# Number Representation 3 - Negative Numbers

## *Signed and Unsigned Numbers*

Numbers can be signed or unsigned. **Unsigned** numbers are always positive while **signed** numbers can be positive or negative.

Consider three bits. There are eight different combinations of ones and zeros (these are 000, 001, 010, 011, 100, 101, 110 and 111). These eight combinations can be made to represent the range of numbers from 0 to 7 or the numbers from -4 to 3. We say that the range "from 0 to 7" is unsigned and the range "from -4 to 3" is signed.

## *Sign and Magnitude*

How can negative numbers be represented? One method is called **sign and magnitude** where the leftmost bit is reserved to represent the sign. 1 indicates negative and 0 indicates positive. Let us take an example with 8 bits.

| sign | 64 | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | $77_{10}$ |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | $-77_{10}$ |

Now let us consider 4 bits. All the 4-bit values are represented in the following table.

| Binary Sign & Magnitude | Decimal |
|---|---|
| 1111 | -7 |
| 1110 | -6 |
| 1101 | -5 |
| 1100 | -4 |
| 1011 | -3 |
| 1010 | -2 |
| 1001 | -1 |
| 1000 | -0 |
| 0000 | +0 |
| 0001 | +1 |
| 0010 | +2 |
| 0011 | +3 |
| 0100 | +4 |
| 0101 | +5 |
| 0110 | +6 |
| 0111 | +7 |

Note that the value zero is repeated twice. The range of values is between -7 and 7. For one byte, the minimum value would be 11111111 ($-127_{10}$) and the maximum would be 01111111 ($127_{10}$).

Apart from the repeated zero, another problem with sign and magnitude representation is that it is not efficient for computation e.g. making an addition of two numbers etc. In fact, the computer uses another system to represent negative numbers. It is called two's complement representation.

### *One's Complement*

A **one's complement** representation of a binary number is simply obtained by changing every 0 to a 1 and every 1 to a 0. For example, the one's complement of 00101011 is 11010100

### *Two's Complement*

**Two's complement** of a number is obtained in two steps:

1. Find the one's complement
2. Add 1 to it.

In this representation the negative of a number is its two's complement.

Example: having one byte to write a number, express $-87_{10}$ in two's complement.

$87_{10} = 01010111_2$

The one's complement of $01010111_2$ is $10101000_2$

The two's complement is $10101000_2 + 1$ i.e. $10101001_2$

Therefore $-87_{10} = 10101001_2$ (in two's complement).

### *Exercise*

Assuming you have 8-bits to represent a number and you are working in two's complement, find the binary expression of:

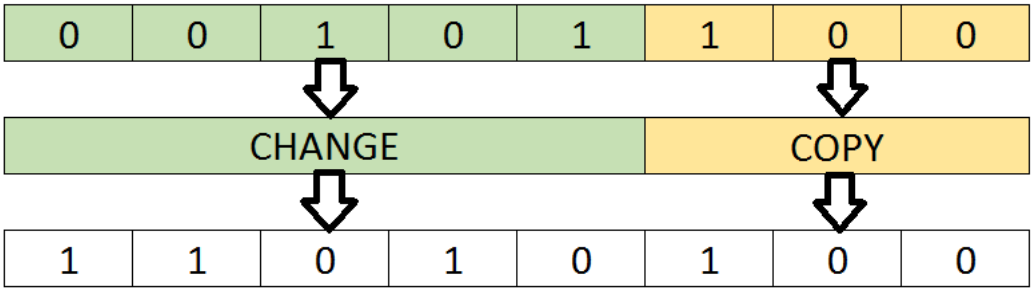
(a) -55
(b) -14
(c) 94

*Faster Computation of Two's Complement*

To work out the two's complement of a binary number do the following:

1. Start from the right-hand side of the number and copy all the zeros until you meet a 1.
2. Copy the 1.
3. From that point onwards change all bits.

Example: Work out the twos complement of 00101100.




Another example: Express $-104_{10}$ in two's complement (in one byte).

$104_{10} = 01101000_2$

$-104_{10} = 10011000_2$

*Subtraction using Two's Complement and Addition*

Suppose we want to calculate num1-num2. We follow these steps:

1. num1 is converted to binary. Call this f.
2. -num2 is converted to binary (two's complement). Call this t.
3. Add f and t but ignore the overflow bit.
4. Read the result.

Example: Perform 88 – 67 using two's complement (numbers are written in one byte).

$88_{10} = 01011000_2$

$67_{10} = 01000011_2$

$-67_{10} = 10111101_2$

$88_{10}$ $01011000_2$ +

$-67_{10}$ $10111101_2$

---

$X00010101_2$

The overflow 1 on the left is ignored.

*Exercise*

Perform the following calculations using two's complement arithmetic. All numbers are memorized in one byte.

(a) 99 – 56, (b) 85 – 106, (c) -25 – 30

*The Weights of a Two's Complement Number*

To know the weights of a two's complement number do this:

a) Give the normal weights as if the number was unsigned.
b) Negate the leftmost weight.

For example the following binary number has the weights as shown below:

| -128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

Therefore the value of $11010100_2$ = -128+64+16+4 = $-44_{10}$. Pay attention that this calculation can only be done if the number is in two's complement.

*Exercise*

Write the decimal value of the following numbers expressed in two's complement:

(a) 10010011
(b) 00110110

*Minimum, Maximum and Range*

Let us take a binary number of 4 bits. Let us find the maximum and minimum values that can be expressed if:

a) The number is unsigned
b) The number is signed and written in sign and magnitude
c) The number is signed and expressed in two's complement.

| Number language | Minimum (binary) | Minimum (decimal) | Maximum (binary) | Maximum (decimal) |
|---|---|---|---|---|
| **Unsigned** | 0000 | 0 | 1111 | 15 |
| **Signed, sign and magnitude** | 1111 | -7 | 0111 | 7 |
| **Signed, two's complement** | 1000 | -8 | 0111 | 7 |

Therefore for a four bit binary number the **ranges** of values are the following:

| | |
|---|---|
| Unsigned | From 0 to 15 |
| Sign and magnitude | From -7 to 7 |
| Two's complement | From -8 to 7 |

*Exercise*

Find the ranges seen above in the case when the number is 6 bits long.

### *Overflow and Underflow*

Consider the two's complement representation. Let us say we have one byte for the numbers. This means that the range is between -128 and 127. Now suppose you add the two numbers 99 and 48. This gives 147. However, 147 is out of range, so the calculation 99+48 would cause an **overflow**. Likewise if we perform -99-48 this would cause an **underflow**.