Floating Point Numbers

Last updated 6/14/23

These slides introduce floating point numbers

Floating Point Numbers

- Scientific Number Representation
  - $1.60217657 \times 10^{-19}$ coulombs
  - $6.0221413 \times 10^{+23}$ units/mole
  - Normalized to have only 1 digit (non-zero) to the left of the decimal point
  - multiplied by a power of 10
  - $5692.3456 \rightarrow 5.6923456 \times 10^{+3}$
  - $.00023456 \rightarrow 2.3456 \times 10^{-4}$
  - format: mantissa $\times 10^{\text{exponent}}$

Floating Point Numbers

- Binary Floating Point Number Representation
  - Normalized to have only 1 digit to the left of the decimal point
    - this must be a 1 since our choices are only 0 and 1 and we don't use 0
  - multiplied by a power of 2

  - 1011.1101 → $1.0111101 \times 2^{+3}$
  - .00011001 → 1.1001×2^{-4}

  - format: mantissa × 2^{exponent}

BUT

- since the mantissa always starts with "1." we can use
  1.fraction × 2^{exponent}

Floating Point Numbers

- It is simpler to work with only positive exponents
  - Bias the exponent
    - With an 8 bit signed binary exponent the range is:

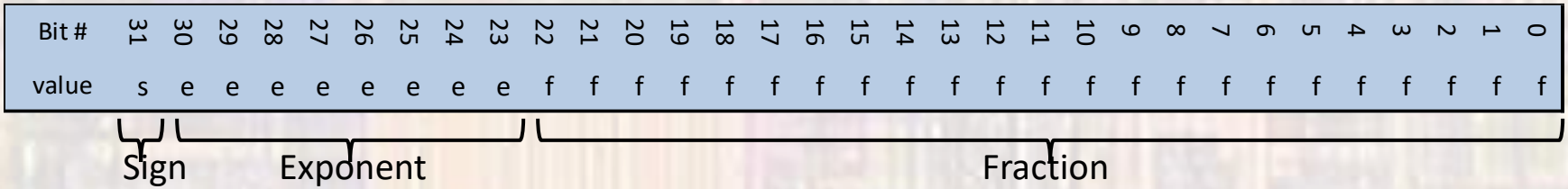
      +127 to -127 (-128 is used for special cases)
    - Add 127 to the desired exponent value (for use in the representation – now considered unsigned)

      actual range is still +127 to -127

      representation range is 254 to 0 (255 is used for special cases)
    - Called an exponent with +127 bias
  - format is now: value = $1.\text{fraction} \times 2^{(\text{exponent} - 127)}$

Floating Point Numbers

- IEEE Standard
  - 32 bit format
  - value = (sign) x 1.fraction × $2^{(\text{exponent} - 127)}$
    - Sign = 0 for positive numbers, Sign = 1 for negative numbers



  - Special cases
    - If E = 255, and F is non-zero, then the value is NaN (Not a Number)
    - If E = 255, F = 0 and S = 1, then the value is -infinity
    - If E = 255, F = 0, and S = 0, then the value is +infinity
    - If E = 0, and F = 0, then the value is 0
  - Range
    - $1.11111111111111111111111_2 \times 2^{+127} = 3.4028 \times 10^{38}$
    - $1.00000000000000000000001_2 \times 2^{-127} = 1.1754 \times 10^{-38}$
    - 23 bit fractional precision ↔ 6 to 7 decimal digits

Floating Point Numbers

- Example showing potential for errors

use IEEE standard floating point to represent: 2,345,678.7109375

2,345,678 = 0010 0011 1100 1010 1100 1110
0.7109375 = 0.10110110

2,345,678.7109375 =
0010 0011 1100 1010 1100 1110 . 1011 0110
= 1. 0 0011 1100 1010 1100 1110 1011 0110 × 2^{21}

sign = 0
exponent = 21 + 127 = 148 = 1001 0100
fraction = 0001 1110 0101 0110 0111 0101 1011 0

will not fit in fraction part of the notation

2,345,678.7109375 →

0 10010100 0001 1110 0101 0110 0111 010

↓ Cont'd

Floating Point Numbers

- Example showing potential for errors – cont'd

convert the IEEE floating point number from the previous slide -
0 10010100 0001 1110 0101 0110 0111 010 back to decimal

sign = 0
exponent = 1001 0100 = 148 → $2^{148-127} = 2^{21}$
fraction = 0001 1110 0101 0110 0111 010

+ $1.0001\ 1110\ 0101\ 0110\ 0111\ 010 \times 2^{21}$
= 1 0001 1110 0101 0110 01110 . 10
= 2345678.5

2,345,678.7109375 vs 2345678.5
error = (0.7109375 – 0.5)/2345678.7109375 = 9×10^{-8}
~7 decimal digits of precision

Floating Point Numbers

- Example 2

```
float foo;
foo = 2.222;
printf("%f", foo);
```

Results in 2.22199988 being printed