CSE306 (Computer Architecture Sessional)

A Report On

Floating Point Adder

Contributor Student ID:

1705068
1705069
1705070
1705071
1705074

Section: B
Subsection: **B1**
Group NO: 3

Introduction:

Floating point is a representation of non-integral numbers. It is also called Fixed Point Binary. Like scientific notation, numbers are represented as a single nonzero digit to the left of the floating-point.In normalized form, the representation consists of sign, exponent and fraction fields such that:

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

To keep a binary number in normalized form, we need a base that we can increase or decrease by precisely the number of bits. The number must be shifted to have one nonzero digit to the left of the binary point.

We have cascaded four 4 bit ALU and used a shifter circuit to implement this adder circuit.

Problem Specification:

In this assignment, we design a floating-point adder circuit that takes two 16 bits long floating points as inputs and provides their sum, another 16 bits long floating point as output.

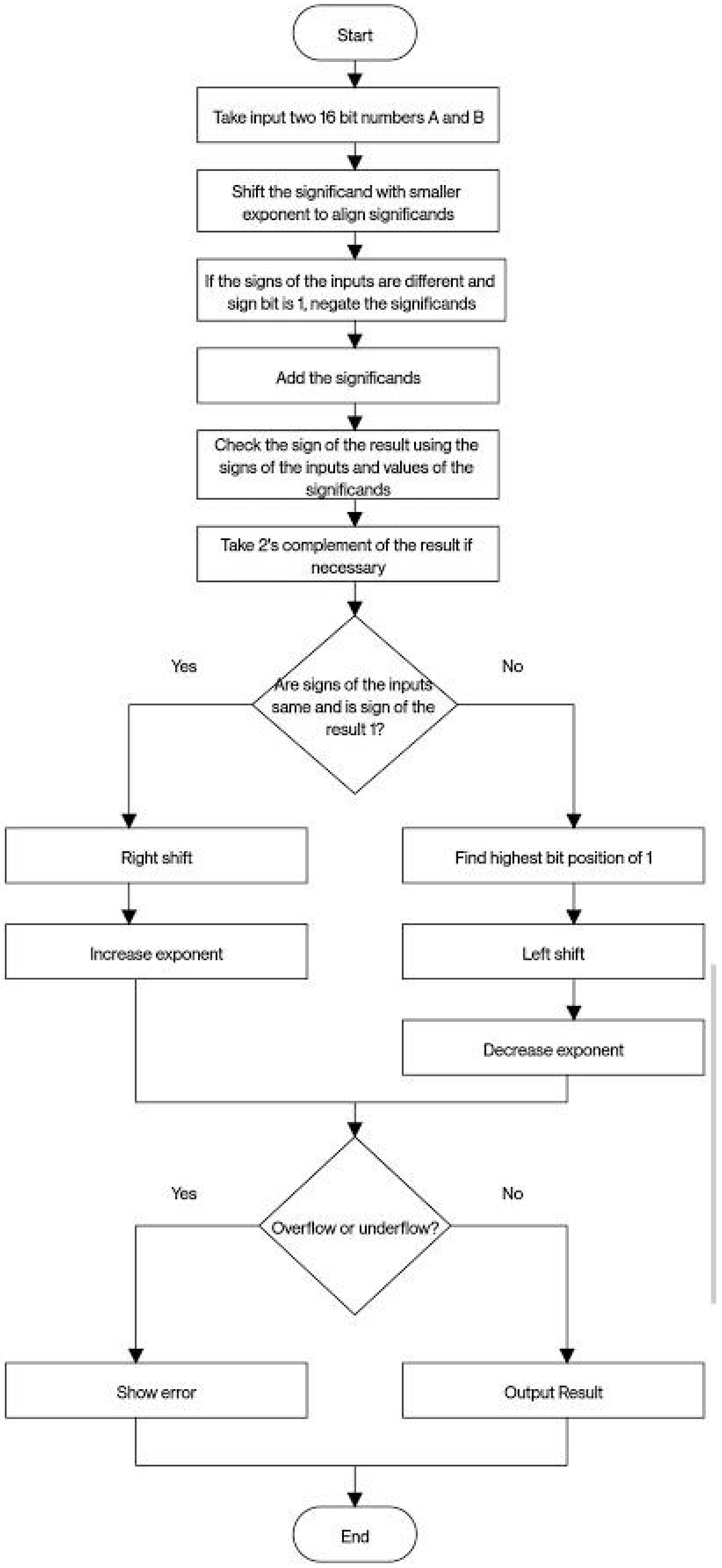
Required Flags:
- Overflow
- Underflow

We are to implement rounding also.The floating point numbers are given in the following form:

Sign **1 bit**	Exponent **4 bit**	Fraction **11 bit**

Figure: 1 bit representation

Flowchart:


Start
Take input two 16 bit numbers A and B
Shift the significand with smaller exponent to align significands
If the signs of the inputs are different and sign bit is 1, negate the significands
Add the significands
Check the sign of the result using the signs of the inputs and values of the significands
Take 2's complement of the result if necessary
Are signs of the inputs same and is sign of the result 1?
Yes
No
Right shift
Increase exponent
Find highest bit position of 1
Left shift
Decrease exponent
Overflow or underflow?
Yes
No
Show error
Output Result
End

Block Diagram:

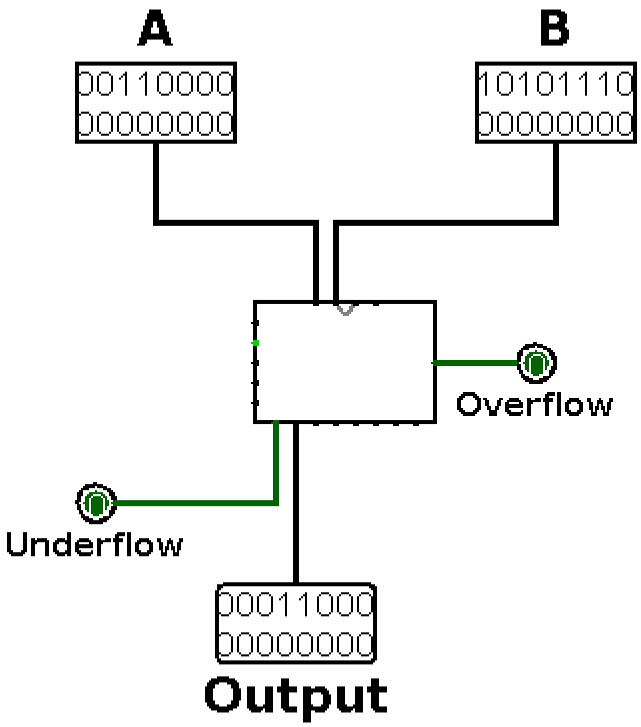


Figure 1: Block Diagram

Circuit Diagram:

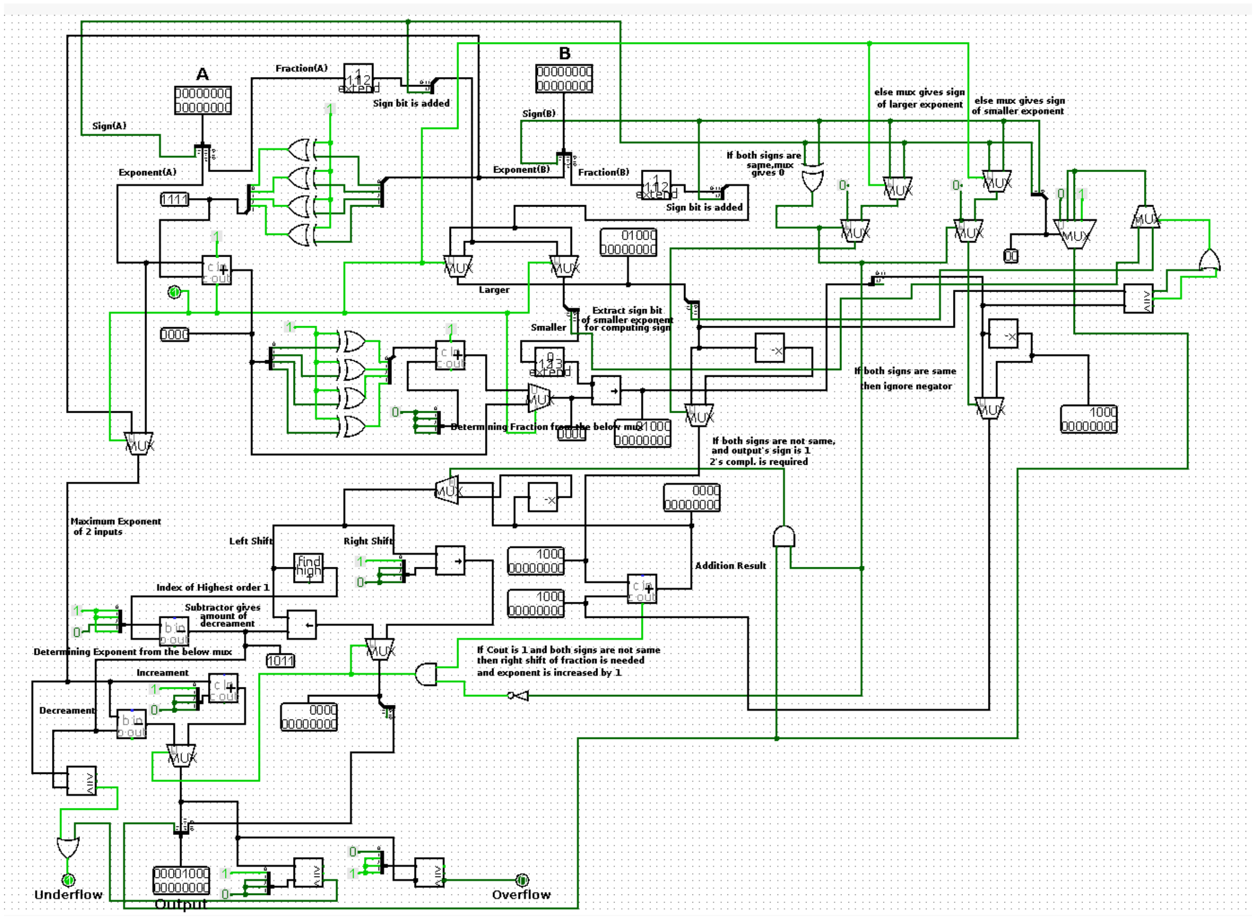

A
B
Sign(A)
Exponent(A)
Fraction(A)
Sign bit is added
Sign(B)
Exponent(B)
Fraction(B)
Sign bit is added
else mux gives sign of larger exponent
else mux gives sign of smaller exponent
If both signs are same, mux gives 0
Larger
Smaller
Extract sign bit of smaller exponent for computing sign
If both signs are same then ignore negator
Determining Fraction from the below mux
If both signs are not same, and output's sign is 1 2's compl. is required
Maximum Exponent of 2 inputs
Left Shift
Right Shift
Index of Highest order 1
Subtractor gives amount of decreament
Determining Exponent from the below mux
Addition Result
If Cout is 1 and both signs are not same then right shift of fraction is needed and exponent is increased by 1
Increament
Decreament
Underflow
Output
Overflow

Figure 2: Floating Point Adder

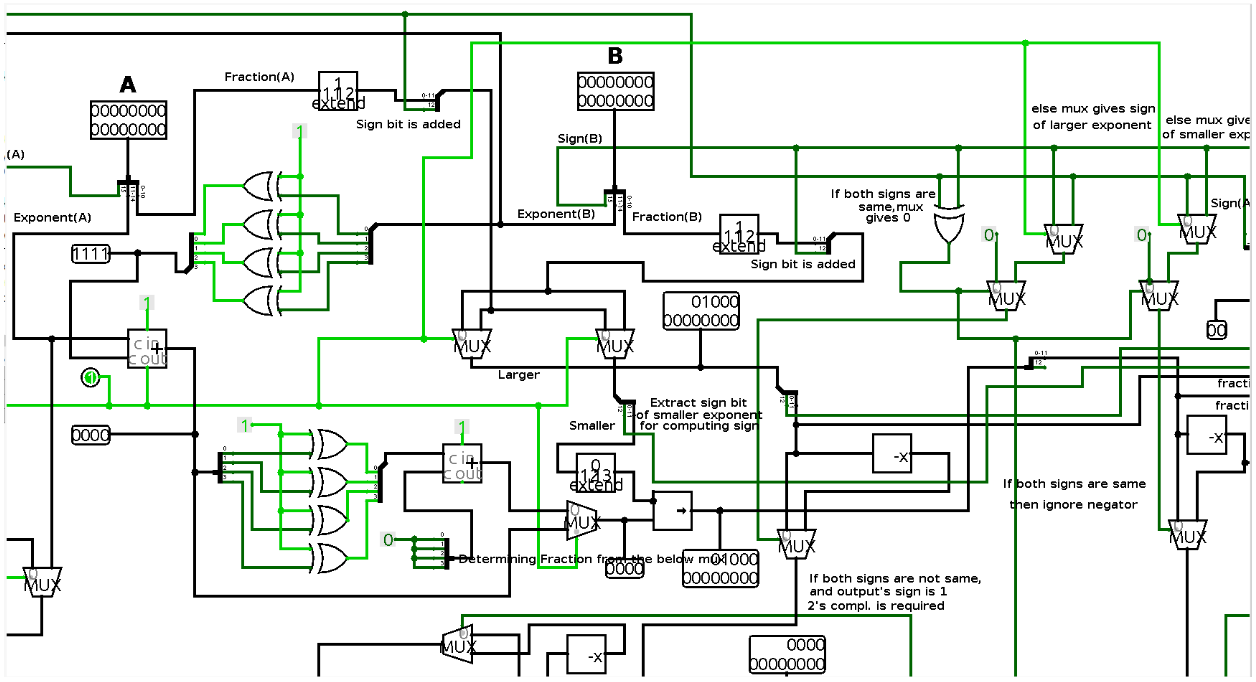

A
B
Fraction(A)
Sign bit is added
Exponent(A)
Sign(B)
Exponent(B)
Fraction(B)
Sign bit is added
Larger
Smaller
Extract sign bit of smaller exponent for computing sign
Determining Fraction from the below mux
If both signs are same,mux gives 0
else mux gives sign of larger exponent
If both signs are same then ignore negator
If both signs are not same, and output's sign is 1 2's compl. is required
MUX
extend

Figure 3: Align Block

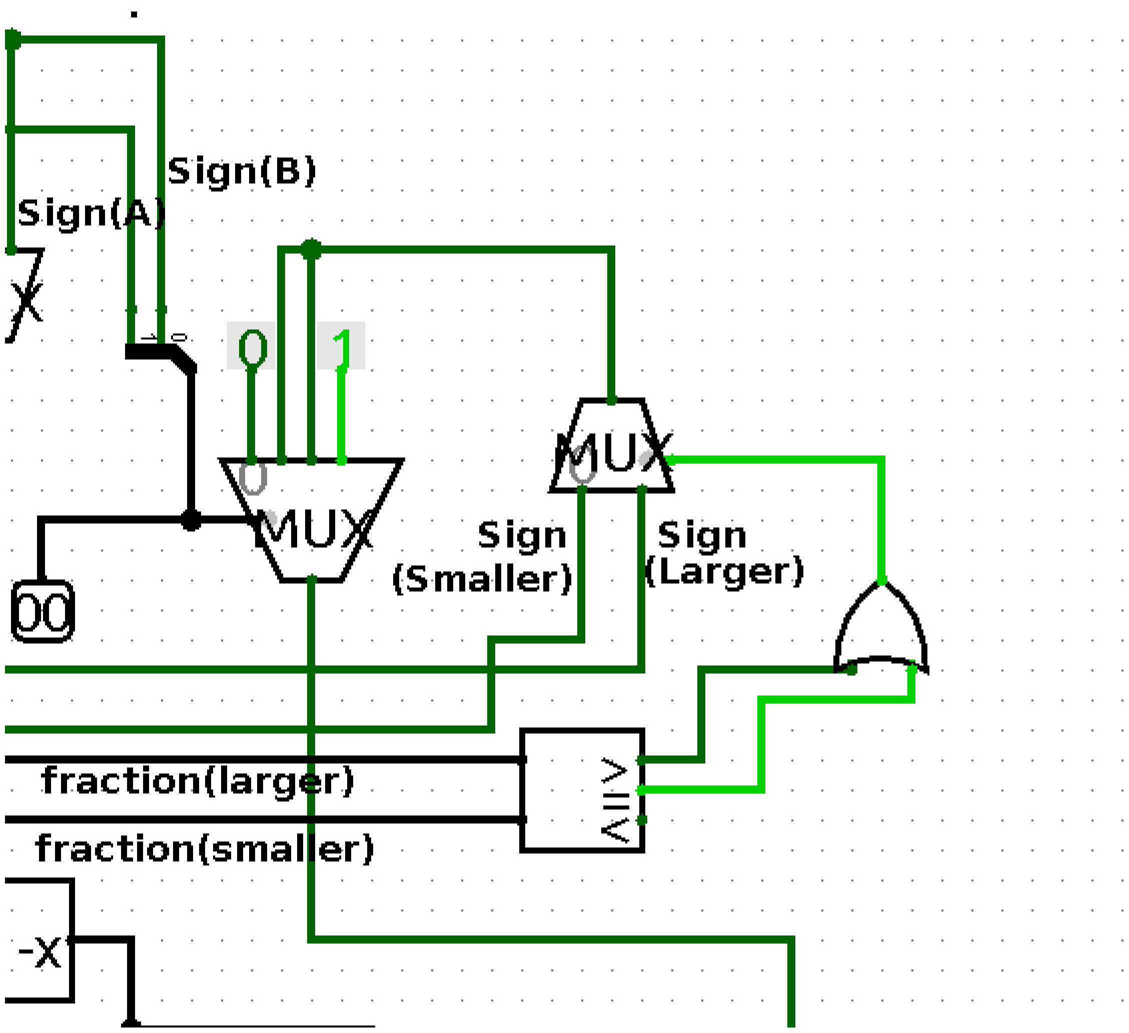


Figure 4: Sign Computing Block

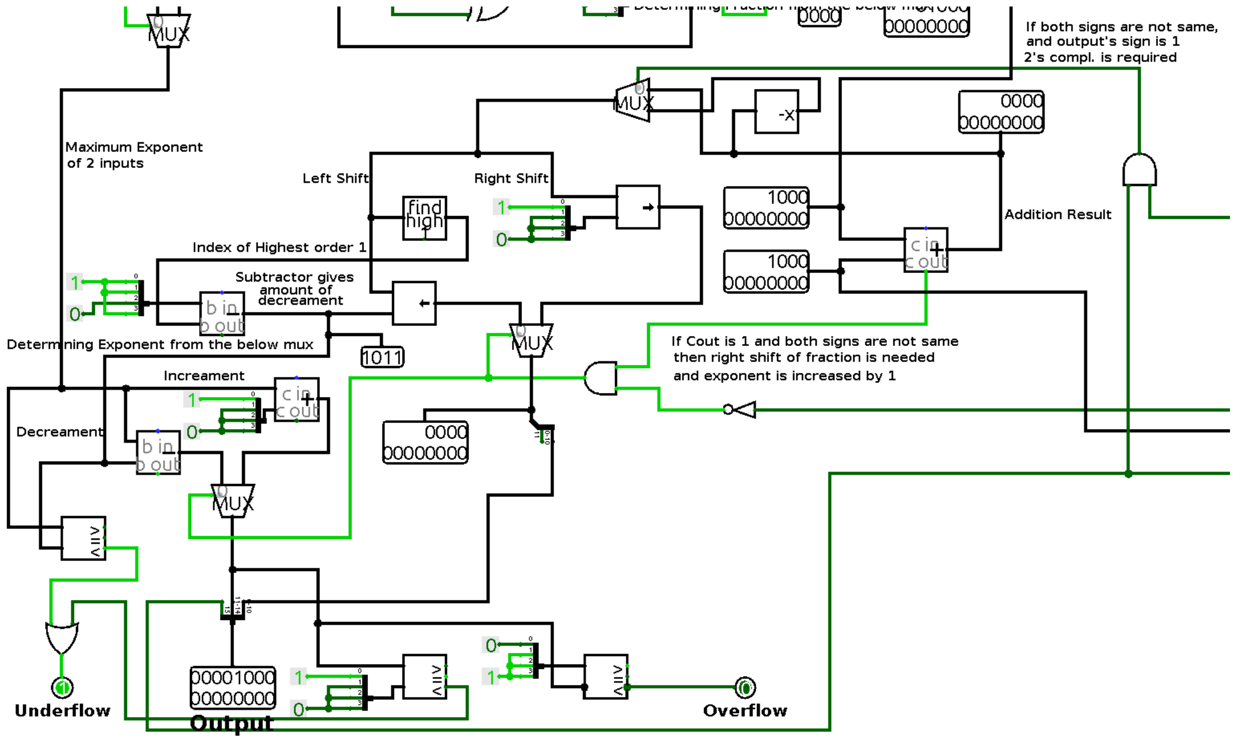

If both signs are not same, and output's sign is 1 2's compl. is required
MUX
-x
0000 00000000
Maximum Exponent of 2 inputs
Left Shift
Right Shift
find high
1000 00000000
1000 00000000
Addition Result
Index of Highest order 1
Subtractor gives amount of decreament
b in b out
c in c out
Determining Exponent from the below mux
1011
If Cout is 1 and both signs are not same then right shift of fraction is needed and exponent is increased by 1
Increament
Decreament
0000 00000000
00001000 00000000
Output
Underflow
Overflow

Figure 5: Normalizer and Round Block

IC Used with Count:

IC	Name	Count
7404	Hex Inverter	1
7408	Quad 2-AND Gate	1
7432	Quad 2-OR Gate	1
7486	Quad 2-XOR Gate	3
7483	4 bit Full Adder	8
74157	Quad 2 to 1 MUX	23
74153	Dual 4 to 1 MUX	1
		Total=38

Component:

Component Name	Count
Comparator	4
Shifter	3
Negator	3
Bit Finder	1
Bit Extender	3
Splitter	20

Simulator:

Logisim 2.7.1

Discussion:

In this assignment, we have implemented a floating-point adder circuit. As the representation of the floating-point number was in Standard IEEE 754 format so, to implement the circuit, we used the well-known "comparison of exponent and shifting based algorithm," and output was shown in normalized form. Also, rounding was done, and overflow and underflow were detected and displayed as output when it was necessary. The output is normalized and truncated.

While implementing the circuit, a minimum number of ICs were used. We used and gates, or gates, not gates, 4 bit full adders, 2-to-1 and 4-to-1 multiplexers and components such as comparators, shifters,negators, bit finder, bit extender and splitters. The circuit has been designed in such a way that a minimum number of gates is required.

After the implementation of the circuit, we verified it with some test cases. The testing process is slightly more tedious than the previous assignment as this implementation demanded more complacy than the previous one. Overall, we tried to make the design as efficient as possible.