# CSC 411

**Computer Organization (Fall 2024)**
**Lecture 7: Floating Point (part 1)**

Prof. Marco Alvarez, University of Rhode Island

## Fractional binary numbers

- Used to represent fractional numbers
- Binary point
  - separates the integer and fractional parts (similar to the decimal point)
  - bits to the right of binary point represent fractional powers of 2

| $2^{i}$ | $2^{i-1}$ | | $2^{1}$ | $2^{0}$ | $2^{-1}$ | $2^{-2}$ | | $2^{-j+1}$ | $2^{-j}$ |
|---|---|---|---|---|---|---|---|---|---|
| $b_i$ | $b_{i-1}$ | … | $b_1$ | $b_0$ | $b_{-1}$ | $b_{-2}$ | … | $b_{-j+1}$ | $b_{-j}$ |

$$\sum_{k=-j}^{i} b_k 2^k$$

11.010 =

1101.101 =

## Practice

- Convert fractional binary numbers to decimal

$$1.10 =$$
$$11.001 =$$
$$100.01 =$$
$$11.111 =$$
$$0.010 =$$

## Practice

- Convert 349.84375 to a fractional binary number

$$\begin{array}{rcll} 349/2 &=& 174 & R \quad 1 \\ 174/2 &=& 87 & R \quad 0 \\ 87/2 &=& 43 & R \quad 1 \\ 43/2 &=& 21 & R \quad 1 \\ 21/2 &=& 10 & R \quad 1 \\ 10/2 &=& 5 & R \quad 0 \\ 5/2 &=& 2 & R \quad 1 \\ 2/2 &=& 1 & R \quad 0 \\ 1/2 &=& 0 & R \quad 1 \end{array}$$

$$\begin{array}{rcl} 0.84375 * 2 &=& 1.6875 \\ 0.6875 * 2 &=& 1.375 \\ 0.375 * 2 &=& 0.75 \\ 0.75 * 2 &=& 1.5 \\ 0.5 * 2 &=& 1.0 \\ 0.0 * 2 &=& 0 \end{array}$$

101011101.110110

## Observations

- Not all decimal fractions have exact binary equivalents
  - can only represent numbers of the form $\frac{x}{2^k}$
    - irrational and some rational numbers can't be represented
  - e.g., $1/3 = 0.33333\ldots$ or $0.1$
    - both require an infinite series of 0s and 1s
  - large repeating decimals can only be approximated within a certain degree of accuracy
- Note that 0.111111… represents a decimal number just below 1.0

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots \frac{1}{2^i} + \ldots = 1.0 - \varepsilon$$

## Practice

- Apply the 'multiply by 2' method to convert 0.1 from decimal to fractional binary
  - explain what is happening
- Type 0.1 + 0.2 on a python terminal
  - explain the resulting value

## Normalized scientific notation

- Fractional binary numbers in the form $1.xxx\ldots \times 2^k$
  - analogous to decimal scientific notation
  - always has a single 1 to the left of the binary point
  - exponent adjusts to maintain the same value
- Examples
  - 101.11 => normalized $1.0111 \times 2^2$
  - 0.00101 => normalized $1.01 \times 2^{-3}$
  - 1000.101 => normalized $1.000101 \times 2^3$

Forms the basis of the IEEE 754 standard

## Practice

- Convert to **normalized scientific notation**
  - single 1 to the left of the binary point

$$10.10 = \underbrace{1.010}_{\text{significand}} \times \underbrace{2}_{\text{base}}{}^{\overbrace{1}^{\text{exponent}}}$$

$$110.001 =$$

$$1111.1010 =$$

$$0.010101 =$$

$$0.0000010 =$$

# IEEE 754 standard

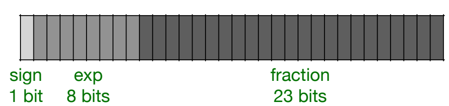
## IEEE standard 754

- Defines a common format for representing real numbers in computers
  - developed in response to divergence of representations
  - first standardized in 1985 and revised in 2008
  - supported by major CPUs (almost universally adopted)
  - provides different **precision levels** (**half, single, double, quadruple**) for various needs
- Standardizes the following layout :
  - more **exp** bits leads to a wider range of numbers
  - more **fraction** bits leads to higher precision

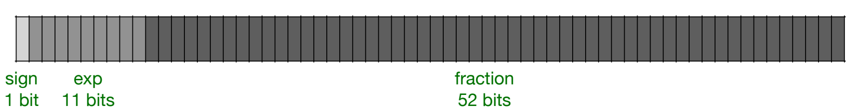
| | | |
|---|---|---|
| `sign` | `exp` | `fraction` |

## Common precision levels

- **Single-precision** (32 bits) — `float`
  - good balance of performance and range




- **Double-precision** (64 bits) — `double`
  - higher precision for demanding calculations



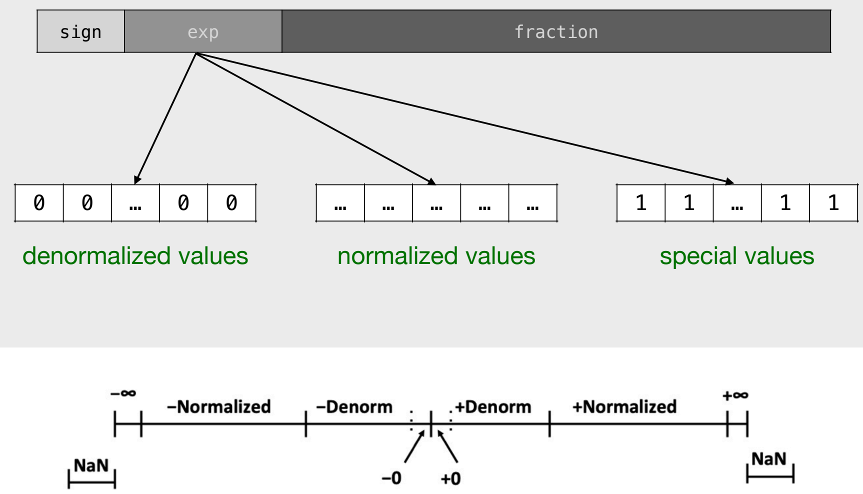

## Floating point encoding/decoding

$$(-1)^s M 2^E$$

- sign bit S
  - 0 for positive, 1 for negative
- exponent E
  - magnitude of the number, the term $2^E$ scales the mantissa
- mantissa M
  - a.k.a. **significand**, it captures the significant digits of the number
    - normally in range $[1.0, 2.0)$

# Floating point categories

## Floating point categories




## Floating point categories

- Normalized values
  - represent the majority of values, using the full precision of the significand, and a wide range of magnitudes
  - **exp** != 000…000 and **exp** != 111…111
- Denormalized values
  - used for <u>very small numbers</u>, extending the range of representable values closer to zero
  - lower precision compared to normalized numbers
  - **exp** == 000…000
- Special numbers
  - represent special cases like NaN, infinity (positive and negative)
  - **exp** == 111…111

# Normalized values

## Normalized values

$$(-1)^s M 2^E$$

- **E = exp - bias**
  - exp is the unsigned integer represented by the bits in the exp field
  - bias is $2^{k-1} - 1$, where $k$ is the length in bits of the exp field
    - single precision (bias = $2^{8-1} - 1 = 127$)
    - double precision (bias = $2^{11-1} - 1 = 1023$)
- **M = 1.bb…bb**
  - bb…bb are the bits in the fraction field
  - M is the decimal that corresponds to 1.bb…bb
    - note that M has an implied leading 1

| sign | exp | fraction |
|---|---|---|

## Practice (encoding)

$$(-1)^s M 2^E$$

- Assume a float F = 2024.0
  - convert to fractional binary
    - 11111101000
  - make it form 1.xxx
    - 1.1111101000 x $2^{10}$
  - write M and frac
    - M = 1.1111101000
    - frac = 11111010000000000000000
  - write exp using E = exp - bias
    - exp = E + bias = 10 + 127 = 137 = 10001001
  - final binary sequence
    - 0 10001001 11111010000000000000000

0x44FD0000

## Practice (encoding)

$$(-1)^s M 2^E$$

- Assume a float F = 0.5
  - convert to fractional binary
    - 0.1
  - make it form 1.xxx
    - 1.0 x $2^{-1}$
  - write M and frac
    - M = 1.0
    - frac = 00000000000000000000000
  - write exp using E = exp - bias
    - exp = E + bias = -1 + 127 = 126 = 01111110
  - final binary sequence
    - 0 01111110 00000000000000000000000

0x3F000000

## Practice (encoding)

$$(-1)^s M 2^E$$

- Convert the following decimals to hexadecimal representation (single precision)
  - 1.3125
  - 3.53125

# Practice (decoding)

$(-1)^s M 2^E$

- Assume a float F = 0x43CDA000
  - write the binary
    - divide into **s**, **exp**, **frac**
    - 0 10000111 10011011010000000000000
  - calculate M
    - M = 1.10011011010000000000000 = 1.6064453125
  - calculate E
    - E = exp - bias = 135 - 127 = 8
  - write final number
    - $(-1)^0 * 1.6064453125 * 2^8 = 411.25$

# Practice (decoding)

$(-1)^s M 2^E$

- Convert the following values in hexadecimal representation (single precision) to decimal
  - 0x41850000
  - 0x43cdb400