

## Floating Point Arithmetic operation

### Basic Considerations :

There are two part of a floating-point number in a computer - a mantissa  $m$  and an exponent  $e$ . The two parts represent a number generated from multiplying  $m$  times a radix  $r$  raised to the value of  $e$ . Thus

$$m \times r^e$$

The mantissa may be a fraction or an integer. The position of the radix point and the value of the radix  $r$  are not included in the registers. For example, assume a fraction representation and a radix 10. The decimal number 537.25 is represented in a register with  $m = 53725$  and  $e = 3$  and is interpreted to represent the floating-point number

$$.53725 \times 10^3$$

A floating-point number is said to be normalized if the most significant digit of the mantissa is nonzero. So the mantissa contains the maximum possible number of significant digits. We cannot normalize a zero because it does not have a nonzero digit. It is represented in floating-point by all 0's in the mantissa and exponent.

Floating-point representation increases the range of numbers for a given register. Consider a computer with 48-bit words. Since one bit must be reserved for the sign, the range of fixed-point integer numbers will be  $+(2^{47} - 1)$ , which is approximately  $+10^{14}$ . The 48 bits can be used to represent a floating-point number with 36 bits for the mantissa and 12 bits for the exponent. Assuming fraction representation for the mantissa and taking the two sign bits into consideration, the range of numbers that can be represented is

$$+(1 - 2^{-35}) \times 2^{2047}$$

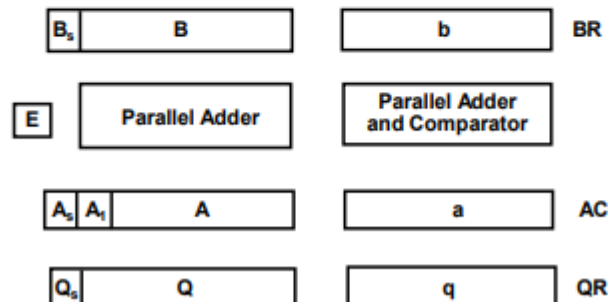


## FLOATING POINT ARITHMETIC OPERATIONS

$$F = m \times r^e$$

where m: Mantissa  
r: Radix  
e: Exponent

### Registers for Floating Point Arithmetic



Assuming that each floating-point number has a mantissa in signed-magnitude representation and a biased exponent. Thus the AC has a mantissa whose sign is in  $A_s$ , and a magnitude that is in A. The diagram shows the most significant bit of A, labeled by  $A_1$ . The bit in this position must be a 1 to normalize the number. Note that the symbol AC represents the entire register, that is, the concatenation of  $A_s$ , A and a.

In the similar way, register BR is subdivided into  $B_s$ , B, and b and QR into  $Q_s$ , Q and q. A parallel-adder adds the two mantissas and loads the sum into A and the carry into E. A separate parallel adder can be used for the exponents. The exponents do not have a distinct sign bit because they are biased but are represented as a biased positive quantity. It is assumed that the floating-point number are so large that the chance of an exponent overflow is very remote and so the exponent overflow will be neglected. The exponents are also connected to a magnitude comparator that provides three binary outputs to indicate their relative magnitude.

The number in the mantissa will be taken as a fraction, so the binary point is assumed to reside to the left of the magnitude part. Integer representation for floating point causes certain scaling problems during multiplication and division. To avoid these problems, we adopt a fraction representation.

The numbers in the registers should initially be normalized. After each arithmetic operation, the result will be normalized. Thus all floating-point operands are always normalized.

### Addition and Subtraction of Floating Point Numbers

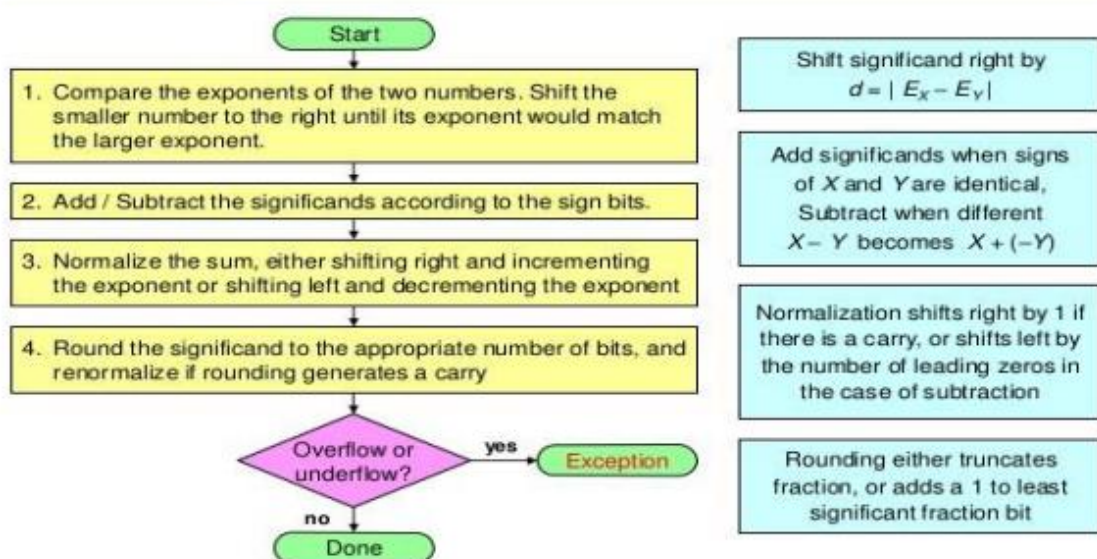
During addition or subtraction, the two floating-point operands are kept in AC and BR. The sum or difference is formed in the AC. The algorithm can be divided into four consecutive parts:

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

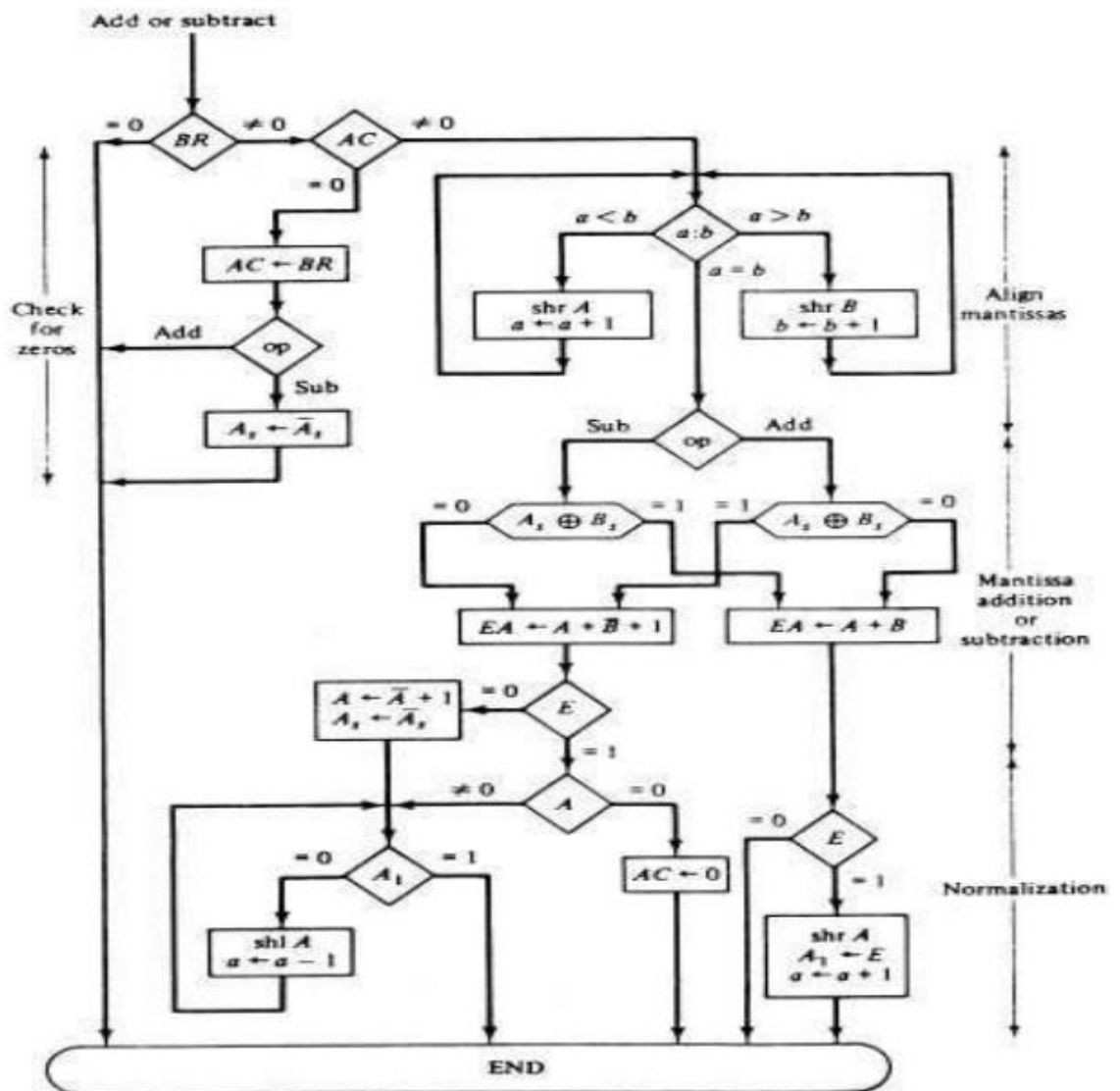
A floating-point number cannot be normalized, if it is 0. If this number is used for computation, the result may also be zero. Instead of checking for zeros during the normalization process we check for zeros at the beginning and terminate the process if necessary. The alignment of the mantissas must be carried out prior to their operation. After the mantissas are added or subtracted, the result may be un-normalized. The normalization procedure ensures that the result is normalized before it is transferred to memory.

If the magnitudes were subtracted, there may be zero or may have an underflow in the result. If the mantissa is equal to zero the entire floating-point number in the AC is cleared to zero. Otherwise, the mantissa must have at least one bit that is equal to 1. The mantissa has an underflow if the most significant bit in position A1, is 0. In that case, the mantissa is shifted left and the exponent decremented. The bit in A1 is checked again and the process is repeated until A1 = 1. When A1 = 1, the mantissa is normalized and the operation is completed.

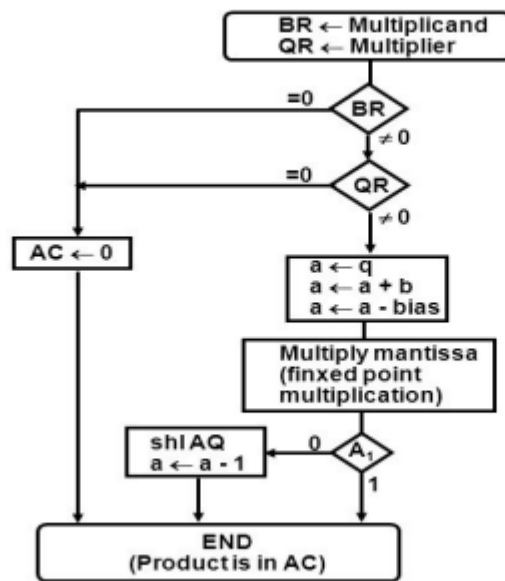
## Floating Point Addition / Subtraction



### Algorithm for Floating Point Addition and Subtraction



## FLOATING POINT MULTIPLICATION



## FLOATING POINT DIVISION

