CSO-Recitation 03

CSCI-UA 0201-007

R03: Assessment-01 & Debugging with gdb

Today’s Topics

- Weekly assessment-01
- Breakout exercise in Wed’s lecture
- Debugging with gdb

Assessment 01

Answers and explanations

Assessment-01

- No late submission allowed from assessment-02
- Very few people still didn’t do it, remember to do assessment-02
- Review your grades one day after the due (from assessment-02)

Q1

Facebook has 2.7 billion users. If it is to use an unsigned int as user-id, what's the smallest sized int can it use?

a) 1-byte

b) 2-byte

c) 3-byte

[✓ d) 4-byte]

e) 8-byte

2.7 billion = $2.7*10^9 \approx 10^{3*3} \approx 2^{10*3} = 2^{30}$

# of patterns of n-bits: 2^n

Q2 Signed int

Which of the following signed 1-byte int (in binary format) is the smallest?

a) 00000000

[✓ b) 10000001]

c) 11111111

d) 00000001

e) 10000011

f) 01111110

MSB represents the sign

Two’s complement: bit pattern -> signed int

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

Q3 binary to hex & Q4 overflow(unsigned)

Q3: Convert bit pattern 10111110 to hex notation. You must prefix your answer with 0x.

- 0xbe

1011 -> 2^3+2^1+2^0=11 -> b
1110 -> 2^3+2^2+2^1=14 -> e

Q4: Which of the following 1-byte **unsigned** subtraction operation will overflow?

a) 0xff - 0x0f
[✓ b) 0x0f - 0xff]
[✓ c) 0x01 - 0x0f]
d) 0x0f - 0x01

Size (bytes)	Bit pattern of smallest	Bit pattern of largest	Range
1	0x00	0xff	[0, 2^8-1]
2	0x0000	0xffff	[0, 2^16-1]
4	0x00000000	0xffffffff	[0, 2^32-1]
8	..	..	..

Q5 Overflow (signed)

Which of the following 1-byte **signed** addition operation will overflow?

a) 0xff + 0xfe

b) 0x1f + 0xff

[✓ c) 0x71 + 0x70]

d) 0x05 + 0xfe

[✓ e) 0x80 + 0x8f]

Change it to bit patterns -> addition/subtraction -> see whether

- two positive numbers add to a negative number
- two negative numbers add to a positive number

a) 0xff + 0xfe:

11111111 -> -1

11111110 -> -2

111111101 -> -2^7+2^6+2^5+…+2^2+2^0=-3

c) 0x71 + 0x70:

01110001 -> 2^6+2^5+2^4+2^0

01110000 -> 2^6+2^5+2^4

10000001 -> -2^7+2^0 =-126

Q5 Overflow (signed)

Which of the following 1-byte **signed** addition operation will overflow?

a) 0xff + 0xfe

b) 0x1f + 0xff

[✓ c) 0x71 + 0x70]

d) 0x05 + 0xfe

[✓ e) 0x80 + 0x8f]

Some “special” bit patterns:

- Smallest: 0x80
- Biggest: 0x7f
- 0xff : -1

Q6 Hex to int

If x has bit pattern 0xffffffff, what's the value of x?

[✓ a) -1, if x is signed int]

b) -1, if x is unsigned int

[✓ c) 2^{32}-1, if x is unsigned int]

d) 2^{31}-1, if x is unsigned int

Q7 & Q9 Git

Q7: You've created a new file named quiz.html in your cloned git repository, which command or sequence of commands must you run in order for this file to be saved to github.com?

- git add quiz.html; git commit -m "my commit message"; git push origin master

Q9: Which Git command do you use to check the list of untracked and/or modified files in your repository?

- git status

Q8 Makefile

Which of the following statement about Makefile is wrong?

a) A target is the thing we are trying to build

b) Object files can be either a target and some other target's dependencies

[✓ c) Each rule can only have one command]

d) By default, running the *make* command builds the first target

Q10 Virtual Machine

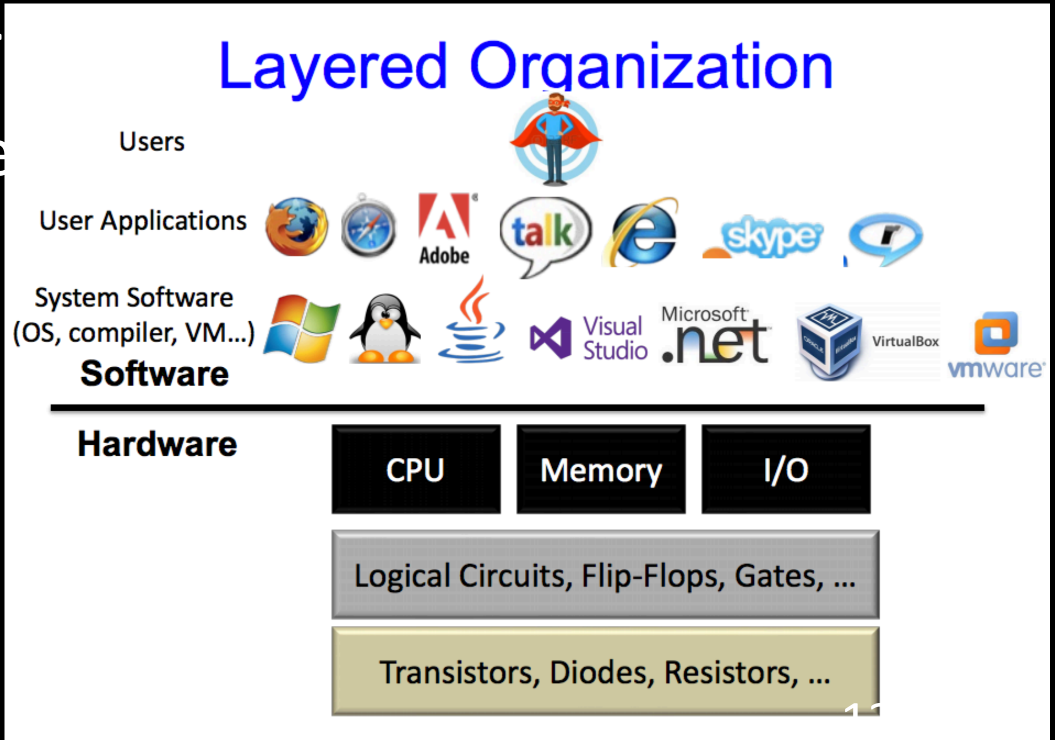
In the layered structure of a Computer, which layer does the Virtual Machine (like the Oracle VirtualBox you're all are using) AND the OS which is installed on it (lubuntu) belong to?

a) User Applications AND System Softwar

[✓ b)] System Software AND System Software

c) CPU AND System Software

d) CPU AND User Applications



Layered Organization

Users

User Applications

System Software
(OS, compiler, VM...)

Software

Hardware

CPU

Memory

I/O

Logical Circuits, Flip-Flops, Gates, ...

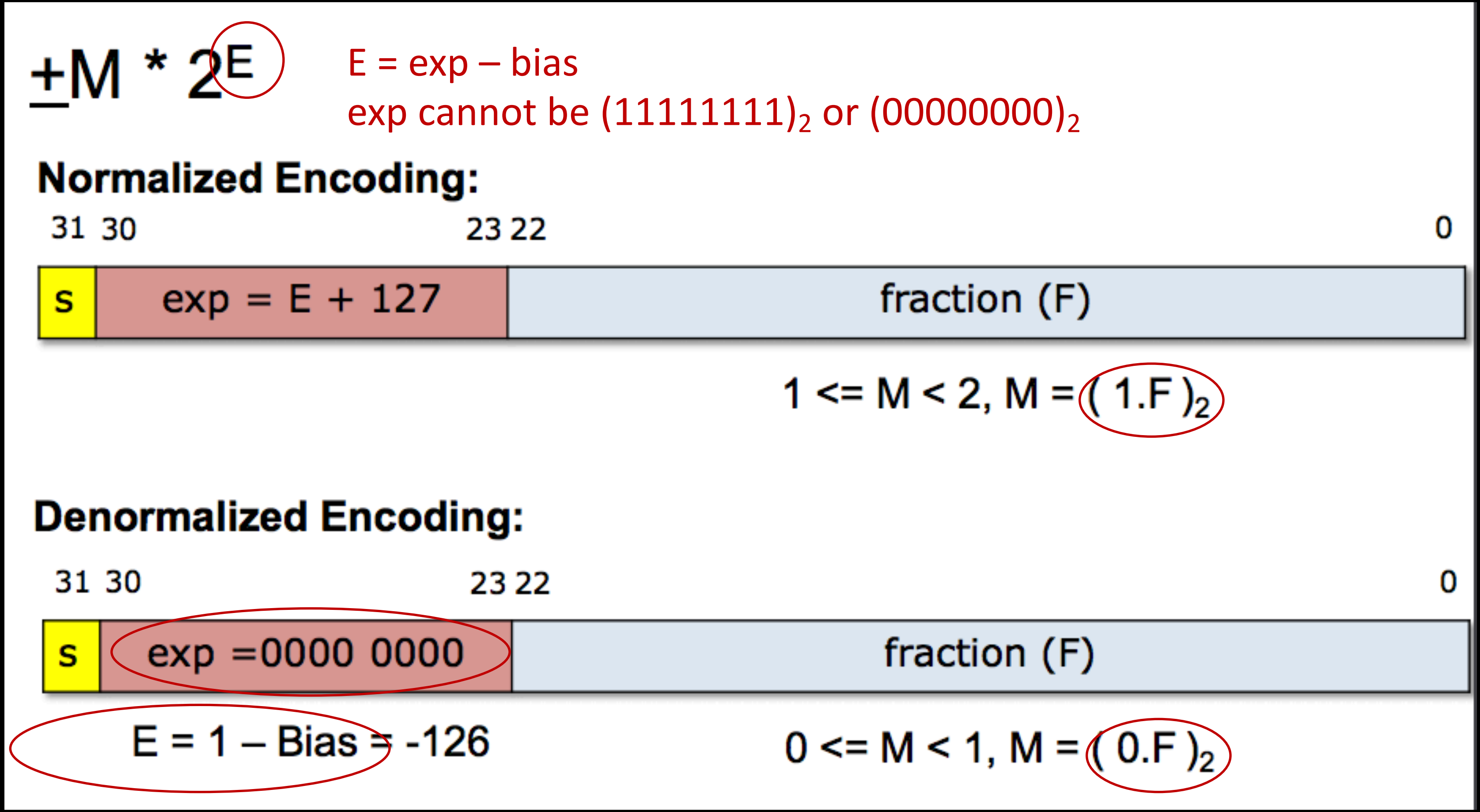
Transistors, Diodes, Resistors, ...

Breakout exercise

Review IEEE FP

- $\pm M * 2^E$
- Normalized Encoding → Using bias in representing E
- Denormalized Encoding → Represent values close and equal to 0
- Zeros
- Special values

Review IEEE FP



$\underline{+}M * 2^{E}$

E = exp – bias

exp cannot be $(11111111)_2$ or $(00000000)_2$

Normalized Encoding:

31 30 23 22 0

s | exp = E + 127 | fraction (F)

$1 <= M < 2$, $M = (1.F)_2$

Denormalized Encoding:

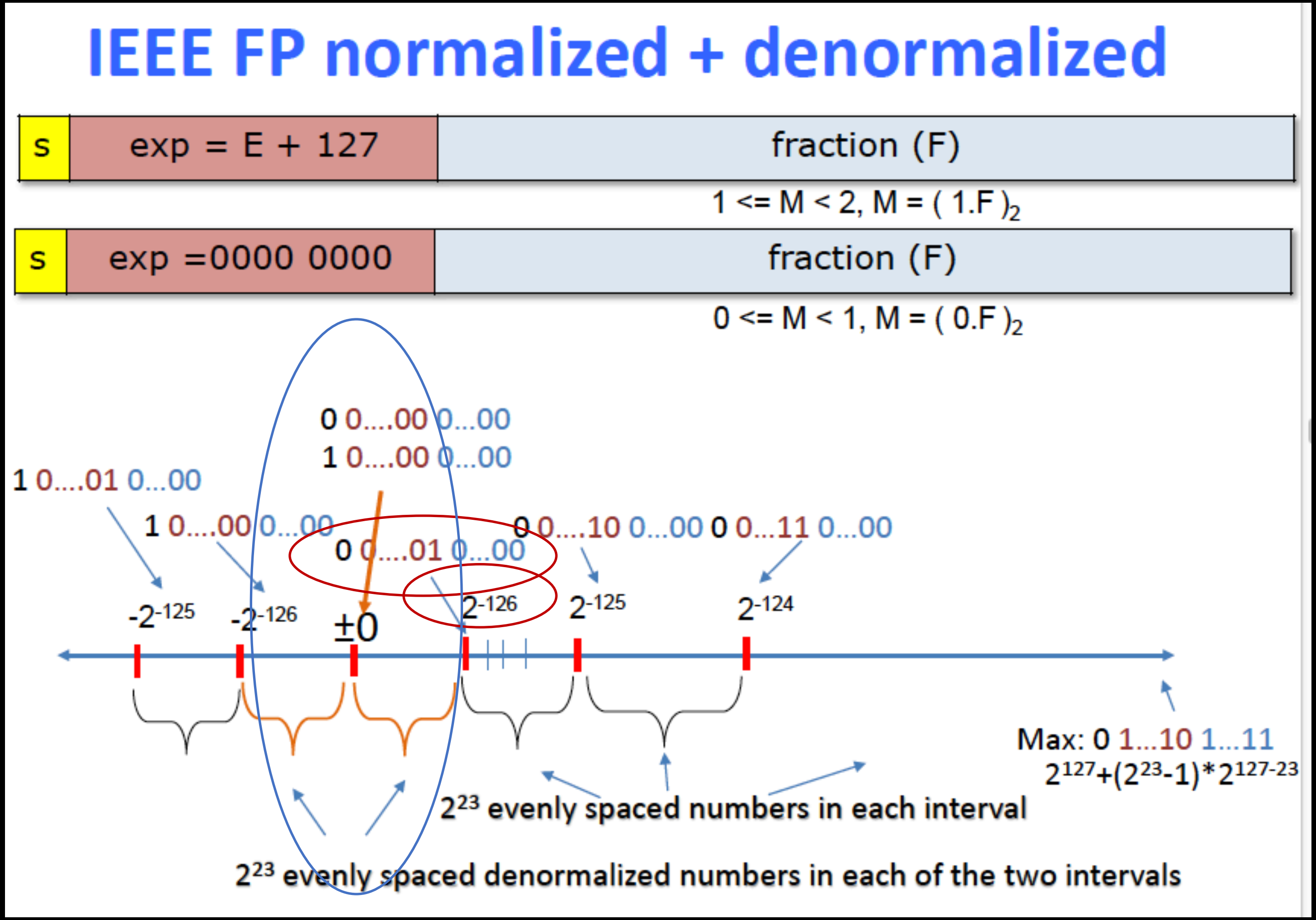
31 30 23 22 0

s | exp =0000 0000 | fraction (F)

E = 1 – Bias = -126

$0 <= M < 1$, $M = (0.F)_2$

Review IEEE FP



IEEE FP normalized + denormalized

s	exp = E + 127	fraction (F)

$1 <= M < 2, M = (1.F)_2$

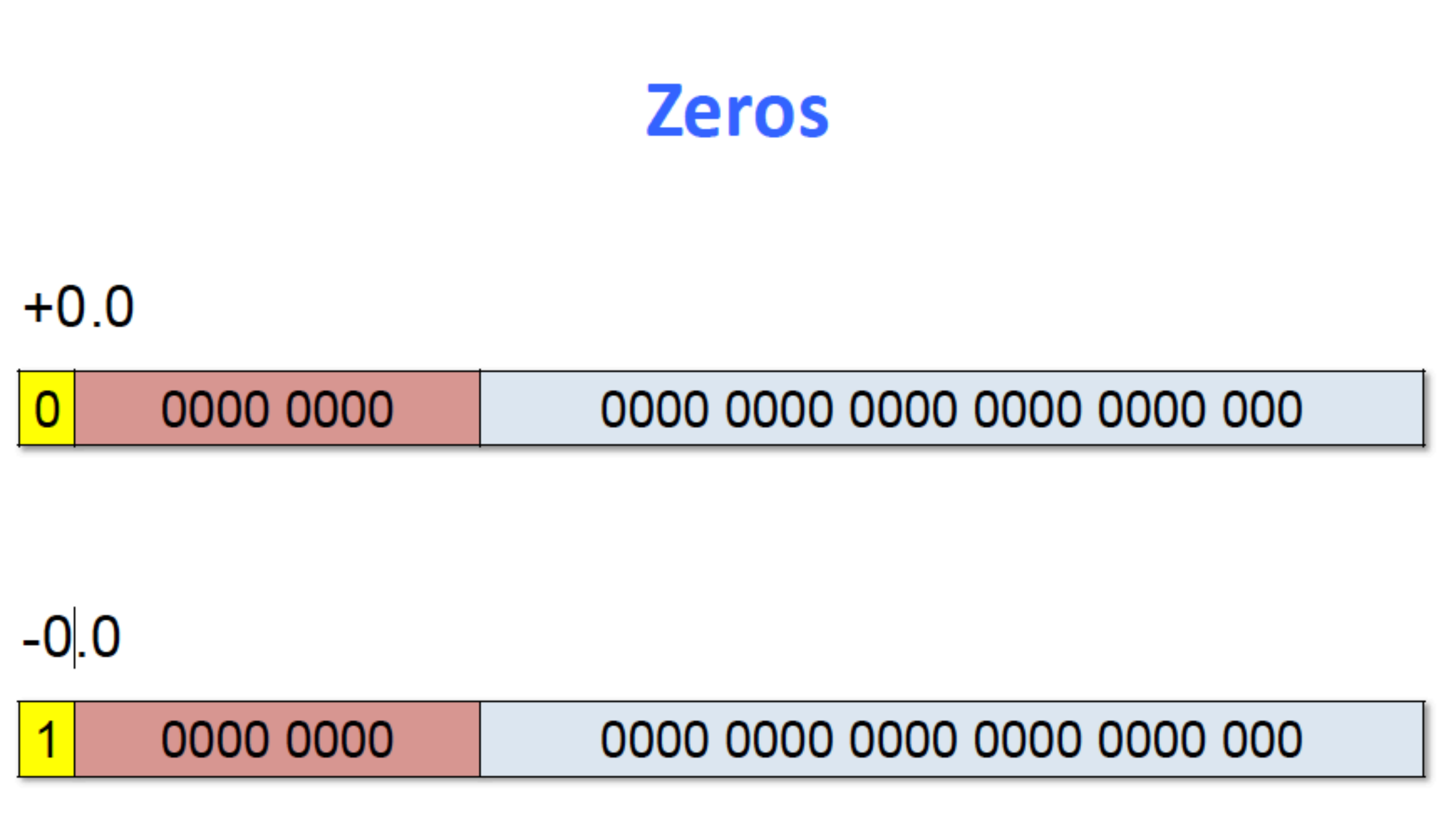
s	exp =0000 0000	fraction (F)

$0 <= M < 1, M = (0.F)_2$

2^{23} evenly spaced numbers in each interval

2^{23} evenly spaced denormalized numbers in each of the two intervals

Review IEEE FP



Zeros

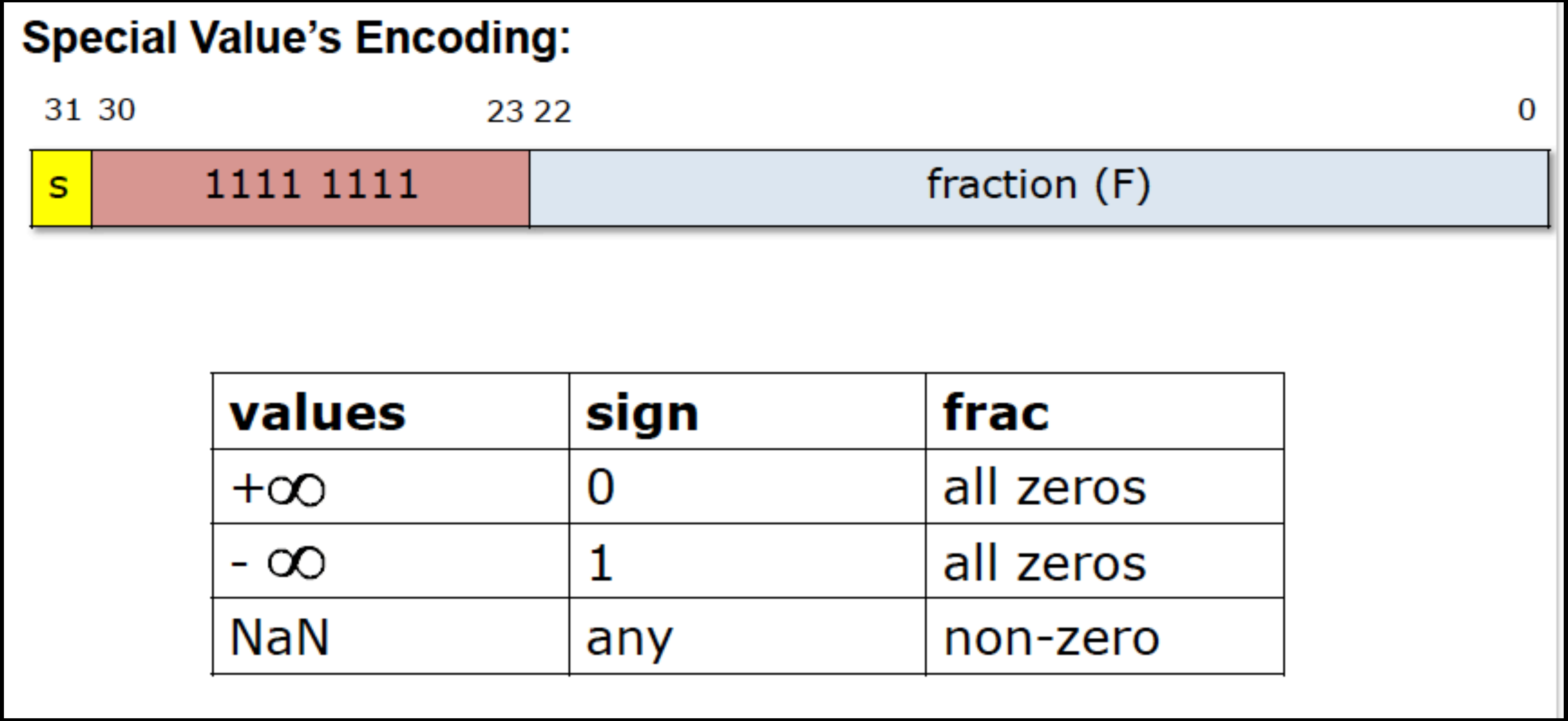
+0.0

0 | 0000 0000 | 0000 0000 0000 0000 0000 000

-0.0

1 | 0000 0000 | 0000 0000 0000 0000 0000 000

Review IEEE FP



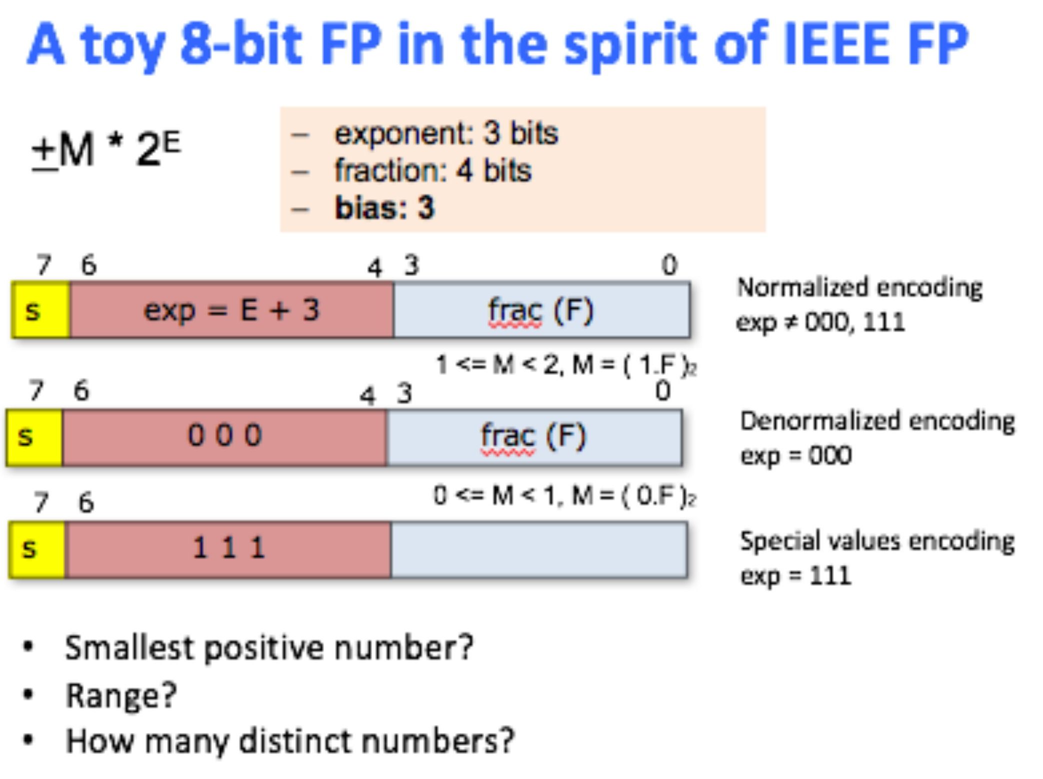
Special Value's Encoding:

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

values	**sign**	**frac**
+∞	0	all zeros
- ∞	1	all zeros
NaN	any	non-zero

Breakout exercise

- Smallest positive number?
  - Denormalized encoding
  - 0 000 0001
  - $E=1\text{-bias} = 1-3 = -2$
  - $M=(0.F)_2=(0.0001)_2 = 2^{-4}$
  - $FP= M*2^E = 2^{-4} * 2^{-2} = 2^{-6}$
- Range? (largest number)
  - Normalized encoding
  - 0 110 1111
  - $E=\text{exp-bias} = (110)_2-3 = 6-3 =3$
  - $M=(1.F)_2=(1.1111)_2 = 2-2^{-4}\ (=2^0+2^{-1}+2^{-2}+2^{-3}+2^{-4})$
  - $FP= M*2^E = (2-2^{-4})*2^3=2^4-2^{-1}=15.5$
  - Range: $[-15.5, 15.5]$ / $[-(2-2^{-4})*2^3, (2-2^{-4})*2^3]$



A toy 8-bit FP in the spirit of IEEE FP

$\pm M * 2^E$

- exponent: 3 bits
- fraction: 4 bits
- **bias: 3**

Normalized encoding
exp ≠ 000, 111

$1 <= M < 2$, $M = (1.F)_2$

Denormalized encoding
exp = 000

$0 <= M < 1$, $M = (0.F)_2$

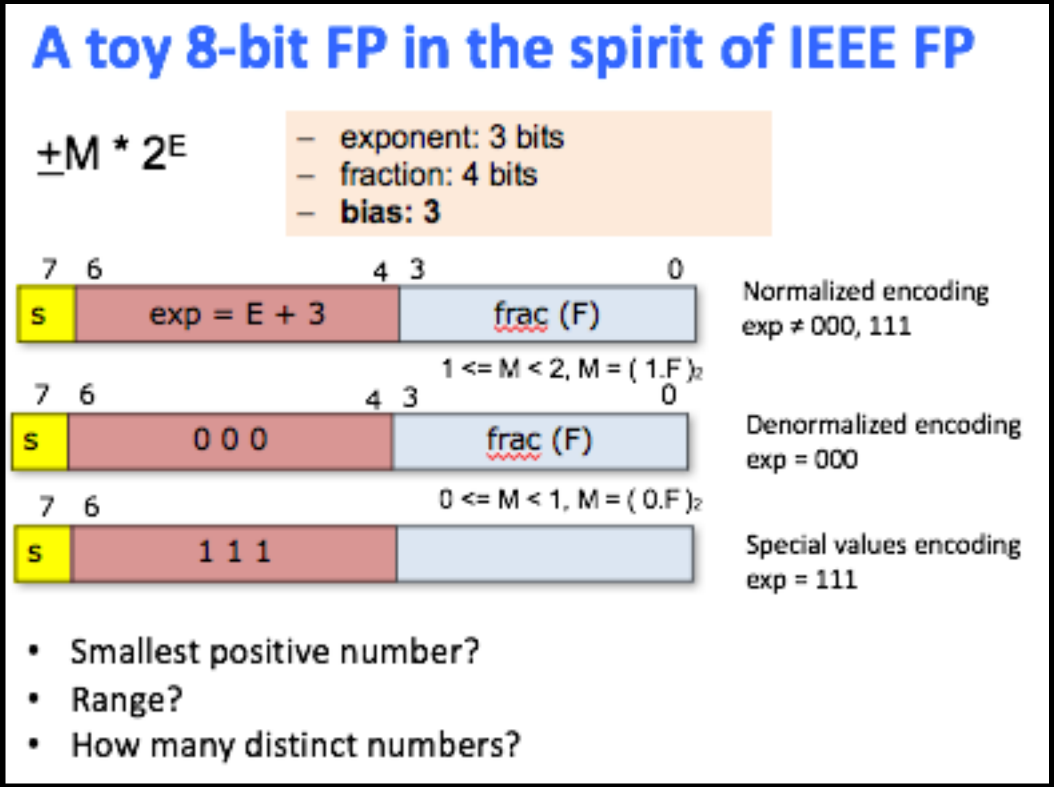
Special values encoding
exp = 111

- Smallest positive number?
- Range?
- How many distinct numbers?

Breakout exercise

- How many distinct numbers?
  - Don’t count the special values
- Method 1:
  - Total bit-patterns: 2^8
  - Special values:
    - X 111 XXXX
    - $2*2^4 = 2^5$
  - 0: 2 bit patterns
  - # distinct numbers = $2^8 – 2^5 -1$
- Method-2:
  - Normalized: $2 * (2^3-2) * 2^4$
  - Denormalized: $2 * 1* 2^4 -1$

$$2^8-2^6+2^5-1 = 2^8-2^5(2-1)-1 = 2^8-2^5-1$$



A toy 8-bit FP in the spirit of IEEE FP

$\pm M * 2^E$

- exponent: 3 bits
- fraction: 4 bits
- **bias: 3**

Normalized encoding
exp ≠ 000, 111

$1 <= M < 2$, $M = (1.F)_2$

Denormalized encoding
exp = 000

$0 <= M < 1$, $M = (0.F)_2$

Special values encoding
exp = 111

- Smallest positive number?
- Range?
- How many distinct numbers?

Getting started with GDB

How to use it and why you should

What is debugging?

- Just because your code compiles doesn’t mean it does what you want
  - It could loop forever, crash, or otherwise just not work correctly
  - Writing tests helps you find out that your code doesn’t work correctly, but you might need more help figuring out why your code doesn’t correctly
- A debugger can help you by providing a number of helpful tools
  - In this class we will use gdb, the GNU debugger

What is debugging?

- GDB lets you
  - Run your program
  - Stop your program at a certain point
  - Print out the values of certain variables at that point
  - Examine what your program is doing
  - Change things within your program to see if it helps

How do you use GDB?

- Add the -g flag when you compile with gcc
  - This flag tells gcc to include debugging information that gdb can use
  - gcc -g main.c -o myprogram
- Run your program with gdb
  - Run gdb ./myprogram
  - You will then be given an interactive shell where you can issue commands to gdb
    - Run your program, look at variables, etc., using the commands
  - To exit the program just type quit (or just q)

Some common gdb commands

- help
  - Gdb provides online documentation. Just typing *help* will give you a list of topics. Or just type *help command* and get information about any other command.

Short Name	Long Name	What do it do?
r	run	Begins executing the program – you can specify arguments after the word run
s	step	Execute the current source line and stop before the next source line, going inside functions and running their code too
n	next	Continue until the next source line, counting called functions as a single line
p	print	Prints the value of an expression or variable
l	list	Prints out source code
q	quit	Exit gdb

step through the program one line at a time

Some more advanced gdb commands

Set the breakpoint at the beginning of the function

Short Name	Long Name	What do it do?
b	break	Sets a breakpoint at a specified location (either a *function* name or *line number*)
c	continue	Continues executing after being stopped by a breakpoint
bt	backtrace	Prints out information on the call stack, i.e. where in the program's execution it is being stopped at
f	frame	Prints information on the current frame / allows you to change frames
i	info	Prints out helpful information (e.g. info args and info locals)

Debugging an infinite loop

- Set a breakpoint inside the loop
  - Or just run it and hit control-c (signal)
- *list* the code
  - This is so you can see the loop condition
- *step* over the code
- Check the values involved in the loop condition
  - Are they changing the right way? Are the variables changing at all?

Debugging a crash

- *run* your program
- Use *bt* to see the call stack
  - You can also use *where* to see where you were last running
- Use *frame* to go to where your code was last running
- Use *list* to see the code that ran
- Check the locals and args to see if they are bad