Bangladesh University of Engineering and Technology

# CSE-306

## Assignment-2

## A 32 bits Floating Point Adder Design

**Lab Group**: B1

**Sub-group**: 1

**Date**: January 8, 2023


**Group Members:**

Nazmus Sakib (1905061)

Sayem Shahad (1905064)

Mahir Labib Dihan (1905072)

Souvik Ghosh (1905073)

Salman Sayeed (1905079)

## Introduction

Floating-Point Adder is a combinational circuit which takes two floating points as inputs and provides their sum which is another floating point as output. Its implementation requires some basic n bits adder, subtractor, shifter, multiplexer, comparator, and some other basic gates.

Floating-Point Adder is designed to perform "Floating Point arithmetic" which is by far the most used way of approximating real number arithmetic for performing numerical calculations on modern computers. The floating-point numbers representation is established on the following scientific notation:

**the decimal point is not set in a fixed position in the bit sequence, but its position is indicated as a base power.**

All the floating-point numbers are composed by four components:

1. Sign: it indicates the sign of the number (0 positive and 1 negative)

2. Significant: it sets the value of the number

3. Exponent: it contains the value of the base power (biased)

4. Base: the base (or radix) is implied and it is common to all the numbers

(2 for binary numbers)

The steps involved in the design of a Floating-Point Adder are as follows:

1. Extracting signs, exponents and fractions of both A and B numbers.

2. Treating the special cases:

    - Operations with A or B equal to zero
    - Operations with $\pm\infty$
    - Operations with NaN

3. Finding out what type of numbers are given:

    - Normalized
    - Unnormalized

(For simplicity of our design, we are assuming that numbers are given in

normalized form)

4. By comparing the exponent of the numbers, finding out their difference which is actually the required shifting amount and also the smaller & larger number.

5. Shifting the lower exponent number fraction to the right [Exp1 - Exp2] bits. Setting the output exponent as the highest exponent.

6. Working with the operation symbol and both signs to calculate the output sign and determine the operation to do.

7. Addition of the numbers and detection of overflow (carry bit)

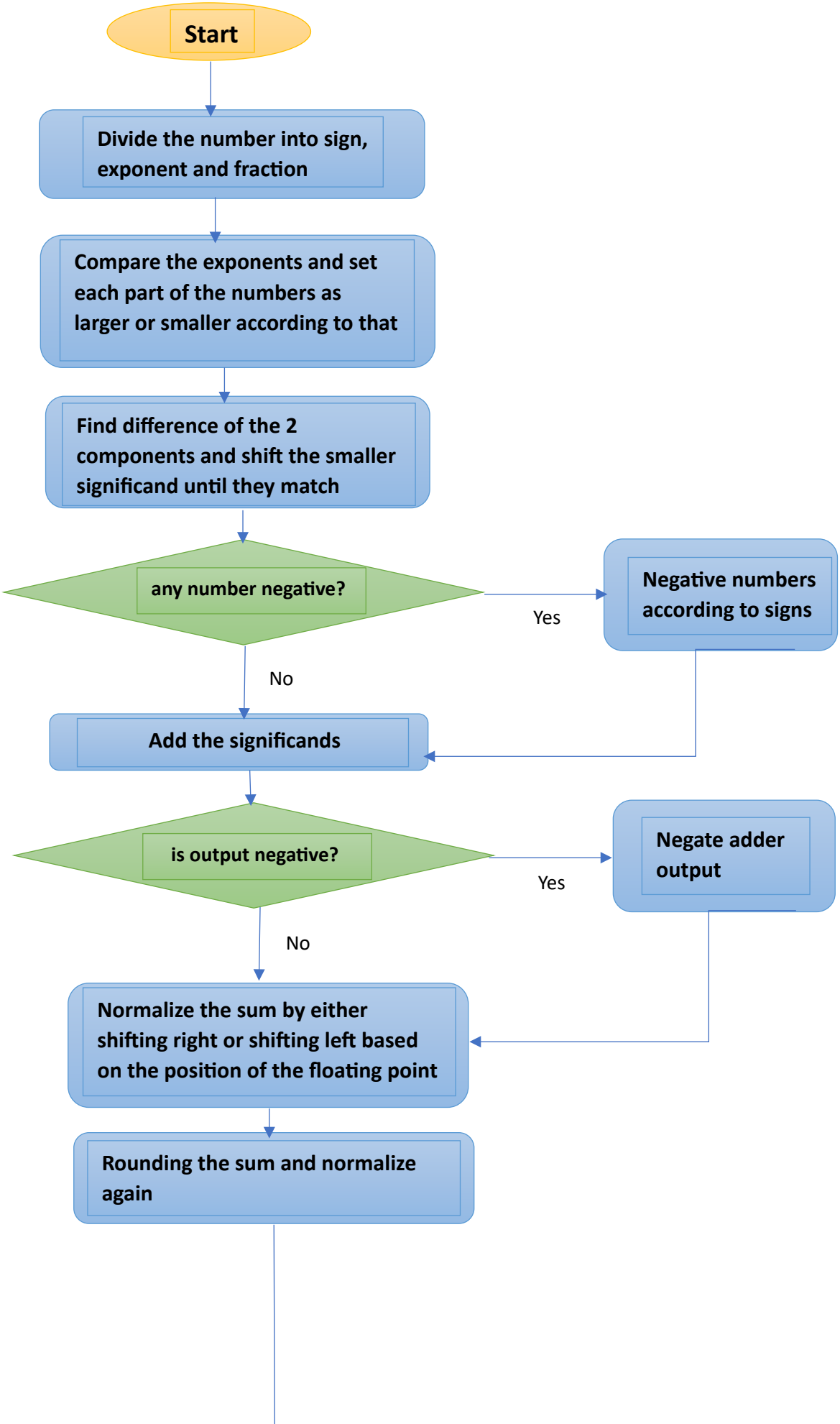
8. Standardizing fraction shifting it to the left up the first one will be at the first position and updating the value of the exponent according with the carry bit and the shifting over the fraction.

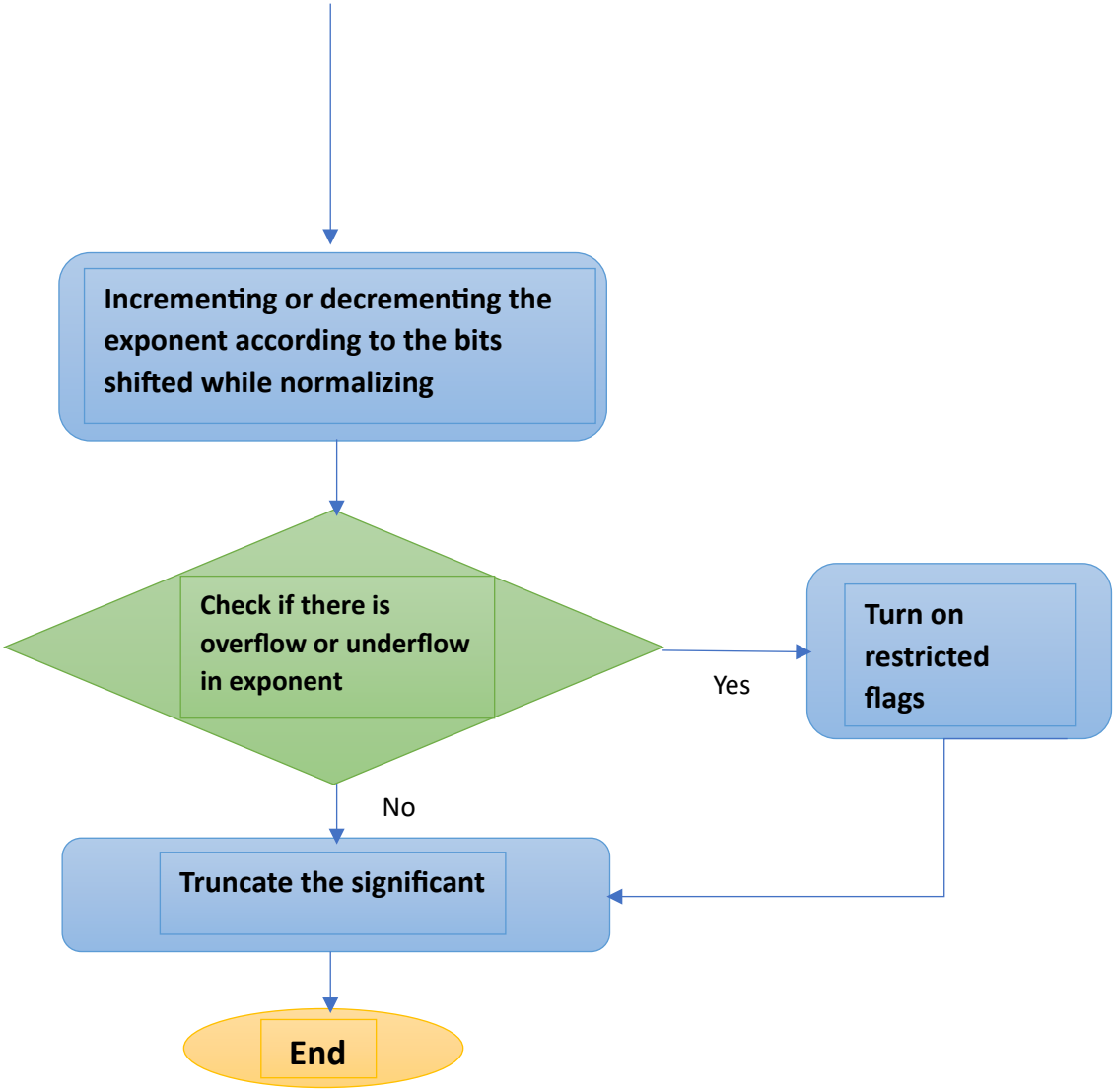
## Problem Specification:

In this assignment, we are required to design a floating point adder circuit that takes two floating points as inputs and provides their sum, another floating point as output. Each floating point will be 32 bits long with the following representation:

| Sign | Exponent | Fraction |
|---|---|---|
| 1 bit | 12 bits | 19 bits (Lowest bits) |

## Flowchart:


Start
Divide the number into sign, exponent and fraction
Compare the exponents and set each part of the numbers as larger or smaller according to that
Find difference of the 2 components and shift the smaller significand until they match
any number negative?
Yes
Negative numbers according to signs
No
Add the significands
is output negative?
Yes
Negate adder output
No
Normalize the sum by either shifting right or shifting left based on the position of the floating point
Rounding the sum and normalize again



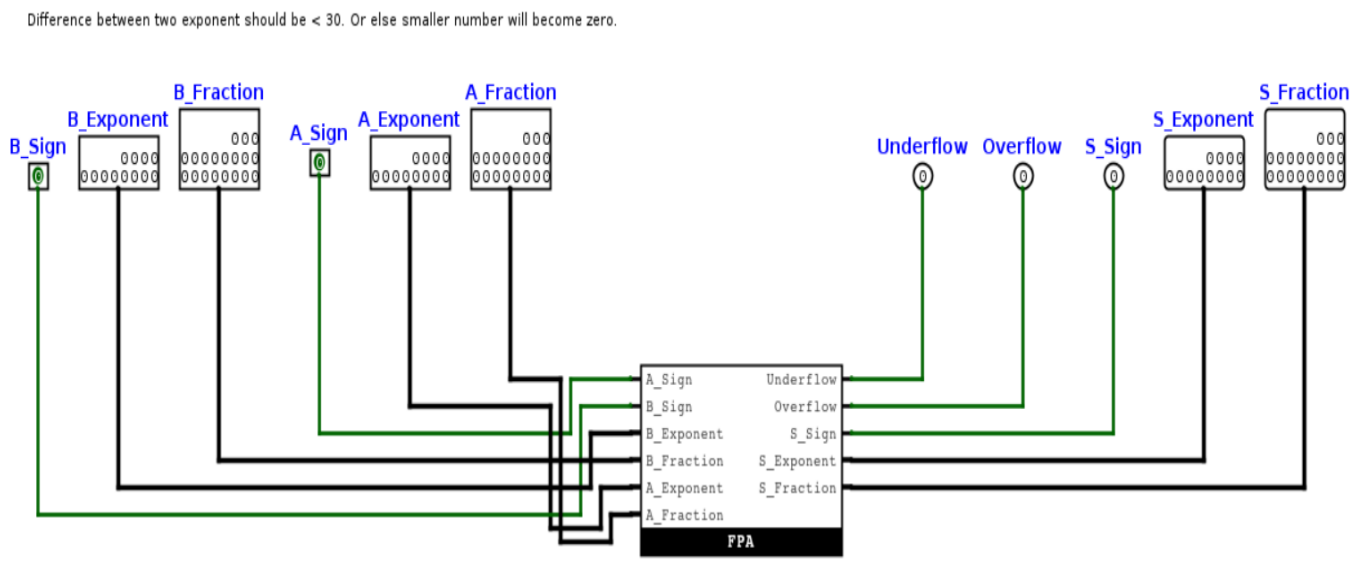

## Equation for Rounding:

| LSB | To add | G | R | S | Action |
|---|---|---|---|---|---|
| X | 0 | 0 | 0 | 0 | Truncate |
| X | 0 | 0 | 0 | 1 | Truncate |
| X | 0 | 0 | 1 | 0 | Truncate |
| X | 0 | 0 | 1 | 1 | Truncate |
| 0 | 0 | 1 | 0 | 0 | Round to even |
| 1 | 1 | | | | |
| X | 1 | 1 | 0 | 1 | Round up |
| X | 1 | 1 | 1 | 0 | Round up |
| X | 1 | 1 | 1 | 1 | Round up |

Equation for Rounding

G.(LSB+R+S)

# High-Level Block Diagram:




# Circuit Diagram:

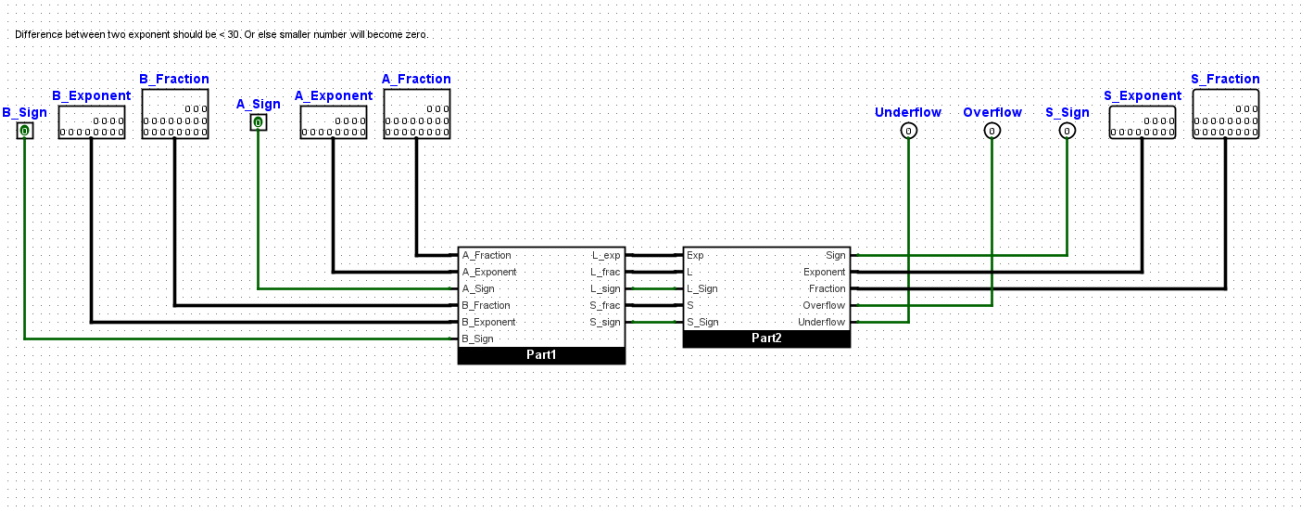



**Figure-0:** Floating point adder

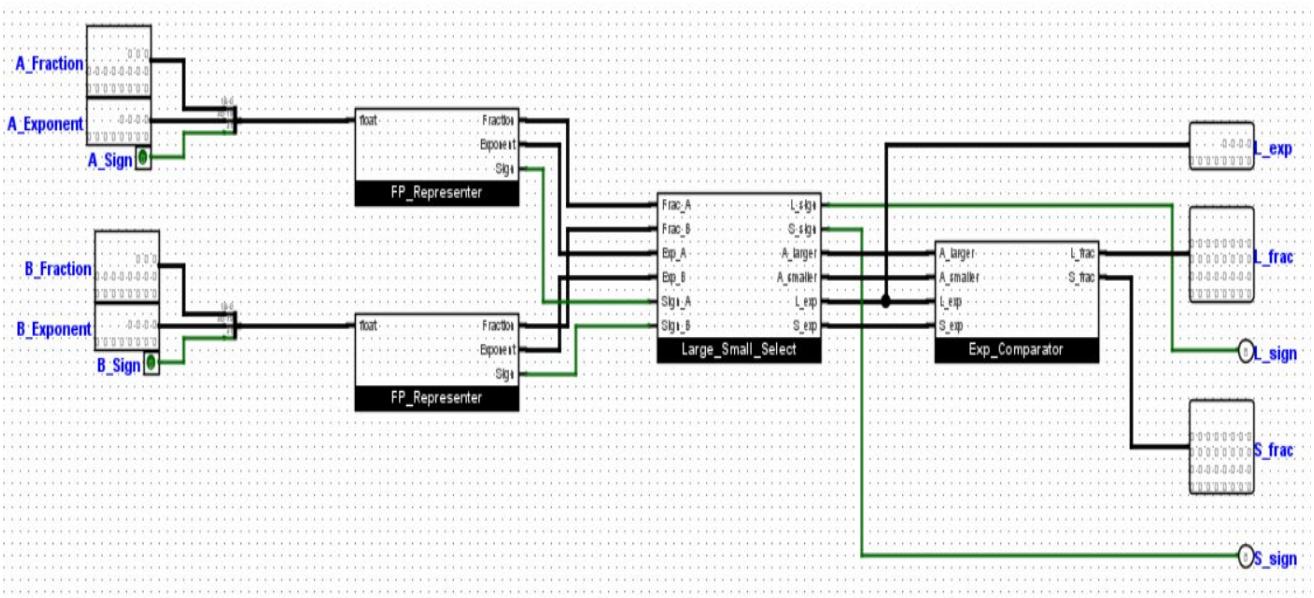



**Figure 1:** Floating point adder – Part1

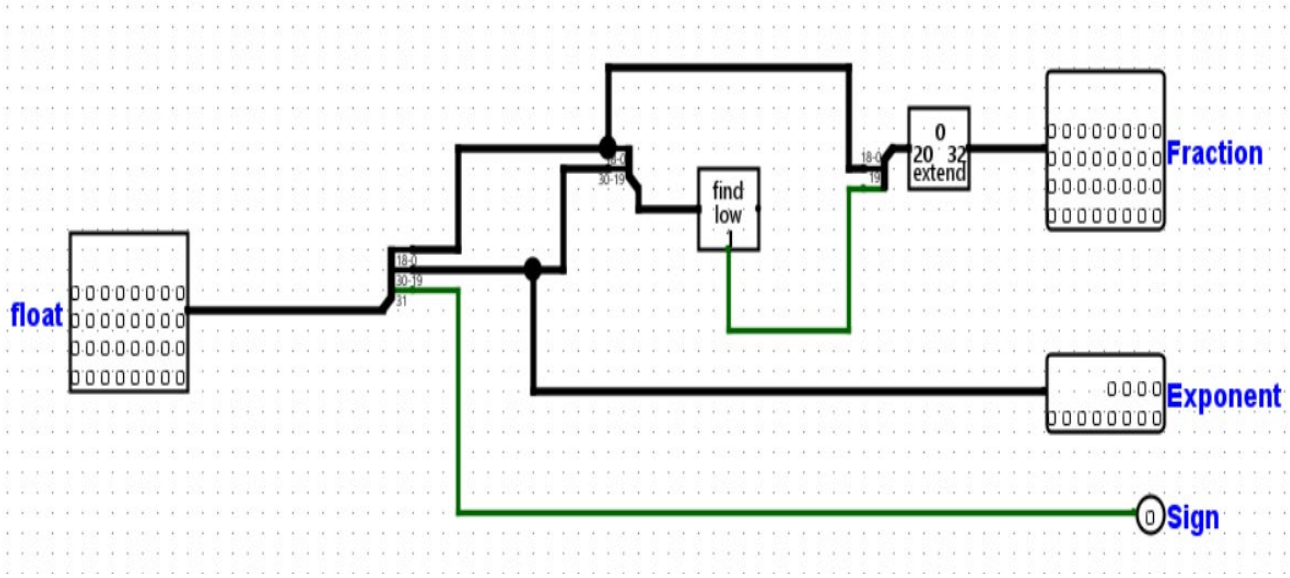



**Figure-2:** Dividing the floating number into sign, fraction and exponent

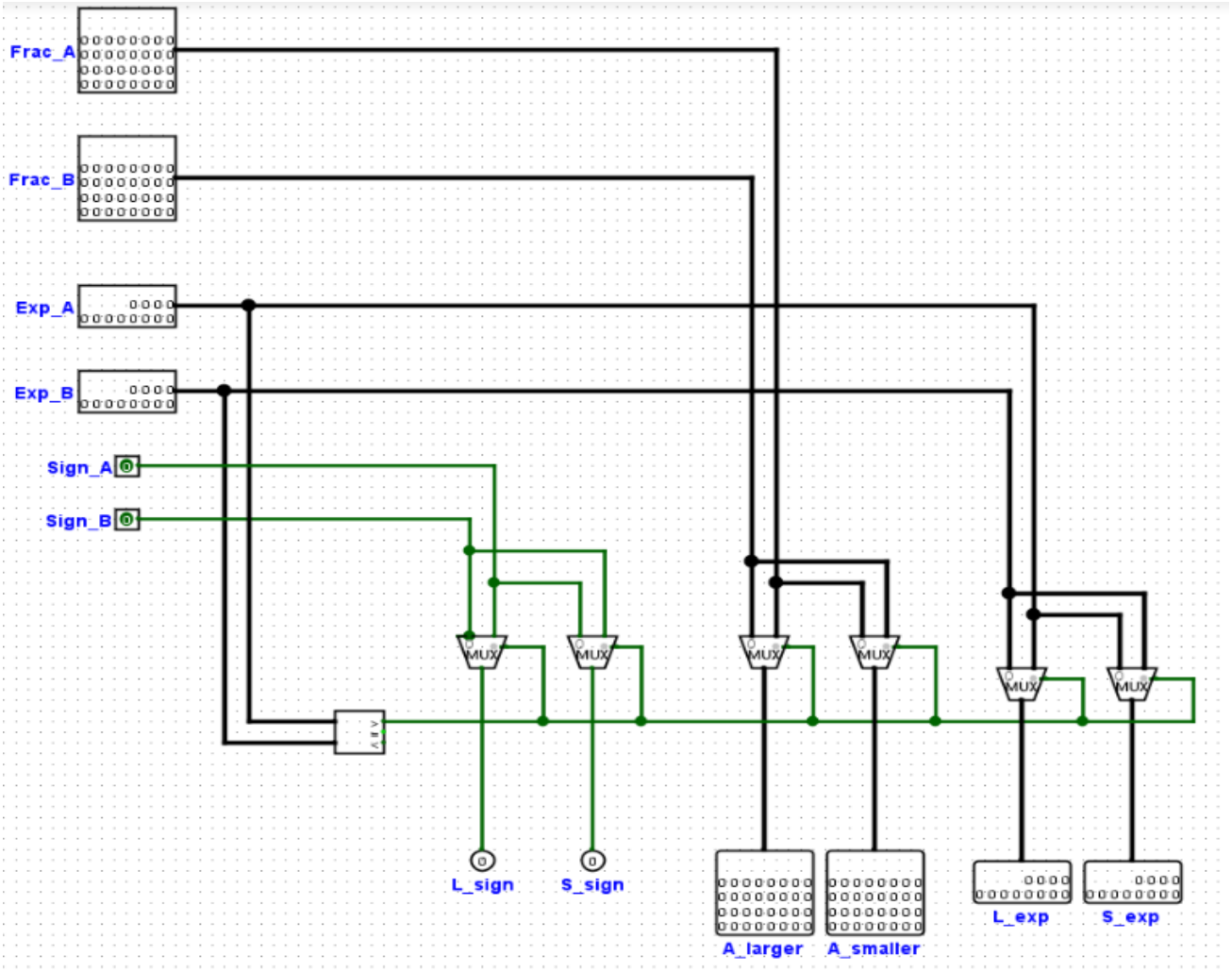



**Figure-3:** Selecting large and small number between two numbers

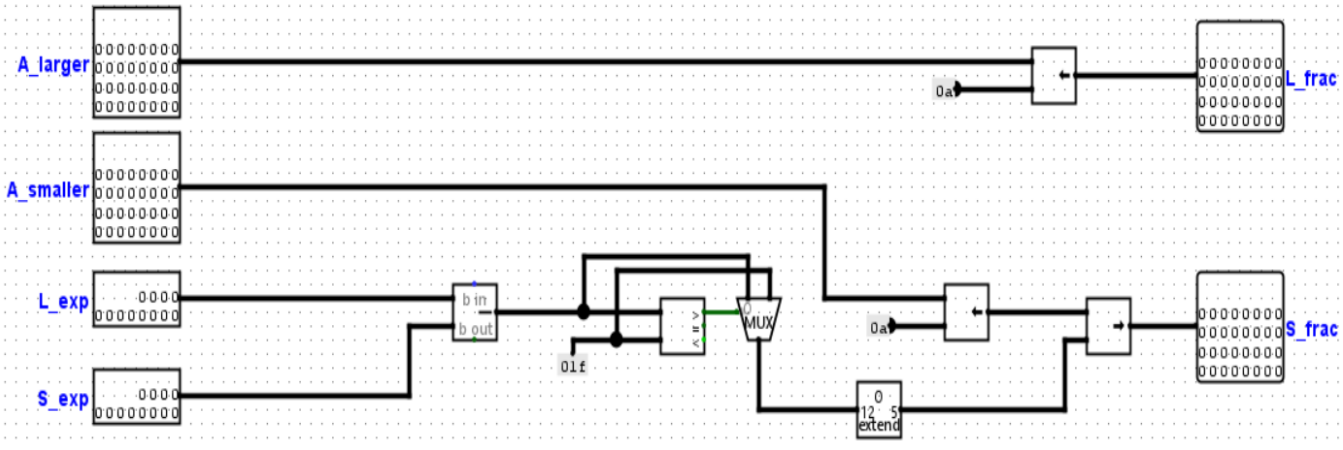



**Figure-4:** Shifting number with smaller exponent

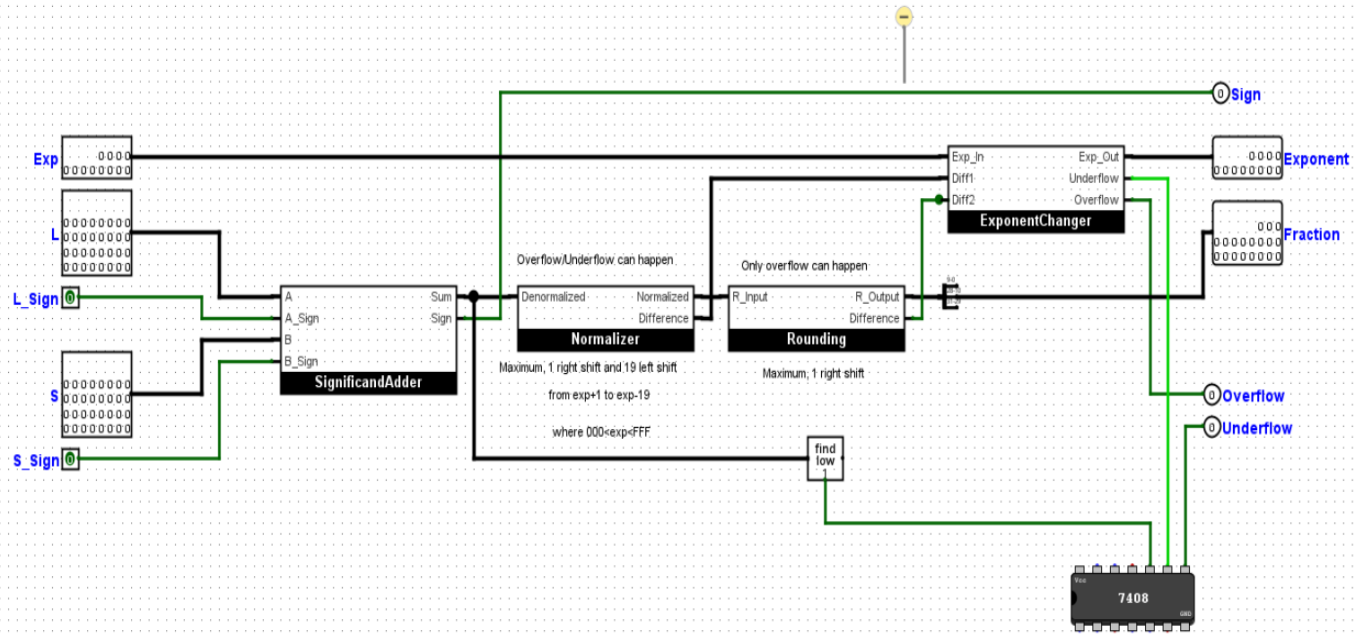



**Figure-5:** Floating point adder – Part2

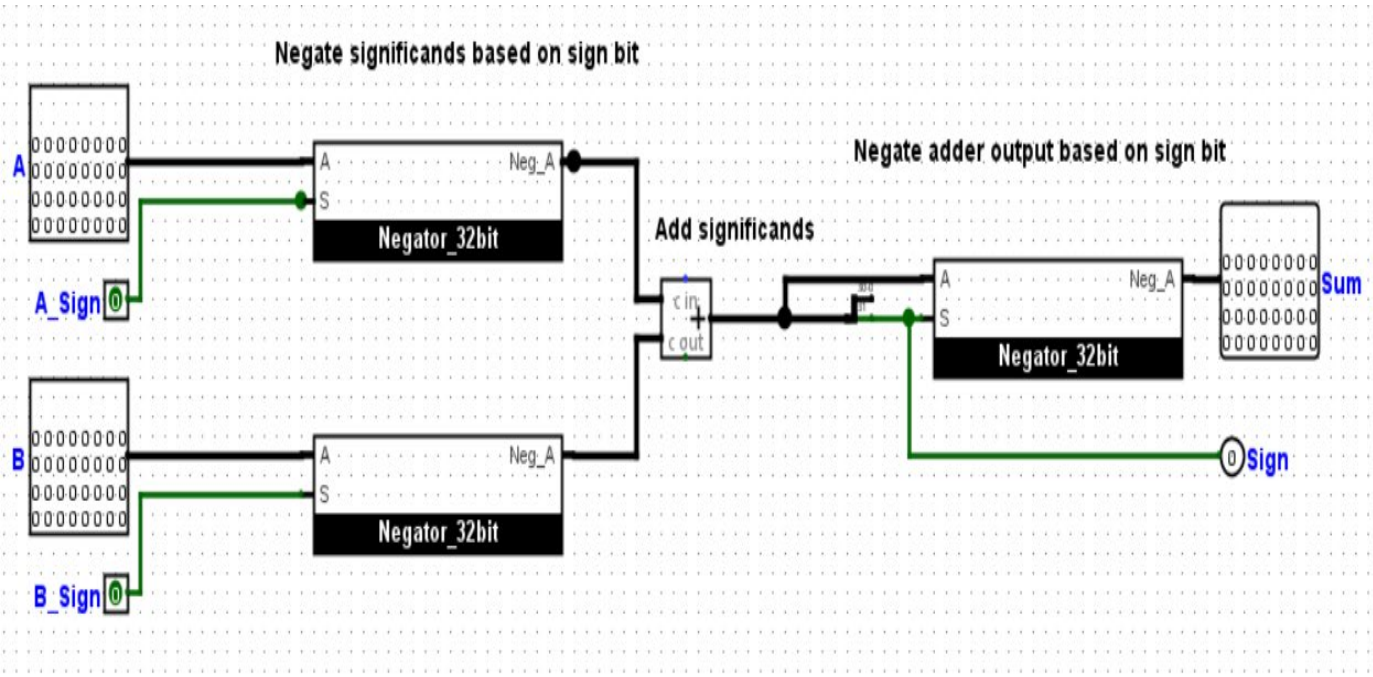



**Figure-6:** Adding significands

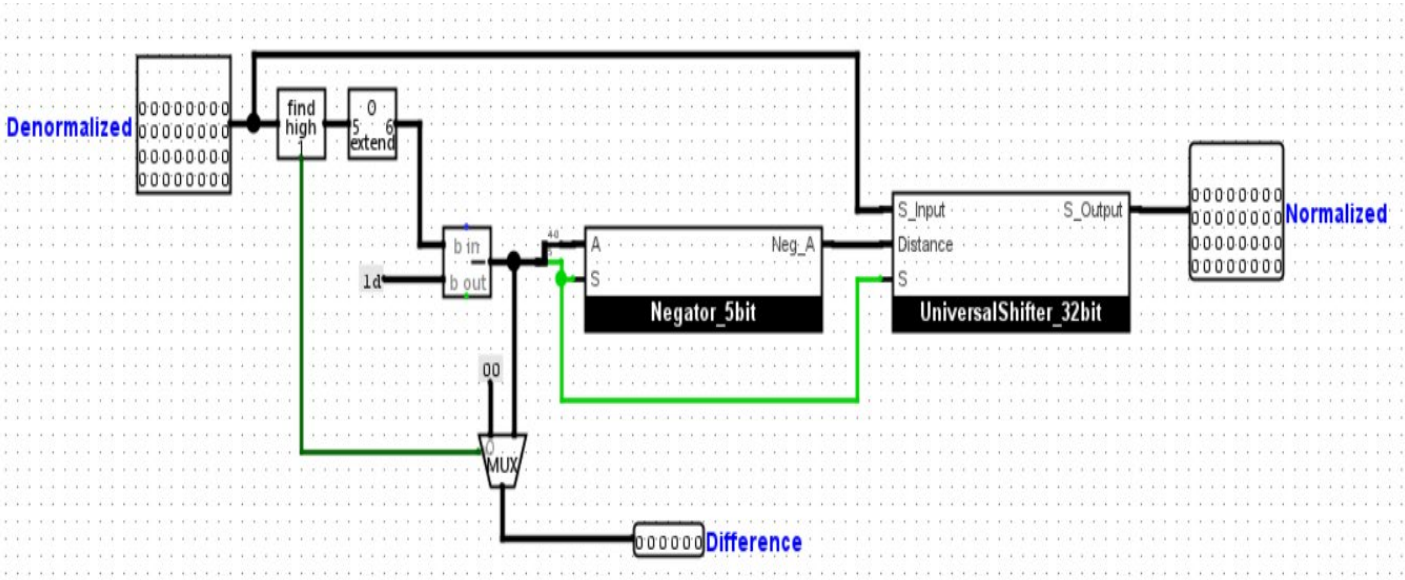



**Figure-7:** Normalizing

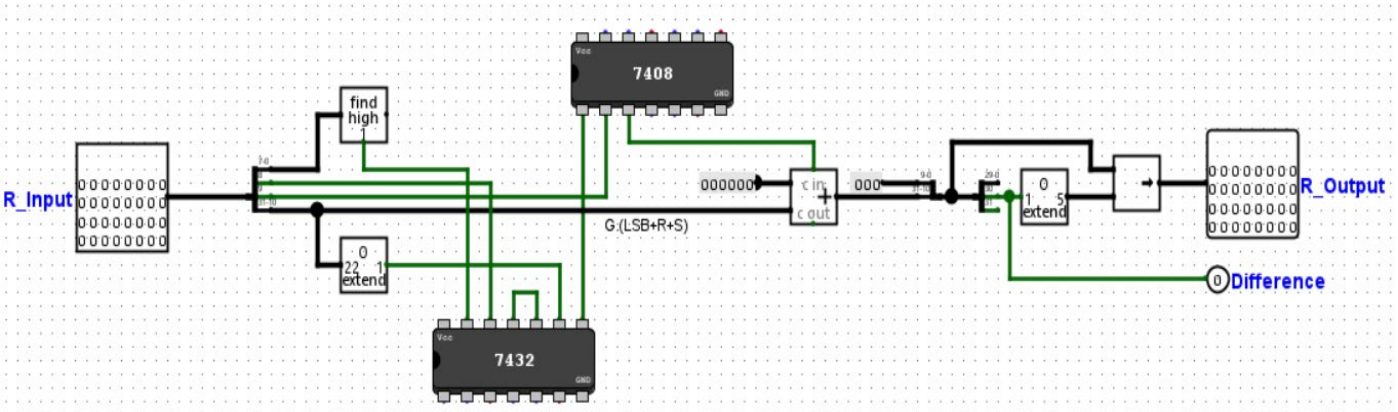



**Figure-8:** Rounding

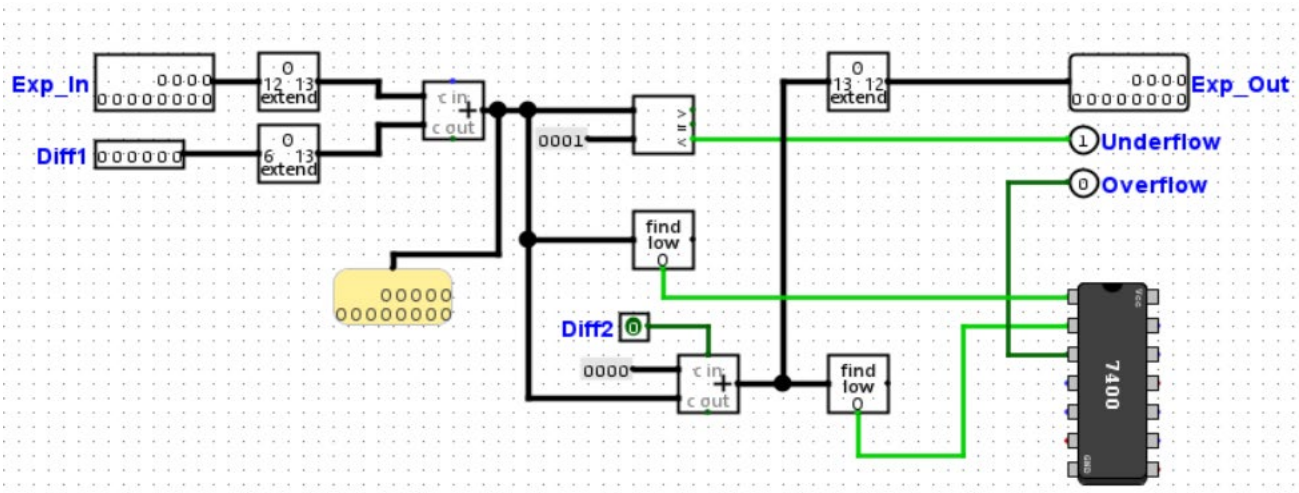



**Figure-9:** Increment/Decrement exponent

**Used Components:**

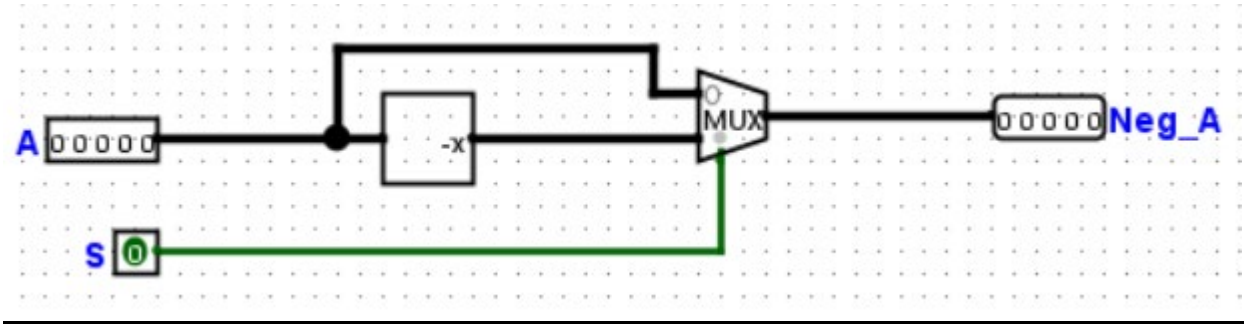



**Figure-11:** 5-bit negator

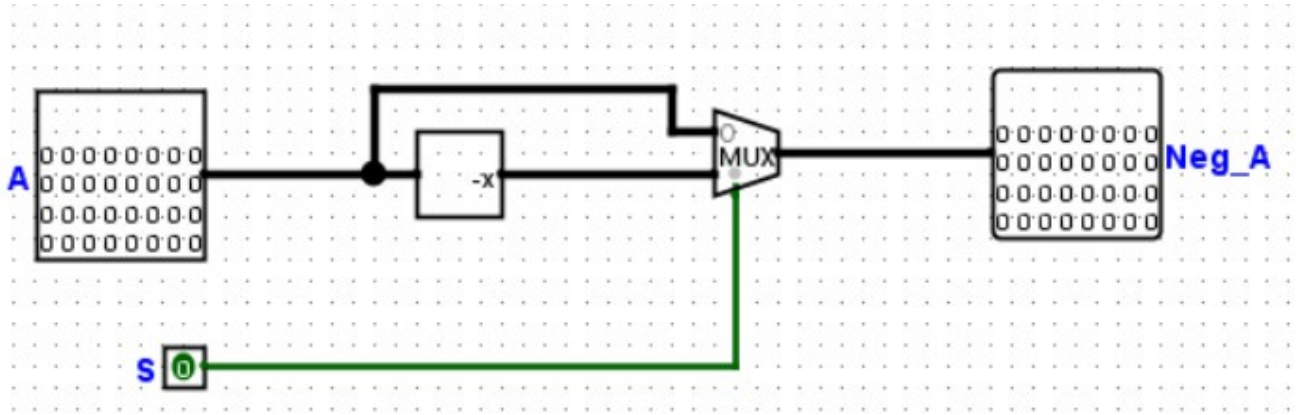



**Figure-12:** 32-bit negator

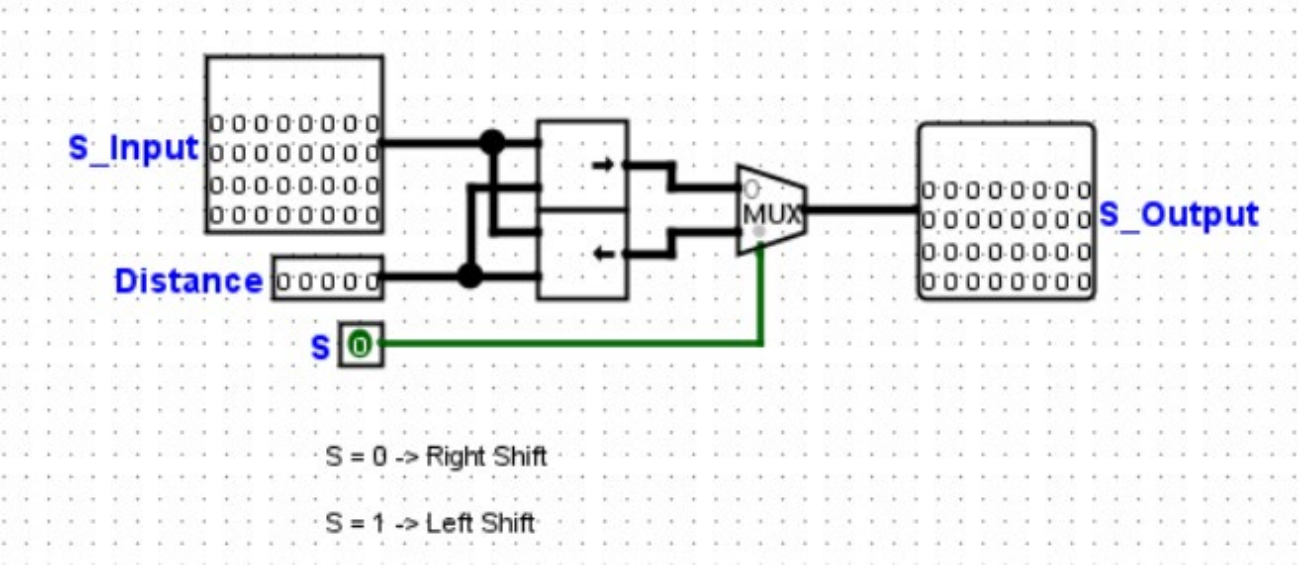



**Figure-13:** Universal Shifter 32 bit

## ICs used with count as a chart:

| Component | Count |
|---|---|
| 12-bit subtractor | 1 |
| 6-bit subtractor | 1 |
| 13-bit adder | 2 |
| 22-bit adder | 1 |
| 32-bit adder | 1 |
| 5-bit negator | 1 |
| 32-bit negator | 1 |
| 12-bit comparator | 1 |
| 13-bit comparator | 1 |
| 5-bit MUX | 1 |

| | |
|---|---|
| 12-bit MUX | 3 |
| 32-bit MUX | 4 |
| 1-bit MUX | 2 |
| 32-bit shifter | 6 |
| Bit finder | 6 |
| Bit extender | 8 |
| 7432 | 1 |
| 7408 | 2 |
| 7400 | 1 |

## Simulator used:

Logisim Version: 3.8.0

## Discussion:

While implementing the circuit, we had to change our design several times. Some designs required a lot of ICs. To optimize number of ICs in our design, we had to discard those. Again, we invested enough of our time in cross-checking corner cases, overflow, underflow conditions for each of sample test cases cautiously.

However, because of truncating and rounding up the sum, sometimes we encountered precision loss of minimum one or maximum two bits. Also, in some cases we reused some logic gate outputs to minimize the number of ICs used. Considering all these aspects, we finally implemented the most optimized design we could find.