# **Number Systems (Part 2)**

**Representing Negative Numbers in Binary**

We can represent negative numbers in several ways. The simplest of all is to set the most left digit as a flag bit (e.g. 0 for positive and 1 for negative).

This would be represented as:

**Method 1:**

|  |  |
| --- | --- |
| Example 1 | Example 2 |
| 0100 = 4 | 0011 = 3 |
| 1100 = -4 | 1011 = -3 |

The disadvantage of this is you can only represent a power of 2 less that would you should be able to represent with a range of numbers. E.g. If you had 5 digits, that would be you could store 32 numbers normally, but because you only have 4 digits available (as the left most bit is used to represent the sign); You can only store 16 numbers.

Another method that is the most popular is called 2’s compliment. It is widely used as it is simplistic yet has little draw backs. 2’s compliment is used by flipping the polarity of a digit and add one to the number.

This would be represented as:

**Method 2:**

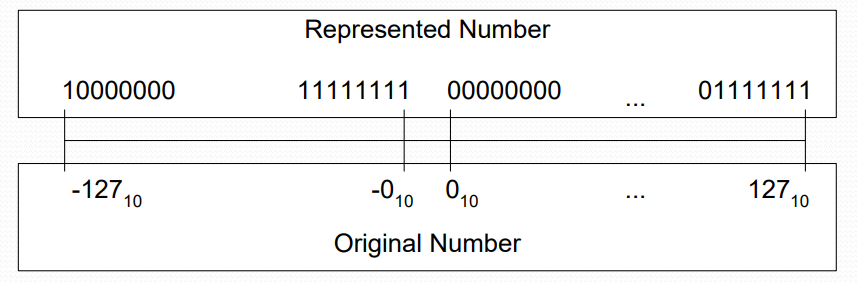
|  |  |  |
| --- | --- | --- |
| Steps | Example 1 | Example 2 |
| Step 1 | 10011001 = 153 | 10110100 = 180 |
| Step 2: Flip Digits | 01100110 | 01001011 |
| Step 3: Add 1 | 01100110  +1  = 01100111 | 01001011  +1  = 01001100 |

**What is meant by 9’s, 1’s, 10’s and 2’s Compliment?**

**Definition:** 1’s compliment is similar to what we are after doing while representing negative numbers in binary. All we have to do with 1’s complement is flip the polarity of the digits. E.g. (Original Number: 10110, 1’s Compliment Number: 01001)

Example of this being used:

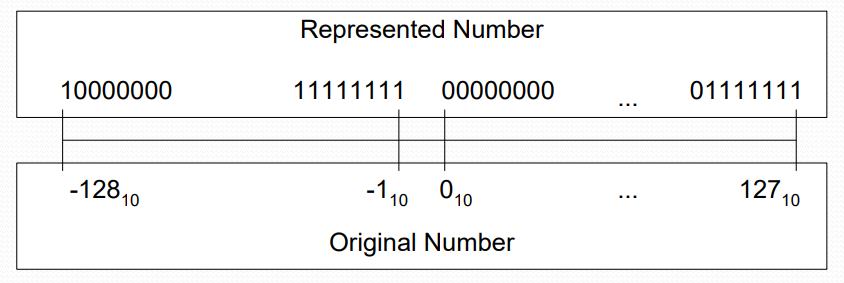
Consider the number -7, which in binary would be -111.  
To get the 1’s compliment of this, you subtract 00000111 from 11111111 = 11111000.  
Which could also be understood as you simply flipping the polarity as I mentioned.



**Definition:** 2’s compliment is what we are after doing while representing negative numbers in binary. All we have to do with 2’s complement is do what we done in 1’s compliment but also add 1 to the answer.

Example of this being used:

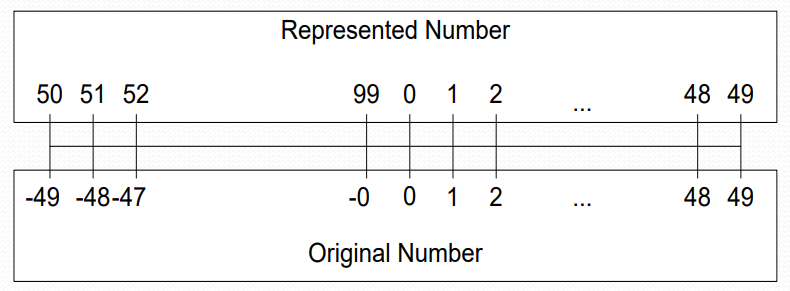
Consider the number -6. Which in binary would be -110.  
To get the 2’s compliment of this you get the largest possible binary number and add one to that, then subtract 110 from it.  
Which would give you an answer 11111010.



**Definition:** 9’s compliment is used by subtracting the absolute number you choose from 99. Shown is a picture describing how 9’s compliment is used.

Example of this being used:

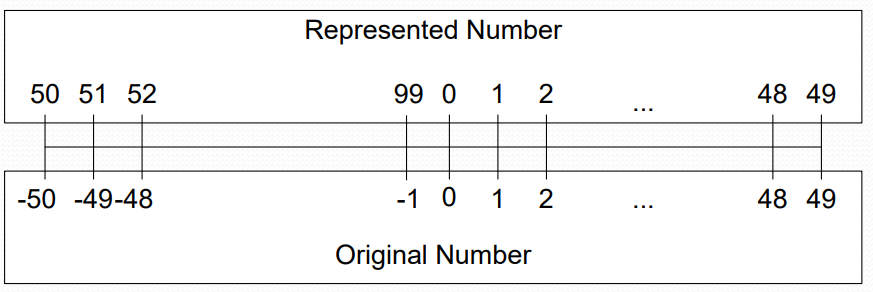
Consider the number -5.  
To get the 9’s compliment of this we subtract the 5 from 99 = 94.



**Definition:** In 10’s compliment you can represent the negative numbers by the largest number + 1 minus the absolute value of the negative number.  
Notice the unique representation of 0.

Example of this being used:

10’s complement = 9’s complement +1



**What is the Difference Between Overflow and Carry?**

**Definition:** Overflow occurs when you cannot properly represent the result as a signed value (you overflowed into the sign bit).

**Definition:** Carry occurs when you cannot properly represent the result as an unsigned value (no sign bit required).

|  |  |
| --- | --- |
| Example 1 | Example 2 |
| (+**18) + (+29)** | (-21**) + (+29)** |
| 010010 | 110101 |
| + 011101 | + 011101 |
| 101111 = (-47) | 1100010 = (50) |
| Overflow | Overflow |
| No Carry | No Carry |
| The Result is incorrect | The Result is incorrect |

**What is Exponential Notation?**

Exponential Notation is just a form of formatting numbers.

Example:

Consider the number 27658.

* In exponential notation we can write this at 27658 \* 100
* Or we can write it at 2.7658 \* 104
* Or 276.58 \* 102
* Or 2765800 \* 10-2

In the example, \* 10x, this just means, Multiple the mantissa by 10 to the power of x (x refers to the amount of time you multiple 10 by 10).

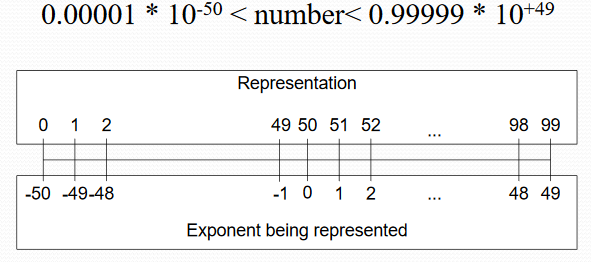
**What is Excess N Representation?**

Excess N representation shift s all values by N. It is a way of formatting floating numbers.

This method:

* Allows to store an exponential range of -50 to 49.
* Is simpler to use for exponents than the complementary form.

An illustration to explain representation:



**Converting a Decimal Fraction to a 32-Bit Floating Number**

To represent a decimal fraction as a 32-bit floating number we need to follow the following method:

5.1875 is our number we want to convert.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| First, we convert the whole number | |  |  | | --- | --- | | 5 = 101 (mmm) |  | | |
| Secondly, Then the fraction number | |  |  | | --- | --- | | .2 \* 2 = .4 | 0 | | .4 \* 2 = .8 | 0 | | .8 \* 2 = .6 | 1 | | .6 \* 2 = .2 | 1 | |  |  |   = 0011 (oooo) | |
| Our exponent is | 1 0 0 0 0 0 1 1 (eeeeeeee) | |
| We then connect all the parts together | Sign + eeeeeeee + mmm + oooo + as many zero as it takes to fill a total of 32 bits | |
| Floating Number | 01000000101001100000000000000000 |
|  |  | |

**How to add two Floating Point Numbers**

To add floating point numbers on a computer, everything needs to be perfect or else the computer will fail to compute the sum. Example: The exponents of the two floating point numbers must be the same; this is because the computer disregards the exponent while adding the numbers and just adds the numbers as if they were integers. The computer adds on the exponent again at the end of the calculation.

An example of this calculation:

Consider the decimals 3 and 5.2

Step 1: Follow the example shown above on how to convert a decimal fraction to a 32-bit floating number.

Step 2: After you have converted the decimals you should get:

01000000 01000000 00000000 00000000 for 3 (which can get broken down into the following parts, shown by the picture)



01000000 10100110 01100110 01100110 for 5.2 3 (which can get broken down into the following parts, shown by the picture)



We then add the mantissa’s

|  |
| --- |
| 0.110000000000000000000000 |
| +1.010011001100110011001100 |
| 10.000011001100110011001100 |
|  |

**How to multiply two Floating Point Numbers**

Consider the multiplication of these two numbers: 1.110 × 1010 × 1.100 × 10-5

Step 1: Add the exponents to find the new exponent: 10 + (-5) = 5

Step 2: Convert the exponent to binary: 101

Step 3: Get the sign of the number, convert to binary: 0

Step 2: Multiply the mantissa: 1.110 \* 1.100 = 10.101

Step 3: Combine the sign of the number, the exponent and the mantissa to form the final floating number: 01010000010101000000000000000000

**References**

**Excess N Representation – Slide 70 Notes**

**Decimal to 32 bit floating number -** [**https://www.youtube.com/watch?v=25Gr3ppaAUM**](https://www.youtube.com/watch?v=25Gr3ppaAUM)

**How to add two floating point numbers -** [**http://www.binaryconvert.com/convert\_float.html**](http://www.binaryconvert.com/convert_float.html)

How to multiply two floating point numbers - https://www.doc.ic.ac.uk/~eedwards/compsys/float/