# COMPUTER ORGANIZATION

## Unit-IV PPT Slides

**Text Books:** (1) Computer Systems Architecture by M. Morris Mano

(2) Computer Organization by Carl Hamacher

# COMPUTER ARITHMETIC

## INDEX

## UNIT-VI PPT SLIDES

# ARITHMETIC OPERATIONS

- Arithmetic operations involve adding, subtracting, multiplying and dividing.
- We can apply these operations to integers and floating-point numbers.

# Arithmetic operations on integers

- All arithmetic operations such as addition, subtraction, multiplication and division can be applied to integers.
- Although multiplication (division) of integers can be implemented using repeated addition (subtraction), the procedure is not efficient.
- There are more efficient procedures for multiplication and division, such as Booth procedures, but these are beyond the scope of this book.
- For this reason, we only discuss addition and subtraction of integers here.

Two’s complement integers

When the subtraction operation is encountered, the computer simply changes it to an addition operation, but makes two’s complement of the second number. In other words:

$$\boxed{A - B \leftrightarrow A + (\overline{B} + 1)}$$

Where $\overline{B}$ is the one’s complement of B and

($\overline{B}$ + 1) means the two’s complement of B

- We should remember that we add integers column by column.
- The following table shows the sum and carry (C).

| *Column* | *Carry* | *Sum* |
|---|---|---|
| Zero 1s | 0 | 0 |
| One 1 | 0 | 1 |
| Two 1s | 1 | 0 |
| Three 1s | 1 | 1 |

Sign-and-magnitude integers

- Addition and subtraction for integers in sign-and-magnitude representation looks very complex.
- We have four different combinations of signs (two signs, each of two values) for addition and four different conditions for subtraction.
- This means that we need to consider eight different situations.

Addition/subtraction of integers in sign-and-magnitude format

- Eight situations for sign-and-magnitude addition/subtraction

| Operation | ADD Magnitudes | SUBTRACT Magnitudes | | |
|---|---|---|---|---|
| | | $A_M > B_M$ | $A_M < B_M$ | $A_M = B_M$ |
| (+A) + (+B) | $+ (A_M + B_M)$ | | | |
| (+A) + (-B) | | $+ (A_M - B_M)$ | $- (B_M - A_M)$ | $+ (A_M - B_M)$ |
| (-A) + (+B) | | $- (A_M - B_M)$ | $+ (B_M - A_M)$ | $+ (A_M - B_M)$ |
| (-A) + (-B) | $- (A_M + B_M)$ | | | |
| (+A) - (+B) | | $+ (A_M - B_M)$ | $- (B_M - A_M)$ | $+ (A_M - B_M)$ |
| (+A) - (-B) | $+ (A_M + B_M)$ | | | |
| (-A) - (+B) | $- (A_M + B_M)$ | | | |
| (-A) - (-B) | | $- (A_M - B_M)$ | $+ (B_M - A_M)$ | $+ (A_M - B_M)$ |

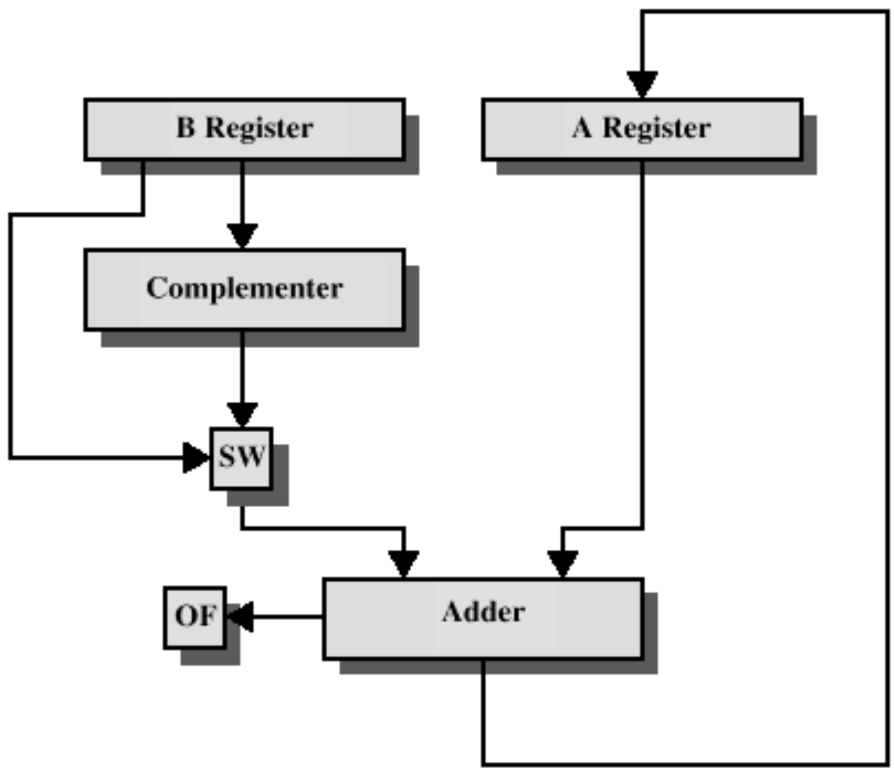
# Addition and Subtraction

- Normal binary addition
- Monitor sign bit for overflow

- Take twos compliment of substahend and add to minuend
  - i.e. a - b = a + (-b)

- So we only need addition and complement circuits

# Hardware for Addition and Subtraction




OF = overflow bit
SW = Switch (select addition or subtraction)

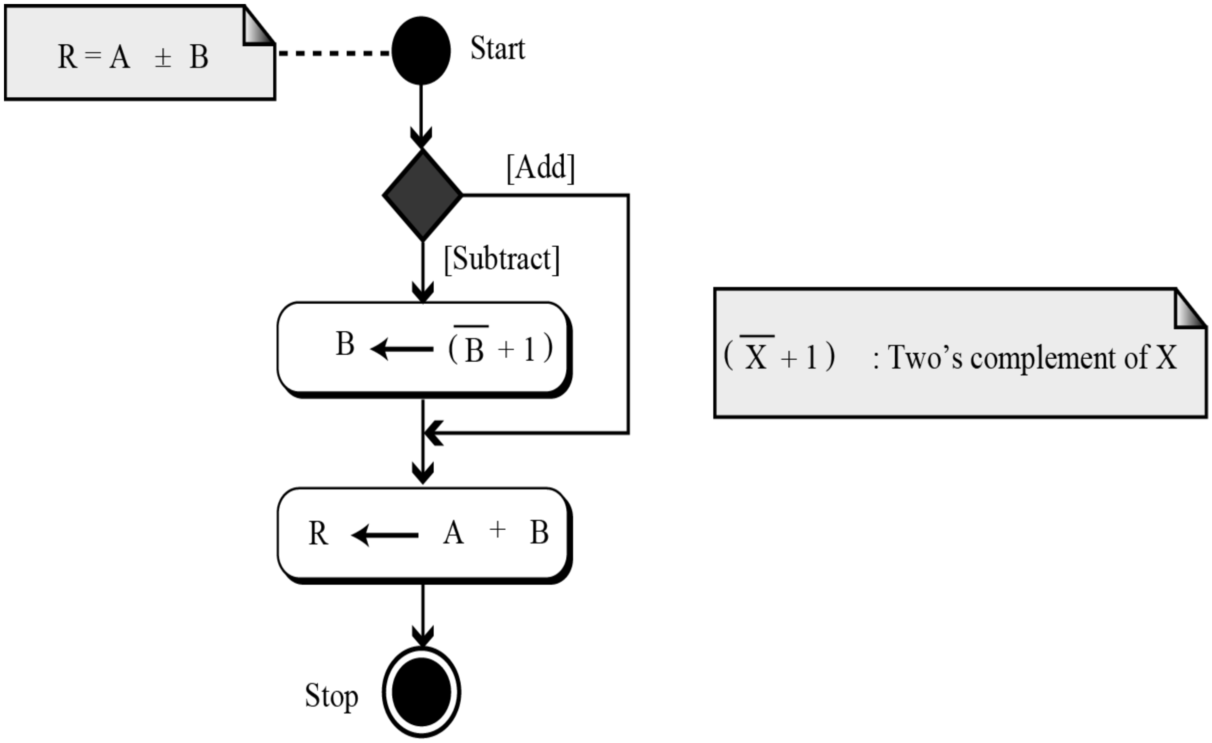



Figure 4.6 Addition and subtraction of integers in two's complement format

## Example 4.16

Two integers A and B are stored in two's complement format. Show how B is added to A.

$A = (00010001)_2 \qquad B = (00010110)_2$

### Solution

- The operation is adding.
- A is added to B and the result is stored in R.
- (+17) + (+22) = (+39).

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | | | | | | Carry |
| | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | A |
| + | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | B |
| | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | R |

## Example 4.17

Two integers A and B are stored in two's complement format. Show how B is added to A.

$$A = (00011000)_2 \qquad B = (11101111)_2$$

### Solution

- The operation is adding.
- A is added to B and the result is stored in R.
- (+24) + (−17) = (+7).

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | | | | | Carry |
| | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | A |
| + | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | B |
| | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | R |

## Example 4.18

Two integers A and B are stored in two's complement format. Show how B is subtracted from A.

$$A = (00011000)_2 \qquad B = (11101111)_2$$

### Solution

- The operation is subtracting.
- A is added to $(\overline{B} + 1)$ and the result is stored in R.
- (+24) – (–17) = (+41).

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | | | | | | Carry |
| | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | A |
| + | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | $(\overline{B} + 1)$ |
| | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | R |

## Example 4.19

Two integers A and B are stored in two's complement format. Show how B is subtracted from A.

$$A = (11011101)_2 \qquad B = (00010100)_2$$

### Solution

- The operation is subtracting.
- A is added to $(\overline{B} + 1)$ and the result is stored in R.
- $(-35) - (+20) = (-55)$.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | | | | Carry |
| | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | A |
| + | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | $(\overline{B} + 1)$ |
| | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | R |

# Alternative method for Addition/subtraction of integers in sign-and-magnitude format

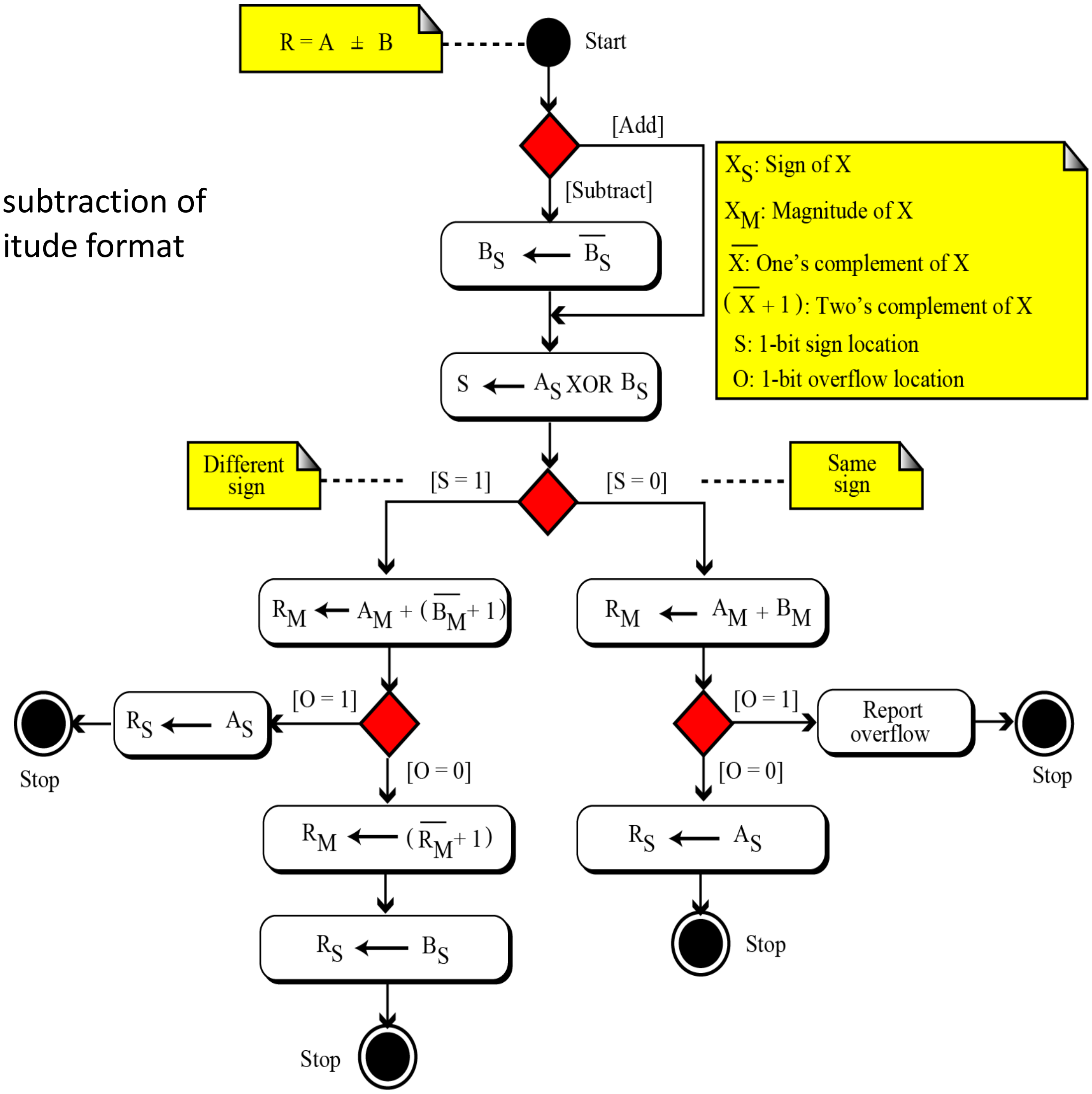



Figure 4.7 Addition and subtraction of integers in sign-and-magnitude format

## Example 4.20

Two integers A and B are stored in two's complement format. Show how B is added to A.

$$A = (01111111)_2 \qquad B = (00000011)_2$$

Solution

- The operation is adding.
- A is added to B and the result is stored in R.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | Carry |
| | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | A |
| + | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | B |
| | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | R |

- We expect the result to be 127 + 3 = 130, but the answer is −126.
- The error is due to overflow, because the expected answer (+130) is not in the range −128 to +127.

## Example 4.22

Two integers A and B are stored in sign-and-magnitude format. Show how B is added to A.

$$A = (0\ 0010001)_2 \qquad B = (1\ 0010110)_2$$

### Solution

- The operation is adding: the sign of B is not changed.
- $S = A_S \text{ XOR } B_S = 1$; $R_M = A_M + (\overline{B_M} + 1)$.
- Since there is no overflow, we need to take the two's complement of $R_M$.
- The sign of R is the sign of B.
- $(+17) + (-22) = (-5)$.

| | | No overflow | | | | | | | | Carry |
|---|---|---|---|---|---|---|---|---|---|---|
| $A_S$ | 0 | | 0 | 0 | 1 | 0 | 0 | 0 | 1 | $A_M$ |
| $B_S$ | 1 | + | 1 | 1 | 0 | 1 | 0 | 1 | 0 | $(\overline{B_M} + 1)$ |
| | | | 1 | 1 | 1 | 1 | 0 | 1 | 1 | $R_M$ |
| $R_S$ | 1 | | 0 | 0 | 0 | 0 | 1 | 0 | 1 | $R_M = (\overline{R_M} + 1)$ |

4.18

## Example 4.23

Two integers A and B are stored in sign-and-magnitude format. Show how B is subtracted from A.

$$A = (1\ 1010001)_2 \qquad B = (1\ 0010110)_2$$

### Solution

- The operation is subtracting: $S_B = \overline{S_B}$.
- $S = A_S \text{ XOR } B_S = 1$, $R_M = A_M + (\overline{B_M} + 1)$.
- Since there is an overflow, the value of $R_M$ is final.
- The sign of R is the sign of A.
- (−81) − (−22) = (−59).

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Overflow → | 1 | | | | | | | | Carry |
| $A_S$ | 1 | | | 1 | 0 | 1 | 0 | 0 | 0 | 1 | $A_M$ |
| $B_S$ | 1 | | + | 1 | 1 | 0 | 1 | 0 | 1 | 0 | $(\overline{B_M}+1)$ |
| $R_S$ | 1 | | | 0 | 1 | 1 | 1 | 0 | 1 | 1 | $R_M$ |

## Overflow detection in two's complement addition

- Overflow: When the signs of the addends are the same, and the sign of the result is different
- You will never get overflow when adding 2 numbers of opposite signs

When we do arithmetic operations on numbers in a computer, we should remember that each number and the result should be in the range defined by
the bit allocation.

# Multiplication

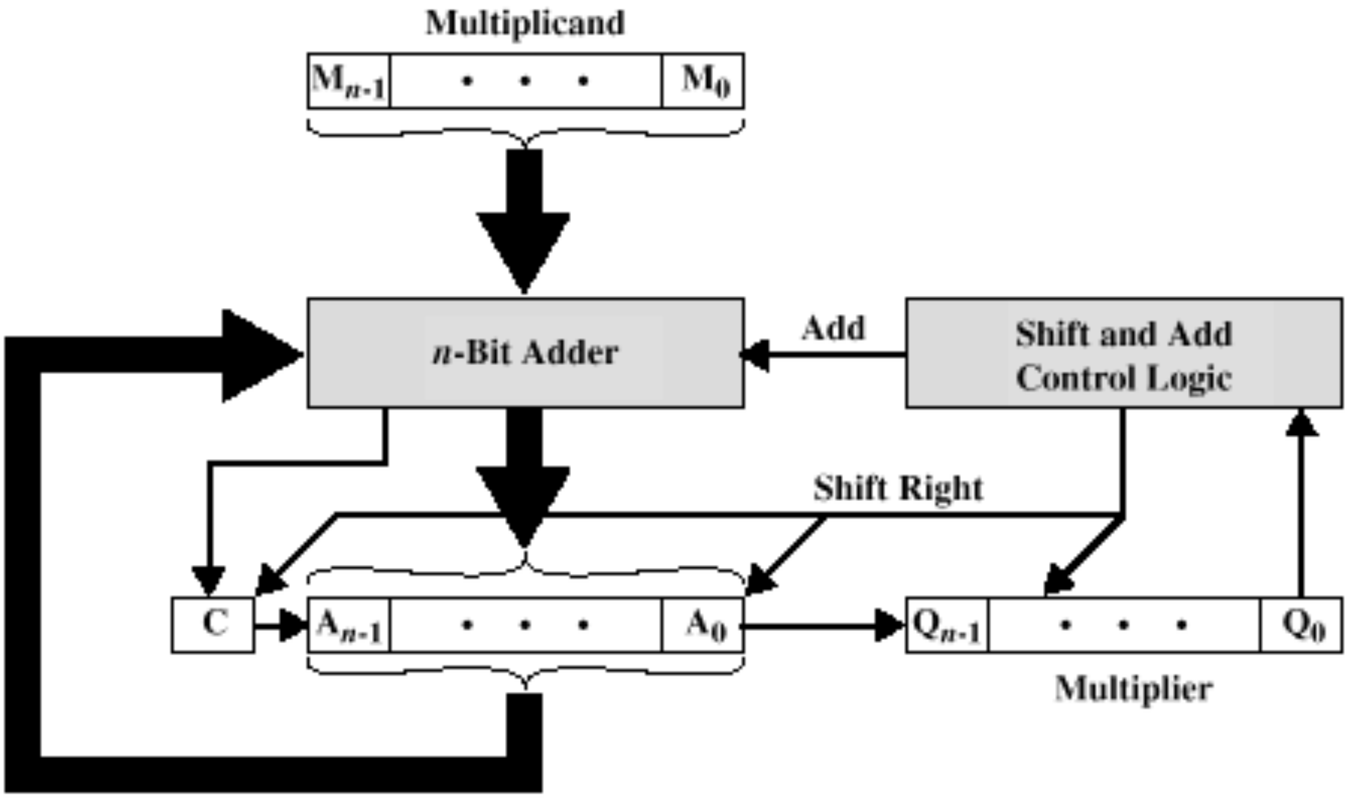
- Complex
- Work out partial product for each digit
- Take care with place value (column)
- Add partial products

# Multiplication Example

```
      1011   Multiplicand (11 dec)
    x 1101   Multiplier     (13 dec)
      1011   Partial products
     0000    Note: if multiplier bit is 1 copy
    1011       multiplicand (place value)
   1011        otherwise zero
  10001111   Product (143 dec)
```

Note: need double length result

# Unsigned Binary Multiplication

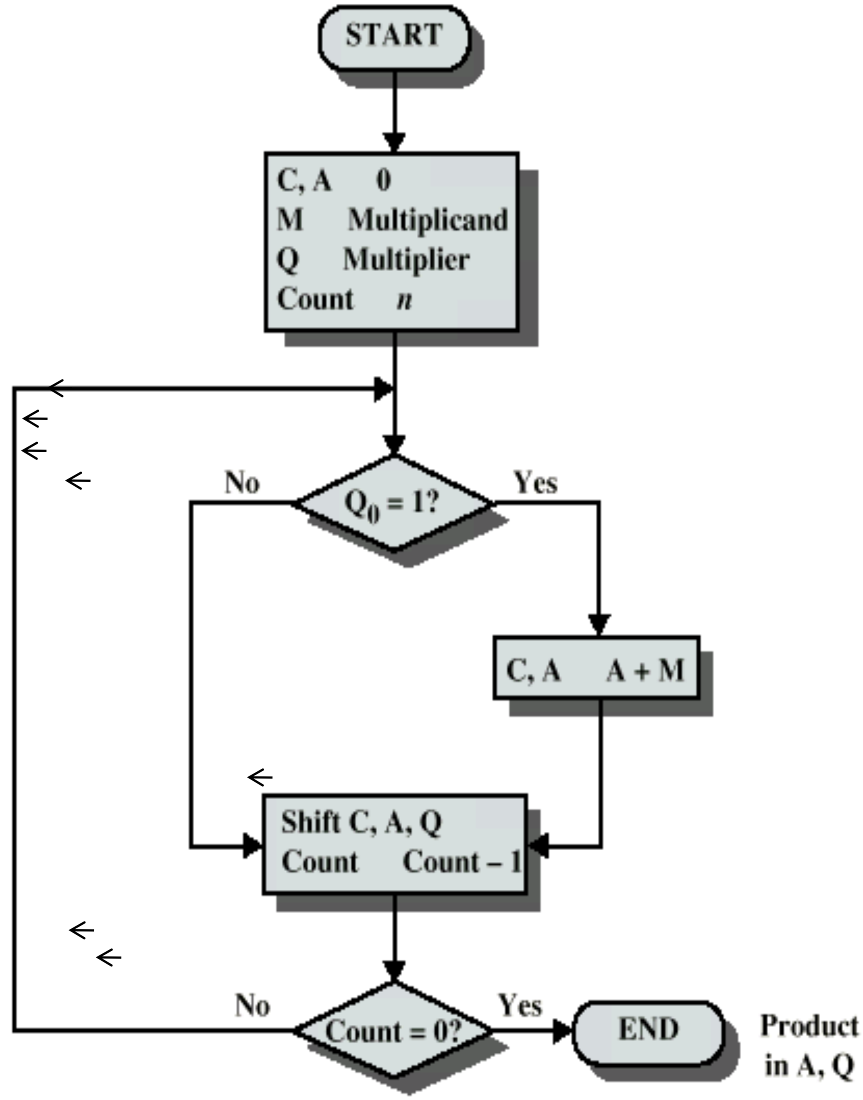



(a) Block Diagram

# Multiplication of Unsigned Binary Integers

```
    1011     Multiplicand (11)
   ×1101     Multiplier (13)
   -----
    1011   }
   0000    }
  1011     } Partial products
 1011      }
--------
10001111     Product (143)
```

Flowchart for Unsigned Binary Multiplication



# Execution of Example

| C | A | Q | M | | |
|---|---|---|---|---|---|
| 0 | 0000 | 1101 | 1011 | Initial Values | |
| 0 | 1011 | 1101 | 1011 | Add | First |
| 0 | 0101 | 1110 | 1011 | Shift | Cycle |
| 0 | 0010 | 1111 | 1011 | Shift | Second Cycle |
| 0 | 1101 | 1111 | 1011 | Add | Third |
| 0 | 0110 | 1111 | 1011 | Shift | Cycle |
| 1 | 0001 | 1111 | 1011 | Add | Fourth |
| 0 | 1000 | 1111 | 1011 | Shift | Cycle |

# Multiplication of Two
# Unsigned 4-Bit Integers Yielding an 8-Bit Result

$$
\begin{array}{r l}
1011 & \\
\times 1101 & \\
\hline
00001011 & 1011 \times 1 \times 2^{0} \\
00000000 & 1011 \times 0 \times 2^{1} \\
00101100 & 1011 \times 1 \times 2^{2} \\
01011000 & 1011 \times 1 \times 2^{3} \\
\hline
10001111 &
\end{array}
$$

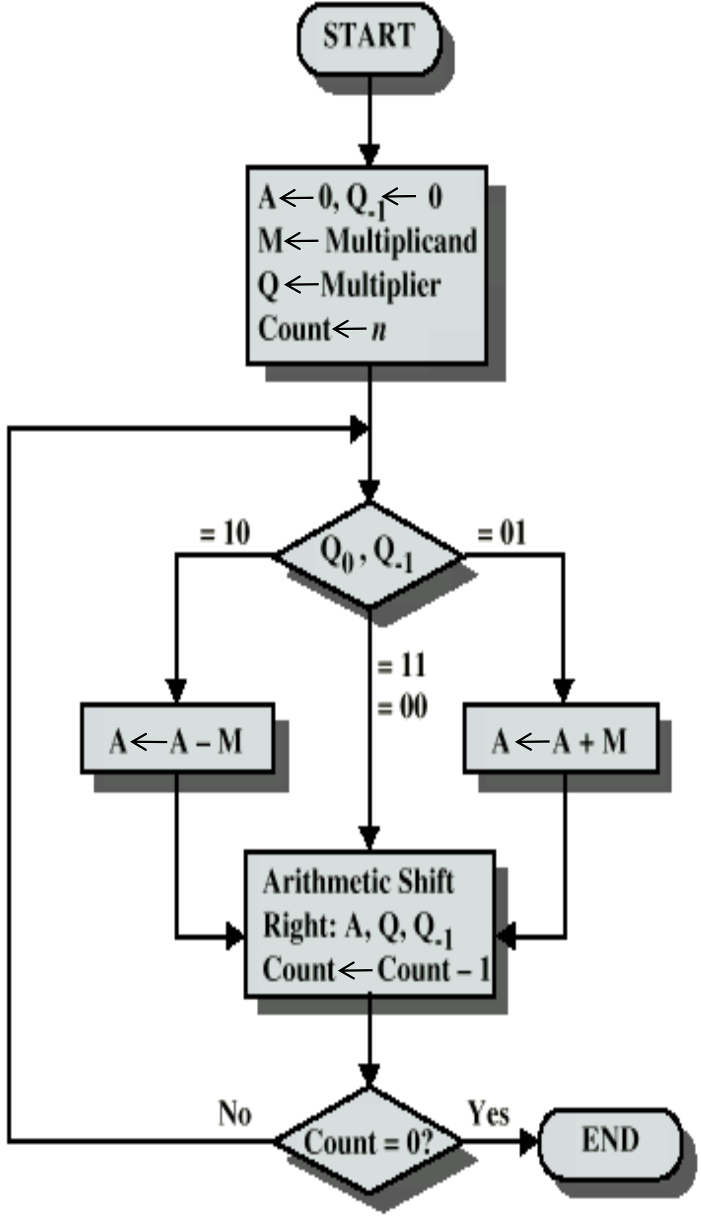
## Comparison of Multiplication of Unsigned and Twos Complement Integers

| | | | |
|---|---|---|---|
| 1001 | (9) | 1001 | (−7) |
| x0011 | (3) | x0011 | (3) |
| 00001001 | $1001 \times 2^0$ | 11111001 | $(-7) \times 2^0 = (-7)$ |
| 00010010 | $1001 \times 2^1$ | 11110010 | $(-7) \times 2^1 = (-14)$ |
| 00011011 | (27) | 11101011 | (−21) |
| (a) Unsigned integers | | (b) Twos complement integers | |

# Multiplying Negative Numbers

- This does not work!
- Solution 1
  - Convert to positive if required
  - Multiply as above
  - If signs were different, negate answer
- Solution 2
  - Booth's algorithm

# Booth’s Algorithm



# Example of Booth’s Algorithm

| A | Q | $Q_{-1}$ | M | | |
|---|---|---|---|---|---|
| 0000 | 0011 | 0 | 0111 | Initial Values | |
| 1001 | 0011 | 0 | 0111 | A  A - M | First Cycle |
| 1100 | 1001 | 1 | 0111 | Shift | |
| 1110 | 0100 | 1 | 0111 | Shift | Second Cycle |
| 0101 | 0100 | 1 | 0111 | A  A + M | Third Cycle |
| 0010 | 1010 | 0 | 0111 | Shift | |
| 0001 | 0101 | 0 | 0111 | Shift | Fourth Cycle |

# Examples Using Booth's Algorithm

```
    0111
   ×0011       (0)
11111001       1–0
0000000        1–1
000111         0–1
00010101       (21)
```

(a) (7) × (3) = (21)

```
    0111
   ×1101       (0)
11111001       1–0
0000111        0–1
111001         1–0
11101011       (–21)
```

(b) (7) × (–3) = (–21)

```
    1001
   ×0011       (0)
00000111       1–0
0000000        1–1
111001         0–1
11101011       (–21)
```

(c) (–7) × (3) = (–21)

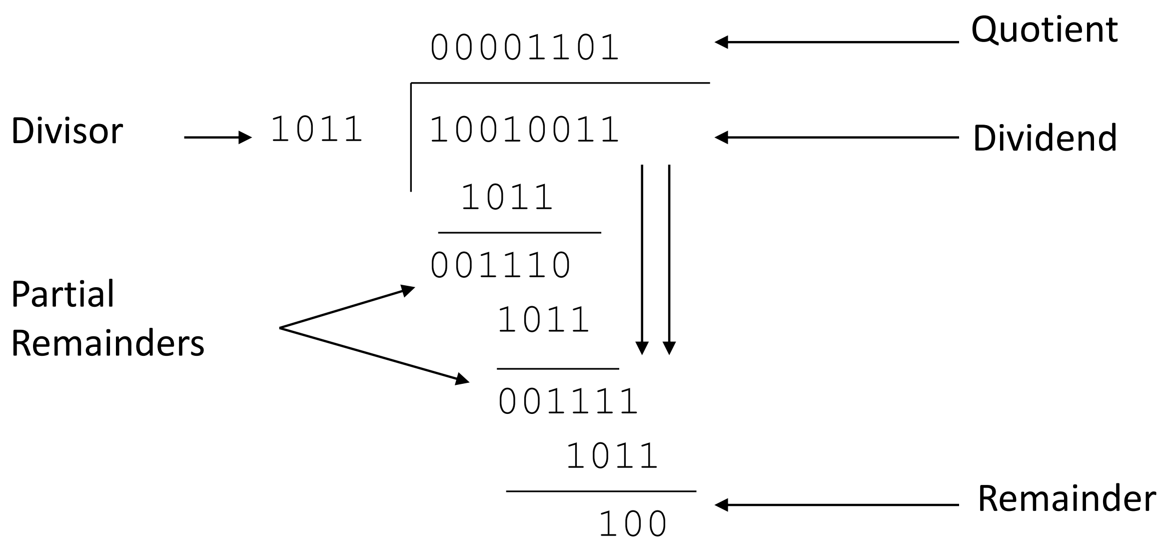
```
    1001
   ×1101       (0)
00000111       1–0
1111001        0–1
000111         1–0
00010101       (21)
```

(d) (–7) × (–3) = (21)

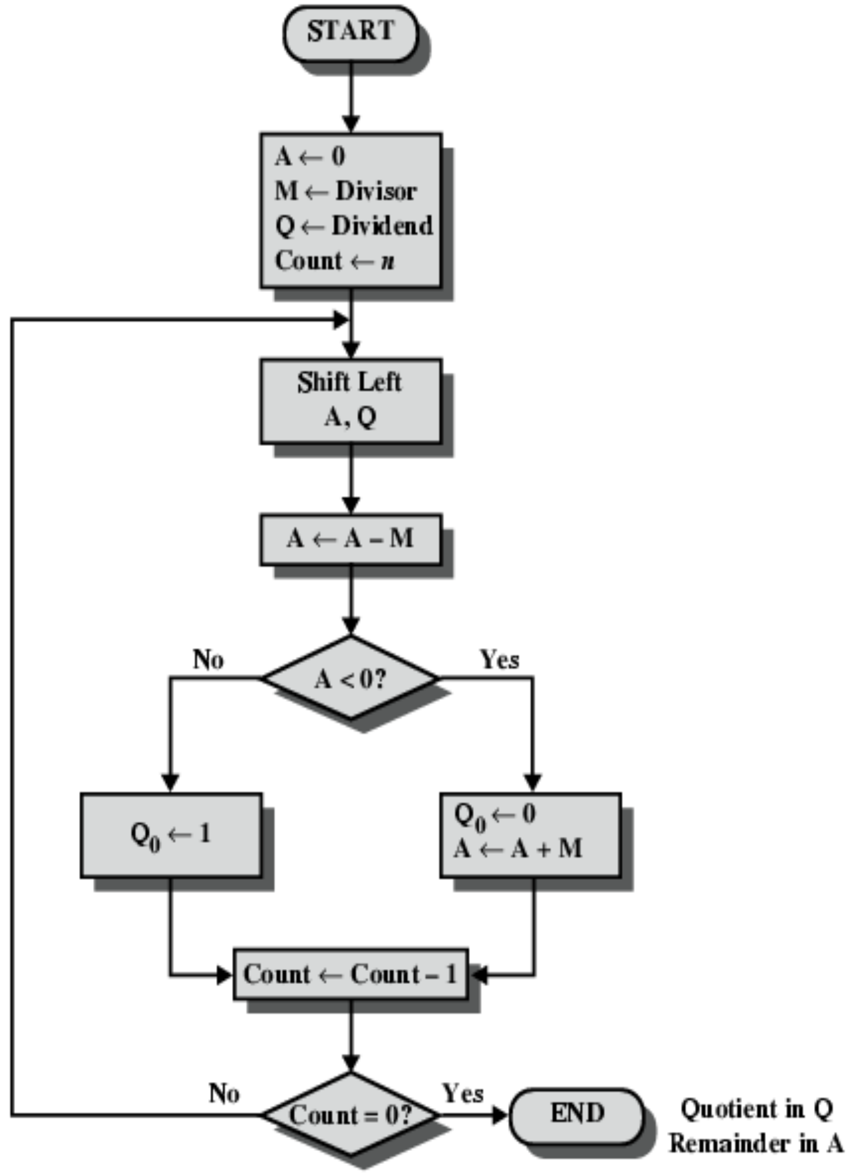
# Division

- More complex than multiplication
- Negative numbers are really bad!
- Based on long division

# Division of Unsigned Binary Integers



# Flowchart for Unsigned Binary Division



# Examples of Twos Complement Division

| A | Q | M = 0011 |
|---|---|---|
| 0000 | 0111 | Initial value |
| 0000 | 1110 | shift |
| 1101 | | subtract |
| 0000 | 1110 | restore |
| 0001 | 1100 | shift |
| 1110 | | subtract |
| 0001 | 1100 | restore |
| 0011 | 1000 | shift |
| 0000 | | subtract |
| 0000 | 1001 | set $Q_0 = 1$ |
| 0001 | 0010 | shift |
| 1110 | | subtract |
| 0001 | 0010 | restore |

(a) (7)/(3)

| A | Q | M = 1101 |
|---|---|---|
| 0000 | 0111 | Initial value |
| 0000 | 1110 | shift |
| 1101 | | add |
| 0000 | 1110 | restore |
| 0001 | 1100 | shift |
| 1110 | | add |
| 0001 | 1100 | restore |
| 0011 | 1000 | shift |
| 0000 | | add |
| 0000 | 1001 | set $Q_0 = 1$ |
| 0001 | 0010 | shift |
| 1110 | | add |
| 0001 | 0010 | restore |

(b) (7)/(–3)

| A | Q | M = 0011 |
|---|---|---|
| 1111 | 1001 | Initial value |
| 1111 | 0010 | shift |
| 0010 | | add |
| 1111 | 0010 | restore |
| 1110 | 0100 | shift |
| 0001 | | add |
| 1110 | 0100 | restore |
| 1100 | 1000 | shift |
| 1111 | | add |
| 1111 | 1001 | set $Q_0 = 1$ |
| 1111 | 0010 | shift |
| 0010 | | add |
| 1111 | 0010 | restore |

(c) (–7)/(3)

| A | Q | M = 1101 |
|---|---|---|
| 1111 | 1001 | Initial value |
| 1111 | 0010 | shift |
| 0010 | | subtract |
| 1111 | 0010 | restore |
| 1110 | 0100 | shift |
| 0001 | | subtract |
| 1110 | 0100 | restore |
| 1100 | 1000 | shift |
| 1111 | | subtract |
| 1111 | 1001 | set $Q_0 = 1$ |
| 1111 | 0010 | shift |
| 0010 | | subtract |
| 1111 | 0010 | restore |

(d) (–7)/(–3)

# Real Numbers

- Numbers with fractions
- Could be done in pure binary
  - $1001.1010 = 2^4 + 2^0 + 2^{-1} + 2^{-3} = 9.625$
- Where is the binary point?
- Fixed?
  - Very limited
- Moving?
  - How do you show where it is?

# Floating Point


(a) Format

- +/- .significand x $2^{exponent}$
- Misnomer
- Point is actually fixed between sign bit and body of mantissa
- Exponent indicates place value (point position)

# Floating Point Examples


(a) Format

$$
\begin{array}{rcl}
1.1010001 \times 2^{10100} & = 0\ 10010011\ 10100010000000000000000 = & 1.638125 \times 2^{20} \\
-1.1010001 \times 2^{10100} & = 1\ 10010011\ 10100010000000000000000 = & -1.638125 \times 2^{20} \\
1.1010001 \times 2^{-10100} & = 0\ 01101011\ 10100010000000000000000 = & 1.638125 \times 2^{-20} \\
-1.1010001 \times 2^{-10100} & = 1\ 01101011\ 10100010000000000000000 = & -1.638125 \times 2^{-20}
\end{array}
$$

(b) Examples

# Signs for Floating Point

- Mantissa is stored in 2s compliment
- Exponent is in excess or biased notation
  - e.g. Excess (bias) 128 means
  - 8 bit exponent field
  - Pure value range 0-255
  - Subtract 128 to get correct value
  - Range -128 to +127

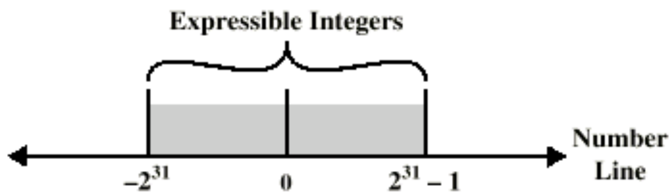
# Normalization

- FP numbers are usually normalized
- i.e. exponent is adjusted so that leading bit (MSB) of mantissa is 1
- Since it is always 1 there is no need to store it
- (c.f. Scientific notation where numbers are normalized to give a single digit before the decimal point
- e.g. $3.123 \times 10^3$)

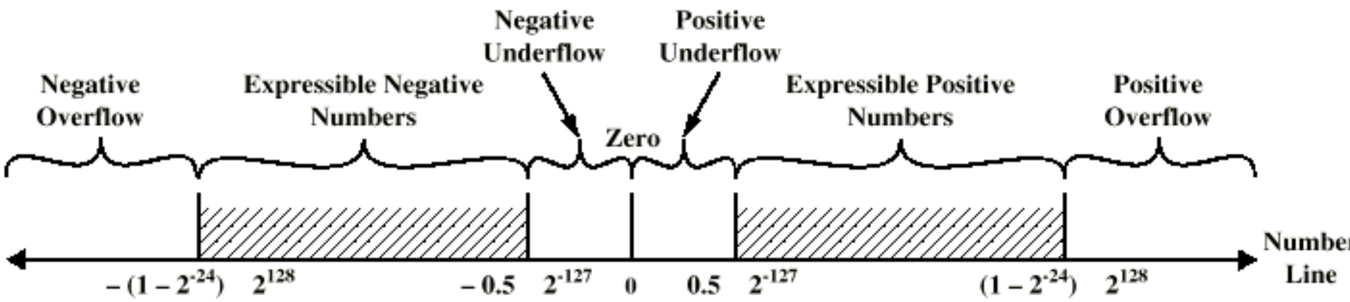
# FP Ranges

- For a 32 bit number
  - 8 bit exponent
  - +/- $2^{256} \approx 1.5 \times 10^{77}$
- Accuracy
  - The effect of changing lsb of mantissa
  - 23 bit mantissa $2^{-23} \approx 1.2 \times 10^{-7}$
  - About 6 decimal places

# Expressible Numbers

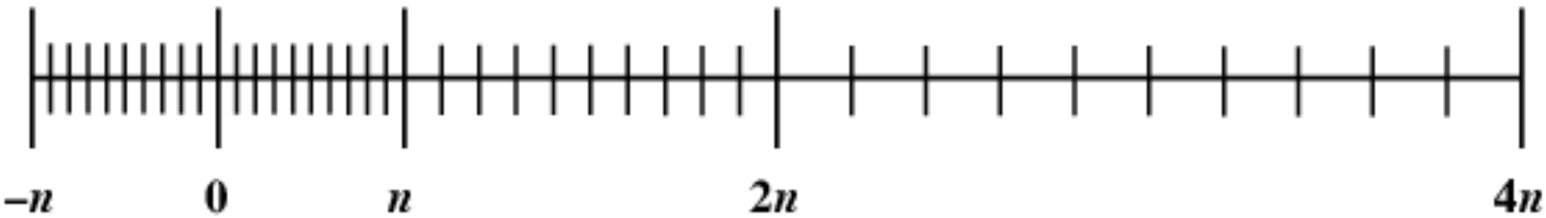


(a) Twos Complement Integers



(b) Floating-Point Numbers

# Density of Floating Point Numbers

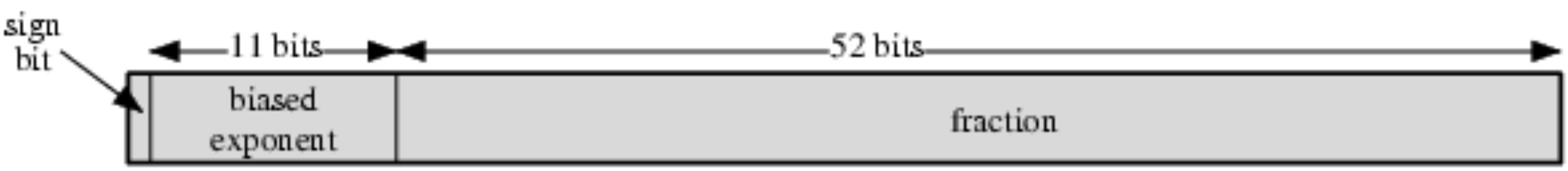


# IEEE 754

- Standard for floating point storage
- 32 and 64 bit standards
- 8 and 11 bit exponent respectively
- Extended formats (both mantissa and exponent) for intermediate results

# IEEE 754 Formats


(a) Single format




(b) Double format

# IEEE 754 Format Parameters

| Parameter | Single | Single Extended | Double | Double Extended |
|---|---|---|---|---|
| Word width (bits) | 32 | ≥ 43 | 64 | ≥ 79 |
| Exponent width (bits) | 8 | ≥ 11 | 11 | ≥ 15 |
| Exponent bias | 127 | unspecified | 1023 | unspecified |
| Maximum exponent | 127 | ≥ 1023 | 1023 | ≥ 16383 |
| Minimum exponent | –126 | ≤ –1022 | –1022 | ≤ –16382 |
| Number range (base 10) | $10^{-38}$, $10^{+38}$ | unspecified | $10^{-308}$, $10^{+308}$ | unspecified |
| Significand width (bits)* | 23 | ≥ 31 | 52 | ≥ 63 |
| Number of exponents | 254 | unspecified | 2046 | unspecified |
| Number of fractions | $2^{23}$ | unspecified | $2^{52}$ | unspecified |
| Number of values | $1.98 \times 2^{31}$ | unspecified | $1.99 \times 2^{63}$ | unspecified |

* not including implied bit

# Interpretation of IEEE 754 Floating-Point Numbers

| | 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 | $\infty$ | 0 | 2047 (all 1s) | 0 | $\infty$ |
| 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 | $f \neq 0$ | $-2^{e-1022}(0.f)$ |

# Floating-Point
# Numbers and Arithmetic Operations

| Floating Point Numbers | Arithmetic Operations |
|---|---|
| $X = X_S \times B^{X_E}$<br>$Y = Y_S \times B^{Y_E}$ | $\left.\begin{aligned} X + Y &= \left(X_s \times B^{X_E - Y_E} + Y_s\right) \times B^{Y_E} \\ X - Y &= \left(X_s \times B^{X_E - Y_E} - Y_s\right) \times B^{Y_E} \end{aligned}\right\} X_E \le Y_E$<br>$X \times Y = \left(X_s \times Y_s\right) \times B^{X_E + Y_E}$<br>$\frac{X}{Y} = \left(\frac{X_s}{Y_s}\right) \times B^{X_E - Y_E}$ |

Examples:

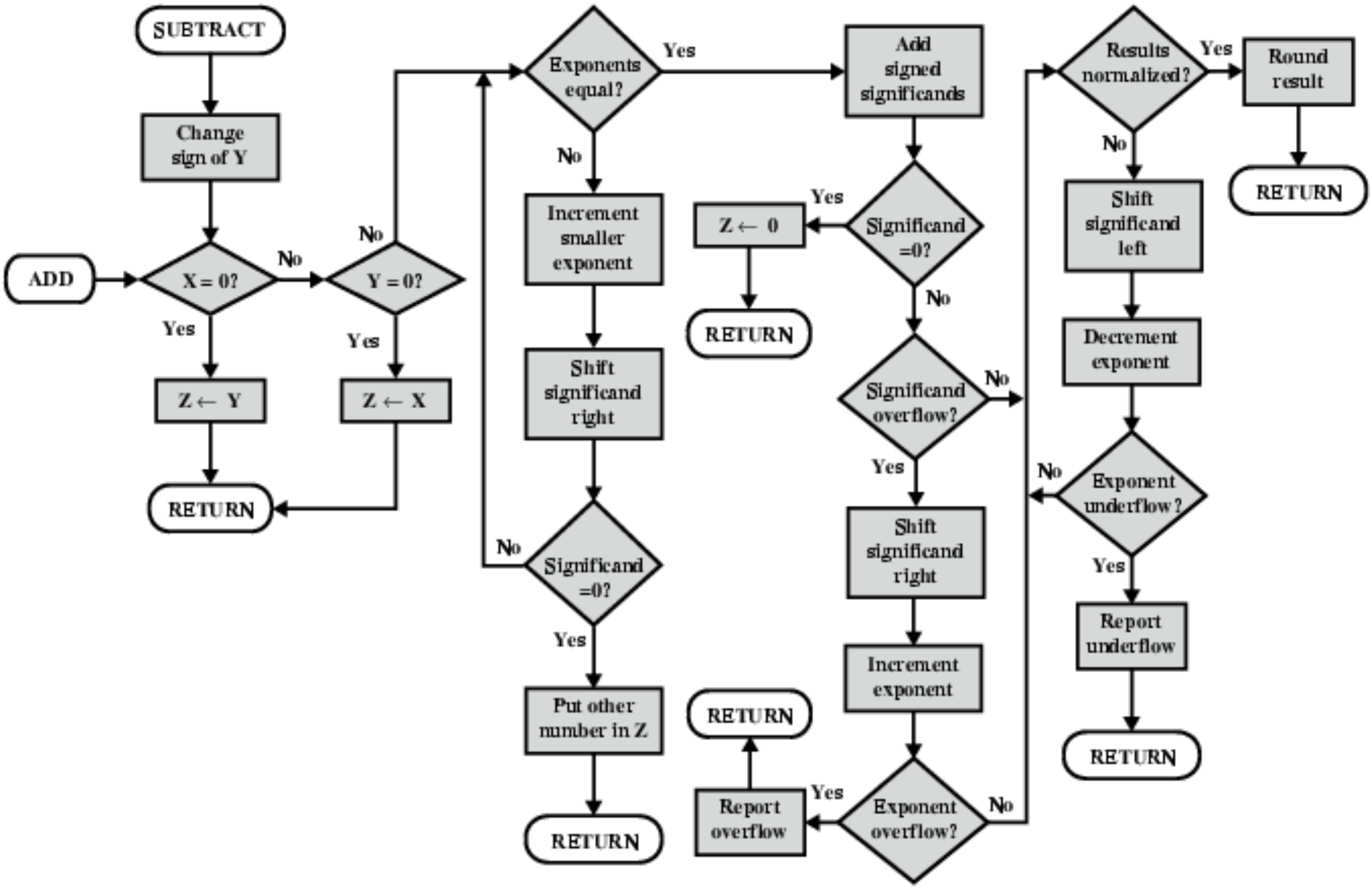
$X = 0.3 \times 10^2 = 30$
$Y = 0.2 \times 10^3 = 200$

$X + Y = (0.3 \times 10^{2-3} + 0.2) \times 10^3 = 0.23 \times 10^3 = 230$
$X - Y = (0.3 \times 10^{2-3} - 0.2) \times 10^3 = (-0.17) \times 10^3 = -170$
$X \times Y = (0.3 \times 0.2) \times 10^{2+3} = 0.06 \times 10^5 = 6000$
$X \div Y = (0.3 \div 0.2) \times 10^{2-3} = 1.5 \times 10^{-1} = 0.15$

# FP Arithmetic +/-

- Check for zeros
- Align significands (adjusting exponents)
- Add or subtract significands
- Normalize result

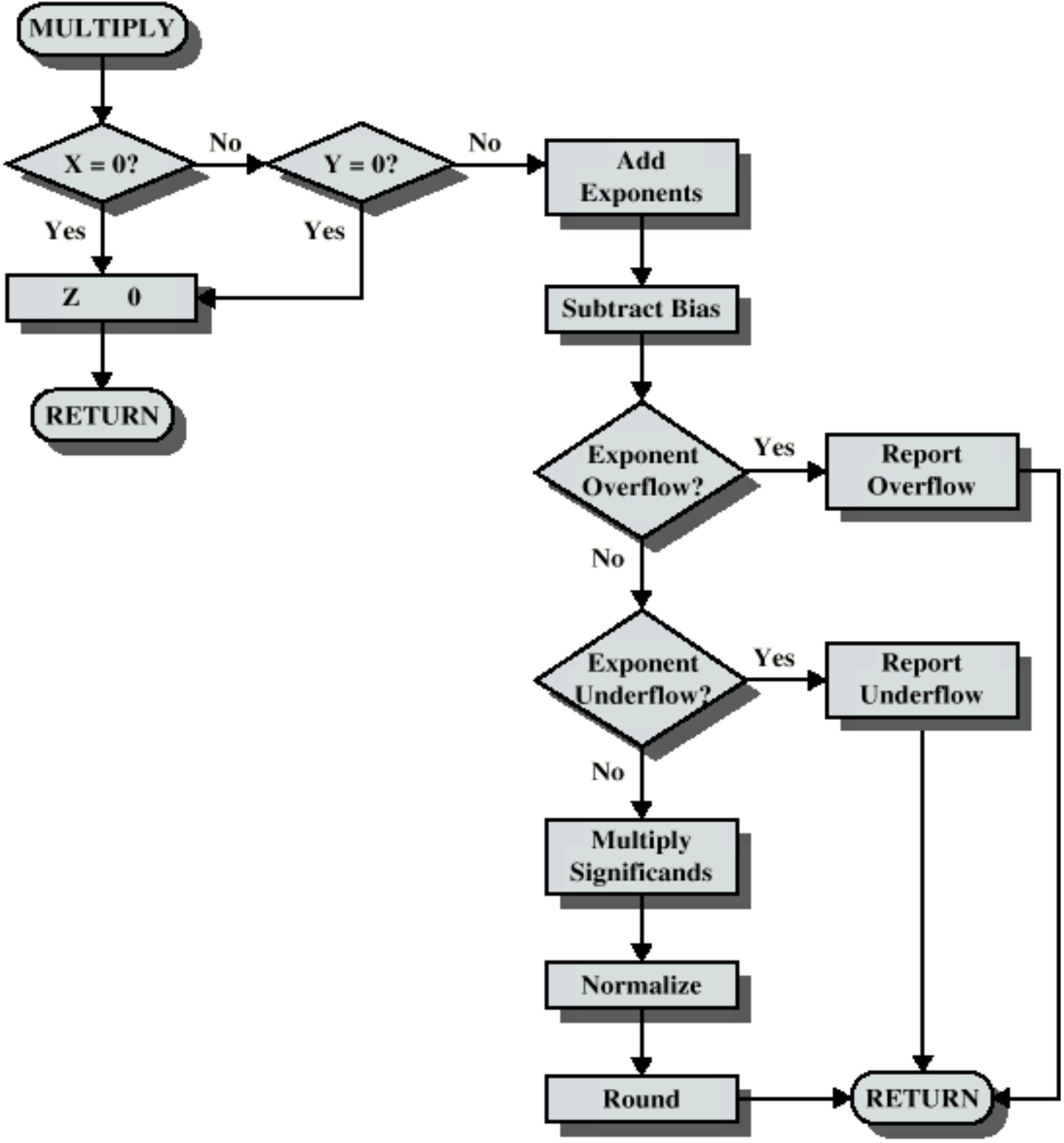
# FP Addition & Subtraction Flowchart

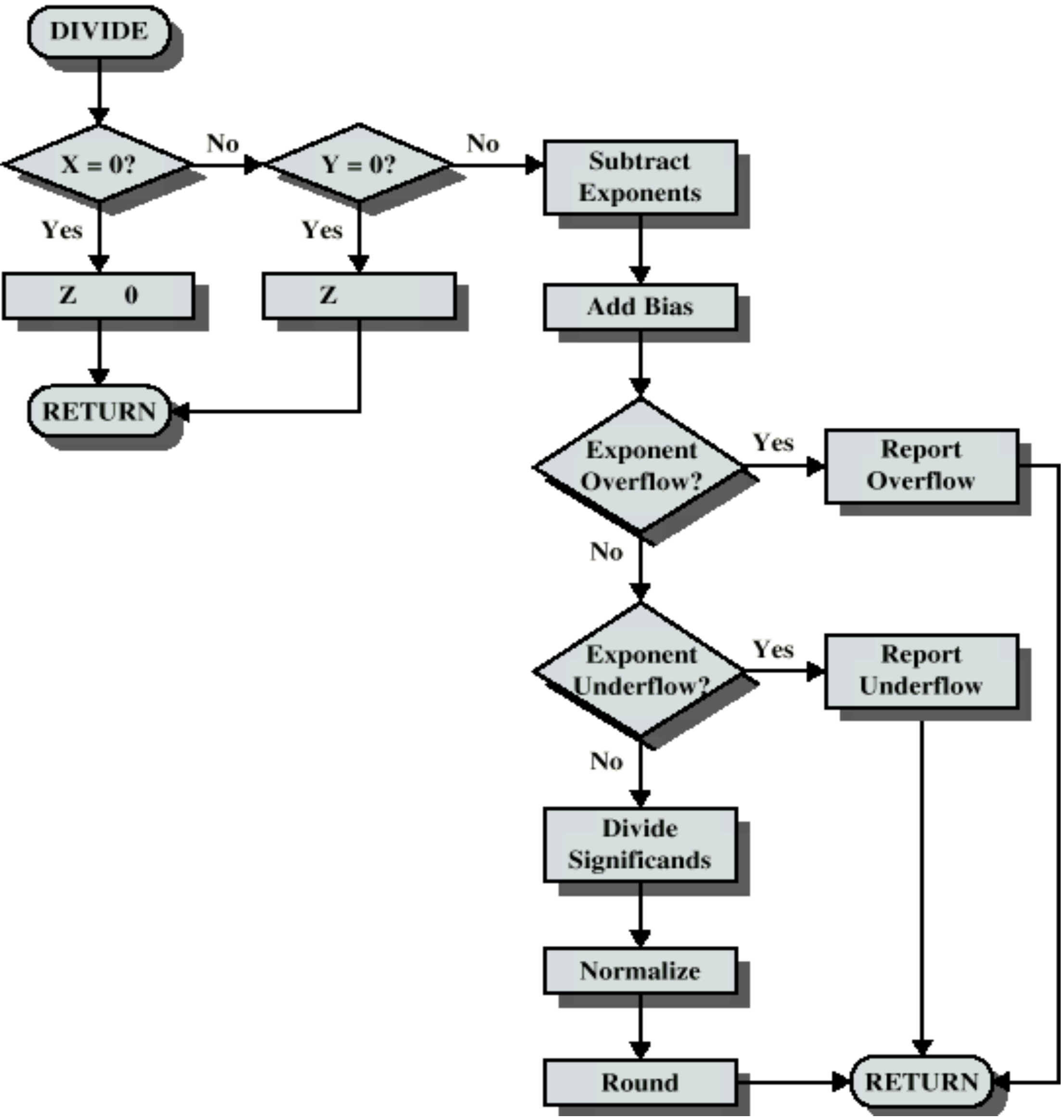


# FP Arithmetic x/÷

- Check for zero
- Add/subtract exponents
- Multiply/divide significands (watch sign)
- Normalize
- Round
- All intermediate results should be in double length storage

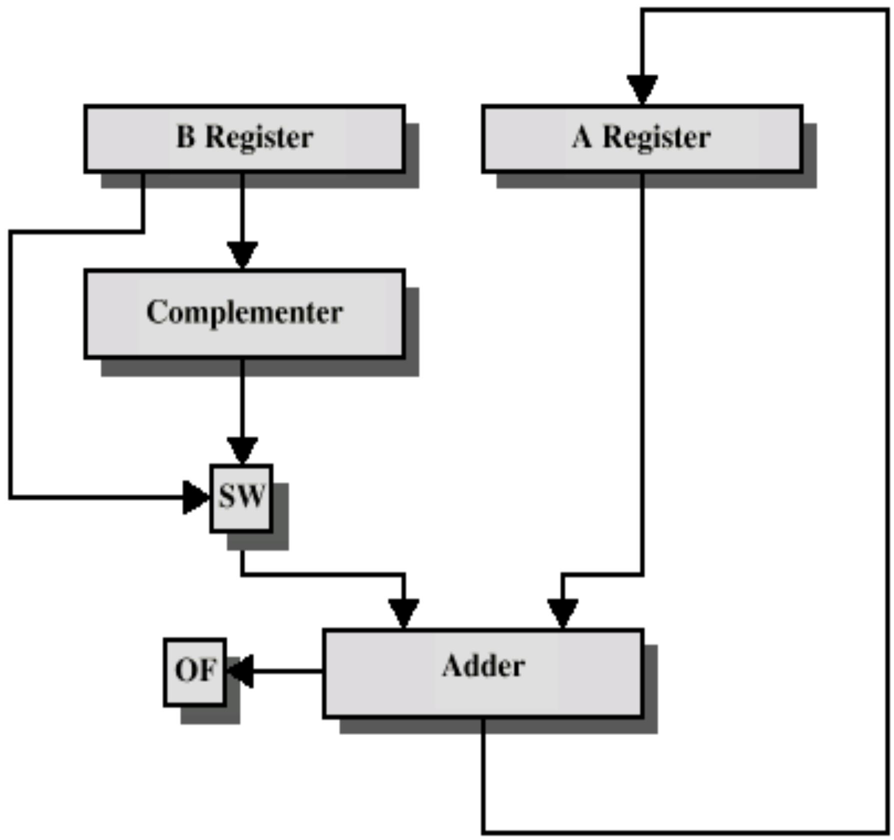
# Floating Point Multiplication






# Floating Point Division

# Hardware for Addition and Subtraction




OF = overflow bit
SW = Switch (select addition or subtraction)

# Arithmetic operations on reals

- All arithmetic operations such as addition, subtraction, multiplication and division can be applied to reals stored in floating-point format.
- Multiplication of two reals involves multiplication of two integers in sign-and-magnitude representation.
- Division of two reals involves division of two integers in sign-and-magnitude representations.
- Since we did not discuss the multiplication or division of integers in sign-and magnitude representation, we will not discuss the multiplication and division of reals, and only show addition and subtractions for reals.

# Addition and subtraction of reals

- Addition and subtraction of real numbers stored in floating-point numbers is reduced to addition and subtraction of two integers stored in sign-and-magnitude (combination of sign and mantissa) after the alignment of decimal points.
- Figure 4.8 shows a simplified version of the procedure (there are some special cases that we have ignored).

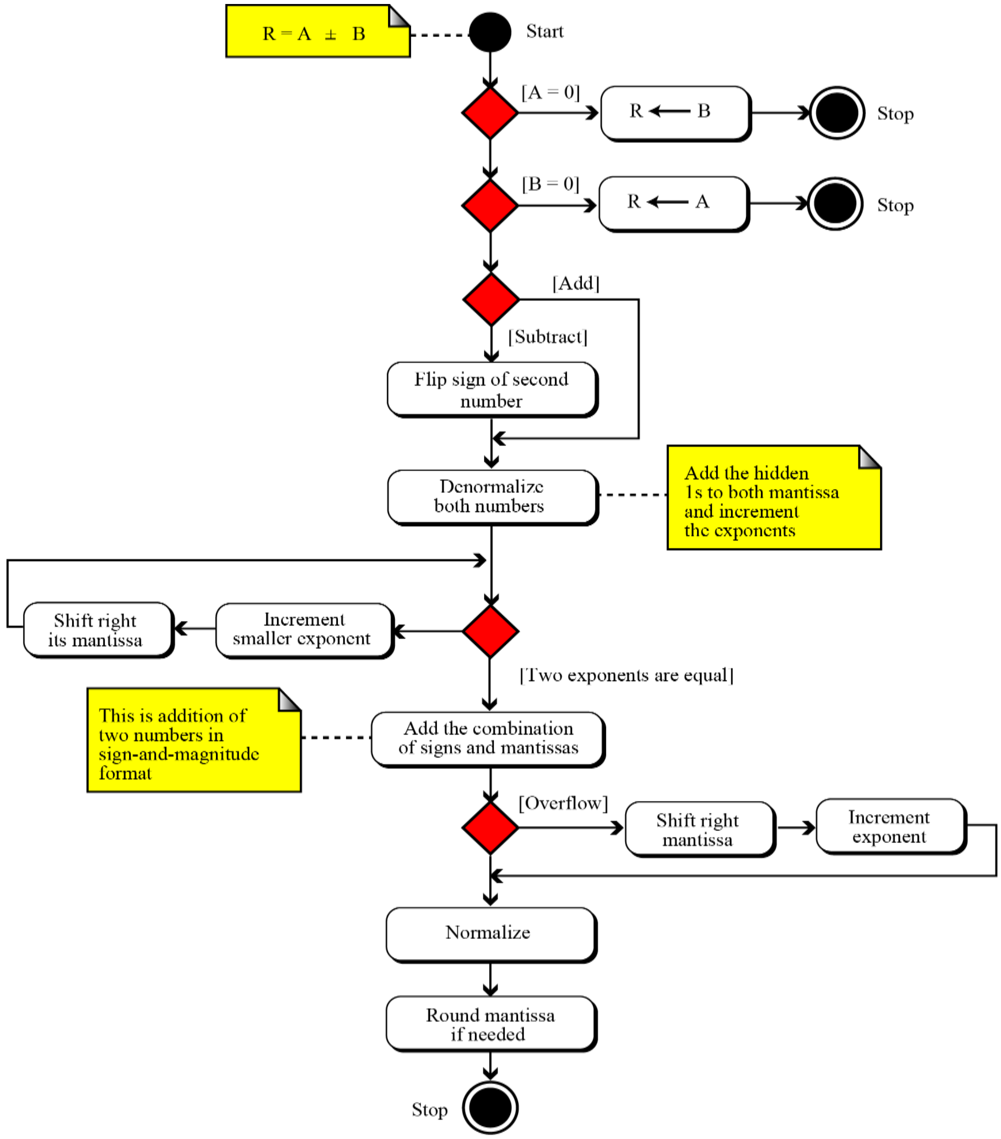





Figure 4.8 Addition and subtraction of reals in floating-point format

## Example 4.24

Show how the computer finds the result of
(+5.75) + (+161.875) = (+167.625).

### Solution

As we saw in Chapter 3, these two numbers are stored in floating-point format, as shown below, but we need to remember that each number has a hidden 1 (which is not stored, but assumed).

| | S | E | M |
|---|---|---|---|
| A | 0 | 10000001 | 01110000000000000000000 |
| B | 0 | 10000110 | 01000011110000000000000 |

## Example 4.24 (Continued)

- The first few steps in the UML diagram (Figure 4.8) are not needed.
- We de-normalize the numbers by adding the hidden 1s to the mantissa and incrementing the exponent.
- Now both de-normalized mantissas are 24 bits and include the hidden 1s.
  - They should be stored in a location that can hold all 24 bits.
  - Each exponent is incremented.

| | S | E | Denormalized M |
|---|---|---|---|
| A | 0 | 10000010 | 101110000000000000000000 |
| B | 0 | 10000111 | 101000011110000000000000 |

Example 4.24 (Continued)

- Align the mantissa
  - Increment the exponent of the first number five times
  - Shift the first mantissa to the right by five positions

| | S | E | Denormalized M |
|---|---|---|---|
| A | 0 | 10000111 | 00000101110000000000000 |
| B | 0 | 10000111 | 10100001111000000000000 |

- Now we do sign-and-magnitude addition, treating the sign and the mantissa of each number as one integer stored in sign-and-magnitude representation.

| | S | E | Denormalized M |
|---|---|---|---|
| R | 0 | 10000111 | 10100111101000000000000 |

## Example 4.24 (Continued)

- There is no overflow in the mantissa, so we normalize.

| | S | E | M |
|---|---|---|---|
| R | 0 | 10000110 | 01001111010000000000000 |

- The mantissa is only 23 bits, no rounding is needed.
- $E = (10000110)_2 = 134$, $M = 0100111101$.
- In other words, the result is $(1.0100111101)_2 \times 2^{134-127} = (10100111.101)_2 =$ 167.625.

## Example 4.25

Show how the computer finds the result of
(+5.75) + (−7.0234375) = − 1.2734375.

### Solution

- These two numbers can be stored in floating-point format, as shown below:

| | S | E | M |
|---|---|---|---|
| A | 0 | 10000001 | 01110000000000000000000 |
| B | 1 | 10000001 | 11000001100000000000000 |

- De-normalization results in:

| | S | E | Denormalized M |
|---|---|---|---|
| A | 0 | 10000010 | **1**0111000000000000000000000 |
| B | 1 | 10000010 | **1**1100000110000000000000 |

Example 4.25 (Continued)

- Alignment is not needed (both exponents are the same)
- We apply addition operation on the combinations of sign and mantissa.
- The result is shown below, in which the sign of the result is negative:

| | S | E | Denormalized M |
|---|---|---|---|
| R | 1 | 10000010 | 00101000110000000000000 |

- Now we need to normalize.
  - We decrement the exponent three times
  - Shift the de-normalized mantissa to the left three positions:

| | S | E | M |
|---|---|---|---|
| R | 1 | 01111111 | 01000110000000000000000 |

Example 4.25 (Continued)

- The mantissa is now 24 bits, so we round it to 23 bits.

| | S | E | M |
|---|---|---|---|
| R | 1 | 01111111 | 01000110000000000000000 |

- The result is $R = -\,2^{127-127} \times 1.0100011 = -\,1.2734375$, as expected.

## Example 4.21

Two integers A and B are stored in sign-and-magnitude format (we have separated the magnitude for clarity).Show how B is added to A.

$$A = (0\ 0010001)_2 \qquad B = (0\ 0010110)_2$$

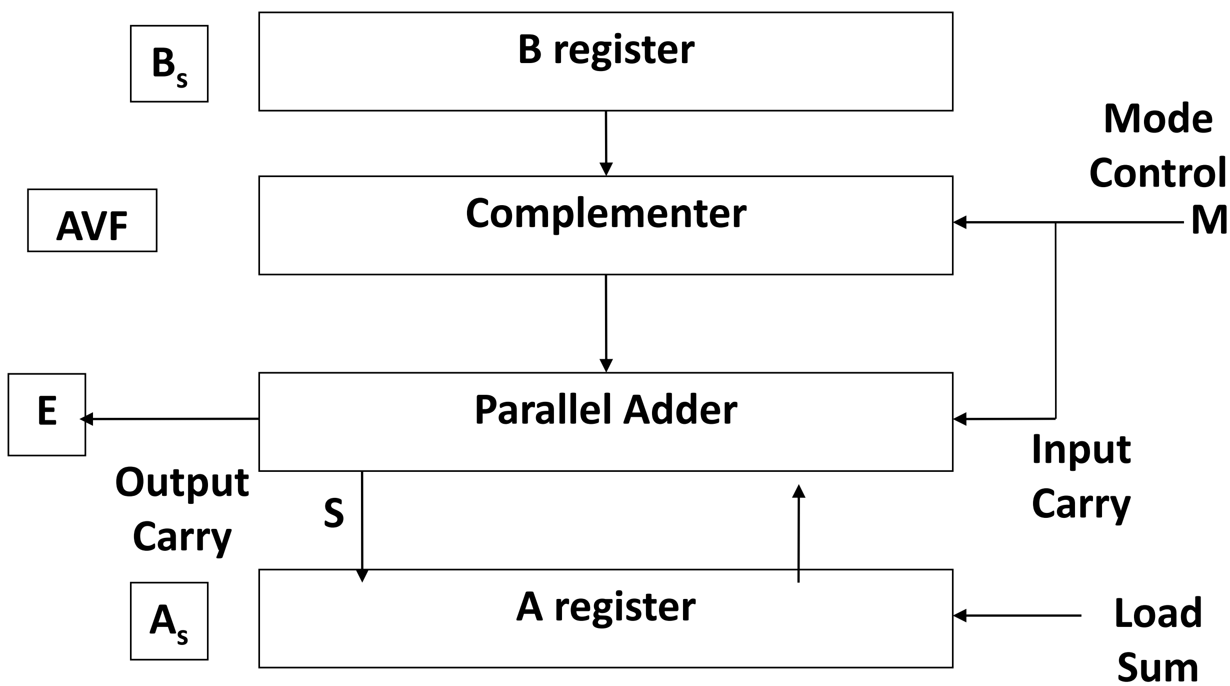
Solution

- The operation is adding.
- A is added to B and the result is stored in R.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | Carry |
| | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | A |
| + | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | B |
| | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | R |

- We expect the result to be 127 + 3 = 130, but the answer is −126.
- The error is due to overflow, because the expected answer (+130) is not in the range −128 to +127.

# Addition and Subtraction with Signed-Magnitude Data Hardware Design



## ❑ Flowchart of addition and subtraction of integers in sign-and-magnitude format