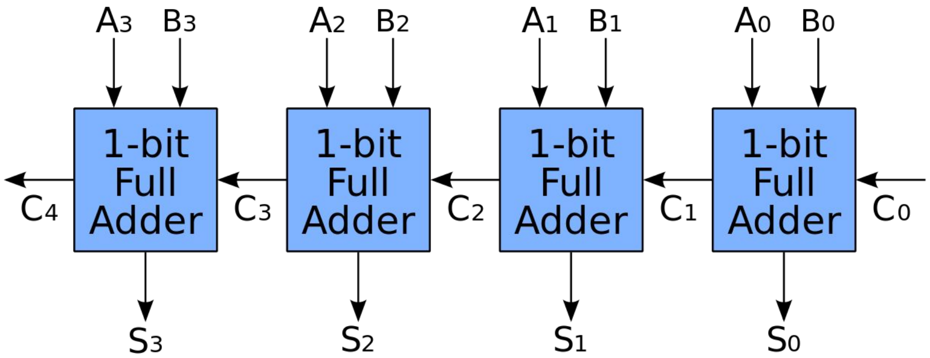
MONTANA
STATE UNIVERSITY

Floating Point

Representing Decimals Efficiently

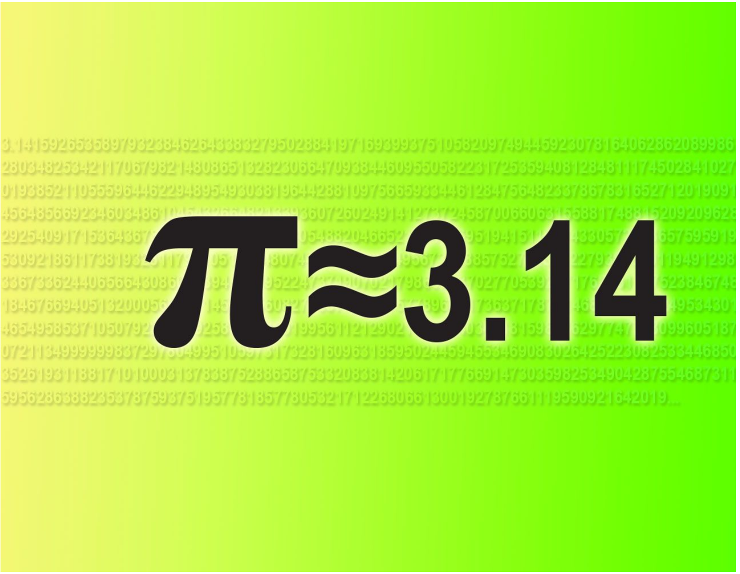
Last Lecture

- In the last lecture we looked at
  - Binary representations for *integer values*
  - We looked at hex and character representations
  - Now we are going to consider *decimal values*



Real Numbers

- Representing non-integer values
    - E.g. pi = 3.1415…
- One obvious mechanism: dedicate a certain number of bits to the right hand side of the decimal
    - This is known as *fixed point*



Fixed Point

- Advantages
  - Simple
  - Can use the same mathematical circuitry as integers
- Disadvantages
  - Can only represent a small domain or number OR support a small number of decimal places
- Fixed point is still used for specialized computations
  - E.g. finance

0003.1415

1010.1001

Floating Point

- Realization
  - What if we allowed the point to float?
  - Dedicate a variable number of bits to the left hand and right hand side of the decimal point?

$$1.2345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10}_{\text{base}}^{\overbrace{-4}^{\text{exponent}}} .$$

12345,4

Floating Point

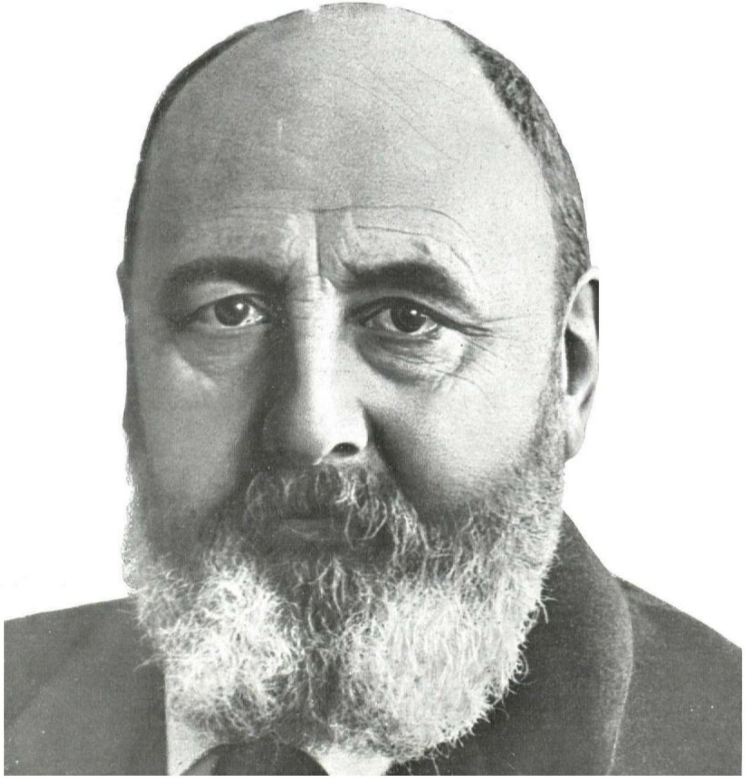
- Advantages
  - Far greater range of numbers can be represented
  - No wasted leading or trailing 0 bits
- Disadvantages
  - More complex
  - Different circuitry for mathematical operations

$$1.2345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10}_{\text{base}}^{\overbrace{-4}^{\text{exponent}}} .$$

12345,4

Floating Point History

- First known use of floating point was in a electro-mechanical computing machine designed by Leonardo Torres y Quevedo
    - Spanish engineer
    - Designed first computer game (chess) and was a pioneer in remote control
        - *How come I've never heard of him? Good question.*



Floating Point History

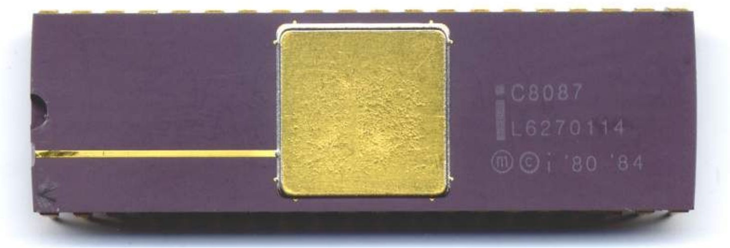
- Konrad Zuse
  - German engineer
  - Designed the world's first programmable computer, the Z1
    - Included 24-bit binary floating point
  - Also developed the first high-level programming language, Plankalkül
    - *How come I've never heard of him? Another good question.*

Floating Point History

- 1950s to 1980s
  - Many competing floating point standards
    - IBM 7*
    - UNIVAC
    - Etc.
- IEEE 754 - 1985
  - Established a floating point standard for the industry
    - Intel
    - Motorola

Floating Point History

- Intel 8087
  - The first x87 floating-point coprocessor for the 8086 line of microprocessors
  - 20% to 500% faster for many operations
- Eventually this functionality was folded into the main CPU with the advent of the 486 chip



Floating Point History

- William Kahan
  - Primary architect behind the IEEE 754 standard
  - The “Father of Floating Point”
  - Wrote the program *paranoia* to test floating point implementations
    - Found the floating point bug in pentium division
  - Won the Turing award for his contributions

Floating Point Today

- X86-64
  - Includes registers for floating point values
  - 128 bits (!!!)
  - XMM0-XMM7 (part of x86-32 SSE)
  - XMM8-15 (available in 64-bit mode only)

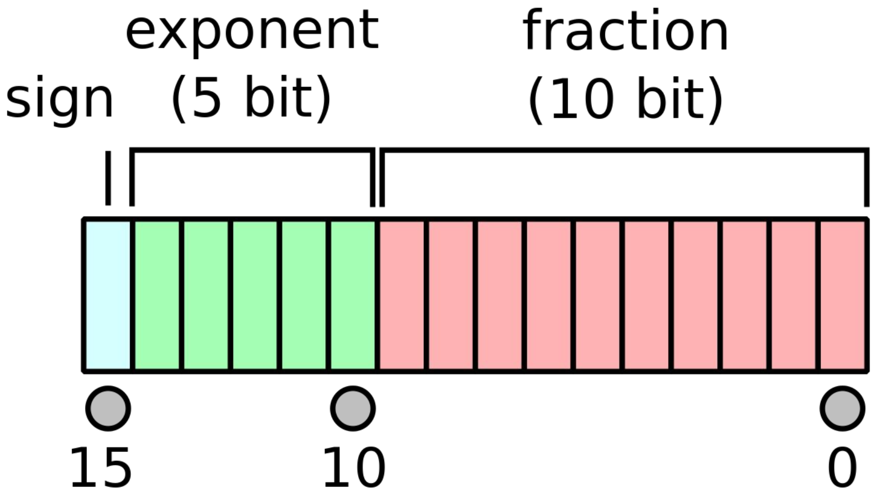
Floating Point Details

- IEEE 754 Floating Point Representation
  - Various levels of precision
    - Half - 16 bits
    - Single - 32 bits
    - Double - 64 bits
    - Extended - 80 bits
    - Quad - 128 bits

$$1.2345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10}_{\text{base}}^{\overbrace{-4}^{\text{exponent}}} .$$

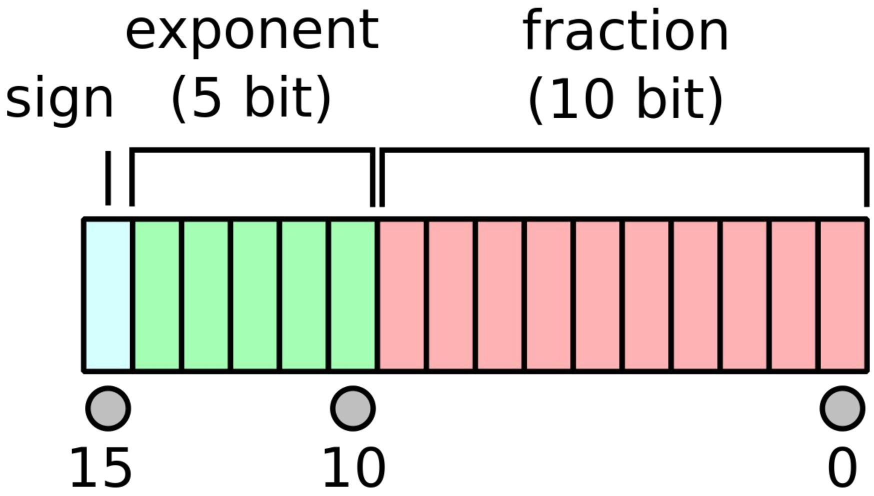
Floating Point Details

- IEEE 754 Half
  - A total of 16 bits
    - 1 *sign bit*
    - 5 *exponent bits*
    - 10 *significand* (fraction) bits
- Sign bit is obvious
    - 0 positive
    - 1 negative



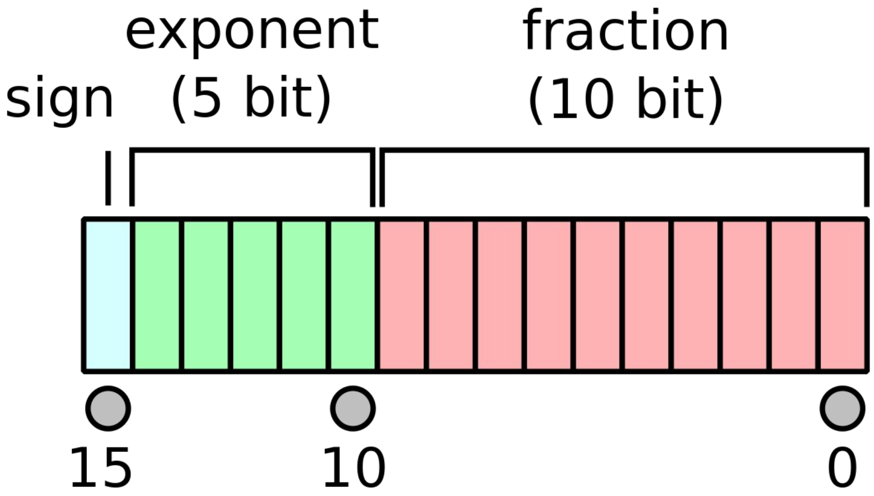
Exponent

- Exponent is a 5 bit value, giving 2^5 = 32 different possibilities
    - Because the exponent is unsigned and we wish to express negative values it is *biased* at 15
        - The unsigned value has 15 subtracted from it to get the actual exponent value
        - This allows values of 15 to -14 for the exponent



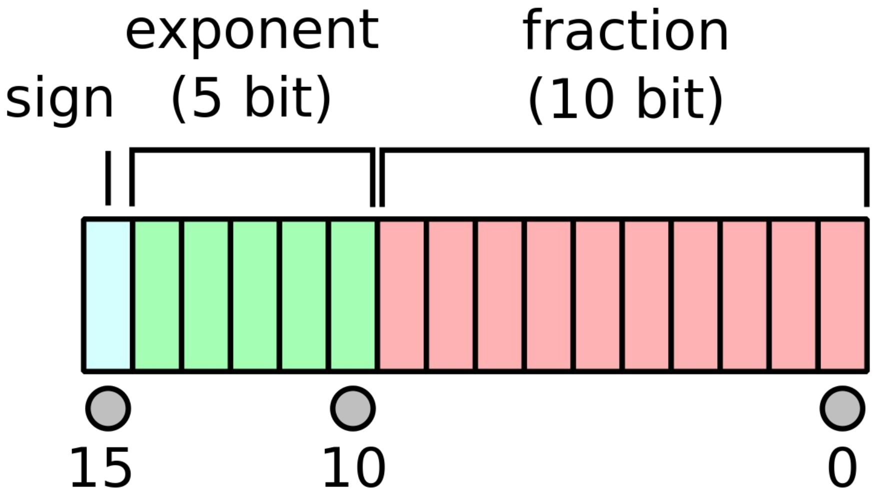
Exponent

- IEEE 754 Half
  - What about exponent value 00000?
    - Typically means 0
    - Can also mean subnormal numbers
      - Very small numbers below the normal floating point range
  - What about the exponent value 11111?
    - Means infinity or NaN, depending on significand



Significand (Fraction)

- Fraction is 10 bits
- Gives us 2^10 (1024) possible values
  - Values are expressed in terms of x/1024
    - E.g 0000000001 → 1/1024th
- Significand value is added to 1 to get a number somewhere between 1 and 2
  - Except subnormal case, when it 0 is added to it



Float Value Calculation

- Consider this 16 bit floating point number
- Sign bit: 0 (positive)
- Exponent 1 → 2^(1-15) = 2^-14
- Fraction = 0000000000 → (1 + 0/1024)

$0\ 00001\ 0000000000_2$

$$2^{-14} \times (1 + \frac{0}{1024})$$

0.000061035156

Binary Fractions

- We are used to decimal notation
- What does 1.2345 mean in terms of fractions?
  - 1 + 2,245/10,000
- What does 1.01011 mean in binary?
  - 1 + (01011/100000) *binary*
  - 1 + (11/32) *decimal*
  - 1 + .34375 *decimal*
  - 1.34375

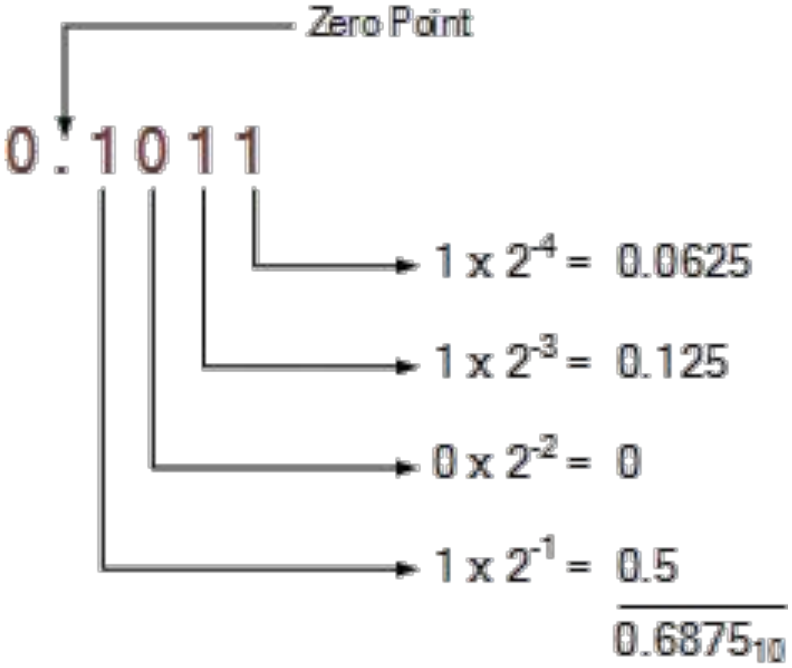
$0\ 00001\ 0000000000_2$

$$2^{-14} \times (1 + \frac{0}{1024})$$

0.000061035156

Binary Fractions

- Another example:
- 0.1011
  - 1011/10000 binary
  - 11/16 = 0.6875
- Or look at it in terms of the twos places...



Float Value Calculation

- Consider another 16 bit floating point number
- Sign bit: 0 (positive)
- Exponent 13 → 2^(13-15) = 2^-2
- Fraction = 0101010101 → (1 + 341/1024)
- *This is the closest 16 bit floating point can represent 1/3rd*

$0\ 01101\ 0101010101_2$

$$2^{-2} \times (1 + \frac{341}{1024})$$

0.33325195

Float Precision

- This is a serious limitation of floating point: It can only be an approximation of many values

$$0\ 01101\ 0101010101_2$$

$$2^{-2} \times (1 + \frac{341}{1024})$$

$$0.33325195$$

Float Rounding Rules

- Floating Point has different rounding rules
  - Round to nearest, ties to even
  - Round to nearest, ties away from zero
  - Round toward 0
  - Round toward +∞
  - Round toward -∞

$$0\ 01101\ 0101010101_2$$

$$2^{-2} \times (1 + \frac{341}{1024})$$

$$0.33325195$$

Float Precision Implications

- You must be very careful when using floating point!
- Floating point is a bad idea when dealing with fixed precision numbers
  - E.g. Money!
- That's OK though, our industry is smart enough to understand this...

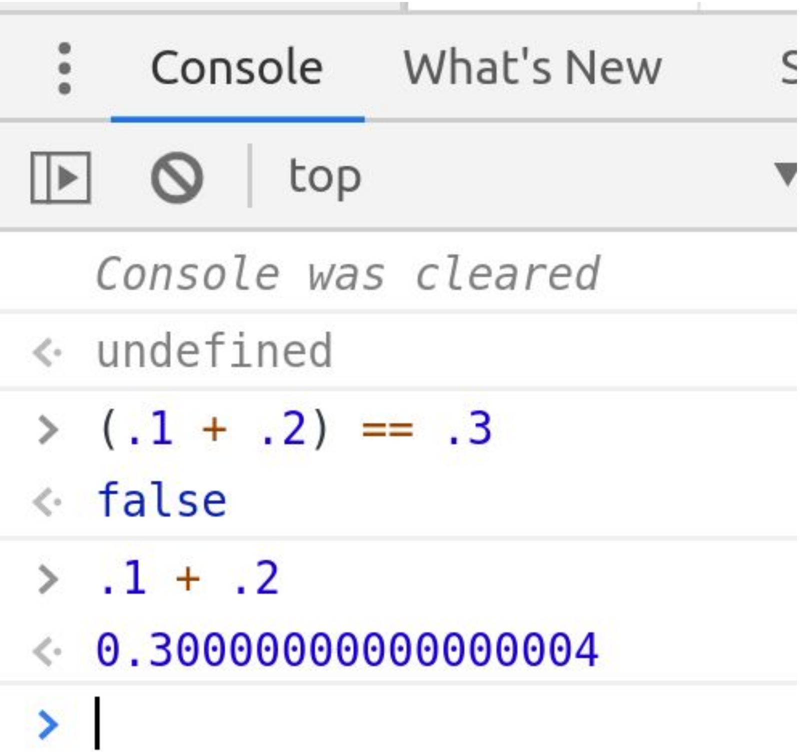
$0\ 01101\ 0101010101_2$

$$2^{-2} \times (1 + \frac{341}{1024})$$

0.33325195

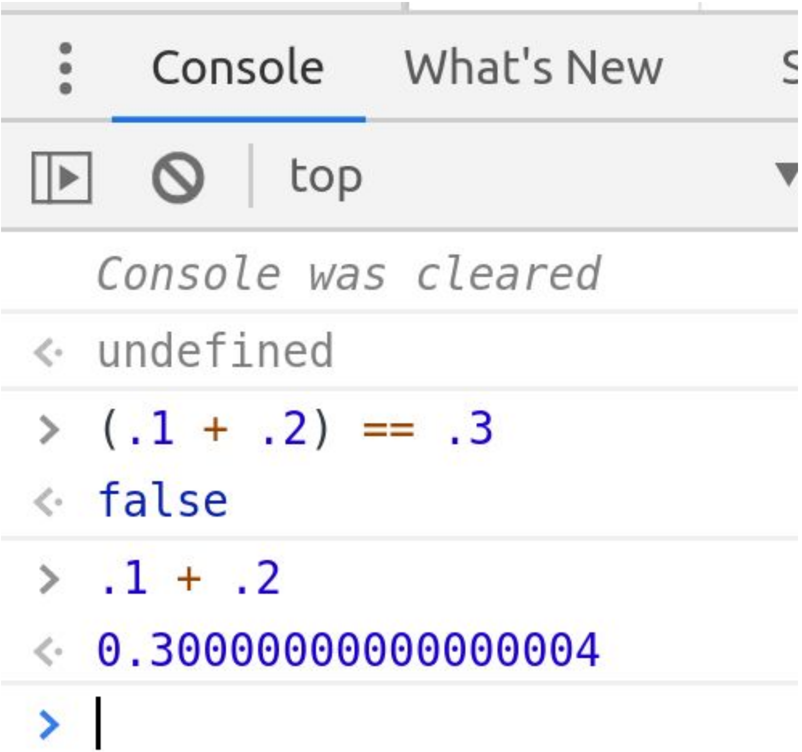
Float Precision Implications

- You must be very careful when using floating point!
- Floating point is a bad idea when dealing with fixed precision numbers
  - E.g. Money!
- That’s OK though, our industry is smart enough to understand this...



Float Precision Implications

- Uhhhhh
- Javascript uses 32 bit floats for numbers
- So *obviously true* mathematical statements are… false
- People are increasingly writing software in javascript...



Float Precision Implications

> mfw

Float Precision Implications

- The Patriot Missile Incident
  - February 1991
  - Precision issue in a MIM-104 Patriot missile battery prevented it from intercepting an incoming SCUD missile
  - 28 soldiers killed

Float Special Values

- Float has some *special* values
  - A signed 0 value
    - Equal to one another
    - Division by one or the other leads to a signed infinity
  - Infinities
    - + and - infinities
    - Satisfy standard infinity math (e.g 3 + +∞ = +∞)
  - NaN (Not a Number)
    - Represents invalid values such as 1/0 or sqrt(-1)

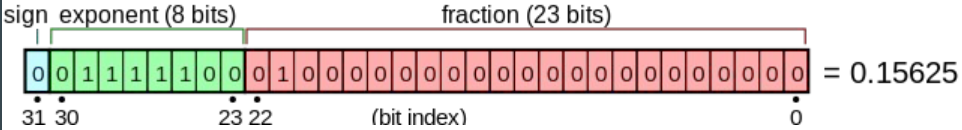
$0\ 01101\ 0101010101_2$

$$2^{-2} \times (1 + \frac{341}{1024})$$

0.33325195

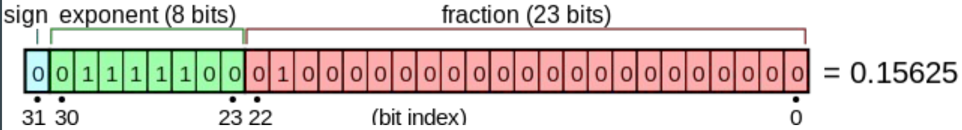
Larger Floats

- We have been looking at half floats, but this same logic generalizes to any length of bits
  - float - typically 32 bits
  - double - 64 bits
- With more bits, more precision



Larger Floats

- We have been looking at half floats, but this same logic generalizes to any length of bits
  - float - typically 32 bits
  - double - 64 bits
- With more bit, more precision
  - Still not perfect precision however



Floating Point Today

- Recall, on X86-64
  - Floating point values are passed in separate registers
  - 128 bits of precision are available
  - XMM0-XMM7 (part of x86-32 SSE)
  - XMM8-15 (available in 64-bit mode only)
  - Separate assembly instructions for working with them

Floating Point

- We took a look at how to represent non-integer values in binary
- Initially *Fixed Point* was used
- *Floating Point* representation is more efficient
    - But also more complex!
- We took a look at 16 bit floats
    - Larger floats are just more of the same
- And we looked at problems with Floating Point precision
    - Javascript is a very terrible programming language
- *REMEMBER: IT'S JUST BITS*

MONTANA
STATE UNIVERSITY