COMP10002 Workshop Week 11

Representation of integers and floats, Ex. 13.1
File Operations: Ex13.2

1. Try your hand at Exercise 11.3, **time limit: 10 minutes**

2. Assignment 2:
   1. make progress & ask questions,
   2. submit on Gradescope

A number can be written using different base

Base 10 (decimal system) uses 10 digits 0..9 — 10 is written using 2 digits: 10

Base 2 (binary) uses 2 digits 0..1 — 2 is written using 2 digits: 10

Base 16 (hexadecimal) uses 16 digits 0..9, A..F — 16 is written using 2 digits: 10

the number $\ldots\ d_3\ d_2\ d_1\ \mathbf{d_0}\,.\ d_{-1}\ d_{-2}\ \ldots\ _{(B)}$ (in base B)

has the value $\ldots + d_3 x B^3 + d_2 x B^2 + d_1 x\, B^1 + \mathbf{d_0} + d_{-1} x B^{-1} + d_{-2} x B^{-2} \ldots$

Examples:

$1011.101_{(2)} = \ ?_{(10)}$

$3A.2_{(16)} = \ ?_{(10}$

Practical advice: for conversions, remember:

128	64	32	16	8	4	2	1	0.5	0.25	0.125
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}

Note: Converting between bases 2 and 16 is easy!

$16=2^4$ ➔ 1 hexadecimal digit is equivalent to 4 binary digits.

$110101101_{(2)}$	⇔	$1\ 1010\ 1101_{(2)}$	⇔	$1AD_{(16)}$
11111011011	⇔		⇔	?
111.110001	⇔		⇔	?
?	⇔		⇔	5F.B6

Notes: hexadecimal is normally used for writing binary numbers in a shorter format

Exercises: Converting Decimal→Binary

Recap:

To change a decimal x to binary: represent x as the sum of power of two.

Example: $23.375 = 16 + 0x8 + 4 + 2 + 1 + 0x0.5 + 0.25 + 0.125$

$2^4 + \quad 2^2 + 2^1 + 2^0 + \quad + 2^{-1} + 2^{-2}$

So: $23.375 = 10111.011_{(2)}$

Exercises: 132 → $_{(2)}$

89.375 → $_{(2)}$

Representation of integers (in computers) using w bits

- Note that we use a fixed amount of bits `w`
- Make difference between `unsigned` and `signed` integers (`unsigned int` and `int` in C)

`unsigned` integers :

Range: $0..2^w-1$ ($0..255$ for $w=8$)

Representation: Just convert to binary, then add `0` to the front to have enough `w` bits.

Examples: represent 89 and 257 as unsigned int with w= 8

Representation of signed integers using w bits

signed integer range: -2^{w-1} to $2^{w-1}-1$

(-128 .. +127 for w=8)

(-2^{31} .. $+2^{31}-1$ for w=32)

- To represent signed integers x:
  - Positive numbers ($x \geq 0$):
    - just like unsigned: the binary representation of x in w bits,
    - → the first bit will always be 0.
  - Popular Method for negative numbers ($x<0$):
    - using *twos-complement* of |x| in w bit,
    - →the first bit will always be 1.

Negative int: Finding twos-complement representation in w bits in 3 steps

Suppose that we need to find the twos-complement representation for `-x`*, where* `x` *is positive, in* `w=16` *bits. Do it in 3 steps:*

1) *Write binary representation of* `|x|` *in* `w` *bits*
2) *Find the rightmost* `one-bit`
3) *Inverse (ie. flip 1 to 0, 0 to 1) all bits on the left of that rightmost* `one-bit`

find the 2-comp repr of `-40`	**Bit sequence**			
1) bin repr of 40 in 16 bits	0000	0000	0010	1000
2) find the `rightmost 1`	0000	0000	0010	1000
3) inverse its left	1111	1111	1101	1000

Why? Think about finding (-x) so that x + (-x) = 0, where x is a bit pattern.

Ex. 13.01

Suppose that a computer uses w= 6 bits to represent integers. Calculate the two-complement representations for 0, 4, 19, -1, -8, and -31; Verify that 19-8 = 11;

value	representation
0	
4	
19	
35	
-1	
-8	
-31	

verify!		
+	19	
	-8	
=	11	

Representation of floats

Many different ways! We learnt 2 formats:

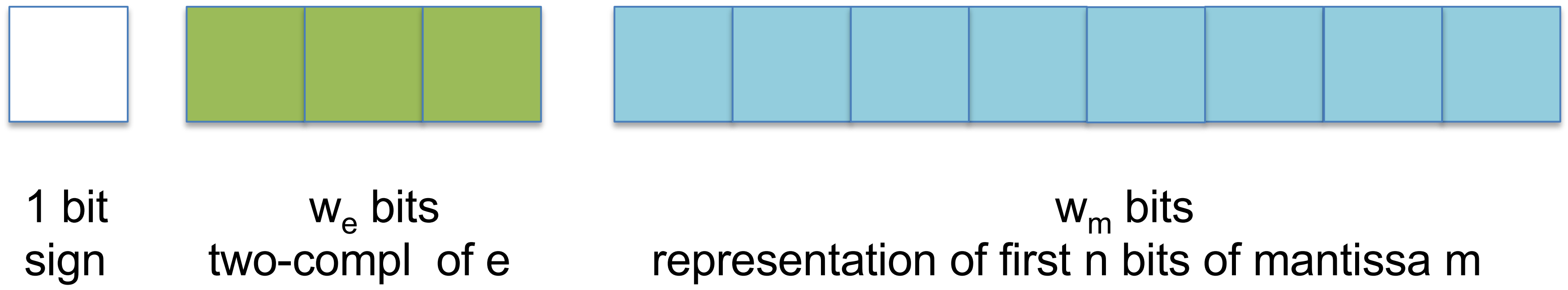
- *format-1*: using 16 bits, as described in `lec09.pdf` and in the text book
- *format-2* : using 32 bits, which is is an `IEEE standard`, that
    - is employed in most of modern computers for type `int`,
    - is demonstrated in the lecture, and
    - you can experiment with using the program `floatbits.c` `(lec09.pdf`, around `page 26)`.

General Principle for representing a `float x`? (for example, `x= -20.75`) :

- to get rid of the sign (`+` or `-`) and the dot (`.`) by replacing `x` with 3 *integers*: sign `s`, mantissa `m`, exponent `e` so that:
    - `s` is `0` (for positive) or `1` (for negative)
    - `|x|` is equivalent to $0.m \times 2^{e}$ (*format-1*) or $1.m \times 2^{e}$ (*format-2*)

Representation of floats (format-1: w=16, as described in lec09.pdf)

x=(`sign, e , m )`



Convert $/x/$ to binary form, and transform so that:

$$|x| = 0.b_0b_1b_2\ldots \times 2^e \quad \textit{where } b_0 = 1$$

e is called exponent, $m = b_0b_1b_2\ldots$ is called mantissa

x is represented as the triple (`sign, e, m)` as shown in the diagram.

$|x|$ is equivalent to $0.m \times 2^e$

Ex. 13.02

Suppose w_s=1, w_e=3, w_m=12, what's the representation of 2.0, -2.5, 7.875 ?

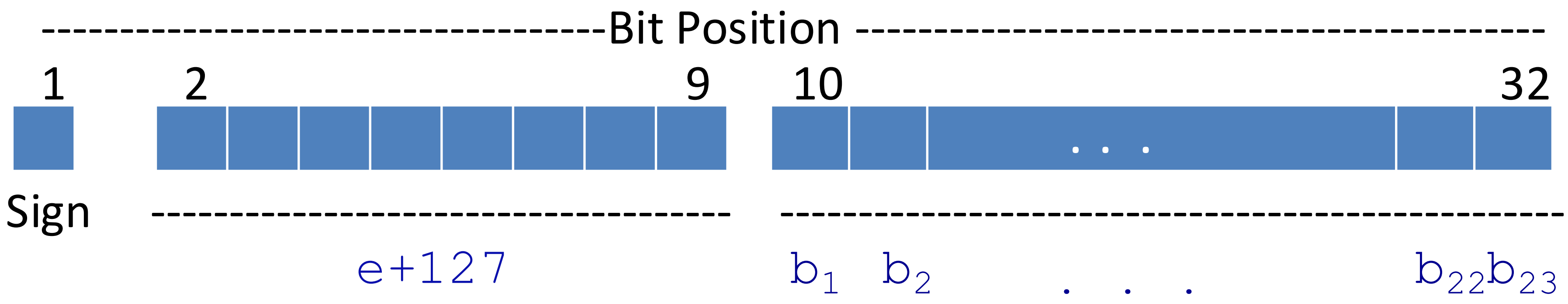
value	binary form	S= ? m= ? e= ?	representation
2.0	10.0		
-0.375			
7.875			

Representation of 32-bit float: (format-2: IEEE 754, as in floatbits.c)

w_s=1, w_e=8, w_m=23

$|x| = \mathbf{1}.b_1b_2\ldots \times 2^e$

$|x|$ is equivalent to $1.m \times 2^e$



Here:

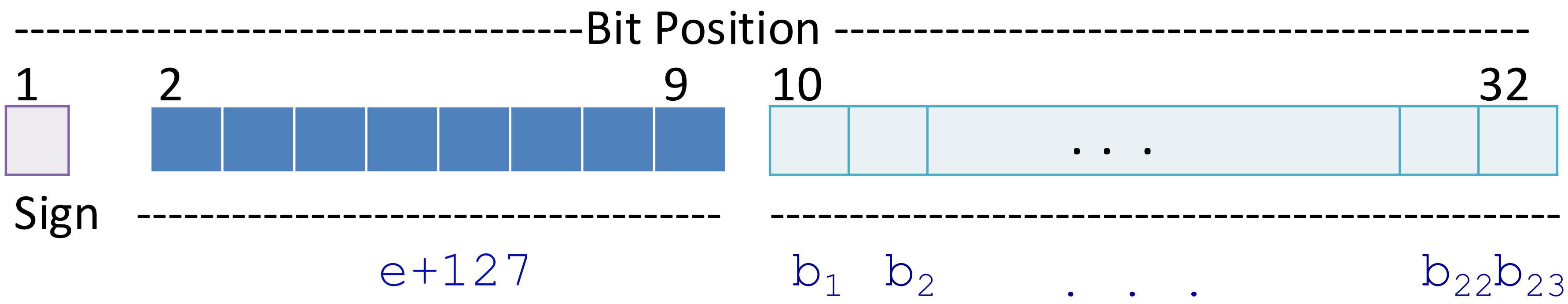
- The sign bit is 0 or 1 as in the previous case
- e is represented in the excess-127 format: as the unsigned value $e+127$ in $w_e=8$ bits
- The integer part 1 is omitted from the representation, and the mantissa is represented as just $b_1b_2\ldots b_{23}$

Class Exercise: IEEE representation for -3.375, 0.125

Note: There are 256 possible values for e. But valid e is -126 → +127, corresponding to bit patterns values 0000 0001 → 1111 1110

Representation of 32-bit float: (IEEE 754, as in floatbits.c)

w_e=8



Un-important note on the exponent part:

- Valid `e` is -126 → +127, corresponding to bit patterns `0000 0001` → `1111 1110`
- Pattern `0000 0000` used for representing `0.0`,
- Pattern `1111 1111` used to represent `infinity`,
- And, `0.0` is all 32 `zero-bit`, and `infinity` is all 32 `one-bit`.

Class Exercise: IEEE representation for 3.5

Representation of 32-bit float: (IEEE 754, as in floatbits.c)

- w_s=1, w_e=8, w_m=23
- $|x| = 1.m \times 2^e$

Example: x= -3.5

In binary: x= -11.1 → $|x| = 1.11 \times 2^1$

→ sign bit: 1

→ e=1 is represented as e+127=128 in 8 bits

→ e is represented as

→ m= 23-bit mantissa: 110 0000 0000 0000 0000 0000

→ Final representation:

0100 0000 0110 0000 0000 0000 0000 0000

or 4 0 6 0 0 0 0 0 $_{(16)}$

LAB TIME: Assignment 2 + exercise 11.3

- 2b|~2b : do exercise 11.3 in less than 10 minutes
- Work on assignment 2, ask questions

LAB: (Do Assignment 2 || Do Ex 11.3) && Q&A for Assignment 2

Ex 11.3: The Unix `tee` command writes its `stdin` through to `stdout` in the same way that the `cat` command does. But it also creates an additional copy of the file into each of the filenames listed on the command-line when it is executed.

Implement a simple version of this command.

Hint: you will need an array of files all opened for writing.

```
./program file1 file2
Hello world!
[^D]
Hello world!
```

[The program will also create two files named `file1` and `file2`, both containing `"Hello world!\n"` as the file content.]

```
//  here is an implementation of command cat
//  change it to that for tee

int main(int argc, char *argv[]) {
  char c;

  while ( (c=getchar()) != EOF) {
    putchar(c);
  }

  return 0;
}
```

Additional Slides

Decimal→Binary: A procedure for converting Integer Part

*Changing integer x to binary: Just divide x and the subsequent quotients by 2 until getting zero. The sequence of remainders, **in reverse order** of appearance, is the binary form of x.*

Example: 23

operation	quotoion	remainder
23 :2	11	1
11:2	5	1
5:2	2	1
2:2	1	0
1:2	0	1

So: $23 = 10111_{(2)}$ $11 = {}_{(2)}$ $46 = {}_{(2)}$

Decimal→Binary: General Method for the Fraction Part

Problem: Converting fraction could be complicated

0.1= 0x0.5 + 0x0.25 + 0x0.125 + 0.0625 + 0.0375 (= …)

Easy method: *Multiply it, and subsequent fractions, by 2 until getting zero.*
Result= sequence of integer parts of results, in appearance order. Examples:

0.375				0.1		
operation	int	fraction		operation	int	fraction
.375 x 2	0	.75		.1 x 2	0	.2
.75 x 2	1	.5		.2 x 2	0	.4
.5 x 2	1	.0		.4 x 2	0	.8
				.8 x 2	1	.6
				.6 x 2	1	.2

So: $0.375 = 0.011_{(2)}$ $0.1 = 0.00011(0011)_{(2)}$

part inside ( ) is repeated indefinitely