FLOATING POINT MULTIPLICATION

A PREPRINT

Sahil Kesharwani
Electronics and Communication Engineering
Indian Institute of Information Technology
Nagpur
Roll No: BT18ECE032

Aniruddha Gawate
Electronics and Communication Engineering
Indian Institute of Information Technology
Nagpur
Roll No: BT18ECE025

ACKNOWLEDGEMENT

We express our deep sense of gratitude to our guide professor **Dr Vipin Kamble**, for his valuable guidance and encouragement. We would also like to express our gratitude to all those who were involved directly or indirectly with the completion of this project.

INDEX

TOPICS Page No

Keywords:- Behavioural style · Structural style · Floating point numbers

INTRODUCTION :

A number representation specifies some way of encoding a number, usually as a string of digits. There are several mechanisms by which strings of digits can represent numbers. In common mathematical notation, the digit string can be of any length, and the location of the radix point is indicated by placing an explicit "point character"(dot or comma) there. If the radix point is not specified, then the string implicitly represents an integer and the unstated radix point would be off the right-hand end of the string, next to the least significant digit. In fixed-point systems, a position in the string is specified for the radix point. In scientific notation, the given number is scaled by a power of 10, so that it lies within a certain range—typically between 1 and 10, with the radix point appearing immediately after the first digit. The scaling factor, as a power of ten, is then indicated separately at the end of the number. Floating-point representation is similar in concept to scientific notation.

Floating Point Representation :

With a fixed-point notation, it is possible to represent a range of positive and negative integers centred on 0. By assuming a fixed binary or radix point, this format allows the representation of numbers with a fractional component as well. This approach has limitations. Very large numbers cannot be represented, nor can very small fractions. Furthermore, the fractional part of the quotient in a division of two large numbers could be lost. For decimal numbers, we get around these limitations by using scientific notation. Thus, 976,000,000,000,000 can be represented as 9.76×10^{14} and 0.00000000000976 can be represented as 9.76×10^{-14}). What we have done, in effect, is dynamically to slide the decimal point to a convenient location and use the exponent of 10 to keep track of that decimal point. This allows a range of very large and very small numbers to be represented with only a few digits. This same approach can be taken with binary numbers 2 as the base. We can represent an 8-bit binary number in the form-

This number can be stored in a binary word with three fields:

- Sign- minus or plus
- Significand S
- Exponent E

Sign of Significand (1 bit)	Biased exponent (3 bit)	Significand (4 bit)

The base B is implicit and need not be stored because it is the same for all numbers. Typically, it is assumed that the radix point is to the right of the leftmost, or most significant, a bit of the significand. That is, there is one bit to the left of the radix point. The principles used in representing binary floating-point

numbers are best explained with an example. Above fig shows an 8-bit floating-point format. The leftmost bit stores the sign of the number (0 = positive, 1 = negative). The exponent value is stored in the next 3 bits. The representation used is known as a biased representation. A fixed value, called the bias, is subtracted from the field to get the true exponent value. Typically, the bias equals ($2^{k-1} - 1$), where k is 3. With a bias of 3($2^2 - 1$), the true exponent values are in the range −2 to +3. The final portion of the word (4 bits in this case) is significant. Any floating-point number can be expressed in many ways. To simplify operations on floating-point numbers, it is typically required that they are normalized. A normalized number is one in which the most significant digit of the significand is nonzero. For base 2 representation, the most significant is '1'. Thus, a normalised number is one in the form

$$\mp 1.bbbbbbbb \quad X \quad 2^{\mp E}$$

So, now the range of the normalised floating-point no is:

• Negative numbers between $-(2 - 2^{\mp 4}) \times 2^3$ and -2^{-2}

• Positive numbers between 2^{-2} and $(2 - 2^{-4}) \times 2^3$

Floating Point Multiplication :

Overview

- Check if operands are zero.
- Add/Subtract exponents and then Subtract/Add bias.
- Check the overflow/underflow.
- Multiply/Divide significands.
- Normalize the result.

FLOATING POINT MULTIPLICATION ALGORITHM :

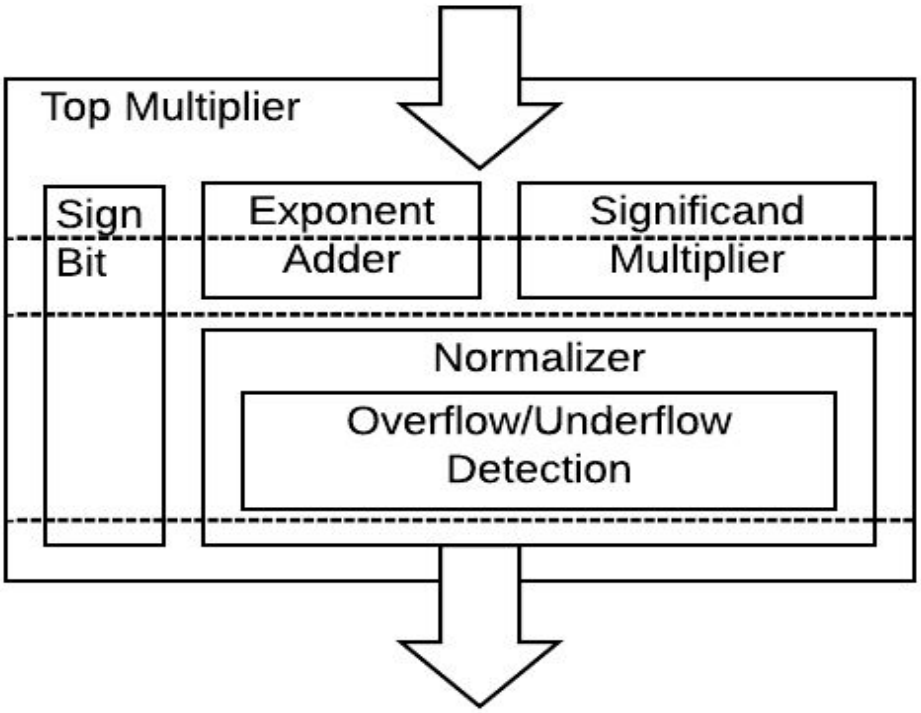
The way IEEE 754 multiplication works is identical to the scientific notation. But simply multiplying the coefficients and adding the exponents won't work as we have to have some extra constraints such as overflow and underflow as the bits limits some application.

So we have to follow extra steps for implementation-

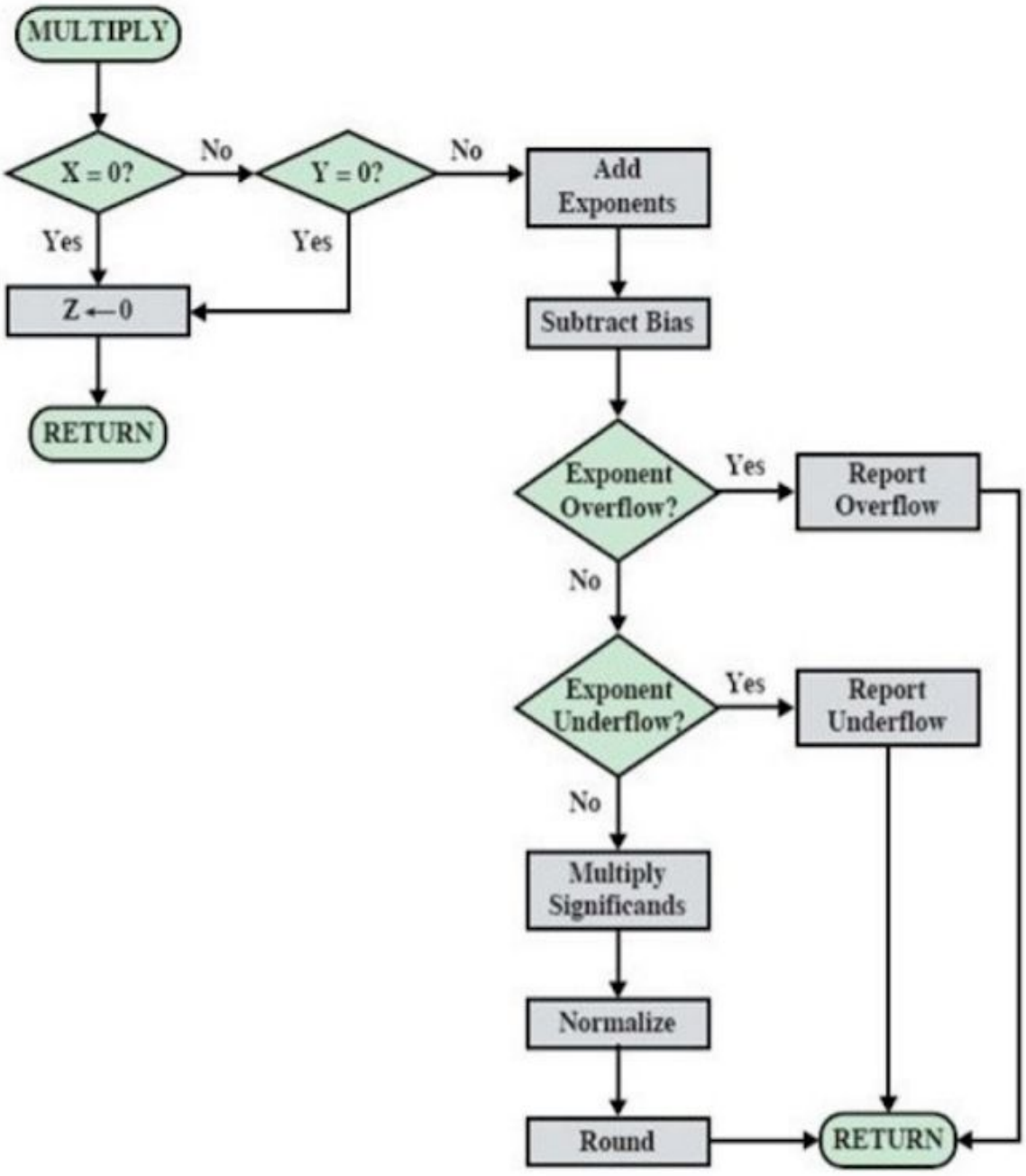
1. Checking if either of operands (A and B) is zero (early out) and also if are infinite value.
2. Checking for potential exponents overflow and throwing success errors.
3. Dividing the 8 bits as a sign, exponent and mantissa for both numbers.
4. Then adding the exponents and subtracting the bais as in this case is 3 for 8bit number.
5. Again checking for potential exponents overflow and throwing success errors.
6. Computing mantissa_z = mantissa_A * mantissa_B using resize function.
7. Normalizing the final mantissa_z and shifting exponentially.

8. Checking for potential exponents overflow and throwing success errors.
9. Now computing sign for the final answer.
10. Assigning the final value to the z.

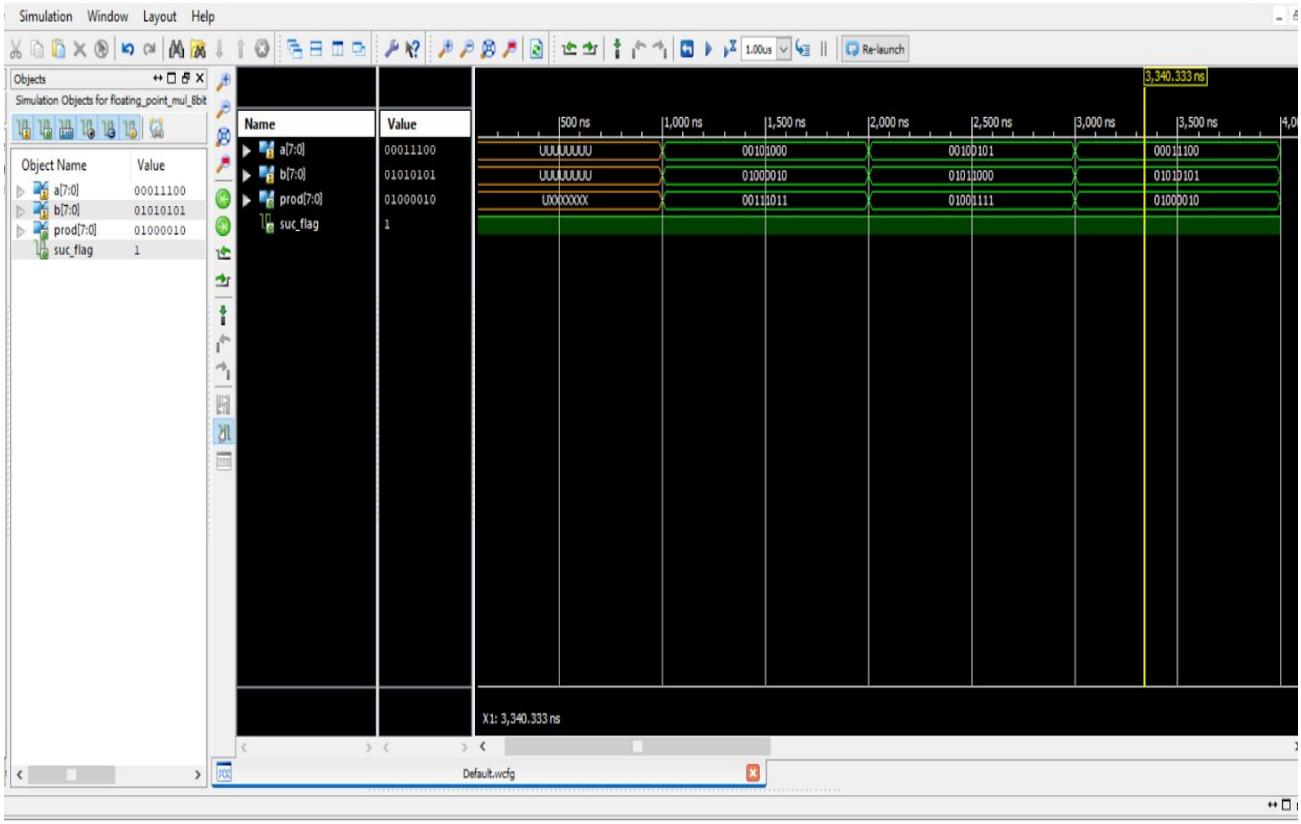
FLOATING POINT MULTIPLIER WITH PIPELINED STAGES :



Flowchart-


MULTIPLY
X = 0?
No
Y = 0?
No
Add Exponents
Yes
Yes
Z ← 0
Subtract Bias
RETURN
Exponent Overflow?
Yes
Report Overflow
No
Exponent Underflow?
Yes
Report Underflow
No
Multiply Significands
Normalize
Round
RETURN

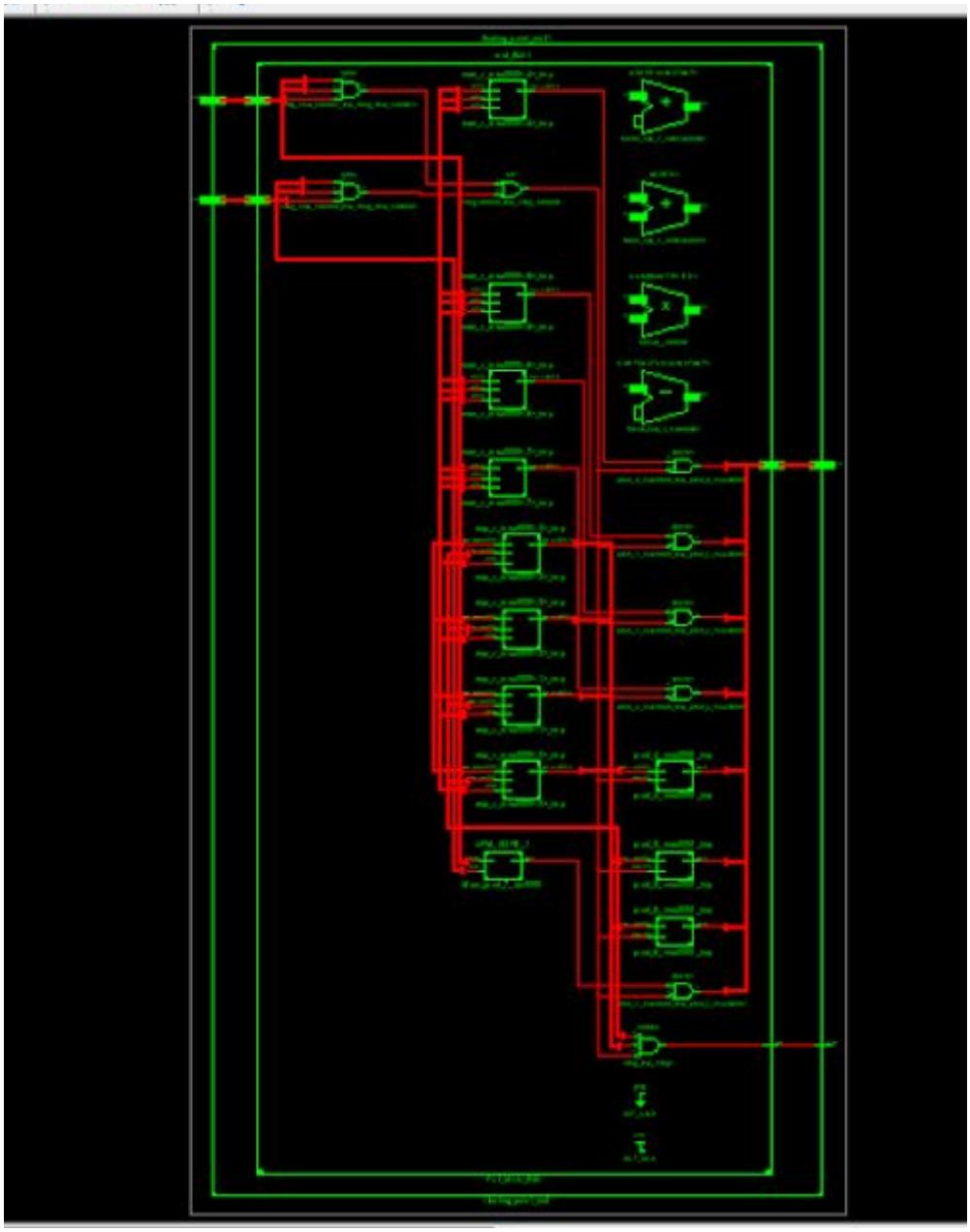
RESULT



Floating Point Representation

Sr.No	First Operand	Second Operand	Product	Op_Flag
1)	00011100	01010101	01000010	1
2)	00100101	01011000	01001111	1
3)	00101000	01000010	00111011	1
4)	01110000	01010000	01110000	0

RTL Schematic-



****All The Codes are attached along the report.**