Software Systems

Integer Representation- Lecture01 2022/2023

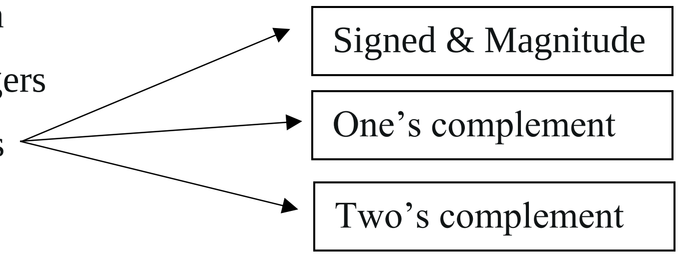
Dr. Azam E. Al-Rawachy
SOFTWARE DEPARTMENT | COLLEGE OF COMPUTER SCIENCE AND MATHEMATICS

1 Introduction

A computer number format is the internal representation of numeric values in digital device hardware and software, such as in programmable computers and calculators. Numerical values are stored as groupings of bits, such as bytes and words. The encoding between numerical values and bit patterns is chosen for convenience of the operation of the computer; the encoding used by the computer's instruction set generally requires conversion for external use, such as for printing and display. Different types of processors may have different internal representations of numerical values and different conventions are used for integer and real numbers.

2 Number Representation

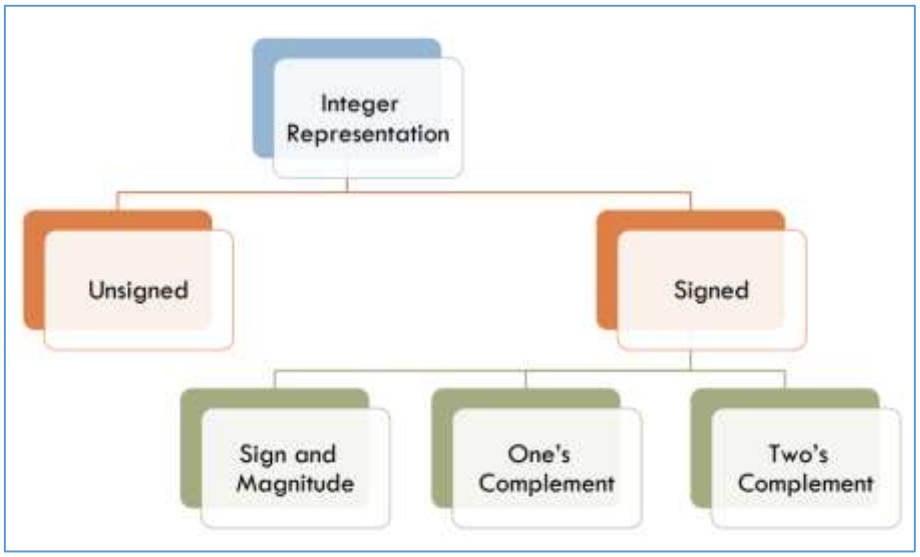
- Integer representation
  - Unsigned Integers
  - Signed integers



- Floating point representation (next week)

2.1 Integer Representation

Integer Number is a whole number without fractions, it can be positive or negative!



2.2 Binary Numbers Representation

Our computer can understand only (0, 1) language. The binary numbers are represented in both ways, i.e., signed and unsigned. Unsigned binary numbers do not have sign bit, whereas signed binary numbers uses signed bit as well or these can be distinguishable between positive and negative numbers. A signed binary is a specific data type of a signed variable.

The decimal number system has base **10** and binary number system has base **2**. While converting decimal number to binary number, the base of decimal number **10** will be changed into the base of binary number system **2**. All the decimal numbers hold their equivalent binary number.

2.2.1 Decimal to Binary Conversion

Ex: $(42)_{10} = (101010)_2$

2	42	
2	22	0
2	10	1
2	5	0
2	2	1
2	1	0
2	0	1

Try it for yourself: $(145)_{10} = (1001\ 0001)_2$

2.2.2 Binary to Decimal Conversion

EX: Convert **10100011** to decimal

$$= (1 \times 2^7) + (0 \times 26) + (1 \times 2^5) + (0 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) =$$

$$128 + 0 + 32 + 0 + 0 + 0 + 2 + 1 = \mathbf{163}$$

2.2.3 Binary to Hexadecimal Conversion

EX: Convert binary number **1101010** into hexadecimal number.

0110 1010

6 A

2.2.4 Hexadecimal to Binary Conversion

EX: Convert **A2B**$_{\mathbf{16}}$ to an equivalent binary number.

A2B$_{\mathbf{16}}$ ➔ 1010 0010 1011$_2$

A	2	B
1010	0010	1011

2.3 Addition & Subtraction

2.3.1 Hexadecimal Numbers

$$\begin{array}{r} 3\,4\,8\,2\,6\,3 \\ + \\ 7\,3\,C\,6\,9\,8 \\ \hline A\,8\,4\,8\,F\,B \end{array} \quad \begin{array}{r} 3\,9\,6\,F\,E\,2 \\ + \\ 5\,3\,8\,A\,A\,A \\ \hline 8\,C\,F\,A\,8\,C \end{array} \qquad \begin{array}{r} 9\,8\,E\,A\,7\,F\,\_ \\ - \\ 8\,9\,F\,F\,6\,3 \\ \hline 0\,E\,E\,B\,1\,C \end{array} \qquad \begin{array}{r} A\,6\,8\,9\,B\,E\,\_ \\ - \\ 8\,8\,F\,D\,E\,3 \\ \hline 1\,D\,8\,B\,D\,B \end{array}$$

F-3= C,

7-6=1

A-F= We have to borrow 16 from E ➔ A+16-F= 26-15= B

E-F-1= Borrow 16 from 8 ➔ E+16-F-1= 14+16 -15-1= E

8-9-1= Borrow 16 from 9 ➔ 8+16-9-1=24 -10= 14= E

9-8-1=0

2.3.2 Binary Numbers

Ex1: 11001 + 111 = **00000**

```
 1 1 1 1      Carry overs
 1 1 0 0 1
     1 1 1
1 0 0 0 0 0
```

$$2^{-1} = \frac{1}{2^1} = \frac{1}{2} = 0.5$$
$$2^{-2} = \frac{1}{2^2} = \frac{1}{4} = 0.25$$
$$2^{-3} = \frac{1}{2^3} = \frac{1}{8} = 0.125$$
$$2^{-4} = \frac{1}{2^4} = \frac{1}{16} = 0.0625$$
$$2^{-5} = \frac{1}{2^5} = \frac{1}{32} = 0.03125$$

Ex2: 11.25 + 3.375 ➔1011.01 + 11.011 = **14.625**

```
   1 1   1
  1011.01
+   11.011
----------
  1110.101
```

EX3: 3.5-2.75 = 0.75

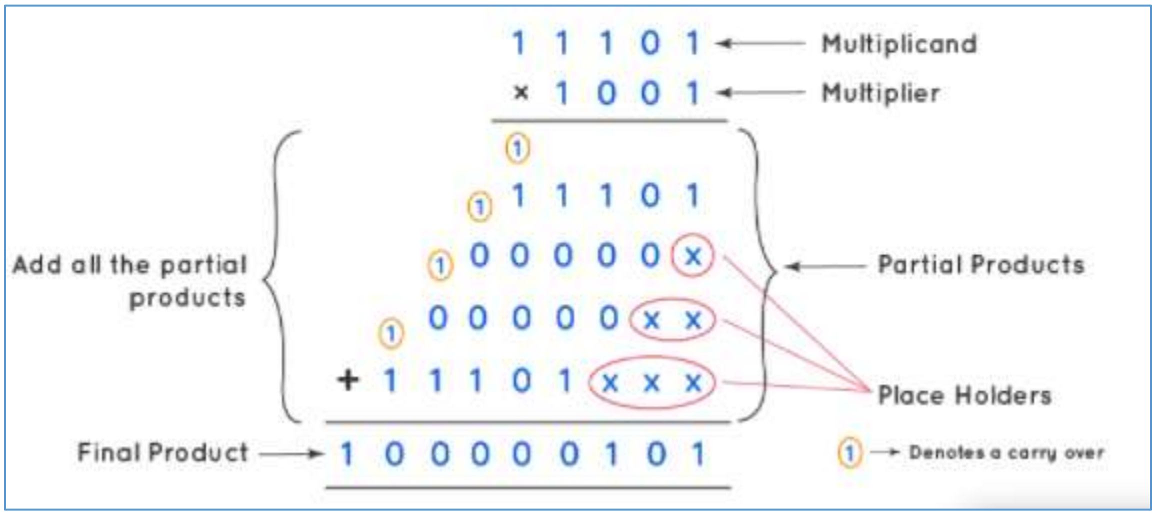
```
  0 0 11 .1 0 0 0
                  +
  1 1 0 1.0 1 0 0
-----------------
  0 0 0 0.1 1 0 0
```

$0010.\ 1100_2$ $(2.75)_{10}$ should be converted to 2'S complement as follow:

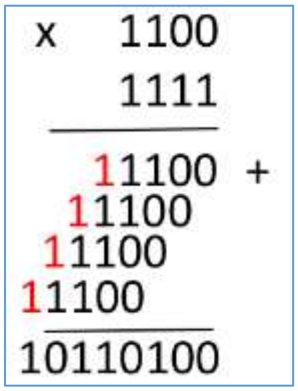
1101. 0100

2.3.3 Multiplication

EX4: 29 * 9 = 261 ➔ $(11101)_2 * (1001)_2$



EX5: 12 * 15= (180)



```
x     1100
      1111
    ______
     11100 +
    11100
   11100
  11100
  ________
  10110100
```

Try it by yourself!! 8*25 =(?)$_{10}$ =(?)$_2$

------------------ ----------------------

2.4 Unsigned representation

The maximum unsigned integer depends on the number of bits (N) allocated to represent the unsigned integer in a computer 0 ➔ (2^N-1)

No. of bits	Range
8	0 to 255
16	0 to 65535

2.5 Signed Numbers

Every Computer Programmer must understand signed and unsigned numbers and their significance. Positive numbers are represented as ***unsigned numbers***. So we don't need to use a ***Positive sign*** in front of them. However, when it comes to negative numbers we use ***Negative signs***. This shows that the number is negative and different from a positive unsigned value. This is why it's represented as ***signed numbers***.

2.5.1 Sign-and-Magnitude Representation:

For signed binary numbers the most significant bit (MSB) is used as the sign bit. If the sign bit is “0”, this means the number is positive in value. If the sign bit is “1”, then the number is negative in value. The remaining bits in the number are used to represent the magnitude of the binary number in the usual unsigned binary number format way. The table below shows the signed and unsigned representation of the 4-bit system.

Decimal Representation	Unsigned Representation	Signed Representation
7	0111	0111
6	0110	0110
5	0101	0101
4	0100	0100
3	0011	0011
2	0010	0010
1	0001	0001
0	0000	0000
-0	--------	1000
-1	--------	1001
-2	--------	1010
-3	--------	1011
-4	--------	1100
-5	--------	1101
-6	--------	1110
-7	--------	1111

However, using this sign-magnitude method can result in the possibility of two different bit patterns having the same binary value. For example, +0 and -0 would be 0000 and 1000 respectively as a signed 4-bit binary number. So we can see that using this method there can be two representations for zero, a positive zero (0000_2) and also a negative zero (1000_2) which can cause big complications for computers and digital systems. Besides, subtracting two number in this system produce inaccurate results.

$$\begin{array}{lr} 5 & 0101 \\ +(-5) & 1101 \\ \hline & 0010 \end{array} \qquad \begin{array}{lr} 6 & 0110 \\ +(-6) & 1110 \\ \hline & 0100 \end{array}$$

2.5.2 One's Complement Representation

A ones' complement system or ones' complement arithmetic is a system in which negative numbers are represented by the inverse of the binary representations of their corresponding positive numbers. In such a system, a number is negated (converted from positive to negative or vice versa) by computing its ones' complement.

Decimal Representation	Unsigned Representation	ones' complement
7	0111	0111
6	0110	0110
5	0101	0101
4	0100	0100
3	0011	0011
2	0010	0010
1	0001	0001
0	0000	0000
-0	--------	1111
-1	--------	1110
-2	--------	1101
-3	--------	1100
-4	--------	1011
-5	--------	1010
-6	--------	1001
-7	--------	1000

However, unlike two's complement, these numbers have not seen widespread use because of issues such as **the result is off by 1**, that negating zero results in a distinct negative zero-bit pattern, less simplicity with arithmetic borrowing, etc.

$$\begin{array}{lr} 5 & 0101 \\ +(-5) & 1010 \\ \hline -0 & 1111 \end{array} \quad \begin{array}{lr} 6 & 0110 \\ +(-6) & 1001 \\ \hline -0 & 1111 \end{array} \quad \begin{array}{lr} 5 & 0101 \\ +(-3) & 1100 \\ \hline 1 & 0001 \end{array} \quad \begin{array}{lr} 6 & 0110 \\ +(-2) & 1101 \\ \hline 3 & 0011 \end{array}$$

2.5.3 Two's Complement Representation

Two's complement is the most common method of representing signed integers on computers, the 2’s complement of a number can be found by find the 1’s complement (inverting every 1 and 0) and adding 1 to the result. The main advantage of two’s complement over the previous one’s complement is that there is no double-zero problem plus it is a lot easier to generate the two’s complement of a signed binary number. Therefore, arithmetic operations are relatively easier to perform when the numbers are represented in the two’s complement format.

Decimal Representation	Unsigned Representation	ones' complement	Two’s complement
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	--------
-1	--------	1110	1111
-2	--------	1101	1110
-3	--------	1100	1101
-4	--------	1011	1100
-5	--------	1010	1011
-6	--------	1001	1010
-7	--------	1000	1001

$$\begin{array}{lr} 5 & 0101 \\ +(-5) & 1010 \\ \hline -0 & 1111 \\ +1 & 0001 \\ \hline 0 & 0000 \end{array} \quad \begin{array}{lr} 6 & 0110 \\ +(-6) & 1001 \\ \hline -0 & 1111 \\ +1 & 0001 \\ \hline 0 & 0000 \end{array}$$

$$\begin{array}{lr} 4 & 0100 \\ +(-5) & 1011 \\ \hline -1 & 1111 \end{array} \quad \begin{array}{lr} 2 & 0010 \\ +(-5) & 1011 \\ \hline -3 & 1101 \end{array}$$

It is worth mentioning that the negative 2’s complement number can be found by looking at right-to-left and write down all digits through when you reach a 1. Then, invert the rest of the digits.

$$\begin{array}{c} 0110 \\ \underline{????} \\ 0000 \end{array} \quad \begin{array}{c} 0110 \\ \underline{??10} \\ 0000 \end{array} \quad \begin{array}{c} 0110 \\ \underline{1010} \\ 0000 \end{array}$$

$$\begin{array}{c} 0011 \\ \underline{????} \\ 0000 \end{array} \quad \begin{array}{c} 0011 \\ \underline{???1} \\ 0000 \end{array} \quad \begin{array}{c} 0011 \\ \underline{1101} \\ 0000 \end{array}$$

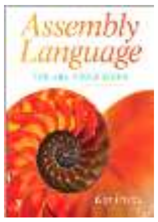
3 REFERENCES

"The 8088 and 8086 microprocessors Programming, interfacing, Software, Hardware and application "

By Walter A. Triebel and Avtar Singh

4th edition

2003



"Assembly Language for x86 Microprocessor"

By KIP R. IRVINE

7th Edition

2015