Floating Point Tutorial

This floating point **tutorial** covers **IEEE 754 Standard Floating Point** Numbers,floating point conversions,Decimal to IEEE 754 standard floating point, floating point standard to Decimal point conversion,floating point Arithmetic,IEEE 754 standard Floating point multiplication Algorithm ,floating point Addition Algorithm with example,floating point Division Algorithm with example and more.

IEEE 754 Standard Floating Point Numbers

This Tutorial attempts to provide a brief overview of IEEE Floating point Numbers format with the help of simple examples, without going too much into mathematical detail and notations. At the end of this tutorial we should be able to know what are floating point numbers and its basic arithmetic operations such as addition, multiplication & division. An IEEE 754 standard floating point binary word consists of a sign bit, exponent, and a mantissa as shown in the figure below. IEEE 754 single precision floating point number consists of 32 bits of which
1 bit = sign bit(s).
8 = Biased exponent bits (e)
23 = mantissa (m).

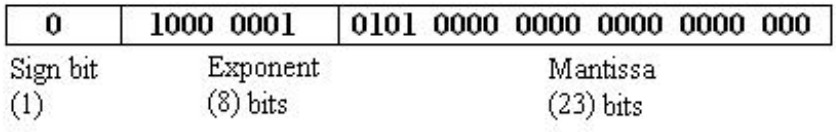


Fig 1: IEEE 754 Floating point standard floating point word

The Decimal value of a normalized floating point numbers in IEEE 754 standard is represented as

$$\text{Decimal Equivalent Value} = (-1)^{s} \times 1.m \times 2^{(e-bias)}$$

$$bias = 2^{(e-1)} - 1$$

Fig 2: Equation-1

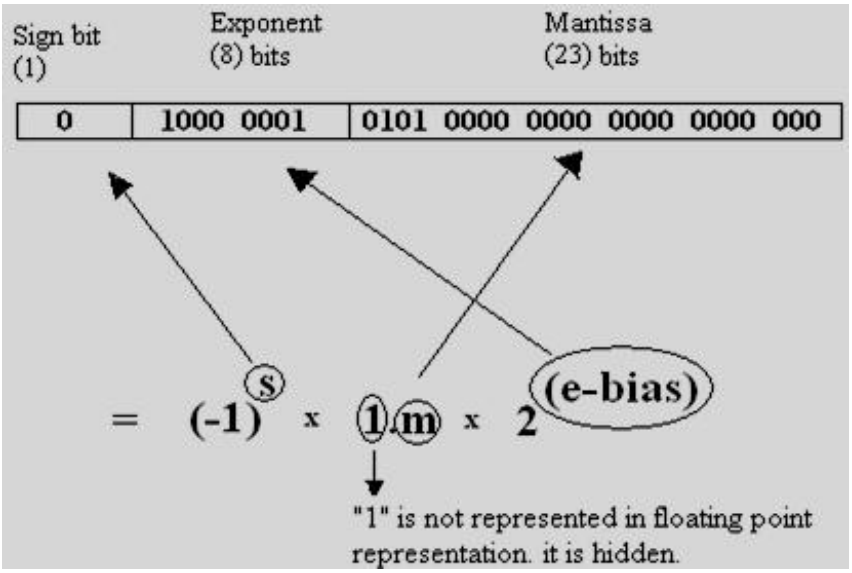


Fig 3

Note: "1" is hidden in the representation of IEEE 754 floating point word, since it takes up an extra bit location and it can be avoided. It is understood that we need to append the "1" to the mantissa of a floating point word for conversions are calculations.

For example in the above fig 1: the mantissa represented is 0101_0000_0000_0000_0000_000 in actual it is (1.mantissa) =1. 0101_0000_0000_0000_0000_000.

To make the equation 1, more clear let's consider the example in figure 1.lets try and represent the floating point binary word in the form of equation and convert it to equivalent decimal value.

Floating point binary word X1=

	S1	E1	M1
X1 =	0	1000 0001	0101 0000 0000 0000 0000 000

Fig 4

Sign bit (S1) =0. Biased Exponent (E1) =1000_0001 (2) = 129(10). Mantissa (M1) =0101_0000_0000_0000_0000_000

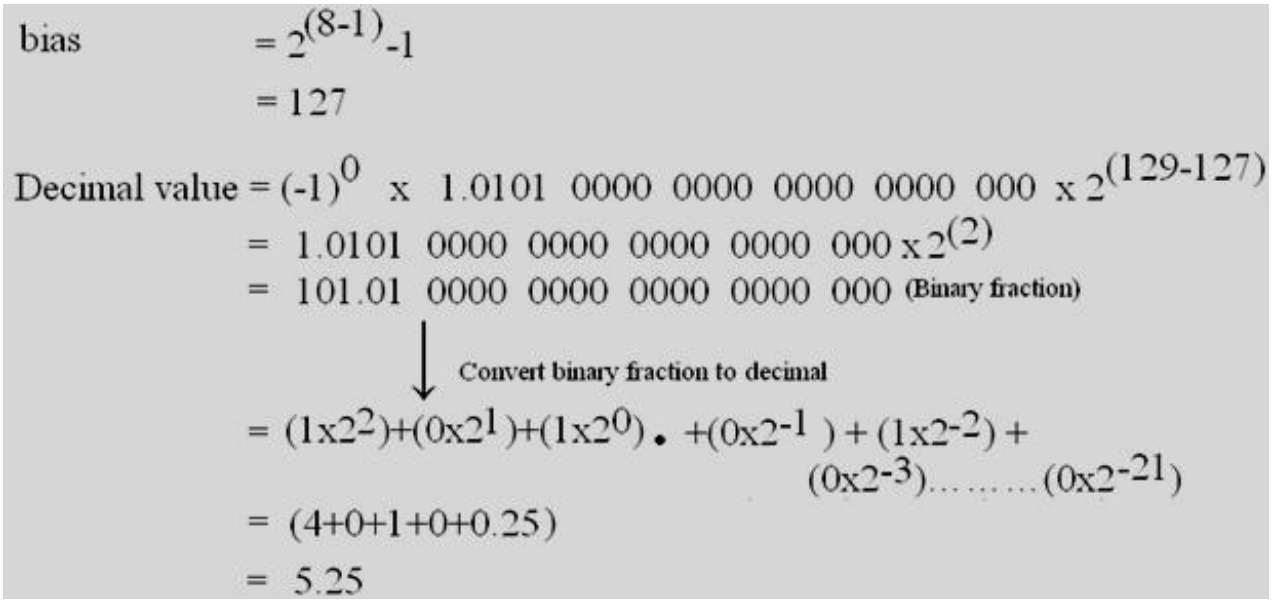


Fig 5

IEEE 754 standard floating point conversions

Let's look into an example for decimal to IEEE 754 floating point number and IEEE 754 floating point number to decimal conversion, this will make much clear the concept and notations of floating point numbers.

Decimal to IEEE 754 standard floating point

Let take a decimal number say 286.75 lets represent it in IEEE floating point format (Single precision, 32 bit). We need to find the Sign, exponent and mantissa bits.

1) Represent the Decimal number $286.75_{(10)}$ into Binary format
$286.75_{(10)} = 100011110.11_{(2)}$

2) The binary number is not normalized, Normalize the binary number. Shift the decimal point such that we get a 1 at the very end (i.e 1.m form).

Fig 6

We had to shift the binary points left 8 times to normalize it; exponent value (8) should be added with bias. We got the value of mantissa.

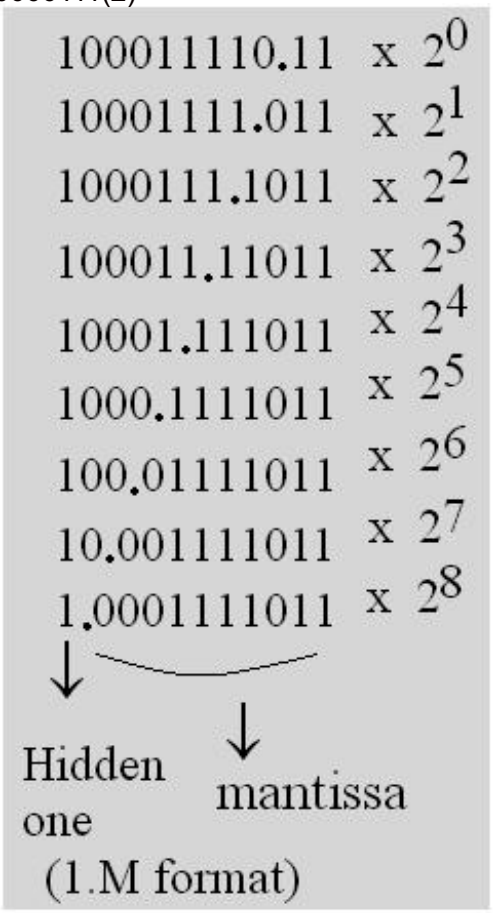
Note: In Floating point numbers the mantissa is treated as fractional fixed point binary number, Normalization is the process in which mantissa bits are either shifted right or to the left(add or subtract the exponent accordingly) Such that the most significant bit is "1".

3) Bias $=2^{(e-1)} - 1$,
In our case e=8(IEEE 754 format single precision).
Bias = $2^{(8-1)} - 1 = 127$
(This is the bias value for single precision IEEE floating point format).

4) The biased exponent e is represented as E = exponent vale obtained after normalization in step 2 + bias E = 8 +

127 = 135(10) , convert this to binary and we have our exponent value E = 10000111(2)

5) We have our floating point number equivalent to 286.75



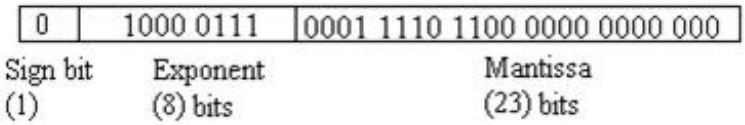


Fig 7

Now with the above example of decimal to floating point conversion, it should be clear so as to what is mantissa, exponent & the bias.

IEEE 754 standard floating point standard to Decimal point conversion

Lets inverse the above process and convert back the floating point word obtained above to **decimal**. We have already done this in section 1 but for a different value.

$$
\begin{aligned}
\text{Decimal num} &= 1.\ 0001\ 1110\ 1100\ 0000\ 0000\ 000 \times 2^{(e\text{-bias})} \\
&= 1.\ 0001\_1110\_1100\_0000\_0000\_000 \times 2^{(135-127)} \\
&= 1.\ 0001\_1110\_1100\_0000\_0000\_000 \times 2^{(8)} \\
&= 100011110.\ 11000000000000 \text{(binary fraction)} \\
&= 286.75(10)
\end{aligned}
$$

Fig 8

IEEE 754 standard floating point Arithmetic

Let us look at **Multiplication**, **Addition**, **subtraction** & **inversion** algorithms performed on IEEE 754 floating point standard. Let us consider the IEEE 754 floating point format numbers X1 & X2 for our calculations.

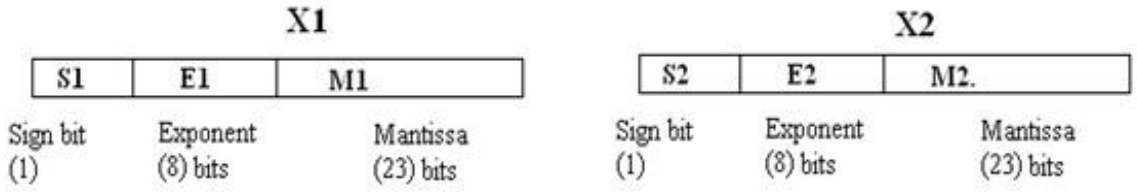


Fig 9

IEEE 754 standard Floating point multiplication Algorithm

A brief overview of floating point multiplication algorithm have been explained below, X1 and X2.

Result X3 = X1 * X2
$= (-1)^{s1} (M1 \times 2^{E1}) * (-1)^{s2} (M2 \times 2^{E2})$

S1, S2 => Sign bits of number X1 & X2.
E1, E2: =>Exponent bits of number X1 & X2.
M1, M2 =>Mantissa bits of Number X1 & X2.

1) Check if one/both operands = 0 or infinity. Set the result to 0 or inf. i.e. exponents = all "0" or all "1".
2) S1, the signed bit of the multiplicand is XOR'd with the multiplier signed bit of S2. The result is put into the resultant sign bit.
3) The mantissa of the Multiplier (M1) and multiplicand (M2) are multiplied and the result is placed in the resultant field of the mantissa (truncate/round the result for 24 bits).
=M1 * M2
4) The exponents of the Multiplier (E1) and the multiplicand (E2) bits are added and the base value is subtracted from the added result. The subtracted result is put in the exponential field of the result block.
=E1+E2-bias
5) Normalize the sum, either shifting right and incrementing the exponent or shifting left and decrementing the exponent.
6) Check for underflow/overflow. If Overflow set the output to infinity & for underflow set to zero.
7) If (E1 + E2 - bias) >= to Emax then set the product to infinity.
8) If E1 + E2 - bias) is lesser than/equal to Emin then set product to zero.

Example:

Floating Point Multiplication is simpler when compared to floating point addition. Let's try to understand the Multiplication algorithm with the help of an example.

Let's consider two decimal numbers
X1 = 125.125 (base 10)
X2 = 12.0625 (base 10)

X3= X1 * X2 = 1509.3203125
Equivalent floating point binary words are
X1 =

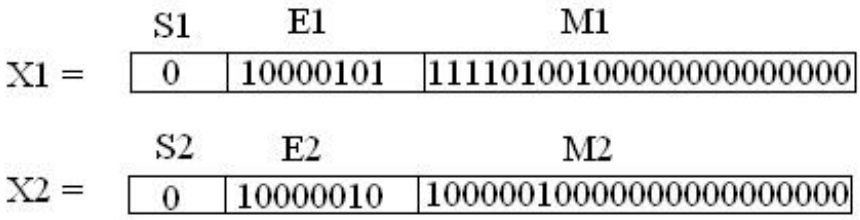


Fig 10

1) Find the sign bit by xor-ing sign bit of A and B
i.e. Sign bit = > (0 xor 0) => 0

2) Multiply the mantissa values including the "hidden one". The Resultant product of the 24 bits mantissas (M1 and M2) is 48bits (2 bits are to the left of binary point)

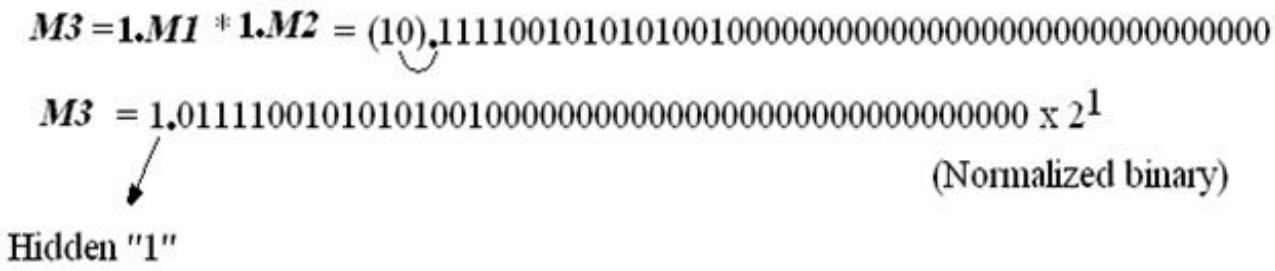


Fig 11

If M3 (48) = "1" then left shift the binary point and add "1" to the exponent else don't add anything. This normalizes the mantissa. Truncate the result to 24 bits. Add the exponent "1" to the final exponent value.

3) Find exponent of the result. = E1 + E2 -bias + (normalized exponent from step 2) = (10000101)2 + (10000010)2 - bias +1 = 133 + 130 - 127 + 1 = 137.

Add the exponent value after normalization to the biased exponent obtained in step 2. i.e. 136+1 = 137 => exponent value.

Note: The normalization of the product is simpler as the range of M_A and M_B is between 1 - 1.9999999.and the range of the product is between (1 - 3.9999999)Therefore a 1 bit shift is required with the adjust of exponent. So we have found mantissa, sign, and exponent bits.

4) We have our final result i.e.

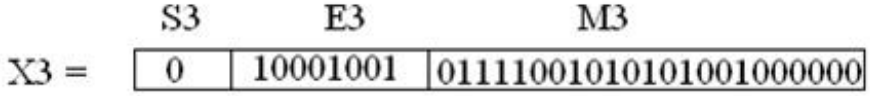


Fig 12

If we convert this to decimal we get X=1509.3203125

Floating Point Multiplication is simpler when compared to floating point addition we will discuss the basic floating point multiplication algorithm. The simplified floating point multiplication chart is given in Figure 4.

IEEE 754 standard floating point Addition Algorithm

Floating-point addition is more complex than multiplication, brief overview of floating point addition algorithm have been explained below
X3 = X1 + X2
$X3 = (M1 \times 2^{E1}) +/- (M2 \times 2^{E2})$

1) X1 and X2 can only be added if the exponents are the same i.e E1=E2.
2) We assume that X1 has the larger absolute value of the 2 numbers. Absolute value of of X1 should be greater than absolute value of X2, else swap the values such that Abs(X1) is greater than Abs(X2).
Abs(X1) > Abs(X2).
3) Initial value of the exponent should be the larger of the 2 numbers, since we know exponent of X1 will be bigger , hence Initial exponent result E3 = E1.
4) Calculate the exponent's difference i.e. Exp_diff = (E1-E2).
5) Left shift the decimal point of mantissa (M2) by the exponent difference. Now the exponents of both X1 and X2 are same.
6) Compute the sum/difference of the mantissas depending on the sign bit S1 and S2.
If signs of X1 and X2 are equal (S1 == S2) then add the mantissas
If signs of X1 and X2 are not equal (S1 != S2) then subtract the mantissas
7) Normalize the resultant mantissa (M3) if needed. (1.m3 format) and the initial exponent result E3=E1 needs to be adjusted according to the normalization of mantissa.
8) If any of the operands is infinity or if (E3>Emax) , overflow has occurred ,the output should be set to infinity. If(E3 < Emin) then it's a underflow and the output should be set to zero.
9) Nan's are not supported.

IEEE 754 standard floating point addition Example:

A = 9.75
B = 0.5625
Equivalent floating point binary words are

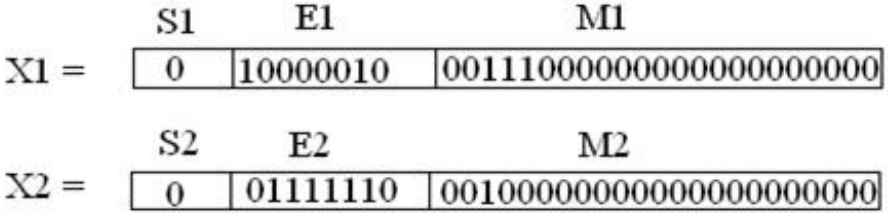


Fig 13

1) Abs (A) > Abs (B)? Yes. 2) Result of Initial exponent E3 = E1 = 10000010 = 130(10) 3) E1 - E2 = (10000010 - 01111110) => (130-126)=4 4) Shift the mantissa M2 by (E1-E2) so that the exponents are same for both numbers.

$$1.M2 = 1.00100000000000000000000$$
$$= 00001.00100000000000000000000$$
$$= 0.00010010000000000000000$$
(Aligned mantissa)

Fig 14

5) Sign bits of both are equal? Yes. Add the mantissa's

Fig 15

$$\begin{array}{rl} & 1.00111000000000000000000 \ (1.M1) \\ + & 0.00010010000000000000000 \ (\text{aligned M2}) \\ \hline & 1.01001010000000000000000 \ \ 1.M3 \end{array}$$

6) Normalization needed? No, (if Normalization was required for M3 then the initial exponent result E3=E1 should be adjusted accordingly)
7) Result

	S3	E3	M3
X3 =	0	10000010	01001010000000000000000

Fig 16

X3 in decimal = 10.3125.
8) If we had to perform subtraction, just change the sign bit of X2 to "1",
Then we would have subtracted the mantissas. Since sign bits are not equal.

$$\begin{array}{rl} & 1.00111000000000000000000 \ (1.M1) \\ - & 0.00010010000000000000000 \ (\text{aligned M2}) \\ \hline & 1.00100110000000000000000 \ \ 1.M3 \end{array}$$

	S3	E3	M3
X3 =	0	10000010	00100110000000000000000

X3 in decimal = 9.1875.

Fig 17

NOTE: For floating point Subtraction, invert the sign bit of the number to be subtracted And apply it to floating point Adder

IEEE 754 standard floating point Division Algorithm

Division of IEEE 754 Floating point numbers (X1 & X2) is done by dividing the mantissas and subtracting the

exponents.

X3 = (X1/X2)
= $(-1)^{S1}$ (M1 x 2^{E1}) / $(-1)^{S2}$ (M2 x 2^{E2})
= $(-1)^{S3}$(M1/M2) $2^{(E1-E2)}$

1) If divisor X2 = zero then set the result to "infinity", if both X1 and X2 are zero's set it to "NAN"
2) Sign bit S3 = (S1 xor S1).
3) Find mantissa by dividing M1/M2
4) Exponent E3 = (E1 - E2) + bias
5) Normalize if required, i.e by left shifting the mantissa and decrementing the resultant exponent.
6) Check for overflow/underflow
If E3 > Emax return overflow i.e. "infinity"
If E3 < Emin return underflow i.e. zero

IEEE 754 standard floating point Division Example:

X1=127.03125
X2=16.9375
Equivalent floating point binary words are

	S1	E1	M1
X1 =	0	10000101	11111100001000000000000

	S2	E2	M2
X2 =	0	10000011	00001111000000000000000

Fig 18

1) S3 = S1 xor S2 = 0
2) E3 = (E1 - E2) + bias = (10000101) - (10000011)+ (1111111)
= 133-131+127 => 129 => (10000001)
3) Divide the mantissas M1/M2

Fig 19

4) Result

$$\begin{array}{rll} & 1.11111100001000000000000 & (1.M1) \\ \div & 1.00001111000000000000000 & (1.M2) \\ \hline & 1.11100000000000000 & \mathbf{1.}\boldsymbol{M3} \end{array}$$

	S3	E3	M3
X3 =	0	10000001	11100000000000000000000

Fig 20

X3 in decimal = 7.5

	Single Precision	Double precision
Word Length(bits)	32	64
Mantissa(bits)	23	52
Exponent(bits)	8	11
Bias	127	1023
Range	3.8×10^{38}	1.8×10^{308}

Table 1: IEEE 754 Floating point Standard

Sign	Exponent	Mantissa(Significant)	Number Represented	Comments
0	00000000	00000000000000000000000	+/- 0	
0	00000000	Non Zero 00000000000000000000001 00000000000000000000010 00000000000000000000011 11111111111111111111111	 +1.4E-45 +2.8E-45 +4.2E-45 1.1754942E-38	Denormalized number If the Exponent is all zeros and mantissa is non zero then the numbers Represented are known as Denormalized numbers. Other term is Denormals.
1	00000000	Non Zero		"- ive" Denormalized number
0	1 to 254	Any number		"+ ive "Floating Point Numbers
1	1 to 254	Any number		"- ive "Floating Point Numbers
0/1	255 (11111111)	Zero	+/- Infinity	
0/1	255 (11111111)	Non Zero	NaN	"Not a Number" (0/0),(inf+/-inf),(0xinf) etc produce Nan's

Table 2: IEEE 754 Number Representation Table