CS2100 Tutorial 1

C and Number Systems

Slides by Theodore, adapted from Prof. Aaron's slides

About myself

Theodore Leebrant

(Theodore/Theo is good!)

- Computer Science + Mathematics
  - was a Programming Languages nerd, mostly with Rust
  - teaching CS3210 (Parallel Programming) as well this semester
- Plays too much Final Fantasy XIV
- Out-of-tutorial communication:
  - Email: theo@comp.nus.edu.sg for consults, questions
  - Telegram: next slide
  - Will reply messages within 24 hours – *except 2 days before deadlines*

Quick admin stuff

Telegram:

- *No groups for this semester sorry :(*
  - *I'm teaching 8 tutorial groups, so that's gonna be a lot of telegram chats*
  - *If I combine all into 1 chat, that's already ¼ of the cohort*
- Reachable via @kagamination on telegram

Slides will be uploaded here:
https://github.com/theodoreleebrant/TA-2425S1

Anonymous feedback: bit.ly/feedback-theodore

CS2100 seen from the Burj Khalifa

Part 1: "Software"	Part 2: "Hardware"	Part 3:"Optimization"
Lowering C to assembly (MIPS) to machine code	How does the processor work + how do we do digital logic design?	Can we improve the simple computer that we learnt?
C, MIPS, and a (prerequisite) of number systems	Datapath & Control, Boolean Alg, Logic Circuits, MSI Components, Sequential logic	Cache, Pipelining

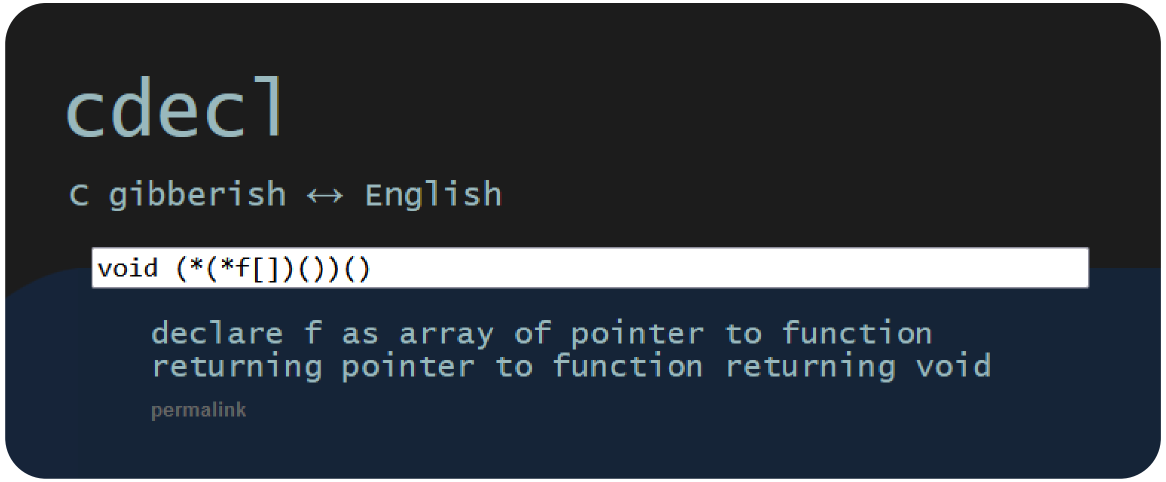
We're going as low-level as it is in the CompSci syllabus.

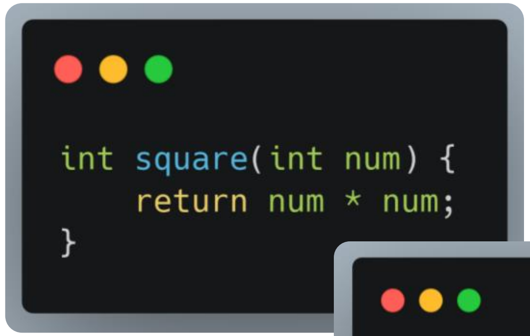
Our goals for the semester

(not just “finish the tutorials & labs” and “pass your exams”)

1. Get familiar with C and MIPS at the most basic level

You might not use it past CS2106, but it’s a good stepping stone to other languages





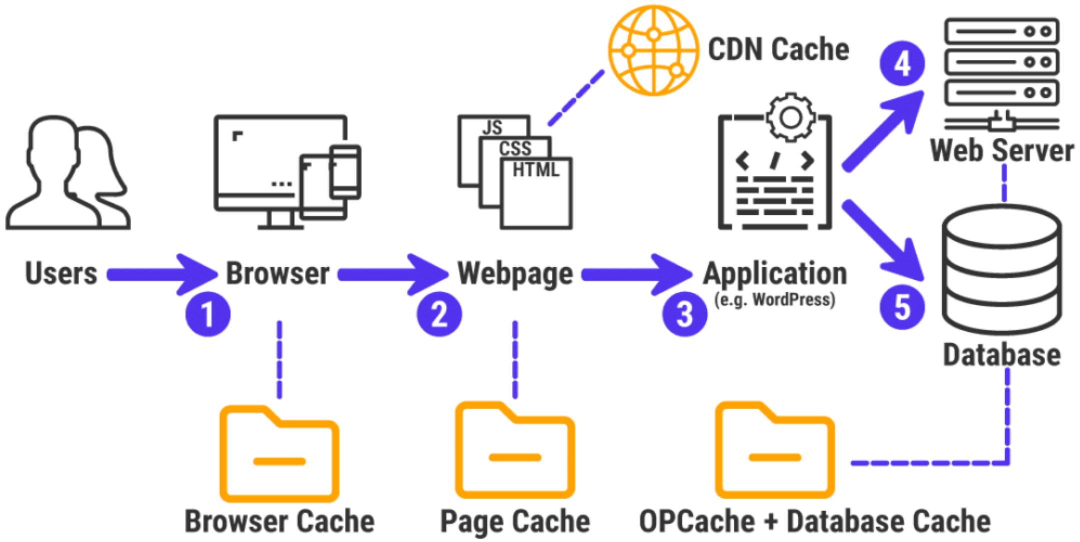
Credit: (left) cdecl.org/?q=void+(*(*f[])())()
(right) godbolt.org/ formatted with carbon.now.sh

Our goals for the semester

(not just “finish the tutorials & labs” and “pass your exams”)

2. Be comfortable with optimization

This course gives you a small view of optimization in the narrow scope of processor optimization, but it is still applicable to a bunch of other scopes.

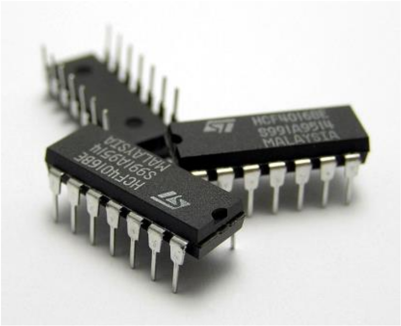


Our goals for the semester

(not just “finish the tutorials & labs” and “pass your exams”)

3. Have fun with hardware!

If you’re in CS, probably the first – and may be the last time you’re holding electrical components in an official course setting :)

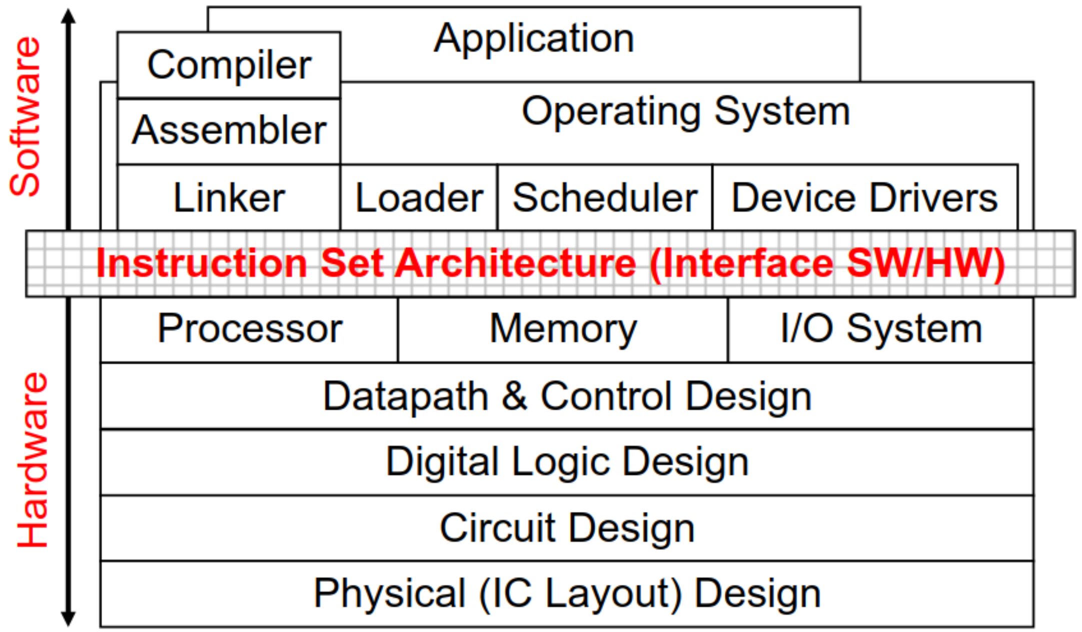


Where do we go past CS2100?

- There's a bunch of transferrable knowledge / skills from CS2100
  - CS2106 Operating Systems
    - continued on CS5250 Advanced OS
    - or CS4223 Multi-Core Architecture
  - CS4212 Compilers
  - Any other modules that requires you to write in C
    - and C++, to a smaller extent
  - Any other modules / internships that requires you to write assembly
    - anything microprocessors definitely, some IoT things

No such thing as stupid questions: *ever*

- "Systems" are made up of many, many parts
- Infinite learning process
- Please feel free to bring anything up
- Personal promise: all questions will be treated equally



General Tutorial Workflow

- Try to do the questions beforehand!
  - If you don't want to do the questions beforehand, **at the very least read the questions** so you're not completely lost when I'm explaining

- Short tutorial: only 1 hour
  - Will start at :05, and finish at :45
  - Feel free to leave for your next class if I overrun past this.

- Attendance marked!
  - Any excused absences let me know via email (theo@comp.nus.edu.sg)

CS2100 Tutorial 01

C and Number Systems

Overview

- Q1: Sign Extension
- Q2: Subtraction in 1s complement
- Q3: Fixed-point binary
- Q4: IEEE754 single-precision representation (“floats”)
- Q5: C basics – iteration, recursion, and arrays
- Q6: C pointers

CS2100 Tutorial 1 Number Systems

0 surveys completed

0 surveys underway

If you have 6 bits to represent a number in *unsigned integer*, how many numbers can you represent?

6

2^6

$2^6 - 1$

2^{6-1}

If you have 6 bits to represent a number in *1s complement*, how many numbers can you represent?

2^6

$2^6 - 1$

$2^6 - 2$

$2^{(6-1)}$

If you have 6 bits to represent a number in *2s complement*, how many number can you represent?

2^6

$2^6 - 1$

$2^6 - 2$

$2^{(6-1)}$

Unsigned, 1s complement, 2s complement

- Two ways of seeing n-bit 1's complement:
    1. If the MSB is 1, it's negative. Flip the rest of the bits to get the number
    2. The MSB has a value of $-(2^{n-1} - 1)$, the rest goes as usual

Example: 4-bit 1's complement

$(1101)_{1s}$

Unsigned, 1s complement, 2s complement

- Two ways of seeing n-bit 2's complement:
  1. If the MSB is 1, it's negative.
     Flip the rest of the bits and add 1 to get the number
  2. The MSB has a value of $-(2^{n-1})$, the rest goes as usual

Example: 4-bit 2's complement

$(1101)_{2s}$

Q1. Sign Extension

- For complement systems, we can extend the sign bit if we increase the number of bits used for the representation

Example:

From 4 bits to 8 bits:

$$(5)_{10} = (0101)_{2s} = (00000101)_{2s}$$

$$(-3)_{10} = (1101)_{2s} = (11111101)_{2s}$$

Note:
Sign extension works for complement systems – in binary, both 1s and 2s complement

Can sign extension work for sign-and-magnitude system?

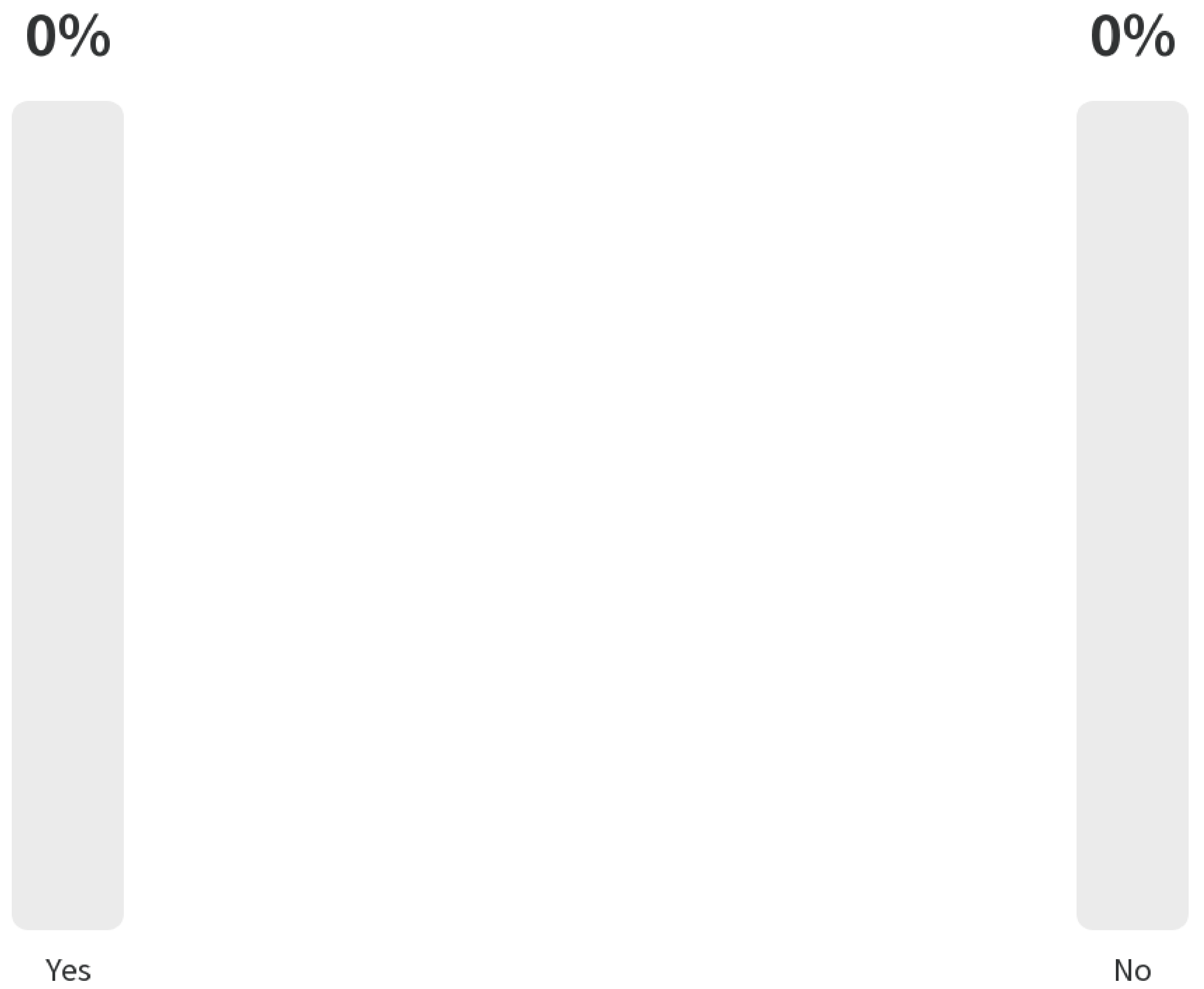
0

Yes

No

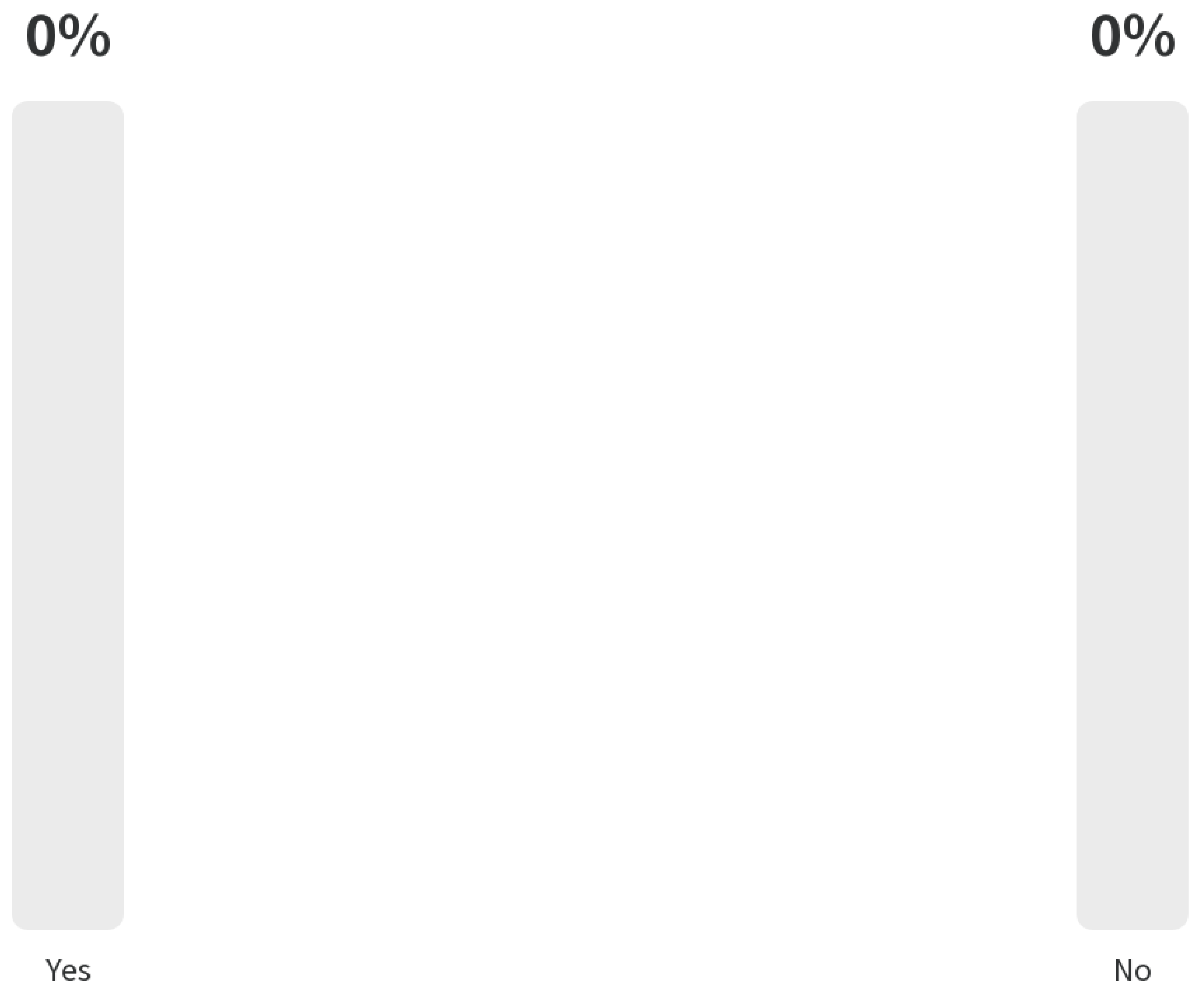
Can sign extension work for sign-and-magnitude system?

0



Can sign extension work for sign-and-magnitude system?

0



Q1. Sign Extension

Question: Show that sign extension does not change the value represented.

Answer:

Straightforward for positive values, since we pad zeroes

For negative numbers:

Reminder
One way to see 2s complement is that the MSB has a value of $-(2^{n-1})$, the rest goes as usual

$$(-3)_{10} = (1101)_{2s} = (\underbrace{1111}1101)_{2s}$$

Value: $-2^3 = -8$

Value: $-2^7 + 2^6 + 2^5 + 2^4 + 2^3$

$= -128+64+32+16+8 = -8$

Q1. Sign Extension

Reminder
One way to see 2s complement is that the MSB has a value of $-(2^{n-1})$, the rest goes as usual

For negative numbers:

$$(-3)_{10} = (1101)_{2s} = (\underbrace{11111}101)_{2s}$$

Value: $-2^3 = -8$

Value: $-2^7 + 2^6 + 2^5 + 2^4 + 2^3$

$= -128+64+32+16+8 = -8$

In general,

$$-2^{n-1}$$

$$-2^{m-1} + (2^{m-2} + 2^{m-3} + \dots + 2^n + 2^{n-1})$$
$$= -2^{m-1} + 2^{n-1}(2^{m-n} - 1) \text{ (sum of GP)}$$
$$= -2^{m-1} + 2^{m-1} - 2^{n-1}$$
$$= -2^{n-1}$$

Q2. Subtraction in 1s complement

(a) 0101.11 – 010.0101
(b) 010111.101 – 0111010.11

Side note:
Read the definition of (r-1)'s complement. Definition stands for more than just binary number systems.

Strategy: Convert A - B to A + (-B)

Why do we not perform subtraction directly?

Note
Adding trailing zeroes is not sign extension.

Extra question: what if these numbers are in 2s complement?

Q3. Decimal → Fixed point binary

Question

Convert from decimal to fixed point binary in 2's complement with 4 bits for the integer portion and 3 bits for the fraction portion

(a) $1.75 = (0001.110)_{2s}$

(b) $-2.5 = (1101.100)_{2s}$

(c) $3.876 \approx (0011.111)_{2s}$

(d) $2.1 \approx (0010.001)_{2s}$

Reminder: always do your conversion to one extra place and round accordingly

Q3. Decimal → Fixed point binary

<u>Question</u>
Convert it back to decimal.

(a) $(0001.110)_{2s} = (0001.110)_2 = 2^0 + 2^{-1} + 2^{-2} = 1.75$

(b) $(1101.100)_{2s} = -(0010.100)_2 = -(2^1 + 2^{-1}) = -2.5$

(c) $(0011.111)_{2s} = (0011.111)_2 = 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} = 3.875$

(d) $(0010.001)_{2s} = (0010.001)_2 = 2^1 + 2^{-3} = 2.125$

Q4. IEEE754 single-precision repr.

Question

Represent -0.078125 in IEEE 754 single-precision representation.

Express your answer in hexadecimal.

Answer

First step is to convert to binary and write it in the normalized form

$-0.078125 \quad = -(0.000101)_2 \quad = -(1.01)_2 \times 2^{-4}$

Q4. IEEE754 single-precision repr.

<u>Answer (cont.)</u>

After we get the normalized form, convert it to binary repr.

$-(1.01)_2 \times 2^{-4}$

Remember that exponent is in excess-127.

Exponent part is
$(-4 + 127)_{10} = (123)_{10}$
$= (01111011)_2$

Sign-bit	8-bit exponent	23-bit mantissa
1	01111011	01000000000000000000000

Q4. IEEE754 single-precision repr.

Answer (cont.)

Lastly, we convert into hexadecimal.

1	01111011	01000000000000000000000

Group into four bits for easier conversion

1011 1101 1010 0000 0000 0000 0000 0000
B D A 0 0 0 0 0

Answer: $(\text{BDA0 0000})_{16}$

Sanity check:
IEEE754 single-precision will always be 8 hexadecimal digits.

Q5. C basics: iteration, recursion, and arrays

Try on your own!

Let me know if you have difficulties.

Q6. C Pointers

Q: Trace the program and write out its output.

```
int a = 3, *b, c, *d, e, *f;

b  = &a;
*b =  5;
c  = *b * 3;
d  =  b;
e  = *b + c;
*d =  c + e;
f  = &e;
a  = *f + *b;
*f = *d - *b;
```

Summary

- Number systems:
  - Integers: unsigned binary, 1s complement, 2s complement, sign-and-magnitude
  - Non-integers: fixed-point representation, IEEE754 single-precision representation
- Sign extensions (vs. adding trailing zeroes)
- C: iteration, recursion, array access / writes, pointers

End of Tutorial 1

- Slides uploaded on github.com/theodoreleebrant/TA-2425S1
- Email: theo@comp.nus.edu.sg
- Anonymous feedback:
  bit.ly/feedback-theodore
  (or scan on the right)