Course Title: Computer Architecture Sessional
Course No: CSE306

Assignment No : 2
Name of the Assignment : Floating Point Adder

Section : B2
Group : 1

Group Members:

Sk. Ruhul Azgor	- *S201805091*
Md. Nazmul Islam Ananto	- *S201805093*
Yeasir Khandaker	- *S201805105*
Sk. Sabit Bin Mosaddek	- *S201805106*
Md. Redwanul Karim	- *S201805111*

Level: 3, Term: 1
Department: CSE

Date of Submission : 23/ 07/ 2022

Introduction:

A Floating Point Adder (FPA) is a combinatorial circuit which takes two 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	10 bits	21 bits (lowest bits)

Here, sign bit indicates whether the number is negative or positive with value 1 and 0 respectively. Next, the exponent is 10 bits long and can keep a value between 0 and 1023. To avoid negative value, exponent is biased by 511. exponent value 0 and 1023 is reserved for representing denormal number, overflow, underflow, no a number etc. Thus the actual range of

exponent in this problem is -510 to 511. After biasing by $511 = (2^9 - 1)$, the exponent value will be between 1 and 1022. Finally, we are required to store the fraction in normal form. We can hide the only 1 before decimal point as it is always 1 in normal form. Now, we store 21 digit of floating point after decimal point. Hence, if S = sign, ~~E = Exopent,~~ E = Exponent, F = Fraction, then the actual value of the floating point will be:

$$N = (-1)^S \times 1.F \times 2^{E-511}$$

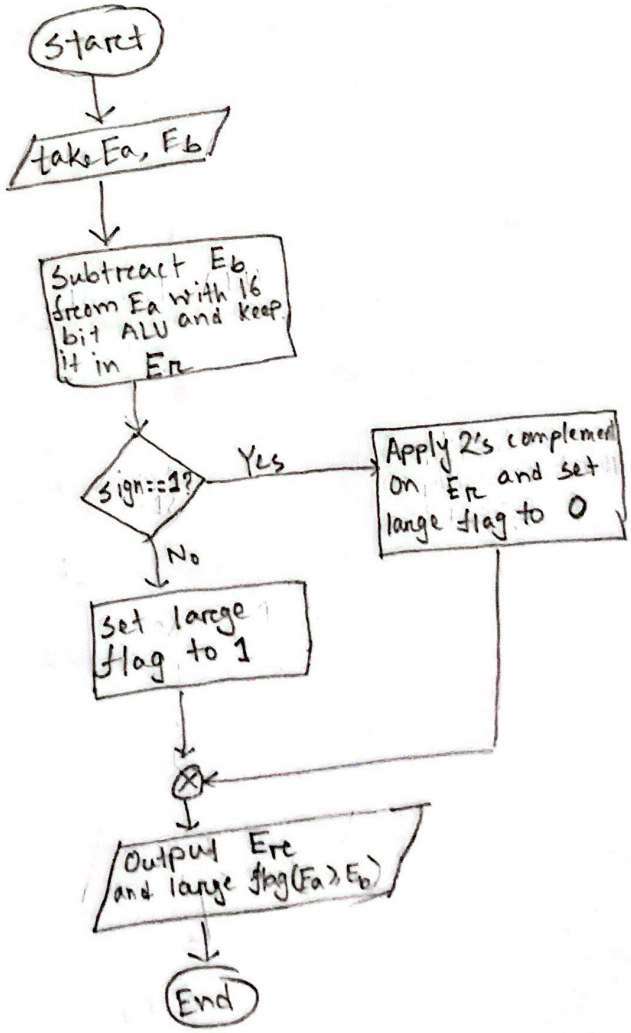
We will be given two number in this form and we have to perform addition operation on them and output the resultant in the similare form. In case of overflow or underflow we have to turn on the corresponding # flag. When biased exponent becomes less than or equal to 0, underflow occurs and when it becomes greater than 1022, overflow happens.

Procedure:

The workflow 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 shift the floating pointer number which have smaller exponent, to the right by the difference of exponents. If the difference is larger than 31 we set the difference to 31 as after shifting the number to right by more than 31. it will become 0. Here, a point to be noted that before shifting the number we made the fraction 32 bit by adding 001 to it's left and 00000000 to it's right. The 1 at the left is our hidden 1 in normalized form and first 0 is sign bit and 1 extra 0 for handling overflow after addition. Last 8 zero's will be important for rounding.

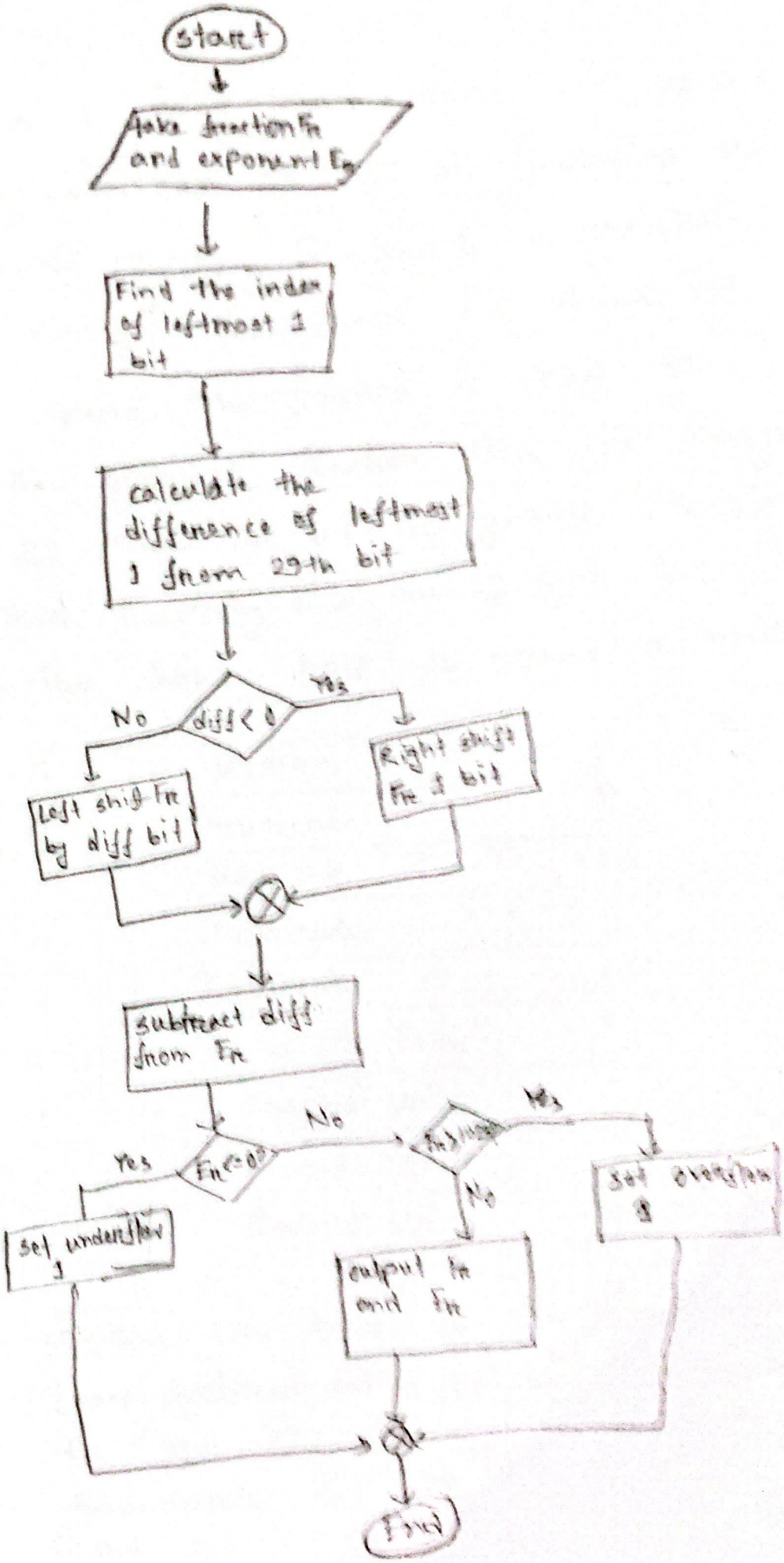
Now, After shifting we checked the sign bit of both number and apply 2's complement if necessary.

3. Addition of fractions:

In this section, we take the furnished fractions of two number and add them using a 32 bit ALU. The result will be a 32 bit fraction with no overflow as we kept one MSB 0 to avoid overflow of any kind.

4. Normalization:

In this part, we make the resultant fraction normal as it may become denormal after addition. Before normalizing, we apply 2's complement if sign bit is on. Now we find the leftmost 1 bit if there is any we try to keep it in 29th bit (0 indexed). We shift the number accordingly and add or subtract from the exponent the shift amount. Now the number is in normal form but we have to check any overflow or underflow. To do so, we check the resulting exponent if it's less than 1, we set the underflow flag 1 or if it's more than 1022, we set overflow 1.


Start
take fraction Fn and exponent En
Find the index of leftmost 1 bit
calculate the difference of leftmost 1 from 23th bit
diff < 0
Yes
No
Right shift Fn diff bit
Left shift Fn by diff bit
subtract diff from En
En<=0?
Yes
No
En>255?
Yes
No
set underflow
set overflow
output fn and En
End

5. ~~Ex~~ Rounding:

Now, this is final part, where we round the resulting fraction to 22 bit (including the 1 before decimal point). To round a number we have to identify ~~Grour~~ Guarid bit, Round bit, sticky bits. Here the fraction is now 32 bit where the original fraction lies in range bit 8 to 29. The 7th bit is guard, 6th bit is round and the sticky bits are (0-5)th bit. 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

Now, let A = ~~whether~~ what we have to add for rounding and L = Least significat bit of original fraction, the 8th bit.

G = ~~Grou~~ Guard bit

R = Round bit

S = Or of all sticky bits

Truth Table:

L	G	R	S	A	Action
0	0	0	0	0	Truncate
0	0	0	1	0	Truncate
0	0	1	0	0	Truncate
0	0	1	1	0	Truncate
0	1	0	0	0	Truncate (to Even)
0	1	0	1	1	Round up
0	1	1	0	1	Round up
0	1	1	1	1	Round up
1	0	0	0	0	Truncate
1	0	0	1	0	Truncate
1	0	1	0	0	Truncate
1	0	1	1	0	Truncate
1	1	0	0	1	Round up (to Even)
1	1	0	1	1	Round up
1	1	1	0	1	Round up
1	1	1	1	1	Round up

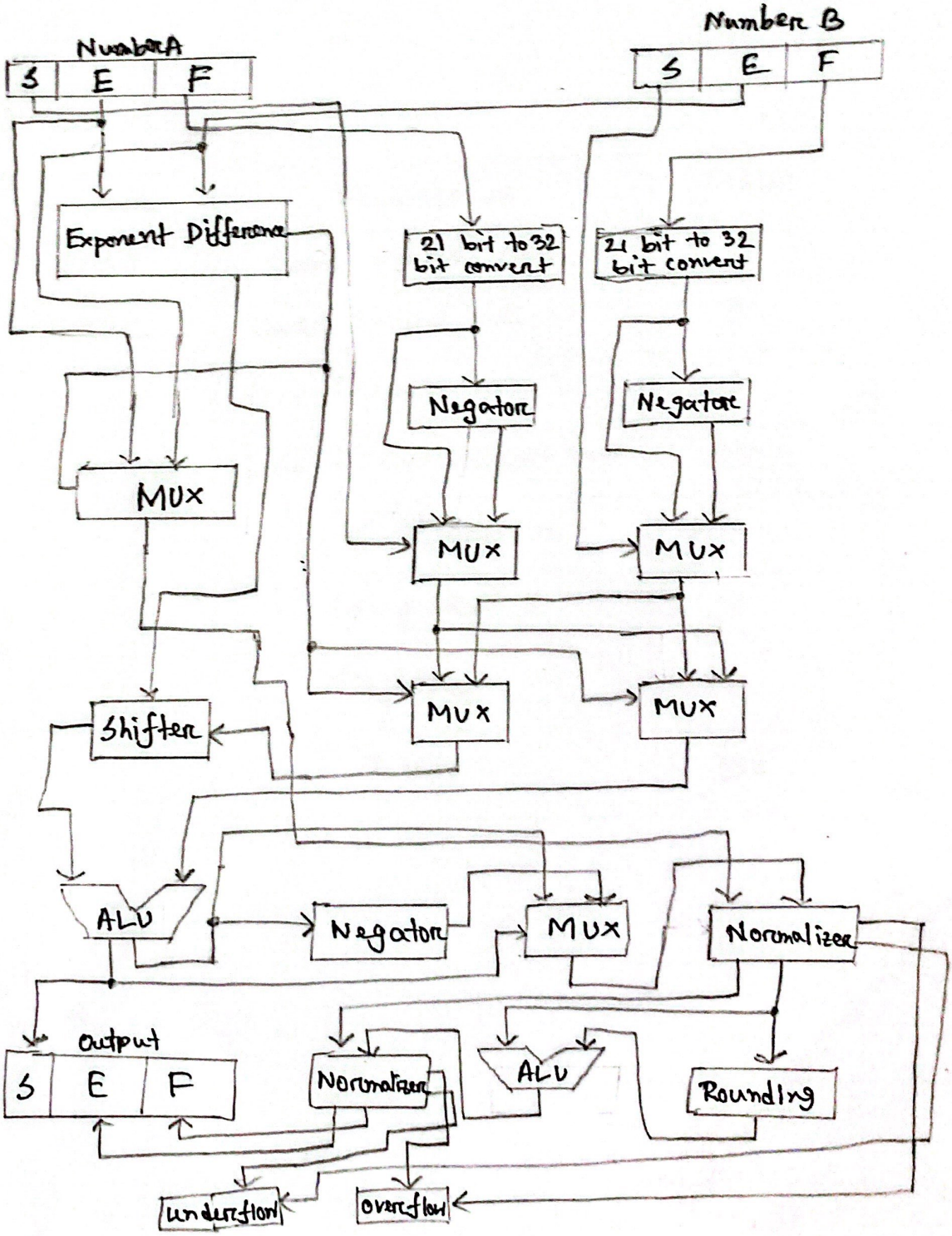
KMap:

LG \ RS	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	1	1	1	1
10	0	0	0	0

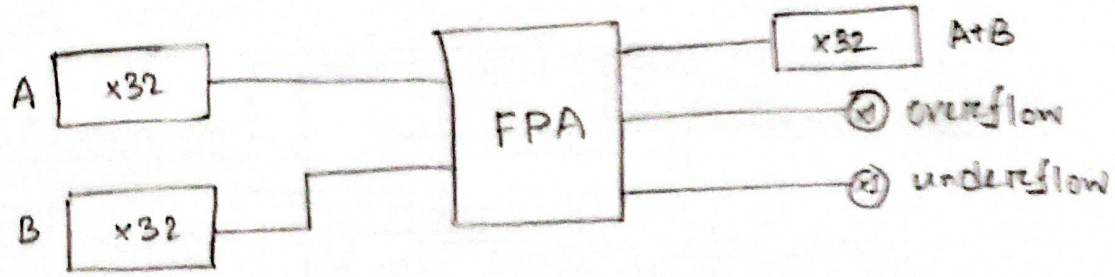
$A = LG + GS + GR$

$= G(L + R + S)$

Now, the overall workflow of 32 bit FPA is shown below: (Calculating Exponent difference and Normalizer module is used to keep the design short and understandable. Workflow of those module is shown in procedure 1 and procedure 4)



Block Diagram:



IC Count:

Number	Description	Count
7408	Quad 2-input AND	1
7432	Quad 2-input OR	5
7486	Quad 2-input XOR	2
74157	Quad 2-Input Multiplexers	89
	Logical shifters	5
	Comparators	7
	Negators	6
	Total	115

Circuit Diagrams:

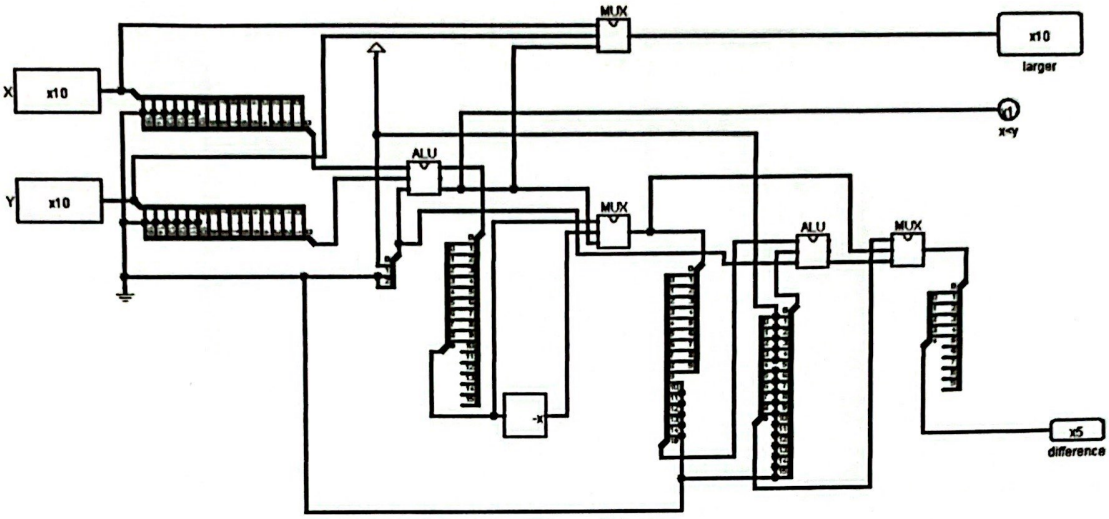


Fig1: Exponent Comparator

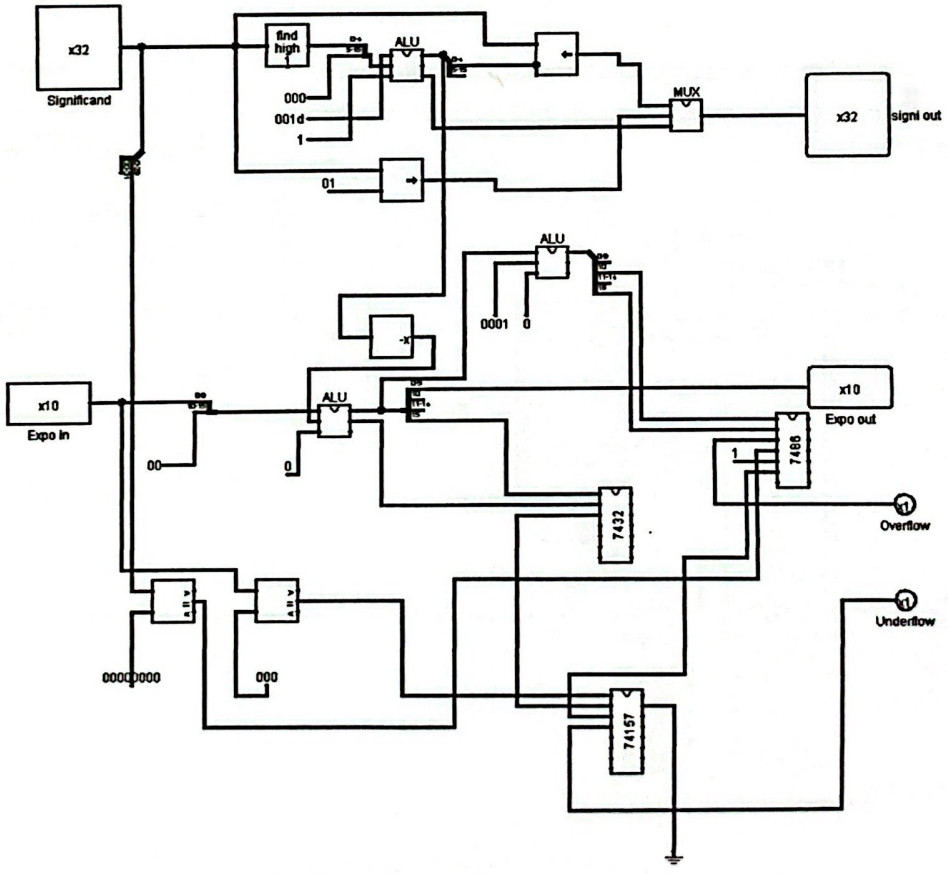


Fig 2: Normalizer

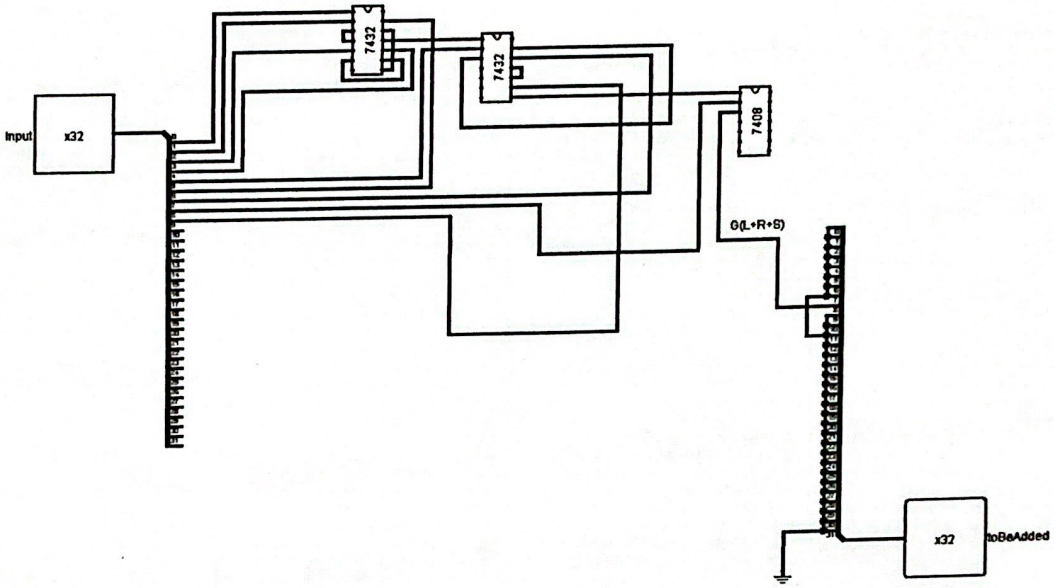


Fig 3: Round to be Added

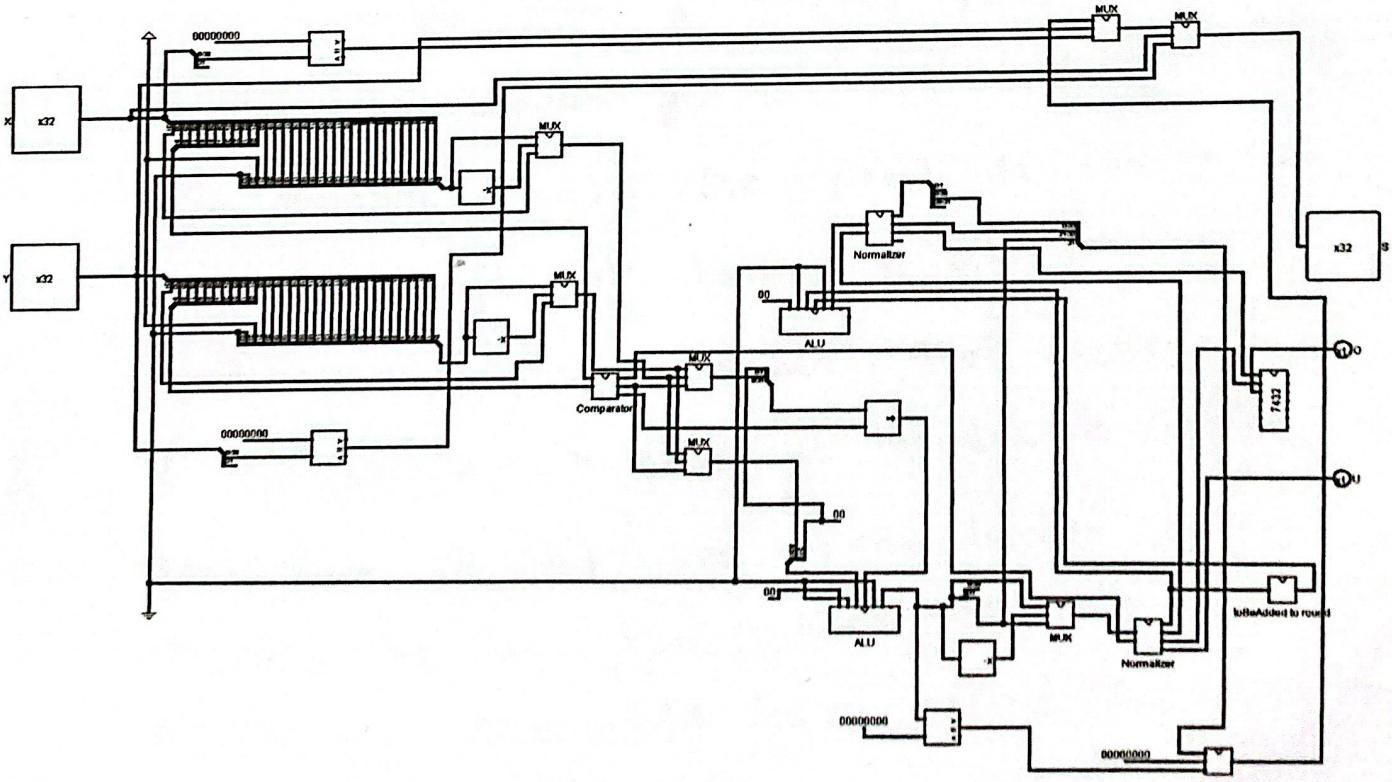


Fig 1: Floating point Adder

Simulator Details:

In this assignment, we used Logisim-2.7.1 which is an open source that allows us to create and simulate digital logic circuits.

Discusion:

In this assignment, a Floating Point Adder which is a subset of IEEE754. Here, we only handled the normalized numbers and we can accurately store a floating point number which lies between $\pm 1 \times 2^{-510}$ and $1.111\ldots11 \times 2^{511}$ (the absolute value). One point to be noted here we could only handle 8 bit precision before rounding the number. That means if the smaller number is needed to shift more than 8 bit, we can lose some precision as we have used 32 bit ALU. A 64 bit ALU could have solved the problem but it would be a lot more complicated to handle such precision.