CSED211: Lab. 2

DataLab2

조현욱, 박용곤

csed211-ta@postech.ac.kr

POSTECH

2023.09.11

Table of Contents

- DataLab 2
  - 2's complement
  - Floating point
- Quiz
- Homework

In Last Session

- Linux commands
- Bit and Byte
- Bitwise operation (~, &, |, ^, <<, >>)
- Homeworks about integer

2's Complement

- -x == ~x +1

- Ex) 5 → 0100
  -5 → 1011

Bits	Signed	Unsigned
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-8	8
1001	-7	9
1010	-6	10
1011	-5	11
1100	-4	12
1101	-3	13
1110	-2	14
1111	-1	15

=

+/- 16

2's Complement Computation

- Addition and subtraction of 2's complement does not need any other operation

- Ex) 15 + (-5) 15 - 35

```
  0000 1111   (15)
+ 1111 1011   (-5)
===========
  0000 1010   (10)
```

```
 11100 000    (borrow)
  0000 1111   (15)
- 0010 0011   (35)
===========
  1110 1100   (-20)
```

Floating Point

- Fractional binary numbers
- IEEE floating point standard

Fractional Binary Numbers

- Representation

2^i	2^{i-1}	•••	2^2	2^1	2^0					
b_i	b_{i-1}	•••	b_2	b_1	b_0	b_{-1}	b_{-2}	b_{-3}	•••	b_{-j}
						2^{-1}	2^{-2}	2^{-3}	•••	2^{-j}

- Limitations
  - Limitation 1. How can we represent $\frac{1}{3}$, $\frac{1}{7}$, …?
    - Can only exactly represent numbers of the form $x/2^k$
  - Limitation 2. How can we represent 2^{i+1}?
    - Limited range of numbers

IEEE Floating Point Standard

- Representation

$$(-1)^s\, M\, 2^E$$

  - S determines whether number is negative or positive
  - M normally a fractional value in range [1.0, 2.0)
  - E weights value by power of two

- Encoding
  - MSB is a sign bit S
  - Exp field encodes E (not exactly the same as E)
  - Frac field encodes M (not exactly the same as M)

s	***e*** **(exp)**	***f*** **(frac)**

Precisions

- Single precision: 32 bits

s	*e*	*f*
1	8 bits	23 bits

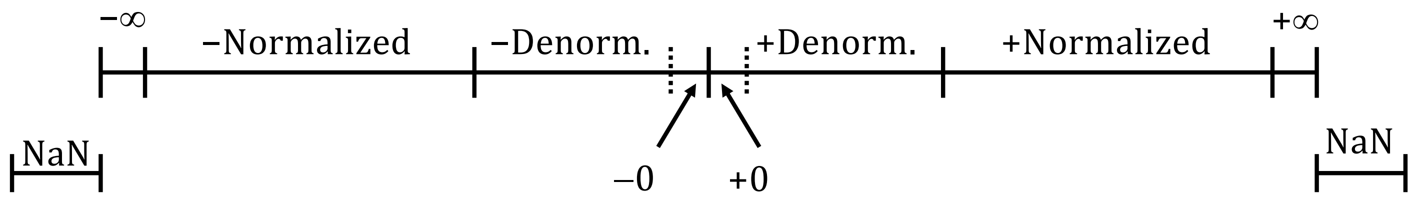
- Double precision: 64 bits

s	*e*	*f*
1	11 bits	52 bits

- Extended precision: (Intel only): 80 bits

s	*e*	*f*
1	15 bits	63 or 64 bits

Visualization: Floating Point Encoding



- Normalized value
- Denormalized value
- Special value

Normalized Value

- When exp != 000…0 and exp != 111…1

$$\mathbf{E = Exp - Bias}$$

  - Exp: unsinged value exp
  - Bias: $2^{k-1}-1$, where k is number of exponent bits
    - Single precision: 127
    - Double precision: 1023
  - M = 1 + f (fraction)
    - Minimum when 000…0 (M = 1.0)
    - Maximum when 111…1 (M = 2.0 - ϵ)

Normalized Encoding Example

$v = (-1)^s \times M \times 2^E$
$E = Exp - Bias$

- Value: `float f = 15213.0;`
  - `15213(10) = 11101101101101(2)`
    `= 1.1101101101101(2) × 213`
- Significand
  - *M* = `1.1101101101101` (2)
  - *f* = `11011011011010000000000` (2)
- Exponent
  - *E* = 13
  - *Bias* = 127
  - *exp* (e) = 140 = `10001100` (2)
- Result

0	1000 1100	110 1101 1011 0100 0000 0000
s	**exp (e)**	***f* (frac)**

Denormalized Value

- When exp = 000…0

$$\mathbf{E = 1 - Bias}$$

  - M = 0 + f (fraction)
  - Case1) exp = 000…0, frac = 000…0
    - Represent 0
    - There exist +0 & -0
  - Case2)
    - Very small number closes to 0.0

Special Values

- **When exp = 111…1**
  - Case 1) exp = 111…1, frac = 000…0
    - Infinity
    - Operation that overflow
    - Both positive and negative
  - Case 2) exp = 111…1, frac ≠ 000…0
    - Not-a-Number (NaN)
    - Represents case when no numeric value can be determined

Small Example: 8-bit Floating Point

s	*e*	*f*
1	4 bits	3 bits

- 8-bit floating point representation
  - The sign bit is in the most significant bit
  - The next four bits are the exponent, with a bias of 7
  - The last three bits are the fraction part

- The same general format as IEEE format
  - For normalized and denormalized numbers
  - For special values to represent 0, NaN, and infinity

Dynamic Range (Positive Only)

	s	exp	frac	E	Value	
Denormalized numbers	0	0000	000	-6	0	
	0	0000	001	-6	1/8*1/64 = 1/512	**Closest to zero**
	0	0000	010	-6	2/8*1/64 = 2/512	
	…					
	0	0000	110	-6	6/8*1/64 = 6/512	
	0	0000	111	-6	7/8*1/64 = 7/512	**Largest denorm.**
Normalized numbers	0	0001	000	-6	8/8*1/64 = 8/512	
	0	0001	001	-6	9/8*1/64 = 9/512	
	…					
	0	0110	110	-1	14/8*1/2 = 14/16	
	0	0110	111	-1	15/8*1/2 = 15/16	**Closest to 1 below**
	0	0111	000	0	8/8*1 = 1	
	0	0111	001	0	9/8*1 = 9/8	**Closest to 1 above**
	0	0111	010	0	10/8*1 = 10/8	
	…					
	0	1110	110	7	14/8*128 = 224	
	0	1110	111	7	15/8*128 = 240	
	0	1111	000	n/a	inf	**Largest norm**

Quiz

Homework

Lab Homework 2

- Due: 09/25 23:59 (midnight)

- Upload zip file which contains your source file and report
  - Explain your answer in the report
  - File name format (again): [student_#]_[name].c / .pdf, Lab[lab_#]_[student_#]_[name].zip
  - If you `unzip` in programming server, result should be **directory containing .c/.pdf files**

- Explain your answer in the report

- Refer to 'writeup_lab2' and following description

Homework Instruction

- Use the minimum number of operators as you can

- You are allowed to use only the following:
  - 1. Integer constants 0 through 255(0xFF)
  - 2. Function arguments and local variables
  - 3. Unary integer operations: !, ~
  - 4. Binary integer operations: &, ^, |, +, <<, >>

- Use `dlc` program to verify your solution
  - It will tell you whether you break the rule or not

Homework Instruction (cont.)

- You are expressly forbidden to:
  - 1. Use any control constructs: if, do, while, for, switch
  - 2. Define or use any macros
  - 3. Call any functions
  - 4. Use any other operations: &&, ||, -, or ?
  - 5. Use any data type other than int (cannot use arrays, structs, or unions)

Homework Instruction (cont.)

- You may assume that your machine:
  - 1. Uses 2's complement, 32-bit representation of integers
  - 2. Performs right shifts arithmetically
  - 3. Has unpredictable behavior when shifting an integer by more than the word size

Homework: Problem 1

- negate – return -x
- Example: negate(1) = -1
- Legal Ops: !, ~, &, ^, |, +, <<, >>
- Max ops: 5

```
int negate(int x) {
    // to be implemented
}
```

Homework: Problem 2

- isLess – if x < y then return 1, else return 0
- Examples: isLess(4, 5) = 1
- Legal ops: !, ~, &, ^, |, +, <<, >>
- Max ops: 24

```
int isLess(int x, int y) {
   // to be implemented
}
```

Homework Instruction (Floating Point)

- For this part of the assignment, you will implement some common single-precision floating point operation

- You are allowed to use standard control structures **(but not in previous problems)**
  - Conditional, loops

- You also use both int and unsigned data types **(but not in previous problems)**

- You **may not use unions, structs, or arrays**

- You **may not use any floating-point data types, operation, or constants**
  - Any floating-point operand will be passed to the function as having type unsigned, and any returned floating-point value will be of type unsigned

Homework: Problem 3

- float_abs – return absolute value of f
- Legal ops: Any integer/unsigned operations including ||, &&, if, while
- Max ops: 10

```
unsigned float_abs(unsigned uf) {
   // to be implemented
}
```

Homework: Problem 4

- float_twice – return 2*f
- Legal ops: Any integer/unsigned operations including ||, &&, if, while
- Max ops: 30

```
unsigned float_twice(unsigned uf) {
   // to be implemented
}
```

Homework: Problem 5

- float_i2f – return (float)x
- Legal ops: Any integer/unsigned operations including ||, &&, if, while
- Max ops: 30

```
unsigned float_i2f(int uf) {
   // to be implemented
}
```

Homework: Problem 6

- float_f2i – return (int)f
- Legal ops: Any integer/unsigned operations including ||, &&, if, while
- Max ops: 30

```
int float_f2i(unsigned uf) {
   // to be implemented
}
```

How to Do?

- Download “bits.c” file and implement each function
- Use only legal operation and definition of the function
- Please use PLMS Q&A board instead of sending e-mails to TA
  - Do not delete your question after you get your answer

Q & A