# CSE 306

Computer Architecture Sessional

Offline Assignment-02

Floating Point Adder (FPA)

Section-B2, Group-02

Shahad Shahriar Rahman: 2005092

Golam Mostofa: 2005094

Maisha Maksura: 2005099

Mohammad Ishrak Adit: 2005105

Kazi Redwan Islam: 2005108

Date of Submission: 22/01/2024

# 1.Introduction:

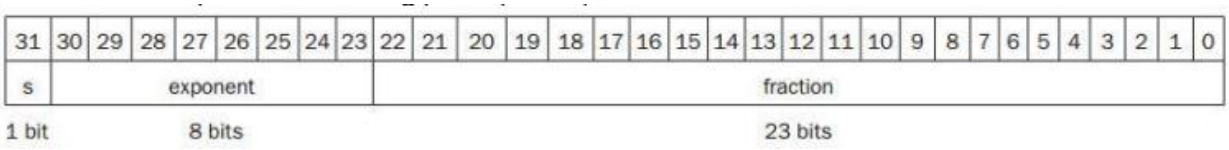
Floating-point is the arithmetic representation of non-integral numbers including very large and small numbers, which uses an integer with specified precision called significand scaled by an integer exponent of a given base. Fractions of all forms can be formatted into the floating-point representation with a certain precision. Floating-point may be considered a form of scientific notation. It is called floating because the radix point can float anywhere to the left or right within the significant digits of the number.

In general, floating-point numbers are of the form

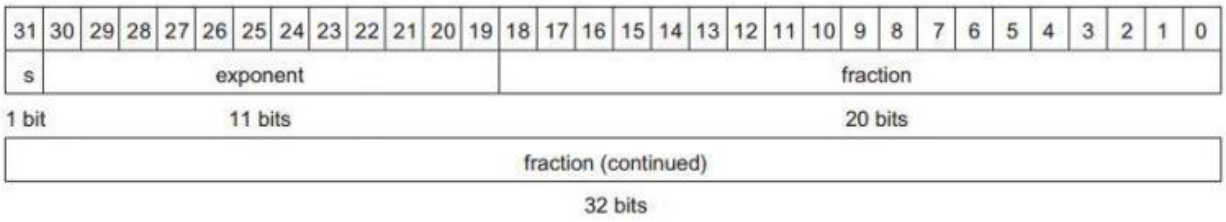
$$(-1)^s * (1 + \text{Fraction}) * 2^{(\text{Exponent - Bias})}$$

- S: Sign Bit (0: non-negative, 1: negative)
- Normalized Significand: $1.0 \leq |\text{significand}| < 2.0$
- Exponent: Biased representation = actual exponent + bias

There are a bunch of ways a floating-point number can be expressed. Among them, the IEEE 754 Standard for Floating-Point Arithmetic is the most widely encountered representation. It was established in 1985 and then gradually gained acceptance and popularity. IEEE 754 can handle signed floating-point numbers.




IEE 754 Floating Point Standard Single Precision floating-point(32-bit)




IEE 754 Floating Point Standard Double Precision floating-point(64-bit)

The algorithm for floating-point addition includes-

- Aligning binary points
- Adding significands
- Normalizing the result and checking for overflow/underflow
- Rounding and renormalizing if necessary

A floating-point unit(FPU) is a part of a computer system especially designed to carry out operations on floating-point numbers. Floating-point adder is a subset of an FPU which performs the addition of two floating-point numbers.

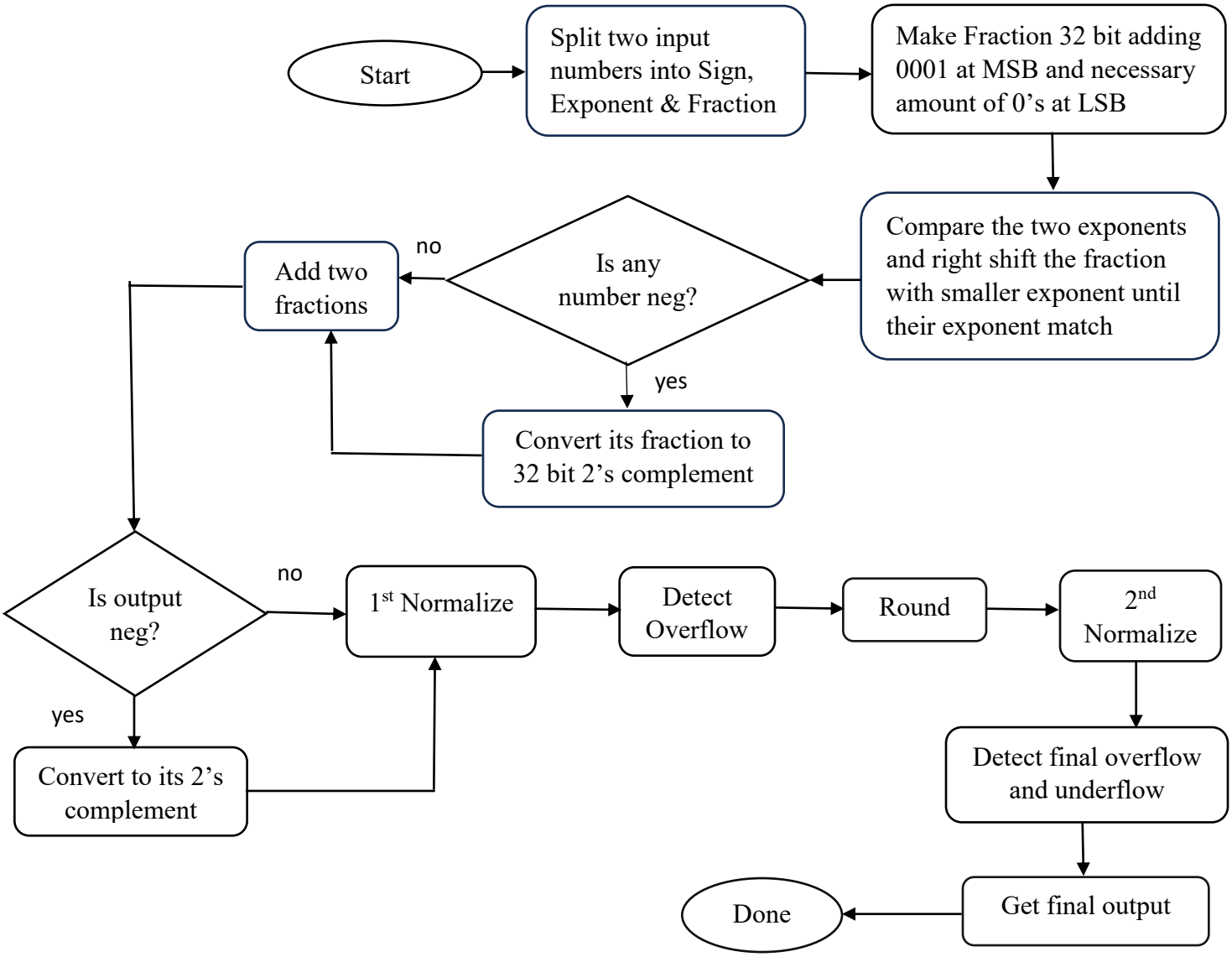
## 2.Problem Specification:

Design a floating point adder circuit which takes two floating point as inputs and outputs their sum, another floating point number. Each floating point will be 32 bits long with following representation:

| **Sign** | **Exponent** | **Fraction** |
|---|---|---|
| 1 Bit | 11 Bits | 20 Bits |

All i/o numerics are in IEEE 754 Standard Format of Binary Representation of floating point numbers.

## 3.Flow Chart:


Start
Split two input numbers into Sign, Exponent & Fraction
Make Fraction 32 bit adding 0001 at MSB and necessary amount of 0's at LSB
Compare the two exponents and right shift the fraction with smaller exponent until their exponent match
Is any number neg?
no
yes
Add two fractions
Convert its fraction to 32 bit 2's complement
Is output neg?
no
yes
1st Normalize
Detect Overflow
Round
2nd Normalize
Convert to its 2's complement
Detect final overflow and underflow
Get final output
Done

## 4.High-level Block Diagram:

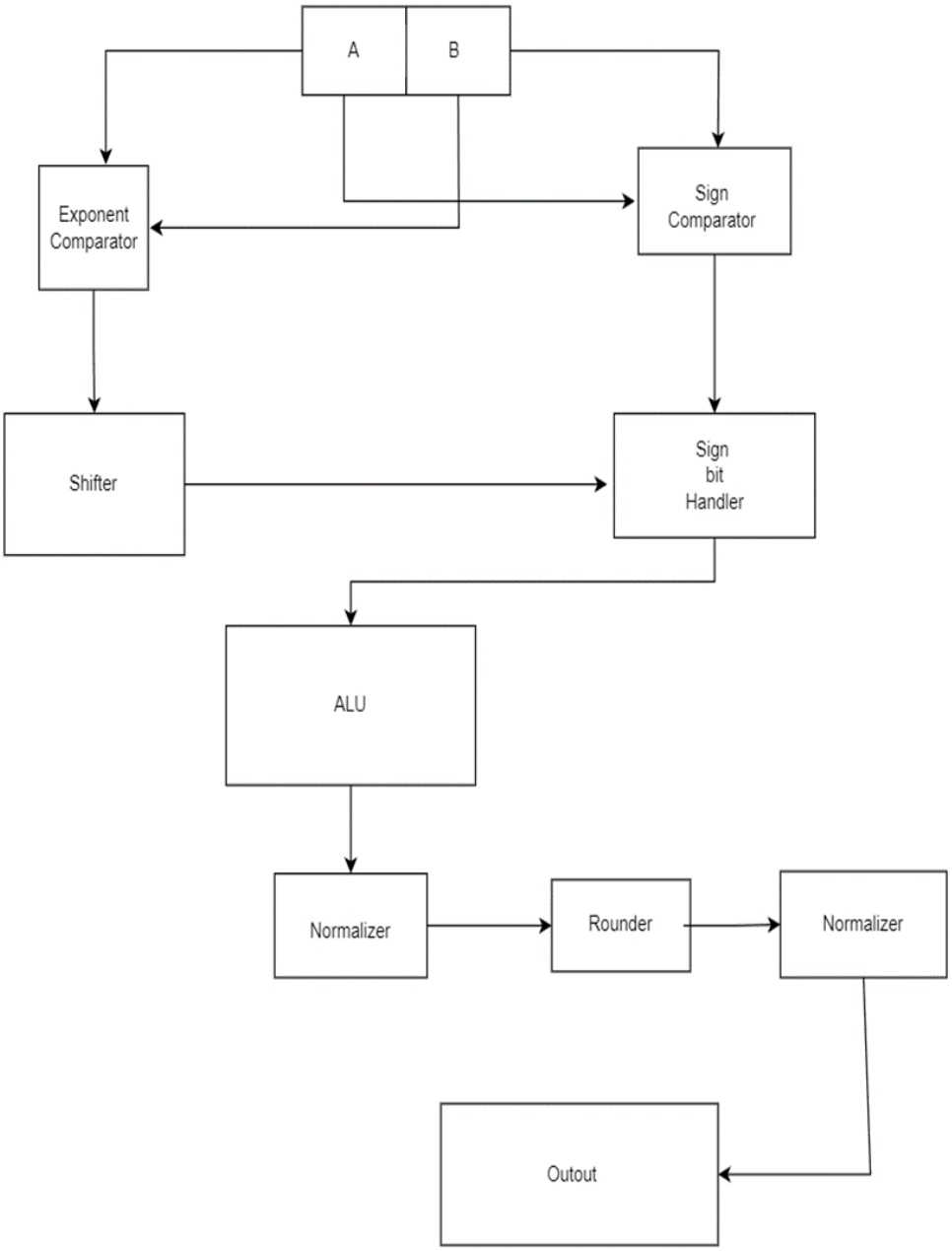



Fig-1: High-level Block Diagram of FPA

## 5.Circuit Diagram:

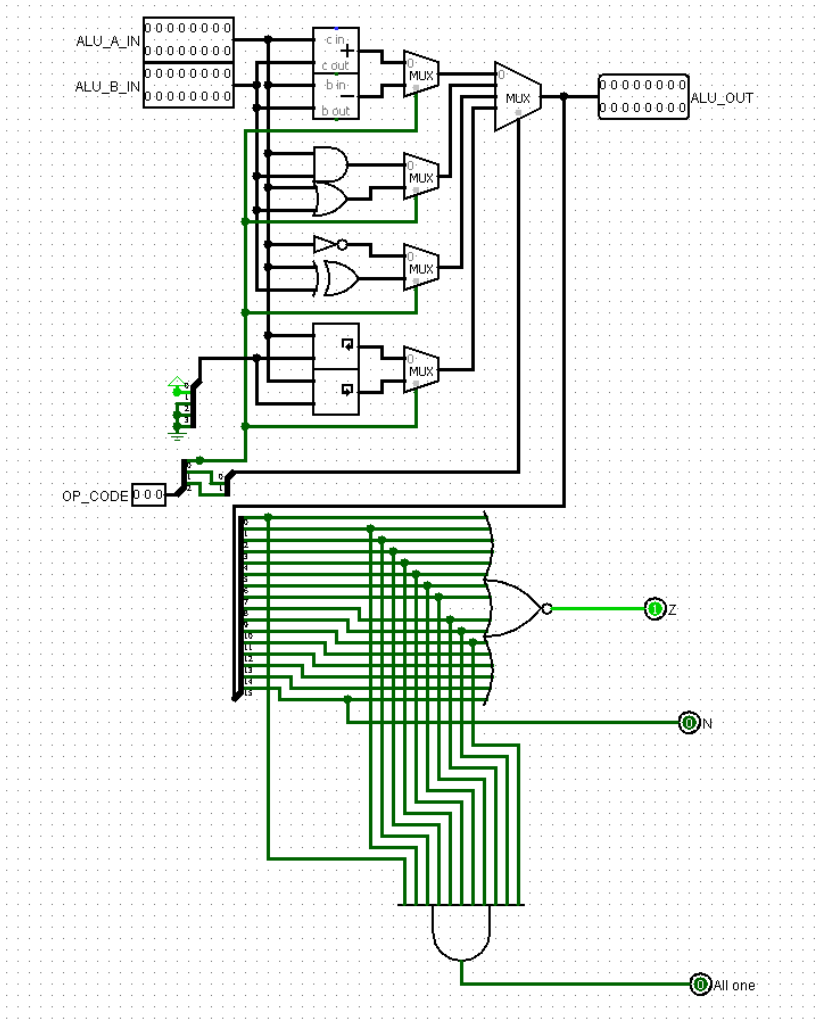



Fig-2: ALU-16 bit

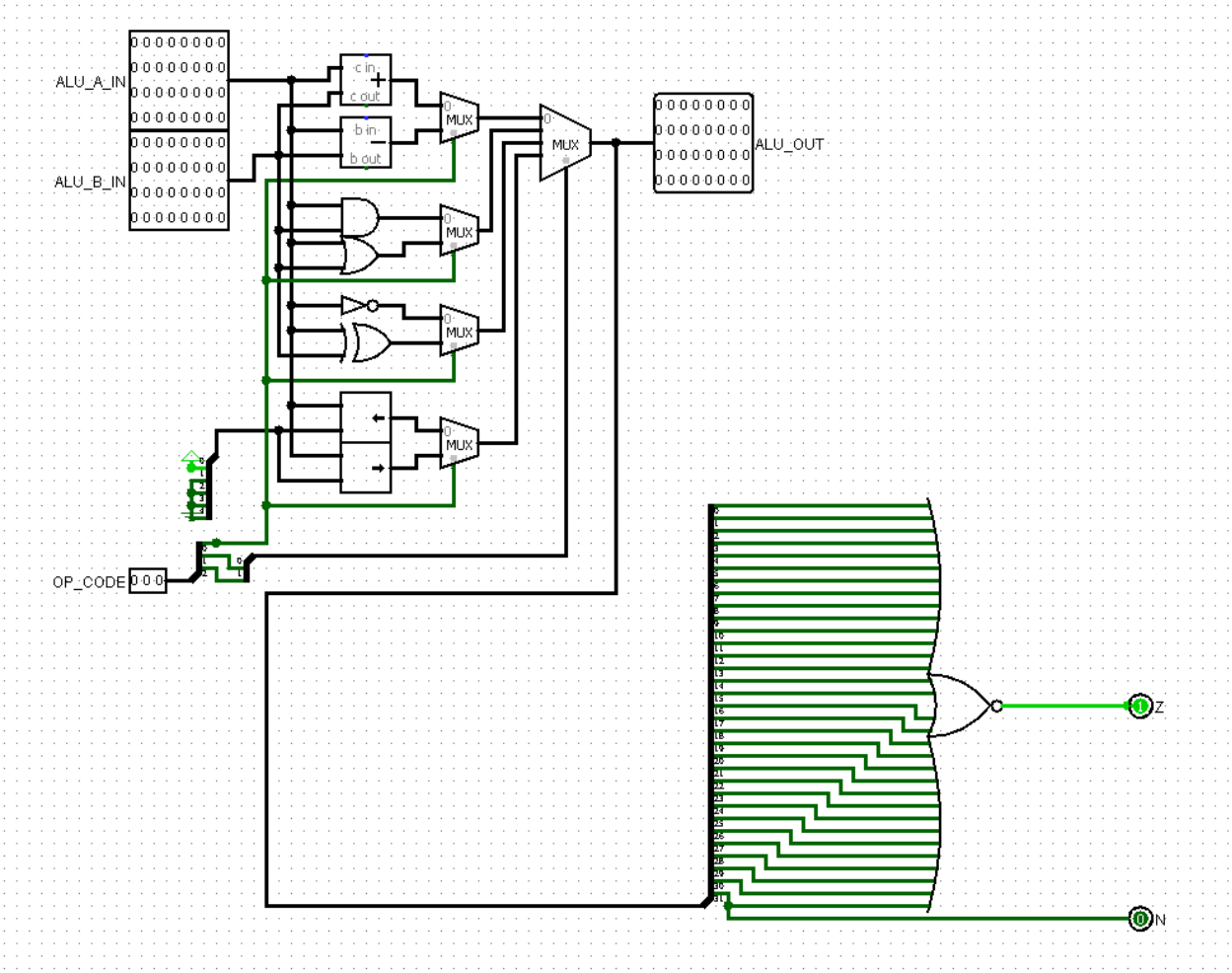



Fig-3: ALU-32 bit

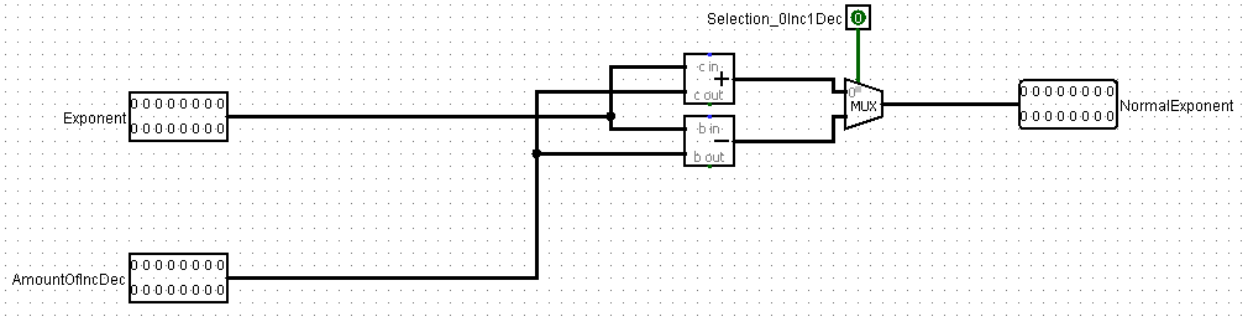



Fig-4: Increment-Decrement circuit

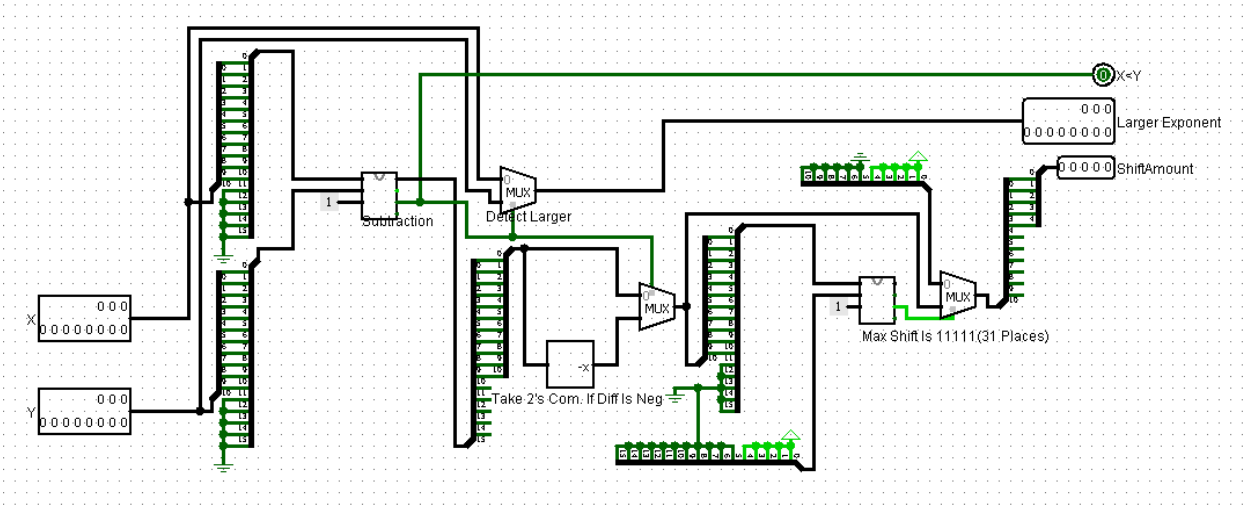



Fig-5: Norm_Comparator Circuit

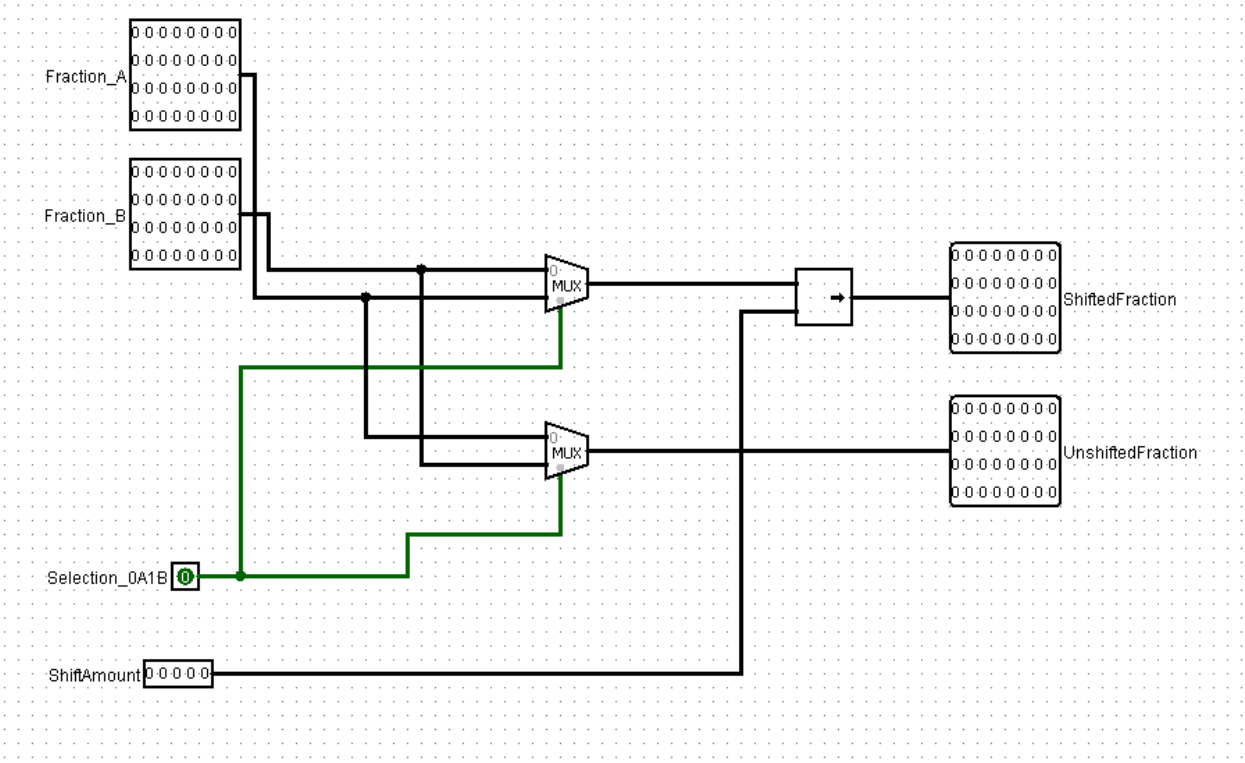



Fig-6: Right Shifter

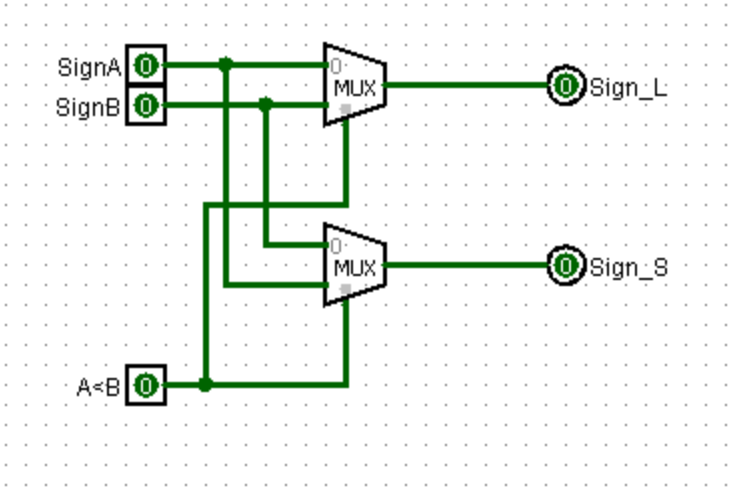



Fig-7: Sign Bit Comparator

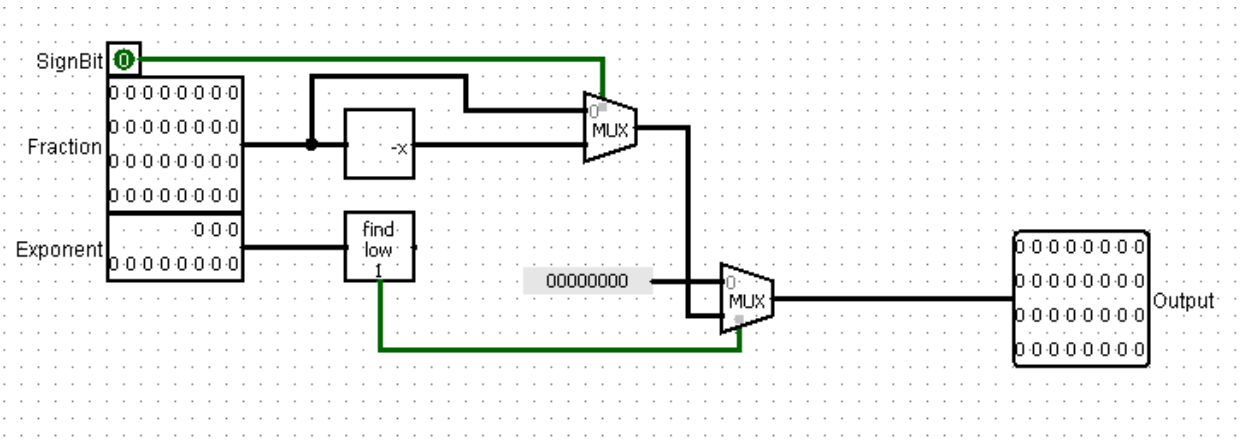



Fig-8: Sign Handler

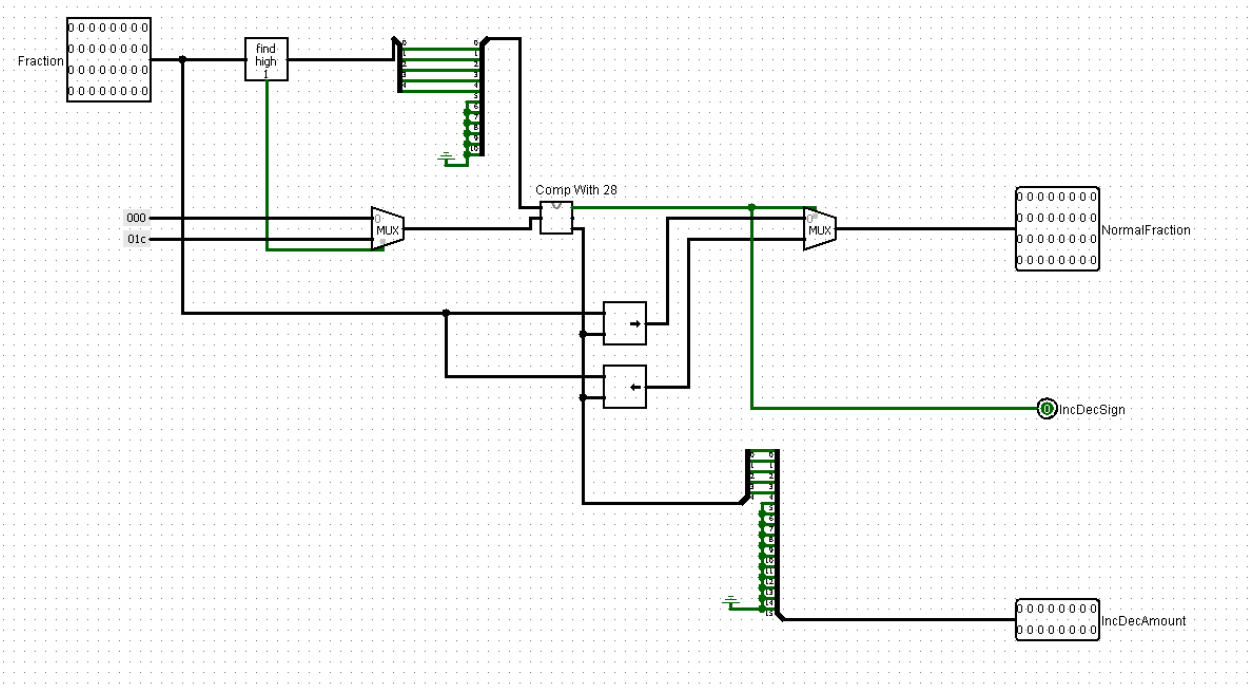



Fig-9: Sign Shifter

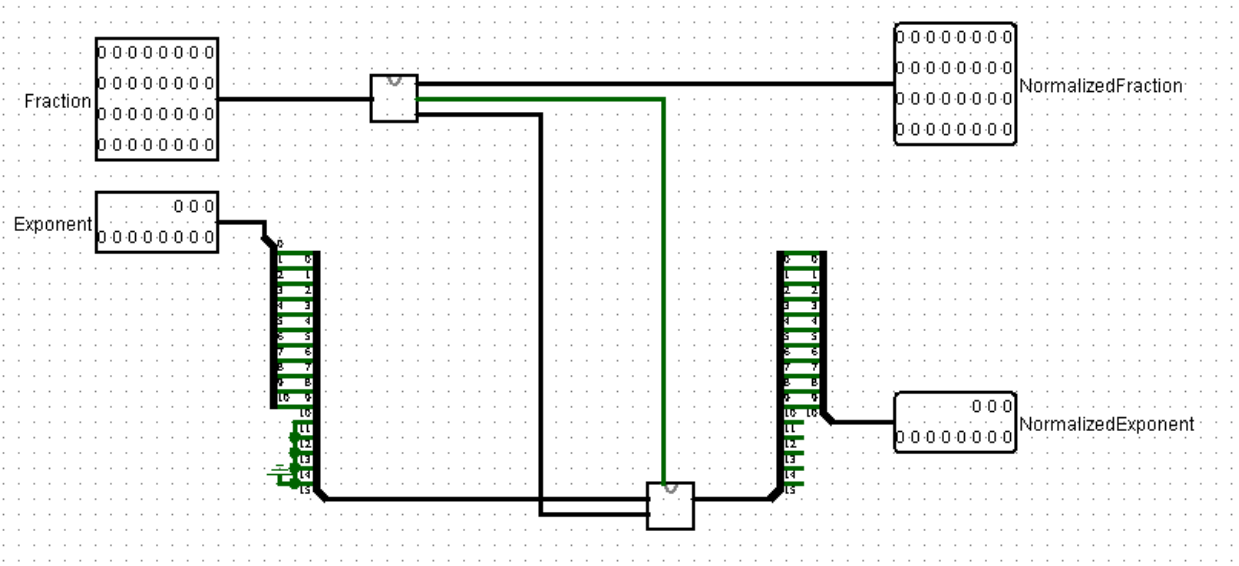



Fig-10: Normalizer

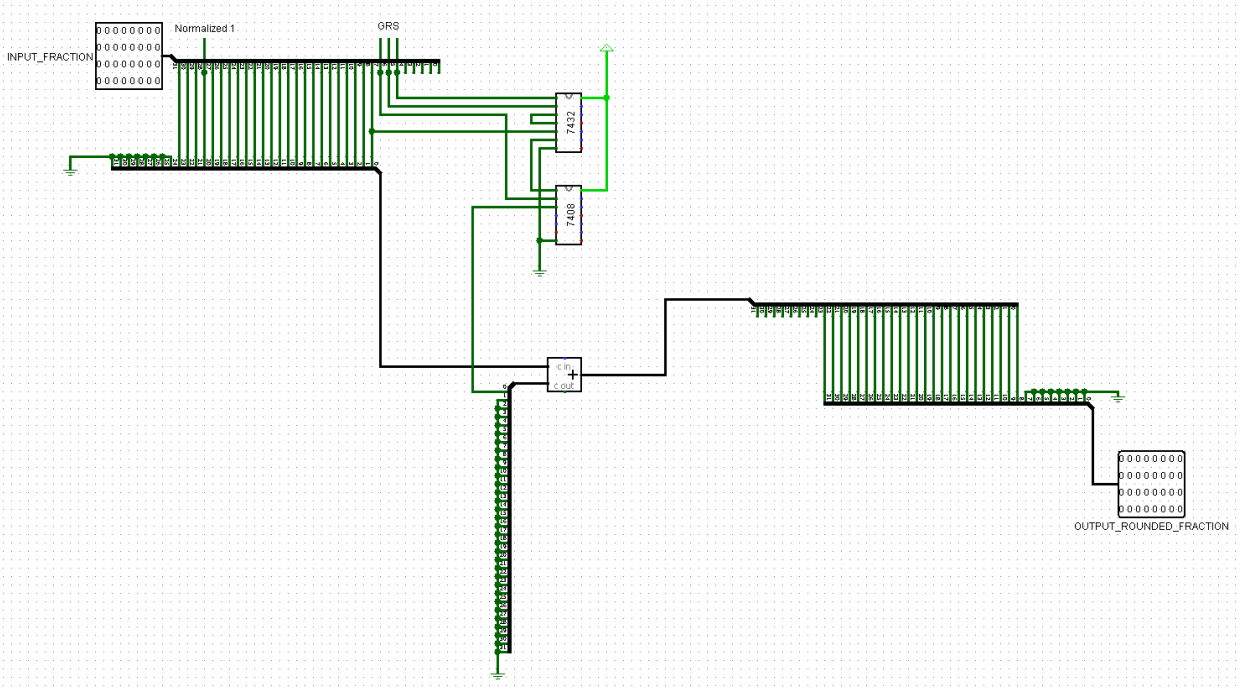



Fig-11: Rounder

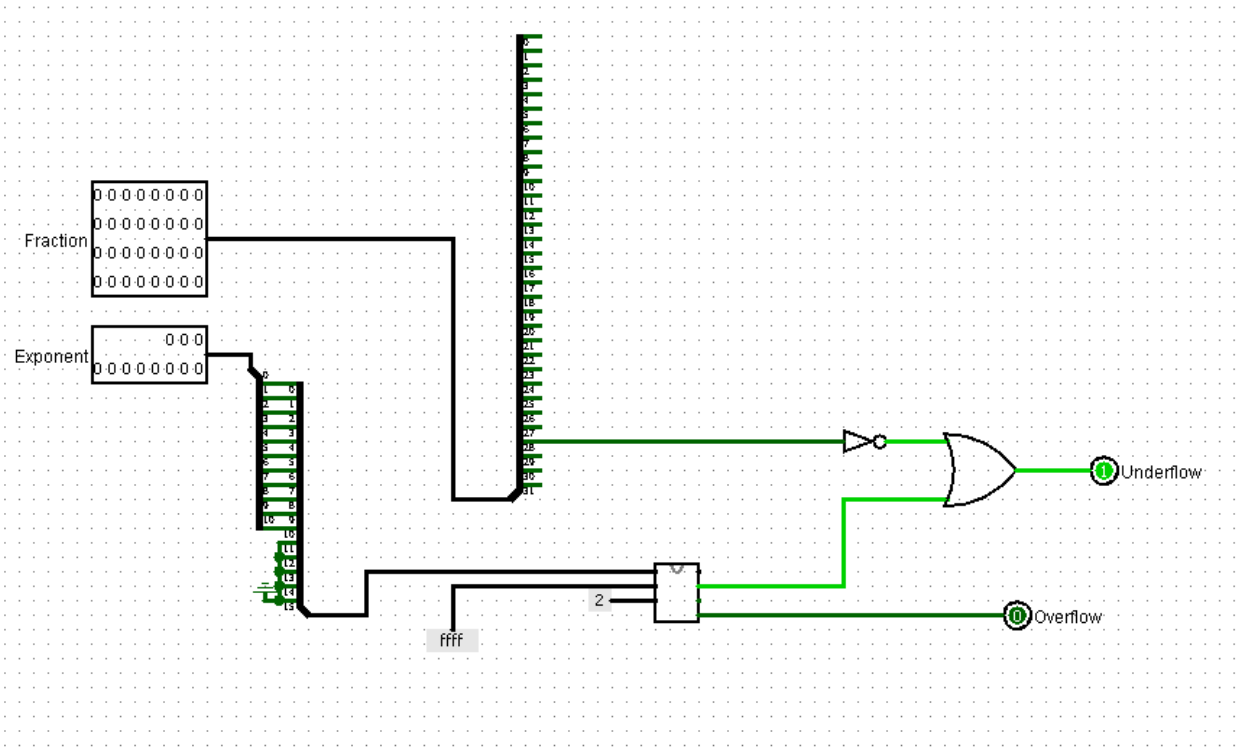



Fig-12: Overflow-Underflow flag generator

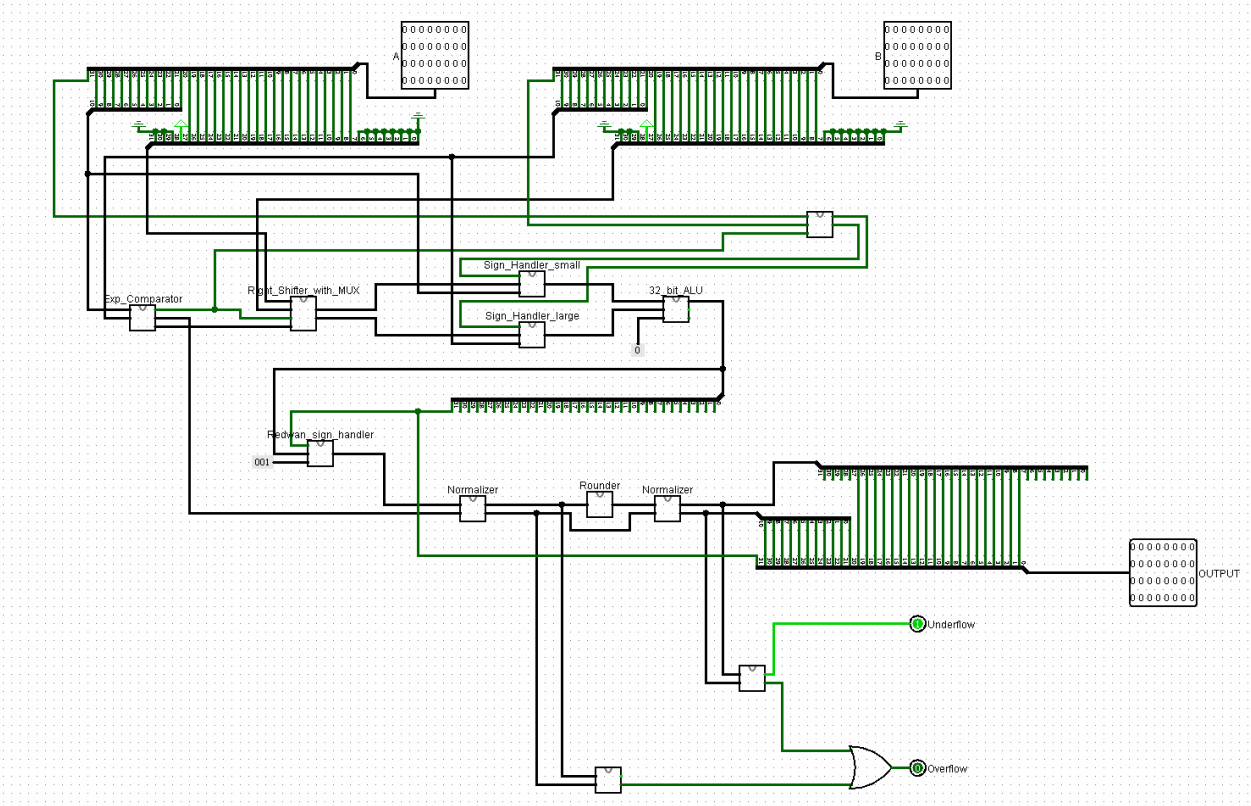



Fig-13: Full Circuit

## 6.ICs Used with Count as a Chart:

| Component | Count |
|---|---|
| IC 74153 | 53 |
| IC 7483 | 3 |
| IC 7404 | 2 |
| IC 7432 | 2 |
| IC 7408 | 1 |

## 7.Simulator:

Logisim 2.7.1

# 8.Discussion:

Due to the unusual representation of floating-point numbers, assembling an adder circuit for such numerals was quite tricky. Several subcircuits for specific subroutines were designed for use before and after the addition operation.

For proper addition, subtraction was required on exponents and addition was required on fraction parts; these were done by ALU circuits, built separately.

To use the 2's complement instead of the given input when the sign bit is negative, we designed the sign handler sub circuit.

In between these subcircuits, we carefully handled the transfer of the different parts of given inputs correctly, ensuring they do not get mixed up.

After addition, we normalized the result by proper amount of shifting of fraction while exponent was incremented or decremented accordingly.

After normalization, we attempted to make the number simpler with value as close as possible to the actual value, by rounding. The rounding subcircuit handles this properly but have the prospect to denormalize the number again. For correction of this effect, we normalize the rounded number one more time.

After all parts (sign bit, exponent, mantissa) of given inputs being handled properly by themselves, we then merged the outcomes to get the proper operation value.

In short, the given assignment was a proper example of splitting a bigger routine into subroutines, compartmentalized calculation of the subsections and obtaining result by merging properly.