MIPS Floating Addition

Group 6

September 11, 2019

Contents

1 Input

There are two input numbers:

Sign	Exponent	Fraction
B15	B14 B13 B12 B11	B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0
A15	A14 A13 A12 A11	A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0

Also there is a set bit, named ”Set”. It is explained later.

Description

- To give input we can use sevral components in Proteus (E.g. Logic-toggle, Logic-state). We are using logic-toggle.
- So there are 2x16 + 1 = 33 logic-toggles.
- This logic-toggles are placed top of the circuit.

Logic-toggles are aligned in this manner:

Sign	Set	Exponent	Fraction
B15 A15	Set	B14 B13 B12 B11 A14 A13 A12 A11	B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0

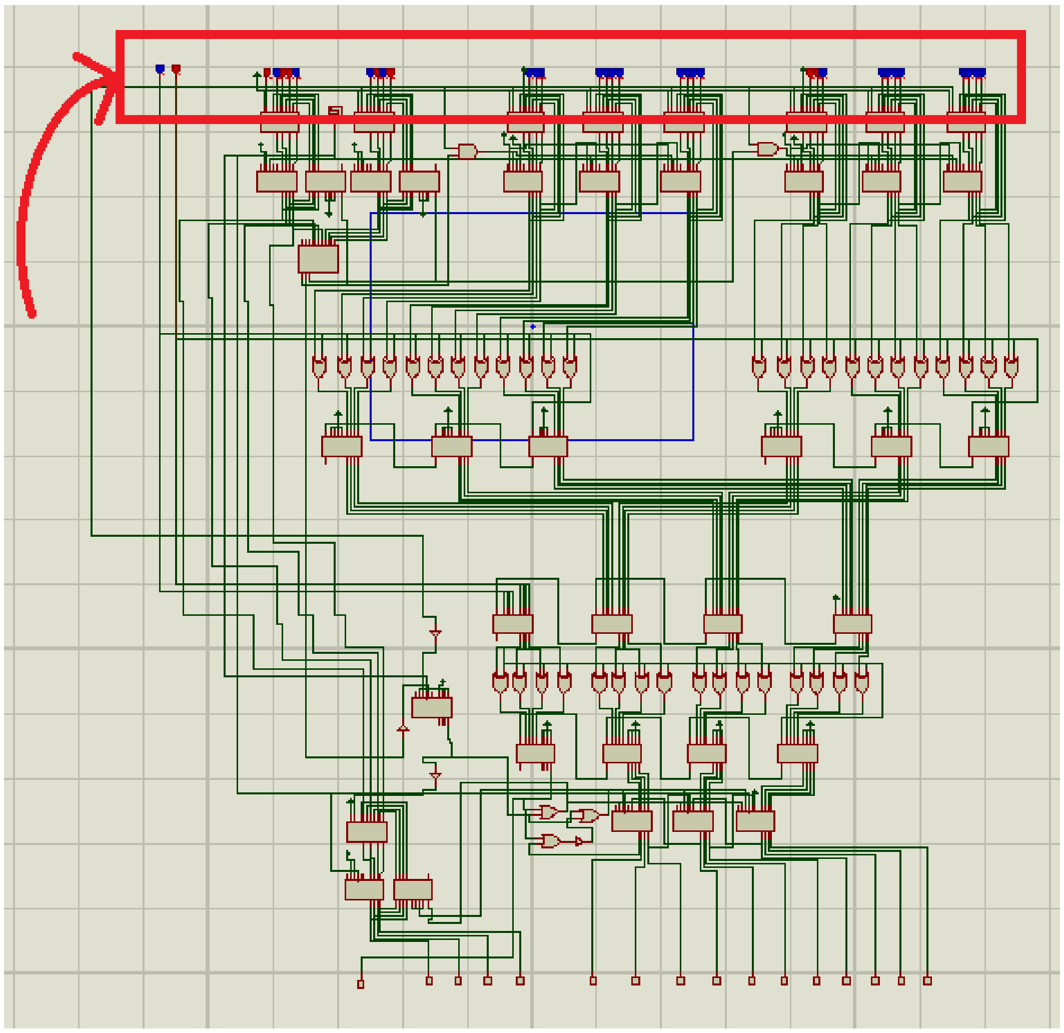


Figure 1: Input Pins

In short all input pins are not side by side for a particular number.

2 Output

Total 16 logic-probes are used to show output. These are placed to the bottom of the circuit. In this case, all output pins are serially aligned.

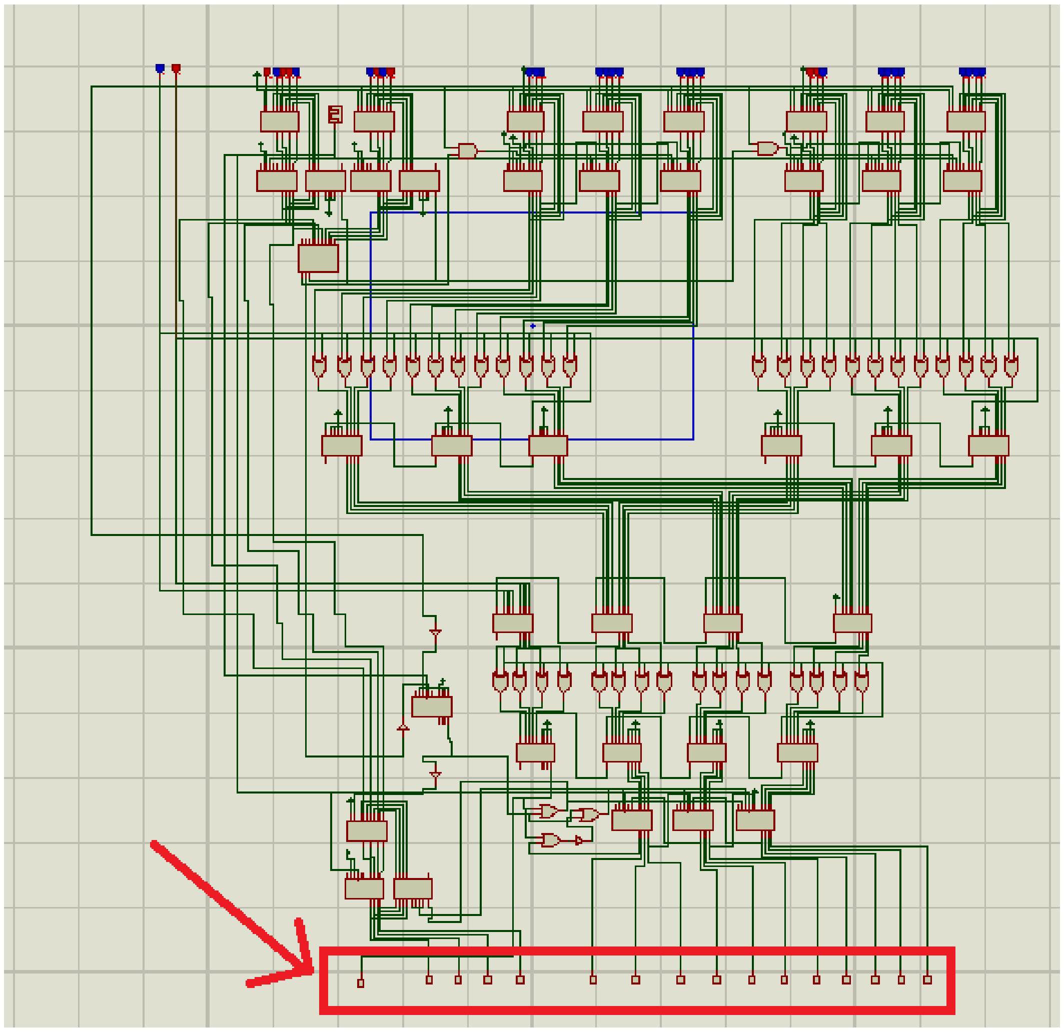
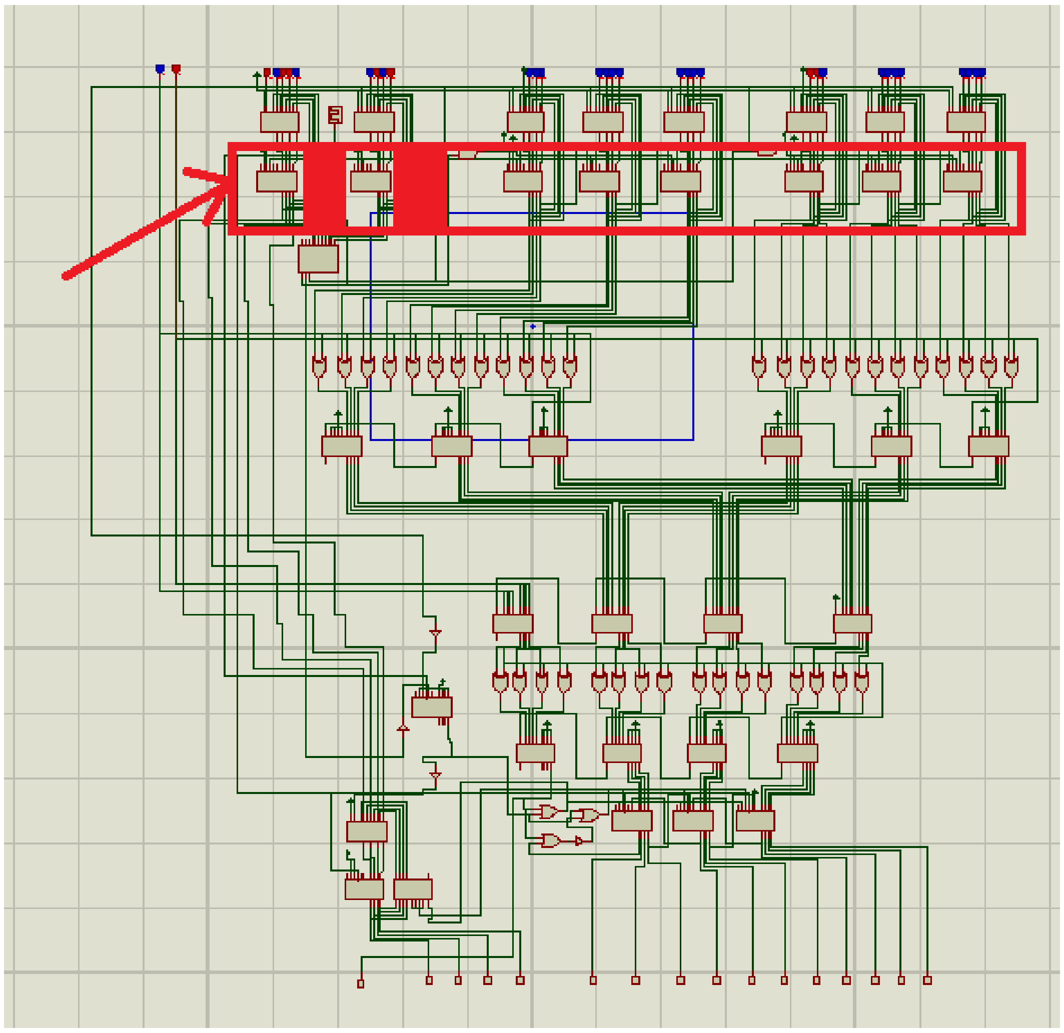


Figure 2: Output Pins

3 Workings

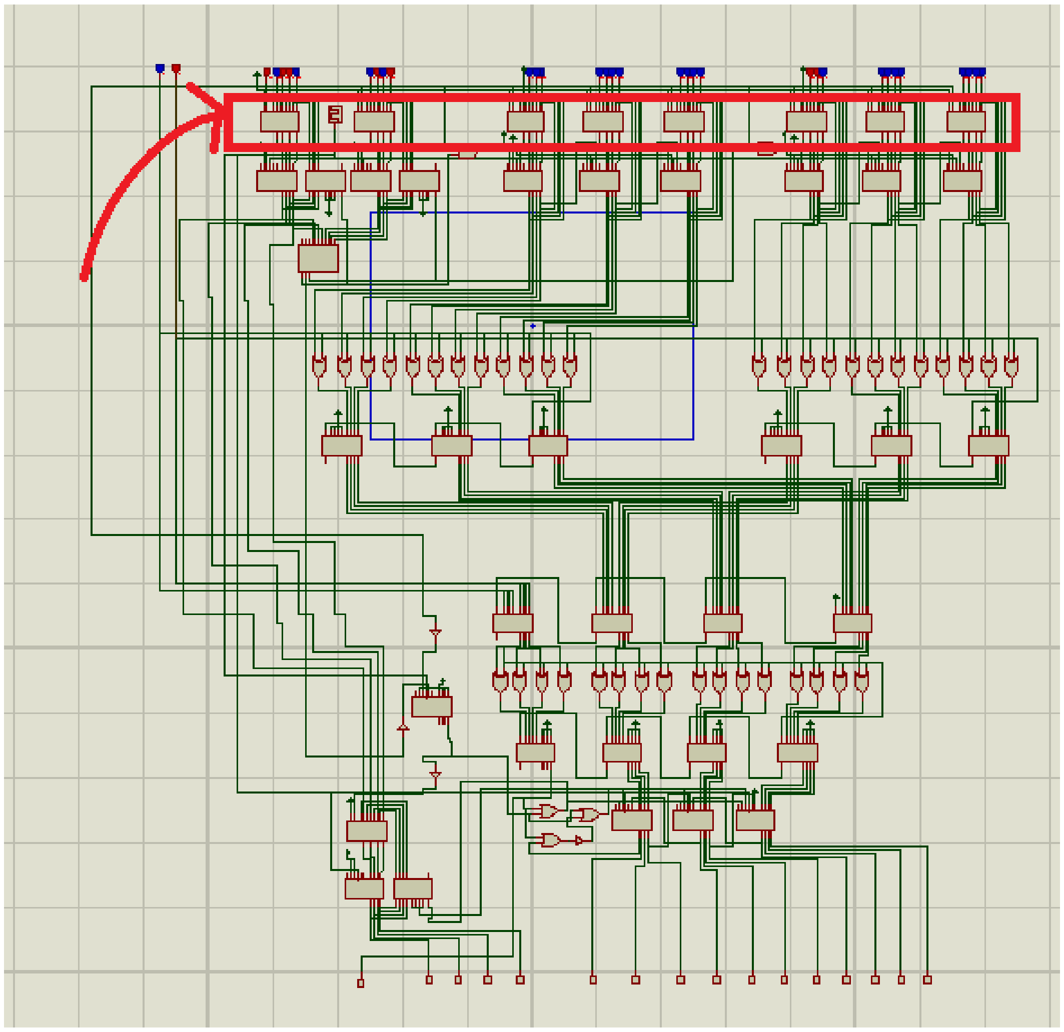
3.1 Input Registers

- Inputs are saved in registers to work with.
- 74LS192 ICs are used here. Each IC can save 4 bits. This ICs can left shift, right shift, hold and can take parallel input.
- For every number, 4 registers are used.
- Sign bits are not saved in the register. Because, we don't change sign bits. But other data may need to be changed. For example:
  - Exponent may need to be incremented.
  - Fraction may need to be shifted.
- 4 bit exponent is saved in a single IC.
- Other 11 bits(i.e fraction bits) are saved in three registers. Here, three registers can save 12 bits. We have an extra bit when we saving 11 bits. This extra bit is MSB and it is set to 1. Because, in MIPS, normalized floating point should always have 1 to the left. In a result, our calculation gets easier.



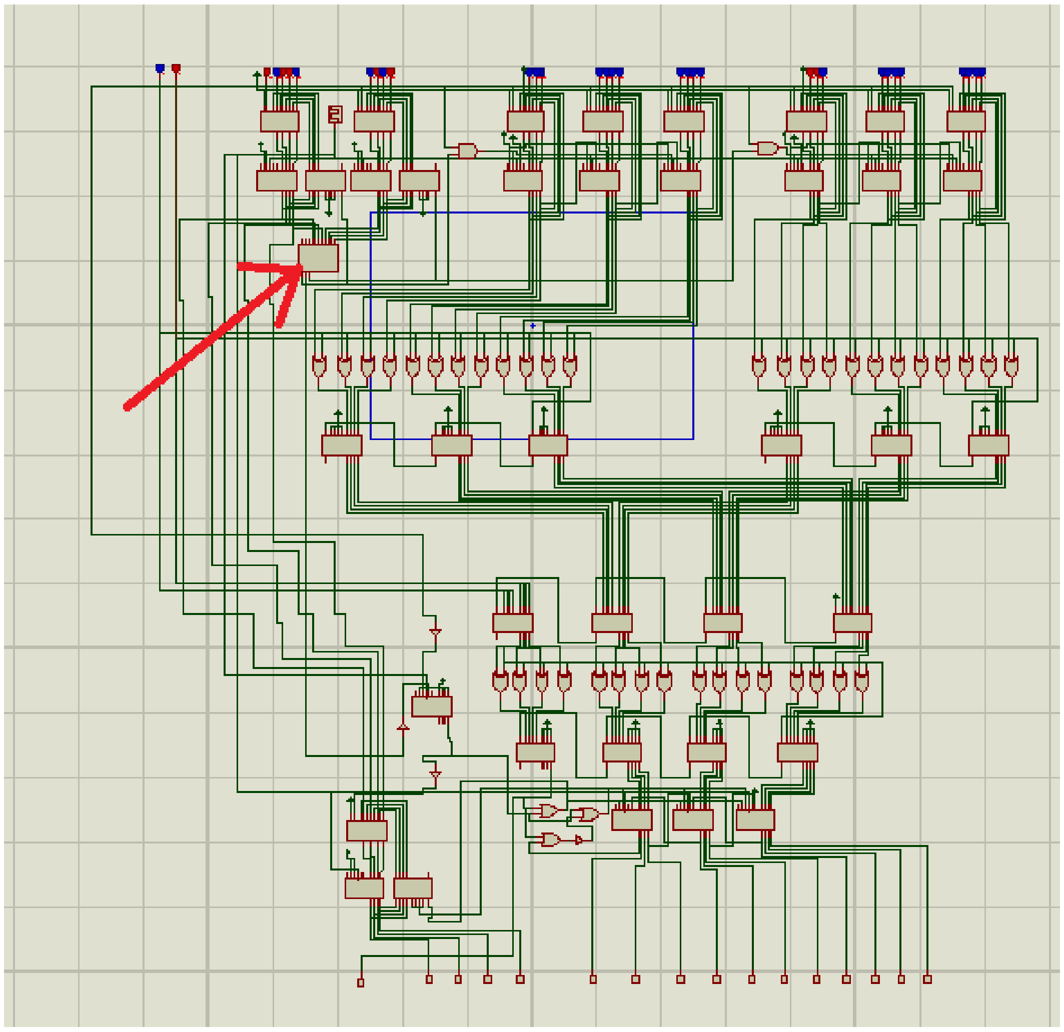
3.2 MUX

- We have seen, there is a ”Set” bit which works as an input.
- We set it to 1, if we finish giving input in all 32 bits.
- This ”Set” bit actually works as a switch of this MUX section.



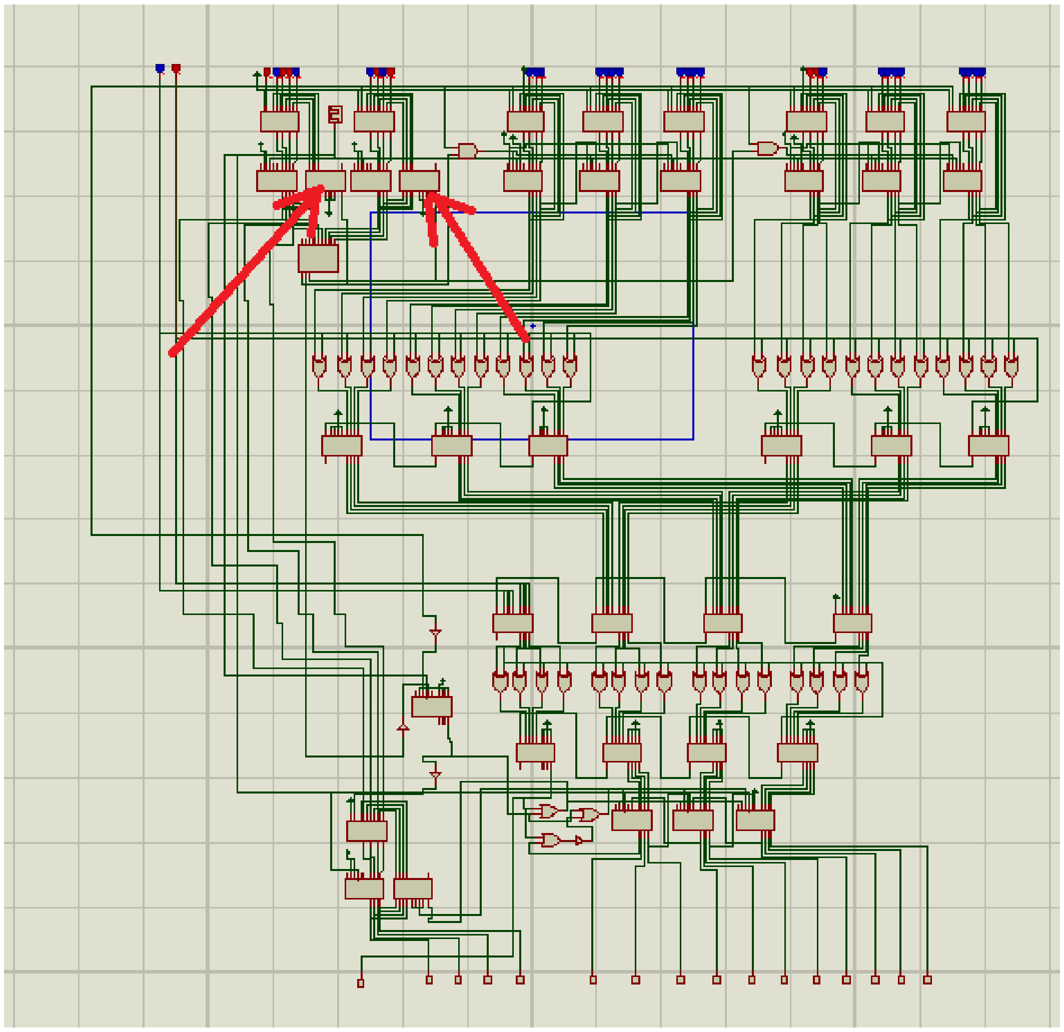
3.3 Comparator

- This is basic control unit of this circuit.
- It compares two exponentials, increments the lower exponential and rights shifts the corresponding significant.
- It takes one clock cycle to right shift once. If difference between two exponentials is n. Then it will take n clock cycles to prepare these two numbers to be added.
- When two exponentials become equal, shifting and incrementing stop.



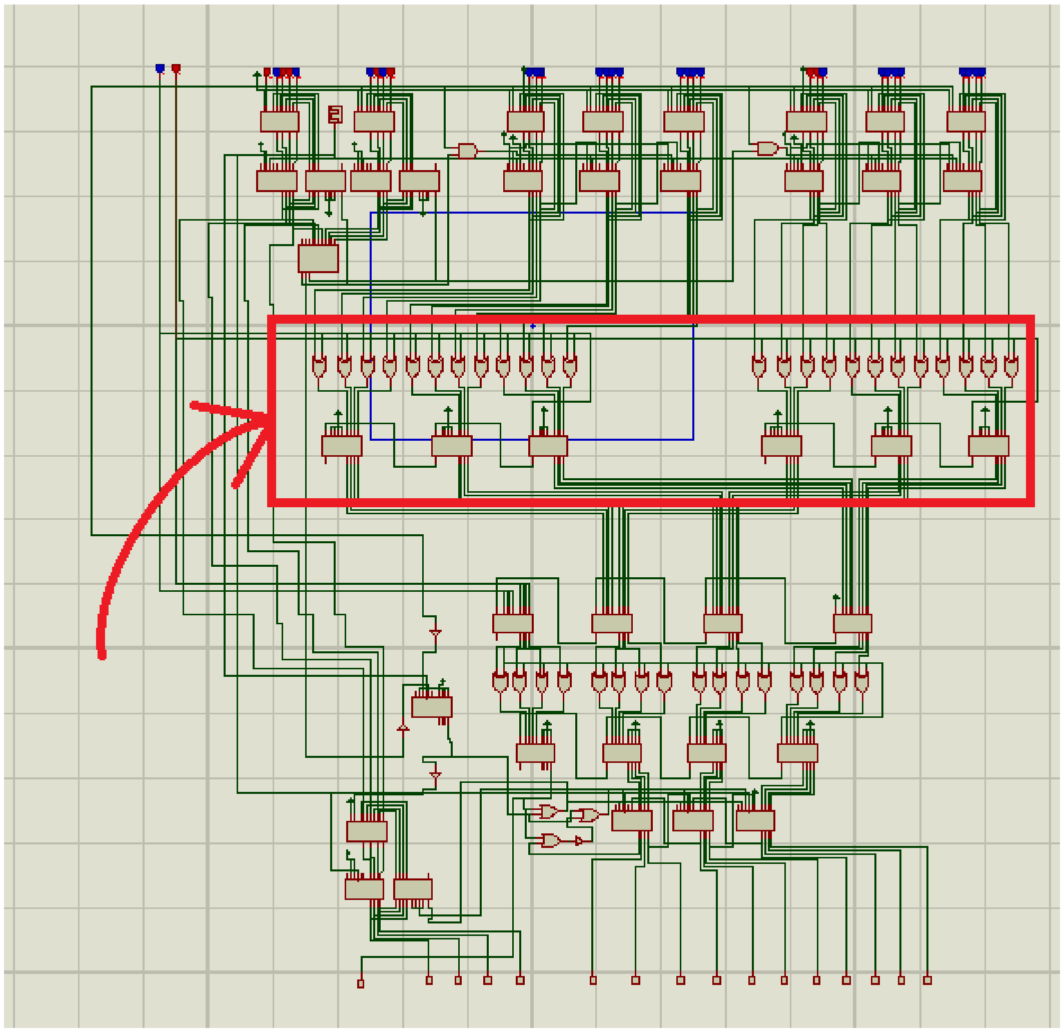
3.4 Adder for the Exponent

- We have seen lower exponent gets incremented in every clock cycle.
- This is done by two 4 bit adders.



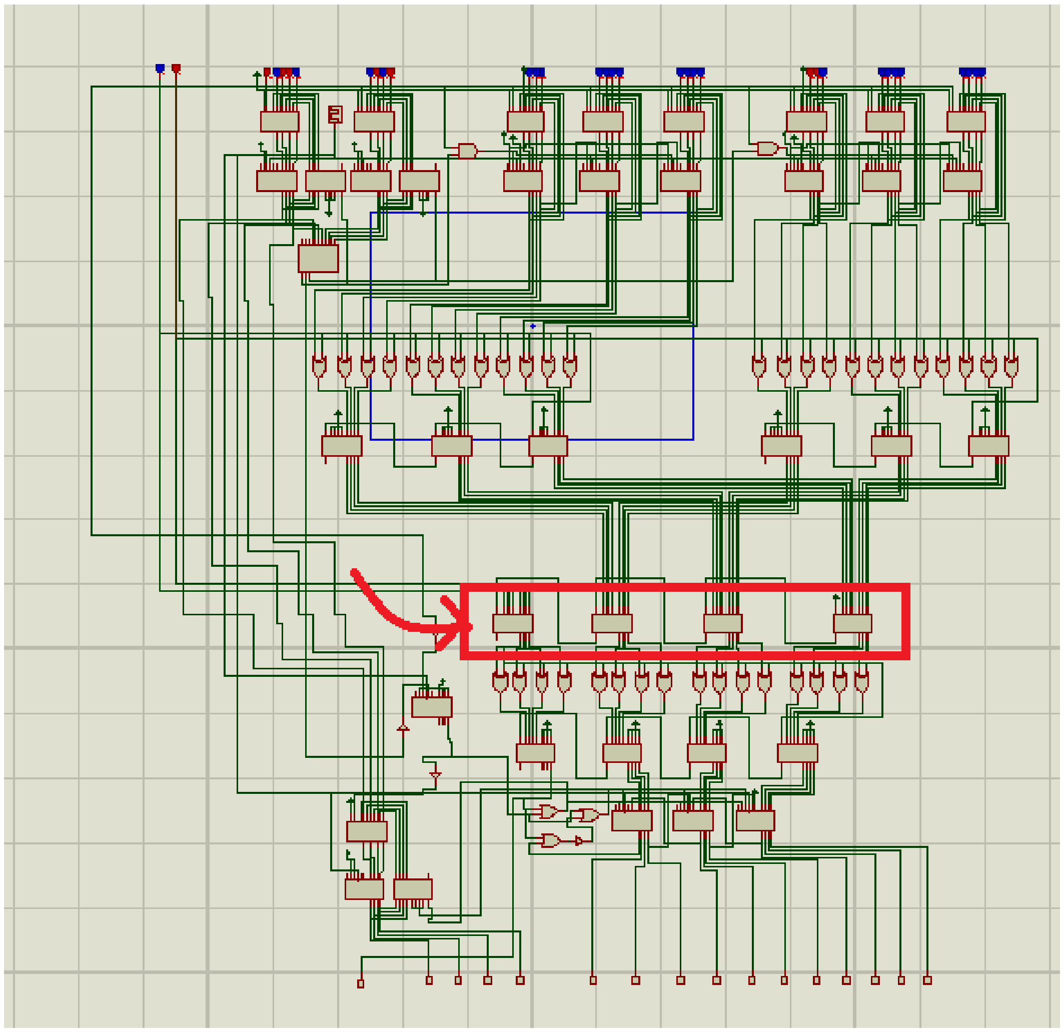
3.5 Negation for Sign Bits

- Comparator prepares two significant to get added.
- But if sign bit of any of these numbers is 1, we have to convert it to 2's complement.
- X-OR gates and adders are used to do that.
- This is a small detail but makes circuit-size much bigger.



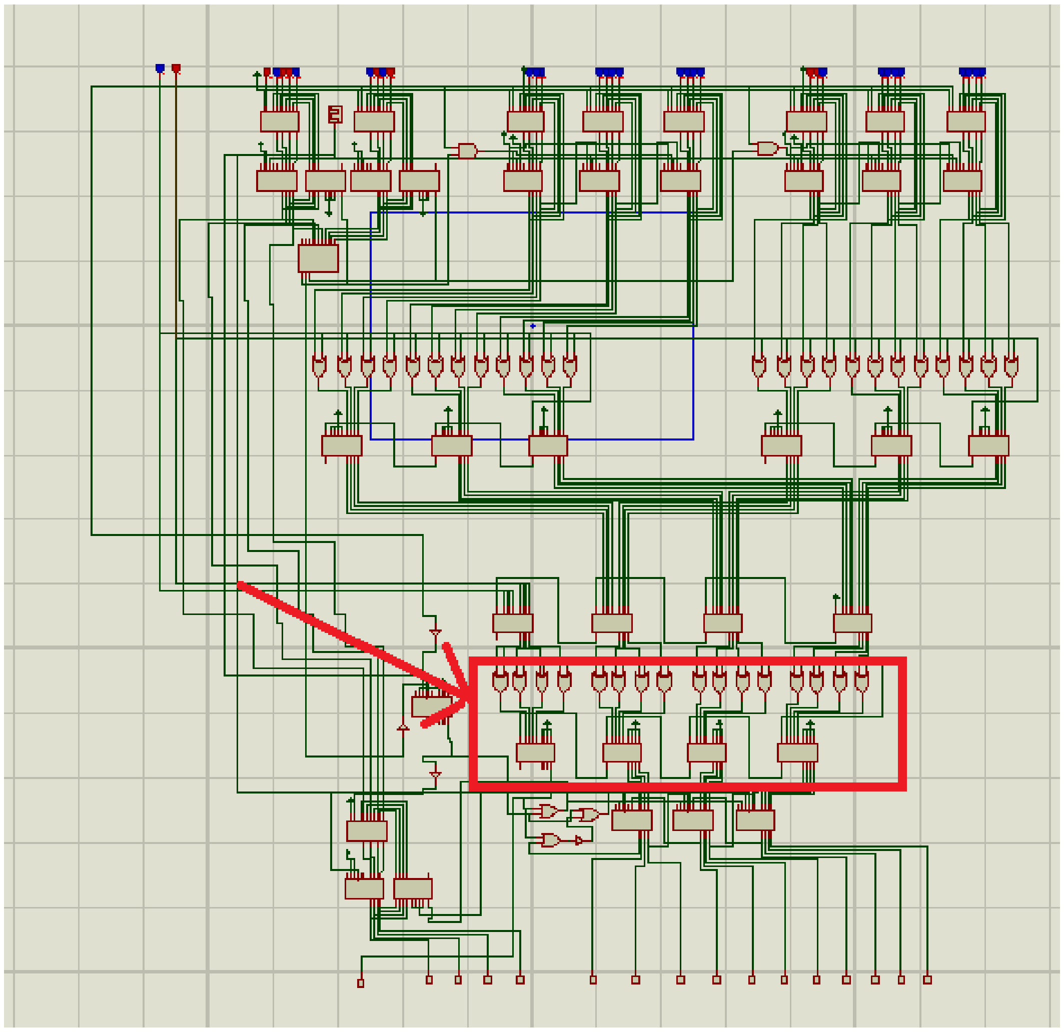
3.6 Main Addition

- This part is actually quite small.
- Four 4-bit adders are used (Specification told us to work with 16 bit ALU).
- We are adding 11 bits basically. Other bits are copied according to sign bit.



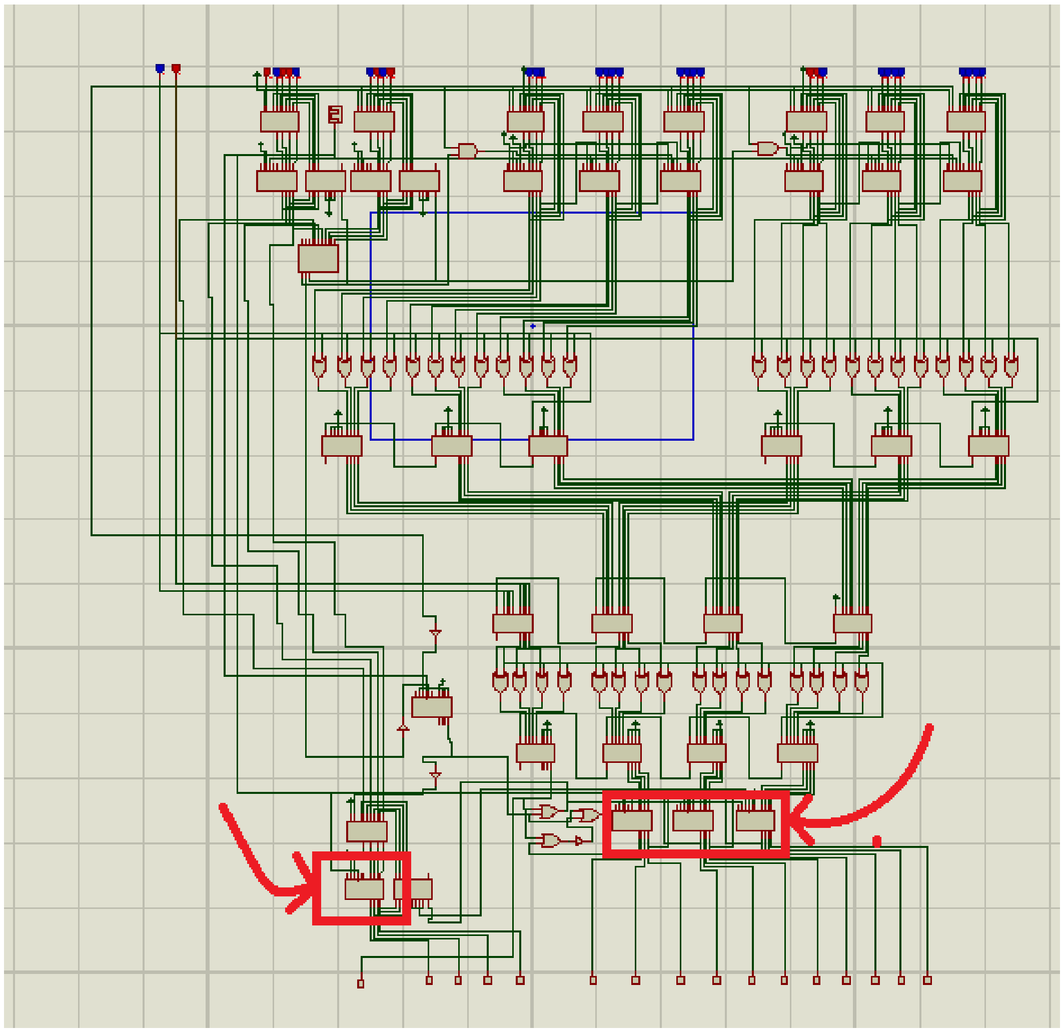
3.7 Negation Again

- The sign bit of answer can be 1.
- In a result, we may need to make this 2's complement.
- Again, small detail; adds up a whole lot of circuit.



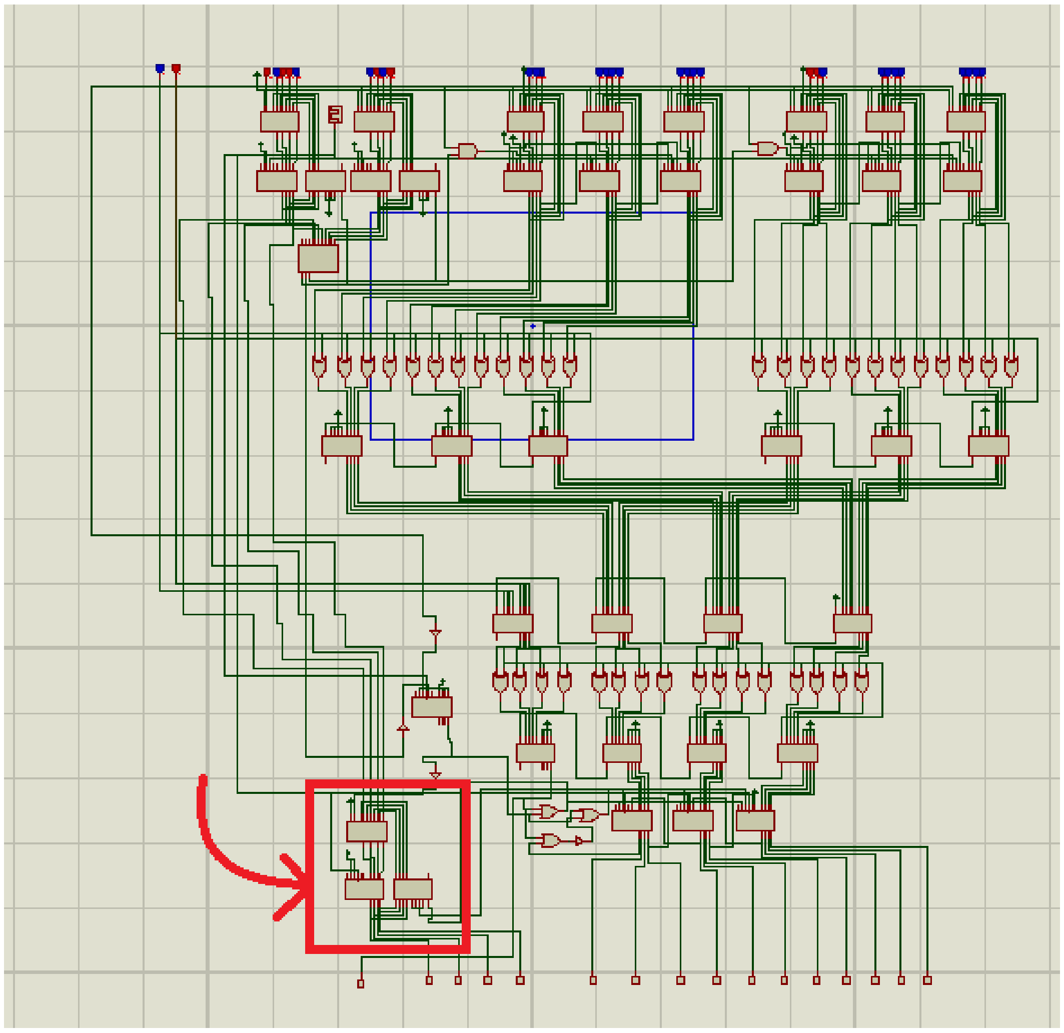
3.8 Result Register

- Finally we can save this result to our "Result Register".
- We notice that it can be normalized or not.



3.9 Normalizing

- Comparator decides when the result should be normalized.
- To normalize, we may need to right/left shift the fraction.
- We need to increment/decrement according to shifting operation. A 4-bit adder is used here.



4 Discussion

- Comparator IC can work with 4 bits. It can compare two 4-bit input. It has three output pins:
  - A>B
  - A<B
  - A=B

  Each pin gives 1 if and only if the corresponding statement is true.
- Clock pulse is given with built-in clock.
- For our implementation,

$$ClockCycle_{total} = ClockCycle_{shifting} + ClockCycle_{normalize} + 1$$

- A hard question is, "When we should start normalizing the result?" The answer is, "After proper shifting, when we get the result from adder." To be specific, the next(or later) clock cycle when the "A=B" pin of comparator gives 1.
- We get the "A=B" signal instantly when two exponents become equal. But we need this signal one clock cycle later.
- The trickiest part is to use this signal at the right time. We did this with a help of a register.
- Additional screen-shots are in the folder. Not included in LaTeX file for simplicity.
- Example from text book was simulated[1].

[1] Kisu test case run kore dekh thik moto output dey kina. Kono bug thakle bolish. Kono kisu change kora lagle change koris.