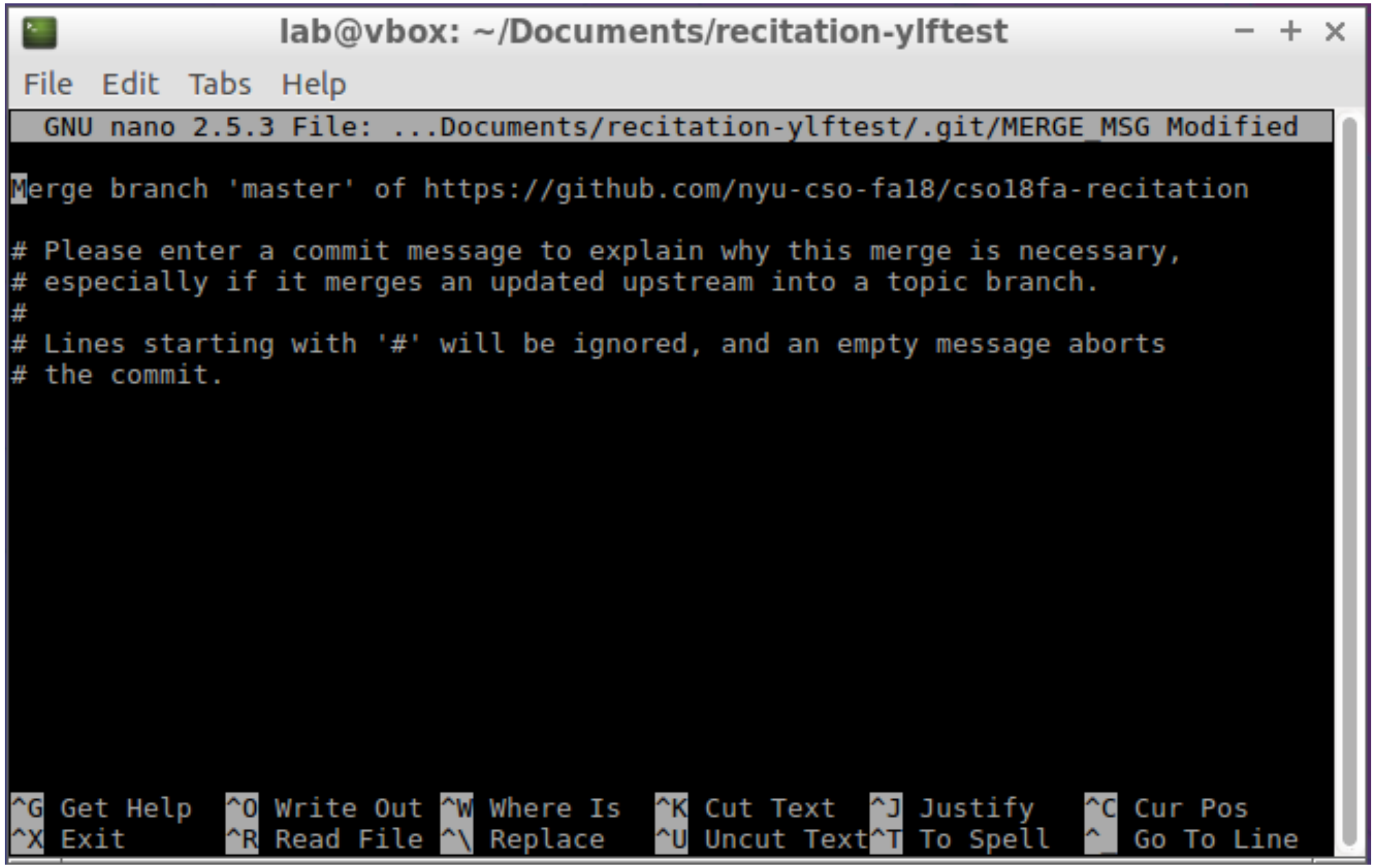
Computer System Organization Recitation [Spring 2019]

CSCI-UA 0201-002

R3: Integer, Float, C Basics

Pull r03 from upstream

- git pull upstream master
- Get a merge message? Hit ctrl + x



Starting from today

- We will do a quick review of lecture contents
  - Quick in-class questions
- Do recitation exercise together
  - You will have several minutes to read and do questions by yourself
  - After that I will release the answer in class

Today's topic

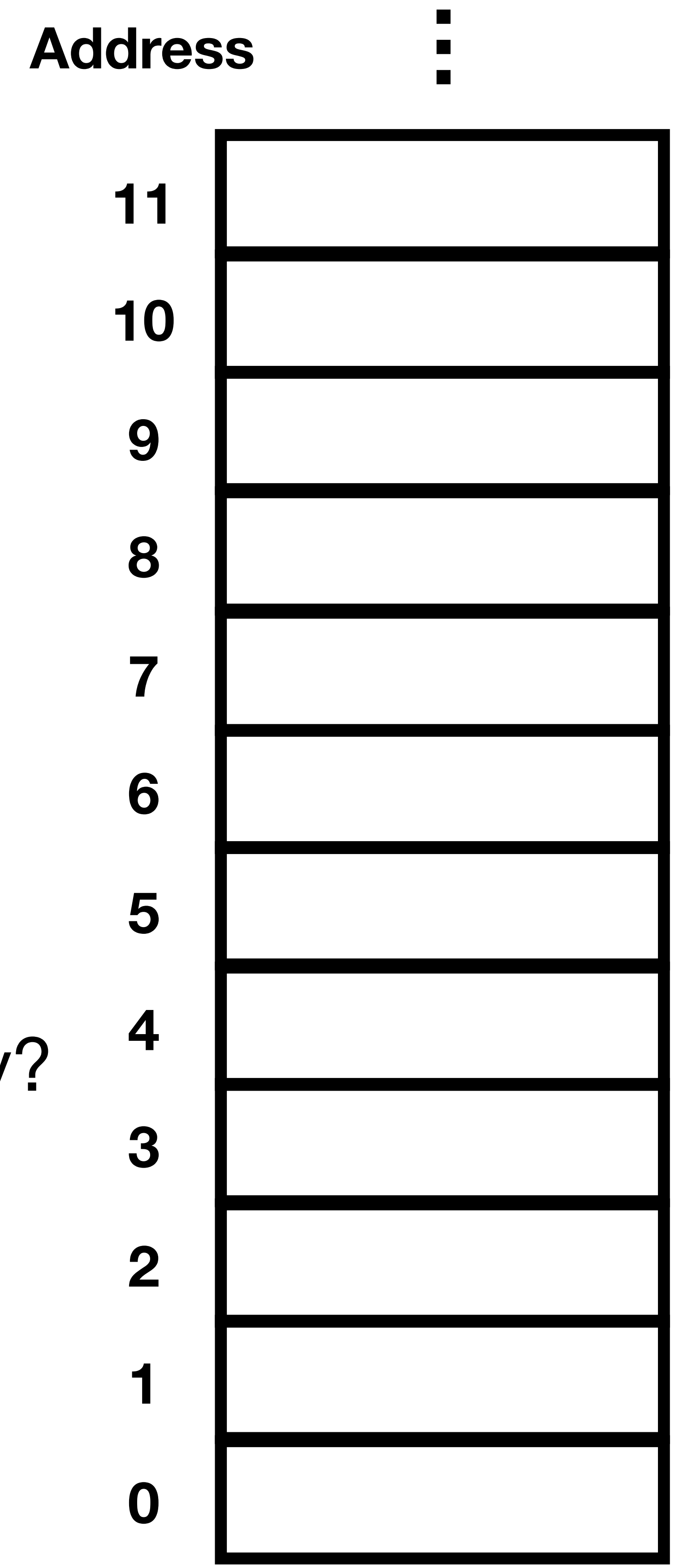
- Memory model
- How to represent integer
- How to represent real numbers
- Bit operators

Bits and Bytes

- Bit
  - smallest storage unit
  - two possible states (0, 1)
  - A typical “8GB” computer memory has 2^36 bits
- Byte
  - 1 byte = 8 bits
  - 1 byte can represent 2^8 = 256 possible states
  - computer memory is addressed at byte granularity

Memory model

- Everything is stored in memory
- Memory is a sequence of cells (logically)
- Each cell is has 8 bits (1 byte)
- Each cell has an address (0-indexed)
  - What's address space of 16MB memory?
  - From 0 to 2^24-1
- Think about everything from memory perspective



Things to memorize

- 1 byte = 8 bits
- common powers of 2: $2^0 = 1$, …, $2^9=512$, $2^{10}=1024$
- 2^{10} B = 1 KB
- 2^{20} B = 1 MB
- 2^{30} B = 1 GB
- 2^{10} is approximately equal to 1000 (10^3)

Quick question

- Which of the following is closest to 1 million?
  - A. 2^10
  - B. 2^20
  - C. 2^30

Representing Non-negative integer

- Base-2 representation

Least Significant Bit (LSB)

$$\boxed{b_{n-1}}b_{n-2}\ldots b_2b_1\boxed{b_0} = \sum_{i=0}^{n-1} b_i * 2^i$$

Most Significant Bit (MSB)

Exercise

- What's the decimal value of the binary representations?

Binary	Decimal	Hexadecimal
0000 0100	2^2 = 4	0x04
0010 0010	2^5+2^1 = 34	0x22
0101 1101	2^6+2^4+2^3+2^2+2^0 = 93	0x5D

Representing signed integers

- 2's complement

$$\vec{b} = \left[b_w, b_{w-1}, \ldots, b_0\right] \qquad val\left(\vec{b}\right) = -b_w 2^w + \sum_{i=0}^{w-1} b_i 2^i$$

- How to tell is whether b is a negative number?
  - b is negative iff MSB b_w is 1

From 2's complement to decimal

- Method 1:
  - Calculate $-b_w 2^w + \sum_{i=0}^{w-1} b_i 2^i$

Example:
1101 1000
=-2^7+2^6+2^4+2^3
=-40

- Method 2:
  - rewrite the above formula

$$-b_w 2^w + \sum_{i=0}^{w-1} b_i 2^i = -2b_w 2^w + b_w 2^w + \sum_{i=0}^{w-1} b_i 2^i$$

$$= \sum_{i=0}^{w} b_i 2^i - b_w 2^{w+1} = \sum_{i=0}^{w} b_i 2^i - b_w 2^n$$

Example:
1101 1000
=2^7+2^6+2^4+2^3-2^8
=216-256=-40

  - So, view the binary as non-negative number, and then subtract $b_w 2^n$, n is number of bits

From negative decimal to 2's complement

- Method 1
  - Convert the absolute value to binary format
  - flip all bits
  - add 1 (don't forget me !!!)

Example:
-40

0010 1000

1101 0111

1101 1000

- Method 2
  - Remember decimal value $D = \sum_{i=0}^{w} b_i 2^i - b_w 2^n$
  - Then $\sum_{i=0}^{w} b_i 2^i = D + b_w 2^n$
  - Add D by 2^n and convert to binary, n is number of bits

Example
-40 + 2^8 = 216
= 1101 1000

Integer types in C

- You need to memorize the size!

64 bits machine

type	size (bytes)	example
(unsigned) char	1	char c = ‘a’
(unsigned) short	2	short s = 12
(unsigned) int	4	int i = 1
(unsigned) long	8	long l = 1

Exercise

- What's the decimal value of the following numbers?

Hex	Binary	char	unsigned char	short	unsigned short
0xB6	1011 0110	-2^7+2^5+ 2^4+2^2+ 2^1=-74	2^7+2^5+ 2^4+2^2+ 2^1=182	182	182
0x7A	0111 1010	2^6+2^5+ 2^4+2^3+ 2^1=122	122	122	122

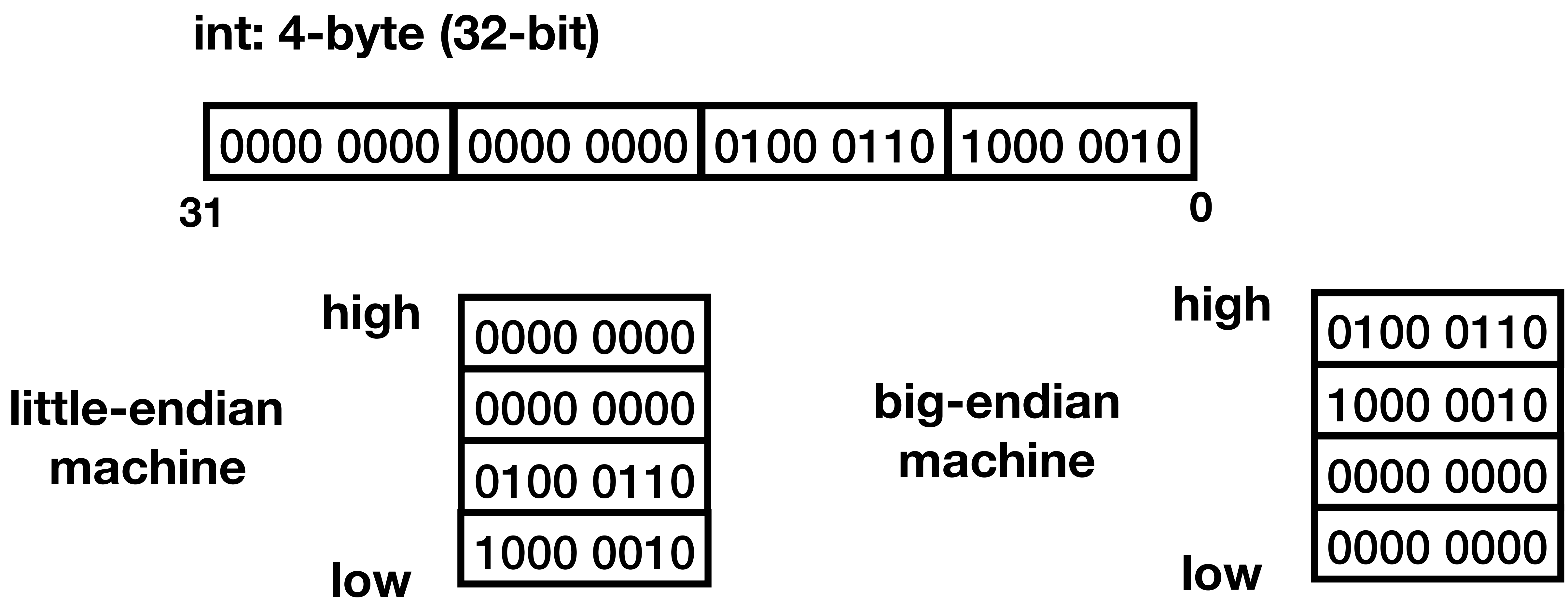
Exercise

- Do it again faster
  - Convert from hex to decimal directly!

Hex	char	unsigned char	short	unsigned short
0xB6	11*16+6-2^8 =-74	11*16+6=182	182	182
0x7A	7*16+10=122	122	122	122

From memory perspective

- How are multi-byte data stored in computer memory?
- Use C int type as an example:



Advantages of little-endian machine

- Overlap computation and memory read
  - Computation usually start from lower bits
  - simultaneously perform memory transfer and calculation

high	0000 0000
	0000 0000
	0100 0110
low	1000 0010

- Faster shortening
  - Casting from int to short?
    - Just read lower two bytes
  - What's the value if cast to char?
  - What's the value if cast to unsigned char?

Unless otherwise noted, we always assume 64-bit little endian machine

Integer overflow

- Assuming one-byte signed integer

```
     1 0 0 0 0 0 0 1        -127
 +   1 0 0 0 0 0 0 1        -127
 ----------------------
   1 0 0 0 0 0 0 1 0        2 ???
```

carry bit discarded

- How to tell if there is an integer overflow
  - pay attention to the data type
  - whenever the computation result goes out of range, it's an overflow
    - for example, 1 + 127 = -128;-128-1=127;…

Representing real numbers

Converting real numbers from decimal to binary

$$r_{10} = (d_m d_{m-1} d_1 d_0 \,.\, d_{-1} d_{-2} \ldots d_{-n})_{10}$$

$$= (b_p b_{p-1} b_1 b_0 \,.\, b_{-1} b_{-2} \ldots b_{-q})_2$$

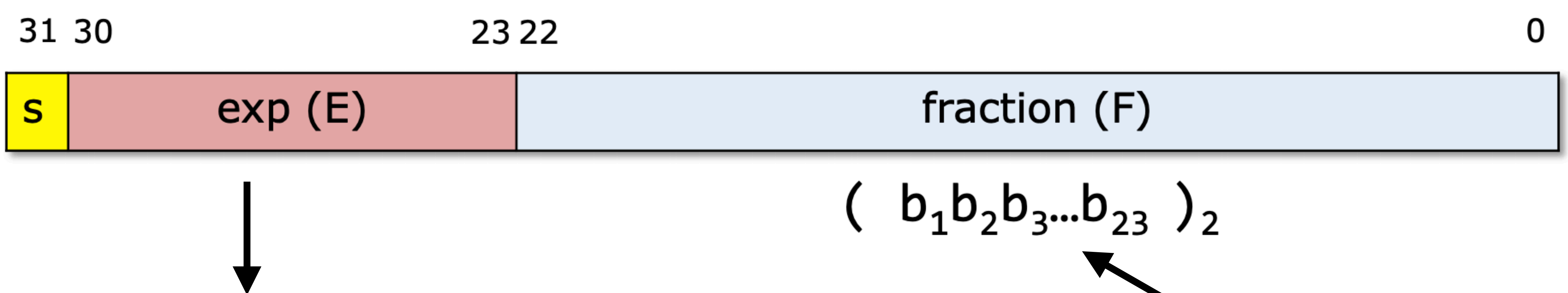
$$\overset{2^p}{b_p}\,\overset{2^{p-1}}{b_{p-1}}\cdots\overset{2}{b_1}\,\overset{1}{b_0}\,.\,\underset{1/2}{b_{-1}}\,\underset{1/4}{b_{-2}}\cdots\underset{1/2^q}{b_{-q}} = \sum_{i=-q}^{p} 2^i \times b_i$$

Single-precision floating point number

$r_{10} = \pm M * 2^{E}$, where $1 <= M < 2$

$M = (1.b_1 b_2 b_3 \ldots b_{23})_2$

M: significant, E: exponent



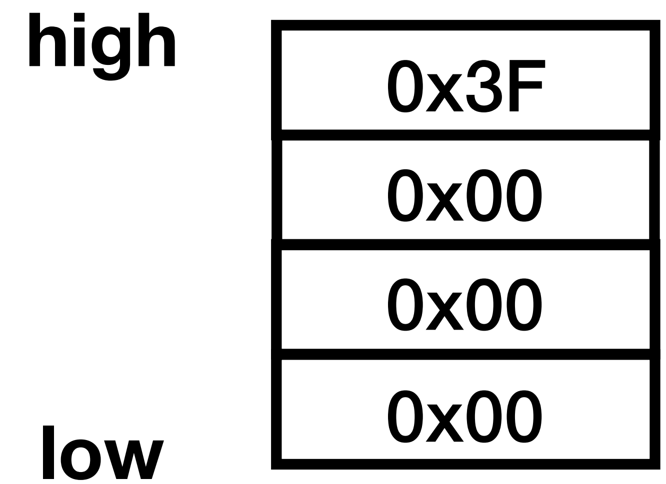
Floating bias 127 for single-precision

Pay attention to the order

Why 127?

Exercise

- If the right-hand side figure is the memory representation of a float number, what's its decimal value?



Little-endian machine (by default)

0x3F000000 = $(0011\ \ 1111\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000\ \ 0000)_2$

S = 0

E = $(011111110)_2$ - 127 = 126 - 127 = -1

F= 0

M = 1.0

Decimal value = +1.0 * 2^{-1} = 0.5

Exercise

- What's hex representation of float number -5.5625?

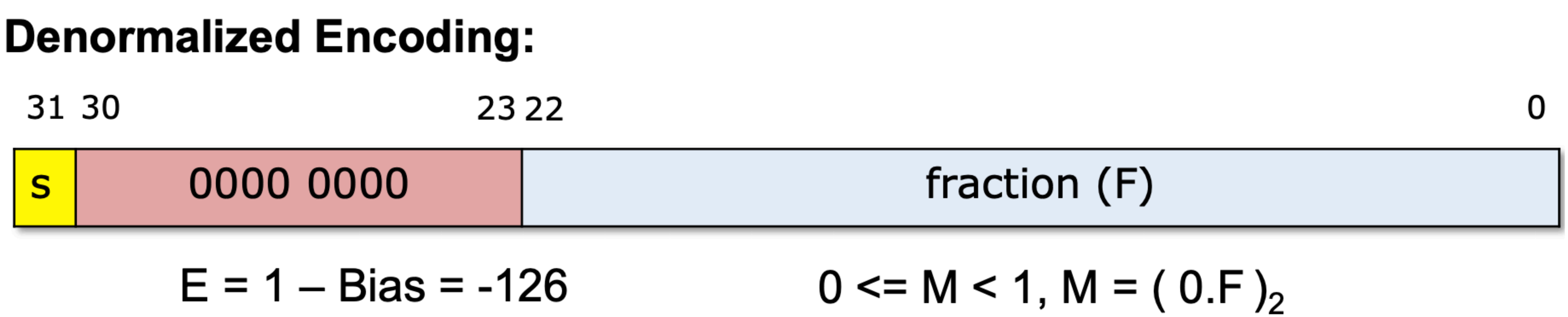
$(-5.5625)_{10} = (-101.1001)_2 = (-1.011001 * 2\text{^}2)_2$

S = 1
E = 2 + 127 = 129 = $(1000\ 0001)_2$
M = 1.011001
F = 011001

$(1100\ \ 0000\ \ 1011\ \ 0010\ \ 0000\ \ 0000\ \ 0000\ \ 0000)_2$
= 0xC0B20000

IEEE denormalized representation

- Why?
  - Represent 0 and values very close to 0
- How?



Zeros

+0

0	0000 0000	0000 0000 0000 0000 0000 000

-0

1	0000 0000	0000 0000 0000 0000 0000 000

+0 = -0

Special Values

Special Value's Encoding:

31	30 … 23	22 … 0
s	1111 1111	fraction (F)

values	sign	frac	Computation Rules
$+\infty$	0	all zeros	$1/{+0} = +\infty$, $3e38+1e38 = +\infty$, $1/\infty = +0$
$-\infty$	1	all zeros	$1/{-0} = -\infty$, $-3e38-1e38 = -\infty$ $1/{-\infty} = -0$
NaN	any	non-zero	sqrt(-1), $\infty\ +\infty$

Other things you need to know

- Double-precision format
  - S (1 bit), E (11 bits), F(52 bits)
- Using floating points are tricky
  - Invalid operation: 0/0, sqrt(-1), $\infty+\infty$
  - Divide by zero: $x/0=\infty$
  - Overflows: result too big to fit
  - Underflows: 0 < result < smallest denormalized value
  - Inexact: rounding!

C bit operators

C operators

- Similar to other languages like Java and Python
  - operations may have different symbols in different language

<table>
<tr><td>Arithmetic</td><td>+, -, *, /, %, ++, --</td></tr>
<tr><td>Relational</td><td>==, !=, >, <, >=, <=</td></tr>
<tr><td>Logical</td><td>&&, ||, !</td></tr>
<tr><td>Bitwise</td><td>&, |, ^, ~, >>, <<</td></tr>
</table>

Bit operators

- Operators that perform bit-wise operations:
  - &: bit-wise and
  - |: bit-wise or
  - ^: bit-wise xor
  - ~: bit-wise negate (flip bits)
  - >>, <<: right and left shift bits
- Why bit operators?
  - Special functionalities
  - Fast! Can usually be done in parallel

How to write multiply by 2?

- Use multiply operator
  - x = x * 2;
  - Complexity: O(n^2), n is number of bits
- Use add operator
  - x = x + x;
  - Complexity: O(n)
- Use bit operator
  - x = x << 1;
  - Complexity: O(1)

Shift operator

- << shift all bits left, and pad 0 at right side
- >> shift bits right
  - Logical shift: always pad 0 at left side
  - Arithmetic shift: pad sign bit (MSB) at left side
- Why do we need arithmetic shift?
  - Semantically, shift means multiply or divide by some power of 2
  - If pad 0, $(-2)_{10} >> 1 = (1111\ 1110)_2 >> 1 = (0111\ 1111)_2 = (127)_{10}$
- Which shift does C use?
  - Unsigned type uses logical shift
  - Signed type uses arithmetic shift

Why can't unsigned type use arithmetic shift as well?

Operator precedence

- What's the result of the following expression?
  - x = 16 >> 3 + 1;
- bit operator has lower precedence than arithmetic operator
  - Use parenthesis whenever you use bit operators!

Lab1

- Mini part:
  - 8 simple C exercises
  - complete and test each exercise individually
  - don’t work in a failure-stop mode!
- Scratch part
  - write stand-alone program from scratch
  - write your own Makefile
  - learn how to use IO to read write files
- Pay attention to coding style (20% of your lab grade)
  - http://cs61.seas.harvard.edu/wiki/2015/Style

Lab1

- Read (updated) lab1 instructions carefully
  - https://nyu-cso.github.io/labs/l1.html
- Lab 1 due 2/25 11pm
- Make sure your code is tested inside VM
- Pay attention to the rules:
  - only modify files you are allow to change
  - check your scratch part output format using the output_format_checker.sh
- Get crazy pointer bugs and segmentation fault?
  - Try GDB first!
  - Come to office hour, we will help you debug for lab 1

Today’s exercise

- go through the readme and e1.c e2.c
- answer questions and complete code