Floating Point

C Bitwise Operations...

- We have the boolean operations
  - `||` boolean or
  - `&&` boolean and
- We also have bitwise operations
  - Treat the data as raw bits and apply them on a bit-by-bit basis
  - `|` bitwise or, `0b0011 | 0b0101 = 0b0111`
  - `&` bitwise and, `0b0011 & 0b0101 = 0b0001`
  - `^` bitwise exclusive or, `0b0011 ^ 0b0101 = 0b0110`

And bit shift operations (Example using 5 bit values)

- **`a << b`**: Shift the value in a to the left by b bits, shifting in 0
  - Equivalent to multiplying by 2^b
  - **`0b00101 << 2 = 0b10100`**
  - Bits off the left are just dropped
    - **`0b10010 << 2 = 0b01000`**
- **`a >> b`**: Shift the value in a to the right by b bits
  - If a is signed, we sign extend (copy the MSB)
    - **`0b10100 >> 2 = 0b11101`**
    - **`0b00100 >> 2 = 0b00001`**
  - If a is unsigned, we zero extend
    - **`0b10100 >> 2 = 0b00101`**
  - Not ***quite*** the same as dividing by 2^b due to how rounding works

IEEE-754

- Today, we'll be learning about a standardized format for representing floating point numbers in computers
- IEEE (Institute of Electronics and Electrical Engineers)
  - Standardizes methods for how we do things in computing
- IEEE-754
  - Established in 1985 to standardize how we represent floating point numbers in binary
  - Most recent update was published in 2019

Goals for IEEE 754 Floating-Point Standard

- Standard arithmetic for all computers
  - Important because computer representation of real numbers is approximate. Want same results on all computers.
- Keep as much precision as possible
- Help programmer with errors in real arithmetic
  - +∞, -∞, Not-A-Number (NaN), exponent overflow, exponent underflow, +/- zero
- Keep encoding that is somewhat compatible with two's complement
  - E.g., +0 in Fl. Pt. is 0 in two's complement
  - Make it possible to sort ***without*** needing to do floating-point comparisons

Scientific Notation

- In decimal, we use scientific notation to shorten the number of digits that numbers take up

$$3.0 \times 10^{8} \text{ m/s}$$

$$6.022 \times 10^{23} \text{ mol}^{-1}$$

Scientific Notation (Normalized Form)

Significand/Mantissa

$$9.2318 \times 10^5$$

Exponent

Decimal Point

Base

Representing Fractions in Binary

$$1\ 0\ 1\ 1\ 0.1\ 0\ 1$$

$2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0 \quad 2^{-1} \quad 2^{-2} \quad 2^{-3}$

$$2^4 + 2^2 + 2^1 \quad + \quad 2^{-1}+2^{-3}$$

21 .625

21.625

Binary in Normalized Form

Significand/Mantissa

$$1.0101 \times 2^{6}$$

Exponent

Binary Point

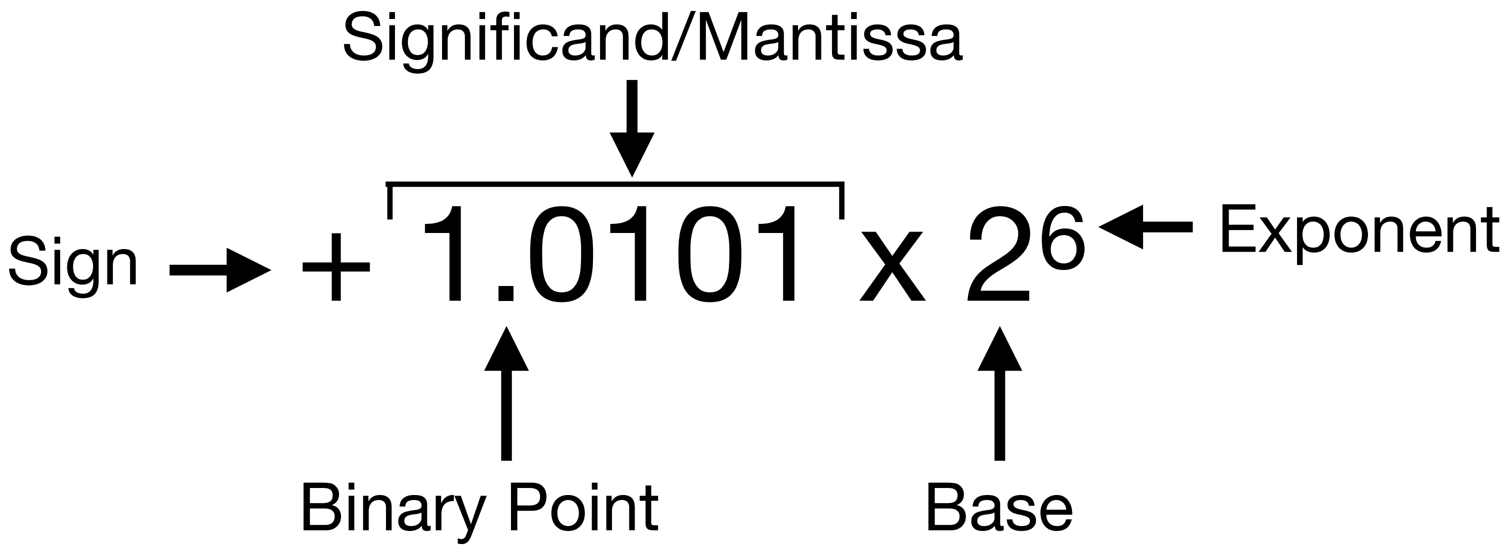
Base

Binary in Normalized Form Example

Convert 0b011010100 to normalized format.

1.10101×2^{7}

Components of Floating Point Numbers



Floating point diagram (32-bit)

Sign

- 0 means positive
- 1 means negative

Mantissa

- In normalized form, there must be one non-zero number to the left of the point
  - In binary, the only non-zero number is 1, which means that any binary number written in normalized format will have a 1 to the left of the point (except 0)
  - We can save room by not storing this 1!
- Pad with zeros to the right

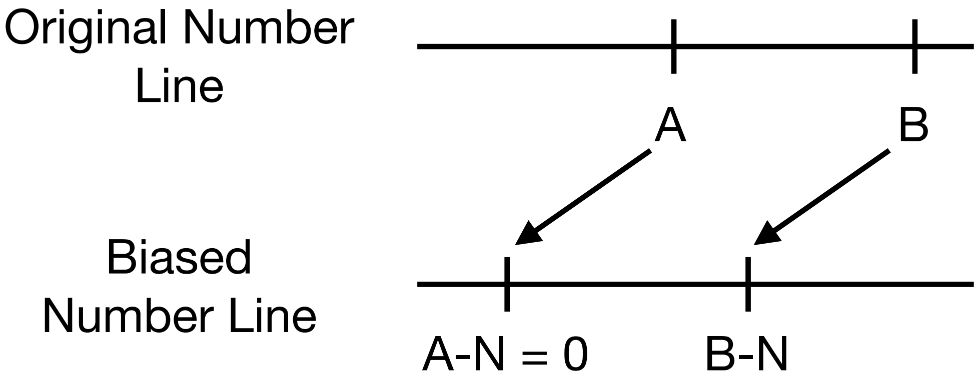
$$1.010110 \times 2^4$$

01011000000000000000000

Exponent

- Exponent is written in biased notation so that the smallest number is written as all zeros.
- The range of the exponent is [-126, 127].
- The exponent is biased by adding 127 to get the number into the range [1, 254]
  - 0 and 255 have special meanings

Exponent
Review of Bias Notation



Confusion over bias notation

- There are different notations with bias encoding
- It's not about memorizing a formula, I just gave one because I know some people prefer that
- It's important to think about the direction in which we are trying to shift the number line
    - If we are trying to shift the number line to the right, then we should be increasing the lower and upper bounds
    - If we are trying to shift the number line to the left, then we should be decreasing the lower and upper bounds

Exponent
Why do we use bias notation?

- Comparison is a common operation (<, >, etc)
- It’s really easy to perform comparisons on biased values because you can just perform an unsigned comparison

Exponent

- Bias formula: $N = -(2^{n-1}-1)$
- For IEEE-754 32-bit floating point numbers, there are 8 exponent bits
  - Bias = $-(2^{8-1}-1) = -127$

Floating Point

$$(-1)^{S} \times 1.\text{mantissa} \times 2^{\text{exponent}-127}$$

Floating Point Examples

Convert the following floating point number to decimal
0b11000000111100000000000000000000

1 | 10000001 | 11100000000000000000000

$(-1)^{S} \times 1.\text{mantissa} \times 2^{\text{exponent}-127}$

$(-1)^{1} \times 1.111 \times 2^{129-127}$

-1.111×2^{2}

-111.1

-7.5

Floating Point Examples

Convert 123.4375 to IEEE-754 32-bit notation

123 = 64 + 32 + 16 + 8 + 2 + 1

0.4375 = 1/4 + 1/8 + 1/16

1111011

0111

1111011.0111

1.1110110111×2^6

Sign = 0

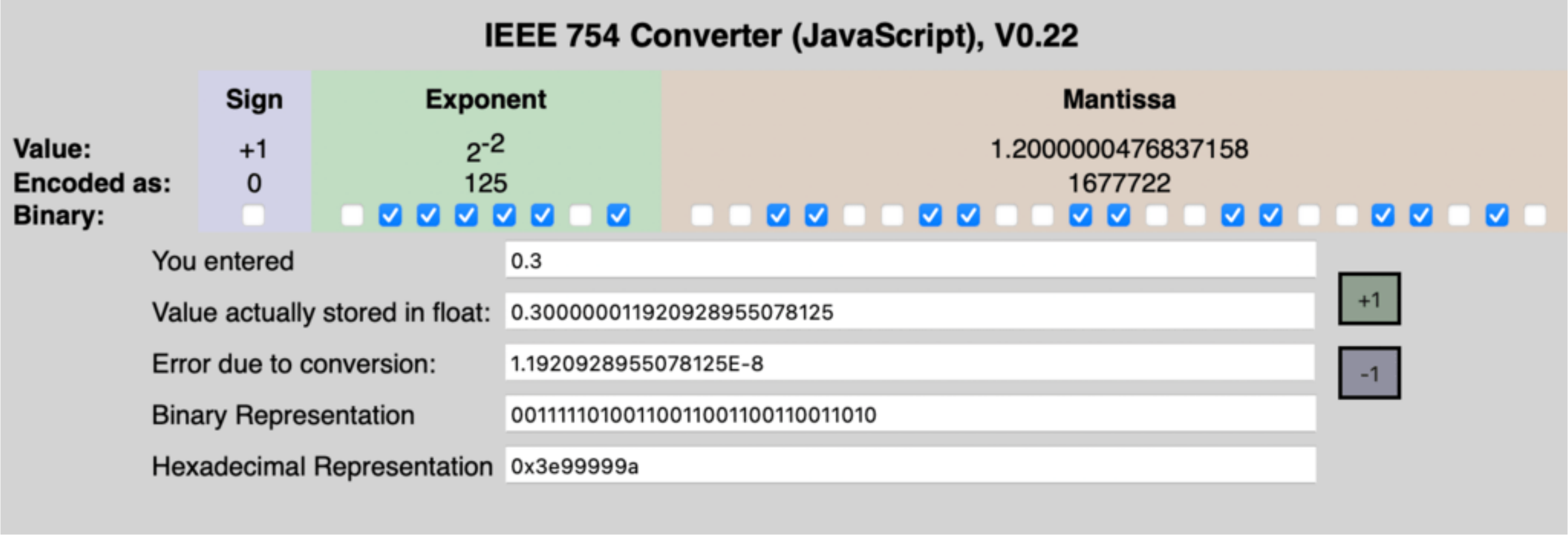
Exponent = 6 + 127 = 133

Mantissa = 1110110111

0 | 10000101 | 11101101110000000000000

Floating Point Tool

- https://www.h-schmidt.net/FloatConverter/IEEE754.html



Rounding

- Rounding can occur
  - During a calculation
  - During conversion
    - Double precision -> single precision value
    - Floating point -> integer

Rounding Modes

- Round to Nearest – round to nearest number; if the number falls midway it is rounded to the nearest value with an even (zero) least significant bit, which means it is rounded up 50% of the time
  - 2.4 -> 2 2.5-> 2
  - -2.6 -> -3 -3.5 -> -4
- Round toward 0 (truncate)
  - 2.001 -> 2
  - -2.999 -> -2
- Round toward +∞
  - 2.001 -> 3
  - -2.999 -> -2
- Round toward –∞
  - 1.999 -> 1
  - -1.001-> -2

How to Represent 0?

- Sign = 0 or 1
- Exponent = all zeros
- Mantissa = all zeros

Floating Point Chart

Type	Exponent	Mantissa
Regular Number	1-254	Anything
Zero	All zeros	All zeros

How to Represent Infinity?

- Sign = 0 or 1 (corresponds to if its positive or negative infinity)
- Exponent = all ones
- Mantissa = all zeros

Floating Point Chart

Type	Exponent	Mantissa
Regular Number	1-254	Anything
Zero	All zeros	All zeros
Infinity	All ones (255)	All zeros

NaN (Not A Number)

- What happens if I take the square root of a negative number or divide by zero?
    - The result not representable or is undefined in computing systems
- Any operation that is not representable or is undefined is encoded as NaN (Not A Number)

What happens to NaN values?

- Usually, NaN values are propagated through arithmetic operations to allow the user to see that some error occurred during the calculation that resulted in a NaN somewhere along the way
- There are a couple of exceptions. We don't cover those in this class

Encoding NaN in IEEE-754

- Sign = 0 or 1
- Exponent = all ones
- Mantissa = nonzero
  - Allows for the definition of multiple distinct NaN values

Floating Point Chart

Type	Exponent	Mantissa
Regular Number	1-254	Anything
Zero	All zeros	All zeros
Infinity	All ones (255)	All zeros
NaN	All ones (255)	Nonzero

Range of Floating Point Values

- What is the smallest positive number that we can represent?

0 | 00000001 | 00000000000000000000000

$$(-1)^{S} \times 1.\text{mantissa} \times 2^{\text{exponent}-127}$$

$$(-1)^{0} \times 1.00000000000000000000000 \times 2^{1-127}$$

$$1 \times 2^{-126}$$

$$2^{-126}$$

Range of Floating Point Values

- What is the largest positive number that we can represent?

0 | 11111110 | 11111111111111111111111

$(-1)^{S} \times 1.\text{mantissa} \times 2^{\text{exponent}-127}$

$(-1)^{0} \times 1.11111111111111111111111 \times 2^{254-127}$

$.11111111111111111111111 = 2^{-1} + 2^{-2} + \ldots + 2^{-23}$

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

$2^{-23}(2^{22} + 2^{21} + \ldots + 1)$

$2^{-23}(2^{23}-1) = 1 - 2^{-23}$

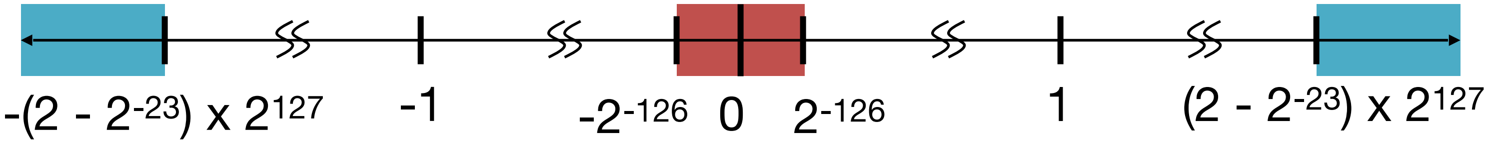
Implicit 1 → $1 + 1 - 2^{-23}$

$2 - 2^{-23}$

$(2 - 2^{-23}) \times 2^{127}$

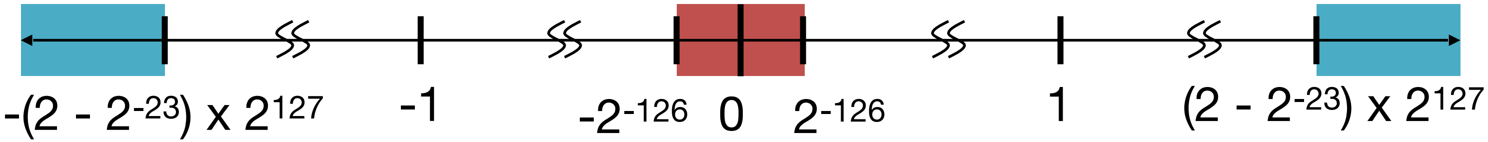
Range of Floating Point Values

- Positive Range
  - $[2^{-126}, (2 - 2^{-23}) \times 2^{127}]$
- Negative Range
  - The only thing that's different is the sign bit, so the range is the same
  - $[-(2 - 2^{-23}) \times 2^{127}, -2^{-126}]$



Range of Floating Point Values

- Overflow = When the magnitude of the value is too large to represent (blue regions)
- Underflow = When the magnitude of the value is too small to represent (red region)



Pause

Floating Point Step Size

- We cannot represent every value between $[2^{-126}, (2 - 2^{-23}) \times 2^{127}]$ because we have a limited number of bits
- There are small gaps in the numbers that we can represent

Floating Point Step Size

$(-1)^S \times 1.\text{mantissa} \times 2^{\text{exponent}-127}$

- What's the next smallest number greater than 2 that we can represent?

2

$(-1)^0 \times 1.0 \times 2^1$

Exponent = 1 + 127 = 128

0 | 10000000 | 00000000000000000000000

0 | 10000000 | 00000000000000000000001

$(-1)^0 \times 1.00000000000000000000001 \times 2^{128-127}$

$(1+2^{-23}) \times 2$

$2+2^{-22}$

Floating Point Step Size

$(-1)^{S} \times 1.\text{mantissa} \times 2^{\text{exponent}-127}$

- What's the next smallest number greater than 4 that we can represent?

4

$(-1)^{0} \times 1.0 \times 2^{2}$

Exponent = 2 + 127 = 129

0 | 10000001 | 00000000000000000000000

0 | 10000001 | 00000000000000000000001

$(-1)^{0} \times 1.00000000000000000000001 \times 2^{129-127}$

$(1+2^{-23}) \times 2^{2}$

$4+2^{-21}$

Floating Point Step Size

$(-1)^S \times 1.\text{mantissa} \times 2^{\text{exponent}-127}$

- If x is the biased exponent and y is the significand
- How do we write our current number in terms of x and y?
  - $(1 + y) * 2^{(x-127)}$
- How do we write the next number in terms of x and y?
  - $(1 + y + 2^{-23}) * 2^{(x-127)}$
- Step-size = next_num - curr_num
  - $(1 + y + 2^{-23}) * 2^{(x-127)} - (1 + y) * 2^{(x-127)}$
  - $2^{-23} * 2^{(x-127)}$
  - $2^{(x-150)}$

Floating Point Step Size

- Step size = $2^{(x-150)}$
- The step size increases by a factor of 2 for every time the exponent increases by 1

Floating Point Step Size

- The gap between 0 and the smallest positive number is 2^{-126}
- What is the gap between the smallest positive number and the next smallest positive number is
  - $2^{(x-150)}$
  - $2^{(1-150)}$
  - 2^{-149}
- There is a larger gap between 0 and the smallest positive number due to the requirement of normalization with an implicit leading one
- Many calculations have values that fall near zero, so let's find a way to represent more values near zero

Floating Point Chart

Type	Exponent	Mantissa
Regular Number	1-254	Anything
Zero	All zeros	All zeros
Infinity	All ones (255)	All zeros
NaN	All ones (255)	Nonzero
???	All zeros	Nonzero

Denormalized Numbers

- Sign
  - Can be positive (0) or negative (1)
- Exponent
  - The exponent field is set to all zeros to encode the denormalized number
- Significand
  - We want to have an implicit leading 0 in order to be able to encode smaller values

Denormalized Numbers

Normalized $(-1)^{S} \times 1.\text{mantissa} \times 2^{\text{exponent}-127}$

Denormalized $(-1)^{S} \times 0.\text{mantissa} \times 2^{-126}$

Exponent = 0 and we need to shift the binary point over by 1 to get an implicit leading 0

Denorm Range

What is the smallest positive denormalized number that can be represented?

0 | 00000000 | 00000000000000000000001

$(-1)^S \times 0.\text{mantissa} \times 2^{-126}$

$(-1)^0 \times 0.00000000000000000000001 \times 2^{-126}$

$2^{-23} \times 2^{-126}$

2^{-149}

What is the largest positive denormalized number that can be represented?

0 | 00000000 | 11111111111111111111111

$(-1)^S \times 0.\text{mantissa} \times 2^{-126}$

$(-1)^0 \times 0.11111111111111111111111 \times 2^{-126}$

$(1-2^{-23}) \times 2^{-126}$

$2^{-126} - 2^{-149}$

Denorm Step Size

$(-1)^{S} \times 0.\text{mantissa} \times 2^{-126}$

- If y is the significand
- How do we write our current number in terms of y?
  - $y * 2^{-126}$
- How do we write the next number in terms of y?
  - $(y + 2^{-23}) * 2^{-126}$
- Step-size = next_num - curr_num
  - $(y + 2^{-23}) * 2^{-126} - y * 2^{-126}$
  - 2^{-149}
- The step size is the same for all denorm values because they all have the same exponent

Floating Point Chart

Type	Exponent	Mantissa
Regular Number	1-254	Anything
Zero	All zeros	All zeros
Infinity	All ones (255)	All zeros
NaN	All ones (255)	Nonzero
Denorm	All zeros	Nonzero

Denorm Examples

Convert the following IEEE-754 floating point number to decimal

1 | 00000000 | 11010000000000000000000

Exponent is 0, mantissa is nonzero => denorm

$(-1)^S \times 0.\text{mantissa} \times 2^{-126}$

$(-1)^1 \times 0.1101 \times 2^{-126}$

1/2 + 1/4 + 1/16 = 0.8125

-0.8125×2^{-126}

-9.55×10^{-39}

Denorm Examples

Write $1.5_{10} \times 2^{-129}$ in IEEE-754 Format

Put in normalized binary form $1.1_2 \times 2^{-129}$ Exponent is too big for normalized

Put in denorm form 0.0011×2^{-126}

$(-1)^S \times 0.\text{mantissa} \times 2^{-126}$

0 | 00000000 | 00110000000000000000000

Floating Point Associativity

- Associativity
  - (X + Y) + Z == X + (Y + Z)
- Because of rounding errors, you can find Big and Small numbers such that:
  - (Small + Big) + Big != Small + (Big + Big)
- Ex: $x = -1.5 \times 10^{38}$, $y = 1.5 \times 10^{38}$, $z = 1.0$

x + (y + z)	(x + y) + z
$-1.5 \times 10^{38} + (1.5 \times 10^{38} + 1.0)$	$(-1.5 \times 10^{38} + 1.5 \times 10^{38}) + 1.0)$
$-1.5 \times 10^{38} + (1.5 \times 10^{38})$	$0 + 1.0$
0	1.0

Floating Point Addition is not associative!

Other Floating Point Notations

- There are other floating point notations that exist to optimize for speed, precision, and/or accuracy

Type	Sign	Exponent	Significand field	Total bits	Exponent bias	Bits precision	Number of decimal digits
Half (IEEE 754-2008)	1	5	10	16	15	11	~3.3
Single	1	8	23	32	127	24	~7.2
Double	1	11	52	64	1023	53	~15.9
x86 extended precision	1	15	64	80	16383	64	~19.2
Quad	1	15	112	128	16383	113	~34.0

https://en.wikipedia.org/wiki/Floating-point_arithmetic

(There are a lot more than this, these are just the basic ones)