Computer Organization and Architecture

6 Floating-point Arithmetic

Tongwei Ren

Sep. 26, 2019

南京大學
NANJING UNIVERSITY

Review

- Decimal arithmetic operations
  - Addition
  - Subtraction

Floating-point Representation

- IEEE Standard 754

	Single Precision (32 bits)				Double Precision (64 bits)			
	Sign	**Biased exponent**	**Fraction**	**Value**	**Sign**	**Biased exponent**	**Fraction**	**Value**
positive zero	0	0	0	0	0	0	0	0
negative zero	1	0	0	-0	1	0	0	-0
plus infinity	0	255 (all 1s)	0	∞	0	2047 (all 1s)	0	∞
minus infinity	1	255 (all 1s)	0	$-\infty$	1	2047 (all 1s)	0	$-\infty$
quiet NaN	0 or 1	255 (all 1s)	$\neq 0$	NaN	0 or 1	2047 (all 1s)	$\neq 0$	NaN
signaling NaN	0 or 1	255 (all 1s)	$\neq 0$	NaN	0 or 1	2047 (all 1s)	$\neq 0$	NaN
positive normalized nonzero	0	$0 < e < 255$	f	$2^{e-127}(1.f)$	0	$0 < e < 2047$	f	$2^{e-1023}(1.f)$
negative normalized nonzero	1	$0 < e < 255$	f	$-2^{e-127}(1.f)$	1	$0 < e < 2047$	f	$-2^{e-1023}(1.f)$
positive denormalized	0	0	$f \neq 0$	$2^{e-126}(0.f)$	0	0	$f \neq 0$	$2^{e-1022}(0.f)$
negative denormalized	1	0	$f \neq 0$	$-2^{e-126}(0.f)$	1	0	$f \neq 0$	$-2^{e-1022}(0.f)$

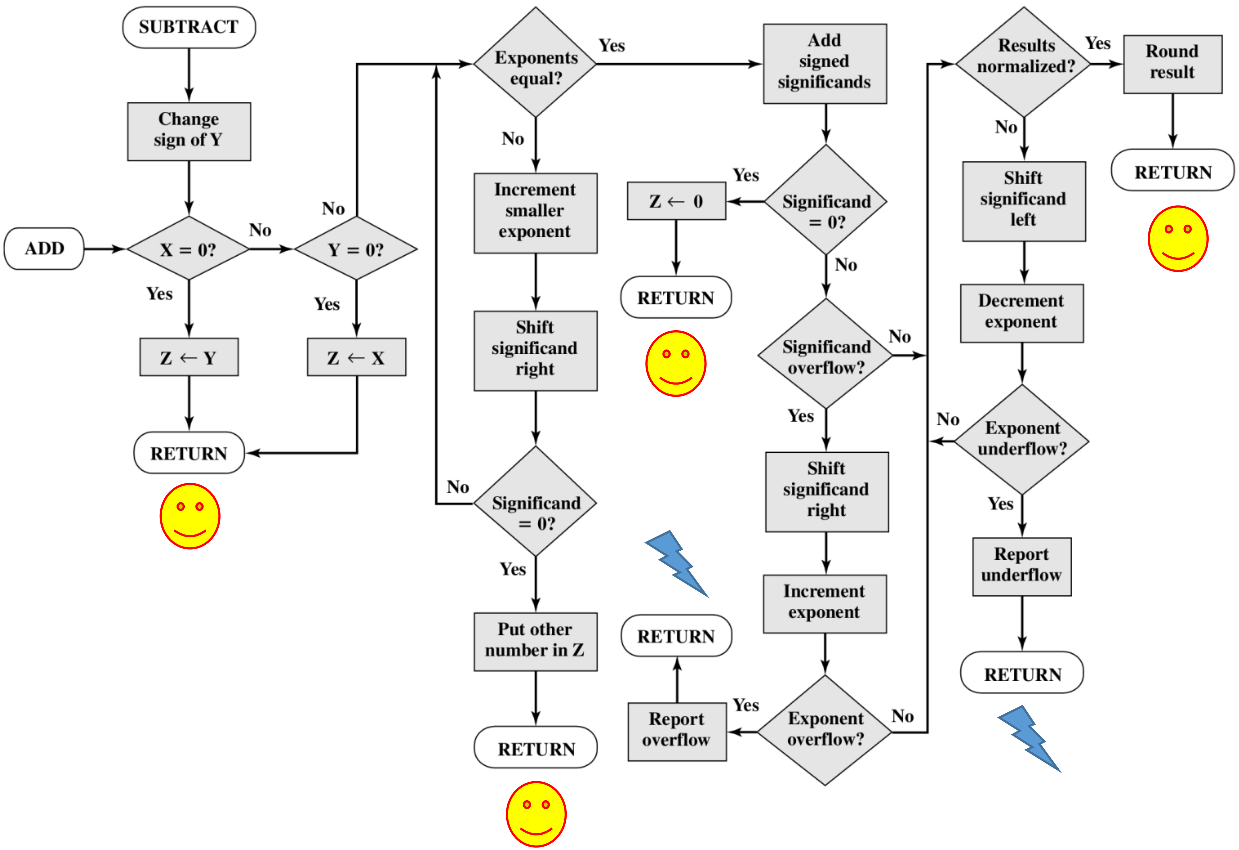
Addition and Subtraction

- It is necessary to ensure that both operands have the same exponent value

$$X + Y = (X_S \times B^{X_E - Y_E} + Y_S) \times B^{Y_E}$$
$$X - Y = (X_S \times B^{X_E - Y_E} - Y_S) \times B^{Y_E}$$
$$X_E \leq Y_E$$

- Procedure
  - Check for zeros
  - Align the significands
  - Add or subtract the significands
  - Normalize the result

Addition and Subtraction (cont.)



Addition and Subtraction (cont.)

- Exponent overflow
  - A positive exponent exceeds the maximum possible exponent value
  - Designated as $+\infty$ or $-\infty$
- Exponent underflow
  - Negative exponent is less than the minimum possible exponent value
  - Reported as 0

Addition and Subtraction (cont.)

- Significand over flow
  - The addition of two significands of the same sign may result in a carry of the most significant bit
  - Fixed by realignment
- Significand under flow
  - In the process of aligning significands, digits may flow off the right end of the significand
  - Some form of rounding is required

Sign Magnitude Addition

- If two operands have same sign, do addition; otherwise, do subtraction
  - Do addition: add directly
    - If the highest bit has carry, overflow
    - Sign is same to addend
  - Do subtraction: add the complement of the second operand
    - If the highest bit has carry, correct (sign is same to minuend)
    - Otherwise, calculate its complement (sign is minus of minuend)

[明鑫, 171250553]

Sign Magnitude Addition (cont.)

- Examples

0.8125+0.625 = 1.4375

```
   1101
+  1010
-------
  10111
```

0.4375

0.625-0.8125=-0.1875

```
   1010
+  0011
-------
   1101
   ↓
   0011
```

- 0.1875

0.8125-0.625 = 0.1875

```
   1101
+  0110
-------
  10011
```

+ 0.1875

Addition and Subtraction (cont.)

- Examples (cont.)

```
0.5 – (-0.4375) = 0.9375
0.5         0 01111110 000…00 (23)
-0.4375     1 01111101 110…00 (21)

01111110 – 01111101 = 01111110 + 10000011 = 00000001

      1 0000…00
    + 0 1110…00
    -----------
      1 1110…00

0 01111110 1110…00 (20)
```

[沈佳楠, 121250118]

Addition and Subtraction (cont.)

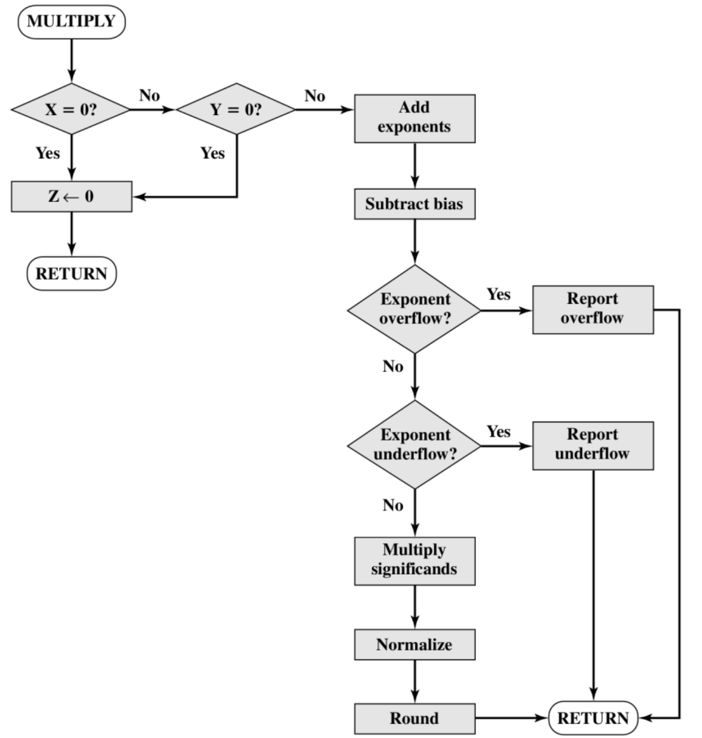
- Examples (cont.)

```
0.5 + (-0.4375) = 0.0625
0.5         0 01111110 000…00 (23)
-0.4375     1 01111101 110…00 (21)

01111110 – 01111101 = 01111110 + 10000011 = 00000001

      1 0000…00
  +   1 0010…00
  -------------
    1 0 0010…00

0 01111011 000…00 (23)
```

[沈佳楠, 121250118]

Multiplication

- If either operand is 0, 0 is reported as the result
- Subtract a bias for the sum of exponents
- Multiple the significands
- Result is normalized and rounded
  - May cause exponent overflow



Multiplication (cont.)

- Example

0.5 × 0.4375 = 0.21875

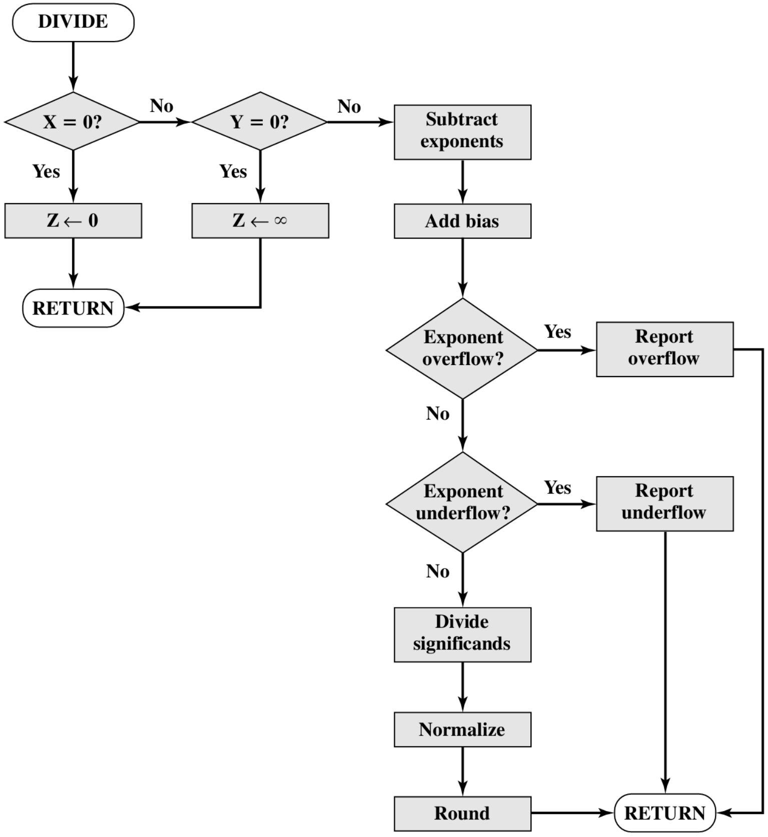
```
   01111110
+  01111101
-----------
   11111011
-  01111111
-----------
   01111100
```

			product	Y
initial			0000…00	1110…00
	0	->	0000…00	01110…0
			……	
	0	->	0000…00	00…0111
	1	+	1000…00	00…0111
		->	0100…00	00…0011
	1	+	1100…00	00…0011
		->	0110…00	00…0001
	1	+	1110…00	00…0001
		->	0111…00	00…0000

0 01111100 110…00 (21)

Division

- If the divisor is 0, an error report is issued, or the result is set to infinity
- A dividend of 0 results in 0
- The divisor exponent is subtracted from the dividend exponent and added bias back
- Divide the significands
- Result is normalized and rounded



Division (cont.)

- Example

0.4375/0.5=0.875

```
   01111101
-  01111110
-----------
   11111111
+  01111111
-----------
   01111110
```

0 01111110 110…00 (21)

		remainder	quotient
	divisor	1000…00	
initial		1110…00	00…0000
enough	-	0110…00	00…0000 1
	<-	1100…00	00…0001
enough	-	0100…00	00…0001 1
	<-	1000…00	00…0011
enough	-	0000…00	00…0011 1
	<-	0000…00	00…0111
not		0000…00	00…0111 0
	<-	0000…00	0…01110
		……	
not		0000…00	01110…0 0
	<-	0000…00	11100…0

Precision Consideration

- Guard bits
  - The length of the register is almost always greater than the length of the significand plus an implied bit
  - The register contains additional bits, called **guard bits**
  - They are used to pad out the right end of the significand with 0s

$$x = 1.00\ldots00 \times 2^1, y = 1.11\ldots11 \times 2^0$$

$$\begin{aligned} x &= 1.000\ldots\ldots00 \times 2^1 \\ \underline{-y} &= \underline{0.111\ldots\ldots11} \times 2^1 \\ z &= 0.000\ldots\ldots01 \times 2^1 \\ &= 1.000\ldots\ldots00 \times 2^{-22} \end{aligned}$$

without guard bits

$$\begin{aligned} x &= 1.000\ldots\ldots00\ 0000 \times 2^1 \\ \underline{-y} &= \underline{0.111\ldots\ldots11\ 1000} \times 2^1 \\ z &= 0.000\ldots\ldots00\ 1000 \times 2^1 \\ &= 1.000\ldots\ldots00\ 0000 \times 2^{-23} \end{aligned}$$

with guard bits

Precision Consideration (cont.)

- Rounding
  - The result of any operation on the significands is generally stored in a longer register
  - When the result is put back into the floating-point format, the extra bits must be disposed of
    - Round to nearest: to the nearest representable number
    - Round toward $+\infty$: up toward plus infinity
    - Round toward $-\infty$: down toward negative infinity
    - Round toward 0: toward zero

Precision Consideration (cont.)

- Examples
  - Assume represent a floating number with 16 bits, in which 1 bit is for sign, 9 bits are for significand, and 6 bits are for exponent (bias is 31)
  - Represent 652.13 and -7.48

+ 652.13	0	101000 010001100	+ 652.0
-7.48	1	100001 110111101	- 7.4765625

Precision Consideration (cont.)

- Examples (cont.)
  - Assume ALU has 16 bits, calculate the followings without and with guard bits
    - 652.13 + (-7.48)
    - 652.13 -(-7.48)

Precision Consideration (cont.)

- Examples (cont.): 652.13 + (-7.48) = 644.65

```
101000 - 100001 = 000111
```

```
    1 010001100
-   0 000000111
---------------
    1 010000101
```

```
    1 010001100 000000
-   0 000000111 011110
----------------------
    1 010000100 100010
```

```
0 101000 010000101
```

645.0

```
0 101000 010000100
```

644.0

Precision Consideration (cont.)

- Examples (cont.): 652.13 - (-7.48) = 659.61

```
101000 - 100001 = 000111
```

```
    1 010001100
+   0 000000111
---------------
    1 010010011
```

```
    1 010001100 000000
+   0 000000111 011110
----------------------
    1 010010011 011110
```

```
0 101000 010010011
```

659.0

```
0 101000 010010011
```

659.0

[薛家豪，121250185]

Summary

- Floating-point arithmetic operations
  - Addition
  - Subtraction
  - Multiplication
  - Division
- Precision Consideration

Thank You

rentw@nju.edu.en

南京大學
NANJING UNIVERSITY