CSE 306
Computer Architecture Sessional

Assignment-2: 32-bit Floating Point Adder Simulation

Lab Section - A1
Group - 06

22 January, 2024

Members of the Group:

i. 2005020 - Mostafa Rifat Tazwar

ii. 2005025 - Most. Sonia Khatun

iii. 2005027 - Swastika Pandit

iv. 2005029 - MD. Minhajul Islam Fuad

v. 2005030 - Fairuz Mubashwera

1 Introduction

A floating point adder is a circuit that can perform addition (and subtraction) operations of floating point numbers.Floating-point number is basically a representation of real numbers with both integer and fractional parts while working in computers.This representation can deal the numbers that are too big or too small for integer representation to deal with.

Floating point representation has three fields with fixed number of bit size : sign field, exponent field and mantissa or fraction field. Generally floating point numbers are of the form:
$(-1)^{Sign} * (1 + Fraction) * 2^{Exponent-Bias}$
Where Bias depends on the number of bits used for representing the exponent field.If exponent field is 'e' bit then $Bias = 2^{e-1} - 1$
A floating-point adder operates by aligning the decimal points of the two numbers to be added and then adding their mantissas. The exponent of the result remains fixed, necessitating shifts and corresponding adjustments, such as incrementing or decrementing, through the normalization and rounding processes. The sign of the result is determined by the signs of the two numbers being added.

A floating-point adder finds application in diverse fields such as scientific and engineering computations, financial modeling, and computer graphics. It is employed in co-processors to execute rapid, hardware-accelerated floating-point arithmetic, particularly beneficial for computationally intensive tasks. The specialized floating-point adder exhibits superior speed compared to the main processor, proving advantageous in critical applications like scientific simulations, data analysis, and other operations demanding high-precision floating-point calculations.

2 Problem Specification

The assignment asks to design a floating point adder circuit which 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	11 bits	20 bits

Table 1: Problem Specification

3 Circuit Diagram

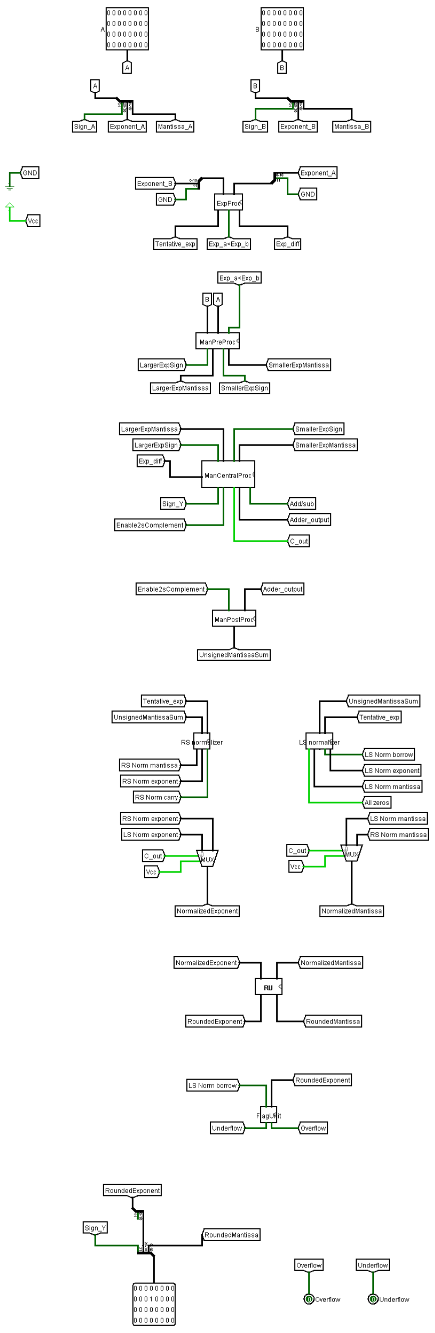


Figure 1: Flow chart of the addition/substraction algorithm

4 Flowchart of the Addition-Subtraction Algorithm

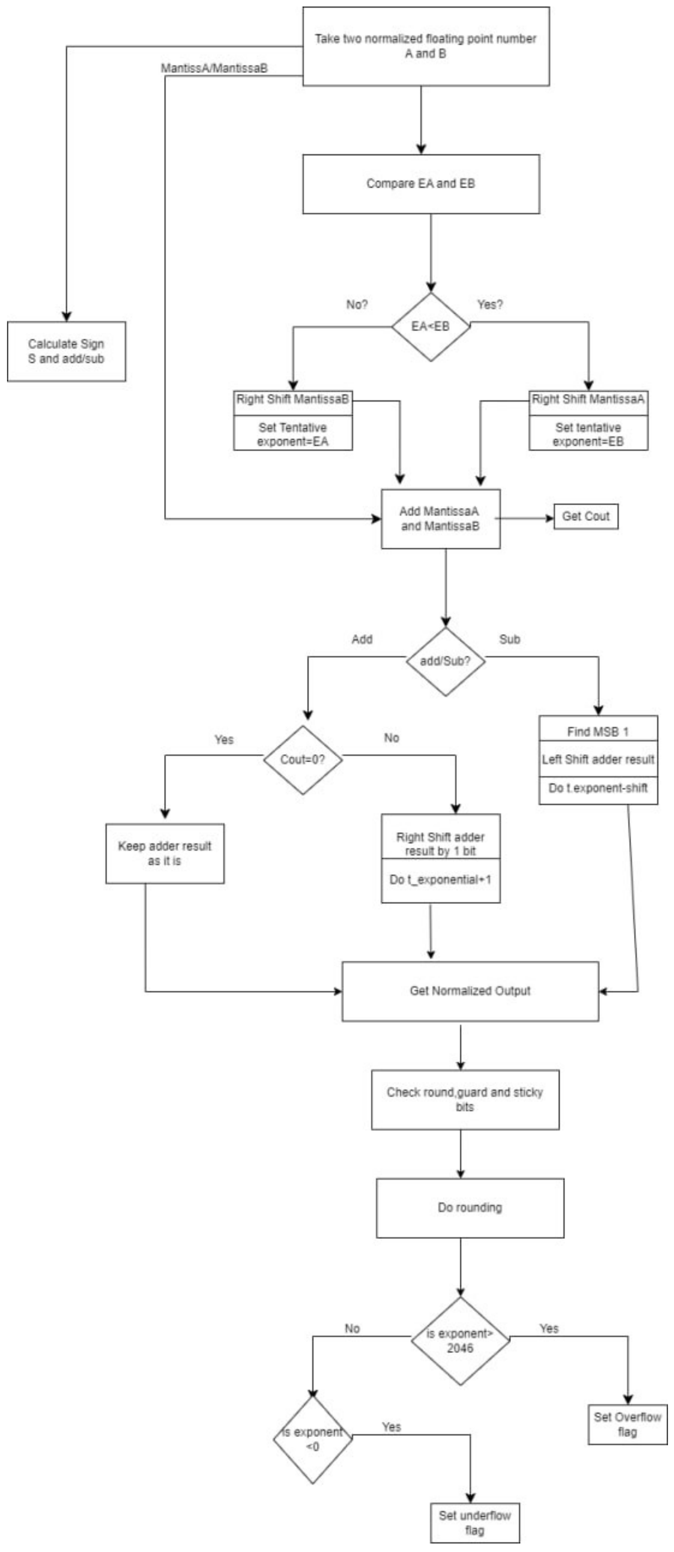


Figure 2: Flow chart of the addition/substraction algorithm

5 Description and Circuit Diagram of Modules

5.1 Exponent Processor Unit

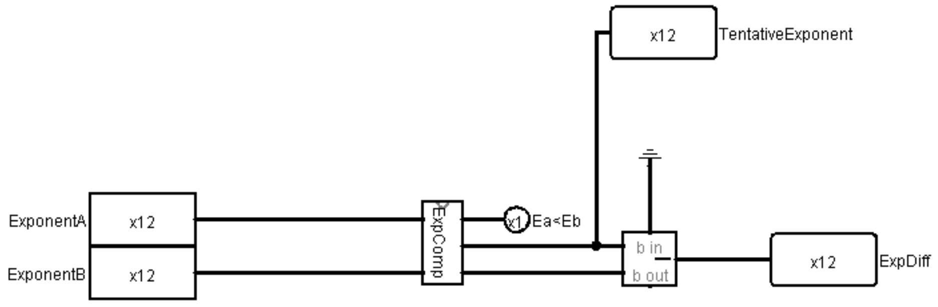


Figure 3: ExponentProcessor.circ

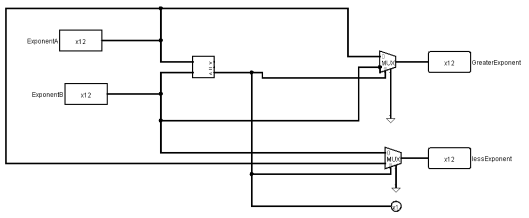


Figure 4: ExponentComparator.circ

5.2 Mantissa Processor Unit

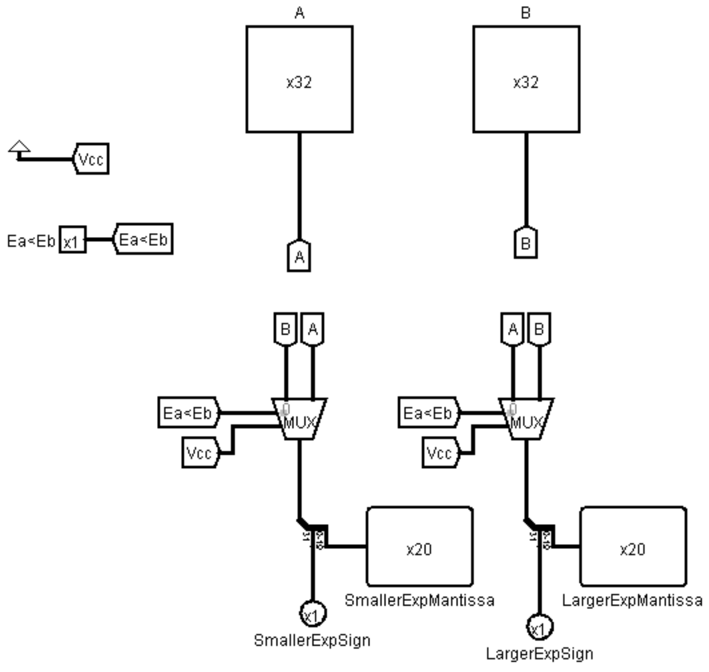


Figure 5: MantissaPreprocessor.circ

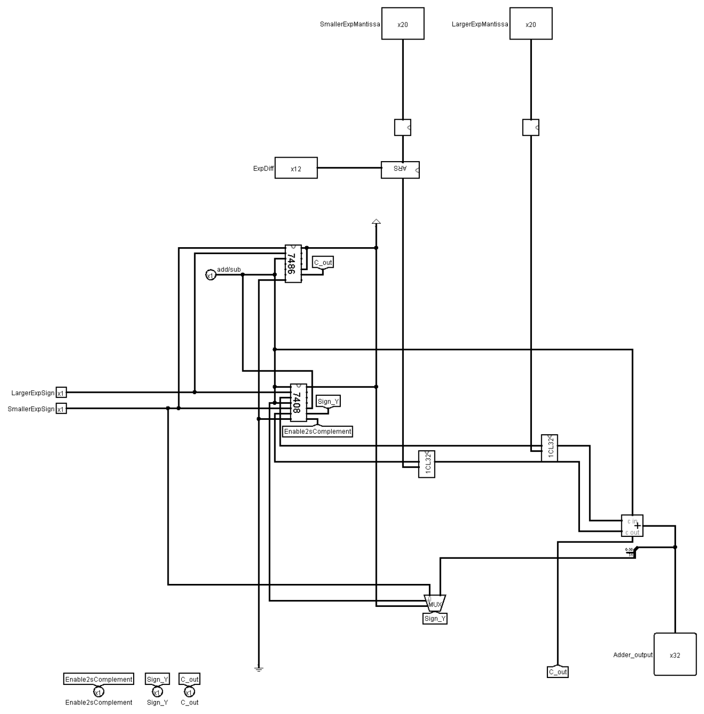


Figure 6: MantissaCentralProcessor.circ

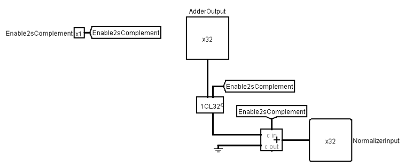


Figure 7: MantissaPostProcessor.circ

5.3 Normalizer Unit

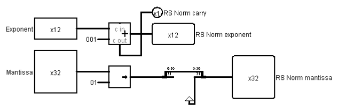


Figure 8: Right Shift Normalization.circ

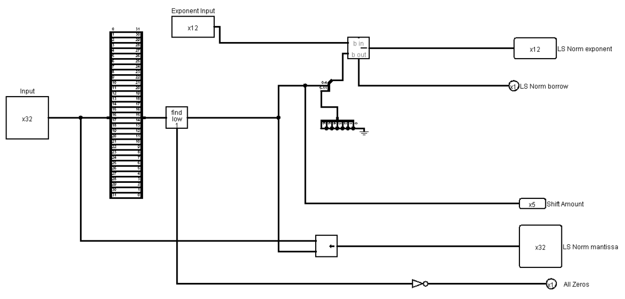


Figure 9: Left Shift Normalization.circ

5.4 Rounding Unit

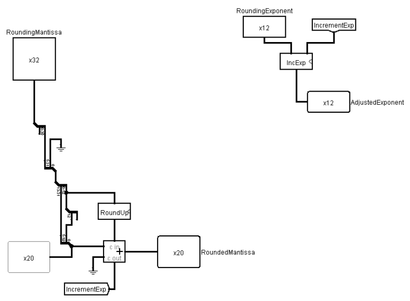


Figure 10: Rounding Unit.circ

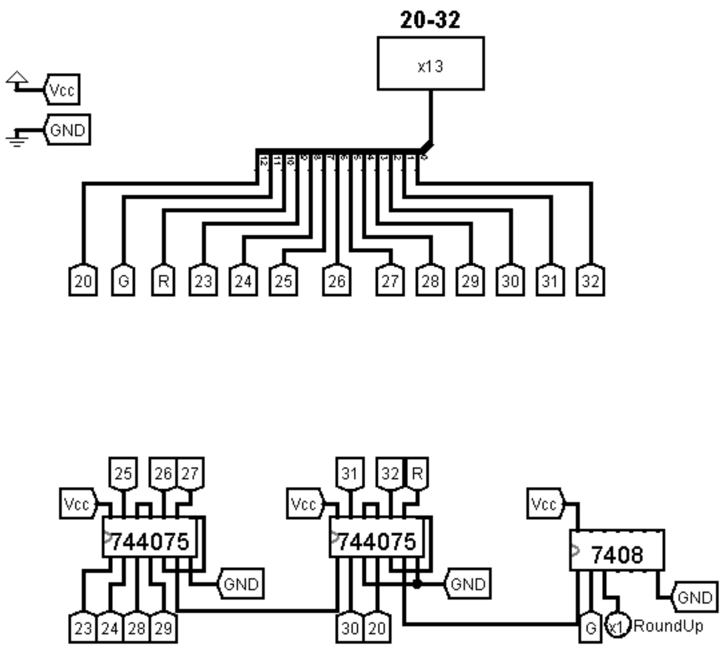


Figure 11: Round Up Flag.circ

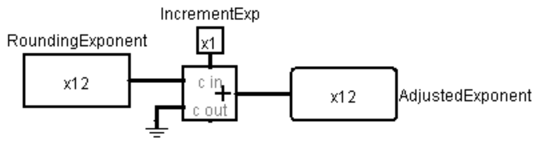


Figure 12: Increment Exponent.circ

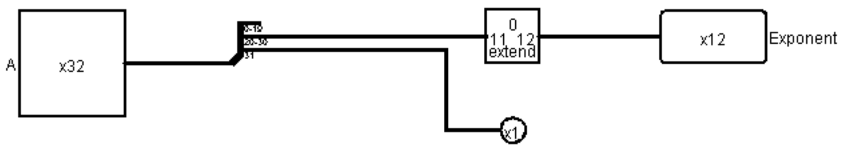
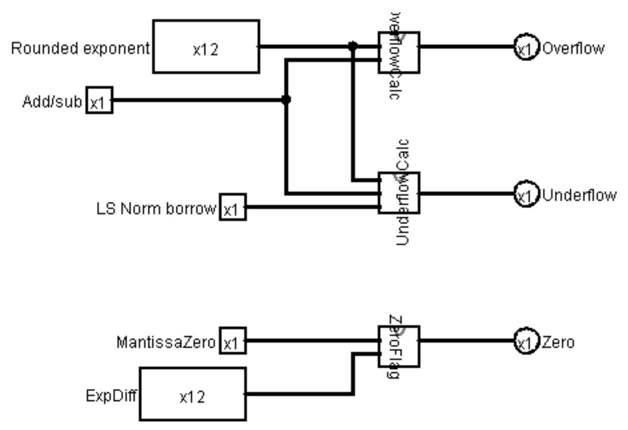


Figure 13: Initial Extractor.circ

5.5 Flag Unit



Input : Rounded exponent from rounding unit and LS Norm borrow from subtract_circuit
Output : Overflow and Underflow flag

Figure 14: Flag Unit.circ

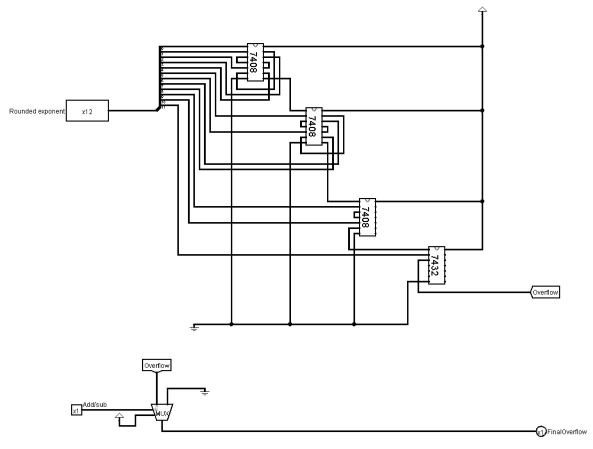


Figure 15: Overflow.circ

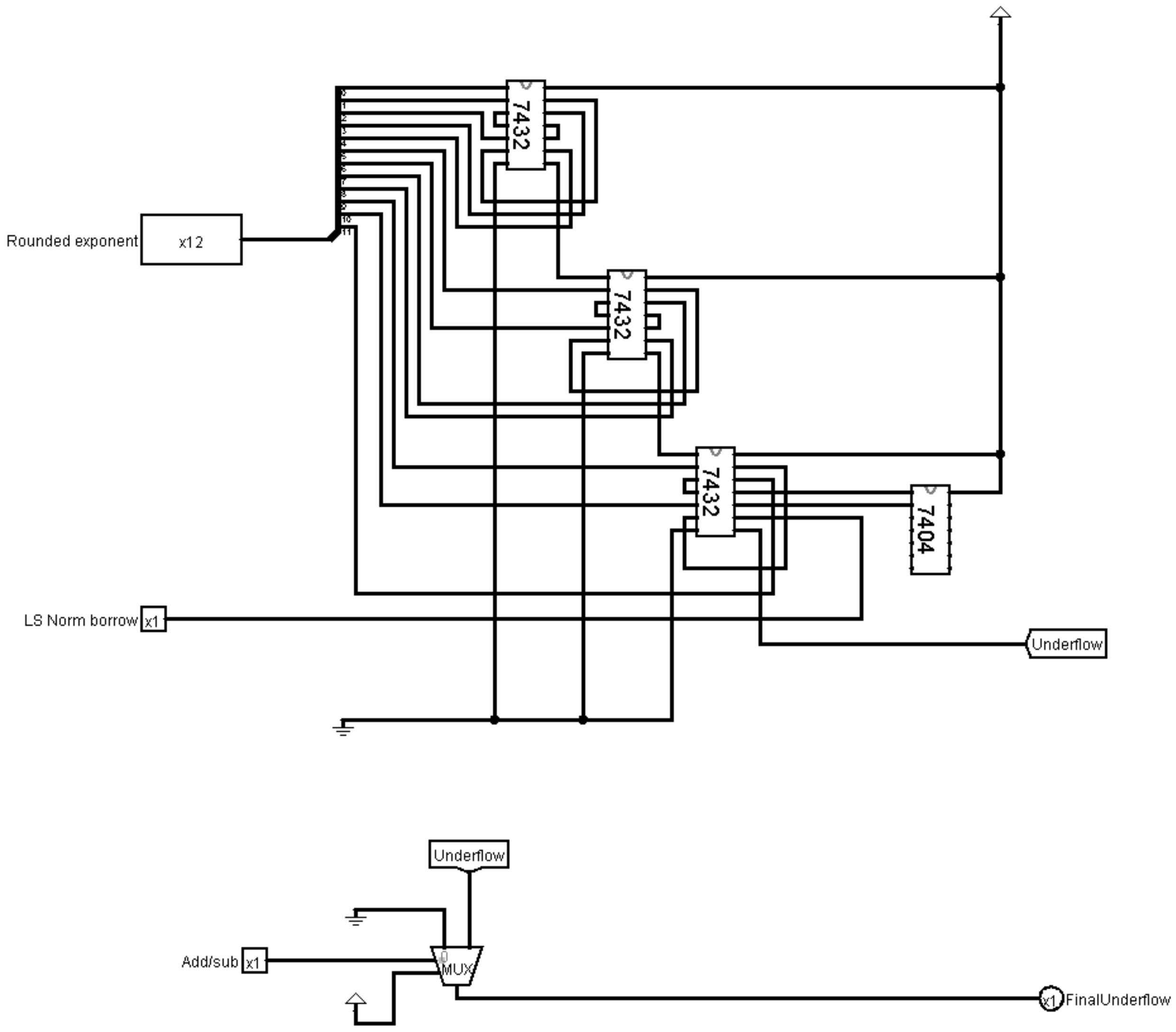


Figure 16: Underflow.circ

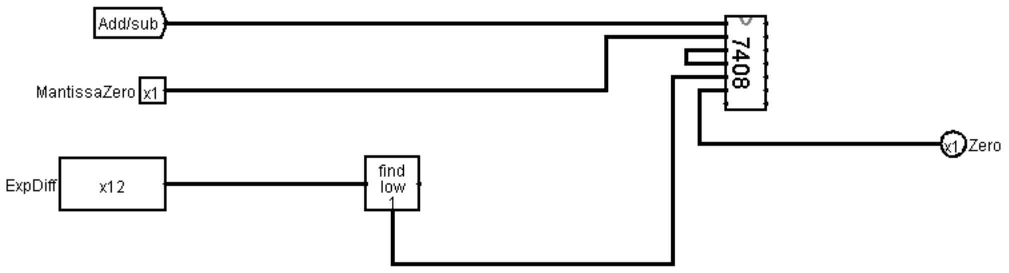


Figure 17: Zero.circ

6 High-level Block Diagram of the Architecture

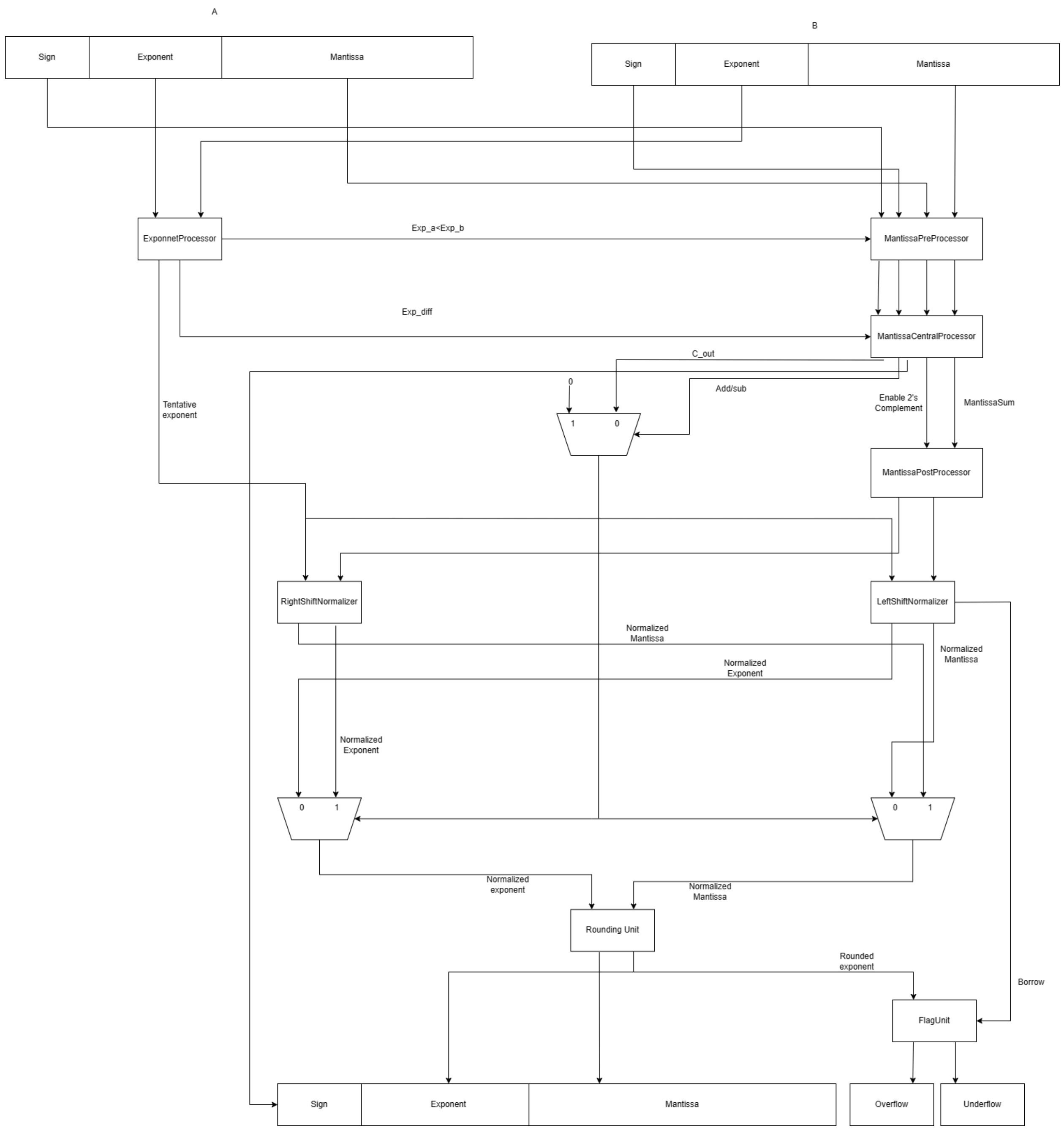


Figure 18: High-level Block Diagram of the Architecture

7 Design Description

7.1 Exponent Processing

To add two floating-point numbers, the radix points must be aligned. This is typically done by shifting the input with the smaller exponent to the right to align it with the larger input. The exponent comparator circuit takes two exponents (12 bits) and give us which one is larger and which one is smaller. The exponent processor takes two exponents (12 bit) as input and calculates the difference of the exponents(12 bits). It also compares the two outputs and gives Ea¡Eb(1 bit) as a flag. It also presents the larger exponent as the tentative exponent(12 bits).

7.2 Rounding

Our output floating point representation had 20 bits for mantissa. However, during our calculations, we extended the mantissa to 32 bits. Hence we rounded the final result to 20 bits. For rounding, we evaluated the 20th bit as Guard bit, 21th as Round bit. If any of the bits on the right of Round bit is set, the sticky bit would be set. Else, it is reset. So it it the logical **OR** operation of those bits. Based on the these bits as well as the 20th bit of the 32 bit mantissa from the normalizer module, we decide whether to round up or truncate based on the following truth table. The round Up flag implies

$20thbit(P)$	G	R	S	$Decision$	$RoundUpFlag(F)$
X	0	X	X	Truncate	0
0	1	0	0	Truncate	0
1	1	X	X	Round up	1
X	1	1	X	Round up	1
X	1	X	1	Round up	1

Table 2: Truth Table for rounding

that we add 1 at the 20th bit when its set, else we truncate it. Moreover, when GRS = 100, we round to even, which implies that we add 1 if the 20th bit is set, rounding up to even. Else, we truncate it.

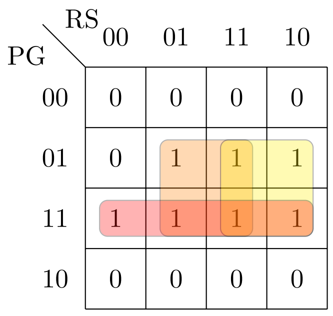


Figure 19: K-Map for Round Up Flag(F)

From the K-map we get,

$$F = G \wedge (P \vee R \vee S)$$

Also, we have to keep the mantissa normalized after rounding. If all of the 20 bits of the mantissa were to be 1, then on rounding up the floating point will be denormalized. To normalize it, we simply add one to the exponent and adjust it. Since all the bits would be reset in this case already, there is no need for right shift.

7.3 Flag unit implementation

In IEEE 754 format, range of exponent is -1022 to +1023. So, if the resultant exponent becomes less than -1022 (i.e. (exponent + bias) ¡ 1), then it is underflow condition. Such condition can arise when (i) (exponent + bias) is 00000000000 or, (ii) less than 000000000. The (i) is detected by OR-ing all the bits of exponent and (ii) is deletected by checking the borrow from the subtractor used in left shifting normalization. The underflow flag is 1 when either of these conditions are met. And for overflow, the resultant exponent needs to be greater than 1023 (i.e. (exponent + bias) ¿ 2046). Such condition can arise when (i) (exponent + bias) is 11111111111 or, (ii) greater than 11111111111. The (i) is detected by AND-ing all the bits of exponent and (ii) is deletected by checking the carry from the adder used in right shifting normalization. The overflow flag is 1 when either of these conditions is true. Also, when exponent becomes -1 (1111111111) in underflow condition, it fullfills the condition (i), even though it is not overflow. To handle this case, overflow flag is made 0 when two numbers with opposite signs are added.

8 ICs used with count as a chart

IC	Number of ICs
7404	1
744075	2
7408	4
7432	4
7486	9
Total	20

Table 3: ICs and their number

9 The Simulator Used along with the Version Number

Logisim - 2.7.1

10 Discussion

The entire project was a pedagogic experience. Several difficulties had to be faced throughout the process.

The circuit was significantly large. So, we implemented the fundamental units of the circuit as modules. Later, we assembled the entire circuit using these modules. This not only helped in the organization of the entire project but also facilitated the debugging process.

In case of debugging, we extensively used the **probe** feature of **Logisim**. Also the circuit required an enormous amount of wire connections. This meant more chances of errors in the circuit. In addition, the debugging process would be tedious. To overcome these issues, we made ingenious use of the **tunnel** feature. Tunnel connects any two points with the same name. Firstly, this made our circuit organized and well-structured. Secondly, this eased the debugging process. Lastly, all the modules became easily interpretable.

We used the default adders, subtractors, multiplexers, and shifters provided with logisim as well as ICs from **7400-lib.circ**. The modules made optimum usage of both kinds of ICs.The default circuits aided in our design so that we could focus more on the main circuit. Even so, we implemented the smaller modules at IC level using the 7400 series ICs. Hence, we were able to learn and make the best usage of the tools available.In the end, we implemented the circuit judiciously.

11 Contribution of Each Member

- 2005020 : Mostafa Rifat Tazwar
  - Circuit Design
    - Arbitrary Right shifter
    - Mantissa Pre and Post processor and debugging
    - Rounding Unit
  - Report Writting
    - Rounding

    - Discussion
    - Problem Specification
    - Circuit Diagram
    - Complete Report Organization

- 2005025 - Most. Sonia Khatun
  - Circuit Design
    - Left Shift normalization for subtraction
    - Right Shift normalization for addition
- 2005027 - Swastika Pandit
  - Circuit Design
    - Mantissa processing
    - Flag unit
    - Connecting the components to form complete circuit
    - Circuit testing and debugging.
  - Report Writting
    - High level block diagram
    - Flag Unit
- 2005029 - MD. Minhajul Islam Fuad
  - Circuit Design
    - Subtractor
- 2005030 - Fairuz Mubashwera
  - Circuit Design
    - Selecting the optimized choice for comparing/subtracting exponents
    - Building Exponent Processor and exponent comparator
    - Debugging and testing the circuit
  - Report Writting
    - Introduction
    - Flow Diagram