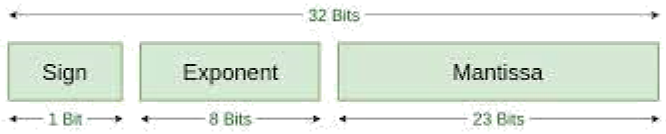


Single Precision
IEEE 754 Floating-Point Standard

Lecture 11:
Floating Point Representation

CS 111: Intro to Computational Science

Spring 2023

Ziad Matni, Ph.D.

Dept. of Computer Science, UCSB

Administrative

- Current homework due today
- New homework out later today

- Quiz 4 on Wednesday
  - Lectures 9 and 10
    - CG, Numerical Stability, LSQ

Floating Point Numbers

Example: -421.987

More generally: -4.21987×10^{2} (called the *normalized form*)

coefficient or *mantissa*

base

sign

exponent

Example of binary number FP: $\mathbf{1.01011 \times 2^{4}}$

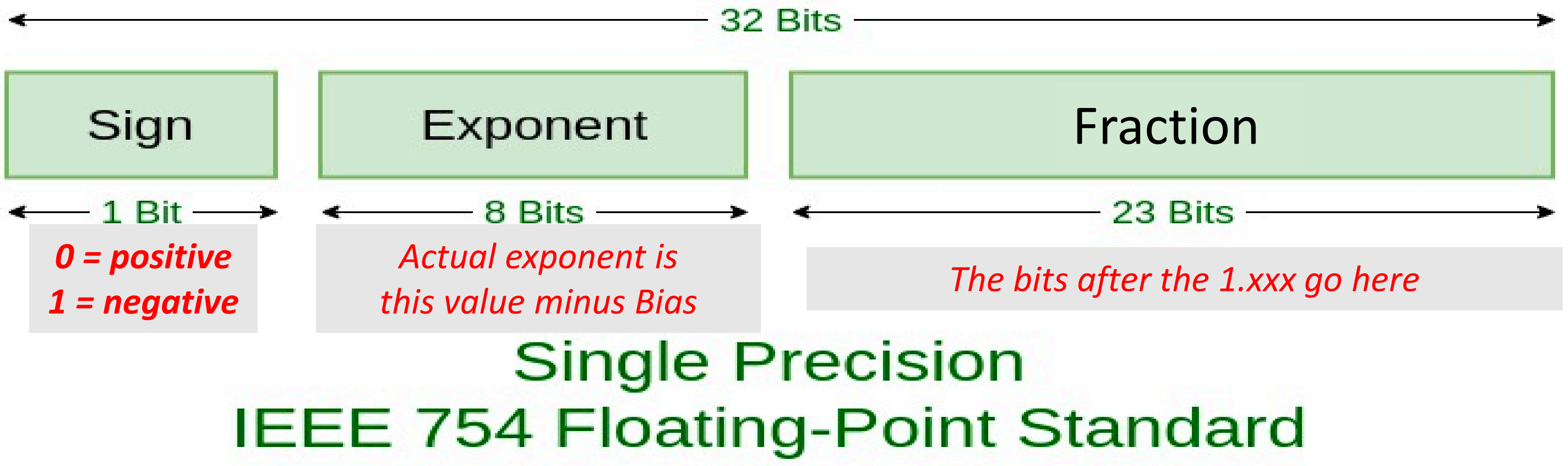
Single Precision FP Variable

type **`float`** in C/C++

Bias = 127

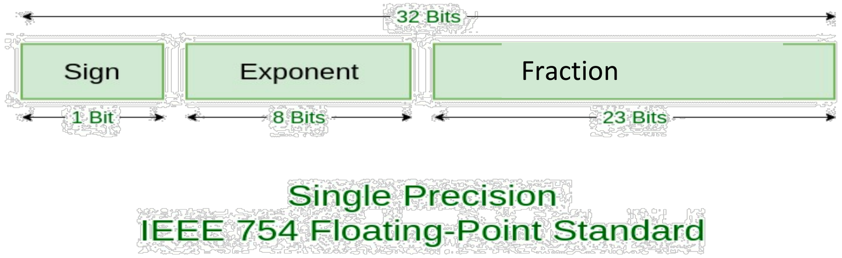
- The **actual** standard form for binary FP is:

$$(-1)^{S} \times (1 + \text{Fraction}) \times 2^{\text{Exponent - Bias}}$$



What is **Bias** for?

Bias = **127**



- The **IEEE 754** standard form is:

$(-1)^S \times (1 + \text{Fraction}) \times 2^{\text{Exponent - Bias}}$

- Take an example, like: S = 0, Fraction = 0, **Exponent = 0**
  - So, $F = 1 \times (1 + 0) \times 2^{-\text{Bias}} = 2^{-\text{Bias}} = 2^{-127}$

- S = 0, Fraction = 0, **Exponent = $2^7 - 1 = 127$**
  - So, $F = 1 \times (1 + 0) \times 2^{127-127} = 2^0 = 1$

- S = 0, Fraction = 0, **Exponent = $2^8 - 1 = 255$**
  - So, $F = 1 \times (1 + 0) \times 2^{255-127} = 2^{128}$

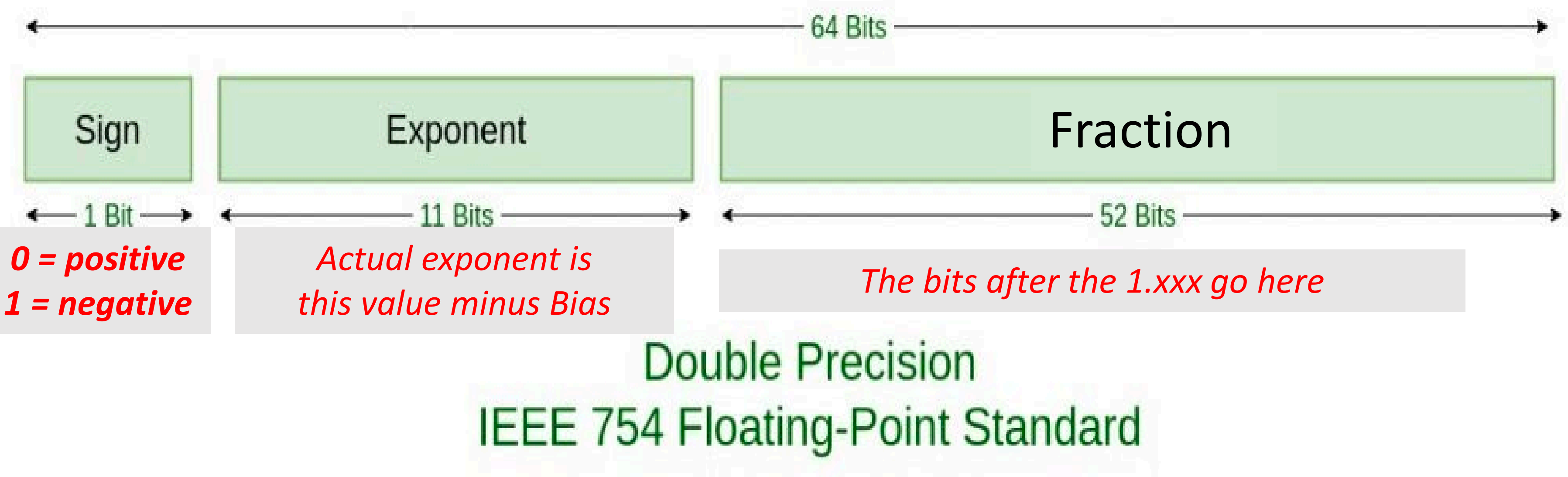
- *So what's the Bias for??!!!*

Single Precision FP Variable

type **double** in C/C++

Bias = **1023**

- Same standard form for binary FP: $(-1)^S \times (1 + \text{Fraction}) \times 2^{\text{Exponent - Bias}}$
- **64** bits instead of 32 bits
- **11** bits for exponent (instead of 8)
- **52** bits for fraction (instead of 23)



0 = positive
1 = negative

Actual exponent is this value minus Bias

The bits after the 1.xxx go here

Other Types?

- IEEE 754 also allows for:
  - Half-precision uses 16 bits
  - Quad uses 128 bits

- IEEE 754 standards are very widespread today
  - Supported in almost all programming languages
  - Used in almost all CPUs

Summarizing **IEEE 754** Floating-Point Standard

$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

	single: 8 bits double: 11 bits	single: 23 bits double: 52 bits
S	Exponent	Fraction

- S = 0 ➔ positive   *S = 1* ➔ *negative*
- The "**1**" in "**1** + Fraction" is ***implicit***
  - i.e. assumed to be part of the calculation (it's not in **S**, **E**, or **F**)
- Fraction written out as: $\mathbf{b_1 b_2 b_3 b_4 b_5 \ldots}$
  - Note that these $\mathbf{b_n}$ represent NEGATIVE powers of 2
  - i.e. $\mathbf{b_1}$ is a position for $\mathbf{2^{-1}} = 0.5$, $\mathbf{b_2}$ for $\mathbf{2^{-2}} = 0.25$, $\mathbf{b_3}$ for $\mathbf{2^{-3}} = 0.125$, *etc…*
- "**Exponent**" is a variable, however "**Bias**" is a **constant** value
  - Single-precision and double-precision have different bias values

Single Precision
IEEE 754 Floating-Point Standard

Example!

$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

- Hex for *single-precision* F-P is given as: **0x3FB00000**
- So:

0011 1111 1011 0000 0000 0000 0000 0000

S = 0 **E** = 0x7F = 127 **F** = 0110…0

- So:

Number $= (+1) \times (1 + 0.011) \times 2^{(127 - 127)}$

$= \mathbf{1.011}$ **(binary number)**

$= 1 + 2^{-2} + 2^{-3}$

$= 1 + 0.25 + 0.125 =$ **1.375**

$2^{-1} = 0.5$
$2^{-2} = 0.25$
$2^{-3} = 0.125$
$2^{-4} = 0.0625$
$2^{-5} = 0.03125$

Single Precision
IEEE 754 Floating-Point Standard

Another Example!!

$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

- Hex word for *single-precision* F-P is: **0xBF300000**
- So:

1011 1111 0011 0000 0000 0000 0000 0000

S = 1 **E** = 0x7E = 126 **F** = 011…0

- So:

Number $= (-\mathbf{1}) \times (1 + 0.\mathbf{011}) \times 2^{(\mathbf{126} - 127)}$

$= -(1 + 2^{-2} + 2^{-3}) \times 2^{-1}$

$= \mathbf{-0.1011\ (bin)}$

$= -(0.5 + 0.125 + 0.0625) =$ **–0.6875**

$2^{-1} = 0.5$
$2^{-2} = 0.25$
$2^{-3} = 0.125$
$2^{-4} = 0.0625$
$2^{-5} = 0.03125$

Example – Class Exercise

$2^{-1} = 0.5$
$2^{-2} = 0.25$
$2^{-3} = 0.125$
$2^{-4} = 0.0625$
$2^{-5} = 0.03125$

- *What is the single-precision FP (in hex) of the number* ***18.125****?*

This is a positive number, so **S = 0**

I am reminded that **0.125** = $\mathbf{2^{-3}}$, i.e. **0.001** in binary

And, I know that **18** in binary is: **10010**

So $\mathbf{18.125}_{(10)}$ = $\mathbf{10010.001}_{(2)}$ = $\mathbf{1.0010001 \times 2^4}$ ***(note how I put that in the 1.xxx format)***

So **F = 0010001000…0** (i.e. everything after the point: remember –> ***23 bits in all***)

And **E = 4 + 127 = 131** = 10000011 (has to be ***8 bits***)

- So: Number in binary is 0 10000011 00100010000000000000000

or 0100 0001 1001 0001 0000 0000 0000 0000 (group every 4b)

= **0x41910000**

A Fly in the Ointment…

$2^{-1} = 0.5$
$2^{-2} = 0.25$
$2^{-3} = 0.125$
$2^{-4} = 0.0625$
$2^{-5} = 0.03125$

- *What is the single-precision FP (in hex) of the number **18.2**?*
  - *What's the problem here?*

- Ok, so **18** in binary is: **10010**
- *But what is **0.2** in binary???*
  - It's not a sum of negative powers of 2…

- It turns out IEEE 754 says **18.2** is **0x4191999A**
  - Which is actually **18.200000762939453125**
  - So there's an error of $7.62939453125 \times 10^{-7}$

- ***Why does this happen?***

Smallest/Largest Values

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
s	exponent								fraction																						
1 bit	8 bits								23 bits																						

Consider Single-Precision Numbers:

- NOTE: Exponent = **0x00** and **0xFF** are reserved values

- Smallest 8bit E is **0x01** ➔ Applying bias, I get the actual exponent = 1 - 127 = –126
- Smallest 23bit Fraction is **0**
- ***This is what gives me the smallest (abs value, non-zero) number in single-precision FP!***

- Largest 8bit E is **0xFE** = 254 ➔ Actual exp. = 127
- Largest 23bit Fraction is 111…11 , which approaches 1 (that's the sum of many $1/2^n$)
- ***This is what gives me the largest (abs value) number in single-precision FP!***

Special IEEE 754 Values: **Zero, Inf**

IEEE 754 allows for special symbols to represent "unusual events"

- When **E = 0x00, F = 0**, this is "zero" (oddly, there's both a -0 and a +0)

- When **S = 0, E = 0xFF, F = 0,** IEEE calls this "***inf***" (i.e. infinity)
  S = 1, E = 0xFF, F = 0 IEEE calls this "***–inf***"

"*inf*" to allow an option for programmers to divide by 0 and not get a CPU interrupt

Special IEEE 754 Values: **NaN**

- IEEE 754 also allows for the result of invalid operations,
  i.e. <u>undetermined numbers</u>

- IEEE calls these "***NaN***" (i.e. "Not a Number")

- Set when **S = 0**, **E = 0xFF**, **F = 1…1** (all 1s)
  - Happens with outcomes of: 0/0 , inf – inf, sqrt(–9), ***etc…***

Consider This Program

```
x = 1.0
while (1.0 + x) != 1.0:
        e = x
        x = x / 2.0
print(e)
```

What is happening?

What happens to var **x**?
What happens to var **e**?

What could this establish for us?

What assumptions are we making about var **x**?

Machine Epsilon (ε)

- Also known as machine precision
  - Is this a small number or a big number??

- The maximum relative error in the rounding procedure in FP
  - That is, $\varepsilon \geq \left|\frac{fl(x) - x}{x}\right|$

- See also: https://en.wikipedia.org/wiki/Machine_epsilon

Quick! To the Python-mobile!

Let's Demonstrate FP Characteristics via Programming!

Your TO DOs!

- Assignment 05 due tonight

- Quiz 4 on Wednesday
  - Lectures 9 and 10
    - CG, Numerical Stability, LSQ

</LECTURE>