BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

DHAKA, BANGLADESH

Floating Point Adder

Course No: CSE306

Couse Title: Computer Architecture Sessional

Session: July,2022

Submitted By:

1905042- Rakibul Hasan Rafi

1905046- Niaz Rahman

1905047- Rakib Abdullah

1905048- Md. Al-Amin Sany

1905052- Bijoy Ahmed Saiem

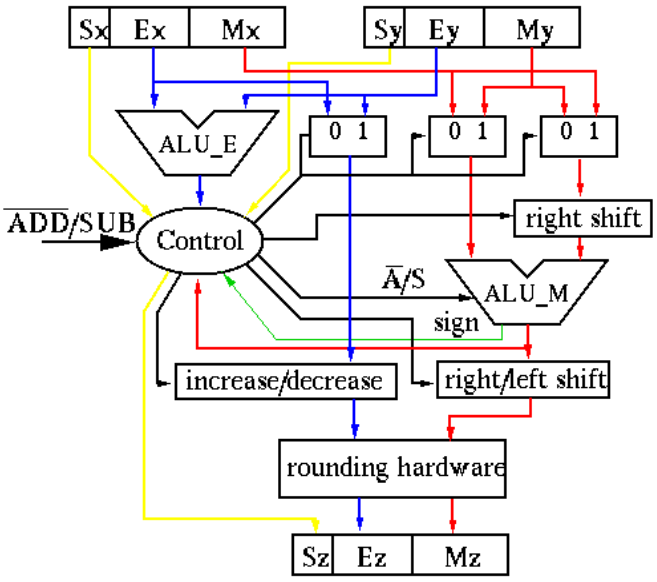
Introduction:

Floating Point is a representation of non-integral numbers which is also known as Fixed Point Binary. These numbers are represented as a single non-zero digit to the left of the floating point. In normalized form, their representation is

$$(-1)^{Sign} * (1+Fraction) * 2^{(Exponent-Bias)}$$

All the floating points are composed by –

- Sign: It indicates whether the number is positive or negative. Sign is assigned '0' for positive and '1' for negative.
- Significant: It sets the value of a number. It is represented as (1+Fraction) where fraction is the value of right to the floating point.
- Exponent: It contains the value of the base power which is biased.
- Bias: It is a value which is added to store exponent and subtracted to restore exponent. If 'k' is the number of bits in the exponent, then bias will be $2^{k-1}-1$
- Base: The total number is used in the number system. It is 2 for binary.



Floating Point Adder is a combinational circuit which takes two floating points as inputs and provides their sum as output. This output is another floating point. It is designed to perform Floating Point Arithmetic which is the most used way of approximating real numbers for performing calculations on modern computers. It requires-

Adder, Subtractor, Shifter, Multiplexer, Comparator.

Problem Specification:

The problem is to design a Floating Point Adder circuit which takes two floating points as inputs and provides their sum as output which is also a floating point.

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

Each floating point is 32 bit long.

Number System: Binary

Base: 2

Bias: 2047

Smallest Value	Largest Value
Exponent: 000000000001 Actual Exponent: 1-2047=-2046 Fraction: 00000000000000000000 Significand: 1.0 Smallest Value: $\pm 1.0*2^{-2046}$	Exponent: 111111111110 Actual Exponent: 4094-2047=2047 Fraction:11111111111111111111 Significand: 2.0(Approx) Largest Value: $\pm 2.0*2^{+2047}$

Relative Precision: approx 2^{-19} ,equivalent to $19* \log_{10}2$=5 digits of precision

Assumption:

Exponents 000000000000 and 111111111111 are reserved

For the simplicity we are neglecting special cases like if either input is equal to

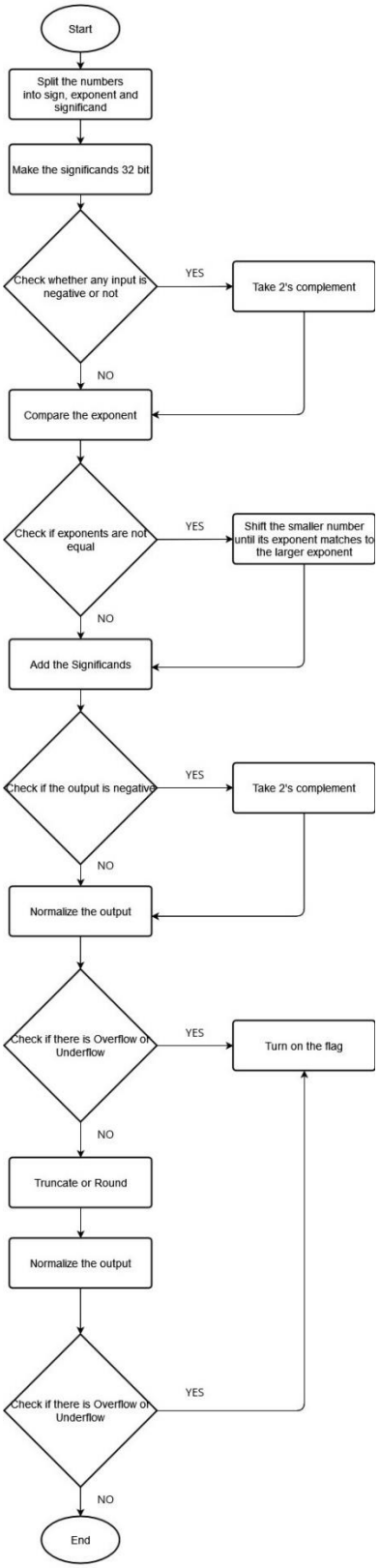
- Zero
- NaN(Not a Number): Exponent=111111111111, Fraction≠00000000000000000000
- ±Infinity: Exponent=111111111111, Fraction=00000000000000000000

For simplicity we are assuming the inputs are already in normalized form.

Normal Numbers: $(-1)^{Sign} * (1+\text{Fraction}) * 2^{(Exponent-Bias)}$

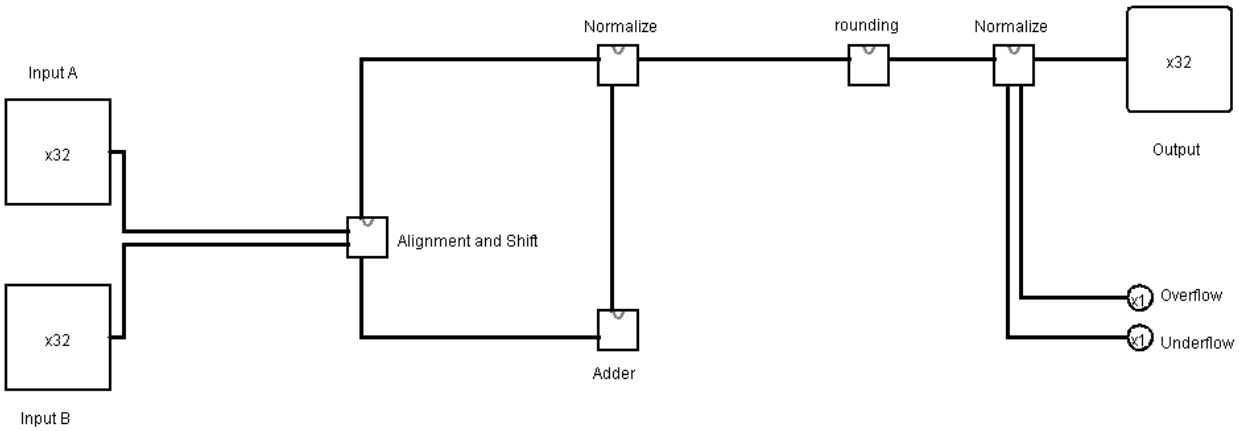
Denormal Numbers: $(-1)^{Sign} * (0+\text{Fraction}) * 2^{(Exponent-Bias)}$

Flowchart:

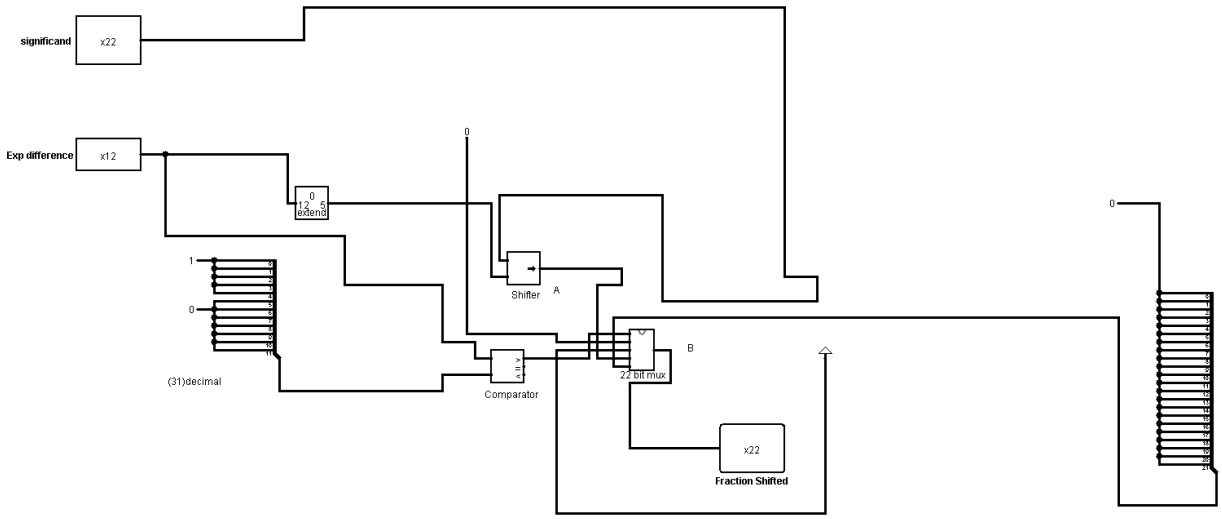

Start
Split the numbers into sign, exponent and significand
Make the significands 32 bit
Check whether any input is negative or not
YES
Take 2's complement
NO
Compare the exponent
Check if exponents are not equal
YES
Shift the smaller number until its exponent matches to the larger exponent
NO
Add the Significands
Check if the output is negative
YES
Take 2's complement
NO
Normalize the output
Check if there is Overflow or Underflow
YES
Turn on the flag
NO
Truncate or Round
Normalize the output
Check if there is Overflow or Underflow
YES
NO
End

Design Steps:

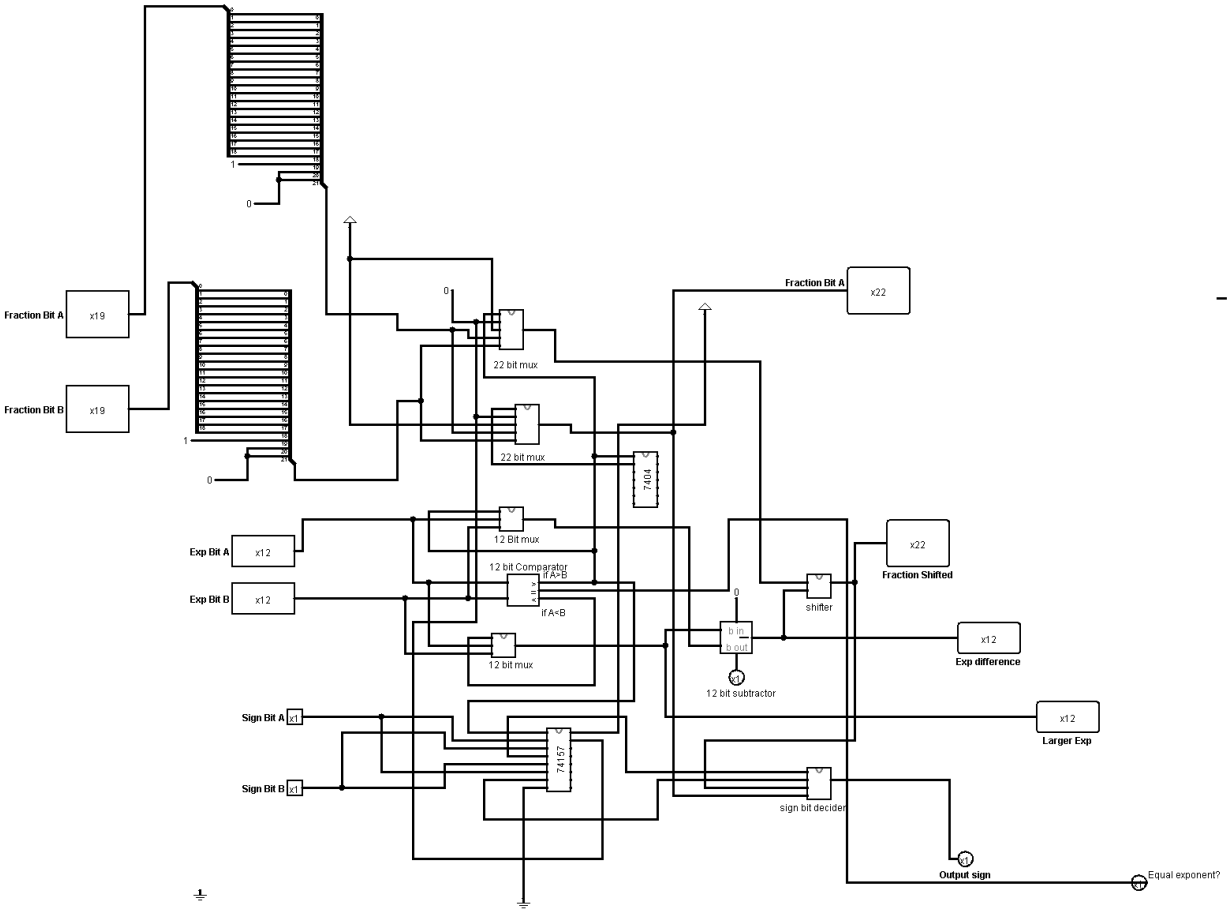
Here, we will discuss the design steps along with the circuits. We have constructed sub-circuit. Using the sub-circuit, we construct the main floating point circuit. We have constructed –



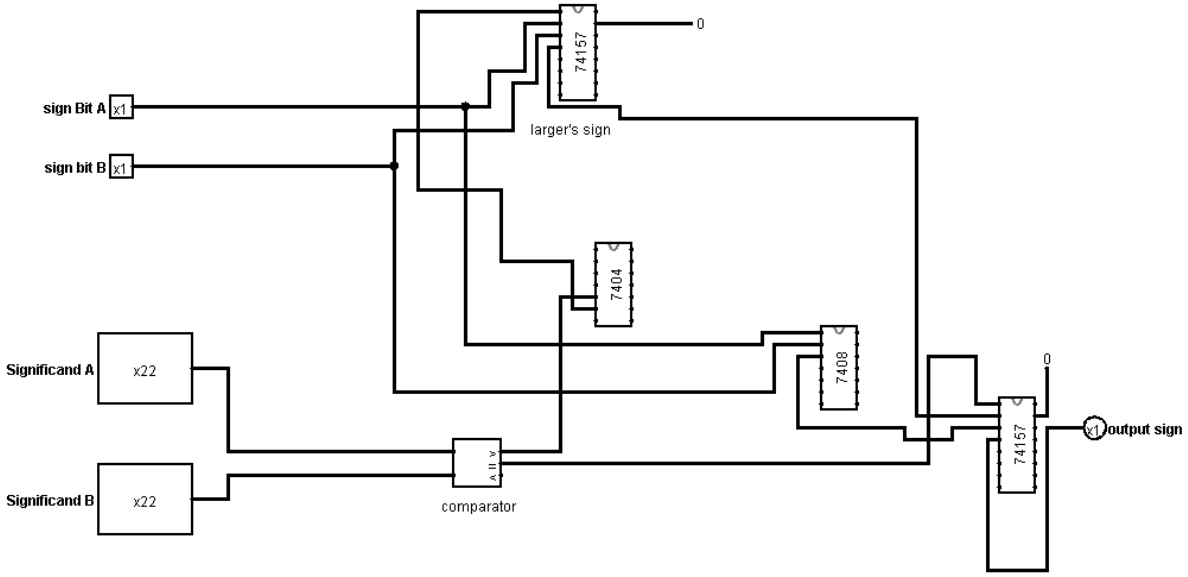
1. **Alignment and Shifting:** First, the inputs are split. Then, the exponents are compared. Shift the smaller number until its exponent matches the large exponent. In this process, the fraction is also aligned. **Circuit diagram of shifter circuit:**



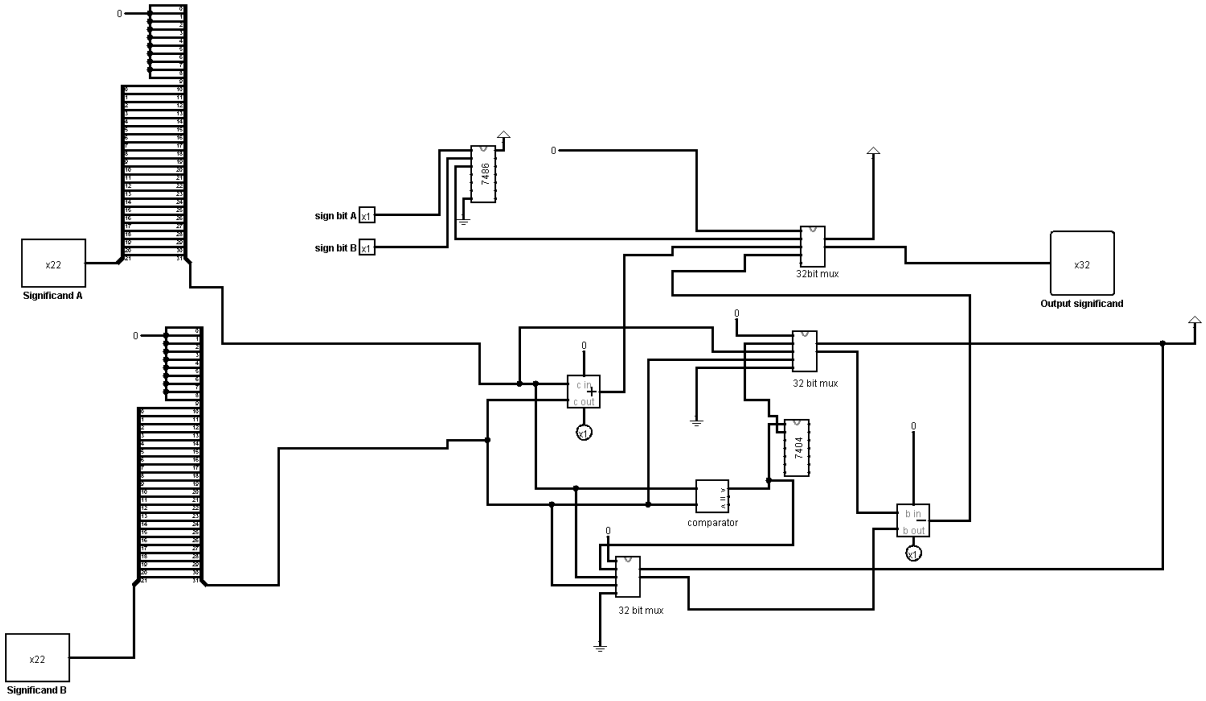
Circuit diagram of alignment circuit:



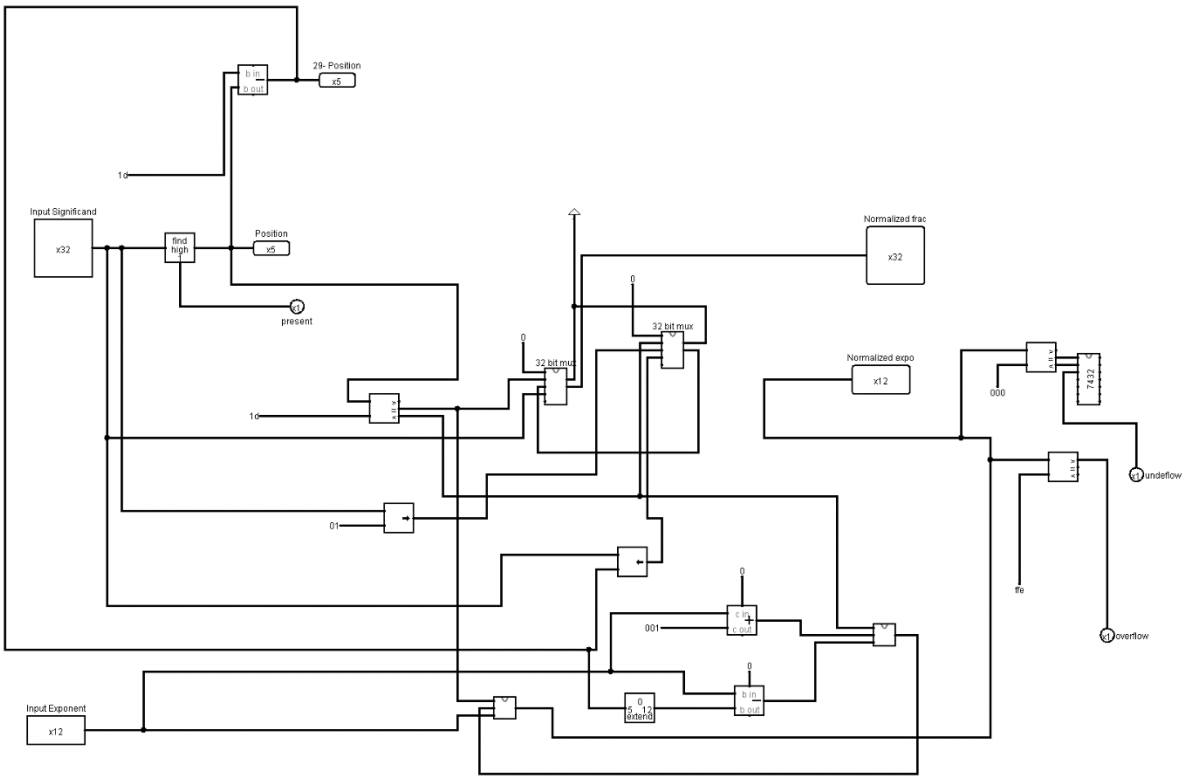
2. **Sign Bit Detection:** Here, the input is checked whether it is positive or negative. If the number is negative, 2's complement is taken. **Circuit diagram of sign bit detection:**



3. **Adder:** After sign bit detection, the aligned and shifted input is added. **Circuit diagram of adder circuit:**



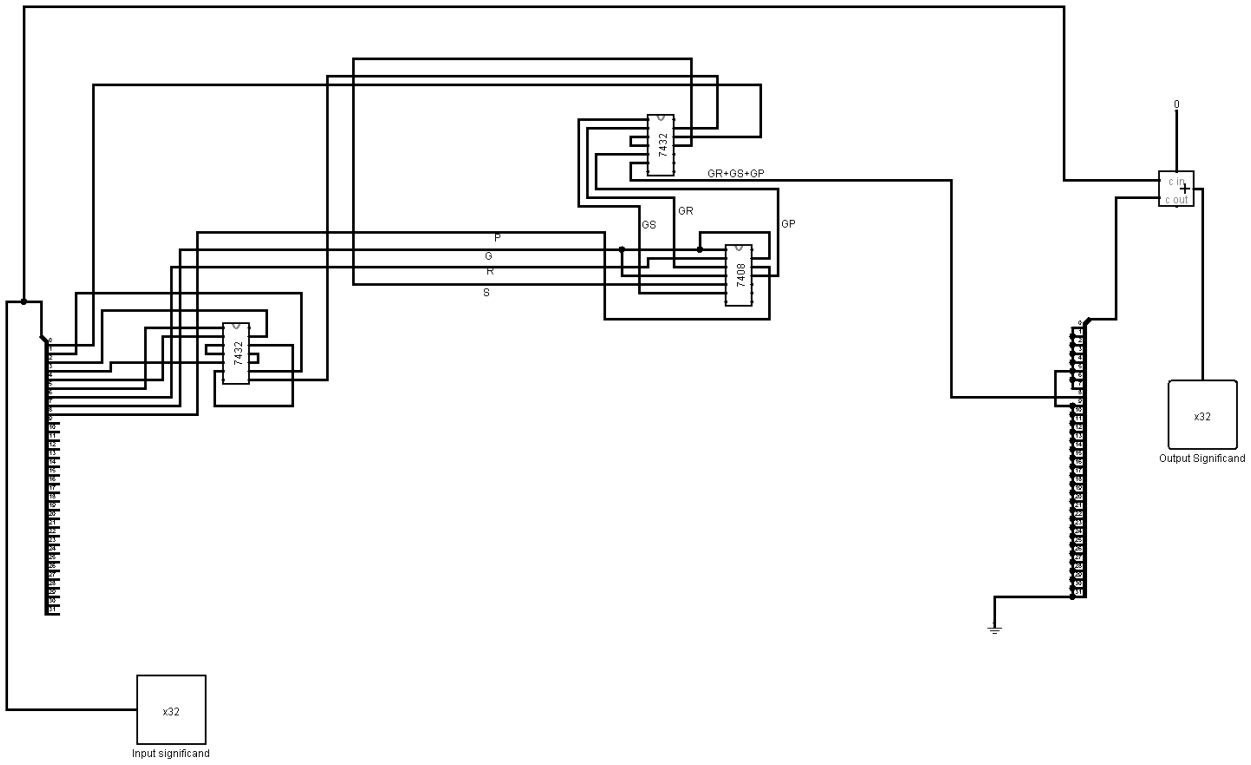
4. **Normalization:** While adding two inputs, the output may be in denormal form. So, the output is normalized. **Circuit diagram of normalization circuit:**



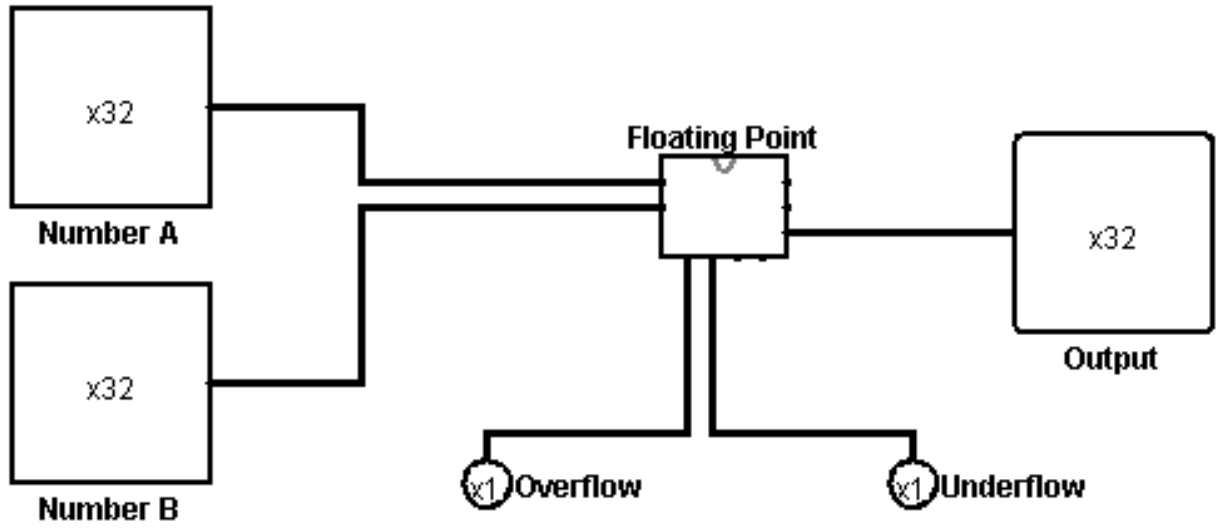
5. **Rounding:** If there are several extra digits, the output is rounded for better accuracy and computation. There are three digits which are called guard digit, round digit, sticky digit.

Guard	Round	Sticky	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

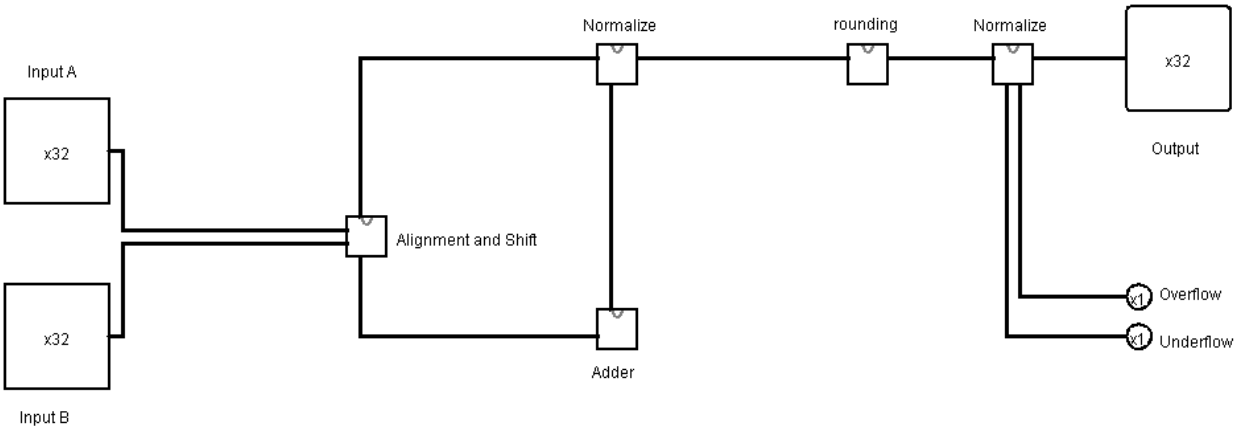
Circuit diagram of rounding circuit:



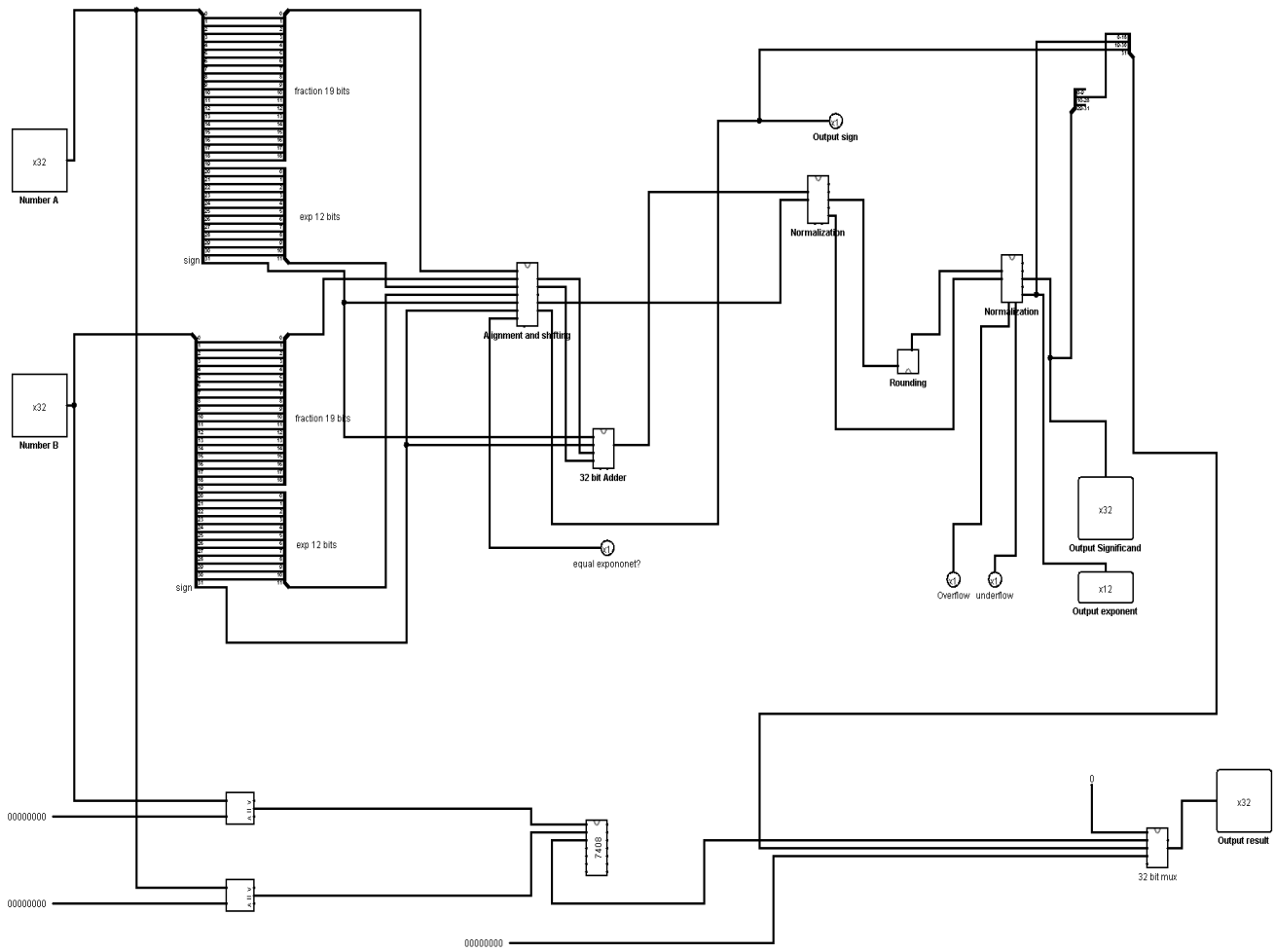
Block Diagram:



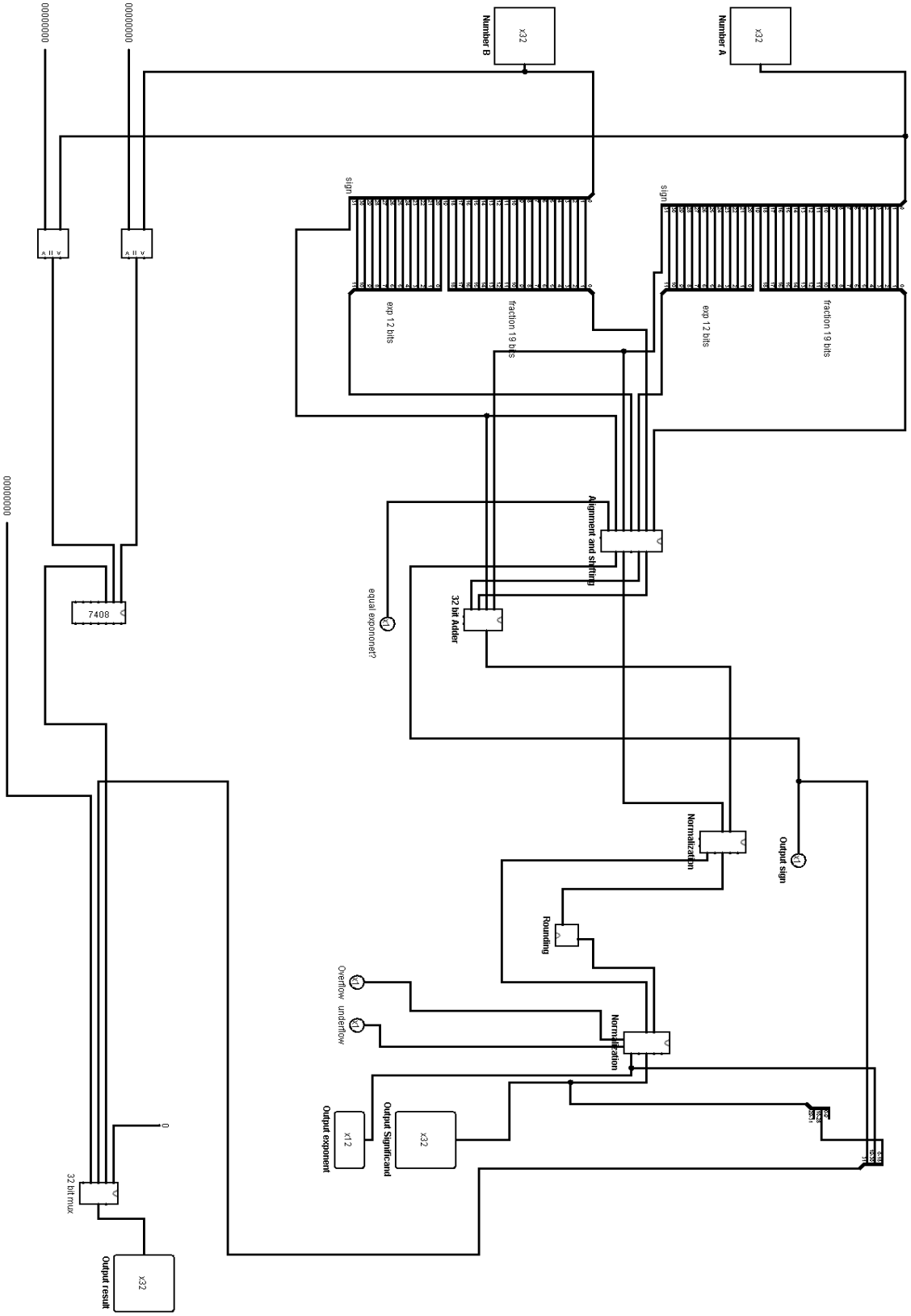
We have also mentioned an outline showing the steps of a Floating Point Adder.



Circuit Diagram:



Circuit Diagram(Landscape View):



ICs and Components:

Component Count:

Component	Count
12bit MUX	2
22bit MUX	3
32bit MUX	6
Adder	2
Bit Extender	2
Bit Finder	1
Comparator	8
Subtractor	3
Shifter	3

IC Count:

IC Name	Action	Count
IC7404	Logical NOT	3
IC7408	Logical AND	3
IC7432	Logical OR	3
IC7486	Logical X-OR	1
IC75157	4x1 MUX	3

Simulator Version:

Logisim V2.7.1

Discussion:

Though it is a software implementation, we faced a lot of difficulties while designing as it is a large circuit with different components and ICs. As a result, we used sub-circuit to make the main implementation which have been discussed in design steps. We have to check corner cases, overflow and underflow condition and have to crosscheck manually. Manually crosschecking is one of the most time-consuming works. However, we also faced problems in rounding and truncating the output. But in most general cases, we always get right answer.

We also have some simulator problem. We got some error like “Simulation halted by an internal error” though we have implemented our circuit correctly. Sometimes, it showing error without any reason. For solving this problem, we have to close and reopen the software again. Many times, it becomes frustrating.

We do not handle the special cases like NaN and infinity as input. We assumed the input as a normal number. So, taking input as a denormal number will give wrong output. That is why, it is expected that, we will treat input as normal and will get output as normal also.

We have detected overflow and underflow also. We have a range limitation. If the output is smaller than lower range then underflow flag will be turned on. Again, if output is higher than upper range then overflow flag will be turned on. That is how we have handled the situation.

We try to find the most optimized implementation. This may not be the best but it works with perfection. We try our best to implement the circuit.