

COMP10002 Workshop Week 11

≤45 minutes	Representation of integers and floats
	Ex. 13.1 and 13.2
LAB	<ul style="list-style-type: none">• Try your hand at Exercise 11.3, time limit: 10 minutes (including reviewing file operations in lec08.pdf)• Assignment 2:<ul style="list-style-type: none">• ask questions during the workshop,• finish by 11:59PM <i>today or 2AM tonight</i> 😊
Require ments in Canvas	<ul style="list-style-type: none">• 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.• <i>And browse through the discussion questions from time to time. Reading them might save you some marks.</i>

A2 situation (deadline 6PM this Friday)

On arrival, 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. none of the above

Numeral Systems

211.39	2	1	1	.	3	9
Position	2	1	0	Dot	-1	-2
Value	2×10^2	1×10^1	1×10^0		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!

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

110101101 →

11111011011 →

110.110001 →

3CD →

AF.B5 →

Converting Binary \rightarrow Decimal

Just expand using the base=2:

$$\begin{array}{c} \dots b_3 \text{ } \mathbf{b_2} \text{ } \mathbf{b_1} \text{ } \mathbf{b_0} \cdot \text{ } \mathbf{b_{-1}} \text{ } b_{-2} \dots (2) \\ \swarrow \quad \searrow \quad \downarrow \quad \searrow \\ \text{is } \dots b_3 \times 2^3 + \mathbf{b_2 \times 2^2} + \mathbf{b_1 \times 2^1} + \mathbf{b_0} + \mathbf{b_{-1} \times 2^{-1}} + b_{-2} \times 2^{-2} \dots \end{array}$$

Examples: 1101 \rightarrow
 1.011 \rightarrow

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: Conversion using definition

Changing integer x to binary: represent x as the sum of power of two.

Example: $23.375 = 16 + 0 \times 8 + 4 + 2 + 1 + 0 \times 0.5 + 0.25 + 0.125$

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

129 →

128 →

127 →

Decimal → Binary: General Method for the Fraction Part

Problem: Converting fraction could be complicated

$$0.1 = 0 \times 0.5 + 0 \times 0.25 + 0 \times 0.125 + 0.0625 + 0.0375 (= \dots)$$

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

Exercise: Converting Decimal->Binary

$$130_{(10)} =$$

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

$$9.23_{(10)} =$$

	x 2 =	
.23		

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: $0 \dots 2^w - 1$ ($0 \dots 255$ for $w=8$)

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

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***

<i>find the 2-comp repr of -40</i>	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	
4	
19	
-1	
-8	
-31	

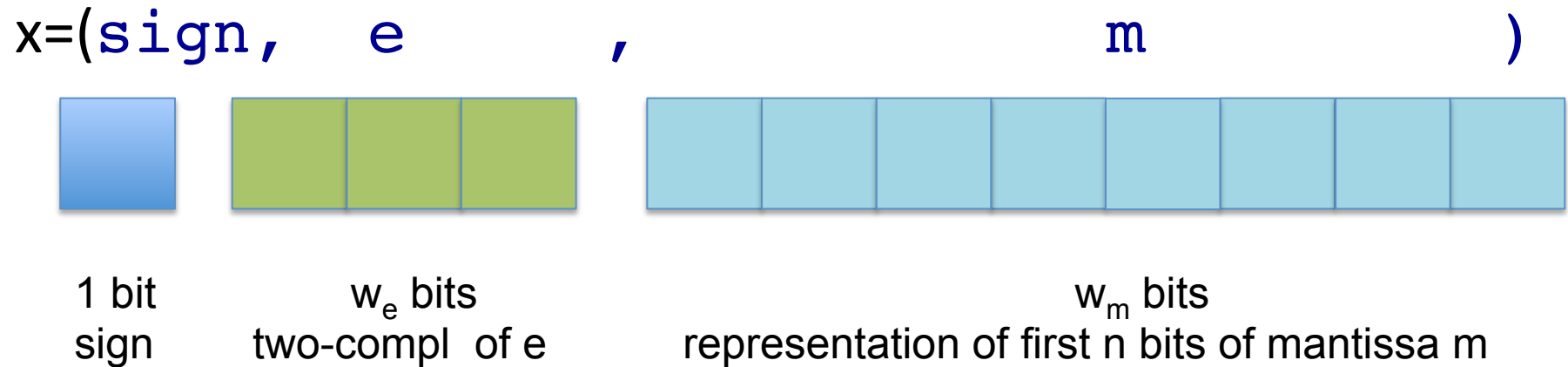
verify!		
+	19	
	-8	
=	11	

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 [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)



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

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

e is called exponent, $m = b_0b_1b_2...$ is called mantissa

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

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

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 ?

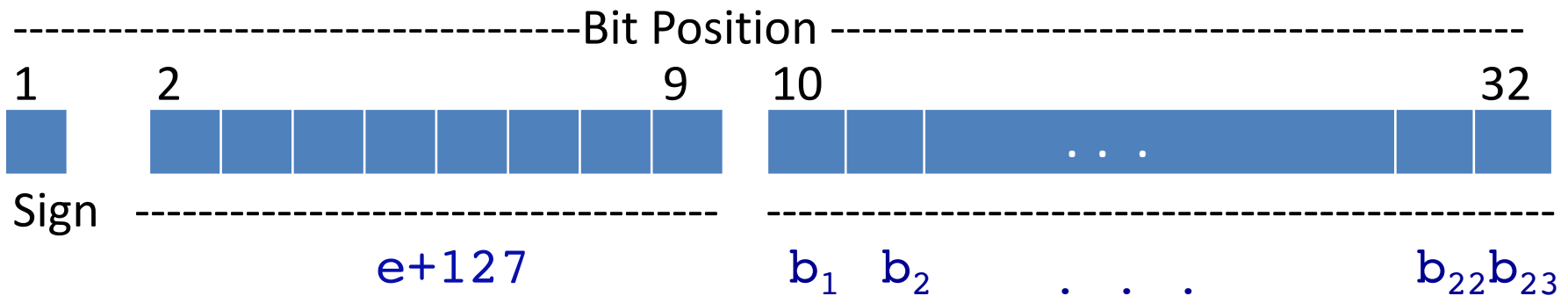
value	binary form		representation
2.0	10.0		
-0.375			
7.875			

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

$$w_s=1, w_e=8, w_m=23$$

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

|x| is equivalent to **1.m x 2^e**



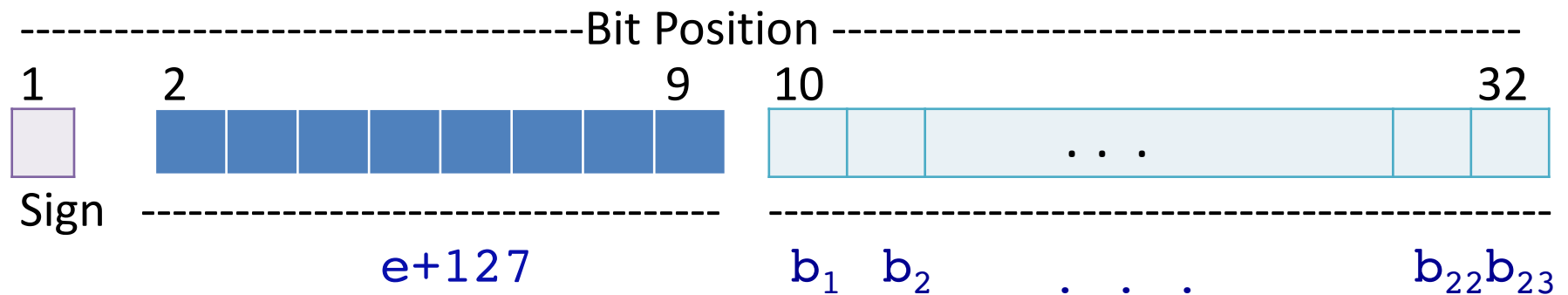
That is:

- 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 $b_1b_2...b_{23}$

Note: There are 256 possible values for e. But valid e is -126 \rightarrow +127, corresponding to bit patterns values 0000 0001 \rightarrow 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 $-126 \rightarrow +127$, corresponding to bit patterns $0000\ 0001 \rightarrow 1111\ 1110$
- 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 $|x| = ??$

→ sign bit: 0

→ $e=?$ is represented as e in 8 bits

→ e is represented as

→ 23-bit mantissa: 000 0000 0000 0000 0000 0000

→ Final representation:

00..0000 0000 0000 0000 0000 0000

or 4 0 6 0 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'd better do work in the main room, and ask questions when needed.
- Each student have a private break-out room:
 - if you want Anh to look at your code, alert him & join your room
 - questions without showing code should be asked in the main room.
 - if you want to discuss with some friends in a room, please let Anh know and he will accommodate (but don't show code to friends)
- Your choice between:
 - joining your break-out room, and
 - staying in the main room.
- Look at the big picture (depending on your progress) first, and ask questions.

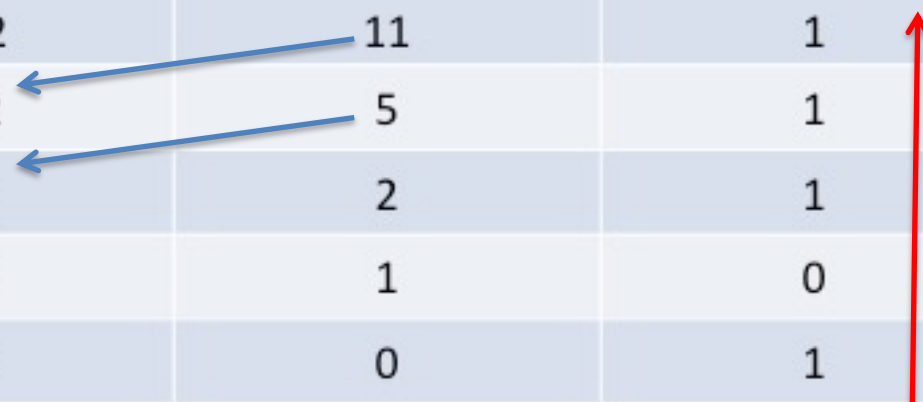
Additional Slides

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	quotient	remainder
23 :2	11	1
11:2	5	1
5:2	2	1
2:2	1	0
1:2	0	1



So: $23 = 10111_{(2)}$ $11 = 1011_{(2)}$ $46 = 101110_{(2)}$