CHAPTER 3-2

Arithmetic for Computers

By Pattama Longani
Collage of arts, media and Technology

REAL NUMBER

- programming languages support numbers with fractions, which are called ***reals*** in mathematics.
- Here are some examples of reals:
  - $3.14159265\ldots_{ten}$ (pi)
  - $2.71828\ldots_{ten}$ (e)
  - 0.000000001_{ten} or $1.0_{ten} \times 10^{-9}$
  - $3{,}155{,}760{,}000_{ten}$ or $3.15576_{ten} \times 10^{9}$

FLOATING POINT

- **Scientific notation**: a single digit to the left of the **decimal point**.
  - -2.34×10^{56}
  - $+0.002 \times 10^{-4}$
  - $+987.02 \times 10^{9}$

normalized

not normalized

- In binary
  - $\pm 1.xxx_2 \times 2^{yyy}$
  - x= ***<u>fraction</u>***, y = ***<u>exponent</u>***
- such numbers is called **floating point** because it represents numbers in which the binary point is not fixed.

FLOATING POINT

- A designer of a floating-point representation must find a compromise between the size of the **fraction** and the size of the **exponent.**
- This representation is called **sign and magnitude**, since the sign is a separate bit from the rest of the number.

s	E (exponent)	F (fraction)
1 bit	8 bits	23 bits

Single Precision Floating Point

EXCEPTION IN FLOATING POINT

- **Overflow** (floating point) happens when a positive exponent becomes too large to fit in **the exponent field**
- **Underflow** (floating point) happens when a negative exponent becomes too large to fit in **the exponent field**
- One way to reduce the chance of underflow or overflow is to offer another format that has a larger exponent field

FLOATING POINT

- **Double precision** – takes two MIPS words

s	E (exponent)	F (fraction)
1 bit	11 bits	20 bits

F (fraction continued)
32 bits

- These formats go beyond MIPS. They are part of the **IEEE 754 floating-point standard**, found in virtually every computer invented since 1980.

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)
- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Single: Bias = 127 $= (2^8/2)-1$
  - Double: Bias = 1023 $= (2^{11}/2)-1$

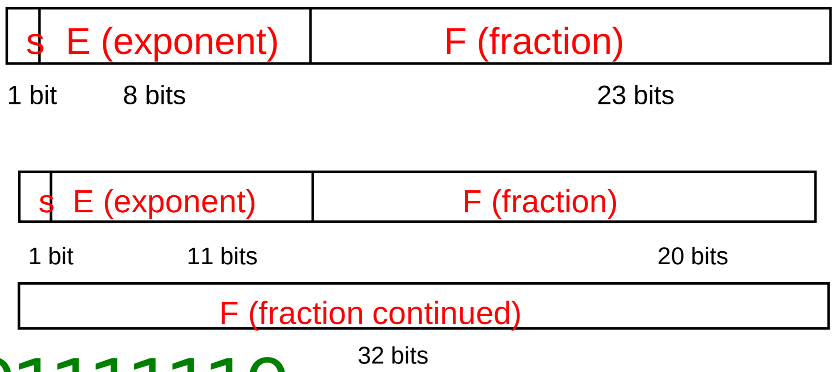
$$(-1)^S \times (1 + (s1 \times 2^{-1}) + (s2 \times 2^{-2}) + (s3 \times 2^{-3}) + (s4 \times 2^{-4}) + \ldots) \times 2^E$$

EXAMPLE : Represent –0.75

- $0.75 = 75/100_{ten} = 3/4_{ten} = 3/2^2{}_{ten}$
  $= 11_2/2^2{}_{ten} = 1.1_2 \times 2^{-1}$ Exponent-Bias
- $(-1)^1 \times 1.1_2 \times 2^{-1}$

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

- S = 1
- Fraction = $1000\ldots00_2$
- Exponent = –1 + Bias
  - Single: –1 + 127 = 126 = 01111110_2
  - Double: –1 + 1023 = 1022 = 01111111110_2



- Single: 1<u>01111110</u>1000…00
- Double: 1<u>01111111110</u>1000…00

EXAMPLE : What number is represented by the single-precision float

1<u>10000001</u>01000…00

s	E (exponent)	F (fraction)
1 bit	8 bits	23 bits

- S = 1
- Fraction = $01000\ldots00_2$
- Exponent = $10000001_2 = 129$
- $x = (-1)^1 \times (1.01_2) \times 2^{(129 - 127)}$ ← Exponent-Bias

$= (-1) \times 1.01_2 \times 2^2$

$= (-1) \times 101_2$

$= -5.0$

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

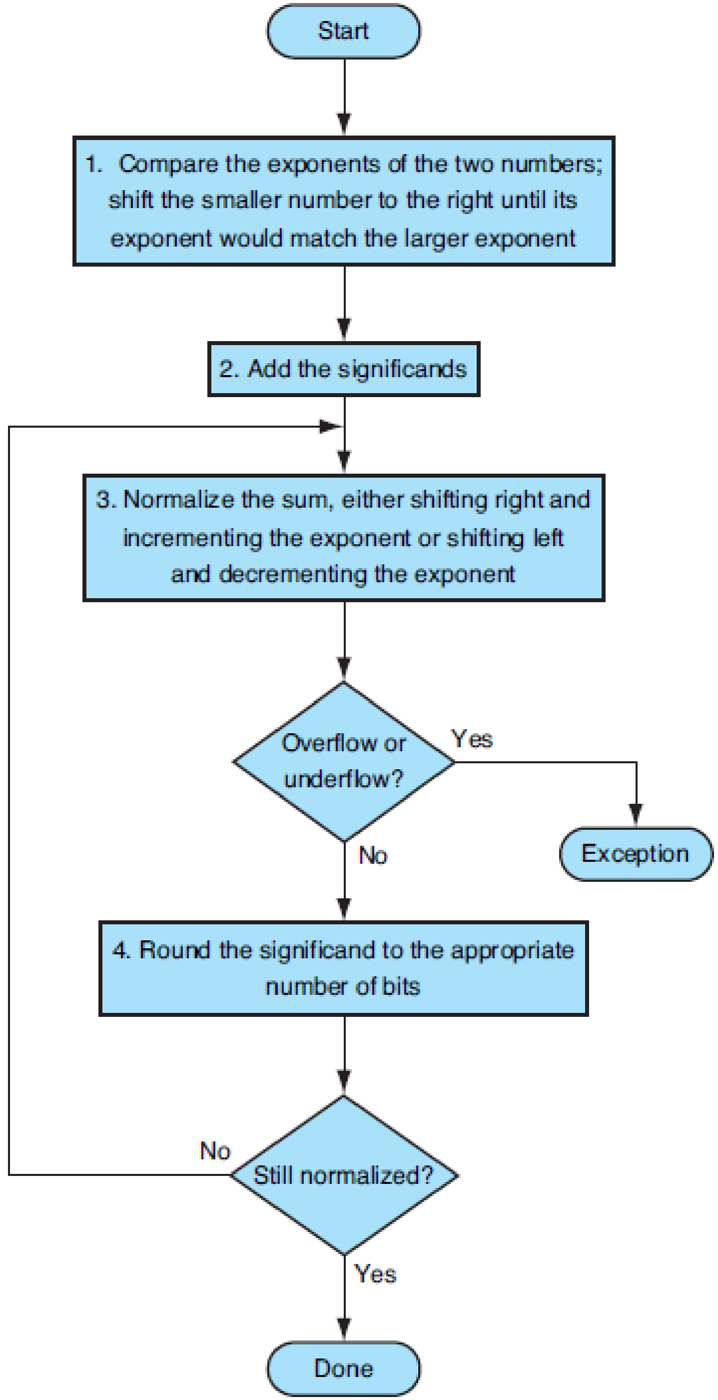
   $9.999 \times 10^1 + 0.016 \times 10^1 = 10.015 \times 10^1$

3. Normalize result & check for over/underflow

   1.0015×10^2

4. Round and renormalize if necessary

   1.002×10^2


Start
1. Compare the exponents of the two numbers; shift the smaller number to the right until its exponent would match the larger exponent
2. Add the significands
3. Normalize the sum, either shifting right and incrementing the exponent or shifting left and decrementing the exponent
Overflow or underflow?
Yes
Exception
No
4. Round the significand to the appropriate number of bits
Still normalized?
No
Yes
Done

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

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

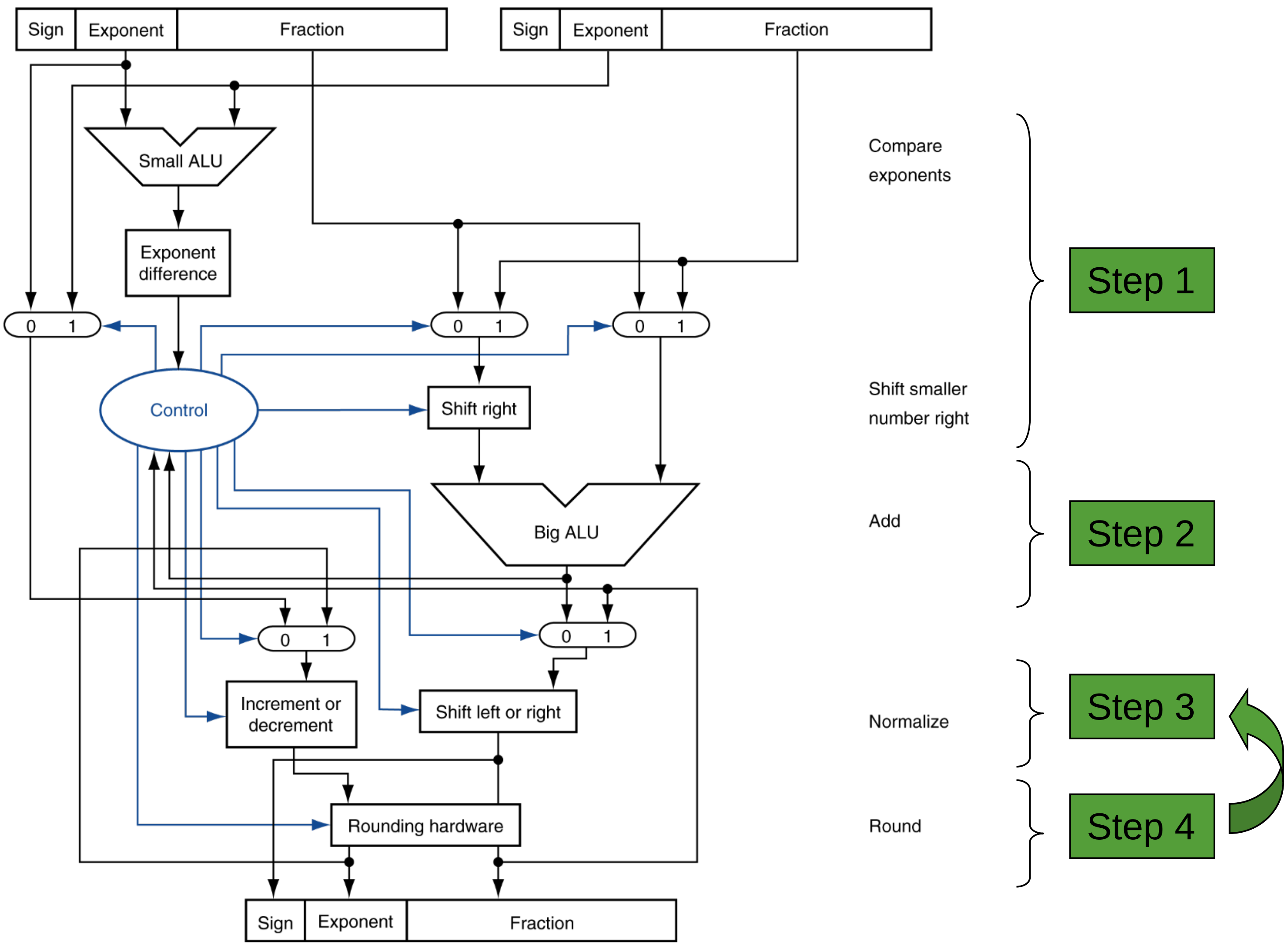
3. Normalize result & check for over/underflow

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

4. Round and renormalize if necessary

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

FP ADDER HARDWARE



FP ADDER HARDWARE

- Much more complex than integer adder
- Doing it in one clock cycle would take too long
  - Much longer than integer operations
  - Slower clock would penalize all instructions
- FP adder usually takes several cycles
  - Can be pipelined