SASTRA DEEMED TO BE UNIVERSITY	**COMPUTER ORGANIZATION AND ARCHITECTURE** **Course Code: CAP403R01 Semester : I / MCA** **Dr. K. Geetha/SAP/CSE/SOC/SASTRA**

1.6 Floating point representation

Floating point numbers

- Programming languages support numbers with fraction
  - Called floating-point numbers
  - Examples:

    3.14159265… (π)

    2.71828… (e)

    0.000000001 or 1.0×10^{-9} (seconds in a nanosecond)

    86,400,000,000,000 or 8.64×10^{13} (nanoseconds in a day)

    last number is a large integer that cannot fit in a 32-bit integer
- We use a scientific notation to represent
  - Very small numbers (e.g. 1.0×10^{-9})
  - Very large numbers (e.g. 8.64×10^{13})
  - Scientific notation: $\pm\, d.\, f_1 f_2 f_3 f_4 \ldots \times 10^{\pm e_1 e_2 e_3}$

Floating point numbers Contd..

- Examples of floating-point numbers in base 10 …
  - 5.341×10^{3} , 0.05341×10^{5} , -2.013×10^{-1} , -201.3×10^{-3}

    decimal point
- Examples of floating-point numbers in base 2 …
  - 1.00101×2^{23} , 0.0100101×2^{25} , -1.101101×2^{-3} , -1101.101×2^{-6}

    binary point
  - Exponents are kept in decimal for clarity
  - The binary number (1101.101) = $2^3+2^2+2^0+2^{-1}+2^{-3}$ = 13.625

binary point

- ✧ Exponents are kept in decimal for clarity
- ✧ The binary number $(1101.101)_2 = 2^3+2^2+2^0+2^{-1}+2^{-3} = 13.625$

❖ Floating-point numbers should be normalized

- ✧ Exactly one non-zero digit should appear before the point
  - In a decimal number, this digit can be from 1 to 9
  - In a binary number, this digit should be 1
- ✧ Normalized FP Numbers: 5.341×10^3 and -1.101101×2^{-3}
- ✧ NOT Normalized: 0.05341×10^5 and -1101.101×2^{-6}

Floating point Representation

❖ A floating-point number is represented by the triple

- ✧ *S* is the Sign bit (0 is positive and 1 is negative)
  - Representation is called sign and magnitude
- ✧ *E* is the Exponent field (signed)
  - Very large numbers have large positive exponents
  - Very small close-to-zero numbers have negative exponents
  - More bits in exponent field increases range of values
- ✧ *F* is the Fraction field (fraction after binary point)
  - More bits in fraction field improves the precision of FP numbers

S	Exponent	Fraction

Value of a floating-point number = $(-1)^S \times \text{val}(F) \times 2^{\text{val}(E)}$

Floating point Standard

- Found in virtually every computer invented since 1980
  - Simplified porting of floating-point numbers
  - Unified the development of floating-point algorithms
  - Increased the accuracy of floating-point numbers
- Single Precision Floating Point Numbers (32 bits)
  - 1-bit sign + 8-bit exponent + 23-bit fraction

S	Exponent^8	Fraction^{23}

- Double Precision Floating Point Numbers (64 bits)
  - 1-bit sign + 11-bit exponent + 52-bit fraction

S	Exponent^{11}	Fraction^{52}
(continued)		

Floating point Normalization

- For a normalized floating point number (S, E, F)

S	E	$F = f_1\ f_2\ f_3\ f_4 \ldots$

- Significand is equal to $(1.F)_2 = (1.f_1 f_2 f_3 f_4 \ldots)_2$
  - IEEE 754 assumes hidden 1. (not stored) for normalized numbers
  - Significand is 1 bit longer than fraction
- Value of a Normalized Floating Point Number is

$$(-1)^S \times (1.F)_2 \times 2^{\text{val}(E)}$$

$$(-1)^S \times (1.f_1 f_2 f_3 f_4 \ldots)_2 \times 2^{\text{val}(E)}$$

$$(-1)^S \times (1 + f_1{\times}2^{-1} + f_2{\times}2^{-2} + f_3{\times}2^{-3} + f_4{\times}2^{-4} \ldots)_2 \times 2^{\text{val}(E)}$$

$(-1)^S$ is 1 when S is 0 (positive), and -1 when S is 1 (negative)

Biased Exponent representation

- How to represent a signed exponent? Choices are …
  - Sign + magnitude representation for the exponent
  - Two's complement representation
  - Biased representation
- IEEE 754 uses biased representation for the exponent
  - Value of exponent = val(E) = E – Bias (Bias is a constant)
- Recall that exponent field is 8 bits for single precision
  - E can be in the range 0 to 255
  - E = 0 and E = 255 are reserved for special use (discussed later)
  - E = 1 to 254 are used for normalized floating point numbers
  - Bias = 127 (half of 254), val(E) = E – 127
  - val(E=1) = –126, val(E=127) = 0, val(E=254) = 127

Biased exponent Contd..

- For double precision, exponent field is 11 bits
  - E can be in the range 0 to 2047
  - $E = 0$ and $E = 2047$ are reserved for special use
  - $E = 1$ to 2046 are used for normalized floating point numbers
  - Bias = 1023 (half of 2046), val(E) = $E - 1023$
  - val(E=1) = –1022, val(E=1023) = 0, val(E=2046) = 1023
- Value of a Normalized Floating Point Number is

$$(-1)^S \times (1.F)_2 \times 2^{E - \text{Bias}}$$
$$(-1)^S \times (1.f_1 f_2 f_3 f_4 \ldots)_2 \times 2^{E - \text{Bias}}$$
$$(-1)^S \times (1 + f_1\times 2^{-1} + f_2\times 2^{-2} + f_3\times 2^{-3} + f_4\times 2^{-4} \ldots)_2 \times 2^{E - \text{Bias}}$$

Single precision - example

- What is the decimal value of this Single Precision float?

1	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Solution:
  - Sign = 1 is negative
  - Exponent = $(01111100)_2 = 124$, E – bias = 124 – 127 = –3
  - Significand = $(1.0100 \ldots 0)_2 = 1 + 2^{-2} = 1.25$ (1. is implicit)
  - Value in decimal = $-1.25 \times 2^{-3} = -0.15625$
- What is the decimal value of?

0	1	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Solution:

  implicit
  - Value in decimal = $+(1.01001100 \ldots 0)_2 \times 2^{130-127} =$
    $(1.01001100 \ldots 0)_2 \times 2^3 = (1010.01100 \ldots 0)_2 = 10.375$

Double precision - example

- What is the decimal value of this Double Precision float ?

0	1	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Solution:
  - Value of exponent = $(10000000101)_2$ – Bias = 1029 – 1023 = 6
  - Value of double float = $(1.00101010 \ldots 0)_2 \times 2^6$ (1. is implicit) = $(1001010.10 \ldots 0)_2 = 74.5$

- What is the decimal value of ?

1	0	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Do it yourself! (answer should be $-1.5 \times 2^{-7} = -0.01171875$)

Largest Normalized Float

- What is the Largest normalized float?
- Solution for Single Precision:

  0 11111110 11111111111111111111111

  - Exponent – bias = 254 – 127 = 127 (largest exponent for SP)
  - Significand = $(1.111 \ldots 1)_2$ = almost 2
  - Value in decimal $\approx 2 \times 2^{127} \approx 2^{128} \approx 3.4028 \ldots \times 10^{38}$
- Solution for Double Precision:

  0 11111111110 11111111111111111111

  11111111111111111111111111111111

  - Value in decimal $\approx 2 \times 2^{1023} \approx 2^{1024} \approx 1.79769 \ldots \times 10^{308}$
- Overflow: exponent is too large to fit in the exponent field

Smallest Normalized float

- What is the smallest (in absolute value) normalized float?
- Solution for Single Precision:

  0 00000001 00000000000000000000000

  - Exponent – bias = 1 – 127 = –126 (smallest exponent for SP)
  - Significand = $(1.000 \ldots 0)_2 = 1$
  - Value in decimal $= 1 \times 2^{-126} = 1.17549 \ldots \times 10^{-38}$
- Solution for Double Precision:

  0 00000000001 00000000000000000000

  00000000000000000000000000000000

  - Value in decimal $= 1 \times 2^{-1022} = 2.22507 \ldots \times 10^{-308}$
- Underflow: exponent is too small to fit in exponent field

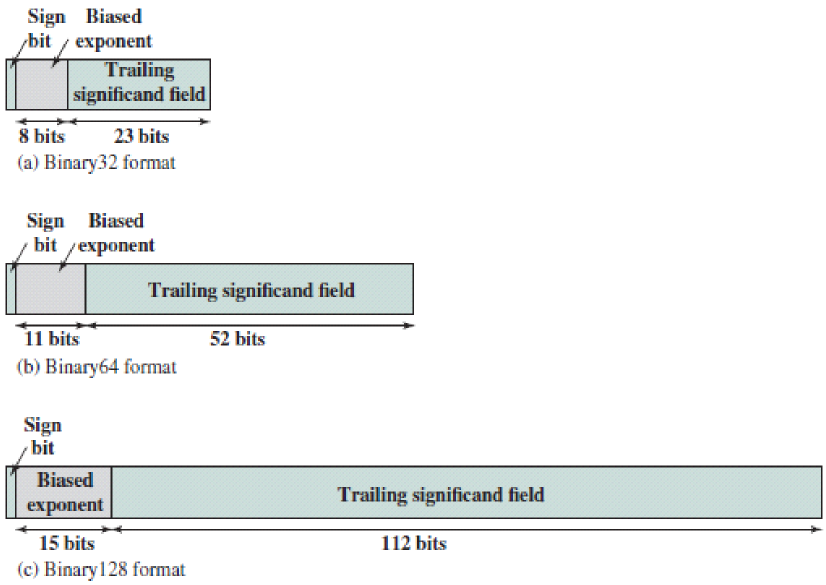

Figure 10.21 IEEE 754 Formats

Arithmetic Operations on Floating-Point Numbers

• **Exponent overflow:** A positive exponent exceeds the maximum possible exponent value. In some systems, this may be designated as or • **Exponent underflow:** A negative exponent is less than the minimum possible exponent value (e.g., is less than). This means that the number is too small to be represented, and it may be reported as 0.

• **Significand underflow:** In the process of aligning significands, digits may flow off the right end of the significand. As we shall discuss, some form of rounding is required.

• **Significand overflow**: The addition of two significands of the same sign may result in a carry out of the most significant bit. This can be fixed by realignment,

Addition and Subtraction

In floating-point arithmetic, addition and subtraction are more complex than multiplication and division. This is because of the need for alignment. There are four basic phases of the algorithm for addition and subtraction:

1. Check for zeros.
2. Align the significands.
3. Add or subtract the significands.
4. Normalize the result.

Table 9.5 Floating-Point Numbers and Arithmetic Operations

Floating Point Numbers	Arithmetic Operations
$X = X_S \times B^{X_E}$ $Y = Y_S \times B^{Y_E}$	$\left.\begin{array}{l} X + Y = (X_S \times B^{X_E - Y_E} + Y_S) \times B^{Y_E} \\ X - Y = (X_S \times B^{X_E - Y_E} - Y_S) \times B^{Y_E} \end{array}\right\} X_E \leq Y_E$ $X \times Y = (X_S \times Y_S) \times B^{X_E + Y_E}$ $\frac{X}{Y} = \left(\frac{X_S}{Y_S}\right) \times B^{X_E - Y_E}$

Examples:

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

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

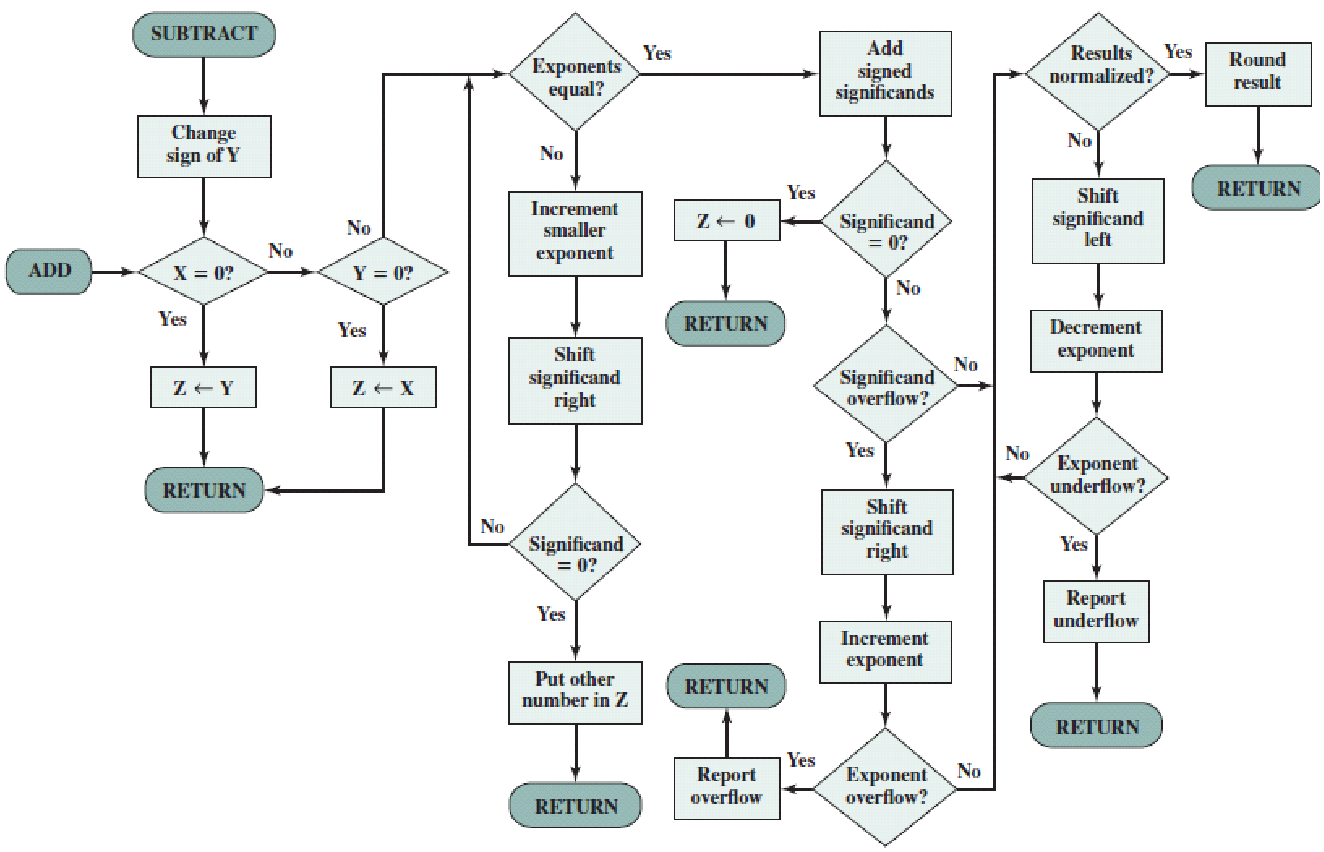

Figure 10.22 Floating-Point Addition and Subtraction (Z ← X ± Y)

In floating-point arithmetic, addition and subtraction are more complex than multiplication and division. This is because of the need for alignment. There are four basic phases of the algorithm for addition and subtraction:

1. Check for zeros.
2. Align the significands.
3. Add or subtract the significands.
4. Normalize the result.

Phase 1. Zero check: Because addition and subtraction are identical except for a sign change, the process begins by changing the sign of the subtrahend if it is a subtract operation. Next, if either operand is 0, the other is reported as the result.

Phase 2. Significand alignment: The next phase is to manipulate the numbers so that the two exponents are equal.

Phase 3. Addition: Next, the two significands are added together, taking into account their signs. Because the signs may differ, the result may be 0. There is also the possibility of significand overflow by 1 digit. If so, the significand of the result is shifted right and the exponent is incremented. An exponent overflow could occur as a result; this would be reported and the operation halted.

Phase 4. Normalization: The final phase normalizes the result. Normalization consists of shifting significand digits left until the most significant digit (bit, or 4 bits for base-16 exponent) is nonzero. Each shift causes a decrement of the exponent and thus could cause an exponent underflow. Finally, the result must be rounded off and then reported. We defer a discussion of rounding until after a discussion of multiplication and division.

Multiplication and Division

Floating-point multiplication and division are much simpler processes than addition and subtraction, as the following discussion indicates.

We first consider multiplication, illustrated in Figure 10.23. First, if either operand is 0, 0 is reported as the result. The next step is to add the exponents. If the exponents are stored in biased form, the exponent sum would have doubled the bias. Thus, the bias value must be subtracted from the sum. The result could be either an exponent overflow or underflow, which would be reported, ending the algorithm.

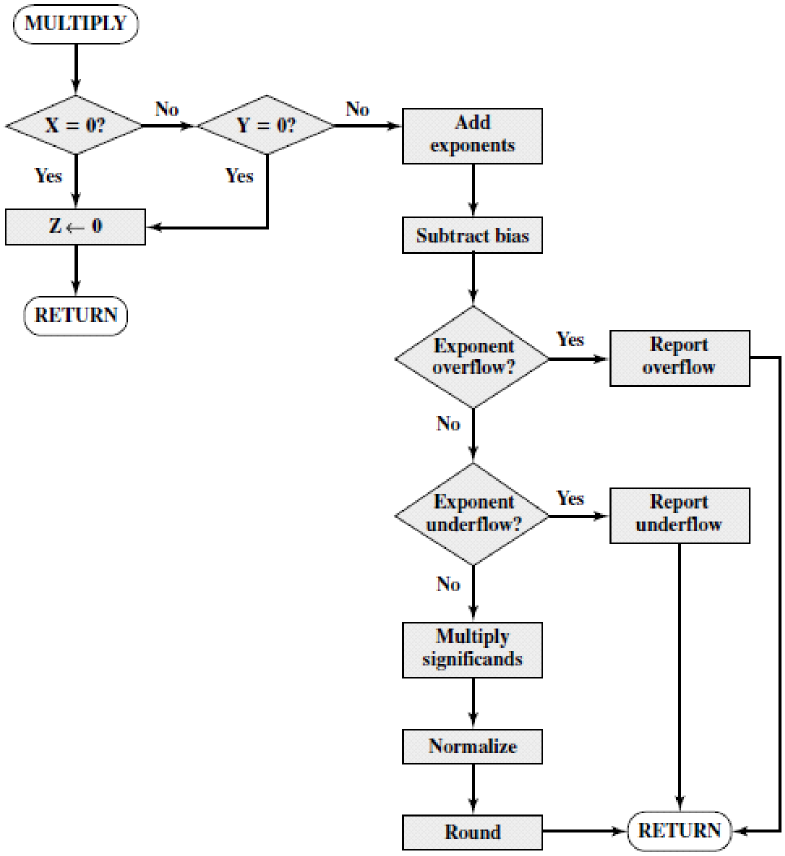

Figure 9.23 Floating-Point Multiplication ($Z \leftarrow X \times Y$)

Floating-Point Division

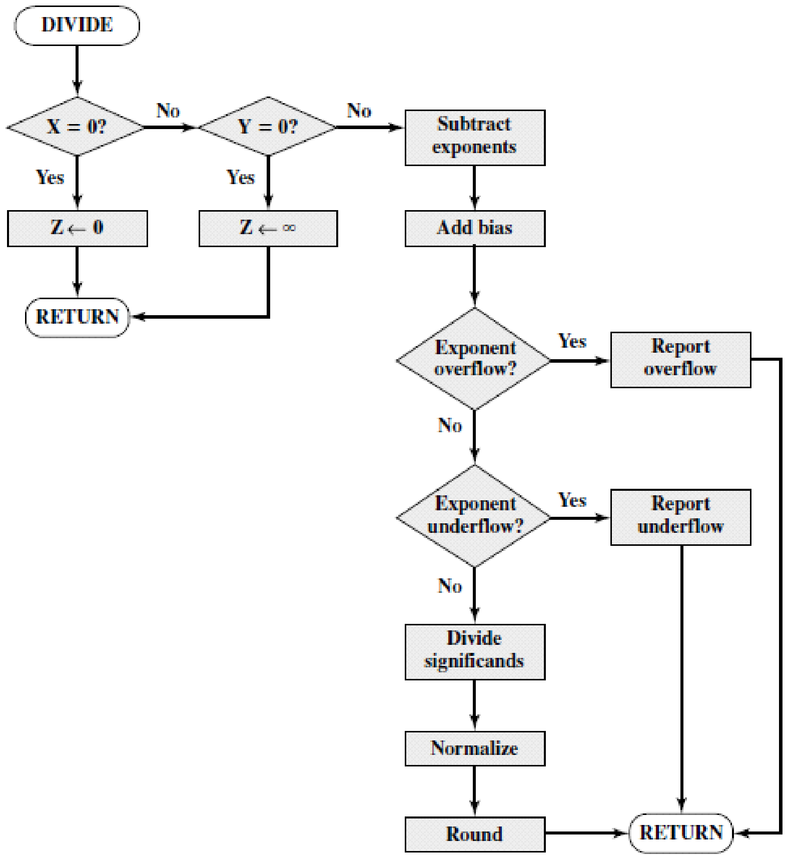

Figure 9.24 Floating-Point Division (Z ← X/Y)