IEEE Floating Point Protocol

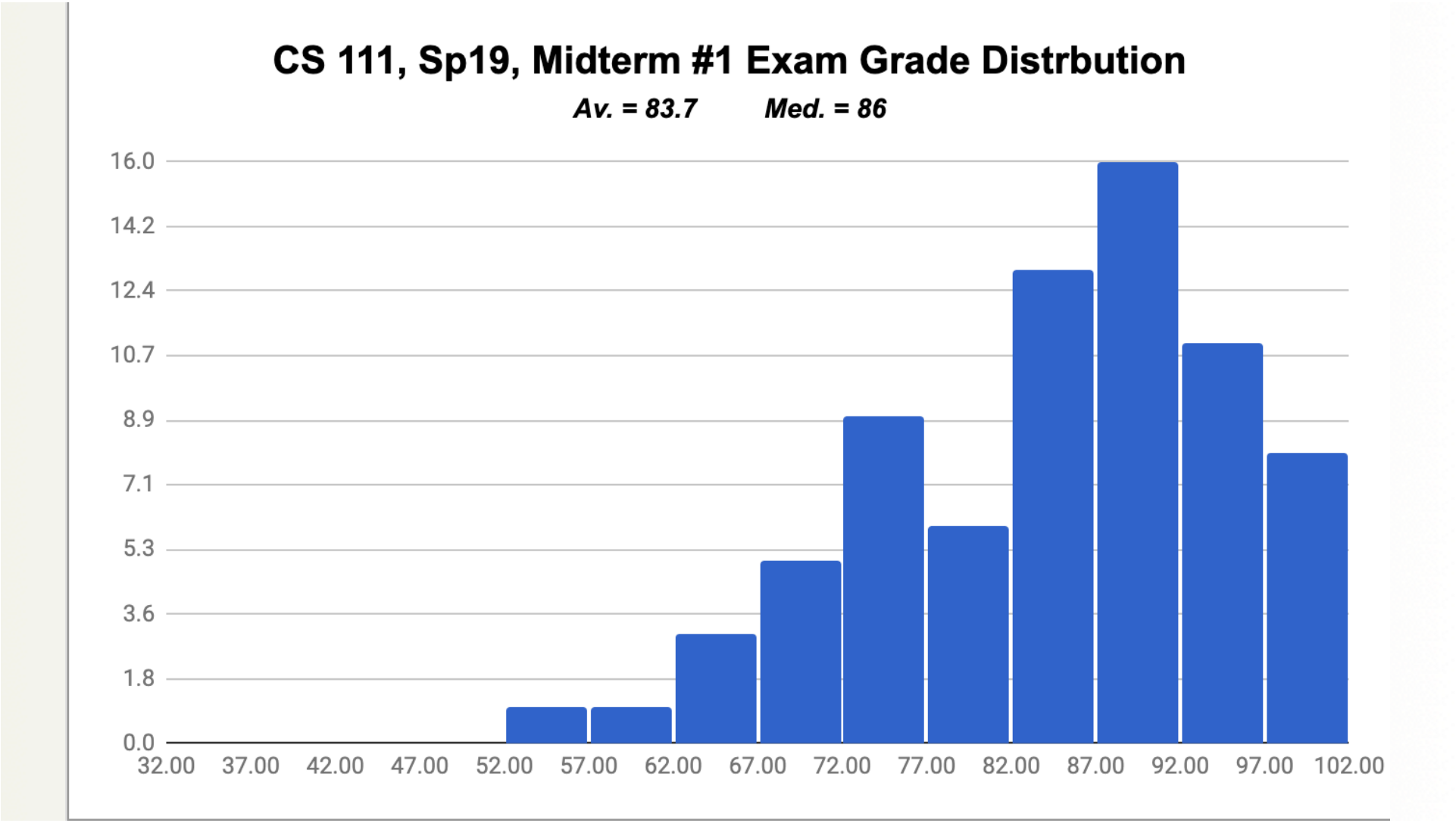
CS 111: Introduction to Computational Science

Spring 2019 Lecture #10

Ziad Matni, Ph.D.

Midterm #1 Exam Results

- Median grade is **86**
- 34% of you got 90% or above
  - Nice!


CS 111, Sp19, Midterm #1 Exam Grade Distrbution
Av. = 83.7
Med. = 86
16.0
14.2
12.4
10.7
8.9
7.1
5.3
3.6
1.8
0.0
32.00
37.00
42.00
47.00
52.00
57.00
62.00
67.00
72.00
77.00
82.00
87.00
92.00
97.00
102.00

Reviewing Your Midterm #1 Exam

- Go to your TA's office hours:
  - Last name is A thru M — See **Steven** (Tu. 1 – 3)
  - Last name N thru Z — See **Shiyu** (Fr. 10 – 12)

- When reviewing your exams:
  - Do not take pictures, do not copy the questions
  - TA cannot change your grade
    - If you have a legitimate case for grade change, the prof. will decide
    - Legitimate case = When we graded, we added the total points wrong
    - Not legitimate case = "Why did you take off *N* points on this question????"

Administrative

- Homework #5
  - Due next week Monday @ 11:59 pm

Number Representation in a Computer

- Consider a 32-bit CPU
- An integer can be represented in 32-bits
  - An *unsigned int* can go from **0** *to* **$+2^{32} - 1$**
  - A *signed int* can go from **-2^{31}** *to* **$+2^{31} - 1$**
    - Signed numbers are represented using "2s complement" method
- Limitations of int representation?
- But how do we represent **floating point**?

***ANS**: Using **scientific notation** IN BINARY!*

Scientific Notation in Base 10

In base-10, expressing 12,450,000 can be written as:

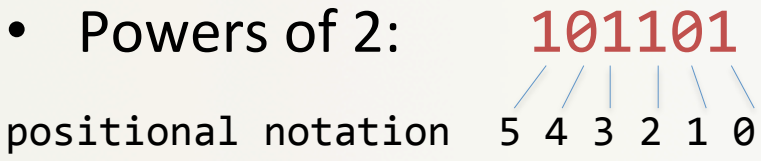
$$1.245 \times 10^{7}$$

- 1245 is called the **mantissa**
- 7 is called the **exponent**
- It's a given that the base is 10 in this example

- BUT what base is used in the machine language of a CPU??!?!

Needle Scratch…

- Ok, a little refresher on binary numbers…
- Powers of 2:



$= 2^5 + 2^3 + 2^2 + 2^0 = 32+8+4+1 = 45$ *(in decimal)*

- Converting decimal *to* binary:

  divide by 2, collect remainders, construct binary

Scientific Notation in Binary (Base 2)

- A number like 29 in binary is:

11101

- So a number like 29.125 (29 + 1/8) is:

11101.001

- OR! **1.1101001×2^4**

Examples of Expressions of Decimal Numbers

- In base 10, expressing one tenth is easy:

10s	1s	1/10	1/100
0	0	1	0

- In base 10, expressing one third is repetitive…

10s	1s	1/10	1/100	1/1000
0	0	**3**	3	3

- In binary (base 2), expressing one tenth is repetitive too:

1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512	1/1024
0	0	0	**1**	**1**	**0**	**0**	1	1	0

repeats

Refresher: Hexadecimals

- Recall: hexadecimal = Base 16
- Convention is to write `0x…` at the start
- Converting from binary to hex is super-easy:
  - How? Collect every 4 bits together (start at LSD)
  - Why? Because $2^4 = 16$

- Example using 16 bits (so, 4 hexs):

```
0001101011110110
0001 1010 1111 0110 = 0x1AF6
```

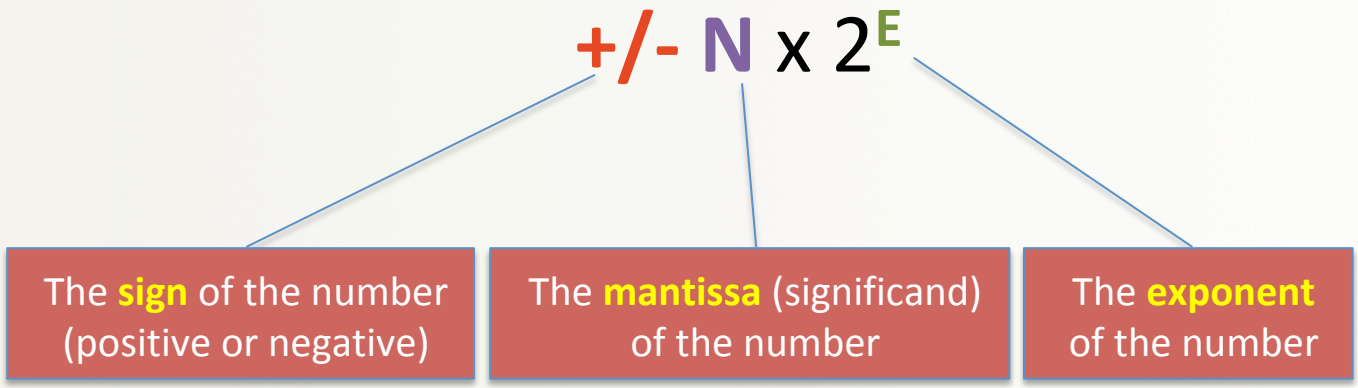
0000	**0**	1000	**8**
0001	**1**	1001	**9**
0010	**2**	1010	**A**
0011	**3**	1011	**B**
0100	**4**	1100	**C**
0101	**5**	1101	**D**
0110	**6**	1110	**E**
0111	**7**	1111	**F**

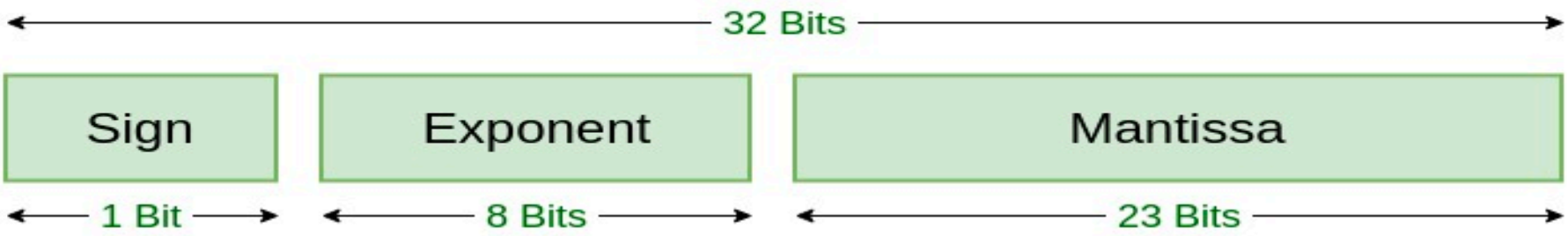
IEEE 754 Floating Point Standard

- For 16-bits
  - Not commonly used for scientific computing
  - Called half-precision FP
- For 32-bits
  - Also called single-precision FP
- For 64-bits
  - Also called double-precision FP
- More bits = more accuracy,
  but also more storage (memory) space, slower computation

For IEEE 754 FP Standard…

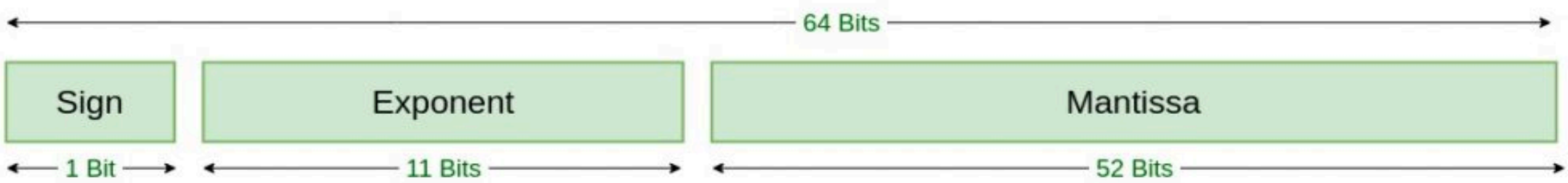
…we need 3 pieces of information to produce a floating point number in this standard:





0 = positive
1 = negative

Single Precision
IEEE 754 Floating-Point Standard



Double Precision
IEEE 754 Floating-Point Standard

The Exponent Field

- This needs to represent **both** positive and negative exponents.
- To do this, a bias is added to the actual exponent in order to get the stored exponent.
  - For IEEE single-precision (32b) floats, this value is **127** (exp. field is 8 bits)
  - For IEEE double precision (64b) floats, this value is **1023** (exp. field has 11 bits)
- To express an exponent of zero, 127 is stored in the exponent field (all 1s).
- A stored value of 200 indicates an exponent of (200−127), or 73

https://steve.hollasch.net/cgindex/coding/ieeefloat.html

The Mantissa Field

- It is assumed that a leading digit of 1 is present in the mantissa
  - Makes sense since w/ binary numbers, the most-significant digit will be a 1
- So, we don't need to store it in the floating-point representation
  - Saves space!
- As a result, a 32-bit floating-point value effectively has 24 bits of mantissa: 23 explicit fraction bits *plus one implicit leading bit of 1*.

https://steve.hollasch.net/cgindex/coding/ieeefloat.html

Example 1

- Recall, the number 29.125 = 1.1101001 x 2^4
- So exponent field has to be 131 (since 131 − 4 = **127**)
- So, a 32-bit representation would be:

sign (1b) *exponent (8b)* *mantissa (23b)*

0 10000011 11010010000000000000000

OR, ordered in 4s (for Hexadecimal version):

0100 0001 1110 1001 0000 0000 0000 0000

0x41E90000

Example 2

- Consider the number 1234.75
- That's: 10011010010.11
- Or: $1.\underline{001101001011} \times 2^{10}$
- So exponent field has to be 137 (since 137 − 10 = **127**)
- So a 32-bit representation is:

0 10001001 00110100101100000000000

OR, ordered in 4s (for Hexadecimal version):

0100 0100 1001 1010 0101 1000 0000 0000

0x449A5800

Python Tie-In

Your To-Dos

- **Homework 5**

</LECTURE>