Bangladesh University of Engineering and Technology

CSE306

Computer Architecture Sessional

Assignment - 2

Implementation of 16 bit Floating Point Adder

Submitted By:

Group: A1 - 1

Kowshic Roy
(S201705001)
Sheikh Azizul Hakim
(S201705002)
Mahdi Hasnat Siyam
(S201705003)
Saiful Islam
(S201705006)
Khandokar Md. Rahat Hossain
(S201705024)

Department of CSE

Submitted To:

Abu Wasif
Assistant Professor

Madhusudan Basak
Assistant Professor

T. M. Tariq Adnan
Lecturer

Department of CSE

Date of Submission : May 30, 2021

1 Introduction

Floating point adder is a combinatorial circuit that performs the addition operation of two signed floating point numbers. In this assignment, we had to design a 16-bit floating point adder circuit which takes two 16-bit floating point numbers as inputs and provides their sum, another 16-bit floating point number, as the output.

2 Problem Specification

Each 16-bit number would be represented as:

Sign 1-bit	Exponent 4-bit				Fraction 11-bit										
s	e_3	e_2	e_1	e_0	f_{10}	f_9	f_8	f_7	f_6	f_5	f_4	f_3	f_2	f_1	f_0
S	E				F										

Table 1: 16 bit representation of numbers

Here we assume that the numbers are in normalized form and their exponents are biased with $B = 2^{4-1} - 1 = 7$. Hence, a number N having representation SEF is equal to

$$N = (-1)^S \times 1.F \times 2^{E-7}$$

Let the two numbers be denoted by $N_a = S_a E_a F_a$ and $N_b = S_b E_b F_b$. The task is to do the summation and output the result in truncated normalized form in the same 16 bit representation($S_r E_r F_r$). IEEE754 standard has some special values for representing infinity, sub-normalized numbers. These were also taken care of. Again, the circuit has underflow and overflow flags to indicate underflow or overflow occurrence respectively.

3 Methodology

The workflow of the FP adder can be divided into several segments which are mentioned below :

1. **Sorting and Reconstructing the numbers**

   If $E_a < E_b$, we swapped the numbers so the N_a always carries the maximum exponent value.
   Here $E_r = E_a$ is our initial resulting exponent which may or may not be changed later.

   Reconstruction method: The implicit 1 (left of radix point) in our representation was introduced before the 11 bit fraction part($1F$) which is our mantissa(M). Now our mantissa is of 12 bits. To make it of 16 bits, we introduced two zeroes before and after the mantissa M. So, finally $M = 001F00$.

   Two mantissas M_a and M_b were reconstructed using *Reconstruction* method mentioned above.

2. **Making the exponents equal**

   We right shifted the mantissa M_b by $(E_a - E_b)$.

3. Adding the Mantissa's

We calculated the resulting Mantissa as well as resulting sign by passing two Mantissas M_a and M_b along with their signs S_a and S_b to a 16 bit modified ALU.

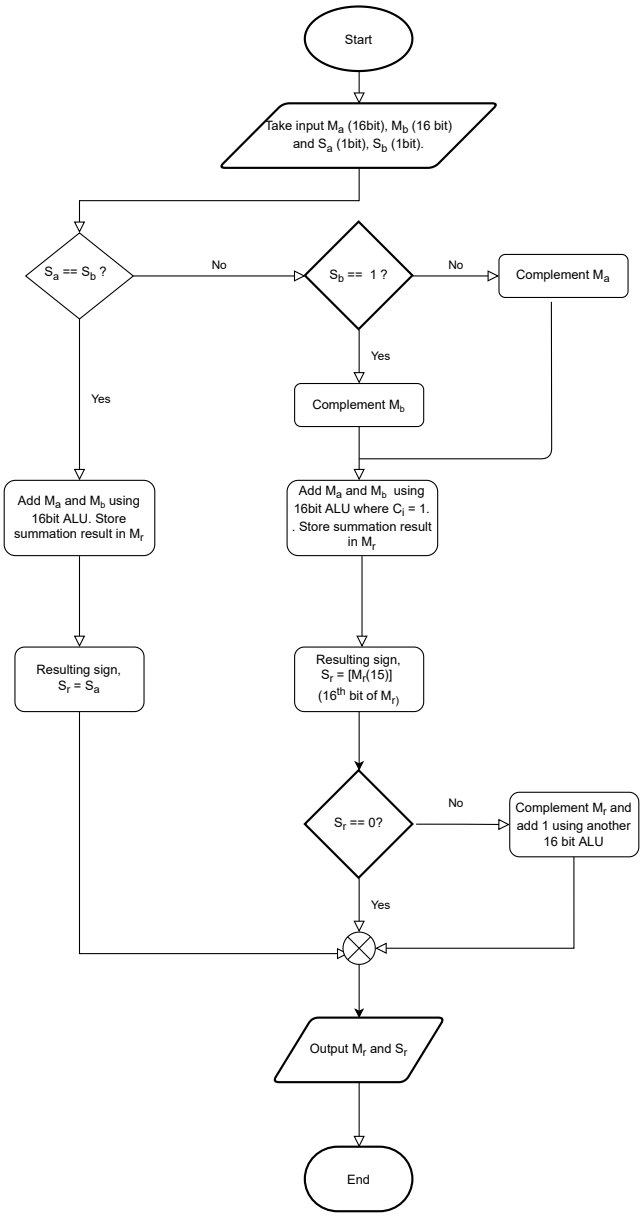

Figure 1: Flowchart of the Modified ALU

4. Normalizing the resulting Mantissa

We normalized M_r and adjusted E_r if necessary.We also checked whether overflow or underflow had occurred.

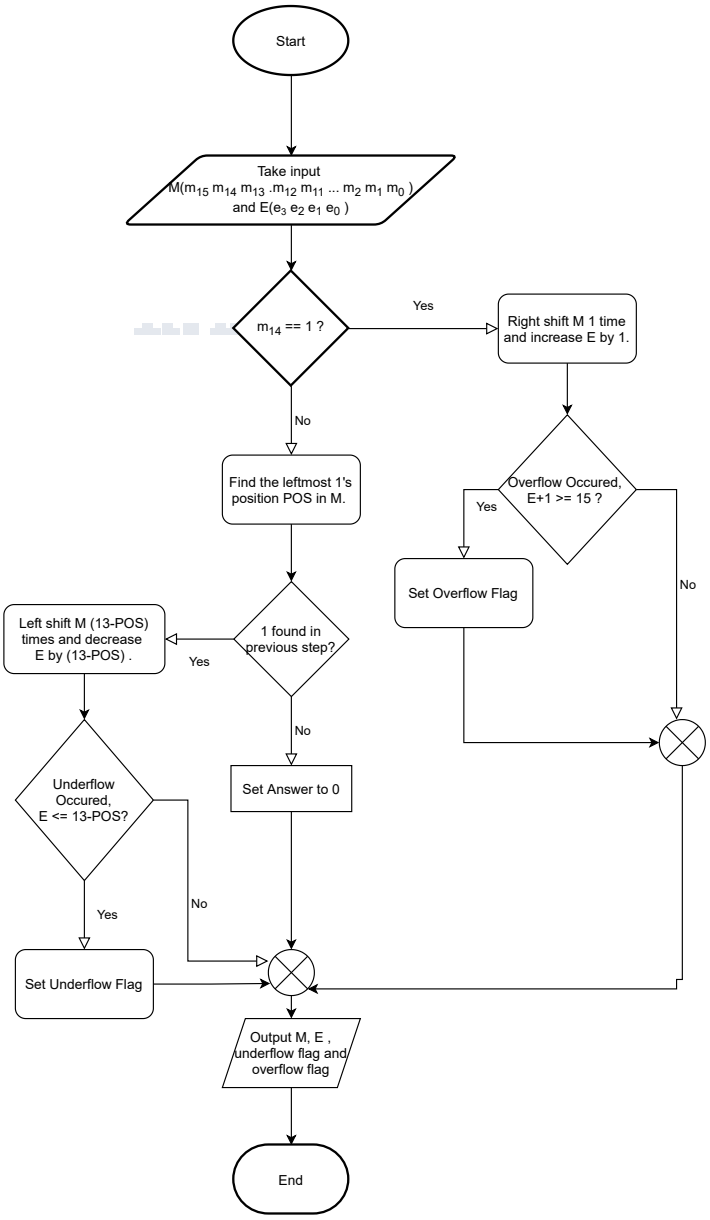

Figure 2: Flowchart of the Normalizer

5. Output

In case of overflow or underflow, corresponding flag bits were set.
In other cases,

sign bit as S_r,
exponent as E_r,
fraction as $m_{12}m_{11}m_{10}\ldots m_4m_3m_2$ of M_r.

Overall Logic Flow Diagram :

Here we present a block diagram of the whole process. Abstraction for modified ALU and normalizer were used to keep things simple.

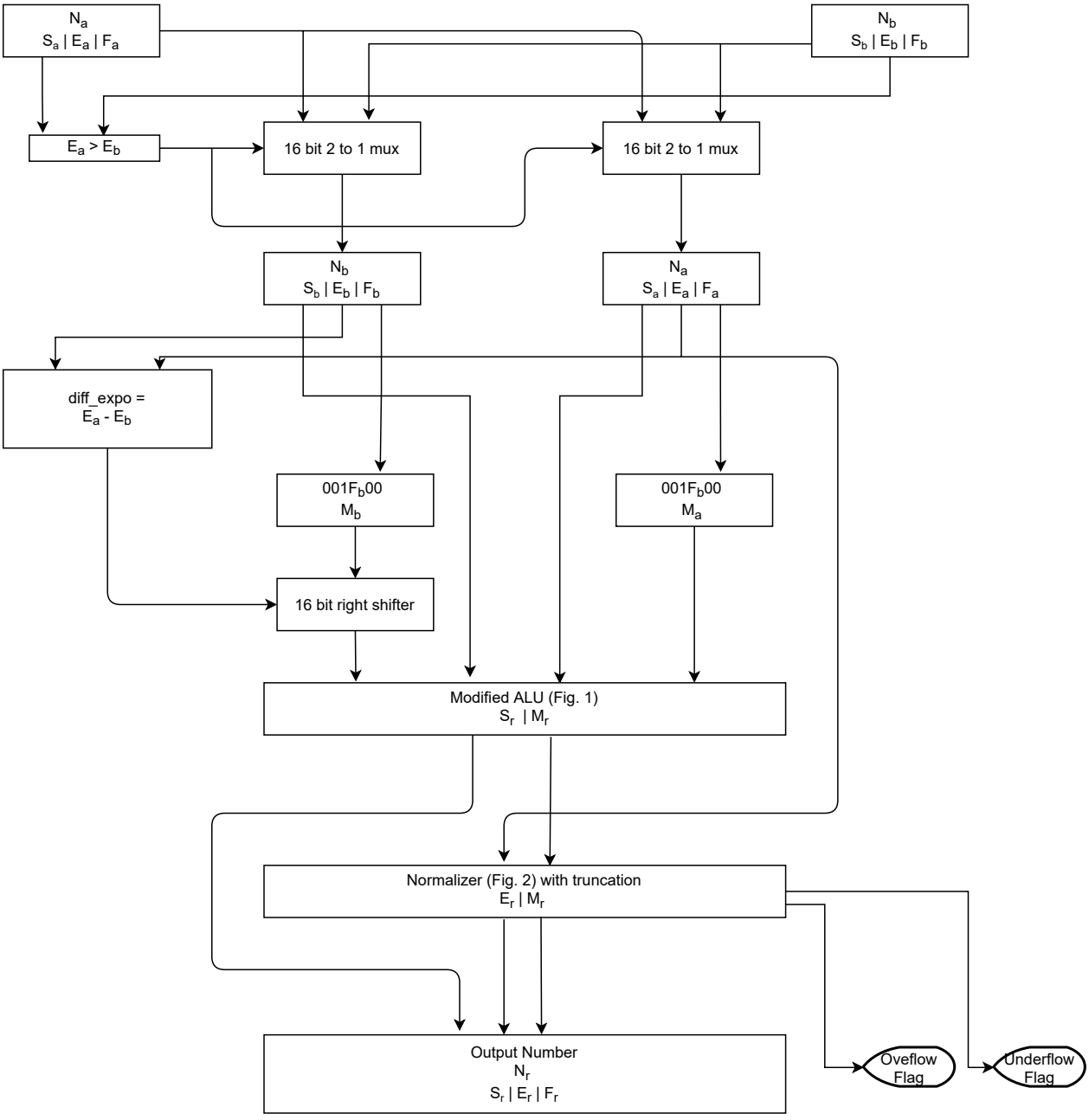

Figure 3: High level view of implementation of the FP Adder

4 Implementation

4.1 Logic Gate Diagrams

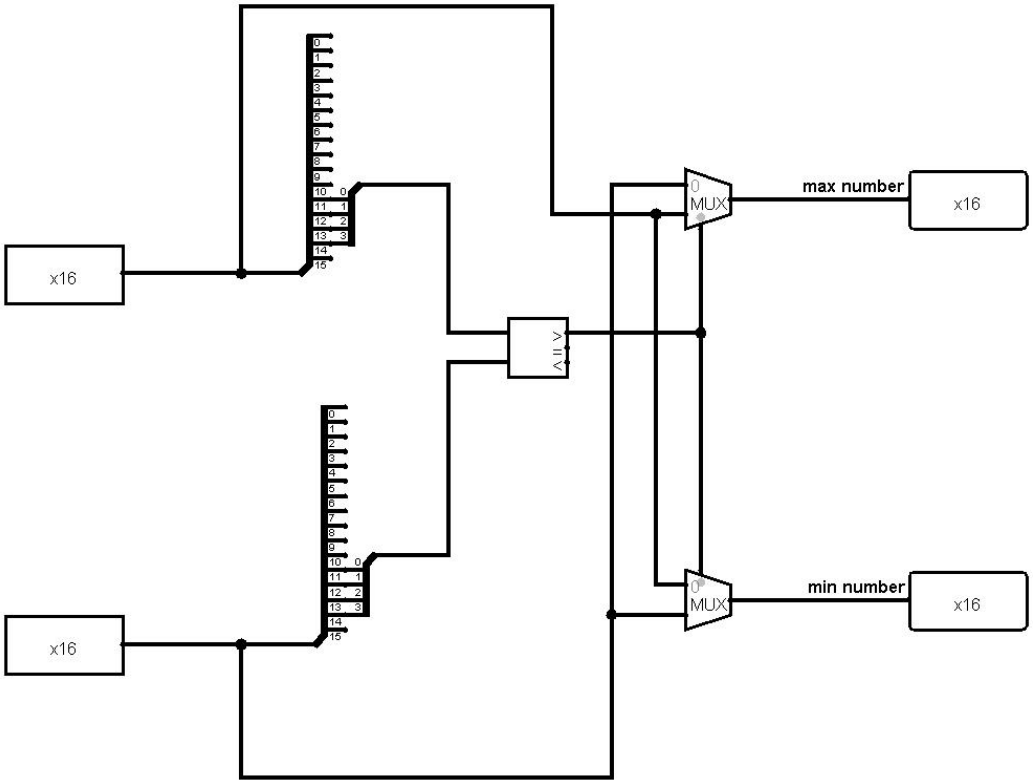

Figure 4: Gate Diagram of the input sorter

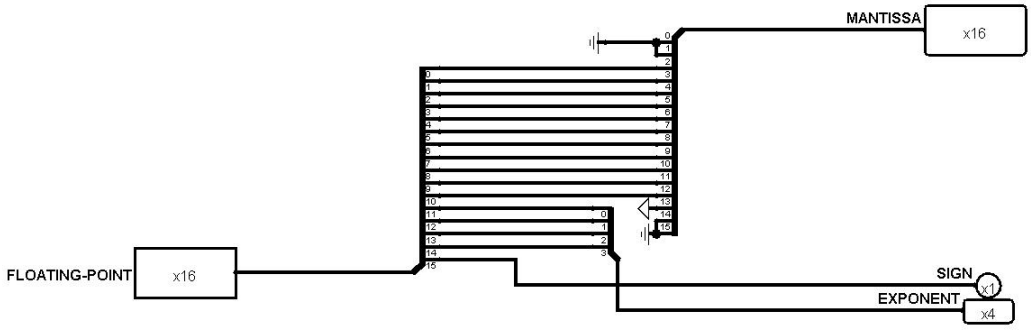

Figure 5: Gate Diagram of the Floating point Decoder

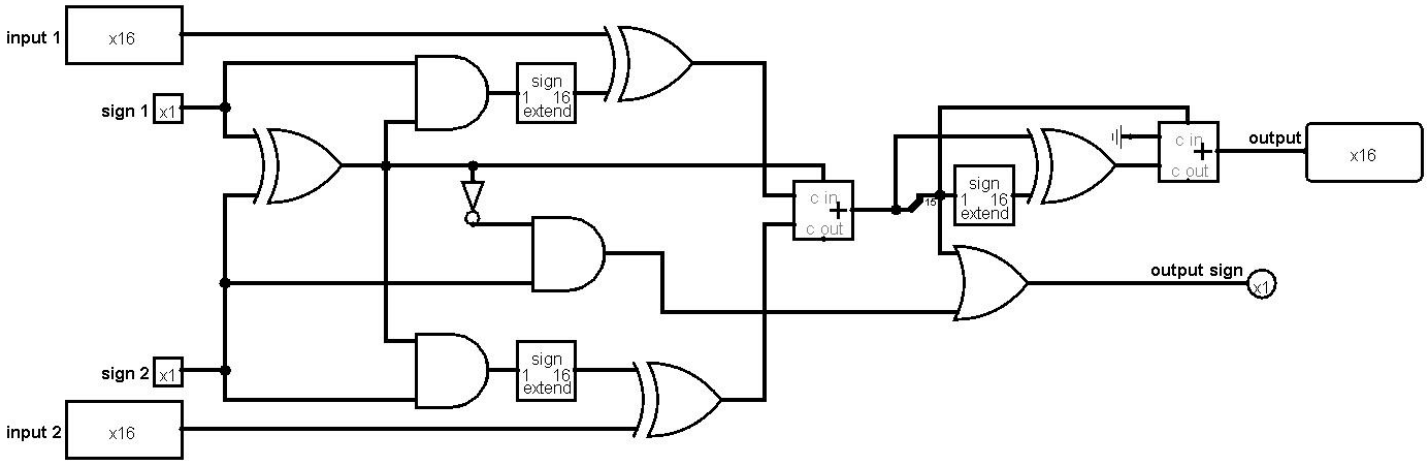

Figure 6: Gate Diagram of the Modified ALU

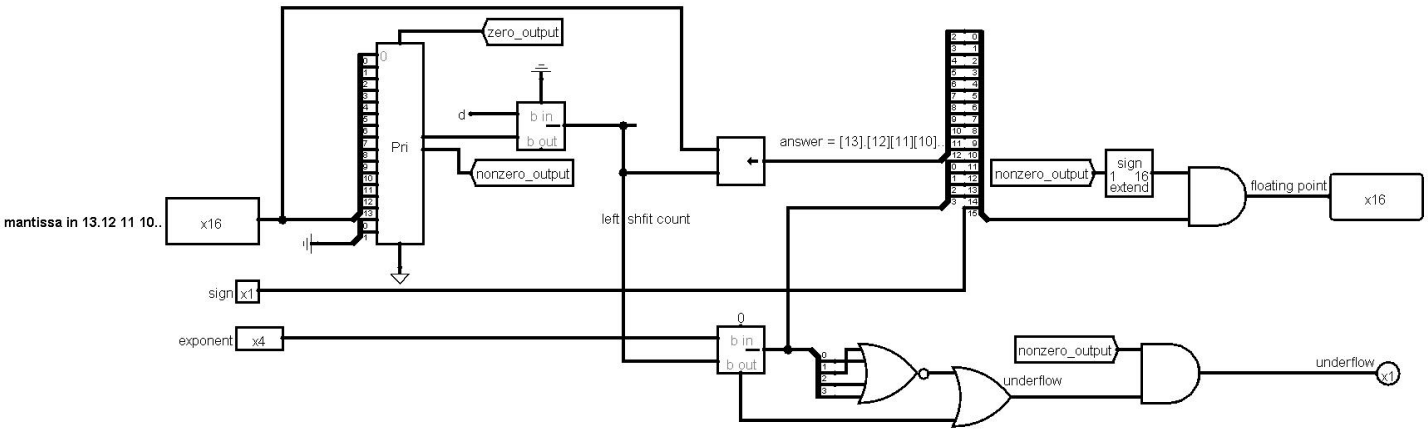

Figure 7: Gate Diagram of the Normalizer

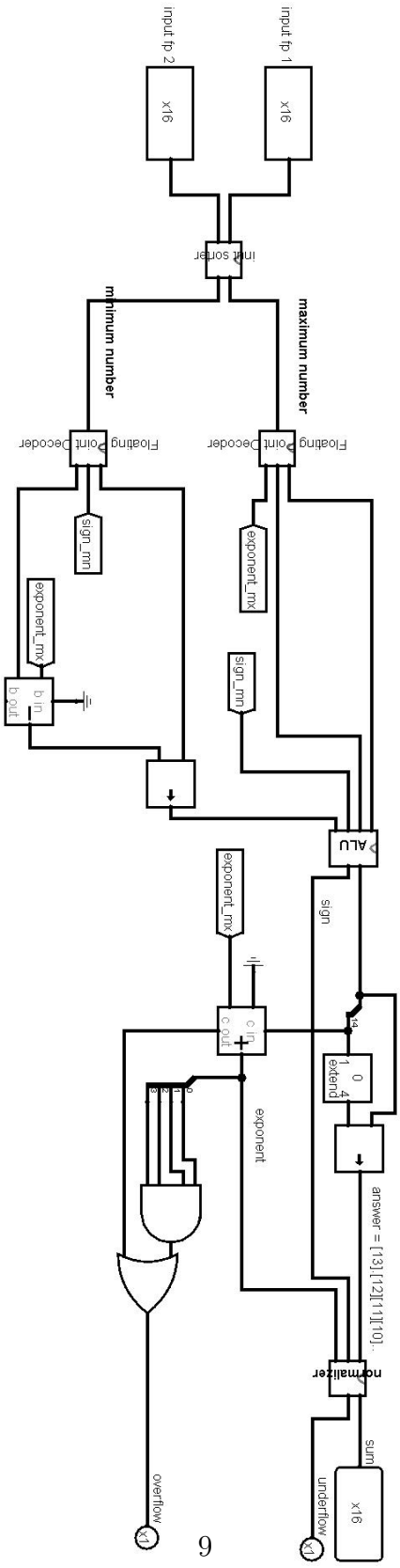

Figure 8: Gate Diagram of the FP Adder

4.2 IC Diagrams

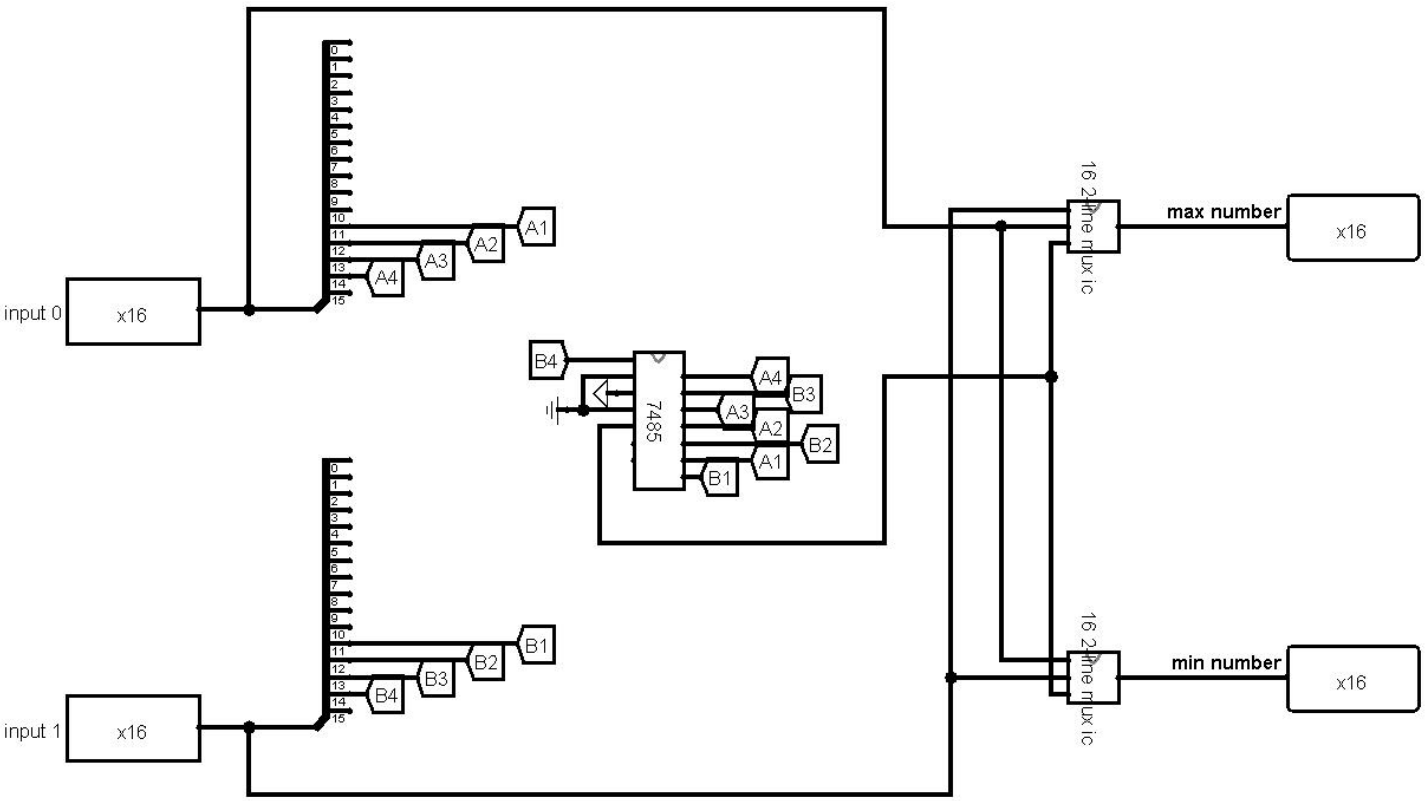

Figure 9: IC Diagram of the Input Sorter

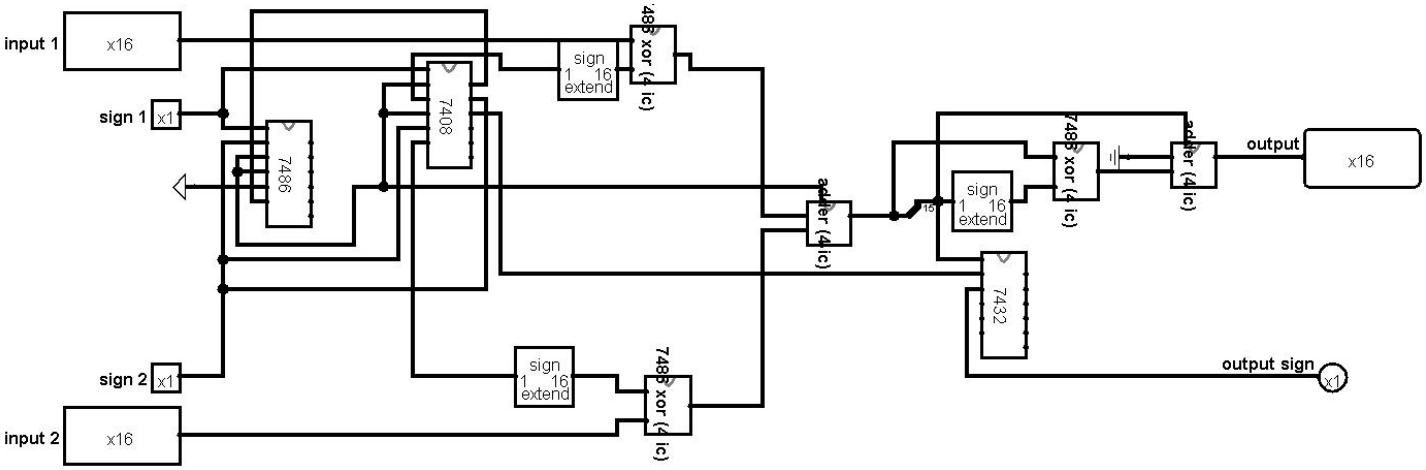

Figure 10: IC Diagram of the Modified ALU

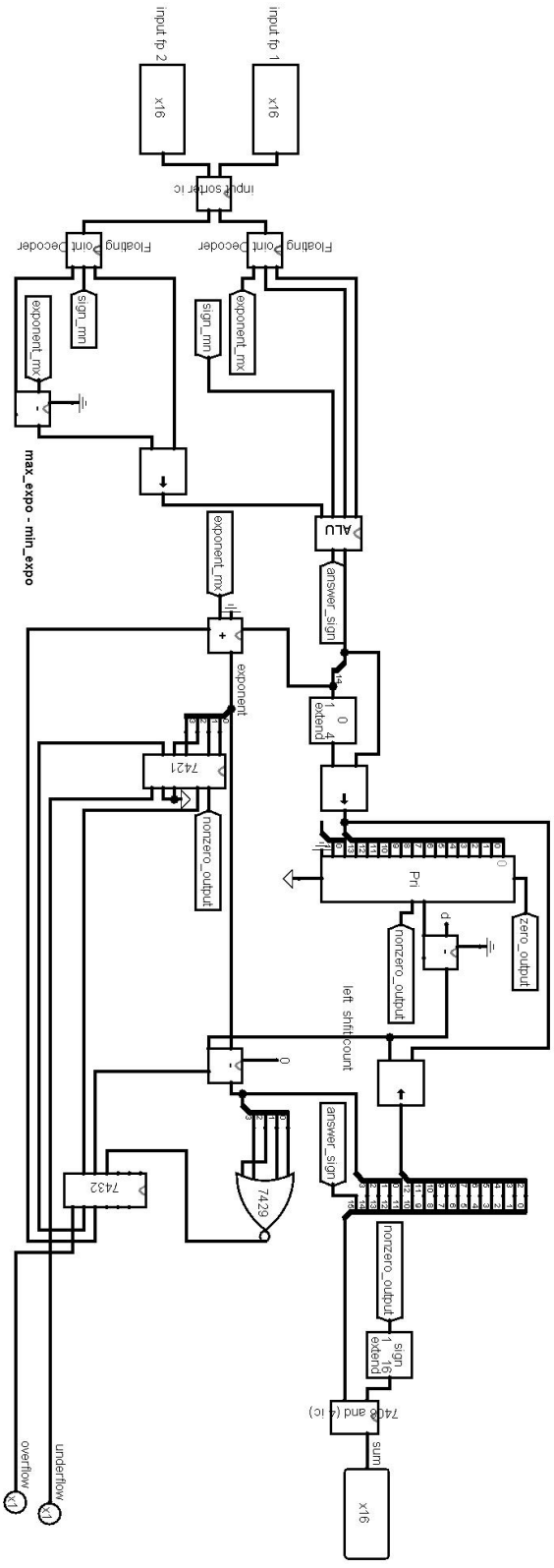

Figure 11: IC Diagram of the FP Adder

5 Block Diagram

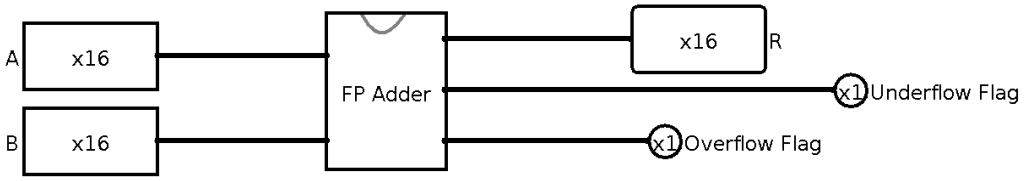

Figure 12: Block Diagram of Floating Point Adder

6 IC count

Part Number	Description	Count
7485	4-bit Magnitude Comparator	1
74157	Quad 2-Input Multiplexer	8
74181	4-bit Slice Arithmetic Logic Unit (ALU)	3
74283	4-Bit Binary Adder	9
7486	Quad 2-Input XOR Gate	13
7408	Quad 2-Input AND Gate	5
7421	Dual 4-Input AND Gate	1
7432	Quad 2-Input OR Gate	1
7429	Dual 4-Input NOR Gate	1
	16-Line to 4-Line Priority Encoder	1
	16 bit Logical Shifter	3
Total		46

Table 2: List of ICs used in the simulation

7 Simulator Details

In this assignment we used **Logisim** v2.7.1, which is an open-source software for simulating digital circuits. More information about this software can be found at: `http://www.cburch.com/logisim/index.html`

8 Discussion

In this experiment, we designed a floating point adder to facilitate a subset of operations supported by IEEE754. Our circuit is only valid for normalized inputs but it can be easily extended for sub-normalized numbers by simply adding two more 4-input OR gate (one 744072 IC unit).

The focus of our design was on the efficiency of design and minimization of ICs used.For example, two "8-line to 3-line priority encoder (IC 74148)" can be concatenated to make"16-line to 4-line priority encoder". We can also implement the 16-bit logical shifter with 16 "16-line to 1-line multiplexer". But we used logisim's default shifter and priority encoder for the simplicity of the design.