CSC 411

Computer Organization (Spring 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	1/2	1/4		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 a decimal to a fractional binary number
  - 349.84375

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

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

101011101.110110

Practice

- Convert to **normalized scientific notation**
  - single non-zero digit to the left of the binary point

$$10.10 = \overset{\text{significand}}{1.010} \times \underset{\text{base}}{2}^{\overset{\text{exponent}}{1}}$$

$$110.001 =$$

$$1111.1010 =$$

$$0.010101 =$$

$$0.0000010 =$$

Observations

- Not all decimal fractions have exact binary equivalents
  - can only represent numbers of the form $\frac{x}{2^k}$
  - e.g., 0.3 is represented as 0.010101…
    - requires an infinite series of 0s and 1s
  - large repeating decimals can only be approximated within a certain degree of accuracy
- 0.111111… represents a number just below 1.0

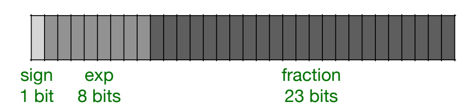
$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots \frac{1}{2^i} + \ldots = 1.0 - \varepsilon$$

IEEE standard 754

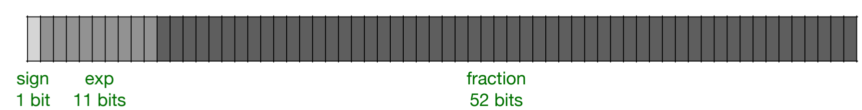
- Defines a common format for representing real numbers in computers
  - developed in response to divergence of representations
  - supported by major CPUs (almost universally adopted)
  - provides different precision levels (single, double, extended) for various needs
- Standardizes the following layout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

Precision

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



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



Precision

Format	Smallest positive value (*)	Largest positive value (*)	Precision (**)
single	~$1.401 \cdot 10^{-45}$	~$3.403 \cdot 10^{+38}$	6-9 digits
double	~$4.941 \cdot 10^{-324}$	~$1.798 \cdot 10^{+308}$	15-17 digits

```
(*) Smallest/largest negative values are the same as their positive
counterparts, but negative.

(**) Precision refers to to the number of significant digits that
can be represented in a number.
```

Floating point encoding/decoding

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

- **sign bit S**
  - -1^0 for positive, -1^1 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
    - scaled by the exponent, normally in range $[1.0, 2.0)$

Normalized and denormalized numbers

- **Normalized values**
  - represent a wide range of values with maximum precision
  - **exp** != 000…000 and **exp** != 111…111
- **Denormalized values**
  - used for very small numbers providing more range, they have lower precision compared to normalized number
  - **exp** == 000…000
- **Special numbers**
  - e.g., NaN, infinity
  - **exp** == 111…111

sign	exp	fraction

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
    - $1.1111101000 \times 2^{10}$
  - 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
    - 1.0×2^{-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 (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
    - $(-1)^0 * 1.6064453125 * 2^8 = 411.25$