# Two's Complement Representation

Integers are represented in a modified form of base two known as two's complement. For illustration purposes, we will consider 8-bit integers:

| $b_7$ | $b_6$ | $b_5$ | $b_4$ | $b_3$ | $b_2$ | $b_1$ | $b_0$ |
|---|---|---|---|---|---|---|---|

In the diagram above $b_0$ denotes the low-order bit. Bit $b_7$ is used to distinguish positive and negative numbers. If $b_7 = 0$ then the number is positive; if $b_7 = 1$ the number is negative.

**Positive Numbers:**

We have bits $b_6, b_5, b_4, b_3, b_2, b_1, b_0$ to represent positive numbers. The columns represent the corresponding powers of two, so a positive number $x$ would be:

$$x = b_6 2^6 + b_5 2^5 + b_4 2^4 + b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0 2^0$$

So for example, the two's complement number:

| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

$= 2^5 + 2^3 + 2^2 + 2^0 = 32 + 8 + 4 + 1 = 45$

The largest positive number we can represent is:

| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|

$= 127$

In general, the largest positive number represented by $n$ bits (in two's complement form) is $2^{n-1} - 1$. The smallest negative number using $n$-bits is $-2^{n-1}$.

In the case of 8 bits we have $2^7 - 1 = 127$. In the case of 16 bits we have $2^{15} - 1 = 32767$ and in the case of 32 bits, we have $2^{31} - 1 = 2147483647$.

**Negative Numbers:**

Negative numbers are formed by taking the positive form, complementing[1] and adding one. Consider the number -37. We start with +37:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | $= 37$ |
| | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | (complement) |
| + | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | (add one) |
| = | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | $= -37$ |

There is a slight asymmetry in the two's complement representation. While the largest (positive) number we can represent in 8 bits is 127, the smallest (negative) number we can represent in 8 bits is -128. For 16-bit integers the range is -32768 to +32767.

There are several advantages of two's complement representation:

- Unique representation for zero.
- Applying the complement-and-add-one process twice brings us back to the same number. I.e., $-(-x) = x$.
- Same logic circuit can be used to add both positive and negative integers and for unsigned integers. Subtraction performed by logic circuits via negating and adding.
- Sanity prevails; i.e., $x + (-x) = 0$.

[1] Here "complement" means to change all the 0 bits to 1 and all the 1 bits to 0.

When adding negative numbers, there is the possibility that the high order bit will "carry out" of the leftmost bit position. This is called overflow. By default, overflow is ignored.

We can demonstrate the limitations of storage with a simple C++ program. For simplicity, we use 16-bit integers.

```
//
// Program to demonstrate two's complement
// using 16-bit signed integers.
//
#include <iostream>
using namespace std ;

int main()
{
   short int a, b, c, d, e, f, g ;

   // First we demonstrate wrap-around.
   a = 32767 ;  // Largest positive 16-bit signed integer.
   b = a + 1 ;
   cout << "a = " << a <<  ", b = " << b << endl ;

   // Wrap-around going the other way.
   c = -32768 ;  // Smallest negative 16-bit signed integer.
   d = c - 1 ;
   cout << "c = " << c <<  ", d = " << d << endl ;

   // Demonstrate two's complement.
   e = 37 ;       //  Just a number.
   f = ~e + 1 ;   //  Bit-wise complement and add one.
   cout << "e = " << e <<  ", f = " << f << endl ;

   // Apply the same complement and add one again.
   g = ~f + 1 ;   //  Bit-wise complement and add one.
   cout << "g = " << g << endl ;
}
```

```
---------------------- Sample Session ----------------------
fini% g++ demo.cc
fini% a.out
a = 32767, b = -32768
c = -32768, d = 32767
e = 37, f = -37
g = 37
```