Lecture 1

Number Systems

Revised by WJ Han

Youpyo Hong, Dongguk University

Number Systems

- We are familiar with decimal number system.

  Ex) $19 = 1\text{x}10^1 + 9\text{x}10^0 = (19)_{10}$

- 1 and 9 are called coefficients and 10 is called the base.

- Bin, oct and hex means 2, 8 and 16, respectively.

- Binary systems have a base 2. Octal and hexadecimal systems have base 8 and 16, respectively.

  $(101)_2 = 1\text{x}2^2 + 0\text{x}2^1 + 1\text{x}2^0$

Notes on Number Systems

- **A coefficient cannot be larger than or equal to base.**

  Ex) $(201)_2$ is a wrong representation.

- **To represent a hexadecimal number, we need new symbols for 10, 11, 12, 13, 14, and 15. For them, we use A, B, C, D, E, and F.**

  Ex) $(26)_{10} = (1A)_{16}$

- **In this class, we will use binary number most frequently. So if a number does not have a base, that means the number is a binary number in this class.**

- **Later, we will use hexadecimal number frequently too.**

Conversion across Different Number Systems

- **The conversion from decimal number to a binary number ?**

  Ex) Convert 144_{10} to a binary number.

  1) Find the largest power of 2 < 144.
  $2^5 = 32$ (too small), $2^6 = 64$ (too small), $2^7 = 128$ (maybe), $2^8 = 256$ (too large). Yes! The answer is $1x2^7 + ?$.

  2) Find the largest power of 2 < $144 - 2^7 = 16$
  $2^4 = 16$. Yes! The answer is $1x2^7 + + 1x2^4 + ?$.

  3) Find the largest power of 2 < $144 - 2^7 - 1x2^4 = 0$. We are done.

  4) The final answer is $10010000 = 1x2^7 + 1x2^4$

- **This is ok but there is a better way.**

Systematic Conversion across Different Number Systems

- The conversion from decimal number to a binary number ?

Ex) Convert 144_{10} to a binary number.

2	144	... 0
2	72	... 0
2	36	... 0
2	18	... 0
2	9	... 1
2	4	... 0
2	2	... 0
	1	

2	144	... 0
2	72	... 0
2	36	... 0
2	18	... 0
2	9	... 1
2	4	... 0
2	2	... 0
	1	

The asnswer is 10010000

2	144	... 0 (2^0)
2	72	... 0 (2^1)
2	36	... 0
2	18	... 0
2	9	... 1
2	4	... 0
2	2	... 0
	1	($2^7 = 128$)

144/128 = 1 + ?

Conversion Examples

- **Convert 1011 to a decimal number.**
- **Convert 201_{10} to a binary number.**
- **Convert 110011 to a hexadecimal number. (This is very easy)**

Conversion of numbers smaller than 1

- **Ex) Convert 0.5_{10} to a binary number.**

$$0.5_{(10)} = 0.a_{-1}a_{-2}a_{-3}... = 2^{-1}*a_{-1} + 2^{-2}*a_{-2} + 2^{-3}*a_{-3} + ...$$

$$0.5_{(10)} = 2^{-1} * 1 = (0.1)_2$$

Interesting observation!

$0.5 * 2 = 1$, this 1 corresponds to a_{-1}

How to find a_{-i}'s systematically?

Systematic Conversion of numbers smaller than 1

$(0.6875)_{10} = (\qquad\qquad)_2$

$0.6875*2 = 1+0.3750 \quad a_{-1} = 1$

$0.3750*2 = 0+0.7500 \quad a_{-2} = 0$

$0.7500*2 = 1+0.5000 \quad a_{-3} = 1$

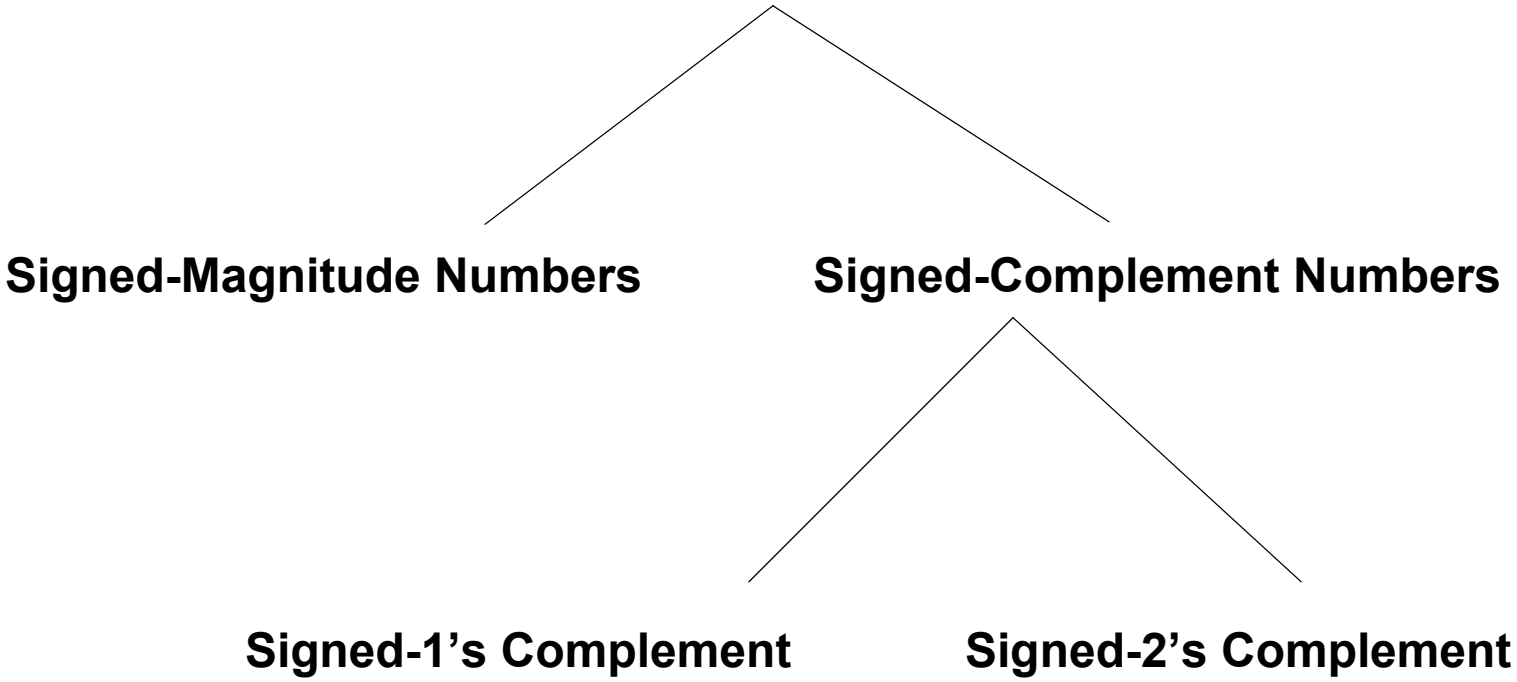
$0.5000*2 = 1+0.0000 \quad a_{-4} = 1$

$(0.6875)_{10} = (0.1011)_2$

Unsigned Numbers and Signed Numbers

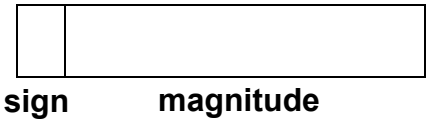
- **Unsigned numbers are all positive numbers.**
- **Unsigned numbers cannot express a negative number.**
- **Signed number can express both positive and negative numbers.**
- **Then, how to express negative numbers?**

Representation of Signed Binary Numbers



Signed-Magnitude Numbers

- A signed-magnitude number consists of one sign bit and magnitude bits. You have to determine the number of magnitude bits.



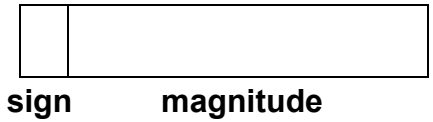
- Typically, MSB (Most Significant Bit) is used for the sign bit and if MSB is 1, that means the number is negative.

0001	(+1)
1001	(-1)

- It looks simple.
- But subtraction is not so simple to implement as a hardware.
- There is a better way.

Two's Complement

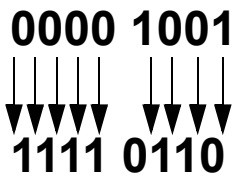
- In two's complement system, there is still a sign bit.



- For positive numbers, there are no difference between signed-magin-itude representation and two's complement representation.

- For negative numbers, take its magnitude and do the following. Let's assume that we want to represent -9.

1. Complement each bit

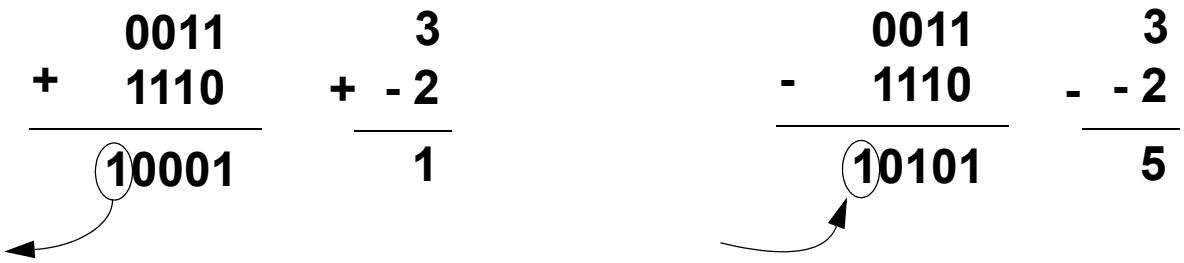


2. Add 1 to LSB

$$\begin{array}{rr} & 1111\ 0110 \\ + & \qquad\quad 1 \\ \hline & 1111\ 0111 \end{array}$$

Arithmetic Operations Using Two's Complement

- We can ignore the sign during operations just like they are unsigned numbers.



Subtractions Using Two's Complement

- Because A - B = A + (-B), we can use addition for subtraction instead.
- Instead of A - B, compute A + (-B). Here (-B) is the two's complement of B.

```
   0011          3
 - 1110        - -2
 ------        ----
  (1)0101        5
```

Since two's complement of (1110) is 0001+1 = 0010

```
   0011
 + 0010
 ------
   0101
```

Overflow

- For two *n* magnitude digit number's addition/subtraction, if the result occupies more than *n* digit, we say an overflow is occurred.

```
   0111        7
+  0110     +  6
------      ----
   1101       13
```

3 mag. digit

```
   1001       -7
+  1110     + -2
------      ----
   0111       -9
```

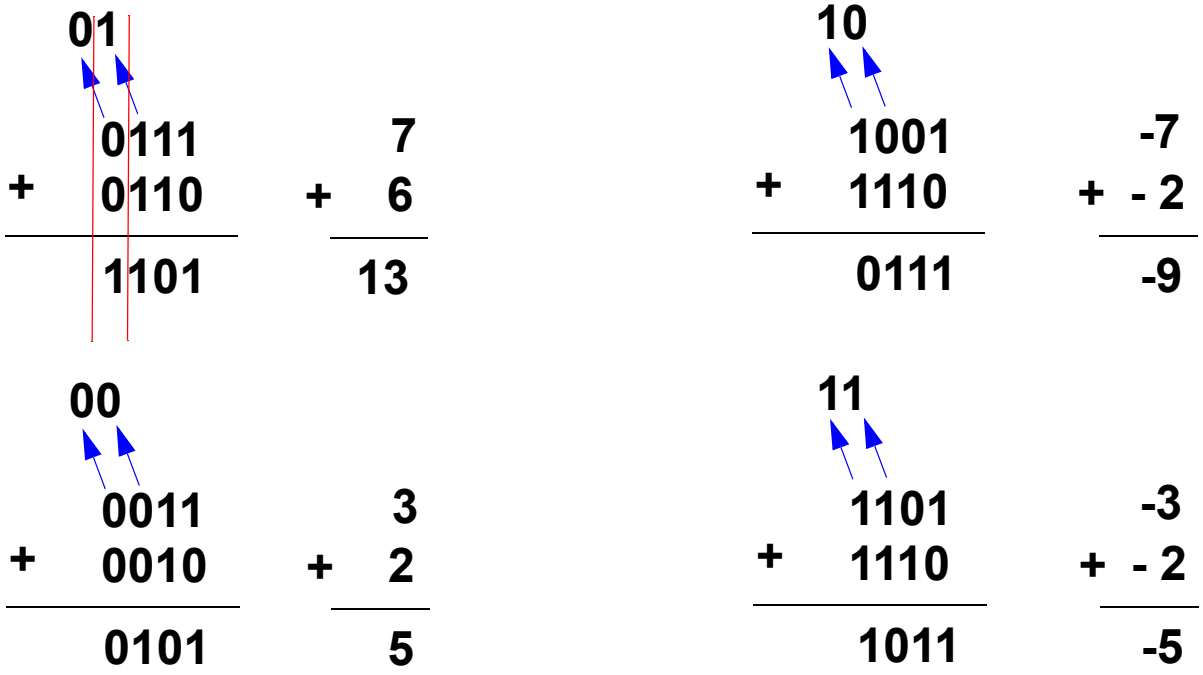
3 mag. digit

- How do you know if there was an overflow?

  - An addition of two numbers with different sign cannot result in an overflow.
  - A subtraction of two numbers with same sign cannot result in an overflow.

Overflow Detection

If the carry into the sign-bit and the carry out of the sign-bit are different, an overflow occurs.



Sign-Extension of a Two's Complement Number

- Suppose you have two numers of different bits.

```
      1101          -3
+  1100 0000    + - 64
-----------    -------
  1100 1101       -51 (X)
```

- Why? You changed the sign of small bit numbers.

```
      1101          0000 1101          13
+  1100 0000   =  + 1100 0000   =  + - 64
-----------       -----------     -------
  1100 1101       1100 1101          -51
```

- How to fix it? You have to move the sign bit to new position and fill the empty bits with the sign bit.