Department Of Computer Application (BCA)
Dr. Rakesh Ranjan

BCA Sem - 2

Computer Organization and Architecture

Floating point representation

Floating Point representation : Floating point number are the number containing two different part – integer part and the fractional part . It is also refers to real number. A notation known as scientific notation which is used to represent real number in computer system. Scientific number may be very large or very small. It is also called exponential notation. It takes the form :

$M \times B^e$

Here M is real value called mantissa

B is the base of the system

E is the integer value called exponent

For example :

$325.123 \rightarrow 3.25123 \times 10^2 \rightarrow 0.325123 \times 10^3$

$0.000000245 \rightarrow 0.245 \times 10^{-6} \rightarrow 2.45 \times 10^{-7}$

The binary number can be represented in the scientific notation by keeping base 2.

Ex:- $1000.0101 \rightarrow 1.0000101 \times 2^3 \rightarrow 0.10000101 \times 2^4$

To represent binary floating point in computer system we use normalized floating form.

Ex un-normalized form	normalized form
$4.3333123 \rightarrow$	$0.43333123 \times 10^{1} \rightarrow 433.33123 \times 10^{-2}$
$567.6543 \rightarrow$	0.5676543×10^{3}
$897654.23 \rightarrow$	0.897654×10^{6}

The binary floating point format store the binary number by dividing into three part : -

1. Sign → the sign part is always 1-bit long and is used to store the information pertaining to the sign of the number.
2. Exponent :- the exponent part always store using excess-N notation. The N- depends upon the number of bit available for storing the exponent . if the exponent is stored in 7 bit the excess-64 method is used. And if 8-bit then excess-127 and so no method is used .
3. Mantissa -> the mantissa of the number is always stored in normalized form of notation.

   Generally floating point data is represented in 32-bit form but in earlier it was represented as 16 bit or at present in 64 bit or 128 bit

   The 16 bit format is

Sign bit	6 bit exponent	9-bit mantissa

32- bit format

Sign bit	8/7 bit Exponent	23/24 bit mantissa

64- bit format

Sign bit	11/10 bit for exponent	52/53 bit for mantissa

Ex 16 bit representation :-

+1010.001 → 0.1010001×2^4

Exponent 6 –bit -> 4 +16 =20 = 010100

Mantissa → 1010001

0	010100	101000100

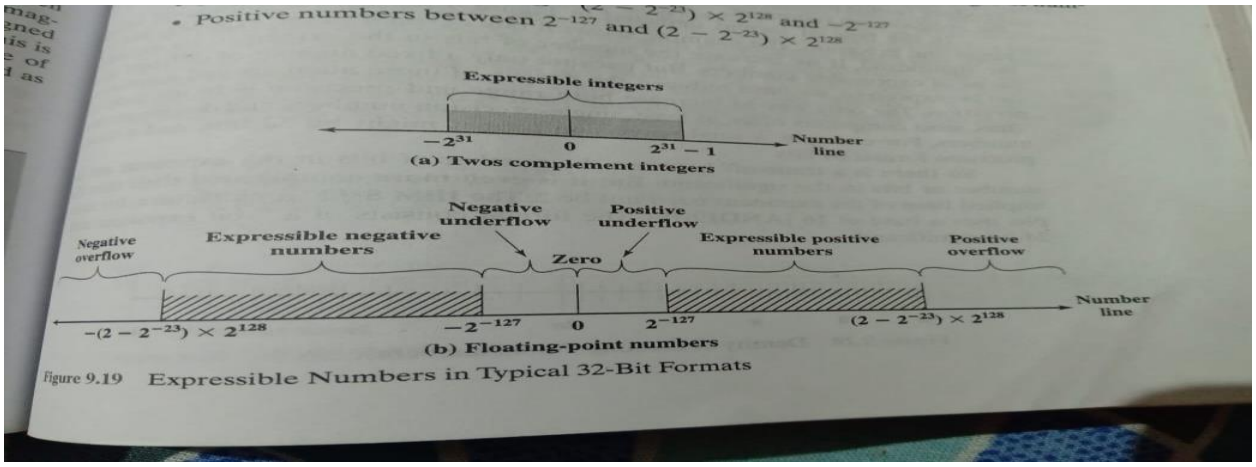

Figure 9.19 Expressible Numbers in Typical 32-Bit Formats

Binary floating point representation : To represent the binary floating point number we represent the number in the form $\pm S \times B^{\pm e}$

The number is stored in binary word with three field

1. Sign : plus /minus
2. Significant S
3. Exponent E

The base B is implicit and need not be stored because it is same for all the number. It is assumed that the radix point is to the right of the leftmost or most significant bit of significant . Sign is represented 0- positive and 1- negative

The exponent value is stored in next 8 bit . the representation is known as biased representation. A fix value is known as bias, is subtracted from the field to get the true exponent value. Bias is equals $(2^{k-1} -1)$ where k is the number of bits in binary exponent.

In 8-bit exponent it will be 127

The significant is generally represented in normalized form in which the most significant digit of the significant is nonzero . for base 2 representation , a normalized number is therefore one in which the most significant bit of the significant is one i.e $\pm 1.bbbbbb\ldots \times 2^{\pm e}$ where b is either 0 or 1.

Here, the left significant is always 1 in case of binary therefore , we do not store this bit and significant is represented in 23-bit. So, the rule is

1. The sign bit is stored in the first bit of the word
2. The first bit of the true significant is always 1 and need not be stored in the significant field .
3. The 127 is added to the true exponent and stored in the exponent field
4. The base is 2 .

$1.6328125 \times 2^{20} \rightarrow 1.1010001 \times 2^{10100}$

0	10010011	10100010000000000000000

-1.6328125×2^{20}

1	10010011	10100010000000000000000

1.6328125×2^{-20}

0	01101101	10100010000000000000000

$127-20 = 107$

$-1.6328125 \times 2^{-20}$

1	01101101	10100010000000000000000

Errors in Arithmetic

Computer Arithmetic errors can be divided into two category

1. Arithmetic error
2. Floating point error

Arithmetic error are the errors generated by the computer system while performing various integer arithmetic operations.

Floating point arithmetic errors are generated in the computer system while performing various floating point arithmetic operations.

Followings are some errors :-

1. Overflow : the overflow error may occur when the computer system attempts to use a number in arithmetic operation that is too large to be handled .
2. Underflow :- may occur when the computer system attempts to use the number arithmetic operation that is too small to be handled . It may generally occur in subtraction and division operation.
3. Truncate error :- this error may occur when computer system attempt to convert one data into another data.
4. Sign error :- it occur when computer system attempt to represent unsigned number in sign representation and vice –versa

Above errors also occur in floating point calculation but some floating point error occur due to

a. The shifting of mantissa to the right for the purpose of representing real value in normalized form.
b. Conversion of floating point decimal into floating point binary
c. Round-off of a very large fraction to the desired number of digits.

0.0000000000010101110

1.0101110×2^{-12}

$.0101110 \times 2^{-00001100\ (-139)}$

HW:

Perform addition

1. 0.001010110 E 0101000 and 0.010001001 E 0100011
2. 0.110110100 e 100011 and 0.110001010 e 100001

Perform subtraction

1. 0.101100011 e 0001001 and 0.011011011 e 0000110

Multiply above number operation

----------x----------