CPEG 422/622
EMBEDDED SYSTEMS DESIGN

Chengmo Yang

chengmo@udel.edu

Evans 201C

LECTURE 10
FLOATING POINT

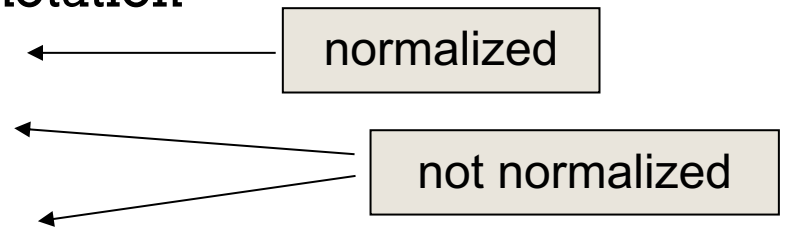
REVIEW

- Last Lecture:
  - Sequential circuit debugging
  - Test your multiplier
  - VHDL programming tips

- This lecture:
  - Floating Point Numbers
  - Floating Point Arithmetic
  - Floating Point Multiplication in VHDL

FLOATING-POINT REPRESENTATION

- Commonly used for representing
  - real numbers (3.1415926)
  - very small and very large numbers
- Like scientific notation
  - -2.34×10^{56}
  - $+0.002 \times 10^{-4}$
  - $+987.02 \times 10^{9}$



- In binary, floating point numbers are represented as

$$\pm 1.xxxxxxx_2 \times 2^{yyyy}$$

- Types `float` and `double` in C

FLOATING-POINT REPRESENTATION

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

- S: sign bit (0 ⇒ non-negative, 1 ⇒ negative)
- Exponent: excess representation: actual exponent + Bias
  - Why?
  - Ensures exponent is unsigned, easy for comparing two FP numbers
- Fraction: part of the significand: 1.0 ≤ Normalize significand < 2.0
  - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)

IEEE 754 REPRESENTATION

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

Current version is IEEE 754-2008. It replaces IEEE 754-1985.

Name	Size (bits)	S (bit)	Exp (bits)	Frac (bits)	Bias (value)
Half precision	16	1	5	10	15
Single precision	**32**	**1**	**8**	**23**	**127**
Double precision	64	1	11	52	1023
Quadruple precision	128	1	15	112	16383

IEEE 754 SINGLE-PRECISION

- Exponents 00000000 and 11111111 reserved
- Smallest+:

  0 00000001 1.00000000000000000000000

  $= 1 \times 2^{1-127} \approx 1.2 \times 10^{-38}$

- Largest+:

  0 11111110 1.11111111111111111111111

  $= 2-2^{-23} \times 2^{254-127} \approx 3.4 \times 10^{+38}$

- Example: represent –0.75
  - $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$
  - S = 1
  - Fraction = $1000\ldots00_2$
  - Exponent = –1 + Bias = –1 + 127 = 126 = 01111110_2
  - Single precision: 10111111001000…00

IEEE 754 SINGLE-PRECISION

- Exponents 00000000 and 11111111 reserved

- Exponent = 000…0
  - Fraction = 000…0 => true 0
  - Fraction ≠ 000…0 => denormalized number
    Smaller than normal numbers, allow for gradual underflow.

- Exponent = 111…1
  - Fraction = 000…0 => ±Infinity
  - Fraction ≠ 000…0 => Not-a-Number (NaN),
    Indicates illegal or undefined result, e.g., 0.0 / 0.0

FLOATING POINT ADDITION/SUBTRACTION

$$(\pm F1 \times 2^{E1}) + (\pm F2 \times 2^{E2}) = \pm F3 \times 2^{E3}$$

- Step 0: **Restore** the hidden bit in F1 and in F2
- Step 1: **Align** fractions by right shifting F2 by E1 - E2 positions (assuming E1 ≥ E2)
- Step 2: **Add/subtract** the resulting F2 to F1 to form F3
- Step 3: **Normalize** F3 (so it is in the form 1.XXXXX …)
  - If F1 and F2 have the same sign → F3 ∈[1,4) → 1 bit right shift F3 and increment E3 (check for overflow)
  - If F1 and F2 have different signs → F3 may require *many* left shifts each time decrementing E3 (check for underflow)
- Step 4: **Round** F3 and possibly normalize F3 again
- Step 5: **Rehide** the most significant bit of F3 before storing the result

FLOATING POINT ADDITION EXAMPLE

Add	$(0.5 = 1.0000 \times 2^{-1}) + (-0.4375 = -1.1100 \times 2^{-2})$

- Step 0: Hidden bits restored in the representation above
- Step 1: Shift significand with the smaller exponent (1.1100) right until its exponent matches the larger exponent
- Step 2: Add/subtract significands

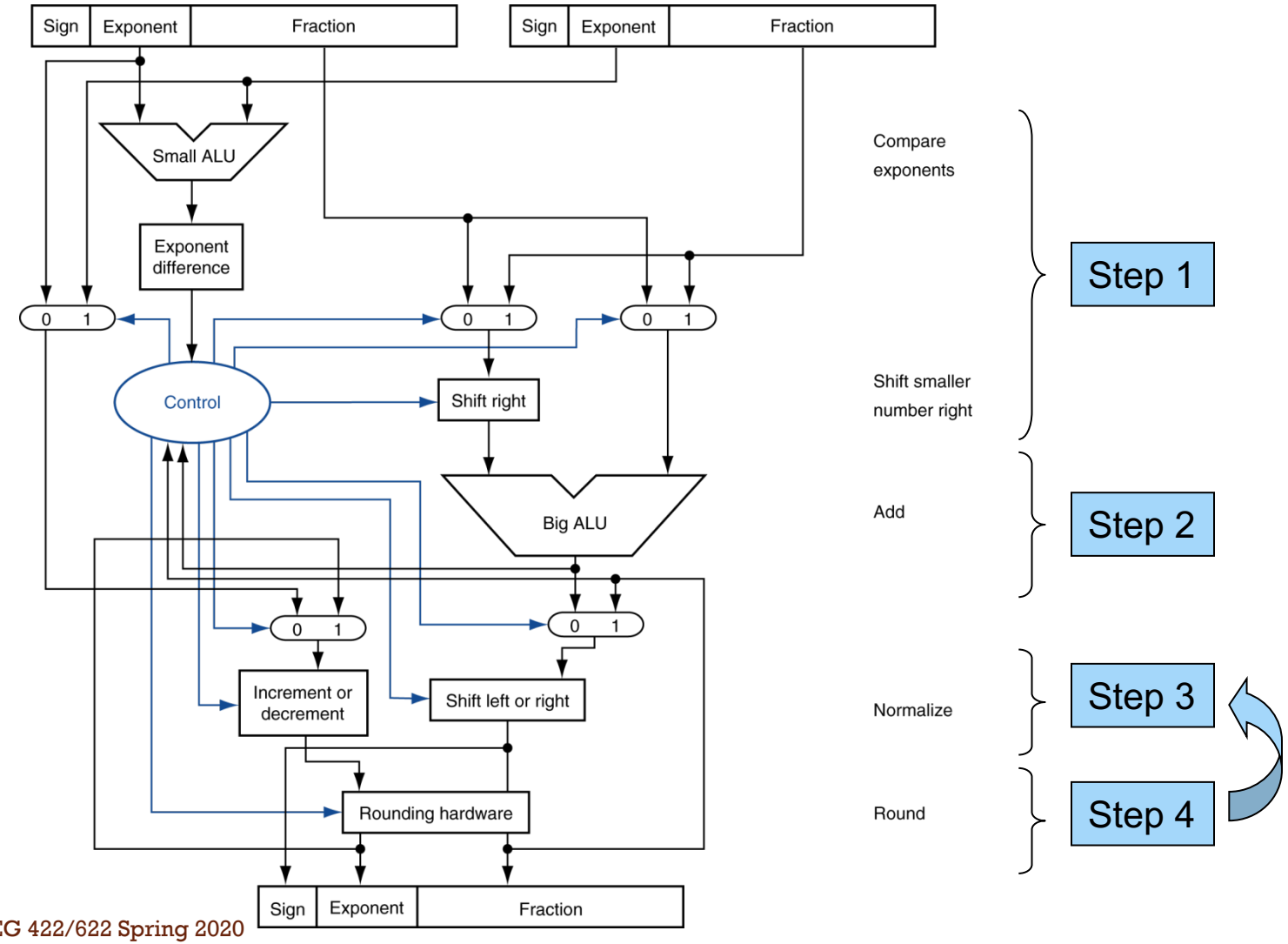
  $1.0000 + (-0.111) = 1.0000 - 0.111 = 0.001$

- Step 3: Normalize the sum, checking for exponent over/underflow

  $0.001 \times 2^{-1} = 1.000 \times 2^{-4}$

- Step 4: The sum is already rounded, so we're done
- Step 5: Rehide the hidden bit before storing

FP ADDER/SUBTRACTOR HARDWARE



SUPPORT FOR ACCURATE ARITHMETIC

IEEE 754 FP rounding modes

- Always round up (toward +∞)
- Always round down (toward -∞)
- Truncate
- Round to nearest even (when the Guard || Round || Sticky are 100) – always creates a 0 in the least significant (kept) bit of F

F = 1 . xxxxxxxxxxxxxxxxxxxxxxx G R S

  - **G**uard bit – used to provide one F bit when shifting left to normalize a result (e.g., when normalizing F after division or subtraction)
  - Round bit – used to improve rounding accuracy
  - Sticky bit – used to support Round to nearest even

FLOATING POINT MULTIPLICATION

$$(\pm F1 \times 2^{E1}) \times (\pm F2 \times 2^{E2}) = \pm F3 \times 2^{E3}$$

- Step 0: **Restore** the hidden bit in F1 and in F2
- Step 1: **Add** the two (biased) exponents and subtract the bias from the sum, so E1 + E2 – 127 = E3

  also determine the sign of the product
- Step 2: **Multiply** F1 by F2 to form F3
- Step 3: **Normalize** F3
  - Since F1 and F2 come in normalized → F3 $\in$[1,4) → 1 bit right shift F3 and increment E3
  - Check for overflow/underflow
- Step 4: **Round** F3 and possibly normalize F3 again
- Step 5: **Rehide** the most significant bit of F3 before storing the result

FLOATING POINT MULTIPLICATION EXAMPLE

Multiply	$(0.5 = 1.0000 \times 2^{-1})$ x $(-0.4375 = -1.1100 \times 2^{-2})$

- Step 0: Hidden bits restored in the representation above
- Step 1: Add the exponents
  Not in bias would be -1 + (-2) = -3
  In bias would be (-1+127) + (-2+127) – 127 = 124
- Step 2: Multiply the significands
  1.0000 x 1.110 = 1.110000
- Step 3: Normalized the product, checking for exp over/underflow
  1.110000×2^{-3} is already normalized
- Step 4: The product is already rounded, so we're done
- Step 5: Rehide the hidden bit before storing

FP MULTIPLIER HARDWARE