Floating Point Processor

Apoorv Agnihotri(16110020)
Mridul Sharma(16110095
Pankaj Vatwani(16110107)
Pratik Puri Goswami(16110121)

Introduction:

As we are living in 21st century and in the past few decades the speed of computers has increased by many folds. Thanks to fast processors for this. Thus we were fascinated by the working of the processors so we tried to implement them using FPGA as our project for Digital Systems. We implemented a floating point processor that uses the IEEE 754 Standard for Half Precision Floating-Point Arithmetic to do arithmetic calculations and representation. We designed a simple Harvard architecture processor with mere four instruction operations, addition, multiplication, subtraction, and division. We used direct referencing to get the data for carrying out our instructions.

The IEEE 754 Standard:

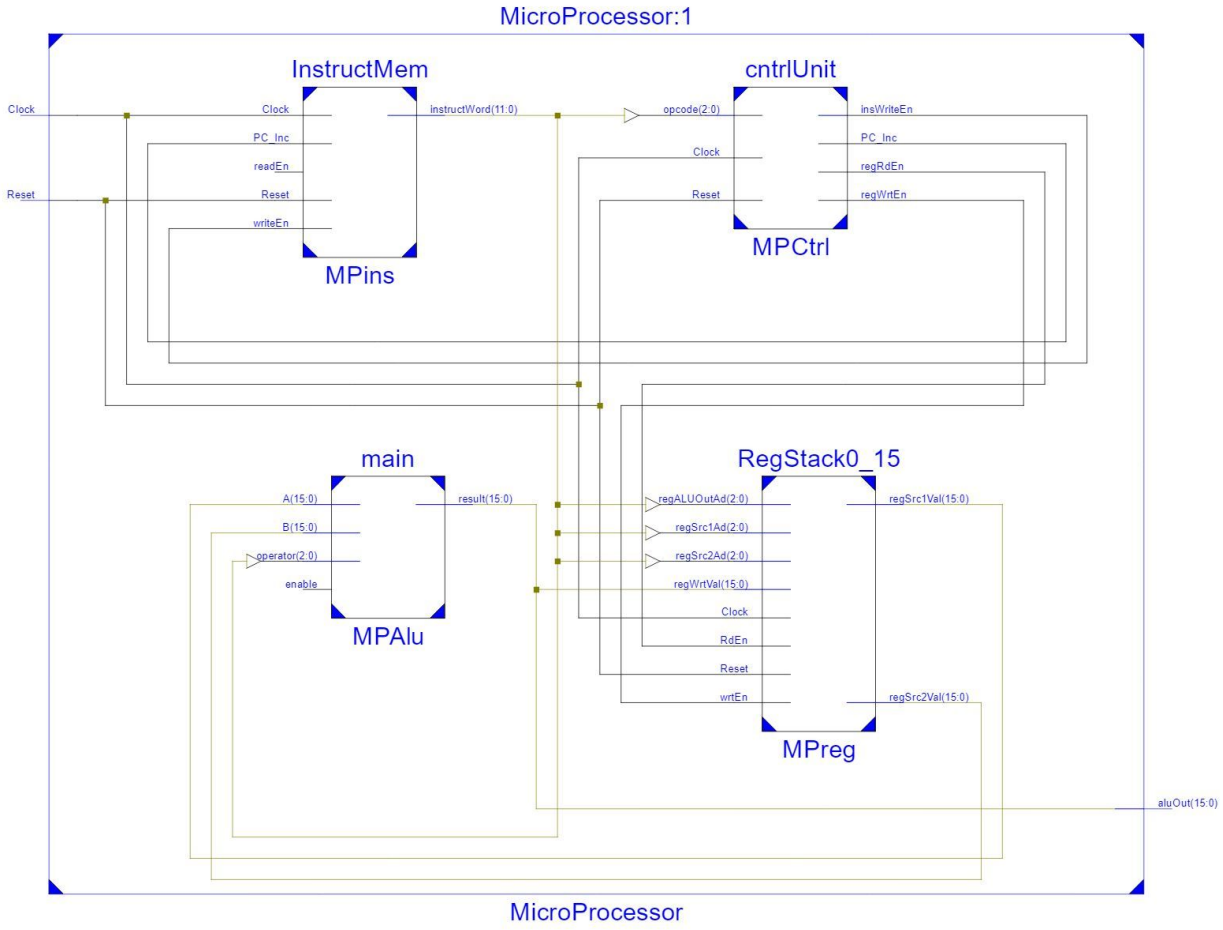
Floating point numbers are represented by IEEE standards. Representing a floating point is different than it's integers. A floating point number can be represented by 16 bits which are divided into 3 parts. First part consists of only one bit and it takes care of the sign of the number and thus called as "sign bit". Second part consists of 5 bits which are
Is the exponent of the number. And the third part represent the decimal part. It is also known as mantessa. It consists of 10 bits. A floating point is always represented in scientific notation so that the part before the decimal is always 1(except 0). First a decimal floating point number is converted into binary floating point number. This can be done by recursively dividing the left part of the decimal by 2 and by recursively multiplying the right part of the decimal by 2.

Once we get the floating point number in decimal, then we represent it in scientific notation such that left part of the decimal is always equal to 1. After representation we can get the exponent and the right part of the decimal. Right part of the decimal becomes mantissa of the IEEE number

and the exponent of the number becomes the exponent part of the IEEE representation.

Processor Block Diagram:

Below is the final RTL of our Microprocessor showing the different components including the Control Unit, Instruction Mem and Data Register (as we followed the Harvard architecture), and the ALU that allows 4 instructions, + | - | * | / .

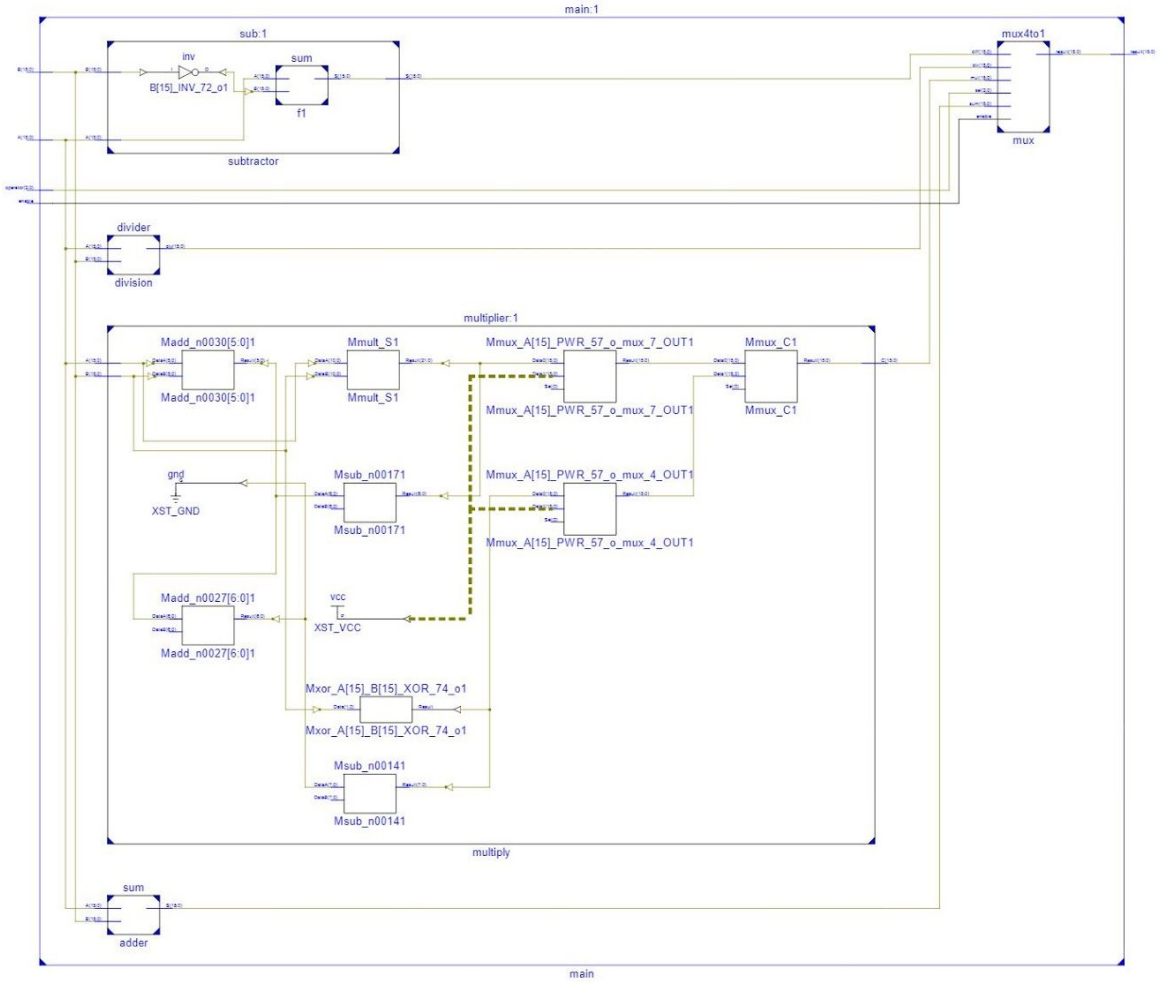


In the microprocessor we used a clock that would direct the control unit to do specific tasks and functions with each clock tick. We programmed and made the other basic parts of microprocessor, the Instruction and Data registers making them independent modules that are called in the main microprocessor module.

Arithmetic Logic Unit (A.L.U) :

The arithmetic logic unit does four operations Add/Multiply/Divide/Subtract. We mostly used the arithmetic operations available in the verilog itself to majorly code

the ALU part. Division in ALU is not supported directly so we used different algorithms that are mentioned in the division part of the report.



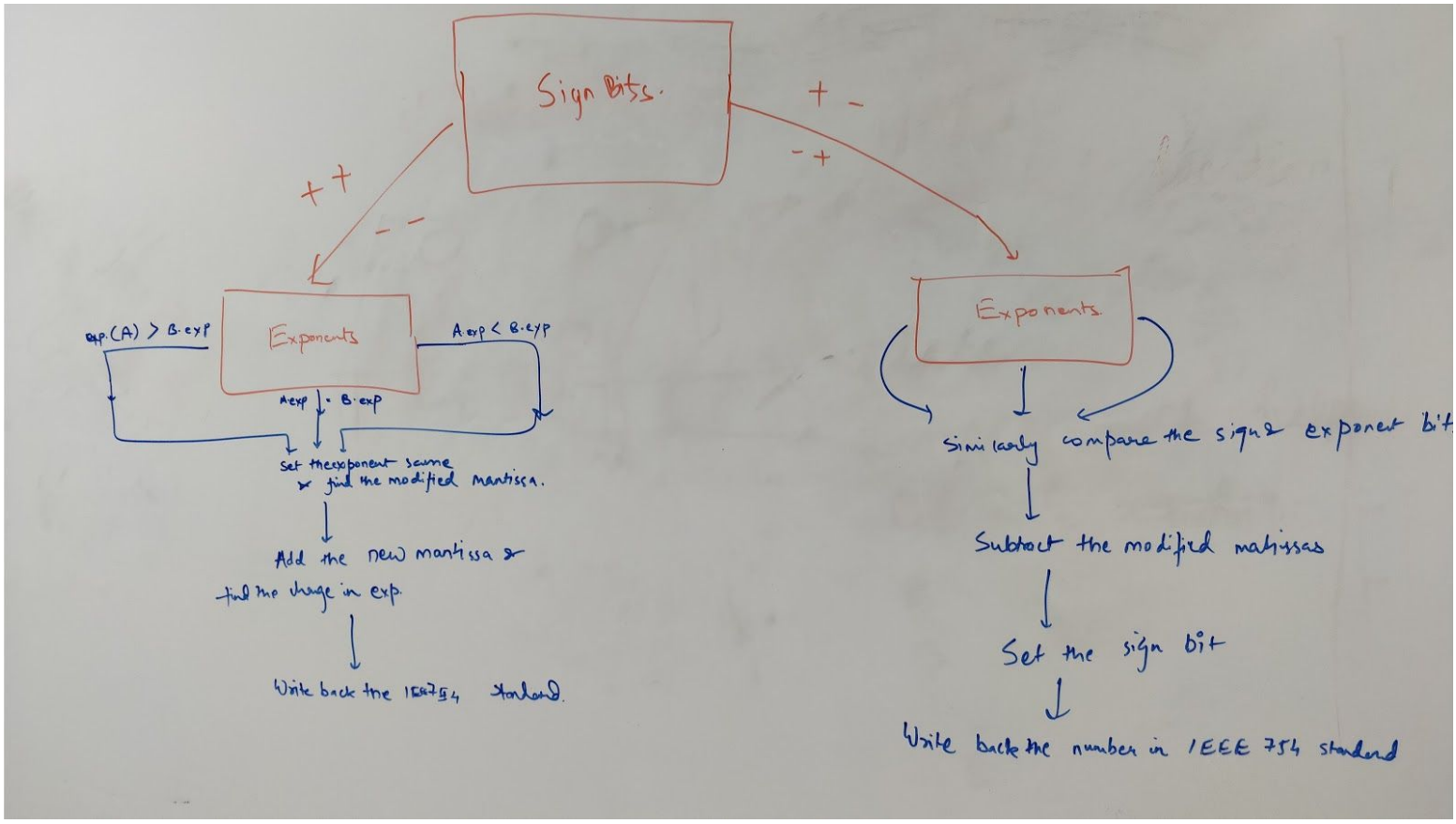
Adder:

An adder is a digital circuit that performs addition of two numbers.
We add two number using direct addition that is provided in verilog code. We follow the below defined procedure to find the sum of all types of the numbers.

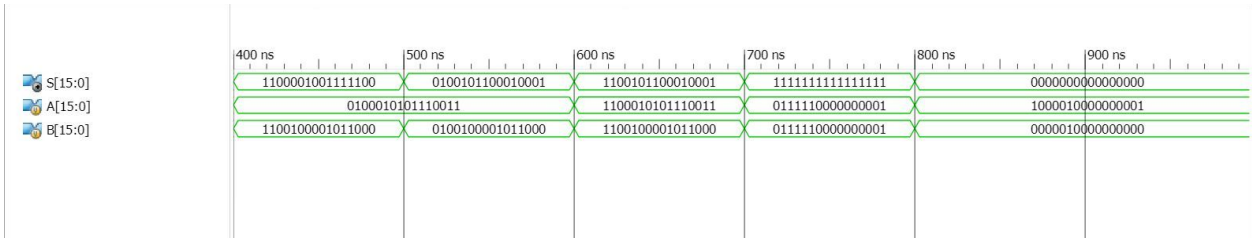
Below is the main procedure we followed to find the IEEE 754 Half precision Floating point.

We first checked if the number provided to us had the same sign or not, this made our code distribute in two branches. Further we made the exponents equal by padding the mantissas with zeros and then added or subtracted according to the signs of the operands. Further we outputted the value of our operation in IEEE 754 Standard of Floating point Representation.

The final part of our code checked for any particular corner cases inputs so as to directly provide the answer.



Simulated Output:

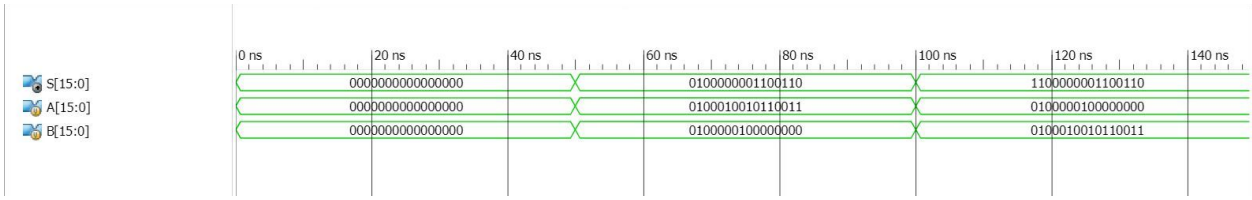


Subtractor:

The subtractor module is nothing but a module that calls the adder circuit by just changing the sign bit of one of the operands and showing the result as it is. The subtractor module is made such that it would give the results ass it is of the adder module.

Subtractor module subtracts two floating point numbers. In order to minimise the hardware one option is to utilise the code written in the adder module. We can do this by changing the sign bit of the number which is being subtracted. Since the only difference between two additive inverse is of sign bit since +x and -x both will have same mantissa as well as same exponent. Thus in order to reduce hardware we are using adder module inside subtractor.

Simulated Output:



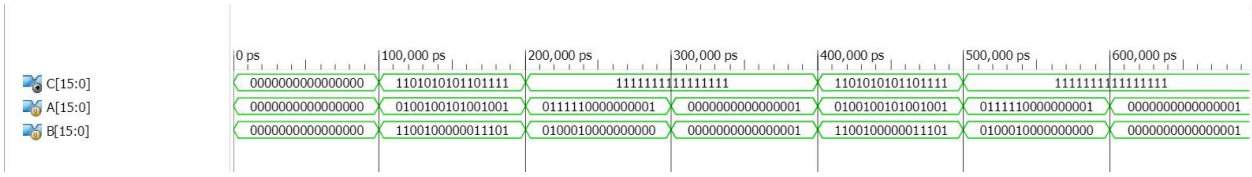
Multiplier:

The multiplier is a digital circuit that multiplies two 16 bit floating point numbers.

Algorithm Used:

The sign bit of the final result is the xor of the sign bit of two numbers that we are multiplying.We multiply the two mantissas with 1 concatenating to each of them which makes each number of 11 bits.The result is a 22 bit number(S[21:0]). We check the most significant bit of the number.If it is one the exponent of the resultant number is the sum of the exponent of two numbers plus one otherwise it is just the sum of exponent of two numbers.The mantissa of the resultant on the other hand will be S[20:11] if the most significant bit is 1 otherwise it is S[19:10]. For the checking of underflow overflow cases if the exponent exceeds 5 bit and becomes a 6 bit number it means there is either a overflow or an underflow in which case we print 1111111111111111.

Simulated Output:



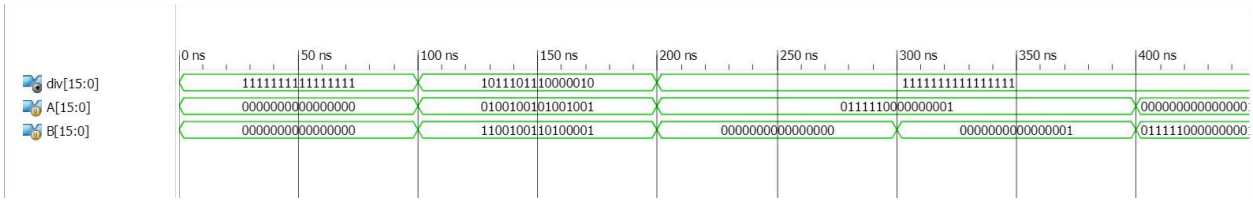
Divider:

When we first tried to implement our algorithm to find the division of two number we tried to directly divide the numbers by using the division sign, but as we got to know that we can not divide two numbers directly using the division symbol, as we used in multiplication. We read about Restoring and Non Restoring algorithms but we were not able to find the correct division of two numbers. At last after trying really hard we took to implement long division from scratch and having flag for decimal place. We were able to find the correct values of the division using this algorithm but the major problem faced was the board are and resources our crude and inefficient algorithm was using.

Algorithm Used:

First we would directly find the sign bit of the result by using a xor gate. Next we directly try to subtract the exponent part of the IEEE 754 number and find the probable exponent of the result. Finally we concatenate a 1 in front of the mantissa and then divide the two numbers using our algorithm. We then check for the shift in the position of the decimal place of our division and update the exponent part of the divider. After updating all the parts of the resultant IEEE number we give to the output.

Simulated Output:



Microprocessor:

It is a circuit designed to perform more than one arithmetic operation. It consists of various parts:

- **Data Register(DR):** DR is a 8 word register which holds the value of operands. Whenever the clock edge comes the control unit reads a instruction from the instruction register. Once the instruction is read then the program counter is increased by 1. After that the instruction is decoded and the op code will define which operation has to be implemented (opcode is the first three bits of the instruction provided). Middle 6 bits will give the address of the two operands(3 bits each). After that the data bus will fetch data from data register. Once the control unit will get the value of the operands then the operands' value will be sent to alu and since the alu is a combinational

circuit it does not wait for clock edge and once the result is returned to the control unit. Then the write_en (writing is enabled in the data register) becomes one and thus the result gets stored in the memory register at an address specified in the last three bits of the instruction code.

- **Instruction Register(IR)**: Stores a 12 bit instruction which contains the information about the operation that needs to be performed, the operands and the place where final result would be stored.The 12 bit instruction is divided into four parts each containing 3 bits.The first three bits (MSB) are further divided in two parts- one containing the first bit and the other containing next two bits.The first bit represents whether we have to stall the PC or not.The next two bits represents the operation needed to be performed. The list of bits and the operation they signify as follows:

00 - Addition

01 - Subtraction

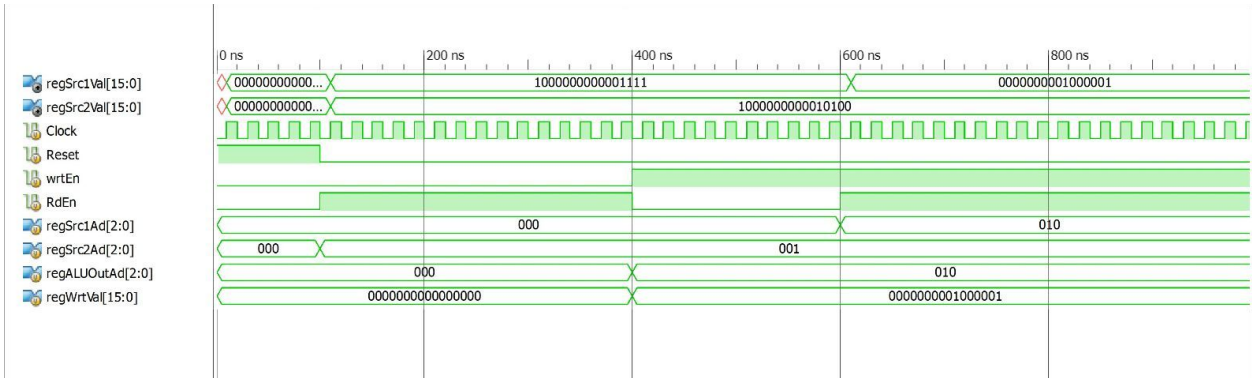
10 - Multiplication

11 - Division

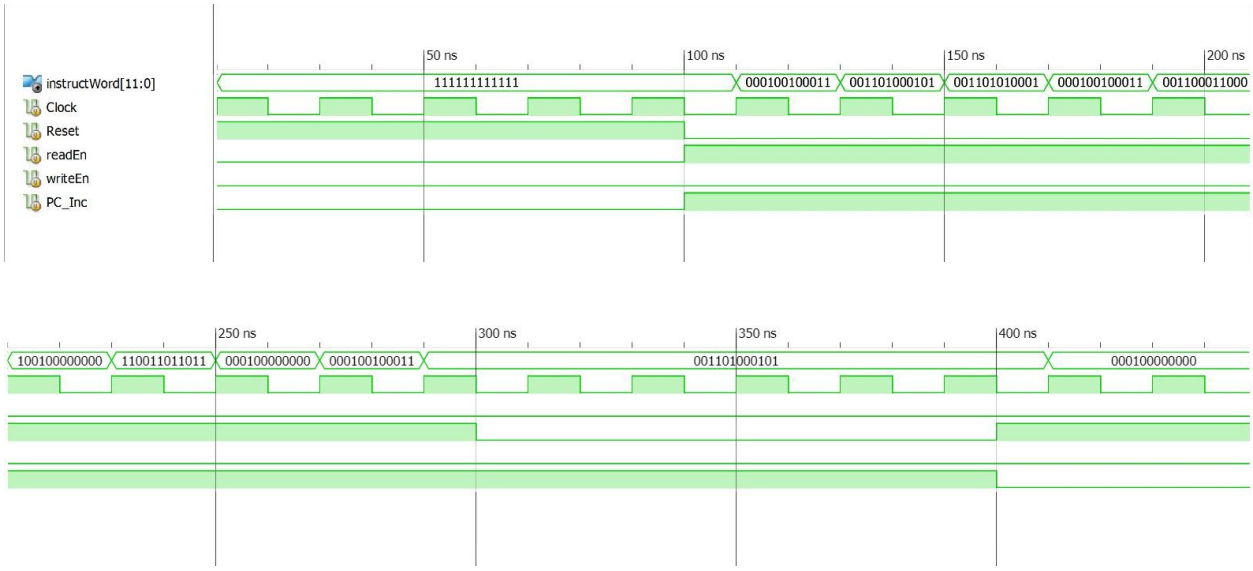
The next three bits contain the address of first operand.The next three bits contain the address of the second operand.The final three bit consists of address of the location where the result has to be stored.

- **Program Counter(PC):** Keeps the track of instructions in the instruction register.
- **Control Unit(CU):** It takes the instructions from the IR register, decodes it and instructs ALU to perform the tasks accordingly.
- **Arithmetic Logic Unit(ALU):**The arithmetic logic unit performs the operation on the operands present in the data register and returns the result back to it.

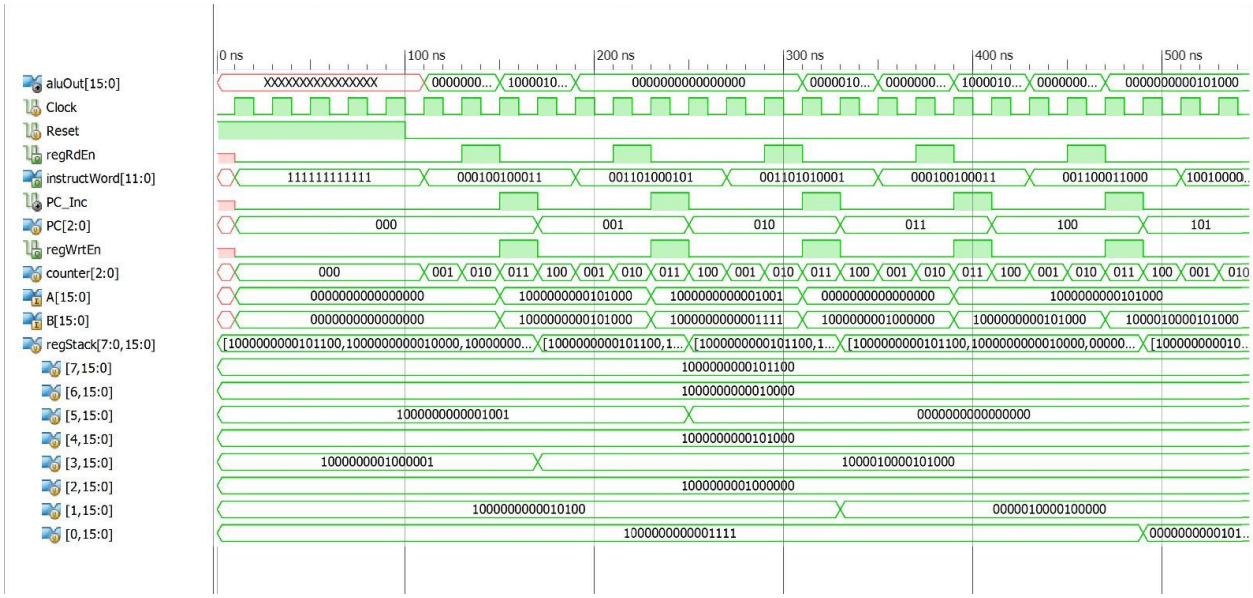
Register Simulated Output:



Instruction Register Simulated Output:



Microprocessor Simulated Output:



Results and Discussions:

The simulated output of all the modules have been shown above.Floating point ALU as well as processor are therefore completed successfully.

We tried to implement division using restoring algorithm but it didn't work for us as it was only giving us the quotient and remainder but we wanted floating point quotients so we used long division method instead and it thereby used more memory.

There is a further scope of optimization in our code.

We tried to implement pipeline processor but were not able to understand much about it and thereby implemented the processor without pipeline.

Conclusion:

At the end we would like to say that we enjoyed working in the project.It was a good learning experience for all of us. We got the hang of the things taught in the class. We also learnt about how are floating point numbers actually represented and some of the algorithms that are used to perform operations on them like boots algorithm for multiplication, restoring for division etc.

Acknowledgement:

We have taken efforts in this project. However, it would not have been possible without the kind support of Prof. Joycee Mekie and Mr. Sarathchandran GM for their constant guidance and supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

Reference:

1) Computer System Architecture by Morris Mano
2) http://weitz.de/ieee/
3) https://www.youtube.com/watch?v=Dl8bAyyoPVo
4) https://www.youtube.com/watch?v=LXF-wcoeT0o