A Tutorial on
2's Complement Number System

Janak H Patel
Dept. of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign

Taking the mystery out of the 2's complement code

Modulo Arithmetic

What is Modulo Arithmetic? We use modulo arithmetic daily without giving it a thought. Clocks use a modulo-12 arithmetic. So, when we add 9 O' clock with 5 more hours we get 2'O clock. What we did was to add 9+5=14 and subtract 12. In mathematical terms 14 mod 12 = 2. In general N mod m is an integer in the finite set {0,1, …, m-1} and is computed as the Remainder of the division N/m. In formal Euclid's Division Theorem –

N = Q · m + R where m is the divider, Q is the Quotient and
R is the Remainder satisfying $0 \leq R \leq (m-1)$
In our number system, Quotient plays no role and can be ignored.
Examples:
5 mod 12 = 5
15 mod 12 = 3
17 mod 12 = 5
12 mod 12 = 0
Now consider the following two modulo additions:
5 + 9 = 2 (mod 12) and 5+(-3) = 2 (mod 12).
In other words, in additions, 9 has the same effect as (-3). If we restrict our operation to addition and subtraction, it can be seen that in modulo 12 arithmetic we can use 9 whenever we need (-3). Similarly, we can use number 8 in place of (-4), 10 in place of (-2) and so on for the complete table.
For –1 use 11
For –2 use 10
For –3 use 9
For –4 use 8
For –5 use 7
For –6 use 6
Therefore, we have a finite number system consisting of 12 integers {0,1,2,3,4,5,6,7,8,9,10,11} which can also be viewed as a system with 12 integers {0,1,2,3,4,5, -6, -5, -4, -3, -2, -1}. Advantage of modulo arithmetic is that we can perform subtraction by using addition on positive numbers. Thus, to perform 5-3 we can perform 5+9 (mod 12).

Another familiar example is use of modulo 360 arithmetic in dealing with angles in plane geometry. We know that angle –90 degrees is same as +270 degrees. Similarly, -180 is same as +180 degrees. Therefore, using positive integers such as +270 to represent negative integers such as –90 does not look strange in measurement of angles.

This is exactly what happens with 2's complement system. Let us take a 4-bit system. It represents 16 distinct integers {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}. All additions and subtractions are performed using modulo 16 arithmetic. Therefore 7+12=3 (mod 16). So is 7-4=3 (mod 16). Therefore, 12 behaves as if we are using (-4). So, to compute the representative integer for (-n), we simply compute 16-n. So

-1 is 15
-2 is 14
-3 is 13
-4 is 12
-5 is 11
-6 is 10
-7 is 9
-8 is 8.

We could keep either +8 or –8 in our number system but we chose to keep –8 for a reason that will become clear later. So, the complete system consists of {0,1,2,3,4,5,6,7,-8,-7,-6,-5,-4,-3,-2,-1}. Again, here also all subtractions are performed using only addition modulo 16. Thus, for example to perform 7-5 we can perform the addition 7+11 (mod 16), where 11 is the code for (-5).

Modulo addition is very natural to binary adders. The carry that goes out of the most significant bit is the modulus! Thus in 4-bit adder the "Carry Out" has a weight of 16, which when goes out means we subtract 16 from a sum which was greater than 15. Therefore, 7+11 gives us a sum of 18, but the carry-out takes away 16, giving us a sum of 2, which is exactly the same as (18 mod 16).

How is modulo arithmetic related to 2's complement arithmetic? The simple answer is, they are the same! In modulo 16 arithmetic, negation of n is simply (16-n). "Complement" means, "negate". Therefore, in 4-bit system (i.e., modulo 16 system), 2's complement of a number n is just (16-n).

Thus 2's complement of 5 is (16-5) = 11. Complete 4-bit modulo-16 system is shown below.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	-8	-7	-6	-5	-4	-3	-2	-1
0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**

Notice that all negative numbers (-8 through –1) have a code whose values are (8 through 15). In binary encoding, numbers 8 through 15 have a 1 in the most significant position, while this bit is 0 for numbers 0 through 7. This is a simple way to see if an encoding is a negative number. Therefore, in 2's complement system it is customary to call the most significant bit the "sign bit." It is not a real symbolic sign like + or -, this bit is an integral part of the number and should never be separated when performing additions and subtractions. One must remember that all operations in the hardware are done on the positive integers (0 through 15) without regard to our interpretation of positive and negative numbers. Now it should be clear why we chose to have -8 but not +8. It is for practical convenience. (In true modulo number system both 8 and -8 are present).

Clarification: "complement" means "negate". Therefore, a 2's complement of n is (2's complement-code of negative of n). The number n can be positive or negative.
E.g., 2's complement of –7 is 2's complement-code of negation of (-7) which is 0111
2's complement of 7 is 2's complement-code of negation of (7), which is 1001.
The terms code and representation are used interchangeably. Thus 2's complement representation of –7 is 1001. So, the meanings for "**2's complement of n**" and "**2's complement-representation of n**" are very different.

Summary

For n-bit integers, 2's complement arithmetic is synonyms with (modulo 2^n) arithmetic. All additions are performed modulo 2^n. For an integer k, 2's complement of k is (2^n - k). For example, in an 8-bit system, addition is performed modulo 256, and 2's complement of k is (256 - k). The modulo additions are performed on the set of integers {0,1, …, 127,128, …, 254,255}. The numbers that are representable in the 2's complement system are: {0,1, ….,126,127,-128,-127,….,-2,-1}.

Mechanical Procedure to Form 2's Complement of a Binary Number

Consider the above 8-bit system. The modulus **256** in the binary form is : **1 0000 0000**. Which is the same as **255 +1 = 1111 1111 + 0000 0001**
Now to form a 2's Complement of say **0011 1001** we subtract it from **256.** In the binary it is **1111 1111 – 0011 1001 + 0000 0001**
The subtraction from **1111 1111** is very easy. Every bit is complemented. This gives us **1100 0110 + 0000 0001** and the result is **1100 0111**
This is the classic mechanical procedure.

- ***To take 2's Complement of a Binary Number, complement each bit and then add 1 to the resulting number***.

The finite set of n-bit integers are: non-negative numbers range from 0 to (2^{n-1} -1) and the negative numbers in our 2's Complement system are the integers from 2^{n-1} to (2^n – 1). All operations are done on integers from 0 to (2^n – 1). The Carry Out of the adder is the Modulus 2^n And the Carry Out is ignored. It is possible that when we add two large positive integers, the sum may not fit in our n-bit system. This is called ***an Overflow***. Programming errors can produce such overflows and should be detected in the hardware.