Number System and Representation

Binary Number System

We have already mentioned that computer can handle with two type of signals, therefore, to represent any information in computer, we have to take help of these two signals.

These two signals correspond to two levels of electrical signals, and symbolically we represent them as 0 and 1.

In our day to day activities for arithmetic, we use the *Decimal Number System*. The decimal number system is said to be of base, or radix 10, because it uses ten digits and the coefficients are multiplied by power of 10.

A decimal number such as 5273 represents a quantity equal to 5 thousands plus 2 hundreds, plus 7 tens, plus 3 units. The thousands, hundreds, etc. are powers of 10 implied by the position of the coefficients. To be more precise, 5273 should be written as:

$$5\times10^3 + 2\times10^2 + 7\times10^1 + 3\times10^0$$

However, the convention is to write only the coefficient and from their position deduce the necessary power of 10.

In decimal number system, we need 10 different symbols. But in computer we have provision to represent only two symbols. So directly we cannot use decimal number system in computer arithmetic.

For computer arithmetic we use **binary number system**. The binary number system uses two symbols to represent the number and these two symbols are 0 and 1.

The binary number system is said to be of base 2 or radix 2, because it uses two digits and the coefficients are multiplied by power of 2.

The binary number 110011 represents the quantity equal to:

$$1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \text{ (in decimal)}$$

We can use binary number system for computer arithmetic.

Representation of Unsigned Integers

Any integer can be stored in computer in binary form. As for example: The binary equivalent of integer 107 is 1101011, so 1101011 are stored to represent 107.

What is the size of Integer that can be stored in a Computer?
It depends on the word size of the Computer. If we are working with 8-bit computer, then we can use only 8 bits to represent the number. The eight bit computer means the storage organization for data is 8 bits.

In case of 8-bit numbers, the minimum number that can be stored in computer is 00000000 (0) and maximum number is 11111111 (255) (if we are working with natural numbers).

So, the domain of number is restricted by the storage capacity of the computer. Also it is related to number system; above range is for natural numbers.

In general, for *n-bit* number, the range for natural number is from $0 \text{ to } 2^n - 1$

Any arithmetic operation can be performed with the help of binary number system. Consider the following two examples, where decimal and binary additions are shown side by side.

01101000	**104**
00110001	**49**
----------	------
10011001	**153**

In the above example, the result is an 8-bit number, as it can be stored in the 8-bit computer, so we get the correct results.

10000001	**129**
10101010	**178**
----------	------
100101011	**307**

In the above example, the result is a 9-bit number, but we can store only 8 bits, and the most significant bit (MSB) cannot be stored.

The result of this addition will be stored as (00101011) which is 43 and it is not the desired result. Since we cannot store the complete result of an operation, and it is known as the overflow case.

The smallest unit of information is known as BIT (Binary digit).

The binary number 110011 consists of 6 bits and it represents:

$$1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

For an n-bit number the coefficient is - a_j multiplied by 2^j where ($0 \leq j < n$)

The coefficient $a(n-1)$ is multiplied by $2^{(n-1)}$ and it is known as most significant bit (MSB).

The coefficient a_0 is multiplied by 2^0 and it is known as least significant bit (LSB).

For our convenient, while writing in paper, we may take help of other number systems like octal and hexadecimal. It will reduce the burden of writing long strings of 0s and 1s.

Octal Number: The octal number system is said to be of base, or radix 8, because it uses 8 digits and the coefficients are multiplied by power of 8. Eight digits used in octal system are: 0, 1, 2, 3, 4, 5, 6 and 7.

Hexadecimal number: The hexadecimal number system is said to be of base, or radix 16, because it uses 16 symbols and the coefficients are multiplied by power of 16. Sixteen digits used in hexadecimal system are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.

Consider the following addition example:

Binary	Octal	Hexadecimal	Decimal
01101000	150	68	104
00111010	072	3A	58
----------	------	------	-----
10100010	242	A2	162

Signed Integer

We know that for *n-bit* number, the range for natural number is from $0 \; to \; 2^n - 1$.

For *n-bit*, we have all together 2^n different combination, and we use these different combination to represent 2^n numbers, which ranges from $0 \; to \; 2^n - 1$.

If we want to include the negative number, naturally, the range will decrease. Half of the combinations are used for positive number and other half is used for negative number.

For *n-bit* representation, the range is from $(-2^{n-1} - 1) \; to \; (+2^{n-1} - 1)$.

For example, if we consider 8-bit number, then range

for natural number is from 0 to 255; but
for signed integer the range is from -127 to +127.

Representation of signed integer

We know that for *n-bit* number, the range for natural number is from $0 \; to \; 2^n - 1$.

There are **three different schemes** to represent negative number:

- **Signed-Magnitude form.**
- **1's complement form.**
- **2's complement form.**

Signed magnitude form:

In signed-magnitude form, one particular bit is used to indicate the sign of the number, whether it is a positive number or a negative number. Other bits are used to represent the magnitude of the number.

For an n-bit number, one bit is used to indicate the signed information and remaining *(n-1) bits* are used to represent the magnitude. Therefore, the range is from $(-2^{n-1} - 1) \; to \; (+2^{n-1} - 1)$.

Generally, Most Significant Bit (MSB) is used to indicate the sign and it is termed as signed bit. 0 in signed bit indicates positive number and 1 in signed bit indicates negative number.

For example, 01011001 represents + 169 and
11011001 represents - 169

What is 00000000 and 10000000 in signed magnitude form?

The concept of complement

The concept of complements is used to represent signed number.

Consider a number system of *base-r* or *radix-r*. There are two types of complements,

- The radix complement or the *r's* complement.
- The diminished radix complement or the *(r - 1)'s* complement.

Diminished Radix Complement:

Given a number *N* in base *r* having *n* digits, the *(r - 1)'s* complement of *N* is defined as $(r^n - 1) - N$ For decimal numbers, $r = 10$ and $r - 1 = 9$, so the 9's complement of *N* is $(10^n - 1) - N$

e.g., 9's complement of 5642 is 9999 - 5642 = 4357.

Radix Complement:

The *r's* complement of an *n-digit* number in base *r* is defined as $(r^N - N)$ for $N \neq 0$ and 0 for $N = 0$. *r's* complement is obtained by adding 1 to the (r - 1)'s complement, since $(r^n - N) = [(r^n - 1) - N] + 1$

e.g., 10's complement of 5642 is 9's complement of 5642 + 1, i.e., 4357 + 1 = 4358
e.g., 2's complement of 1010 is 1's complement of 1010 + 1, i.e., 0101 + 1 = 0110.

Representation of Signed integer in 1's complement form:

Consider the eight bit number 01011100, 1's complements of this number is 10100011. If we perform the following addition:

```
  0 1 0 1 1 1 0 0
  1 0 1 0 0 0 1 1
-------------------
  1 1 1 1 1 1 1 1
```

If we add 1 to the number, the result is 100000000.

Since we are considering an eight bit number, so the 9th bit (MSB) of the result can not be stored. Therefore, the final result is 00000000.

Since the addition of two numbers is 0, so one can be treated as the negative of the other number. So, 1's complement can be used to represent negative number.

Representation of Signed integer in 2's complement form:

Consider the eight bit number 01011100, 2's complements of this number is 10100100. If we perform the following addition:

```
  0 1 0 1 1 1 0 0
  1 0 1 0 0 0 1 1
-------------------
1 0 0 0 0 0 0 0 0
```

Since we are considering an eight bit number, so the 9th bit (MSB) of the result can not be stored. Therefore, the final result is 00000000.

Since the addition of two number is 0, so one can be treated as the negative of the other number. So, 2's complement can be used to represent negative number.

Decimal	2's Complement	1's complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	-----	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
-8	1000	------	-------

Representation of Real Number

Binary representation of 41.6875 is 101001.1011

Therefore any real number can be converted to binary number system. There are **two schemes** to represent real number:

- Fixed-point representation
- Floating-point representation

Fixed-point representation:

Binary representation of 41.6875 is 101001.1011

To store this number, we have to store **two information**,

-- The part before decimal point and
-- The part after decimal point.

This is known as fixed-point representation where the position of decimal point is fixed and number of bits before and after decimal point are also predefined.

If we use 16 bits before decimal point and 7 bits after decimal point, in signed magnitude form, the range is

$$-2^{16} - 1 \; to + 2^{16} - 1$$ and the precision is $2^{(-7)}$

One bit is required for sign information, so the total size of the number is 24 bits

(1(sign) + 16(before decimal point) + 7(after decimal point)).

Floating-point representation:

In this representation, numbers are represented by a mantissa comprising the significant digits and an exponent part of Radix R. The format is:

$$mantissa * R^{exponent}$$

Numbers are often normalized, such that the decimal point is placed to the right of the **first non zero digit**.
For example, the decimal number,

5236 is equivalent to $.5236 * 10^4$

To store this number in floating point representation, we store 5236 in mantissa part and 4 in exponent part.

IEEE standard floating point format:

IEEE has proposed two standards for representing floating-point number:

- Single precision
- Double precision

Single Precision:

S	E	M

S: sign bit: 0 denoted + and 1 denotes -

E: 8-bit excess -27 exponent

M: 23-bit mantissa

Double Precision:

S	E	M

S: sign bit: 0 denoted + and 1 denotes -

E: 11-bit excess -1023 exponent

M: 52-bit mantissa

Representation of Character

Since we are working with 0's and 1's only, to represent character in computer we use strings of 0's and 1's only.

To represent character we are using some coding scheme, which is nothing but a mapping function.
Some of standard coding schemes are: **ASCII, EBCDIC, and UNICODE**.

ASCII: American Standard Code for Information Interchange.
It uses a 7-bit code. All together we have 128 combinations of 7 bits and we can represent 128 character.
As for example 65 = 1000001 represents character 'A'.

EBCDIC: Extended Binary Coded Decimal Interchange Code.
It uses 8-bit code and we can represent 256 character.

UNICODE: It is used to capture most of the languages of the world. It uses 16-bit

Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language. The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, JustSystem, Microsoft, Oracle, SAP, Sun, Sybase, Unisys and many others.