BACS1024 INTRODUCTION TO COMPUTER SYSTEMS

Chapter 3: Floating Point Representation

0. Overview

1. Signed & Unsigned Number Representation
2. Floating-point Number Representation & Standard
3. Bitwise Logical Operations

1. Signed & unsigned Num Representation

1. Signed & Unsigned Number Representation

- **Integer representation**
  - **Unsigned integer**
    - hold a positive value, and no negative value
    - uses the most significant bit (MSB) as a part of the value
    - E.g.: **Unsigned** integer: $1111\ 1111_2 = +255_{10}$
  - **Signed integer**
    - hold both positive and negative numbers
    - uses the most significant bit (MSB) to identify if the number is positive or negative. **0** indicates a **positive** while **1** indicates a **negative**.
    - E.g.: **Signed** integer: $0111\ 1111_2 = +127_{10}$

      $1111\ 1111_2 = -127_{10}$

      However, this made calculation difficult.

1. Signed & Unsigned Number Representation

- **In Signed-and-magnitude representation**

$$\boxed{\mathbf{1}\quad \mathbf{001\ 0011_2}}$$

sign bit (MSB) **Magnitude bits**

- This made calculation difficult.
  - ❑ E.g.: In **signed** integer: $0111\ 1111_2 = 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$

    $= +127_{10}$

    $1111\ 1111_2 = -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$

    $= -1_{10}$

    $\neq -127_{10}$

- Solution: **Two's Complement**

1. Signed & Unsigned Number Representation

- **Two's Complement**
  - 2 steps: 1.) One's complement

    Perform inversion. i.e. change 1's to 0's and 0's to 1's

    2.) Adding 1 to the result of One's complement
  - As the computer stores data in byte basis, therefore **8 bit system** is used.
  - E.g.:

10_{10} = $0000\ 1010_2$

One's complement = $1111\ 0101_2$

Add 1_2 = $\underline{\qquad\qquad 1_2 +}$

Two's complement = $1111\ 0110_2$

= $-(2^7) + 2^6 + 2^5 + 2^4 + 2^2 + 2^1$

= -10_{10}

1. Signed & Unsigned Number Representation

- **Two's Complement**
  - ❑ Uses: To represent negative value & perform subtraction.
  - ❑ E.g.: $30_{10} - 10_{10}$
    $= 30_{10} + (-10_{10})$

$30_{10} = 0001\ 1110_2$

$10_{10} = 0000\ 1010_2$

One's complement = $1111\ 0101_2$

add 1_2 = $\underline{\qquad\qquad 1_2\ +}$

Two's complement = $1111\ 0110_2$

= -10_{10}

Continued:

Decimal	**Binary**
30_{10}	$0001\ 1110_2$
$+ (-10)_{10}$	$+1111\ 0110_2$
20_{10}	$(1)0001\ 0100_2$
	$= 20_{10}$

1. Signed & Unsigned Number Representation

- **Overflow**
  - Occurs when the result of an arithmetic operation does not fit into the fixed number of bits available for the result.
  - Occur only when both operands have the same sign.
  - Detected by the fact that the sign of the result is opposite of both operands.
  - Stored in overflow flag (OF)
- **Carry**
  - Occurs when the result of an arithmetic operation exceeds the fixed number of bits allocated, without regard to the sign.
  - The carry bit is ignored in single precision 2's complement addition and subtraction.
  - Detected when extra '1' bit is generated.
  - Stored in carry flag (CF)

1. Signed & Unsigned Number Representation

- **Overflow**
  - E.g.:

$$\begin{array}{rr} 64_{10} & 0100\ 0000_2 \\ +\ 65_{10} & +\ 0100\ 0001_2 \\ \hline 129_{10} & 1000\ 0001_2 \end{array} = -127_{10}$$

In **8 bit system**, data ranges from:

-128 to +127.

$64_{10} + 65_{10} = 129_{10}$ which is **> 127**

i.e. 129 does not fit into the number of bits available.

Therefore, **OVERFLOW** occur.

Since $129_{10} \neq -127_{10}$, this is **invalid**.

- **Carry**
  - E.g.:

$$\begin{array}{rr} 106_{10} & 0110\ 1010_2 \\ +\ (-\ 2)_{10} & +\ 1111\ 1110_2 \\ \hline 104_{10} & (1)0110\ 1000_2 \end{array} = 104_{10}$$

In **8 bit system**, extra 1 bit is generated. Carry occur.

Carry ignore

2. Floating-point Number Representation & Standard

2. Floating-point Num Representation & Standard

- **Exponential notation**
  - For decimal numbers, floating point is represented in **scientific notation**.
  - E.g.:

    12345

    $= 12345 \times \mathbf{10^{0}}$

    $= 0.12345 \times \mathbf{10^{5}}$

    $= 0.0012345 \times \mathbf{10^{7}}$

    $= 123450000 \times \mathbf{10^{-4}}$
  - Allows a range of overly large & very small numbers to be represented with a few digits.

2. Floating-point Num Representation & Standard

- **Exponential notation**
  - ❑ A number is represented by the combination of 6 specifications.
  - ❑ E.g.: To represent 12345_{10}

$$\pm 0.12345 \times 10^{\pm 5}$$

Sign | **location of decimal point** | **Magnitude / Mantissa** | **Base** | **sign of exponent** | **Exponent**

2. Floating-point Num Representation & Standard

- **Floating Point Representation**
  - ❑ Floating point numbers will be stored and manipulated in the computer using standard predefined format, usually in 8 bits basis.
  - ❑ There are 2 key format applied: **SEEMMMMM** & **IEEE754** notation
  - ❑ The base of exponent and location of the binary point are standardize as part of the format. Therefore, they are not required to be stored at all.

2. Floating-point Num Representation & Standard

- **Floating Point Representation - SEEMMMMM**
  - In **SEEMMMMM** format,
    - 1 digit for sign
    - 2 digits for exponent
    - 5 digits for mantissa, the decimal point location is assumed to be located at the beginning of mantissa.
  - **Excess-N** notation
    - **N** is the chosen middle value
    - E.g.: Excess-50 allow a magnitude ranges as follow:

$$0.00001 \times \mathbf{10^{-50}} < \text{Number} < 0.99999 \times \mathbf{10^{49}}$$

Excess	-48	-49	50	51	52
Exponent	-2	-1	0	1	2

2. Floating-point Num Representation & Standard

- **Floating Point Representation - SEEMMMMM**

❑ **Conversion**:

Excess	-48	-49	50	51	52
Exponent	-2	-1	0	1	2

❖ The base is 10.

❖ The implied decimal point is at the beginning of the mantissa.

❖ Assume that:

✔ Excess-50 is applied.

✔ A 0 represents positive and 5 represents negative.

05324567 = 0.24567 x $\mathbf{10^3}$ = 245.67

54810000 = - 0.10000 x $\mathbf{10^{-2}}$ = - 0.00100000

55555555 = - 0.5555 x $\mathbf{10^5}$ = - 55555

2. Floating-point Num Representation & Standard

- **Floating Point Representation - SEEMMMMM**
  - ❑ **Normalization**
    - ❖ Shift number left by increasing the exponent until leading zeros are eliminated.
  - ❑ **Steps**: (Convert decimal number into SEEMMMMM format)
    1. Provide number with exponent (0 if not yet specified)
    2. Increase / decrease exponent to shift decimal point to proper position.
    3. Decrease exponent to eliminate leading zeros on mantissa.
    4. Correct precision by adding 0's or discarding / rounding least significant digits

2. Floating-point Num Representation & Standard

- **Floating Point Representation - SEEMMMMM**
  - ❑ **Normalization and formatting**
  - ❑ E.g.:
    - ❖ Given 246.8035, normalize it and represent it in SEEMMMMM format.
  - ❑ Steps:
    1. Add exponent : 246.8035 x $\mathbf{10^0}$
    2. Position decimal point : 0.2468035 x $\mathbf{10^3}$
    3. Already normalized, no adjustment is required.
    4. Trim mantissa to 5 digits : 0.24680 x $\mathbf{10^3}$
    5. Convert the number : 05324680

❖ **Assume** that:
- ✔ Excess-50 is applied.
- ✔ A 0 represents positive and 5 represents negative.

2. Floating-point Num Representation & Standard

- **Floating Point Representation - SEEMMMMM**
  - **Normalization and formatting**
  - E.g.:
    - Given -1255 x $\mathbf{10^{-3}}$, normalize it and represent it in SEEMMMMM format.
  - Steps:
    1. The number is already in exponential form.
    2. Position decimal point : - 0.1255 x $\mathbf{10^{1}}$
    3. Already normalized, no adjustment is required.
    4. Trim mantissa to 5 digits : -0.12550 x $\mathbf{10^{1}}$
    5. Convert the number : 55112550

- **Assume** that:
  - ✔ Excess-50 is applied.
  - ✔ A 0 represents positive and 5 represents negative.

2. Floating-point Num Representation & Standard

- **Floating Point Representation - SEEMMMMM**
  - ❑ **Addition and subtraction**
    - ❖ Exponent and mantissa treated separately.
    - ❖ Exponents of numbers must agree.
      - ✔ Align decimal points
      - ✔ Least significant digits may be lost
    - ❖ Overflow of the most significant digit may occurs.
    - ❖ Number must be shifted right and the exponent incremented to accommodate overflow.

2. Floating-point Num Representation & Standard

- **Floating Point Representation - SEEMMMMM**
  - **Addition and subtraction**
    - E.g.: Add the 2 floating-point numbers.

05199520
+ 04967850

❖ **Assume** that:
- ✔ Excess-50 is applied.
- ✔ A 0 represents positive and 5 represents negative.

Align exponents	= 05199520
By adding 2 zeros in front to mantissa	= 0510067850 +
Add mantissa, (1) indicates a carry	= (1)0019850
Carry requires right shift of exponent	= 0 52 10019(850)
Round	= 0 52 10020

2. Floating-point Num Representation & Standard

- **Floating Point Representation - SEEMMMMM**
  - **Addition and subtraction**
    - E.g.: Check result.

❖ **Assume** that:
- ✔ Excess-50 is applied.
- ✔ A 0 represents positive and 5 represents negative.

$$\begin{array}{r} 05199520 \\ \underline{+\ 04967850} \end{array}$$

05199520 = 0.99520 x $\mathbf{10^{1}}$ = 9.520

04967850 = 0.67850 x $\mathbf{10^{-1}}$ = 0.067850 + 10.019850

In sign-magnitude form = 0.1001985 x $\mathbf{10^{2}}$

2. Floating-point Num Representation & Standard

- **Floating Point Representation - SEEMMMMM**
  - ❑ **Multiply and divide**
    - ❖ Mantissa: Multiplied or divided
    - ❖ Exponent: Added or subtracted and adjusted excess value since added trice
    - ❖ E.g.:
      - ✔ Assume that two number with exponent 3, each representing 53.
      - ✔ Adding the two exponent: 53 + 53 = 106
      - ✔ Since 50 (excess-50) is added twice, subtract: 106 – 50 = 56
    - ❖ Normalization necessary to:
      - ✔ Restore location of decimal point
      - ✔ Maintain precision of the result.

❖ **Assume** that:
- ✔ Excess-50 is applied.
- ✔ A 0 represents positive and 5 represents negative.

2. Floating-point Num Representation & Standard

- **Floating Point Representation - SEEMMMMM**
  - ❑ **Multiply and divide**
    - ❖ E.g.: multiply two floating pint numbers

05220000

x 04712500

Add exponent, subtract offset = 75 + 47 – 50 = 49

Multiply mantissa = 0.20000 x 0.12500

= 0.025000000

Normalized the result = 04825000

❖ **Assume** that:

- ✔ Excess-50 is applied.
- ✔ A 0 represents positive and 5 represents negative.

2. Floating-point Num Representation & Standard

- **Floating Point Representation - SEEMMMMM**
  - **Multiply and divide**
    - E.g.: Check result

$05220000 = 0.20000 \times \mathbf{10^{2}} = 20$

$04712500 = 0.12500 \times \mathbf{10^{-3}} = 0.000125$

Multiply $= 20 \times 0.000125$

$= 0.025000000 \times \mathbf{10^{-1}}$

Normalizing and rounding $= 0.25000 \times \mathbf{10^{-2}}$

- **Assume** that:
  - Excess-50 is applied.
  - A 0 represents positive and 5 represents negative.

2. Floating-point Num Representation & Standard

- **Floating Point Representation – IEEE754**
  - ❑ **Typical floating point format in computer**
    - ❖ Consists of 32 bits, divided into:
      - ✔ 1 bit of sign
      - ✔ 8 bits of exponent, excess-127 notation, base 2
      - ✔ 23 bits of mantissa

bit 0	1 … 8	9 … 31
S	E … E	M … M
Sign	Exponent	Mantissa

2. Floating-point Num Representation & Standard

- **Floating Point Representation – IEEE754**
  - ❑ Normalized numbers must always start with a 1, the leading bit is not stored, but is instead implied.
  - ❑ This bit is located to the left of the implied binary position. So, numbers are normalized to the form 1.MMMMMM…

Precision	Single (32-bit)	Double (64-bit)
Sign	1 bit	1 bit
Exponent	8 bits	11 bits
Implied base	2	2
Range	2^{-126} to 2^{127}	2^{-1022} to 2^{1023}
mantissa	23	52
Notation	Excess-127	Excess-1023

2. Floating-point Num Representation & Standard

- **Floating Point Representation – IEEE754**
  - ❑ E.g.: Convert 253.75_{10} to binary floating point form.

$253.\ 75_{10}$

$1111\ 1101.\ 11_2$

Therefore, $253.11_{10} = 1111\ 1101.11_2$

$= 1.111\ 110111 \times 2^{+7}$

0	10000110	11111011100..

Sign	Exponent	Mantissa
1	8	23 bits

2. Floating-point Num Representation & Standard

- **Floating Point Representation - Conversion**
  - ❑ E.g.: Convert of fraction number
  - ❑ $16.5_{10} \rightarrow$ base 2

2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
4	2	1	1/2	1/4	1/8

- ❑ $16.5_{10} = 16_{10} + 0.5_{10}$

  $= 10000_2 + 0.1_2$

  $= 10000.1_2$

2. Floating-point Num Representation & Standard

- **Floating Point Representation - Conversion**
  - E.g.: Convert of fraction number
  - $16.5_{10} \rightarrow$ base 2

2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}
4	2	1	1/2	1/4	1/8

  - $16.5_{10} = 16_{10} + 0.5_{10}$
    $= 10000_2 + 0.1_2$
    $= 10000.1_2$

OR,
0.5 **x** 2 **=** 1.00

(Reading the integer only, in top-down direction)

2. Floating-point Num Representation & Standard

X 2 because it need to change to base 2.

Since it is multiplied using fractional numbers, therefore it must add **0.** in front of the answer. 0.110101_2

- **Floating Point Representation - Conversion**
  - ❑ E.g.: Convert of fraction number
  - ❑ 0.828125_{10} → base 2

0.828125 x 2 = **1**.**656250**

0.**656250** x 2 = **1**.**312500**

0.**312500** x 2 = **0**.625000

0.625000 x 2 = **1**.250000

0.250000 x 2 = **0**.500000

0.500000 x 2 = **1**.000000

Therefore, $0.828125_{10} = 0.110101_2$

2. Floating-point Num Representation & Standard

- **Floating Point Representation - Conversion**
  - E.g.: Convert of fraction number
  - 0.828125_{10} → base 2
  - To validate it:

	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
Decimal	0	1/2	1/4	1/8	1/16	1/32	1/64
	0	0.5	0.25	0.125	0.0625	0.03125	0.015625
Binary	0	1	1	0	1	0	1

Therefore, $0.828125_{10} = 0.110101_2$

2. Floating-point Num Representation & Standard

- **Floating Point Representation - Conversion**
  - ❑ E.g.: Convert of fraction number
  - ❑ 11.110011_2 → base 10

Binary	1	1	.	1	1	0	0	1	1
Decimal	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
	2	1	.	1/2	1/4	1/8	1/16	1/32	1/64
	2	1	.	0.5	0.25	0.125	0.0625	0.03125	0.015625
	3.796875_{10}								

Therefore, $11.110011_2 = 3.796875_{10}$

2. Floating-point Num Representation & Standard

- **Floating Point Representation - Conversion**
  - ❑ E.g.: Convert of fraction number
  - ❑ 10011.10111_2 → base $_{16}$

10011.10111_2

$\underline{\mathbf{000}}1\ \ 0011\ .\ 1001\ \ 1\underline{\mathbf{000}}_{\mathbf{2}}$

$1\quad 3.\quad 9\quad 8_{\mathbf{16}}$

Additional Zero added

2. Floating-point Num Representation & Standard

- **Floating Point Representation - Conversion**
  - E.g.: Convert of fraction number
  - $39.B8_{16} \rightarrow$ base 10

$= (3 \times 16^{1}) + (9 \times 16^{0}) + (.) + (B \times 16^{-1}) + (8 \times 16^{-2})$

$= 48_{10} + 9_{10} + (.)\ 0.6875_{10} + 0.03125_{10}$

$= 57.71875_{10}$

2. Floating-point Num Representation & Standard

- **Floating Point Representation - Conversion**
  - ❑ E.g.: Convert of fraction number
  - ❑ $4F5.09_{16} \rightarrow$ base $_2$

$4 \quad F \quad 5 . 0 \quad 9_{16}$

$\underline{\mathbf{0}}100\ 1111\ 0101\ .\ 0000\ 1001_{2}$

Additional Zero added

3. Bitwise Logical Operations

3. Bitwise logical operations

- Bitwise logical operations are including:

Logical operation	Symbol	Example	
AND= Yield TRUE if both operands are TRUE	(•) , (^)	A • B	A ^ B
OR = Yield TRUE if either/both operand is/are TRUE	(+) , (v)	A + B	A v B
NOT = Inverts the value of its operand	(’) , (¬)	A’	¬ A
XOR = exclusive disjunction	(⊕) , (⊻)	A ⊕ B	A v B

3. Bitwise logical operations

- Bitwise Operator precedence

Expression	Order of Operations
A + B’	NOT, then OR
(A + B)’	OR, then NOT
A + (B • C)	AND, then OR

- Relative order of precedence: **NOT** > **XOR** > **AND** > **OR**

3. Bitwise logical operations

- Bitwise logical operations
- E.g.: **AND**

	1	1	0	0
AND	1	0	1	0
	0	0	0	0

- Bitwise logical operations
- E.g.: **OR**

	1	1	0	0
OR	1	0	1	0
	1	1	1	0

3. Bitwise logical operations

- Bitwise logical operations
- E.g.: **NOT**

OR	1	0
	0	1

- Bitwise logical operations
- E.g.: **XOR**

	1	1	0	0
XOR	1	0	1	0
	0	1	1	0

Chapter Review

Chapter Review

1. Signed and unsigned number representation

- ❑ Signed: Positive only
- ❑ Unsigned: Positive (0) /negative (1)

2. Floating-point Num Representation & Standard

- ❑ Sign-magnitude notation

  $\pm 0.12345 \times 10^{\pm 5}$

- ❑ SEEMMMMM notation

  Excess-50, 1=+ve, 2=-ve

  25512345

- ❑ IEEE754 notation
  - ❖ Single precision (1:8:23)
  - ❖ Double precision (1:11:1023)
- ❑ Floating-point number conversion
  - ❖ B / O / H → D: $2/8/16^{-n}$
  - ❖ D → B / O / H: Mantissa x base
  - ❖ B → O /H: Collapse
  - ❖ H / O → B: Expand

3. Bitwise logical operations

Operator	Symbol	Example	
AND	(•) , (^)	A • B	A ^ B
OR	(+) , (v)	A + B	A v B
NOT	(’) , (¬)	A’	¬ A
XOR	(⊕) , ($\underline{\vee}$)	A ⊕ B	A v B