Computer Arithmetic

Dr. Ahmed H. Zahran
WGB 182
a.zahran@cs.ucc.ie

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
  - Add 0.5_{10} and -0.4375_{10}

    $1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2}$

1. Align binary points (***Shift*** number with smaller exponent)

    $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$

2. ***Add*** significands

    $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$

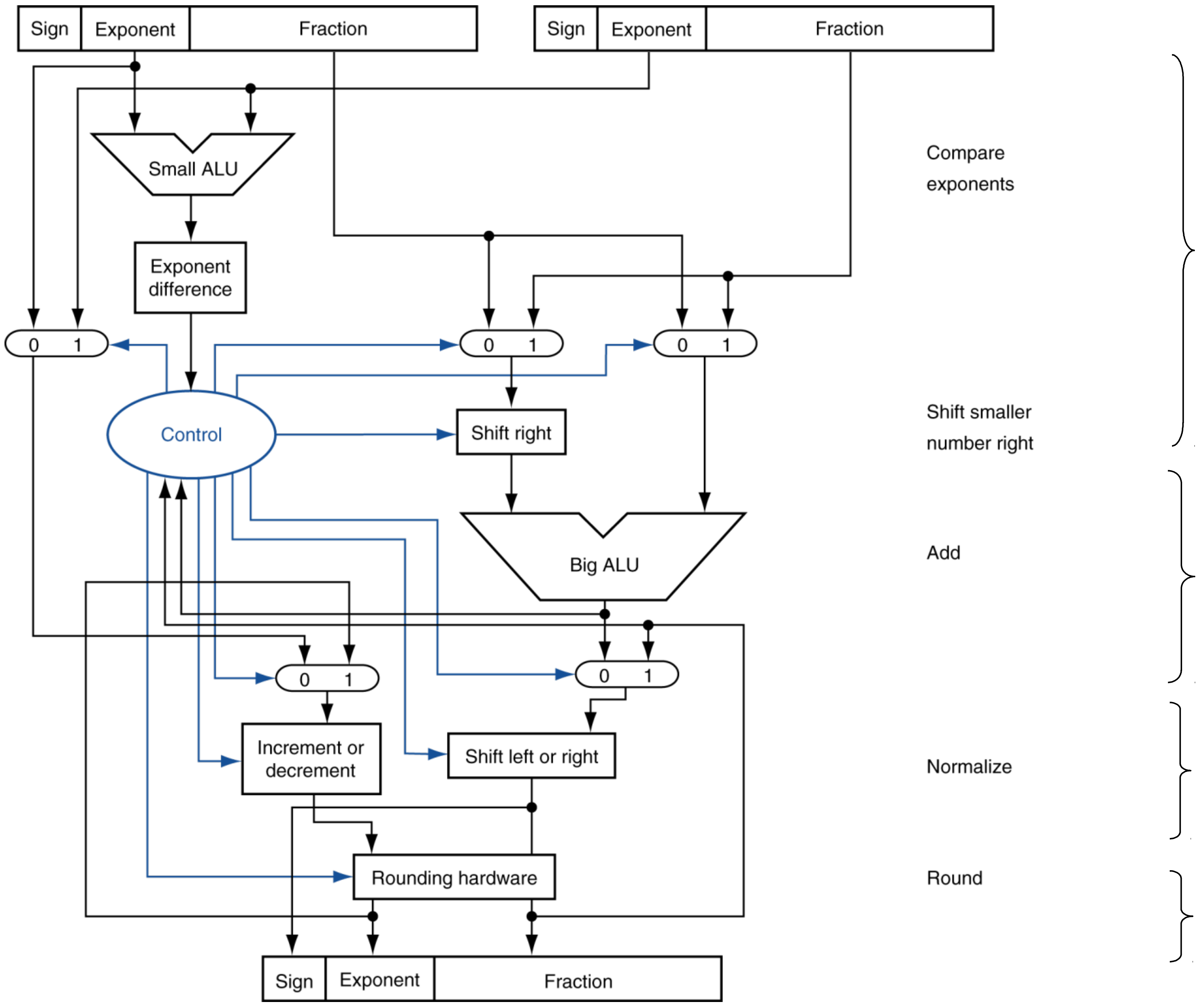
3. ***Normalize*** result & check for over/underflow

    $1.000_2 \times 2^{-4}$, with no over/underflow

4. ***Round*** and renormalize if necessary

- $1.000_2 \times 2^{-4}$ (no change) $= 0.0625_{10}$

FP Adder Hardware



Step 1

Step 2

Step 3

Step 4

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 *the sign*** of result from signs of operands
    - $+1.021 \times 10^{6}$

Floating-Point Multiplication

- Now consider a 4-digit binary example
  - Multiply 0.5_{10} and -0.4375_{10}
    - $1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2}$
- 1. ***Add exponents***
  - Unbiased: $-1 + -2 = -3$
  - Biased: $(-1 + 127) + (-2 + 127) = -3 + 254 - 127 = -3 + 127$
- 2. ***Multiply significands***
    - $1.000_2 \times 1.110_2 = 1.110_2 \ \Rightarrow\ 1.110_2 \times 2^{-3}$
- 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
    - $-1.110_2 \times 2^{-3} = -0.21875_{10}$

FP Arithmetic Hardware

- FP multiplier is of similar complexity to FP adder
  - But uses a multiplier for significands instead of an adder
- FP arithmetic hardware usually does
  - Addition, subtraction, multiplication, division, reciprocal, square-root
  - FP ↔ integer conversion
- Operations usually takes several cycles
  - Can be pipelined

Review

- Revised binary representation
- Explored basic integer operations and the [optimized] hardware implementation of multiplication and division
- Introduced floating point representation [precision and range]
- Explored the floating point adder hardware design
- Explored the procedure of multiplication and division

FP Example: °F to °C

- C code:

```
float f2c (float fahr) {
  return ((5.0/9.0)*(fahr - 32.0));
}
```

  - fahr in $f12, result in $f0, literals in global memory space

- MIPS code:

```
f2c: lwc1  $f16, const5
     lwc1  $f18, const9
     div.s $f16, $f16, $f18
     lwc1  $f18, const32
     sub.s $f18, $f12, $f18
     mul.s $f0,  $f16, $f18
     jr    $
```

Associativity Pitfall

a+(b+c) = (a+b)+c

- Parallel programs may interleave operations in unexpected orders
  - Assumptions of associativity may fail

		(x+y)+z	x+(y+z)
x	-1.50E+38		-1.50E+38
y	1.50E+38	0.00E+00	
z	1.0	1.0	1.50E+38
		1.00E+00	0.00E+00

Floating point addition is not generally associative!

Who Cares About FP Accuracy?

- Important for scientific code
  - But for everyday consumer use?
    - “My bank balance is out by 0.0002¢!”
- The Intel Pentium FDIV bug in 1994
  - The market expects accuracy
  - Costed $500 Million to correct
  - No Christmas bonus for Intel Engineers

Reading

- Sections 3.6, 3.7, 3.9, and 3.10