Arithmetic
Operations on Digital and Binary Numbers (3)

Lecture 04
Book Chapter(s): 1

COE211-Digital Logic Design

الفصل الدراسي الأول 1442-Fall2020

جامعة طيبة فرع ينبع - كلية علوم وهندسة الحاسبات – شطر الطالبات

د. فاطمة الحربي

1s Complements – Addition and Subtraction (1)

- **Algorithm for addition, A + B:**
  1. Perform binary addition on the two numbers
  2. If there is a carry out of the MSB, add 1 to the result
  3. Check for overflow:
     - Overflow occurs if result is opposite sign of A and B

- **Algorithm for subtraction, A – B:**
  1. A – B = A + (–B)
  2. Take 1s complement of B by inverting all the bits
  3. Add the 1s complement of B to A

1s Complements – Addition and Subtraction (2)

- Examples: 4-bit binary system

```
   +3        0011
+  +4     +  0100
----      -------
   +7        0111
----      -------
```

```
   +5        0101
+  -5     +  1010
----      -------
   -0        1111
----      -------
```

```
   -2        1101
+  -5     +  1010
----      -------
   -7       10111
----      +     1
          -------
           1000
```

```
   -3        1100
+  -7     +  1000
----      -------
  -10       10100
----      +     1
          -------
            0101
```

2s Complements – Addition and Subtraction (1)

- **Algorithm for addition, A + B:**
  1. Perform binary addition on the two numbers
  2. Ignore the carry out of the MSB (most significant bit)
  3. Check for overflow:
     - Overflow occurs if the 'carry in' and 'carry out' of the MSB are different, or if result is opposite sign of A and B

- **Algorithm for subtraction, A – B:**
  1. A – B = A + (–B)
  2. Take 2s complement of B by inverting all the bits and adding 1
  3. Add the 2s complement of B to A

2s Complements – Addition and Subtraction (2)

- Examples: 4-bit binary system

```
   +3        0011
+  +4     +  0100
-------   --------
   +7        0111
-------   --------
```

```
   -2        1110
+  -6     +  1010
-------   --------
   -8       11000
-------   --------
```

```
   +6        0110
+  -3     +  1101
-------   --------
   +3       10011
-------   --------
```

```
   +4        0100
+  -7     +  1001
-------   --------
   -3        1101
-------   --------
```

Overflow (1)

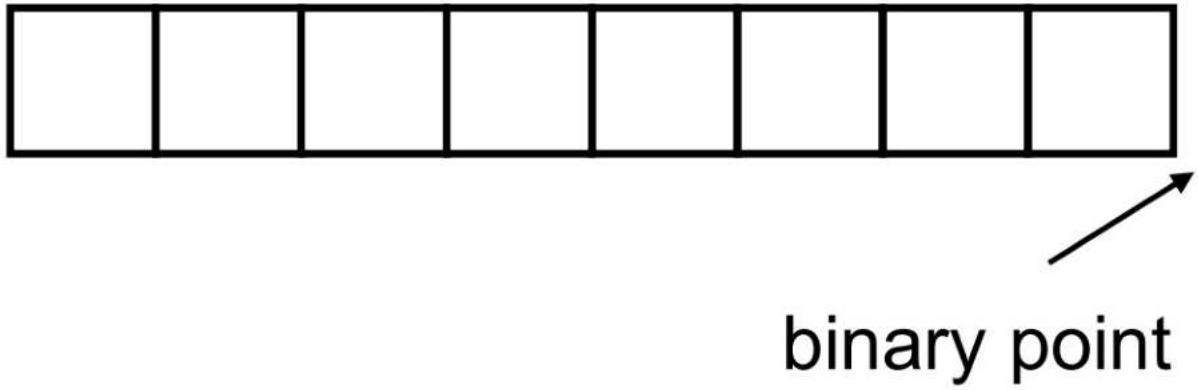
- Signed binary numbers are of a fixed range
- If the result of addition/subtraction goes beyond this range, overflow occurs
- Two conditions under which overflow can occur are:
  - *positive add positive* gives negative
  - *negative add negative* gives positive

Overflow (2)

- Examples: 4-bit numbers (in 2s complement)
- Range : $(1000)_{2s}$ to $(0111)_{2s}$ or (-8_{10} to 7_{10})
  - $(0101)_{2s} + (0110)_{2s} = (1011)_{2s}$
    $(5)_{10} + (6)_{10} = -(5)_{10}$?! (overflow!)
  - $(1001)_{2s} + (1101)_{2s} = (10110)_{2s}$ discard end-carry
    $= (0110)_{2s}$
    $(-7)_{10} + (-3)_{10} = (6)_{10}$?! (overflow!)

Fixed Point Numbers (1)

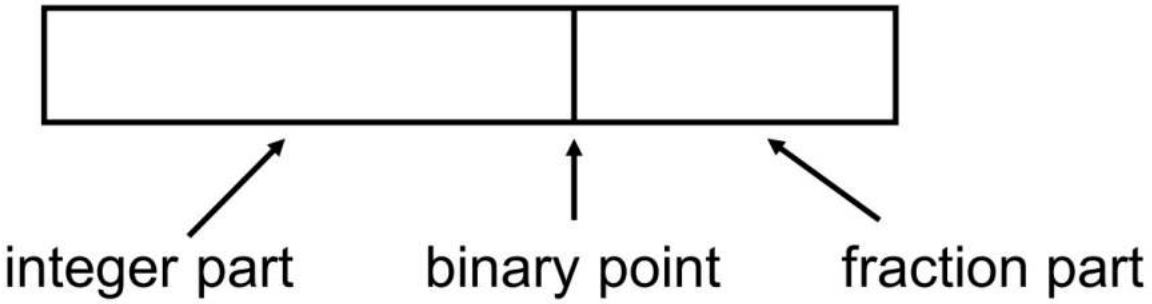
- The signed and unsigned numbers representation given are fixed point numbers
- The **binary point** is assumed to be at a fixed location, say, at the end of the number:



- Can represent all integers between –128 to 127 (for 8 bits).

Fixed Point Numbers (2)

- In general, other locations for binary points possible



- Examples: If two fractional bits are used, we can represent:

$$(001010.11)_{2s} = (10.75)_{10}$$
$$(111110.11)_{2s} = -(000001.01)_{2}$$
$$= -(1.25)_{10}$$

Floating Point Numbers (1)

- Fixed point numbers have limited range
- To represent very large or very small numbers, we use floating point numbers (cf. scientific numbers).
- Examples:

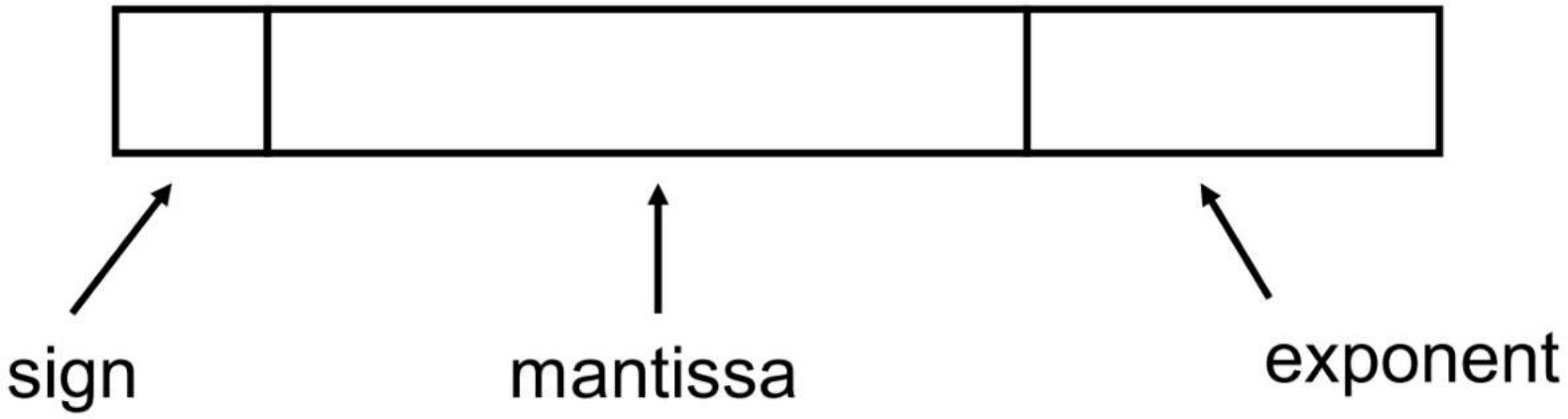
0.23×10^{23} (very large positive number)

0.5×10^{-32} (very small positive number)

-0.1239×10^{-18} (very small negative number)

Floating Point Numbers (2)

- Floating point numbers have three parts:
  sign, mantissa, and **exponent**
- The base (radix) is assumed (usually base 2)
- The sign is a single bit (0 for positive number, 1 for negative).



Floating Point Numbers (3)

- Mantissa is usually in normalised form:
  - (base 10) 23 x 10^{21} normalised to 0.23 x 10^{23}
  - (base 10) -0.0017 x 10^{21} normalised to -0.17 x 10^{19}
  - (base 2) 0.01101 x 2^{3} normalised to 0.1101 x 2^{2}
- **Normalised form:** The fraction portion cannot begin with zero
- More bits in exponent gives larger range
- More bits for mantissa gives better precision

Floating Point Numbers (4)

- Exponent is usually expressed in complement or excess form
- Example:
  - Express $-(6.5)_{10}$ in base-2 normalised form:
    $-(6.5)_{10} = -(110.101)_2 = -0.110101 \times 2^3$
- Assuming that the floating-point representation contains 1-bit sign, 5-bit normalised mantissa, and 4-bit exponent.
  - The above example will be represented as:

1	110101	011

Arithmetic with Floating Point Numbers - Multiplication

- Arithmetic is more difficult for floating point numbers
- Steps for multiplication:
  1. Multiply the mantissa
  2. Add-up the exponents
  3. Normalise

1. Example:

$$(0.12 \times 10^{2})_{10} \times (0.2 \times 10^{30})_{10}$$

$$= (0.12 \times 0.2)_{10} \times 10^{2+30}$$

$$= (0.024)_{10} \times 10^{32} \quad \text{(normalise)}$$

$$= (0.24 \times 10^{31})_{10}$$

Arithmetic with Floating Point Numbers - Addition

- Steps for addition:
  1. Equalise the exponents
  2. Add-up the mantissa
  3. Normalise
- Example:

$$(0.12 \times 10^3)_{10} + (0.2 \times 10^2)_{10}$$
$$= (0.12 \times 10^3)_{10} + (0.02 \times 10^3)_{10} \text{ (equalise exponents)}$$
$$= (0.12 + 0.02)_{10} \times 10^3 \text{ (add mantissa)}$$
$$= (0.14 \times 10^3)_{10}$$

Can you figure out how to perform SUBTRACTION and DIVISION for (binary/decimal) floating-point numbers?