COMP10002 Workshop Week 11

Outlook:

1	Representation of integers and floats (lec09-a → lec09-e video)
2	Ex. 13.1 and 13.2
3	ass2: Q&A
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. • Note that the marking for presentational deductions (as listed in the rubric) will be stricter this time round, so make sure that before you make your final submission you review your Assignment 1 marking comments, to check that you haven't repeated any of the issues you were warned about.

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)

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

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

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

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!

Because 1 hexadecimal digit is equivalent to 4 binary digits.

```
110101101     →
     AFB5     →

111110011011  →
        3CD   →
```

Converting Binary → Decimal

Just expand using the base=2:

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

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` →

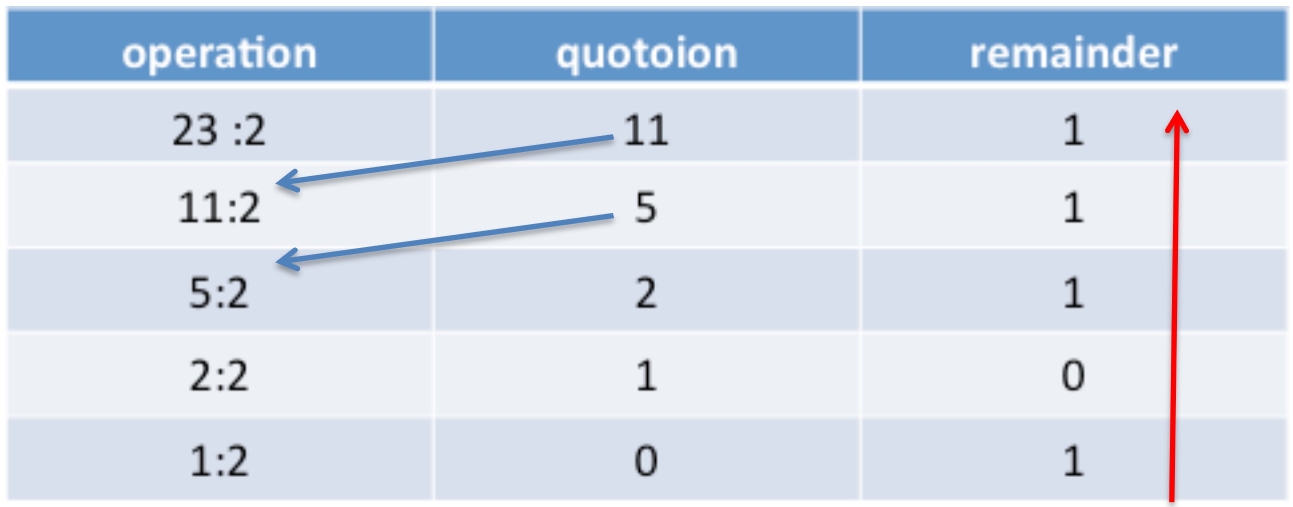
Practical advise: remember

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: 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: Fraction Part

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

Now try convert: 6.875 to binary

Exercise: Converting Decimal->Binary

$7_{(10)} = ?_{(2)} = ?_{(16)}$

$130_{(10)} =$

$6.375_{(10)} =$

$9.2_{(10)} =$

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.

Representation of integers (in computers) using w bits

`signed` integer range: -2^{w-1} to $2^{w-1}-1$
(-128 .. +127 for w=8)

- To represent `signed` integers `x`:
  - Positive numbers: a `0-bit`, followed by the binary representation of `x` in `w-1` bits.
  - Negative numbers: using *twos-complement* of `x` in `w` bit. The first bit will always be `1`.

Finding twos-complement representation in w bits for negative numbers in 3 step

Suppose that we need to find the twos-complement representation of $-x$, *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

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;

Quiz

What is the binary form of $255_{(10)}$?

A. 1000 0000

B. 1111 1111

C. 1 0000 0000

D. 1 1111 1111

Quiz

What is the binary form of $13.625_{(10)}$?

A. `1101.101`

B. `1011.101`

C. `1101.11`

D. `1011.11`

Representation of floats

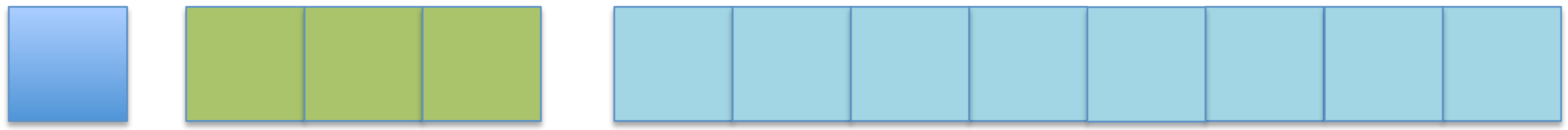
We learnt 2 different formats:

- 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 `p.25)`.

- How to represent? (General approach)

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

x=(sign, e , m)



1 bit	w_e bits	w_m bits
sign	two-compl of e	representation of first n bits of mantissa 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

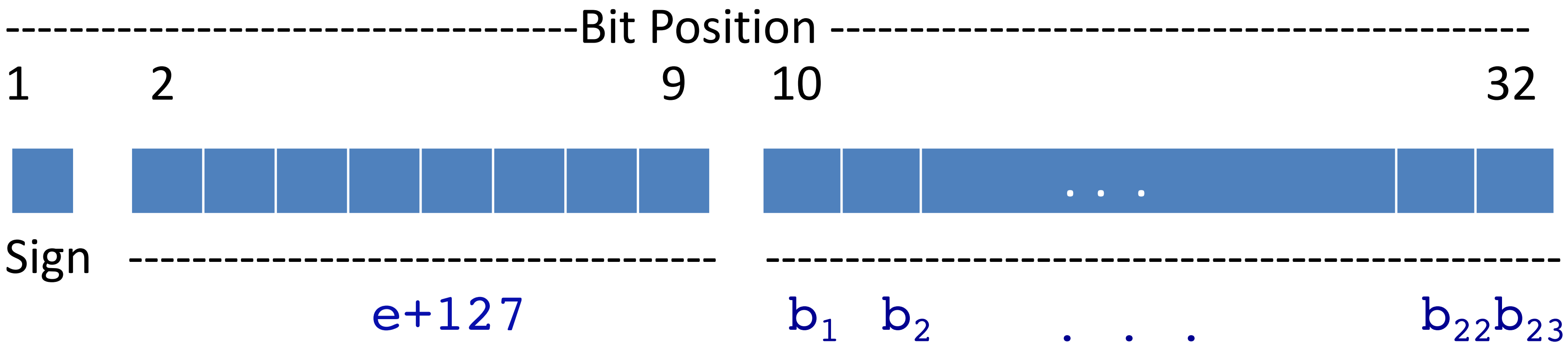
x is represented as the triple (sign, e, m) as shown in the diagram.

Ex. 13.2

Suppose w_s=1, w_e=3, w_m=12, what's the representation of 2.0, -2.5, 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… x 2^e [Note the *different transformation* of $|x|$]



That is:

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

Note: Valid e is -126 → +127, corresponding to values 1 →254 . Value 0 used for representing 0.0, value 255 used to represent infinity. And, zero 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 $|x| = 1.b_1b_2\ldots \times 2^e$

Example: x= 3.5

In binary: x= 11.1= 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

→ 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)
```

ass2: Important Notes

Submissions for Assignment 2 are being collected **via the LMS Assignment facility**,
Go to Assignment 2 Submissions to make your submission. You may submit as many times as you like prior to the deadline.
Prior to the deadline you may also test your program using the VPN-free test/verify site that was used for Assignment 1, see https://submit-web.eng.unimelb.edu.auLinks to an external site..
Important: The VPN-free test/verify service might not work near the deadline, if you wish to test your program with it, *start the assignment early and plan to finish it early too*.

- Submission for Assignment 2 will be via the LMS Assignments page, so that we don't have a recurrence of the load/authentication problems that plagued Assignment 1.
- You should start with ass2-skel.c, sign the declaration by putting on you name and date, feel free to delete/change the provided code.
- The marking for presentational deductions (as listed in the rubric) will be stricter this time round, so make sure that before you make your final submission you review your Assignment 1 marking comments, to check that you haven't repeated any of the issues you were warned about. *(there are much more chances of getting presentation deductions in ass2 in comparison with ass1)*

ass2: Progress? Q&A