Computer Organization

Floating Point Part III

Prof. Roger Luis Uy
De La Salle University
College of Computer Studies

IEEE-754 Rounding rules

- Directed Rounding
  - Round towards 0: also known as *truncation*
  - Round towards +infinity: also known as *rounding up* or *ceiling*
  - Round towards –infinity: also known as *round down* or *floor*
- Rounding to nearest
  - Round to nearest, ties to even
  - Round to nearest, ties away from zero

Round to nearest, ties to even

- Rounds to the nearest value
- If the number falls midway it is rounded to the nearest value with an even (zero) least significant bit
- Default for binary, recommended default for decimal
- +23.5 = ?; +24.5 = ?
- -23.5 = ?; -24.5 = ?
- Also known as banker’s rounding, unbiased rounding, convergent rounding, statistician’s rounding, Dutch rounding, Gaussian rounding, odd-even rounding, broken rounding

Round to nearest, ties away from zero

- Rounds to the nearest value
- If the number falls midway it is rounded to the nearest value above (for positive numbers) or below (for negative numbers)
- +23.5 = ?; +24.5 = ?
- -23.5 = ?; -24.5 = ?
- Also known as round half way from zero or round half way towards infinity

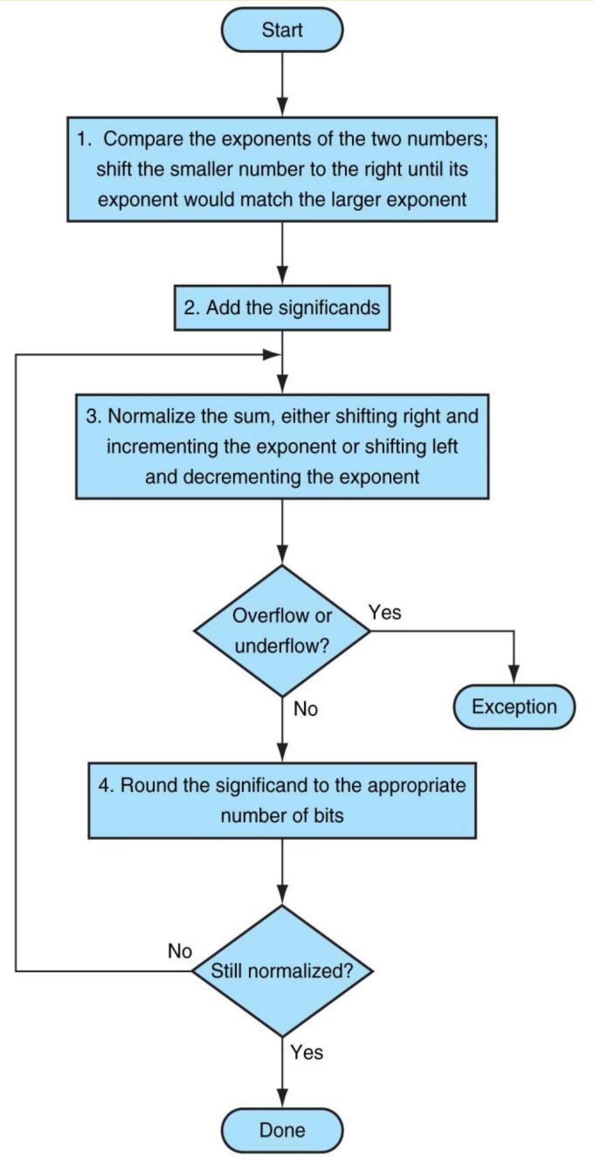
IEEE-754 Rounding rules Example (to 2 significant digits)

Number	Round down	Round up	Truncate	Round to nearest ties away from zero	Round to nearest ties to even
+23.*67*	+23	+24	+23	+24	+24
+23.*35*	+23	+24	+23	+23	+23
-23.*35*	-24	-23	-23	-23	-23
-23.*67*	-24	-23	-23	-24	-24
+11.*50*	+11	+12	+11	+12	+12
+12.*50*	+12	+13	+12	+13	+12
-11.*50*	-12	-11	-11	-12	-12
-12.*50*	-13	-12	-12	-13	-12

IEEE-754 Rounding rules Example (to 7 significant digits)

Number	Round down	Round up	Truncate	Round to nearest ties away from zero	Round to nearest ties to even
+0.100101*110*	+0.100101	+0.100110	+0.100101	+0.100110	+0.100110
+0.100101*010*	+0.100101	+0.100110	+0.100101	+0.100101	+0.100101
-0.001111*110*	-0.010000	-0.001111	-0.001111	-0.010000	-0.010000
-0.001111*001*	-0.010000	-0.001111	-0.001111	-0.001111	-0.001111

Floating Point Addition Algorithm



Floating-point addition. The normal path is to execute steps 3 and 4 once, but if rounding causes the sum to be unnormalized, we must repeat step 3.

Floating-Point Addition

- Consider a 4-digit decimal example
  - $9.999 \times 10^{1} + 1.610 \times 10^{-1}$
- 1. Align decimal points
  - Shift number with smaller exponent
  - $9.999 \times 10^{1} + 0.016 \times 10^{1}$
- 2. Add significands
  - $9.999 \times 10^{1} + 0.016 \times 10^{1} = 10.015 \times 10^{1}$
- 3. Normalize result & check for over/underflow
  - 1.0015×10^{2}
- 4. Round and renormalize if necessary
  - 1.002×10^{2}

Floating-Point Addition

- Now 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) $= 0.0625$

Floating-Point Addition

- Consider a 7-digit decimal example
  - 123456.7 + 101.7654
- 1. Align decimal points
  - Shift number with smaller exponent
  - $1.234567 \text{ x } 10^5 + 0.001017654 \text{ x } 10^5$
- 2. Add significands
  - $1.234567 \text{ x } 10^5 + 0.001018 \text{ x } 10^5 = 1.235585 \times 10^5$
- 3. Normalize result & check for over/underflow
  - 1.235585×10^5 (same)
- 4. Round and renormalize if necessary
  - 1.235585×10^5 (same)

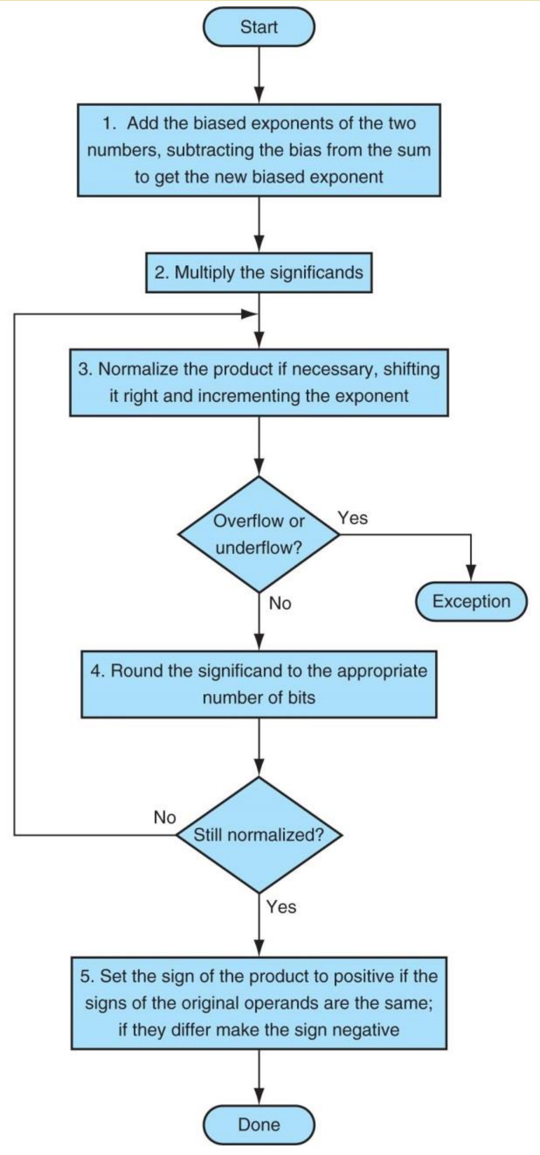
Floating-Point Addition

- Consider a 7-digit decimal example
  - $1.234567 \times 10^{5} + 9.876543 \times 10^{-3}$
- 1. Align decimal points
  - Shift number with smaller exponent
  - $1.234567 \times 10^{5} + 0.000000009876543 \times 10^{5}$
- 2. Add significands
  - $1.234567 \times 10^{5} + 0.0000000 \times 10^{5} = 1.234567 \times 10^{5}$
- 3. Normalize result & check for over/underflow
  - 1.234567×10^{5} (same)
- 4. Round and renormalize if necessary
  - 1.234567×10^{5} (same)

Floating-Point Subtraction

- Consider a 7-digit decimal example
  - 123457.1467 - 123456.659
- 1. Align decimal points
  - Shift number with smaller exponent
  - $1.234571467 \times 10^5 – 1.23456659 \times 10^5$
- 2. Add significands
  - $1.234571 \times 10^5 + 1.234567 \times 10^5 = 0.000004 \times 10^5$
- 3. Normalize result & check for over/underflow
  - 4.000000×10^{-1}
- 4. Round and renormalize if necessary
  - 4.000000×10^{-1} (same)

Floating Point Multiplication Algorithm



The normal path is to execute steps 3 and 4 once, but if rounding causes the sum to be unnormalized, we must repeat step 3.

Floating-Point Multiplication

- Consider a 4-digit decimal example
  - $1.110 \times 10^{10} \times 9.200 \times 10^{-5}$
- 1. Add exponents
  - For biased exponents, subtract bias from sum
  - New exponent = 10 + –5 = 5
- 2. Multiply significands
  - $1.110 \times 9.200 = 10.212 \;\Rightarrow\; 10.212 \times 10^{5}$
- 3. Normalize result & check for over/underflow
  - 1.0212×10^{6}
- 4. Round and renormalize if necessary
  - 1.021×10^{6}
- 5. Determine sign of result from signs of operands
  - $+1.021 \times 10^{6}$

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: $(-1 + 127) + (-2 + 127) = -3 + 254 - 127 = -3 + 127$
- 2. Multiply significands
  - $1.000_2 \times 1.110_2 = 1.1102 \;\Rightarrow\; 1.110_2 \times 2^{-3}$
- 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: +operand × –operand ⇒ –operand
  - $-1.110_2 \times 2^{-3} = -0.21875$

Floating-Point Multiplication

- Consider a 7-digit decimal example
  - $4.734612 \times 10^3 \times 5.417242 \times 10^5$
- 1. Add exponents
  - For biased exponents, subtract bias from sum
  - New exponent = 3 + 5 = 8
- 2. Multiply significands
  - $4.734612 \times 5.417242 = 25.64853898104 \Rightarrow 25.64853898104 \times 10^8$
- 3. Normalize result & check for over/underflow
  - $2.564853898104 \times 10^9$
- 4. Round and renormalize if necessary
  - 2.564854×10^9
- 5. Determine sign of result from signs of operands
  - $+ 2.564854 \times 10^9$

Guard, Round & Sticky Bits

- *Guard bit* the first of the two extra bits kept on the right during intermediate calculations of floating point numbers; used to improve rounding accuracy.
- *Round bit* the second of the two extra bits kept on the right during intermediate calculations of floating point numbers ; used to improve rounding accuracy.
- *Sticky bit* – bit used in rounding in addition to guard and round bit that is set whenever there are nonzero bits to the right of the round bit.

Rounding with Guard & Round Bit

- Consider a 7-digit decimal example
  - 123457.1467 - 123456.659
- 1. Align decimal points
  - Shift number with smaller exponent
  - $1.234571467 \text{ x } 10^5 – 1.23456659 \text{ x } 10^5$
- 2. Add significands (with guard bit and round bit)
  - $1.2345714\mathbf{6} \text{ x } 10^5 + 1.2345665\mathbf{9} \text{ x } 10^5 = 0.00000487 \times 10^5$
- 3. Normalize result & check for over/underflow
  - 4.87×10^{-1}
- 4. Round and renormalize if necessary
  - 4.87×10^{-1} (same) vs. [compare to 4.000000×10^{-1} without guard & round bit]

Rounding with Guard & Round Bit

- Consider a 3-digit decimal example
  - $2.56 \times 10^0 + 2.34 \times 10^2$
- 1. Align decimal points
  - Shift number with smaller exponent
  - $0.0256 \times 10^2 + 2.34 \times 10^2$
- 2. Add significands (with guard bit and round bit)
  - $0.0256 \times 10^2 + 2.3400 \times 10^2 = 2.3656 \times 10^2$
- 3. Normalize result & check for over/underflow
  - 2.3656×10^2 (same)
- 4. Round and renormalize if necessary
  - 2.37×10^2

Rounding without Guard & Round Bit

- Consider a 3-digit decimal example
  - $2.56 \times 10^0 + 2.34 \times 10^2$
- 1. Align decimal points
  - Shift number with smaller exponent
  - $0.0256 \times 10^2 + 2.34 \times 10^2$
- 2. Add significands
  - $0.02 \times 10^2 + 2.34 \times 10^2 = 2.36 \times 10^2$
- 3. Normalize result & check for over/underflow
  - 2.36×10^2 (same)
- 4. Round and renormalize if necessary
  - 2.36×10^2 (same)

Rounding with Guard, Round & Sticky Bit

- Consider a 3-digit decimal example
  - $5.01 \times 10^{-1} + 2.34 \times 10^{2}$
- 1. Align decimal points
  - Shift number with smaller exponent
  - $0.00501 \times 10^{2} + 2.34 \times 10^{2}$
- 2. Add significands (with guard bit, round bit and sticky bit = 1 since there is non-zero after round bit)
  - $0.0050 \times 10^{2} + 2.3400 \times 10^{2} = 2.3450 \times 10^{2}$
- 3. Normalize result & check for over/underflow
  - 2.3450×10^{2} (same)
- 4. Round and renormalize if necessary
  - 2.35×10^{2} (since sticky bit is 1)
  - [If no sticky bit, it will be 2.34×10^{2}]

ULP (unit in the last place)

- Multiple definitions (Kahan 2004, Goldberg 1991, Harrison 1999, Markstein 2000, Overton 2001)
- Basically: gap between the two floating point numbers nearest x, even if x is one of them
- Also: the number of bits in error in the least significant bits of the significand between the actual number and the number that can be represented

ULP (unit in the last place)

Example:

A = 011110001

B = 011111010

ULP = 4

X = 2.3656

Y = 2.3700

ULP = 44

X = 2.3600

Y = 2.3700

ULP = 1

IEEE-754 Exception Handling

- There are five exception handling for IEEE-754
- Invalid Operations
  - Square root of a negative number
  - Returns qNaN by default
- Division by Zero
  - Operation on **finite** operands gives an exact infinite result (e.g., 1/0 or log(0))
  - Returns +/- infinity by default
- Overflow
  - Result is too large to be represented correctly
  - Returns +/- infinity by default

IEEE-754 Exception Handling

- Underflow
  - A result is very small, outside the normal range, and is inexact
  - Returns a de-normalize value by default
- Inexact
  - Example: 1/3
  - Returns correctly rounded result by default

IEEE-754 Exception Handling

- 0/0 $\rightarrow$ NaN
- 1/0 $\rightarrow$ +infinity
- -1/0 $\rightarrow$ -infinity
- infinity +0 $\rightarrow$ infinity
- infinity - 0 $\rightarrow$ infinity
- infinity * 0 $\rightarrow$ NaN
- Infinity / 0 $\rightarrow$ Infinity
- 0 / infinity $\rightarrow$ 0
- Nan +-*/ 0 $\rightarrow$ NaN
- log(-N) $\rightarrow$ NaN
- 0^{-n} $\rightarrow$infinity

infinity+infinity $\rightarrow$ infinity

infinity-infinity $\rightarrow$ NaN*

infinity*infinity $\rightarrow$ infinity

infinity/infinity $\rightarrow$ NaN

NaN + - * / NaN $\rightarrow$ NaN

infinity +-*/ NaN $\rightarrow$ NaN

NaN +-*/ infinity $\rightarrow$ NaN

sqrt(-1) $\rightarrow$ NaN

log(0) $\rightarrow$ -infinity

*$_{\text{infinity}}$ sign should agree else infinity

IEEE-754 Exception Handling

- Additional two exception handling for decimal floating-point operation
- Clamped
  - Result's **exponent** too large for the destination format
  - By default, trailing zeroes will be added to reduce the exponent value but if not possible **overflow** occurs
- Rounded
  - Result's **coefficient** requires more digits than the destination format
  - **Inexact** is signaled if any non-zero digits are discarded