Bangladesh University of Engineering and Technology

Assignment No.: 02

FLOATING POINT ADDER IMPLEMENTATION

Course Number: CSE306

Course Title: Computer Architecture Sessional

Section: A2

Department: CSE

Group:2

Members: 1905033,1905049,1905053

1905055,1905056

Date of Submission: 08.01.2023

Introduction:

A Floating point adder is a combinatorial circuit which takes 2 floating point numbers as input and as the output, it gives us the sum of the two given numbers. In this assignment, we had to implement a 32-bit floating point adder using combinational logic circuits.

Problem specification:

Each floating point will be 32 bit long with the following representation:

Sign	Exponent	Fraction
1 bit	12 bits	19 bits (lowest bits)

Here, sign bit indicates whether the number is negative or positive with value 1 and 0 respectively. Next, the exponent is 12 bits long and can keep a value between 0 to 4095. To avoid negative value, exponent is biased by 2047. Exponent value 0 and 4095 is reserved for representing denormal numbers, overflow, underflow, NaN (not a number) etc. thus the actual range of exponent in this problem is -2046 to 2047. After biasing by $2047=(2^{11}-1)$, the exponent value will be between 1 and 4094. Finally, we are required to store the fraction in normal form. We can hide the only 1 before decimal point since it is always 1 in normalize form. Now, we store 19 digit of floating point after decimal point. Hence, if S= sign bit, E= exponent, F= fraction, then the actual value of the floating point will be:

$N=(-1)^{s} \times 1.F \times 2^{E-2047}$

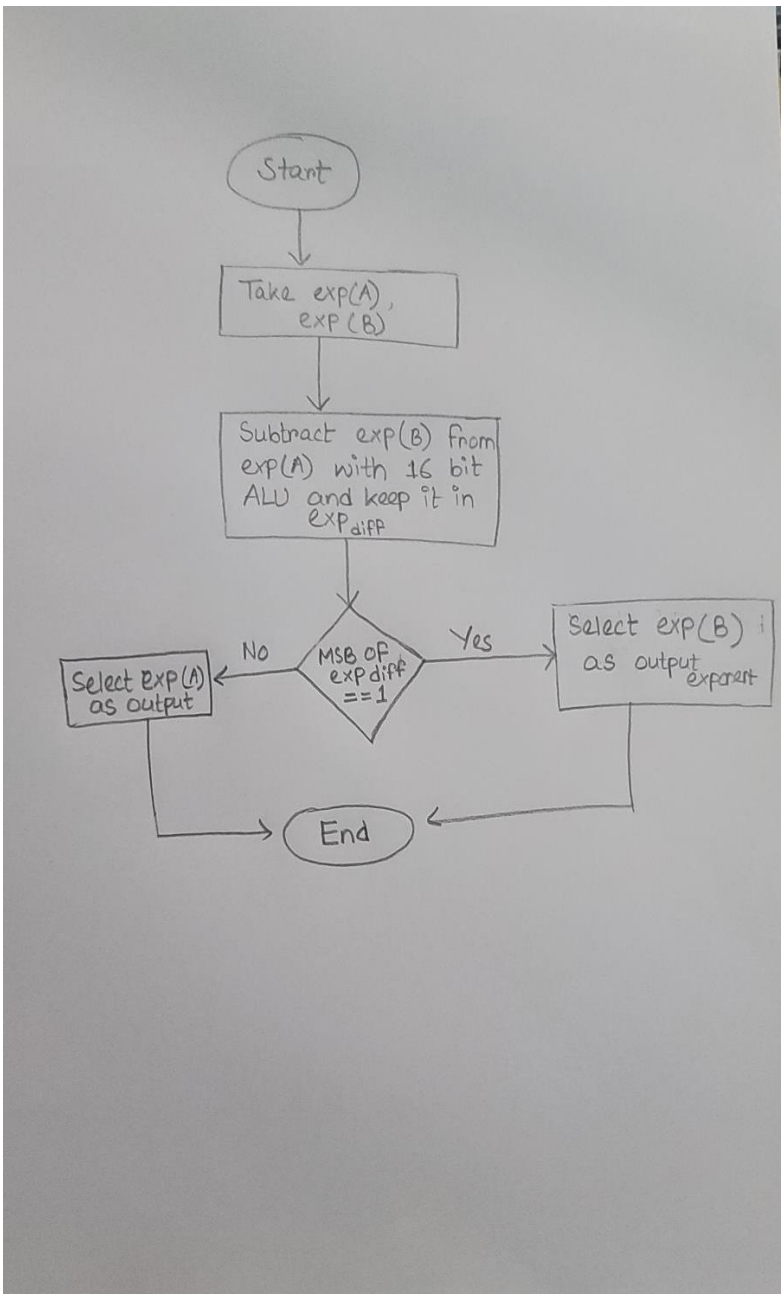
We will be given 2 number in this form and we have to perform addition operation on them and output the resultant in the similar form. In case of overflow or underflow we have to turn on the corresponding flag. When biased exponent becomes less than or equal to 0, underflow occurs and when it becomes greater than 4094, overflow happens.

Procedure:

The work flow of floating point adder can be divided into several subtasks which are mentioned below:

1.Calculating exponent difference and identifying large exponent:

Here, we calculated the absolute difference of exponents, $|E_a-E_b|$ and identified the number which has larger exponent ($E_a \geq E_b$?)



2.Making the exponent Equal and preparing the input fractions:

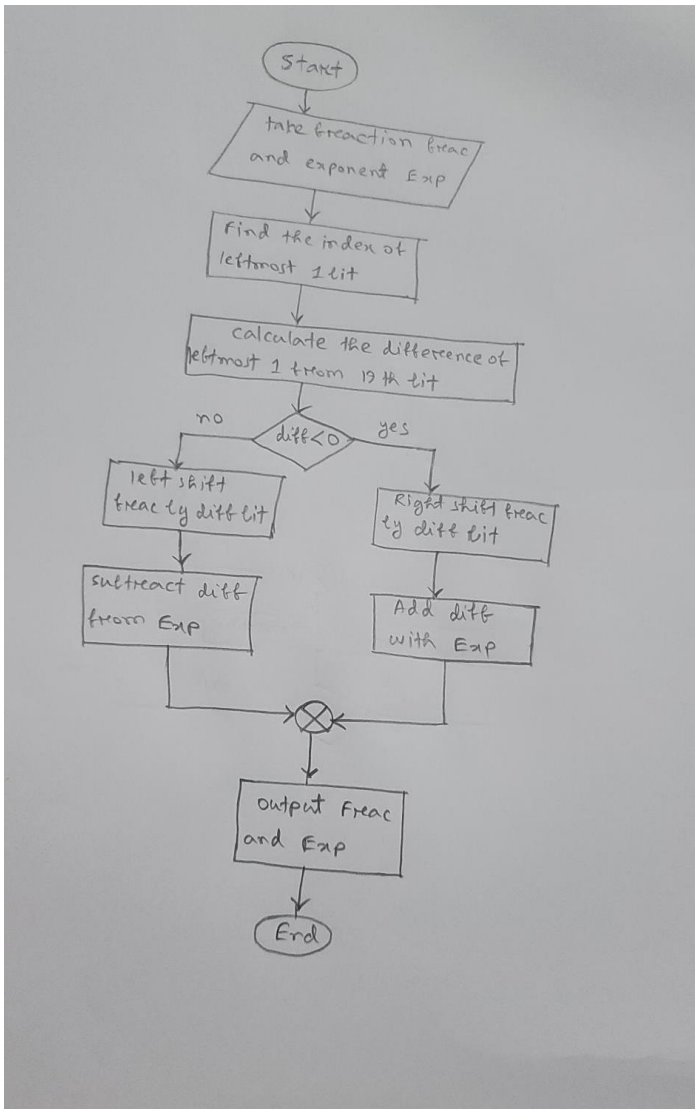
First we add the hidden 1 at the left for both fractions. We shift the floating pointer number which have smaller exponent, to the right by the difference of exponents. Now, after shifting we checked the sign bit of both numbers and apply 2's complement if necessary.

3.Addition of fractions:

We add the processed fractions of the two numbers using 32 bit adder. The result will be a 32-bit fraction and pass it to the mux for normalizing and rounding

4.Normalization:

We need to make the resultant fraction normalized as it may become denormalized after addition. Now we find the location of the first leftmost 1. Our goal is to keep the leftmost 1 in the 19^{th} bit. For this we need to perform shifting either left shift or right shift accordingly. And adjust the exponent accordingly by addition or subtraction of the shifted amount. Now it is ready for rounding.



5. Rounding:

This is the last part where we round the resulting fraction to 20 bit (including the 1 before decimal point). To round a number we have to identify guard bit, round bit, sticky bits. Here the fraction is now 32 bit where the original fraction lies in range bit 3 to 22. The 2^{nd} bit is guard, 1^{st} bit is round and 0^{th} bit is sticky. The table below shows how to round a number:

G	R	S	Action
0	0	0	Truncate
0	0	1	Truncate
0	1	0	Truncate
0	1	1	Truncate
1	0	0	Round to Even
1	0	1	Round Up
1	1	0	Round Up
1	1	1	Round Up

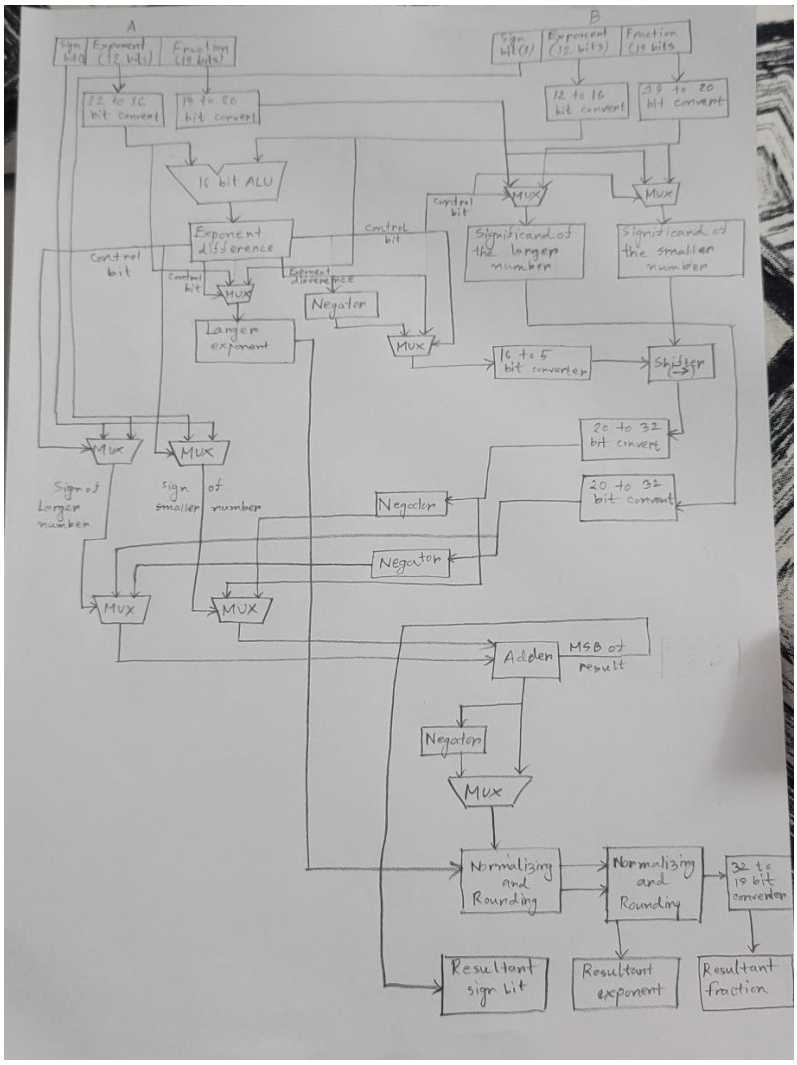
Here,

G=Guard bit

R= Round bit

S= or of all sticky bits

Now, the overall workflow of 32 bit FPA is shown below:



IC count:

Component	Count
Comparator	3
Negator	5
Bit extender	15
Shifter	7
16 bit ALU	1
32 bit adder	3
Subtractor	2
MUX	14
Multiplier	4

Circuit diagrams:

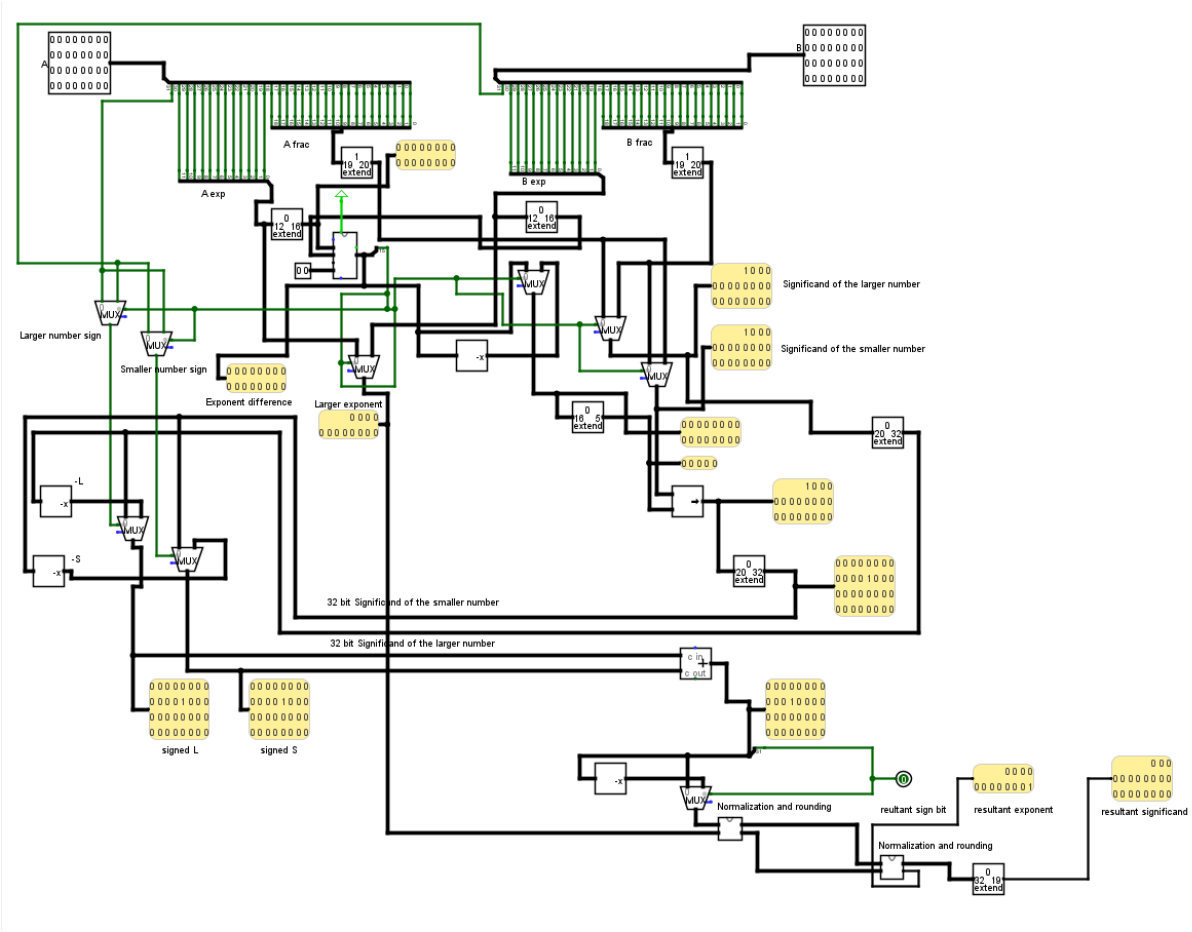


Fig 1: Floating point adder circuit

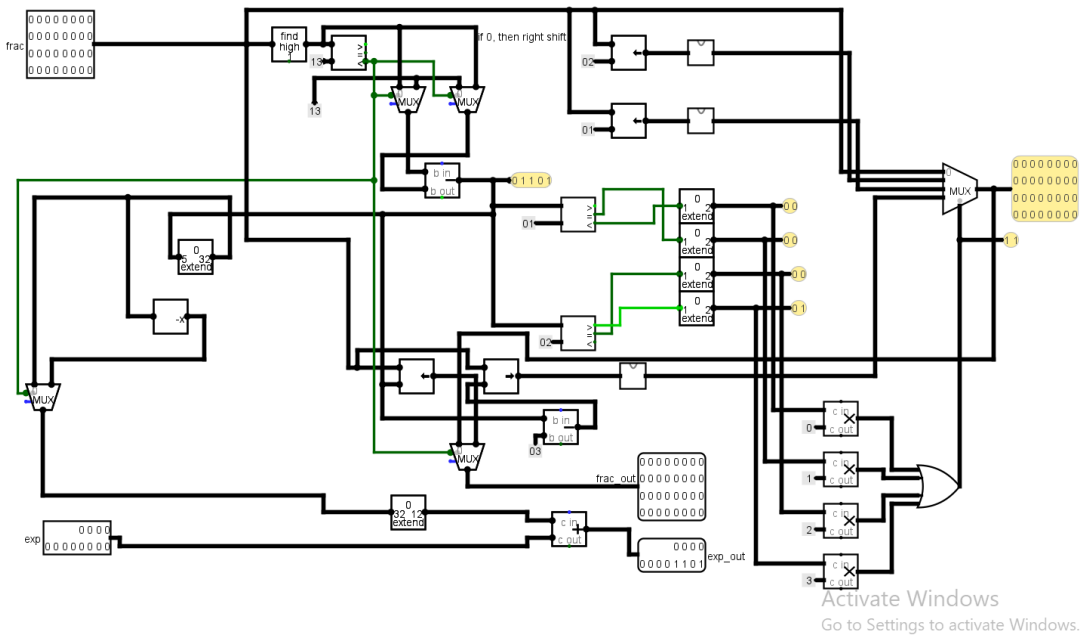


Fig 2: Normalization circuit

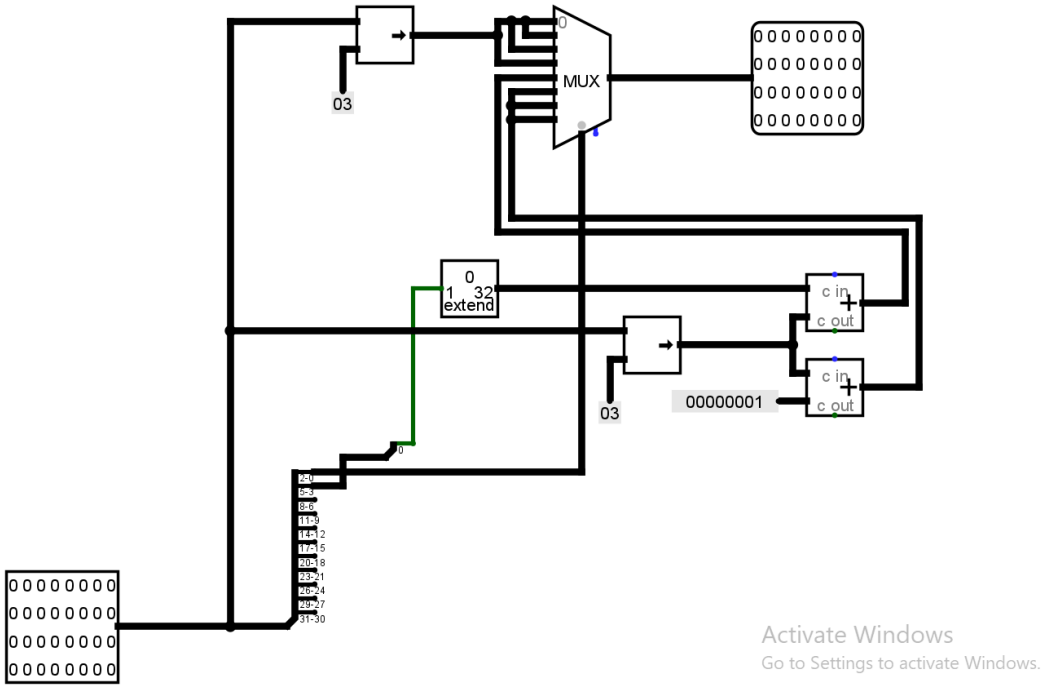


Fig 3: Rounding circuit

Simulator details:

In this assignment, we used Logisim-evolution-2.13.11 which is an open source that allows us to create and simulate digital logic circuits.

Discussion:

In this assignment, a floating point adder which is a subset of IEEE754. Here, we only took normalized numbers as input and we can accurately store a floating point number which lies between $1X2^{-2046}$ and $1.111.....x2^{2047}$ (the absolute value). One point to be noted here is that we only handled 3 bits after the actual fraction for rounding. We could have handled a higher number of bits for sticky bit, but that would complicate our design.