CS465: Computer Systems Architecture

Computer Arithmetic Review

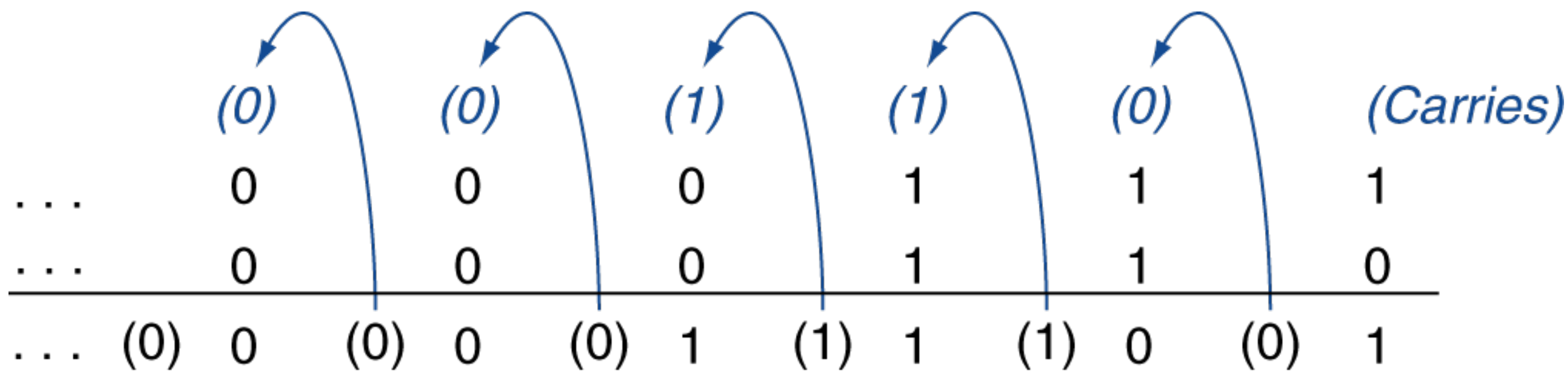
*Slides adapted from Computer Organization and Design by Patterson and Henessey

Outline

- This review includes topics that are well covered in CS367
- Operations on integers (Ch3.2)
  - Addition and subtraction
  - Overflow
- Floating-point numbers (Ch3.5)
  - Representation
  - Addition and multiplication

Example: Integer Addition

- Example: 7 + 6



When Overflow Occurs ?

- Overflow if result out of range
- For signed integers:
  - Adding +ve and –ve operands, no overflow
  - Adding two +ve operands
    - Overflow if result sign is 1
  - Adding two –ve operands
    - Overflow if result sign is 0

Integer Subtraction

- Add negation of second operand
- Example: 7 – 6 = 7 + (–6)

+7:	0000 0000 … 0000 0111
–6:	1111 1111 … 1111 1010
+1:	0000 0000 … 0000 0001

- Overflow if result out of range
  - Subtracting two +ve or two –ve operands, no overflow
  - Subtracting +ve from –ve operand
    - Overflow if result sign is 0
  - Subtracting –ve from +ve operand
    - Overflow if result sign is 1

Floating Point

- Representation for non-integral numbers
  - Including very small and very large numbers
- Like scientific notation
  - -2.34×10^{56}
  - $+0.002 \times 10^{-4}$
  - $+987.02 \times 10^{9}$

normalized

not normalized

- In binary
  - $\pm 1.xxxxxxx_2 \times 2^{yyyy}$
- Types `float` and `double` in C

Floating Point Standard

- Defined by IEEE Std 754-1985
- Developed in response to divergence of representations
  - Portability issues for scientific code
- Now almost universally adopted
- Two representations
  - Single precision (32-bit)
  - Double precision (64-bit)

IEEE Floating-Point Format

	single: 8 bits double: 11 bits	single: 23 bits double: 52 bits
S	Exponent	Fraction

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

- S: sign bit ($0 \Rightarrow$ non-negative, $1 \Rightarrow$ negative)
- Normalize significand: $1.0 \leq |\text{significand}| < 2.0$
  - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
  - Significand is Fraction with the "1." restored
- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Bias = 2^(|exp|-1)-1
  - Single: Bias = 127; Double: Bias = 1023

Single-Precision Range

- Range of normalized numbers
- Exponents 00000000 and 11111111 reserved
- Smallest value
  - Exponent: 00000001
    $\Rightarrow$ actual exponent = 1 – 127 = –126
  - Fraction: 000…00 $\Rightarrow$ significand = 1.0
  - $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$
- Largest value
  - exponent: 11111110
    $\Rightarrow$ actual exponent = 254 – 127 = +127
  - Fraction: 111…11 $\Rightarrow$ significand $\approx$ 2.0
  - $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

Double-Precision Range

- Range of normalized numbers
- Exponents 0000…00 and 1111…11 reserved
- Smallest value
  - Exponent: 00000000001
    $\Rightarrow$ actual exponent = 1 – 1023 = –1022
  - Fraction: 000…00 $\Rightarrow$ significand = 1.0
  - $\pm 1.0 \times 2^{-1022} \approx \pm 2.2 \times 10^{-308}$
- Largest value
  - Exponent: 11111111110
    $\Rightarrow$ actual exponent = 2046 – 1023 = +1023
  - Fraction: 111…11 $\Rightarrow$ significand ≈ 2.0
  - $\pm 2.0 \times 2^{+1023} \approx \pm 1.8 \times 10^{+308}$

Floating-Point Example

- What number is represented by the single-precision float

  1 10000001 01000…00

  - S = 1
  - Fraction = $01000\ldots00_2$
  - Exponent = 10000001_2 = 129

Floating-Point Example

- Represent –0.75
  - $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$
  - S = 1
  - Fraction = $1000\ldots00_2$
  - Exponent = –1 + Bias
    - Single: –1 + 127 = 126 = 01111110_2
    - Double: –1 + 1023 = 1022 = 01111111110_2
- Single: 1011111101000…00
- Double: 1011111111101000…00

Denormalized Numbers

- Exponent = 000...0 $\Rightarrow$ hidden bit is 0

$$x = (-1)^S \times (0 + \text{Fraction}) \times 2^{-126}$$

- Smaller than normalized numbers
  - Allow for gradual underflow, with diminishing precision
- Denormalized with fraction = 000...0

$$x = (-1)^S \times (0 + 0) \times 2^{-126} = \pm 0.0$$

Two representations of 0.0!

Other Special Patterns

- Exponent = 1…1
  - Fraction = all zero $\Rightarrow$ Infinities
  - Result of computations like X/0
  - Allows operations to continue past overflow situations
    - E.g. X/0 > 10
- Exponent = 1…1
  - Fraction = not all zero $\Rightarrow$ Not a number
  - Result of computations like sqrt(-4) or 0/0
  - Support mixing numerical and symbolic computation or other extensions

IEEE 754 FP Definition

S	Exponent	Fraction

- Single precision

<u>Exponent</u>	<u>Fraction</u>	<u>Object</u>
0	0	0
0	nonzero	denom
1-254	anything	+/- fl. pt. #
255	0	+/- infinity
255	nonzero	NaN

- Overflow: exponent is too large to be represented in the bits
- Underflow: a non-zero fraction that is to small to be represented (negative exponent too large to be represented in the bits)

Outline

- This review includes topics that are well covered in CS367
- Operations on integers
  - Addition and subtraction
  - Overflow
- Floating-point numbers
  - Representation
  - Addition and multiplication ←

Floating-Point Addition

- Consider a 4-digit binary example
  - $1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2}$ (0.5 + –0.4375)
- 1. Align binary points
  - Shift number with smaller exponent
  - $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$
- 2. Add significands
  - $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$
- 3. Normalize result & check for over/underflow
  - $1.000_2 \times 2^{-4}$, with no over/underflow
- 4. Round and renormalize if necessary
  - $1.000_2 \times 2^{-4}$ (no change) = 0.0625

Rounding Modes

- Assume that we can only keep two digits to the right of binary point

Value	1.0101	1.0111	-1.0110	1.1010
Round up	1.10	1.10	-1.01	1.11
Round down	1.01	1.01	-1.10	1.10
Truncate (round to zero)	1.01	1.01	-1.01	1.10
Round to nearest even	1.01	1.10	-1.10	1.10

- Round to even
  - If <half, round down; if >half, round up
  - If ==half, use evenness to break the tie

Floating-Point Multiplication

- Now consider a 4-digit binary example
  - $1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2}$ (0.5 × –0.4375)
- 1. Add exponents
  - Unbiased: –1 + –2 = –3
  - Biased: (–1 + 127) + (–2 + 127) = –3 + 254 adjusting by -127 = –3 + 127
- 2. Multiply significands
  - $1.000_2 \times 1.110_2 = 1.110 \Rightarrow 1.110_2 \times 2^{-3}$
- 3. Normalize result & check for over/underflow
  - $1.110_2 \times 2^{-3}$ (no change) with no over/underflow
- 4. Round and renormalize if necessary
  - $1.110_2 \times 2^{-3}$ (no change)
- 5. Determine sign: +ve × –ve ⇒ –ve
  - $-1.110_2 \times 2^{-3} = -0.21875$