2. Algorithm for binary floating point form representation

1) Define if the input number y is one. In that case, select single precision.

2) Show an error message if the number is out of range (-2^{100} and 2^{100}) and exit.

3) Find the e which defines the largest number of 2^e that provides a number smaller than y. Note that 2^{e+1} is larger than y: $2^{e+1}>|y|>2^e$. Use absolute value of y in all calculations due to the fact that the sign will be defined separately. The exponent is e.

4) Divide the input number y with 2^e and define the mantissa m= |y|/2e -1

5) Define bias, exponent bits (eb) and mantissa bits (mb) according to single or double precision.

6) Exponent becomes ex = e+bias

7) Define a zero vector mbit(1,mb). Write a loop from 1 to mb to define the bits of the mantissa m. Multiply the mantissa by 2 and check if it is larger than 1, in that case the bit is 1 and the new value of m=m -1 (the previous one minus 1). If it is equal to 1, the bit is 1 and the loop is terminated. If m is smaller than 1 then the bit is zero.

8) Define a zero vector ebit=zeros(1,eb). Write a loop from 1 to eb. Define the remainder and quotient. If the remainder is 0, then the bit is 0, if the remainder is 1, then the bit is 1. In both cases, if the quotient is equal to zero break the loop. Store the values in the loop as follows: (eb+1-i), where i is the number of iteration in the loop. For example when i=1 we store in element eb of array ebit and when i=eb we store the value in element 1 of ebit. Note that that the loop might stop before we reach eb. In this case the rest values of the array will be zeros.

9) If the sign of the number y is negative the sbit is 1 and if it is positive sbit is 0.

10) The function should provide three output arguments, namely S=sbit, E=ebit and M=mbit.