Computer Organization & Architecture

Floating Point Arithmetic

In this lecture

a) Floating Point Representation

b) Addition

c) Subtraction

d) Division

e) Multiplication

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)
- Normalize significand: 1.0 ≤ |significand| < 2.0
  - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
  - Significand is Fraction with the “1.” restored
- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Single: Bias = 127; Double: Bias = 1203

Single-Precision Range

- Exponents 00000000 and 11111111 reserved
- Smallest value
  - Exponent: 00000001
    $\Rightarrow$ actual exponent = 1 – 127 = –126
  - Fraction: 000…00 $\Rightarrow$ significand = 1.0
  - $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$
- Largest value
  - exponent: 11111110
    $\Rightarrow$ actual exponent = 254 – 127 = +127
  - Fraction: 111…11 $\Rightarrow$ significand ≈ 2.0
  - $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

Double-Precision Range

- Exponents 0000…00 and 1111…11 reserved
- Smallest value
  - Exponent: 00000000001
    $\Rightarrow$ actual exponent = 1 – 1023 = –1022
  - Fraction: 000…00 $\Rightarrow$ significand = 1.0
  - $\pm 1.0 \times 2^{-1022} \approx \pm 2.2 \times 10^{-308}$
- Largest value
  - Exponent: 11111111110
    $\Rightarrow$ actual exponent = 2046 – 1023 = +1023
  - Fraction: 111…11 $\Rightarrow$ significand ≈ 2.0
  - $\pm 2.0 \times 2^{+1023} \approx \pm 1.8 \times 10^{+308}$

Floating-Point Precision

- Relative precision
  - all fraction bits are significant
  - Single: approx 2^{-23}
    - Equivalent to $23 \times \log_{10}2 \approx 23 \times 0.3 \approx 6$ decimal digits of precision
  - Double: approx 2^{-52}
    - Equivalent to $52 \times \log_{10}2 \approx 52 \times 0.3 \approx 16$ decimal digits of precision

Floating-Point Example

- Represent –0.75
  - $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$
  - S = 1
  - Fraction = $1000\ldots00_2$
  - Exponent = –1 + Bias
    - Single: $-1 + 127 = 126 = 01111110_2$
    - Double: $-1 + 1023 = 1022 = 01111111110_2$
- Single: 1011111101000…00
- Double: 1011111111101000…00

Floating-Point Example

What number is represented by the single-precision float

11000000101000...00

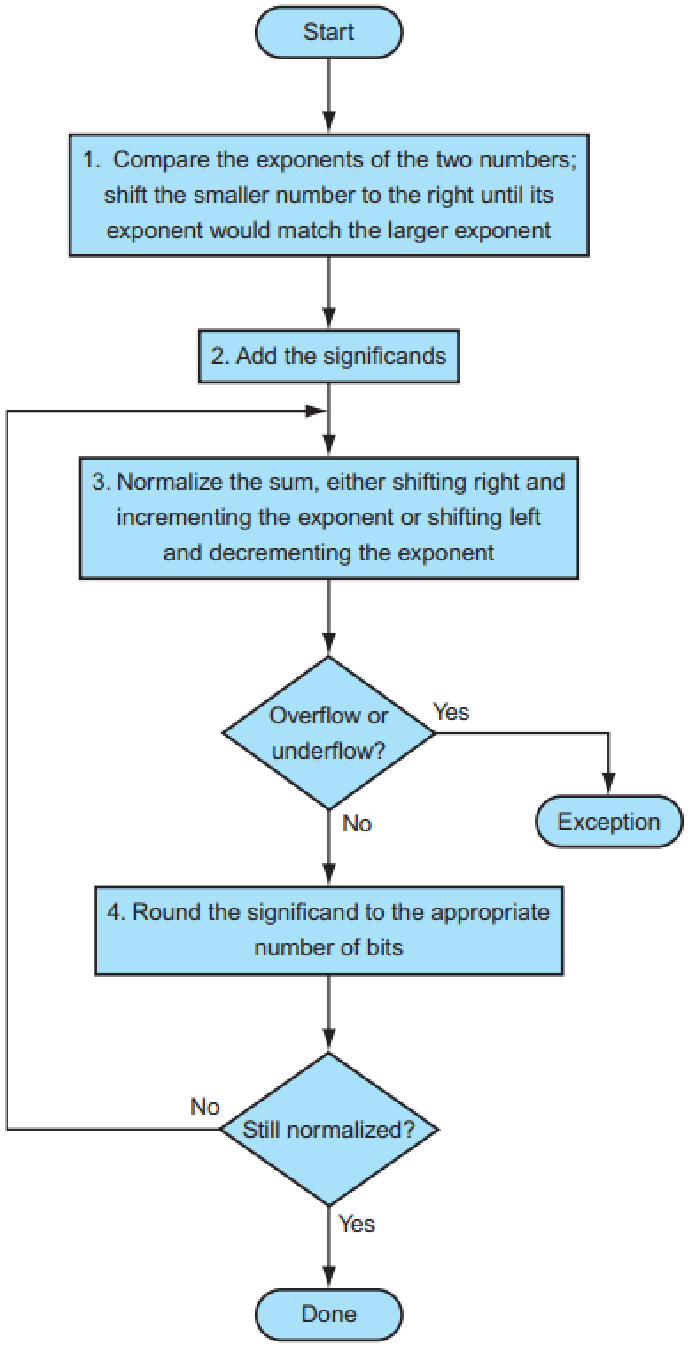
- S = 1
- Fraction = $01000...00_2$
- Exponent = 10000001_2 = 129

- $x = (-1)^1 \times (1 + .01_2) \times 2^{(129 - 127)}$

  $= (-1) \times 1.25 \times 2^2$

  $= -5.0$

Floating Point Addition



Example 1 (5.75 + 14)

5.75 + 14 = 19.75

IEEE representation of ‘5.75’:

0 10000001 01110000000000000000000

[0—129—0.437500]

IEEE representation of ‘14’:

0 10000010 11000000000000000000000

[0—130—0.750000]

Preliminary exponent: MAX(129, 130) = 130

Example 1 Continue...

Mantissa:

- Shift 1.01110. . . by 130−129=1 bit
- the signs are identical → ADD
- add 0.10111 and 1.11000:

$$
\begin{array}{r}
0.\,1\,0\,1\,1\,1 \\
+\,1.\,1\,1\,0\,0\,0 \\
1\,0.\,0\,1\,1\,1\,1
\end{array}
$$

- Normalize exponent and mantissa:

1.001111 × 2(131−127)

Example 1 Continue…

IEEE representation of result:

0 10000011 00111100000000000000000

[0—131—0.234375]

Example 2 (5.75 - 14)

5.75 - 14 = -8.25

IEEE representation of ‘5.75’:

0 10000001 01110000000000000000000

[0—129—0.437500]

IEEE representation of ‘14’:

0 10000010 11000000000000000000000

[0—130—0.750000]

Preliminary exponent: MAX(129, 130) = 130

Example 2 Continue…

Mantissa:

- Shift 1.01110. . . by 130−129=1 bit
- the signs are different → SUB

Example 2 Continue…

From 0.10111 subtract 1.11000:

$$
\begin{array}{lr}
 & 0\,0.\,1\,0\,1\,1\,1 \\
\text{2's-complement}\rightarrow & +1\,0.\,0\,1\,0\,0\,0 \\
 & 1\,0.\,1\,1\,1\,1\,1
\end{array}
$$

sign of difference is negative

2’s-complement of result: 0 1. 0 0 0 0 1

- positive S_a, thus result negative
- no normalization required

Example 2 (5.75 - 14)

IEEE representation of result:

1 10000010 00001000000000000000000

[1—130—0.031250]

Example 3 (2.375 ÷ 8.25)

2.375 ÷ 8.25 = 0.287

IEEE representation of ‘2.375’:

0 10000000 00110000000000000000000

[0—128—0.187500]

IEEE representation of ‘8.25’:

0 10000010 00001000000000000000000

[0—130—0.031250]

Preliminary exponent:128 − 130 + 127 = 125

Example 3 Continue…

Mantissa:

```
                1. 0 0 1 0 0 1 1 0 1 1 ...
1.00001... )    1. 0 0 1 1 0 ...
                1. 0 0 0 0 1
                      1 0 1 0 0 0
                      1 0 0 0 0 1
                          1 1 1 0 0 0
                          1 0 0 0 0 1
                            1 0 1 1 1 0
                            1 0 0 0 0 1
                                1 1 0 1 0 0
```

IEEE representation of result: …

0 01111101 00100110110010011011001

[0—125—0.151515]

Binary Floating-Point Multiplication

Let's try multiplying the numbers 0.5_{ten} and -0.4375_{ten}:

In binary, the task is multiplying $1.000_{two} \times 2^{-1}$ by $-1.110_{two} \times 2^{-2}$.

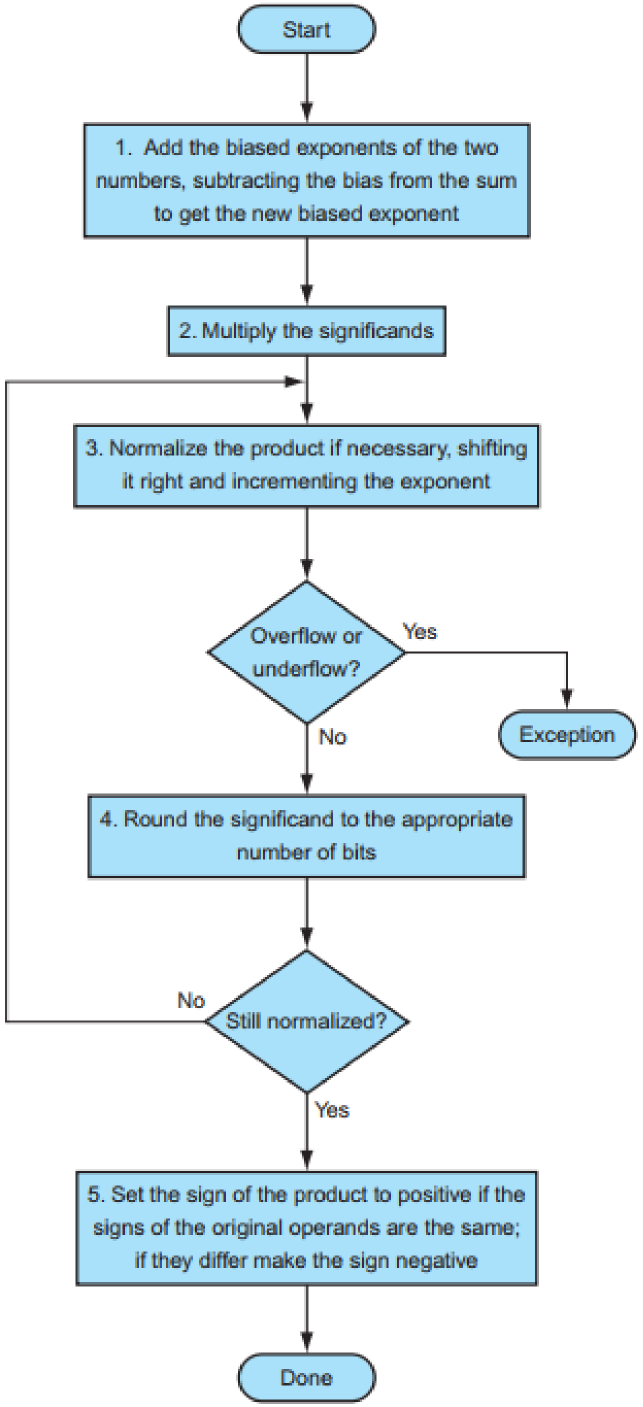
Step 1. Adding the exponents without bias:

$$-1 + (-2) = -3$$

or, using the biased representation:

$$(-1 + 127) + (-2 + 127) - 127 = (-1 - 2) + (127 + 127 - 127)$$
$$= -3 + 127 = 124$$

Binary Floating-Point Multiplication



Binary Floating-Point Multiplication

Step 2. Multiplying the significands:

$$
\begin{array}{r}
1.000_{two} \\
\times \quad 1.110_{two} \\
\hline
0000 \\
1000 \\
1000 \\
1000 \\
\hline
1110000_{two}
\end{array}
$$

The product is $1.110000_{two} \times 2^{-3}$, but we need to keep it to 4 bits, so it is $1.110_{two} \times 2^{-3}$.

Binary Floating-Point Multiplication

Step 3. Now we check the product to make sure it is normalized, and then check the exponent for overflow or underflow. The product is already normalized and, since $127 \geq -3 \geq -126$, there is no overflow or underflow. (Using the biased representation, $254 \geq 124 \geq 1$, so the exponent fits.)

Step 4. Rounding the product makes no change:

$$1.110_{two} \times 2^{-3}$$

Binary Floating-Point Multiplication

Step 5. Since the signs of the original operands differ, make the sign of the product negative. Hence, the product is

$$-1.110_{\text{two}} \times 2^{-3}$$

Converting to decimal to check our results:

$$\begin{aligned}-1.110_{\text{two}} \times 2^{-3} &= -0.001110_{\text{two}} = -0.00111_{\text{two}}\\ &= -7/2^5{}_{\text{ten}} = -7/32_{\text{ten}} = -0.21875_{\text{ten}}\end{aligned}$$

The product of 0.5_{ten} and -0.4375_{ten} is indeed -0.21875_{ten}.

Block diagram of an arithmetic unit dedicated to Floating-point addition