Chapter 4

Computer Arithmetic

4.1 Number Systems

- A number system uses a specific radix (base). Radices that are power of 2 are widely used in digital systems. These radices include
  - binary (base 2),
  - quaternary (base 4),
  - octagonal (base 8), and
  - hexagonal (base 16).
- An unsigned integer number *A* can be represented using *n* digits in base *b* as follows: $A = (a_{n-1}a_{n-2} \cdots a_2 a_1 a_0)_b$
- In the *positional* representation, each digit a_i is given by

$0 \le a_i \le (b-1)$

4.1 Number Systems

- Using positional representation, the decimal value of the unsigned integer number *A* is given by $A = \sum_{i=0}^{n-1} a_i \times b^i$.
- Using *n* digits, the largest value for an unsigned number *A* is given by $A_{\max} = b^n - 1$.
- Consider the use of *n* digits to represent a real number *X* in radix *b* such that the most significant *k* digits represents the integral part while the least significant *m* digits represents the fraction part.
- The value of *X* is given by:

$$X = \sum_{i=-m}^{k-1} x_i \times b^i = x_{k-1}b^{k-1} + x_{k-2}b^{k-2} + \ldots + x_1 b^1 + x_0 b^0 + x_{-1}b^{-1} + \ldots + x_{-m}b^{-m}$$

4.1 Number Systems

- Radix Conversion Algorithm
  - A repeated multiplication of the fractional part of X (X_f) by r_2 retaining the obtained integers as the required digits, will result in the required representation of the fractional part in the new radix.
  - However, the fractional part conversion may not terminate after a finite number of repeated multiplications.
- Negative Integer Representation
  - There exists a number of methods for representation of negative integers:
    - the *sign-magnitude,*
    - *radix complement,* and
    - *diminished radix complement.*

4.1 Number Systems

- Sign-Magnitude
  - The most significant bit (out of the n bits used to represent the number) is used to represent the sign of the number such that a “1’’ in the most significant bit position indicates a negative number while a "0" in the most significant bit position indicates a positive number.
  - The remaining *(n-1)* bits are used to represent the magnitude of the number.
  - Although simple, the sign-magnitude representation is complicated when performing arithmetic operations.
  - In particular, the sign bit has to be dealt with separately from the magnitude bits.

4.1 Number Systems

- Radix Complement
  - A positive number is represented the same way as in the sign-magnitude.
  - However, a negative number is represented using the b’s complement (for base b numbers).
- Diminished Radix Complement
  - This representation is similar to the radix complement except for the fact that no “1” is added to the least significant bit after complementing the digits of the number, as is done in the radix complement.
  - The main disadvantage of the diminished radix representation is the need for a correction factor whenever a carry is obtained from the most significant bit while performing arithmetic operations.

4.2 Integer Arithmetic

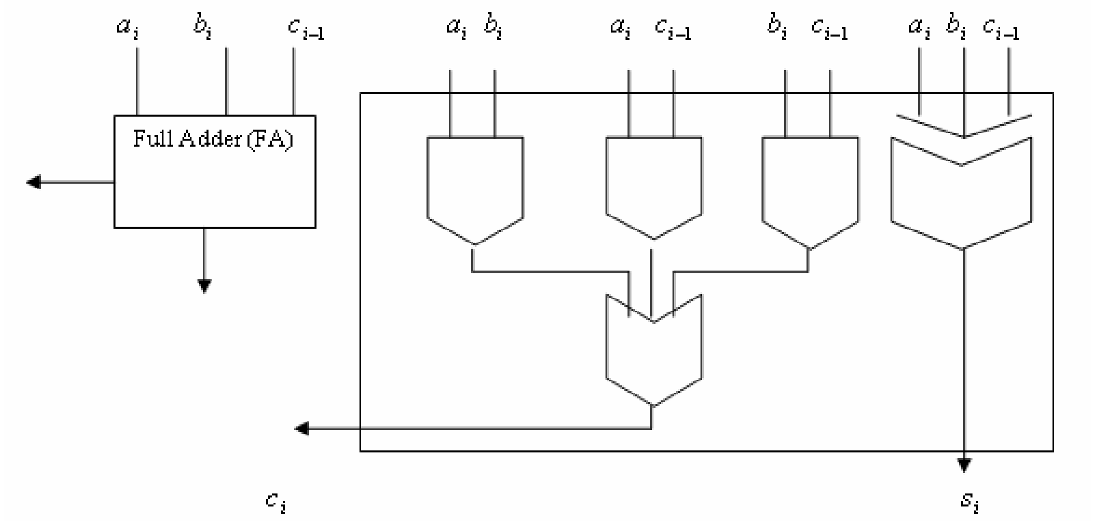
- Two's Complement (2's) Representation
  - In order to represent a number in 2’s complement, we perform the following two steps:
    - Perform the Boolean complement of each bit (including the sign bit)
    - Add 1 to the least significant bit (treating the number as an unsigned binary integer), i.e. $-A = \overline{A} + 1$.
- Two's Complement Arithmetic
  - Addition:
    - Addition of two *n-bit* numbers in 2's complement can be performed using an *n-bit* adder.
    - Any carry-out bit can be ignored without affecting the correctness of the results, as long as the results of the addition is in the range -2^{n-1} to $+2^{n-1}-1$.

4.2 Integer Arithmetic

- Two's Complement Arithmetic
  - Subtraction:
    - In 2's complement, subtraction can be performed in the same way addition is performed.
  - Hardware Structures for Addition and Subtraction of Signed Numbers:
    - The addition of two *n-bit* numbers *A* and *B* requires a basic hardware circuit that accepts three inputs, i.e., $a_i, b_i, \text{and } c_{i-1}$.
    - These three bits represent respectively the two current bits of the numbers *A* and *B* (at position *i)* and the carry bit from the previous bit position (at position *i-1)*.
    - The circuit should produce two outputs, i.e., $s_i \text{ and } c_i$ representing respectively the sum and the carry.

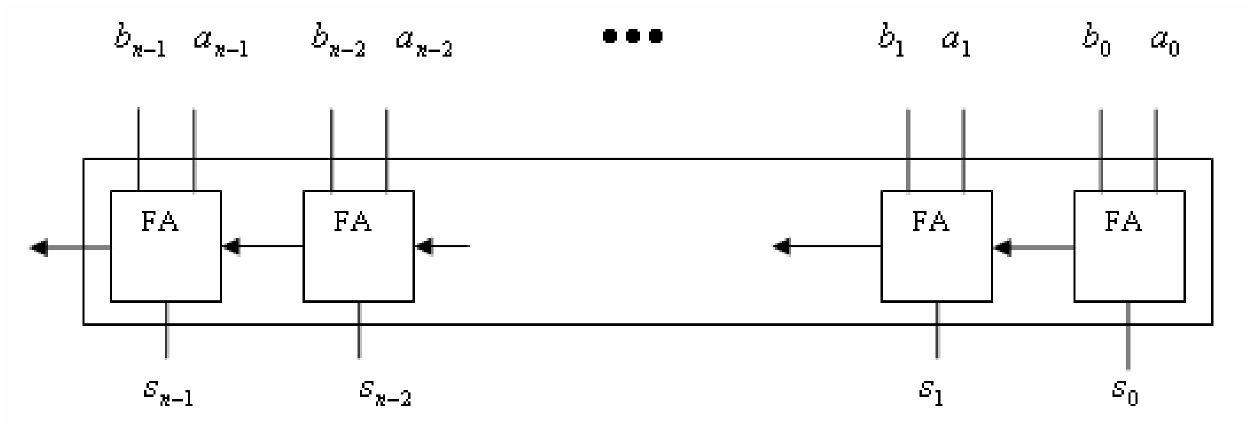
4.2 Integer Arithmetic

- Two's Complement Arithmetic
  - Hardware Structures for Addition and Subtraction of Signed Numbers:
    - The output logic functions are given by $s_i = a_i \oplus b_i \oplus c_{i-1}$ and $c_i = a_i b_i + a_i c_{i-1} + b_i c_{i-1}$
    - The circuit used to implement these two functions is called a *full-adder (FA).*



4.2 Integer Arithmetic

- Two's Complement Arithmetic
  - Hardware Structures for Addition and Subtraction of Signed Numbers:
    - Addition of two *n*-bit numbers *A* and *B* can be done using *n* consecutive *FAs* in an arrangement known as a *carry-ripple through adder (CRT)*

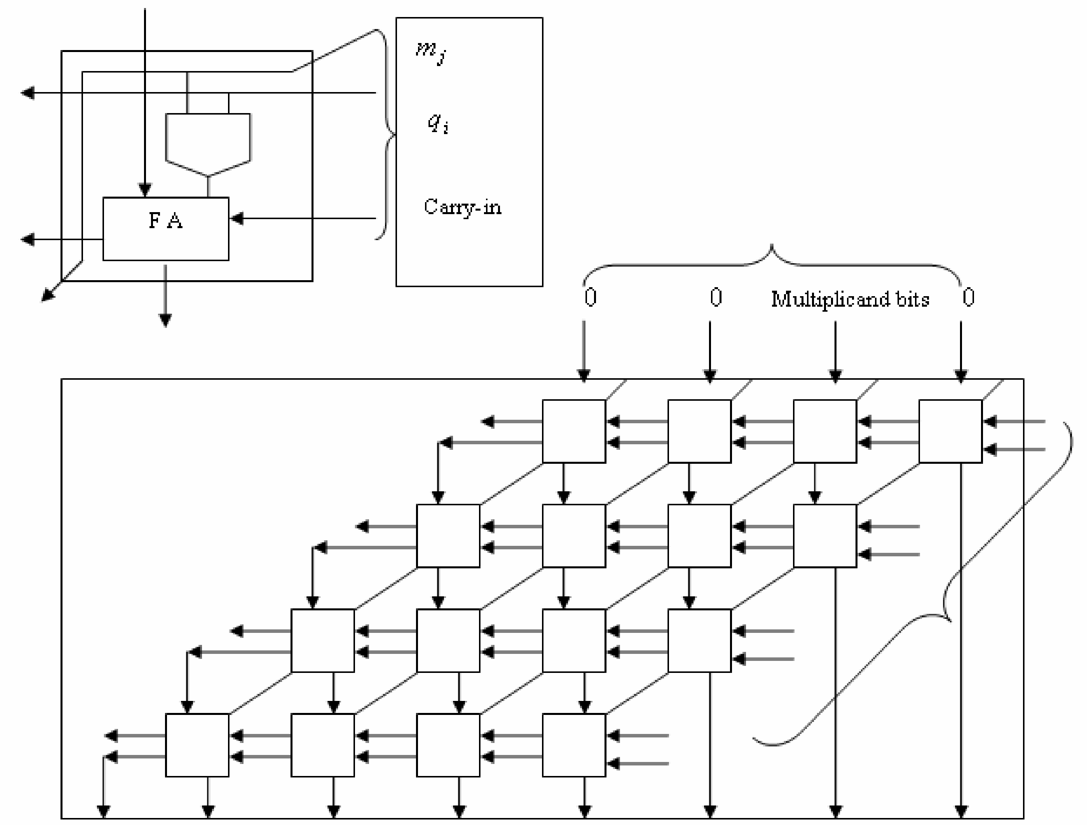


4.2 Integer Arithmetic

- Two's Complement Arithmetic
  - Multiplication:
    - The two input arguments are the multiplier *Q* given by: $Q = q_{n-1}q_{n-2}\ldots q_1 q_0$ and the multiplicand *M* given by: $M = m_{n-1}m_{m-2}\ldots m_1 m_0$.
    - A number of methods exist for performing multiplication.
      - The paper and Pencil Method (for unsigned numbers)
        - » This is the simplest method for performing multiplication of two unsigned numbers.
        - » The above multiplication can be performed using an array of cells each consisting of a FA and an AND.
        - » Each cell computes a given partial product.

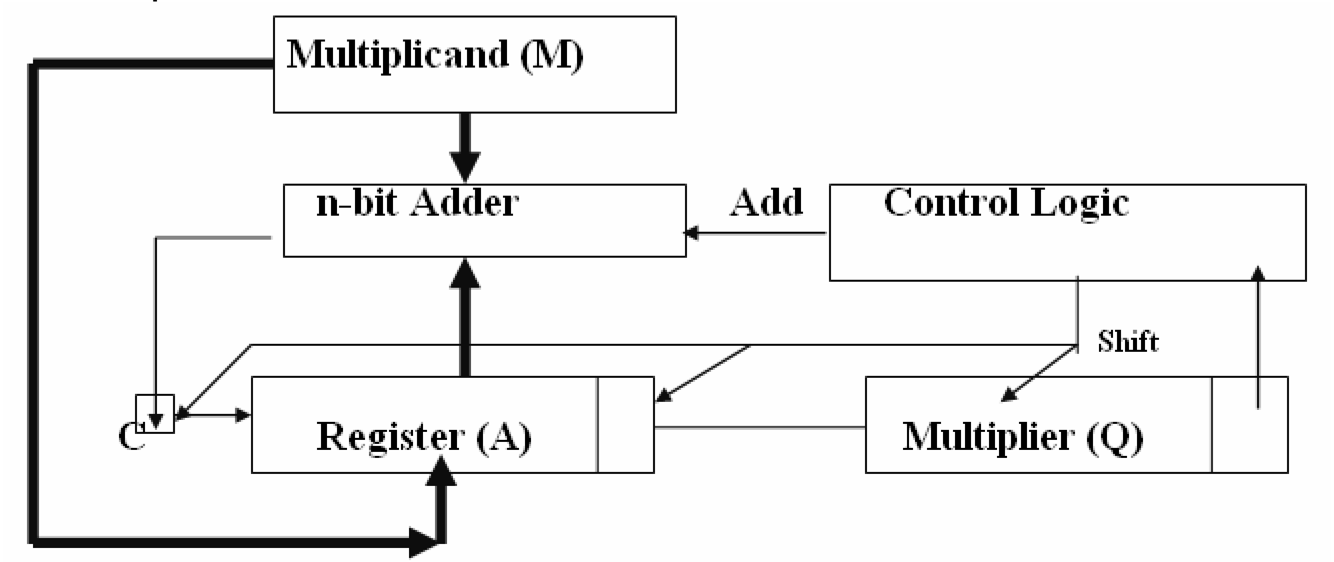
4.2 Integer Arithmetic

- Two's Complement Arithmetic
  - Multiplication - The paper and Pencil Method



4.2 Integer Arithmetic

- Two's Complement Arithmetic
  - Multiplication:
    - The Add-Shift Method
      - Multiplication is performed as a series of (*n*) conditional addition and shift operations such that if the given bit of the multiplier is 0 then only a shift operation is performed, while if the given bit of the multiplier is 1 then addition of the partial product and a shift operation are performed.

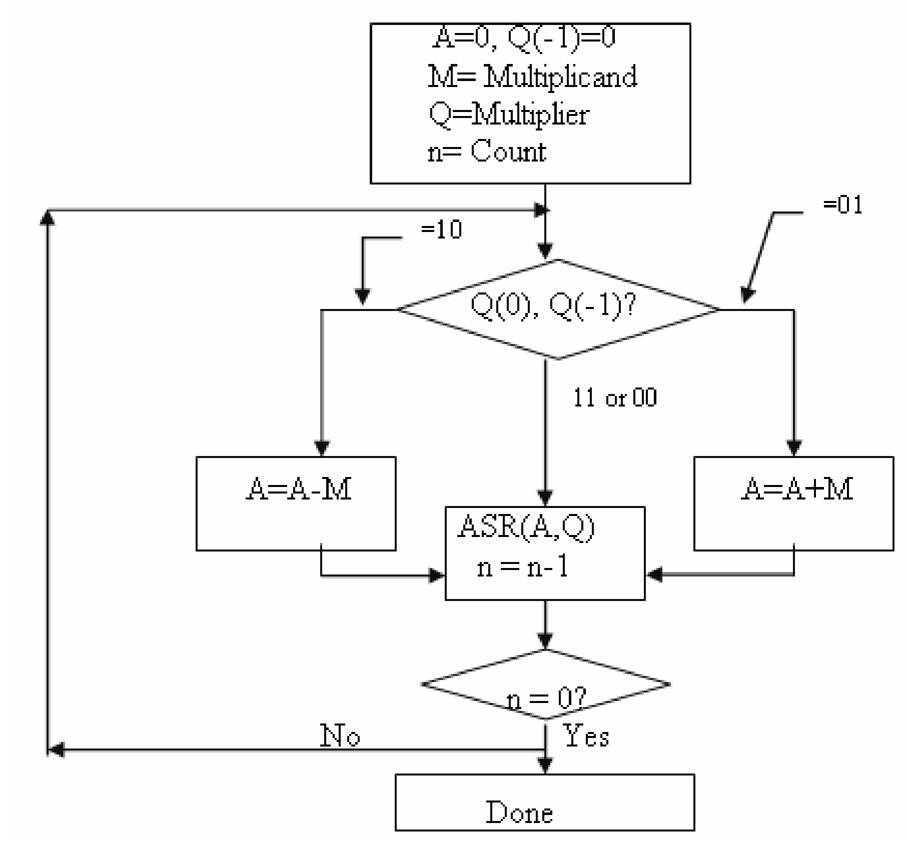


4.2 Integer Arithmetic

- Two's Complement Arithmetic
  - Multiplication:
    - The Booth's Algorithm
      - two bits of the multiplier, *Q(i)Q(i-1)*, $(0 \le i \le n-1)$ are inspected at a time.
      - The action taken depends on the binary values of the two bits, such that:
        - » if the two values are respectively 01, then $A \leftarrow A + M$
        - » if the two values are 10, then $A \leftarrow A - M$
        - » No action is needed if the values are 00 or 11.

4.2 Integer Arithmetic

- Two's Complement Arithmetic
  - Multiplication - The Booth’s Algorithm

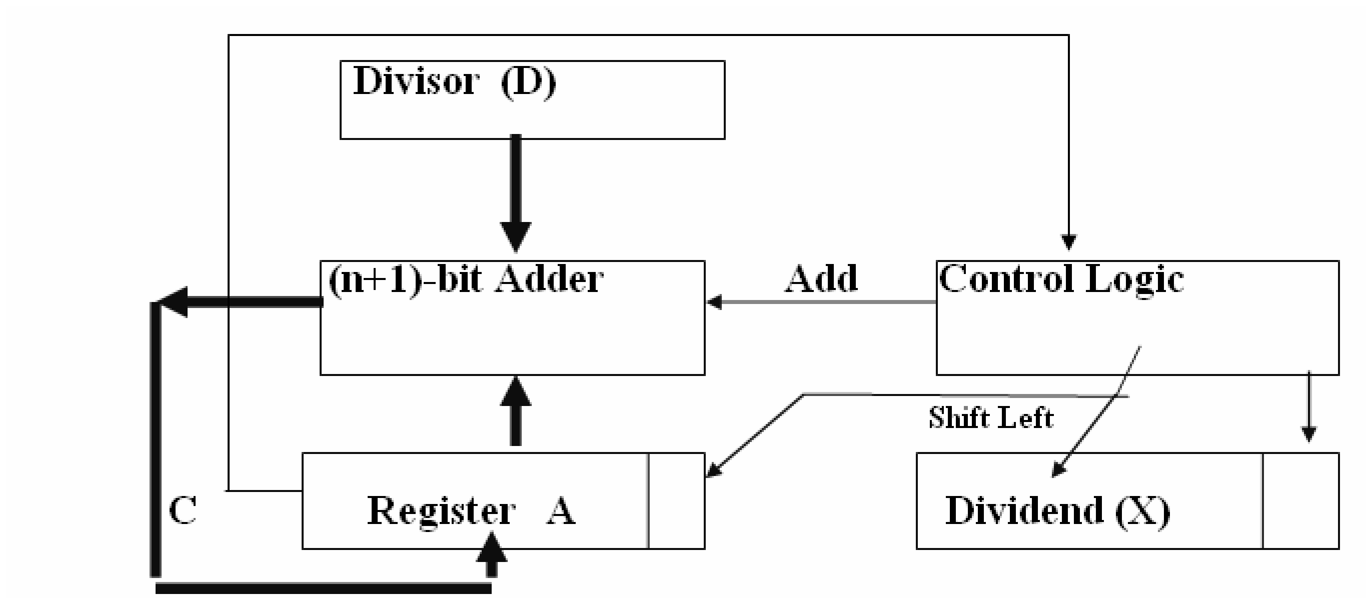


4.2 Integer Arithmetic

- Two's Complement Arithmetic
  - Division:
    - Among the four basic arithmetic operations, division is considered the most complex and most time consuming.
    - An integer division operation takes two arguments, the dividend *X* and the divisor *D*.
    - It produces two outputs, the quotient *Q* and the remainder *R*.
    - The four quantities satisfy the relation $X = Q \times D + R \text{ where } R < D$
    - A number of complications arise when dealing with division:
      - *D* = 0
      - the requirement that the resulting quotient should not exceed the capacity of the register holding it.
        - » This can be satisfied if $Q < 2^{n-1}$, where *n* is the number of bits in the register holding the quotient. This implies that the relation $X < 2^{n-1}D$ must also be satisfied. Failure to satisfy any of the above conditions will lead to an *overflow* condition.

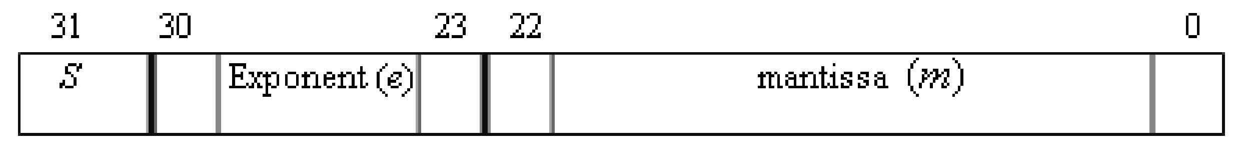
4.2 Integer Arithmetic

- Two's Complement Arithmetic
  - Division – Binary division structure



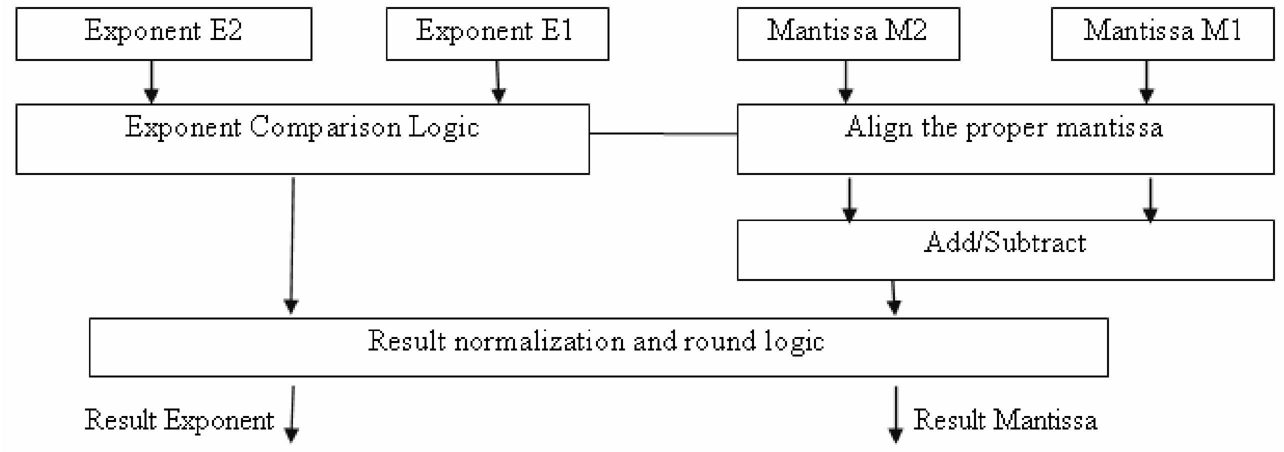
4.3 Floating-Point Arithmetic

- Floating-Point Representation (Scientific Notation)
  - A floating-point (FP) number can be represented in the following form: $\pm m * b^{e}$
    - where *m*, called the *mantissa*, represents the fraction part of the number and is normally represented as a signed binary fraction,
    - *e* represents the exponent, and
    - *b* represents the base (radix) of the exponent.



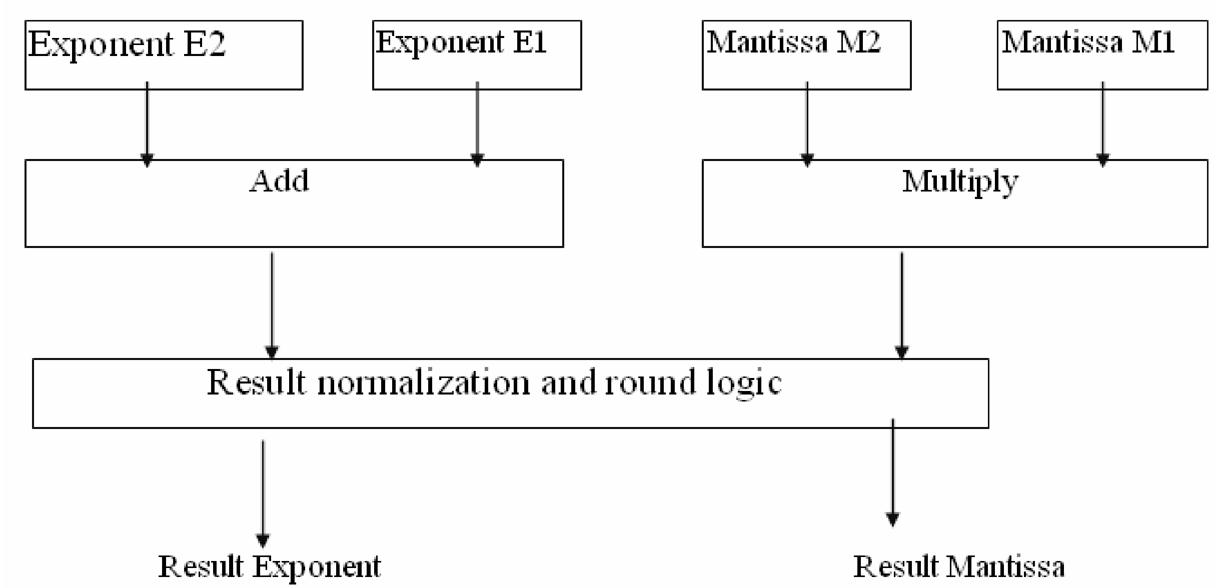
4.3 Floating-Point Arithmetic

- Floating-Point Representation (Scientific Notation)
  - Steps required to add/subtract two floating-point numbers:
    - Compare the magnitude of the two exponents and make suitable *alignment* to the number with the smaller magnitude of exponent.
    - Perform the addition/subtraction
    - Perform *normalization* by shifting the resulting mantissa and adjusting the resulting exponent.


Exponent E2
Exponent E1
Mantissa M2
Mantissa M1
Exponent Comparison Logic
Align the proper mantissa
Add/Subtract
Result normalization and round logic
Result Exponent
Result Mantissa

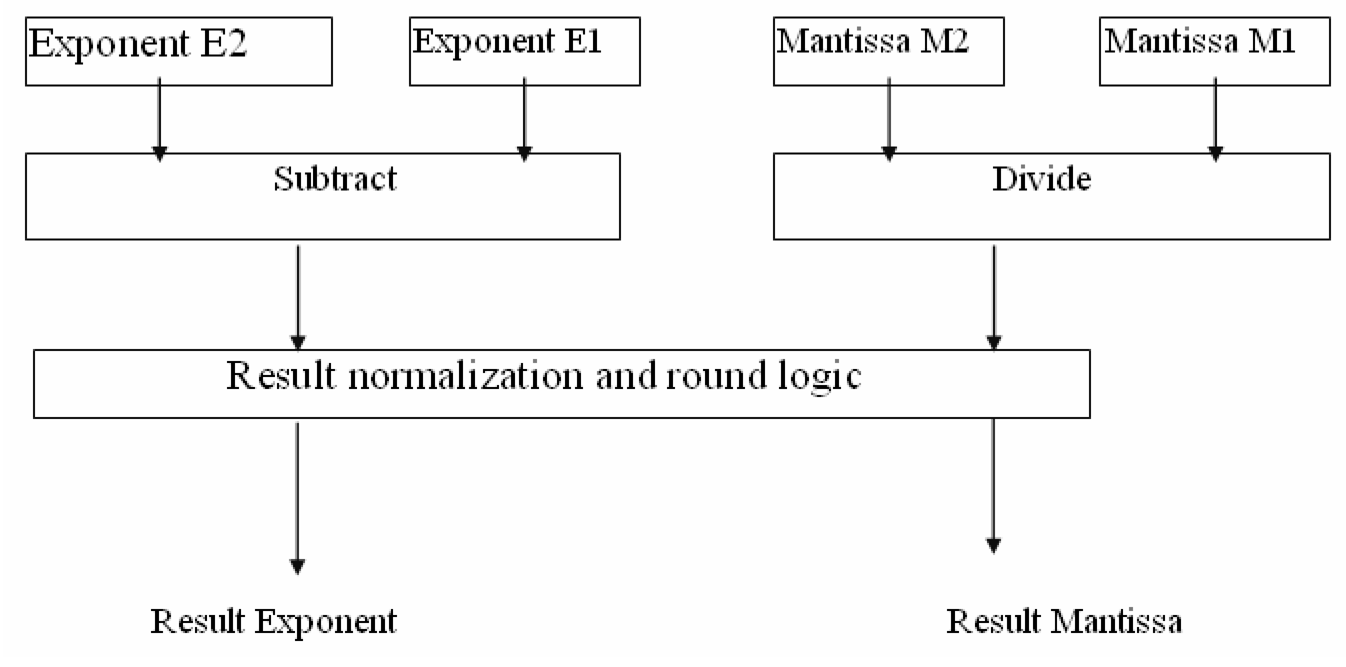
4.3 Floating-Point Arithmetic

- Floating-Point Representation (Scientific Notation)
  - Steps required to multiply two floating-point numbers:
    - Compute the exponent of the product by adding the exponents together,
    - Multiply the two mantissas, and
    - Normalize and round the final product.



4.3 Floating-Point Arithmetic

- Floating-Point Representation (Scientific Notation)
  - Steps required to divide two floating-point numbers:
    - Compute the exponent of the result by subtracting the exponents,
    - Divide the mantissa & determine the sign of the result, and
    - Normalize and round the resulting value, if necessary.

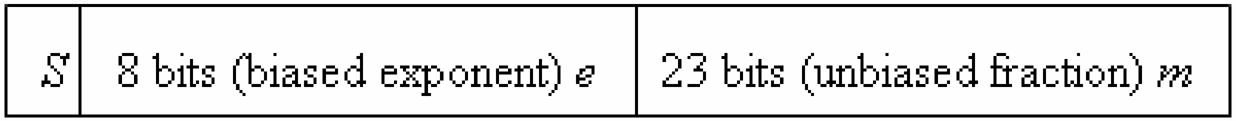


4.3 Floating-Point Arithmetic

- The IEEE Floating-Point Standard
  - There are basically two IEEE standard floating-point formats:
    - These are the *basic* and the *extended* formats.
    - In each of these, IEEE defines two formats:
      - the single-precision and
      - the double-precision.
    - The single-precision format is 32-bit and the double-precision is 64-bit.
    - The single extended format should have at least 44 bits and the double extended format should have at least 80 bits.

4.3 Floating-Point Arithmetic

- The IEEE Floating-Point Standard
  - The IEEE single-precision representation:



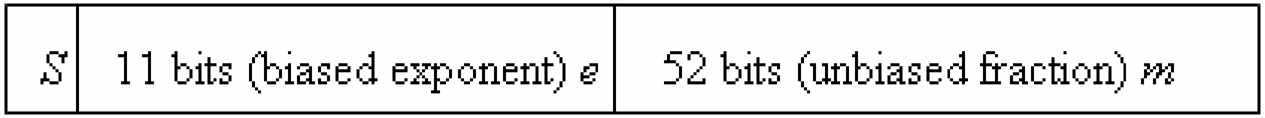
  - In the single-precision format, base 2 is used, thus allowing the use of a hidden bit. The exponent field is 8 bits.

4.3 Floating-Point Arithmetic

- The 8-bit exponent allows for any of 256 combinations. Among these, two combinations are reserved for special values. These are:
  - *e=0* is reserved for zero (with fraction *m = 0)* and denormalized numbers (with fraction *m≠0*).
  - *e=255* is reserved for ±∞ (with fraction *m=0*) and not a number (NaN) (with fraction *m≠0*).
- The single extended IEEE format extends the exponent field from 8 to 11 bits and the mantissa field from 23+1 to 32 resulting in a total length of at least 44 bits.
- The single extended format is used in calculating intermediate results.

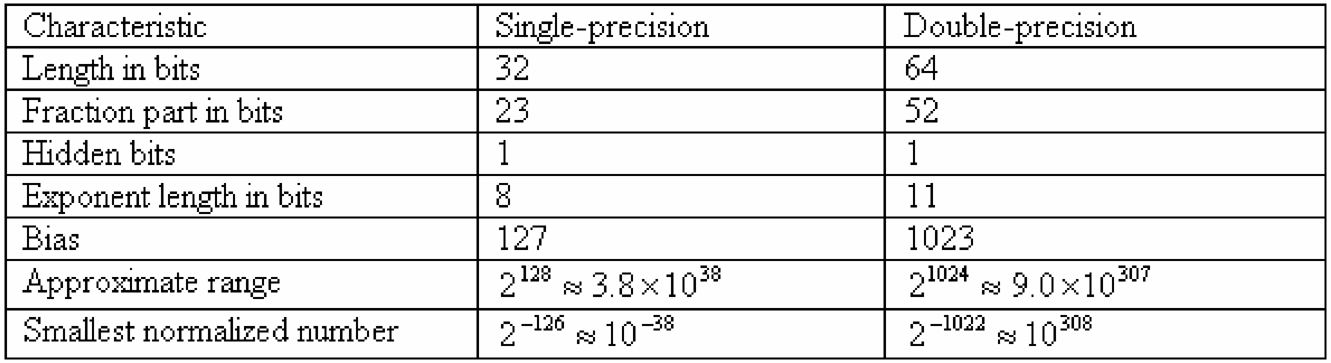
4.3 Floating-Point Arithmetic

- Double-precision IEEE Format
  - The exponent field is 11 bits and the significant field is 52 bits.
  - The format is:



S	11 bits (biased exponent) e	52 bits (unbiased fraction) m

- Characteristics of the IEEE single and double floating-point formats



Characteristic	Single-precision	Double-precision
Length in bits	32	64
Fraction part in bits	23	52
Hidden bits	1	1
Exponent length in bits	8	11
Bias	127	1023
Approximate range	$2^{128} \approx 3.8 \times 10^{38}$	$2^{1024} \approx 9.0 \times 10^{307}$
Smallest normalized number	$2^{-126} \approx 10^{-38}$	$2^{-1022} \approx 10^{308}$

4.4 Summary

- In this chapter, a number of issues related to computer arithmetic were discussed.
- The discussion started with an introduction to number representation and radix conversion techniques.
- We then discussed integer arithmetic and in particular, the four main operations, i.e., addition, subtraction, multiplication, and division.
  - In each case, we have shown basic architectures and organization.

4.4 Summary

- The last topic discussed in the chapter was floating-point representation and arithmetic.
- We have also shown the basic architectures needed to perform basic floating-point operations.
- We ended our discussion in the chapter with the IEEE floating-point number representation.