CSE306: Computer Architecture Sessional

**ASSIGNMENT 2: FLOATING POINT ADDER**

# SUBMITTED BY:

Group No : 03

Sub-section: A1

Group Members:

1. 1705007
2. 1705011
3. 1705013
4. 1705023
5. 1705025

Department: Computer Science & Engineering

Level: 3; Term: 1

Submission Date: June 2021.

**Problem Specification:** In this assignment, we have to design a floating point adder circuit which takes two floating point numbers as inputs and output their sum which is another floating point number.

**Introduction:** A Floating Point Adder is a digital circuit used to add two floating point numbers presented in IEEE 754 Standard format of their binary representation.

**Floating Point Representation:** A designer of a floating-point representation must find a compromise between the size of the fraction and the size of the exponent. Floating-point numbers are usually a multiple of the size of a word.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Sign | Exponent | | | | Fraction | | | | | | | | | | |

In general, floating-point numbers are of the form (-1)s× F × 2E.

Here, F means the value in the fraction field and E involves the value in the exponent field. MIPS does something slightly more sophisticated.

According to IEEE 754 floating-point standard, the floating point numbers are of the form

(1)S × (1 + Fraction) × 2E.

IEEE 754 uses a bias of 127 for single precision, so an exponent of -1 is represented by the bit pattern of the value -1+127ten , or 126ten = 0111 1110two , and +1 is represented by 1 + 127, or 128ten = 1000 0000two . The exponent bias for double precision is 1023. Biased exponent means that the value represented by a floating-point number is really

(-1)s× (1+Fraction) × 2(Exponent-Bias)

**Floating Point Adder Algorithm:**

Given inputs are A(S1(1 bit), E1(4bits), F1(11bits)) and B(S2(1 bit), E2(4bits), F2(11bits)).

1. As the given numbers are normalized, we first compare the exponent and fraction of the two given numbers using the component ***Comparator.***
2. ***Comparator*** gives us the exponent difference and also tells us which number is bigger. Using the exponent difference we align the two numbers using ***Shifter*** (left or right).\
3. If the two given numbers has different sign then we uses the ***Bit-inverter*** to alter the negative number to positive.
4. Finally, we add the two numbers using two built-in adder.
5. Now, after adding the numbers the output may be denormalized. We used ***Normalizer*** to normalize the output. We normalize the fraction by finding the most significant set-bit and change the exponent accordingly.
6. We also set the final sign of the addition using the sign bit of the numbers and the output we get from the ***comparator*** about which one is bigger.
7. After normalizing, if the exponent cannot be expressed by 4 bits or if it becomes less than zero, we turn on the overflow/underflow flag and finally truncate the output.

End

Truncate the output

Turn on Overflow or

Underflow flag

Align the significands according to the maximum exponents

Overflow or Underflow

Set the sign of the final output

Normalize the sum by shifting and increasing or decreasing exponent

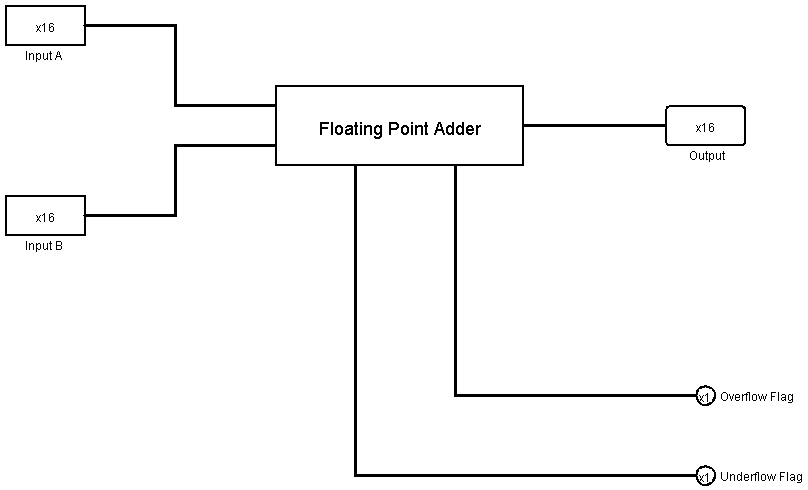
Add the significands

Start

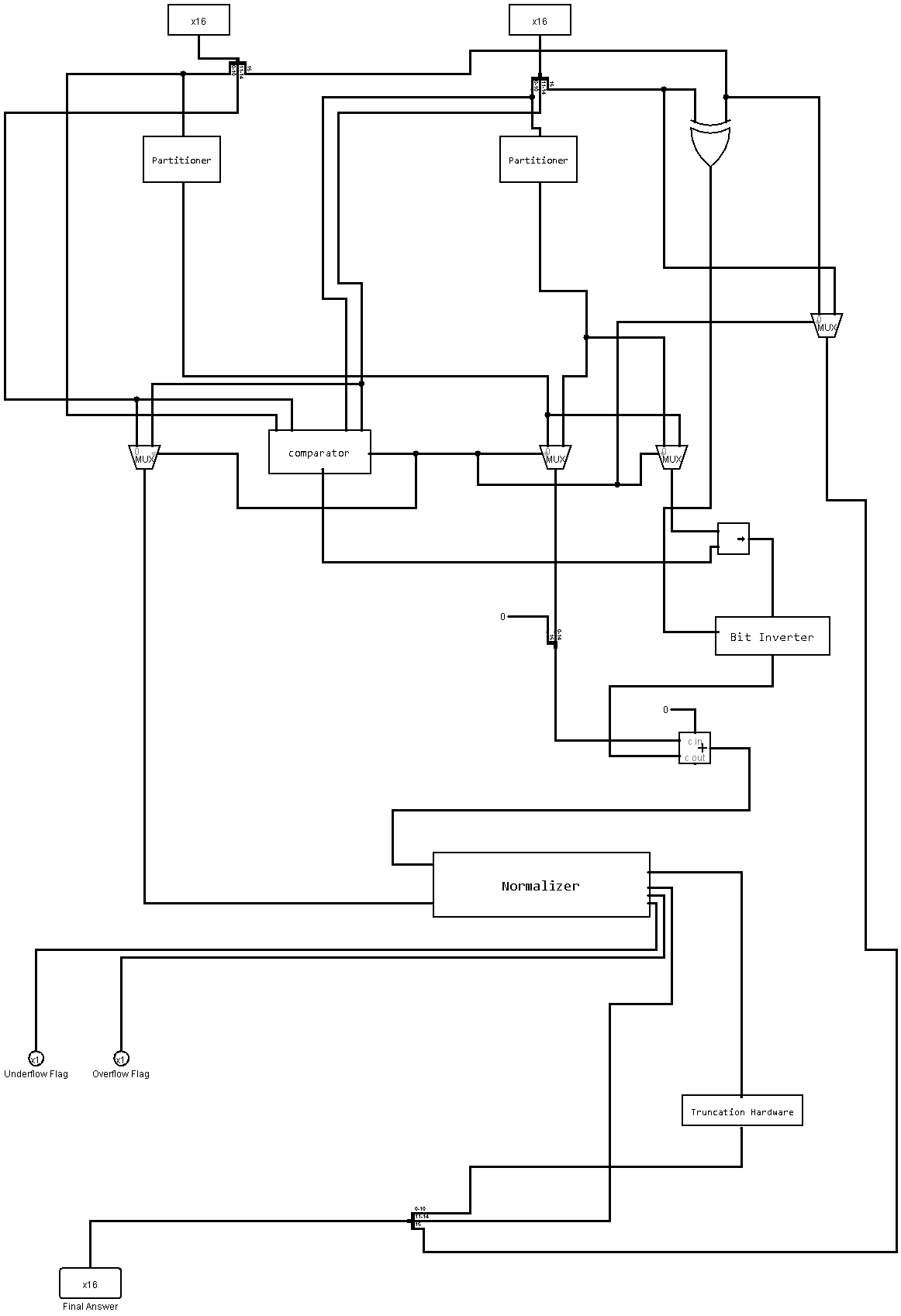
**IC Used With Count:**

|  |  |
| --- | --- |
| IC Number/Name | Count |
| 7408(AND) | 1 |
| 7486(X-OR) | 1 |
| 7404(OR) | 1 |
| 7432(NOT) | 1` |
| 7483(Full Adder) | 1 |
| 2 1 MUX | 10 |
| Full Subtractor (4 bit) | 4 |
| Full Subtractor (5 bit) | 1 |
| Comparator (5 bit) | 1 |
| Negator (4 bit) | 1 |

**Block Diagram:**



**Circuit Diagram:**



**Circuit Diagram of Inner Components:**

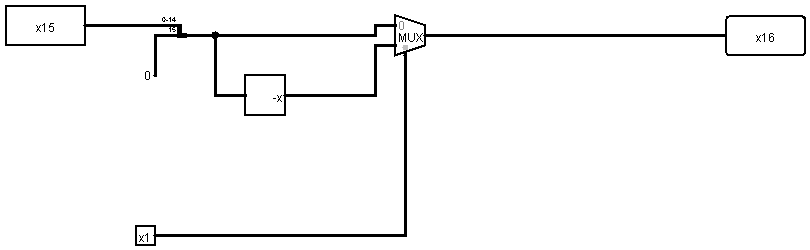


Figure :Bit-inverter

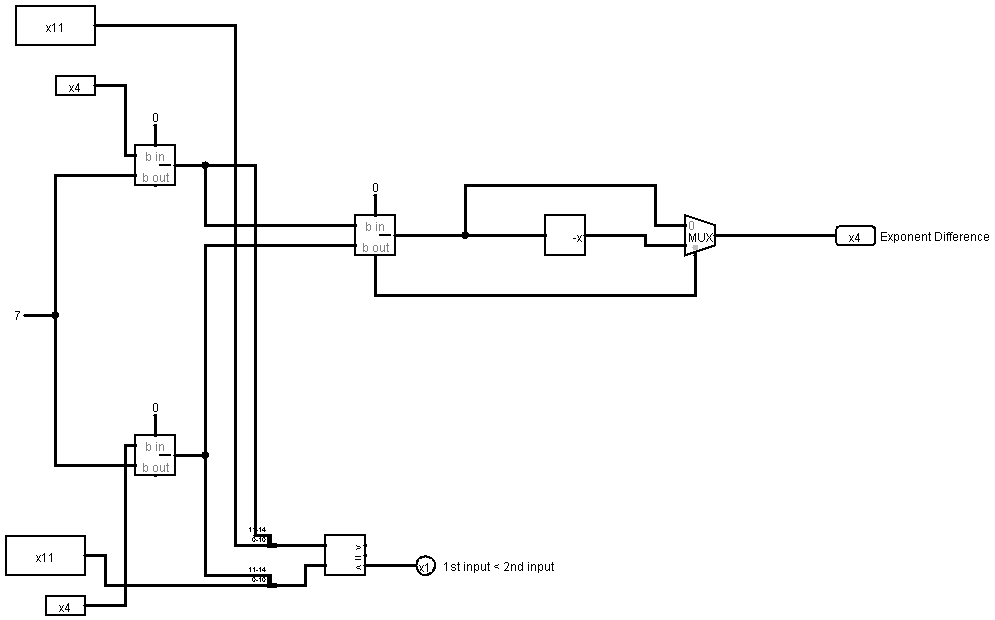


Figure : Comparator

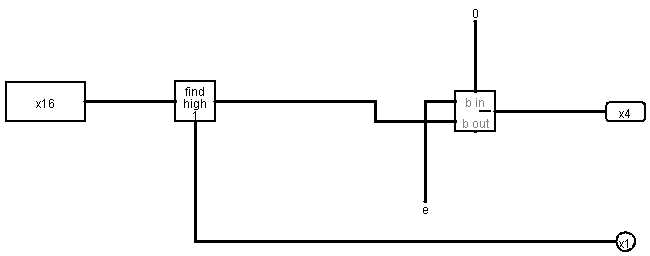
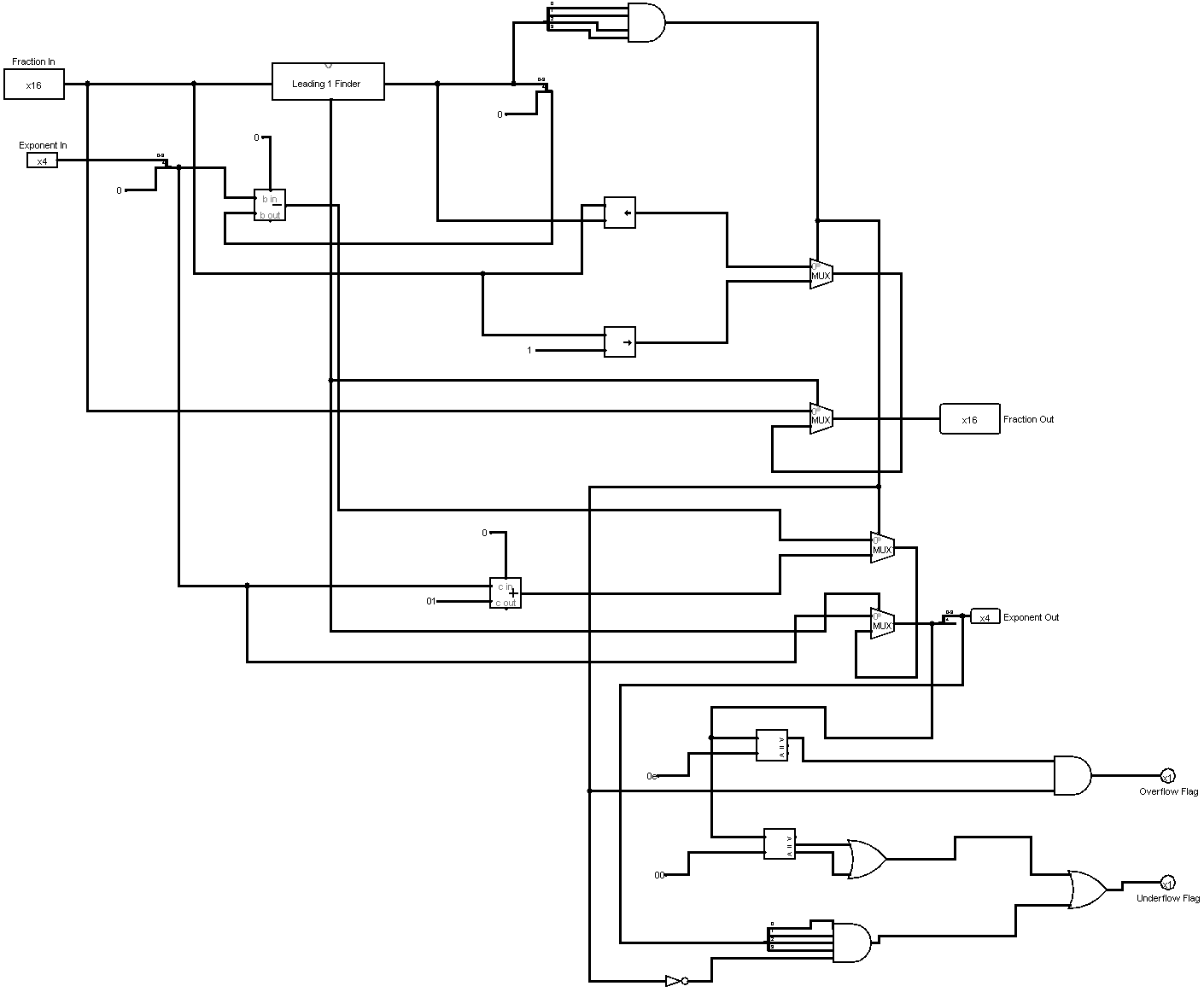


Figure : Leading1-finder



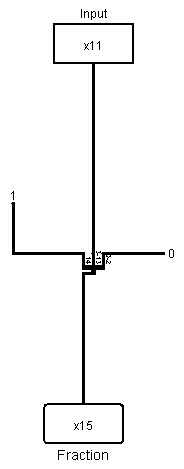
Figure : Normalizer

Figure 6: Partitioner

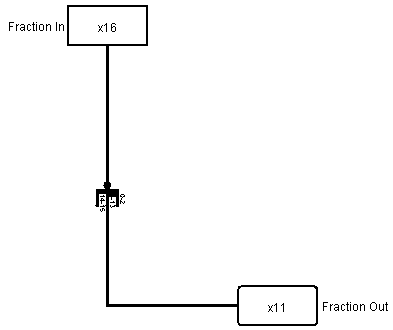


Figure : Truncation hardware

**Simulator Used With Version Number:**

Logisim 2.7.1

**Discussion:**

In this assignment, we designed a Floating Point Adder. For this purpose, we used basic gates (AND, OR, NOT, NOR), universal gates (XOR, XNOR) and some other necessary gates(MUX, Shifter, Bit Finder, Adder, Subtractor, Bit Extender, Negator). The Floating Point Adder takes two 16 bit binary numbers A and B which are basically binary representations of a floating-point number. Our implemented circuit outputs the added value of A and B. We used the known technique of designing the floating-point adder. We followed the provided flowchart, built the circuit diagram, and implemented the circuit. While designing, we put emphasis on simplifying the circuit. We asserted our design by testing various inputs and matching the corresponding outputs. On the simulator, alongside the full circuit, we made a block circuit so that all signals can be visualized easily. However, testing the outputs and completing the data table, we successfully finished our simulation