COMP10002 Workshop Week 11

≤45 minutes	Representation of integers and floats
	Ex. 13.1 and 13.2

LAB	• Try your hand at Exercise 11.3, time limit: 10 minutes (including reviewing file operations in lec08.pdf)
	• Assignment 2:
	  • ask questions during the workshop,
	  • finish by 11:59PM *today or 2AM tonight* ☺

Requirements in Canvas	• Discuss Exercises 13.1 and 13.2. Try your hand at Exercise 11.3 • Then focus the rest of your energy on Assignment 2. Make sure that you make regular submissions right through the week. • *And browse through the discussion questions from time to time. Reading them might save you some marks.*

A2 situation (deadline 6PM this Friday)

Please send me a letter on

how far you went with assignment 2?

A. finished ass2, or just need to do some refinements or improvements along the marking rubric
B. finished stage 1
C. finished stage 0
D. finished none, but have clear idea on how to do and can certainly finish stage 0 very soon
E. still struggling with stage 0

And which deadline is occupying your mind:
1- 6PM this Friday
2- oct 26 ☺

Numeral Systems

214.39	2	1	1	.	3	9
Position	2	1	0	Dot	-1	-2
Value	2×10^2	1×10^1	4×10^0		3×10^{-1}	9×10^{-2}

→ *base* = 10 (decimal)

Other bases: binary (base= 2), octal (base= 8), hexadecimal (16)

$21.3_{(10)} = 2 \times 10^1 + 1 \times 10^0 + 3 \times 10^{-1}$

$1001_{(2)} = 1 \text{x} 2^3 + 0 \text{x} 2^2 + 0 \text{x} 2^1 + 1 \text{x} 2^0 = 9_{(10)}$

$5B_{(16)} = 5 \text{x} 16^1 + 11 \text{x} 16^0 = 91_{(10)}$

Note: hexadecimal uses 6 additional digits: A, B, C, D, E, F with the values 10-15

Converting between bases 2 and 16 is easy!

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

```
1 1010 1101      →  1AD
111 1101 1011    →
110.1100 0100     →   6.C8

      3CD      →  11 1100 1101
    AF.B5      →
```

Converting Binary → Decimal

Just expand using the base=2:

$$\ldots\ b_3\, b_2\, b_1\, \mathbf{b_0} \,\mathbf{.}\, b_{-1}\, b_{-2}\ \ldots\ _{(2)}$$

is $\ldots\ b_3 x 2^3 + b_2 x 2^2 + b_1 x\ 2^1 + \mathbf{b_0} + b_{-1} x 2^{-1} + b_{-2} x 2^{-2} \ldots$

Examples: `1101` →

`1.011` →`1 + 0 + 0.25 + 0.125 = 1.375`

Practical advice: 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}

Decimal→Binary: Convertion using definition

Changing integer 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

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

129 → 10000001

128 → 10000000

127 → 1111111

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: $0.375 = 0.011_{(2)}$ $0.1 = 0.0(0011)_{(2)}$

Exercise: Converting Decimal->Binary

$130_{(10)}$ =

$6.625_{(10)}$ = 110.101 0.125

$9.23_{(10)}$ = 1001.001110…

	x 2 =	
.23	0	.46
.46	0	.92
.92	1	.84
.84	1	.68
.68	1	.32
.32	0	.64

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. 18 → 10010 → 0001 0010

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 ($x \geq 0$):
    - the binary representation of `x` in `w` bits,
    - the first bit will always be `0`.
  - Negative numbers ($x < 0$):
    - using *twos-complement* of $|x|$ in `w` bit,
    - the first bit will always be `1`.

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. 13.1 (grok's W12)

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	000000
4	000400
19	010011
1	000001
-1	111111
	001000
-8	111000
31	011111
-31	100001

verify!		
+	19	010011
	-8	111000
=	11	001011

Representation of floats

We learnt 2 different formats:

- one as described in `lec09.pdf` and in the text book
- another is an `IEEE standard`, which is:
  - employed in most of modern computers,
  - demonstrated in the lecture, and
  - you can find/experiment with using the program `floatbits.c (lec09.pdf`, around `page 26)`.

- How to represent a `float x`? (General approach) :
  - replace `x` by 3 intergers: sign `s`, mantissa `m`, exponent `e`
  - `s` can be `0` (positive) or `1` (negative)
  - $|x|$ is equivalent to $0.m \times 2^e$ or $1.m \times 2^e$

Representation of floats (w=16, as described in lec09.pdf)

x=(`sign, e , m )`

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

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

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= 1101.10= 1101.10 x 2^0 = 110.110x2^1= 0.1101100000x2^4

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

Ex. 13.2 (grok's W12)

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

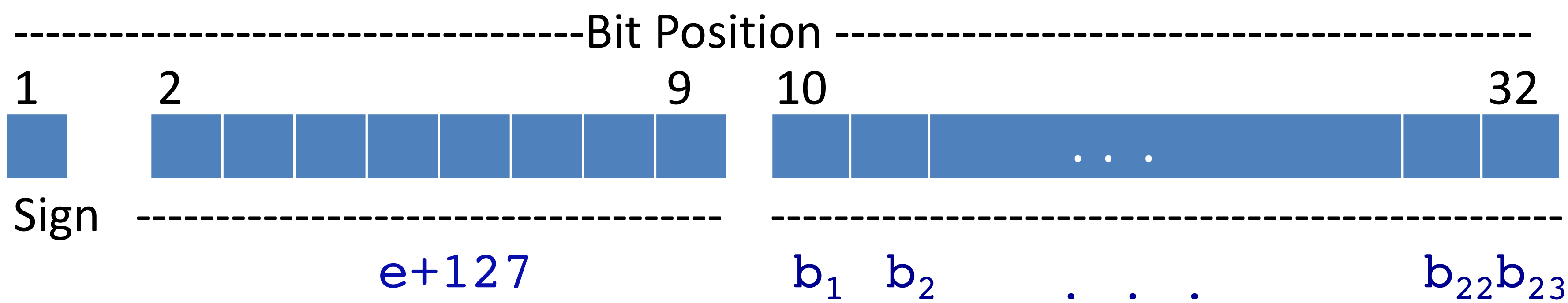
value	binary form	|x| is equivalent to $0.m \times 2^e$	representation
2.0	10.0	0.100 x 2^2, m= 100… e=2	0 010 1000 0000 0000
-0.25	-0.01	-0.1 x 2^(-2), m= 1… e=-2 2->010	1 110 1000 0000 0000
7.875			

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

w_s=1, w_e=8, w_m=23

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

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



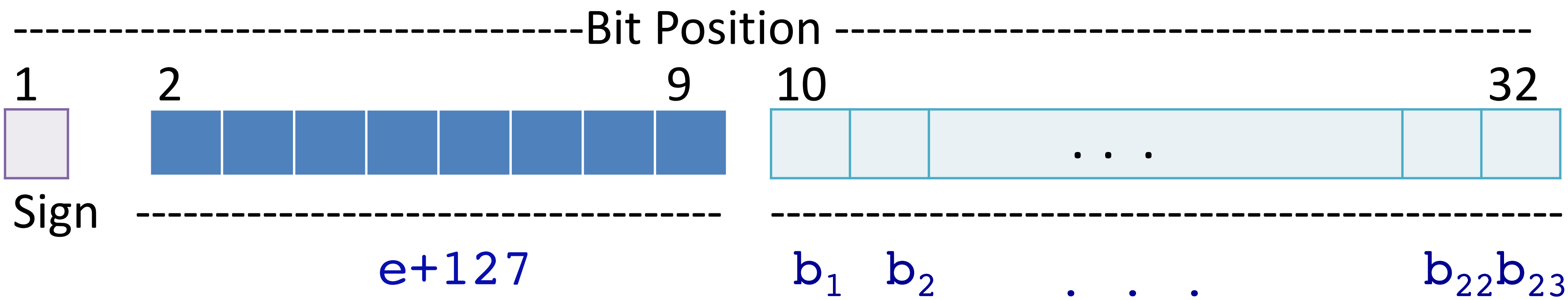
That is:

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

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`.

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 converted to |x|= 1.11 x 2^1

➔ sign bit: 0

➔ e=1 is represented as e+127= 128 in 8 bits ->

➔ e is represented as 1000 0000

➔ 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 $_{(16)}$

Ex 11.3 (grok's W11)

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;
}
```

LAB TIME: Assignment 2 + exercise 11.3

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

PROTOCOL

- You can just do your work, and ask questions when needed.
- Your choice between:
  - joining your break-out room, and
  - staying in the main room.
- You can discuss, but not show your code.
- Look at the big picture (depending on your progress) first, and ask questions.
- It's possible for Anh to have a look at your code, but you should:
  - ask him first, and he will put you in a "lock-down" room where you can show your code
  - prepare questions well so that you will probably need just <= 5 minutes to work with Anh

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