}$ to the bitwise complement of a number.

Having B, we can use an adder to calculate $a + B$ in Equation 3. So far, we have circumvented use of a subtractor, but to achieve the final result, we need to subtract M from $a + B$. How can we get this done? In the above example, we are considering four-bit numbers, hence, the largest value of S will be for $a = 1111_{(2)}$ and $b = 0000_{(2)}$ which gives $S = 1111_{(2)} = 15_{(10)}$. So four bits are sufficient to represent $a - b$. Choosing $M = 10000_{(2)} = 16_{(10)}$, and adding M to $a - b$, we are altering only the fifth bit position of $a - b$. We can use this observation to subtract M from $a + B$ in Equation 3. In other words, to perform the subtraction of Equation 3, we only need to discard the bit at the fifth position. This is, in fact, equivalent to performing modulo-M calculations which means that we are restricting the result of the calculations to be less than or equal to $M - 1$.

The above discussion is summarized as follows: if a and b are two k-bit numbers,

the subtraction $a - b$ can be calculated by adding $M - b$ to a and discarding the bit position $k + 1$. Here M, called the complementation constant, is equal to 2^k.

In the modulo-M arithmetic, $M - b$ acts as the opposite of b and is called the two's complement of b (for $M = 2^k$). This opposite nature is obvious because adding b to $M - b$ gives M which is equal to 0 in the modulo-M arithmetic. Based on this idea, we can define the opposite of b as $M - b$. As discussed above, we can calculate the two's complement of a number by first calculating its bitwise complement and then adding 1 to the result.

Example 1:

Assume that we are working with unsigned four-bit numbers. Subtract $b = 0110_{(2)} = 6_{(10)}$ *from* $a = 1011_{(2)} = 11_{(10)}$ *using the two's complement representation.*

Since we are dealing with four-bit numbers, M will be $2^k = 2^4$. Based on the above discussion, we can represent $-b$ with $B = M - b$. We obtain

$$B = (M - 1) - b + 1 = 01111_{(2)} - 0110_{(2)} + 00001_{(2)} = 01001_{(2)} + 000($$

Now, we can add B to a which gives

$$a + B = 1011_{(2)} + 01010_{(2)} = 10101_{(2)}$$

Discarding the fifth bit, we obtain

$$a - b = 0101_{(2)} = 5_{(10)}$$

Example 2:

Assume that we are working with unsigned five-bit numbers. Subtract $b = 01001_{(2)} = 9_{(10)}$ *from* $a = 10111_{(2)} = 23_{(10)}$ *using the two’s complement representation.*

Since we are dealing with five-bit numbers, M will be 2^5. We can represent $-b$ with $B = M - b$. We obtain

$$B = (M - 1) - b + 1 = 011111_{(2)} - 01001_{(2)} + 000001_{(2)} = 010110_{(2)} +$$

Now, we can add B to a which gives

$$a + B = 10111_{(2)} + 010111_{(2)} = 101110_{(2)}$$

Discarding the sixth bit, we obtain

$$a - b = 01110_{(2)} = 14_{(10)}$$

How to Represent Signed Numbers?

Adding b to $M - b$ gives M which is equal to zero in modulo-M arithmetic. That’s why, in the modulo-M arithmetic, we can consider $M - b$ as the opposite of b. The second column of Table 1 below lists all the possible combinations with three bits. The corresponding decimal value of these three-bit numbers are given in the first column. Assuming $M = 2^3$, we obtain the opposite of the decimal values in the first column as given in the third column. You can easily verify that the sum of the values in the second and third columns is equal to zero in modulo-M arithmetic. The table shows that a three-bit number may have two different interpretations. For example, 010 can be interpreted as either the positive decimal number 2 (the second column of the table) or a negative decimal number -6 (the third column of the table). To distinguish between these two cases, we can use an extra bit in the

leftmost bit position. This bit, called a sign bit, specifies if a given number is positive or negative.

Table 1

Decimal value of b	Representation of +b	Representation of -b
0	000	000
1	001	111
2	010	110
3	011	101
4	100	100
5	101	011
6	110	010
7	111	001

When the sign bit is zero, the number is interpreted as a positive one. In this case, the remaining three bits will give the magnitude of the decimal value according to the first column of the table. For example, $0010_{(2)}$ will be considered as $+2$. When the sign bit is one, the number will be interpreted as a negative value and we must use the third column of the table. For example, $1010_{(2)}$ will be interpreted as -6. In fact, we are using four bits to represent a signed number whose magnitude can be coded with three bits.

Now, let’s include the sign bit in the leftmost position of our three-bit numbers and examine the possible combinations. Figure 1 shows all the possible combinations for a four-bit number. If we consider these numbers as unsigned values, we will obtain the decimal values outside of the circle. For example, $1010_{(2)}$ is equal to decimal 10 considered as an unsigned number. However, if we assume these four-bit numbers as signed values, then, we will obtain the signed decimal values inside the circle. For example, $1101_{(2)}$ is a negative number since sign bit is one. Using the third column of the above table or, equivalently, calculating its two’s complement, we see that $101_{(2)}$ corresponds to decimal 3, hence, $1101_{(2)} = -3_{(10)}$.

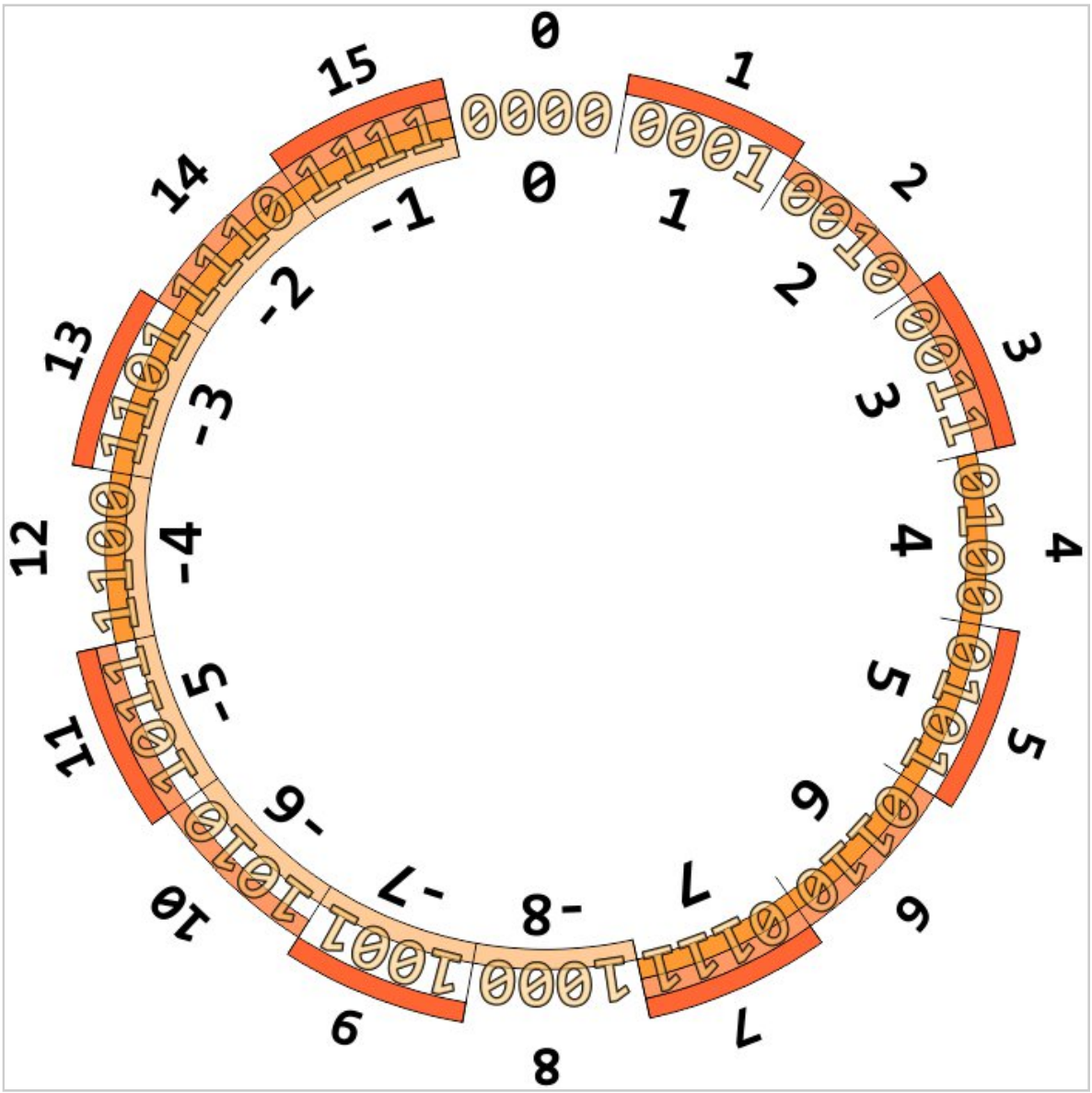

Figure 1. *The four-bit two’s complement representation circle.*

Considering the decimal values of Figure 1, we can verify the complement concept with the complementation constant equal to $2^4 = 16$. For example, a negative value, say -5, is represented by 11 which is actually $16 - 5$. Please note that all the negative numbers have a sign bit of one. This helps us to easily recognize a negative number in a system which deals with signed numbers.

Moreover, Figure 1 illustrates the range of the numbers that can be covered in the signed and unsigned representations. As shown in this figure, interpreting as

unsigned numbers, we can incorporate four bits to represent values from 0 to 15. However, with a signed representation, we can use four bits to code the values from -8 to $+7$. Note the asymmetric range of the signed two’s complement representation. We will discuss in a future article that this asymmetric range of two’s complement representation can lead to overflow and the programmer needs to be prudent to avoid possible mistakes resulting from this behavior.

To summarize this section, when we are assuming a signed two’s complement representation, we should first consider the sign bit. If the sign bit is zero, we can find the corresponding decimal value as we do with unsigned numbers. However, when the sign bit is one, the corresponding decimal value is the opposite of the two’s complement of the number.

Example 3:

Assuming a signed two’s complement representation, find the decimal value corresponding to $a = 01110_{(2)}$ *and* $b = 10110_{(2)}$.

Since a is positive, its decimal equivalent will be $a = 2^3 + 2^2 + 2^1 = 14$. However, b is a negative number and hence

$$b = -(two's\ complement\ of\ b) = -(01001_{(2)} + 00001_{(2)}) = -(01010_{(2)})$$

Now that we are familiar with the concept of signed two’s complement representation, we can more easily perform additions and subtractions.

Example 4:

Assume that we are working with signed five-bit numbers. Subtract $b = 00101_{(2)}$ *from* $a = 01001_{(2)}$ *using the two’s complement representation.*

Both a and b are positive and we have $a = 9_{(10)}$ and $b = 5_{(10)}$. To calculate $a - b$, we need to find $-b$ and add it to a. We obtain

$$-b = \ two's\ complement\ of\ b = 11010_{(2)} + 00001_{(2)} = 11011_{(2)}$$

Then, we obtain

$$a + (-b) = 01001_{(2)} + 11011_{(2)} = 100100_{(2)}$$

Since we are working with five-bit numbers, we should discard the sixth bit of the result. Therefore, we have

$$a - b = 00100_{(2)} = 4_{(10)}$$

Considering the decimal values, we can verify that the result is correct.

Example 5:

Assume that we are working with signed five-bit numbers. Subtract $b = 00110_{(2)}$ *from* $a = 10111_{(2)}$ *using the two’s complement representation.*

Since we are dealing with signed numbers, b is positive and equal to $6_{(10)}$.
However, a is a negative number and hence

$$a = -(two's\ complement\ of\ a) = -(01000_{(2)} + 00001_{(2)}) = -(01001_{(2)}$$

To calculate $a - b$, we need to find $-b$ and add it to a. We obtain

$$-b = \ two's\ complement\ of\ b = 11001_{(2)} + 00001_{(2)} = 11010_{(2)}$$

Then, we have

$$a + (-b) = 10111_{(2)} + 11010_{(2)} = 110001_{(2)}$$

We are working with five-bit numbers, so we should discard the sixth bit of the result. Hence, we obtain

$$a - b = 10001_{(2)}$$

We can find the decimal equivalent of this negative number as

$$a - b = -(01111_{(2)}) = -15_{(10)}$$

In these two examples, we discarded the sixth bit but the results were correct because the two’s complement representation is based on modulo arithmetic.

Example 6:

Assume that we are working with signed five-bit numbers. Subtract $b = 01001_{(2)}$ *from* $a = 10111_{(2)}$ *using the two’s complement representation.*

Since we are dealing with signed numbers, b is positive and equal to $9_{(10)}$.
However, a is a negative number and hence

$$a = -(two's\ complement\ of\ a) = -(01000_{(2)} + 00001_{(2)}) = -(01001_{(2)}$$

To calculate $a - b$, we need to find $-b$ and add it to a. We obtain

$$-b = two's\ complement\ of\ b = 10110_{(2)} + 00001_{(2)} = 10111_{(2)}$$

Then, we have

$$a + (-b) = 10111_{(2)} + 10111_{(2)} = 101110_{(2)}$$

We are working with five-bit numbers, so we should discard the sixth bit of the result. Hence, we obtain

$$a - b = 01110_{(2)} = 14_{(10)}$$

While we were expecting the result to be $a - b = -9_{(10)} - 9_{(10)} = -18_{(10)}$, our calculation has led to a positive number. This is due to the fact that the largest positive number that can be represented with five bits is $01111_{(2)} = 15_{(10)}$. This example shows that we always have to check the result of the calculations to make sure that overflow has not occurred and the results are valid. Overflow occurs if two positive numbers are added together and the result is negative, or if two negative numbers are added together and the result is positive. With the two’s complement representation, it is not possible to have overflow if a positive and a negative number are added together. To read about a simple method of overflow detection, see section 2.4.2 of this book (https://www.amazon.com/Essentials-Computer-Organization-Architecture/dp/1449600069).

The Block Diagram of a Simple Adder/Subtractor

As mentioned at the beginning of the article, the main goal of the two’s complement representation is to develop a technique which allows us to perform both addition and subtraction operations using a single circuit. To this end, we will

need an adder which takes two binary numbers, a and b, along with an input carry, c_{in}, and calculates the sum $a + b + c_{in}$. Figure 2 shows the block diagram of the two’s-complement based adder/subtractor. In this figure, when $Sub/\overline{Add}$ is zero, the “Selective Component” applies y to the b input of the adder and c_{in} would be zero. Hence, the adder output will be $a + b + c_{in} = x + y$. However, when $Sub/\overline{Add} = 1$, the “Selective Component” applies the bitwise complement of y, shown by y^{compl} in the figure, to the b input of the adder. Moreover, in this case, c_{in} will be one. Therefore, the adder will compute $a + b + c_{in} = x + (y^{compl} + 1)$. As mentioned in the previous sections, the term $y^{compl} + 1$ is equal to the two’s complement of y. Hence, the adder output will actually calculate $x - y$.

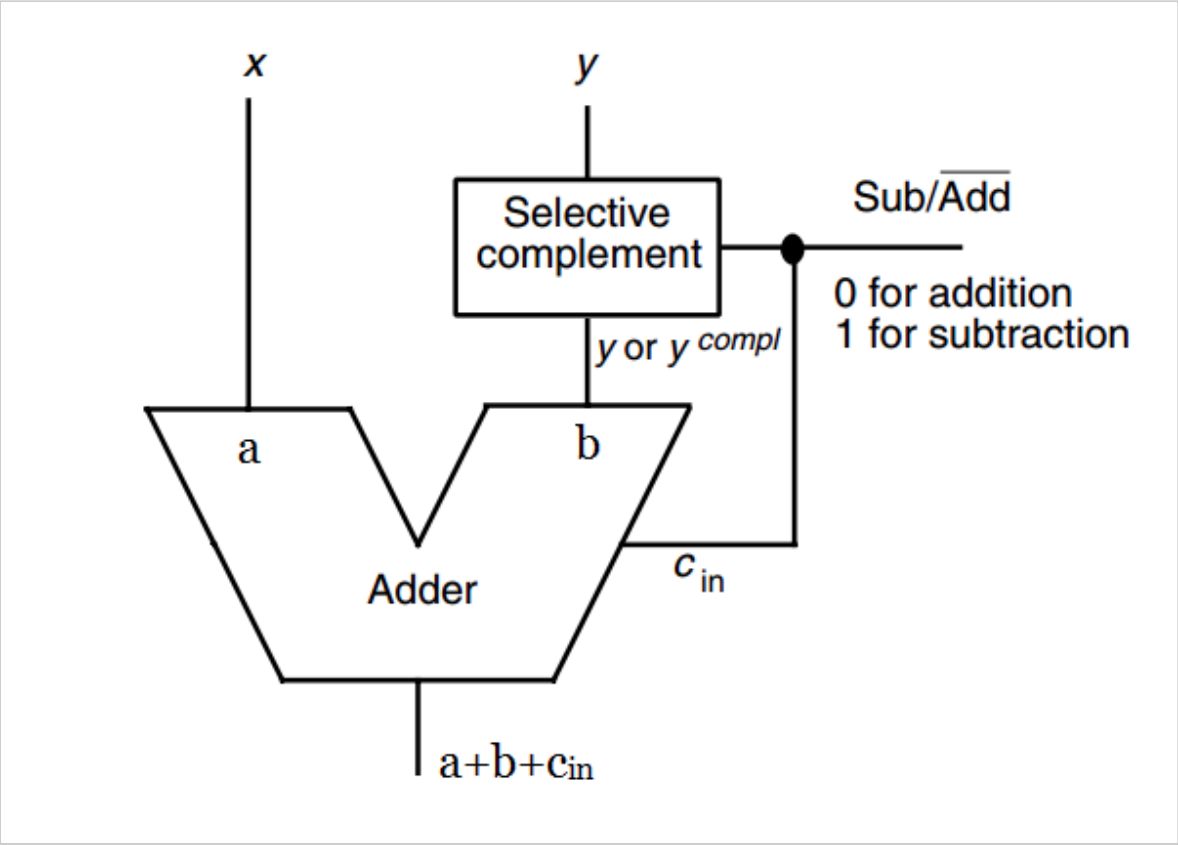

Figure 2. *The block diagram of an adder/subtractor. Image courtesy of Computer Arithmetic: Algorithms and Hardware Designs (https://www.amazon.com/Computer-Arithmetic-Algorithms-Electrical-Engineering/dp/0195328485).*

Summary