What about signed fixed point?

- Could also have a signed-magnitude fixed-point number
  - MSB represents positive (0) or negative (1)
- It is possible to have a fixed-point two's complement number
  - Would it be any different?

How does arithmetic work with fixed point?

- Addition is the same!
  - If the two numbers have the same scale
- Subtraction is the same!
  - If the two numbers have the same scale
- Multiplication is the same!
  - But it must keep the correct number of fraction bits in the product…
  - Both numbers are "scaled" so the result has double the "scale"

Example adding two fixed point numbers

- 0011.1100 + 0000.0110
- What are the values?

- What is the result?

Example adding two fixed point numbers

- 0011.1100 + 0000.0110
- What are the values?
  - 0011.1100 = 1+2+½ +¼

- What is the result?

Example adding two fixed point numbers

- 0011.1100 + 0000.0110
- What are the values?
  - 0011.1100 = 1+2+½ +¼
    = 3.75

- What is the result?

Example adding two fixed point numbers

- 0011.1100 + 0000.0110
- What are the values?
  - 0011.1100 = 1+2+½ +¼
    = 3.75

0000.0110 = 0 + ¼ +⅛ = .375

- What is the result?

Example adding two fixed point numbers

- 0011.1100 + 0000.0110
- What are the values?
  - 0011.1100 = 1+2+½ +¼
    = 3.75

0000.0110 = 0 + ¼ +⅛ = .375

- What is the result?

0011.1100

\+ 0000.0110

100.0010

Example adding two fixed point numbers

- 0011.1100 + 0000.0110
- What are the values?
  - 0011.1100 = 1+2+½ +¼
    = 3.75

0000.0110 = 0 + ¼ +⅛ = .375

- What is the result?

0011.1100

\+ 0000.0110

100.0010 = 4 + ⅛ = 4.125

Floating Point Numbers

Consider: A x 10^B, where A is one digit

A	B	A x 10^B
0	any	0
1 .. 9	0	1 .. 9
1 .. 9	1	10 .. 90
1 .. 9	2	100 .. 900
1 .. 9	-1	0.1 .. 0.9
1 .. 9	-2	0.01 .. 0.09

How to do scientific notation in binary?
Standard: IEEE 754 Floating-Point

Real numbers

- Our decimal system handles non-integer *real* numbers by adding yet another symbol - the decimal point (.) to make a *fixed point* notation:
  - e.g. 3,456.78 = $3*10^3 + 4*10^2 + 5*10^1 + 6*10^0 + 7*10^{-1} + 8.10^{-2}$
- The *floating point*, or scientific, notation allows us to represent very large and very small numbers (integer or real), with as much or as little precision as needed:
  - Unit of electric charge e = 1.602 176 462 * 10^{-19} Coul.
  - Volume of universe = 1 * 10^{85} cm^3
    - the two components of these numbers are called the mantissa and the exponent

Real numbers in floating point

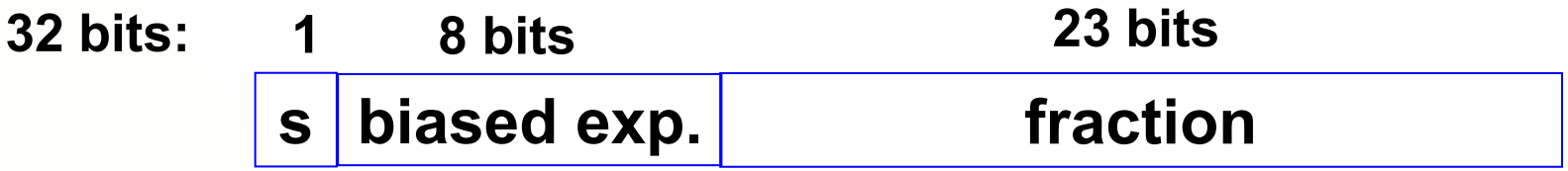
- We mimic the decimal floating point notation to create a "hybrid" binary floating point number:
  - We first use a "binary point" to separate whole numbers from fractional numbers to make a fixed point notation:
    - e.g. 00011001.110 => $1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 1*2^{-2}$ => 25.75

      ($2^{-1} = 0.5$ and $2^{-2} = 0.25$, etc.)
  - We then "float" the binary point:
    - 00011001.110 => 1.1001110×2^4

      mantissa = 1.1001110, exponent = 4
  - Now we have to express this without the extra symbols (x, 2, .)
    - by convention, we divide the available bits into three fields:

      sign, mantissa, exponent

IEEE-754 fp numbers Single Precision



$$N = (-1)^s \times 1.\textbf{fraction} \times 2^{(\textbf{biased exp.} - 127)}$$

- Sign: 1 bit
- Mantissa: 23 bits
  - We "normalize" the mantissa by dropping the leading 1 and recording only its fractional part

- Exponent: 8 bits
  - In order to handle both +ve and -ve exponents, we add 127 to the actual exponent to create a "biased exponent":
    - 2^{-127} => biased exponent = 0000 0000 (= 0)
    - 2^{0} => biased exponent = 0111 1111 (= 127)
    - 2^{+127} => biased exponent = 1111 1110 (= 254)

IEEE-754 fp numbers

- Example:
  - 25.75 => 00011001.110 => 1.1001110×2^4
  - sign bit = 0 (+ve)
  - normalized mantissa (fraction) = (1.)100 1110 0000 0000 0000 0000
  - biased exponent = 4 + 127 = 131 => 1000 0011
  - so 25.75 => 0 1000 0011 100 1110 0000 0000 0000 0000 => 0x41CE0000

How to convert 64.2 into IEEE SP

- Get a binary representation for 64.2
  - Binary of left of radix/decimal point is: 1000000
  - Binary of right of radix/decimal:
    - Successively multiply value by 2 and compare to 1
      - 0.2 x 2 = 0.4 less than 1 so… **0**
      - 0.4 x 2 = 0.8 less than 1 so… **0**
      - 0.8 x 2 = 1.6 g.t. 1 so… **1**
      - 0.6 x 2 = 1.2 g.t. 1 so… **1**
      - 0.2 x 2 = 0.4 **0**
      - 0.4 x 2 = 0.8 **0**
      - 0.8 x 2 = 1.6 **1**
      - 0.6 x 2 = 1.2 **1**

Quiz 2 is available on Canvas right now!

(continued)

- Binary for .2: .0011 0011 0011 0011
- 64.2 is: 1000000.0011001100110011…

- Normalize binary form
  - Produces: 1.0000000011001100110011 x 2^6

(continued)

- 3. Turn true exponent into bias 127
  - E = 6 + 127 = 133 = 10000101
- 4. Put it together:
  - 23-bit F is: (1.)0000 0000 0011 0011 0011 0
- S E F is:
  - S = 0
  - E = 10000101
  - F = 00000000110011001100110
- In hex:
  - 0x42806666

    0100 0010 1000 0000 0110 0110 0110 0110

Convert IEEE SP to real

- What is the decimal value for this SP FP number 0xC228 0000?
  - Convert to binary

    1100 0010 0010 1000 0000 0000 0000 0000
  - Break into S, E, F:
  - E is 10000100 = 132 decimal: 132 – 127= 5
  - F is (1.)0101000…
  - Move decimal over 5: 101010.000…
  - S E F is -42!

Convert IEEE SP to Real

- 0x3F800000

Convert IEEE SP to Real

- 0x3F800000

0011 1111 1000 0000 0000 0000 0000 0000

Convert IEEE SP to Real

- 0x3F800000

0011 1111 1000 0000 0000 0000 0000 0000

S = 0

E = 0111 1111 = 127 - 127

F = 1.0

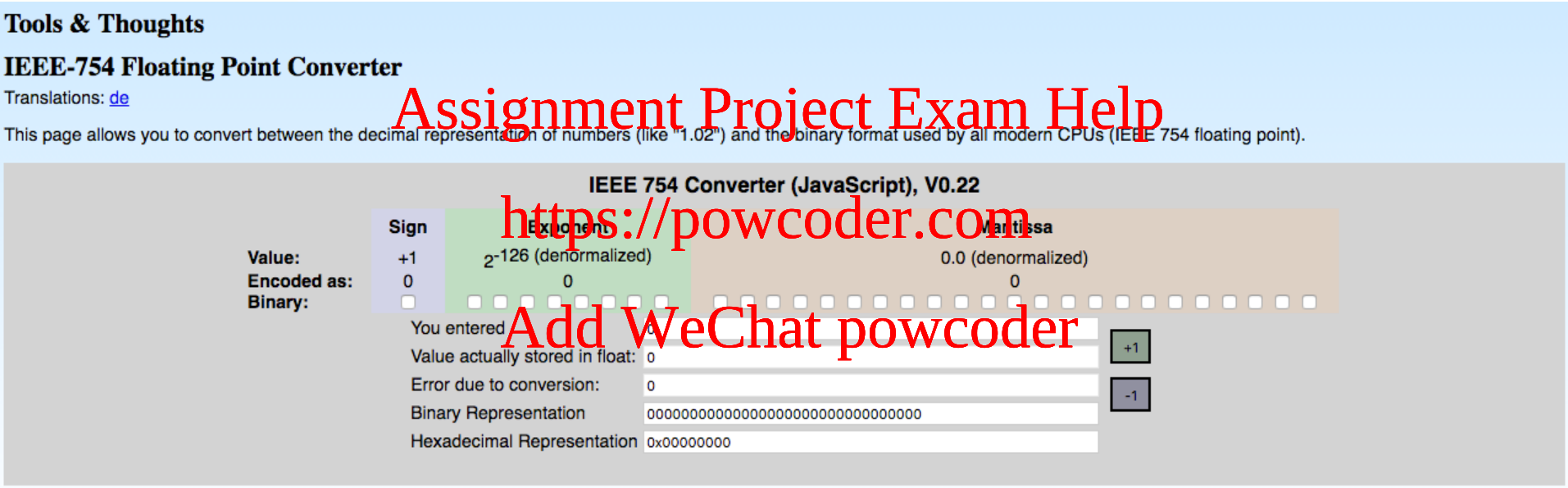
0x3F8 = 1 in single precision floating point

Take Home Practice

- What is 47.625_{10} in SP FP format?
- What is 0x44ed8000 as real number?

Check your Practice

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



IEEE-754 Double Precision

- Double precision (64 bit) floating point

$$N = (-1)^{s} \times 1.\textbf{fraction} \times 2^{(\textbf{biased exp.} - 1023)}$$

How to represent 0, NaN, +/- Infinity?

- Values represented by convention:
- Infinity (+ and -): exponent = 255 (1111 1111) and fraction = 0
- NaN (not a number): exponent = 255 and fraction ≠ 0
- Zero (0): exponent = 0 and fraction = 0
  - Note: exponent = 0 => fraction is de-normalized (i.e. no hidden 1)