Floating point processor

Digital Project

Assisted by - Sarathchandran Gm

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

INDEX

1.)Floating point notation

2.)A.L.U

-> Adder
-> Subtracter
-> Multiplier
-> Divider

3.)Processor

-> Microprocessor
-> Control Unit
-> Register

- Instruction register
- Memory register

-> Multiplexer

FLOATING POINT NUMBERS

- Floating point numbers are numbers with one or more decimal digits after them. For eg. 2.3, -3.457 etc.
- In binary number system floating point numbers are represented in IEEE-754 standard format.
- A number is, in general, represented approximately to a fixed number of significant digits (the significand) and scaled using an exponent in some fixed base; the base for the scaling is normally two, ten, or sixteen.

$$\text{significand} \times \text{base}^{\text{exponent}},$$

- Significand is integer ; base is also integer greater than two and exponent is also a integer. For example:

$$1.2345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10^{\overbrace{-4}^{\text{exponent}}}}_{\text{base}} .$$

IEEE-754 STANDARD

- To express a number say a = 15.4689 in IEEE - 754 standard format the following method is used:
  - First the number is represented in the binary format i.e a_{10} = 15.4689: a_2 = 1111.01111000000001001110 = 1.11101111000000001001110 x 2^3
  - Now base = $2^{(k-1)}$ - 1 (where k is the number of exponent bits); for 16 bit format number of exponent bits = 5, ∴ base = $2^{(5-1)}$ - 1 = 15.
  - In a_2 exponent is 3 from 2^3, to calculate the exponent bit representation in IEEE - 754 standard format we add the base to the exponent = 15+3 = 18 ∴ $exponent_{IEEE}$ = 18 = 10010
  - Now the fractional part for the IEEE standard is the first 10 bits of a_2 decimal point i.e. $fractional_{IEEE}$ = 1110111100
  - Final IEEE - 754 representation for **a = 15.4689 is given by 0 10010 1110111100** where the first 0 is the sign bit; here 0 represents a positive number while 1 represents a negative number.

Limitations of IEEE Standard

We can not represent all the numbers in IEEE standard because we are allowed to use only a finite number of bits. Due to this we can not represent numbers beyond a particular range. Mainly two cases will occur namely

Underflow: The underflow occurs when the number to be represented is less than the lowest limit. The underflow limit is $1.0000000001 \times 2^{-15}$

Overflow: The overflow occurs when the number to be represented is greater than the higher limit. The overflow limit is $1.1111111111 \times 2^{16}$

Arithmetic And Logic Unit (ALU)

- The ALU is used for computing the calculations given as input to the processor.
- Our ALU can be used to add, subtract, multiply or divide two floating point numbers. Eg: a+b, a-b, a*b or a/b.
- The ALU consists of 4 separate modules:
  - Adder
  - Subtracter
  - Multiplier
  - Divider

Adder

- The adder module takes care of the following cases:
  - a+b
  - -a-b -> Here a and b are non-negative real numbers
  - -a+b
- In case of overflow it gives 16’b1 or 1111111111111111 and in case of underflow it give 16’b0 as output.
  - Overflow case:
    - 0 11111 xxxxxxxxxx + 0 11111 xxxxxxxxxx
  - Underflow case:
    - 1 00001 000000001 +0 00001 0000000000

Algorithm Used

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.

Subtracter

- In order to reduce hardware we are using adder module inside subtractor.
- Subtractor simply calls the adder module to compute the case: (a-b) where a and b are non-negative real numbers.
- In case of overflow it gives 16’b1 or 1111111111111111 and in case of underflow it give 16’b0 as output.
  - Underflow case:
    - 0 00001 0000000000 - 0 00001 000000001

Divider

- The divider module works on the long division method and deals with the following cases:
  - a/b
  - -a/b -> here a and b are non-negative real numbers and b ≠ 0
  - a/-b
  - -a/-b
- In case if b = 0 is given as input then a/b = 16’b1.
- Similarly in overflow or underflow case output = 16’b1
  - Overflow case:
    - x 11111 xxxxxxxxxx / {x00000, 10’bx}
  - Underflow case:
    - {x00000, 10’bx} / x 11111 xxxxxxxxxx

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.

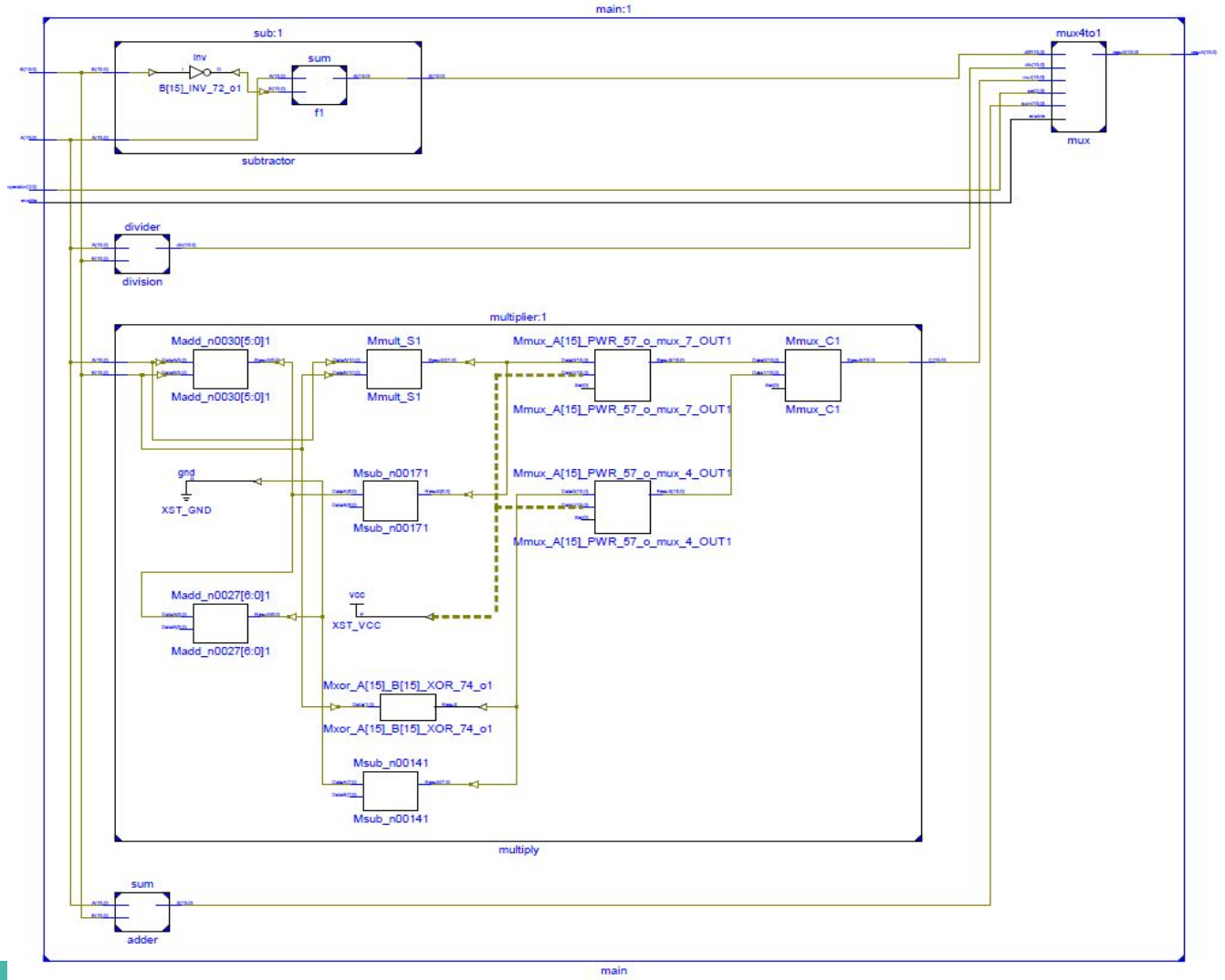
Multiplier

- The multiplier module is used to multiply two floating point numbers. The cases it deals with are as follows:
  - a x b
  - -a x b -> Here a and b are non-negative real numbers
  - a x (-b)
- The multiplier gives 16’b1 as the output for overflow and underflow similar to the adder module.
  - Overflow case:
    - x 11111 xxxxxxxxxx * x 1xxxx xxxxxxxxxx
  - Underflow case:
    - 0 00000 000000001 * 0 00000 000000001

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.

ALU RTL Schematic



Microprocessor

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

- Data Register(DR): Stores the operands and the final result as well.
- 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.
- 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.

Control Unit(CU)

- CU is the brain of the complete processor unit, it takes the reset and Opcode as inputs from the instruction register and gives the following outputs:
  - PC_Inc
  - regRdEn
  - regWrtEn
  - insWriteEn
- These final outputs are obtained by processing the inputs across the ALU according to the instructions given in the instruction Word file.

Data Register(DR)

- Data register is a 8 Word register which stores the value of the operands.
- Since it is harvard architecture, the instruction register is kept separate from data register.
- In order to read from data register we have to make read_en (enable) equal to 1.
- Once the operation is done we store the result in data register by enabling Write_en.

Instruction Register (IR)

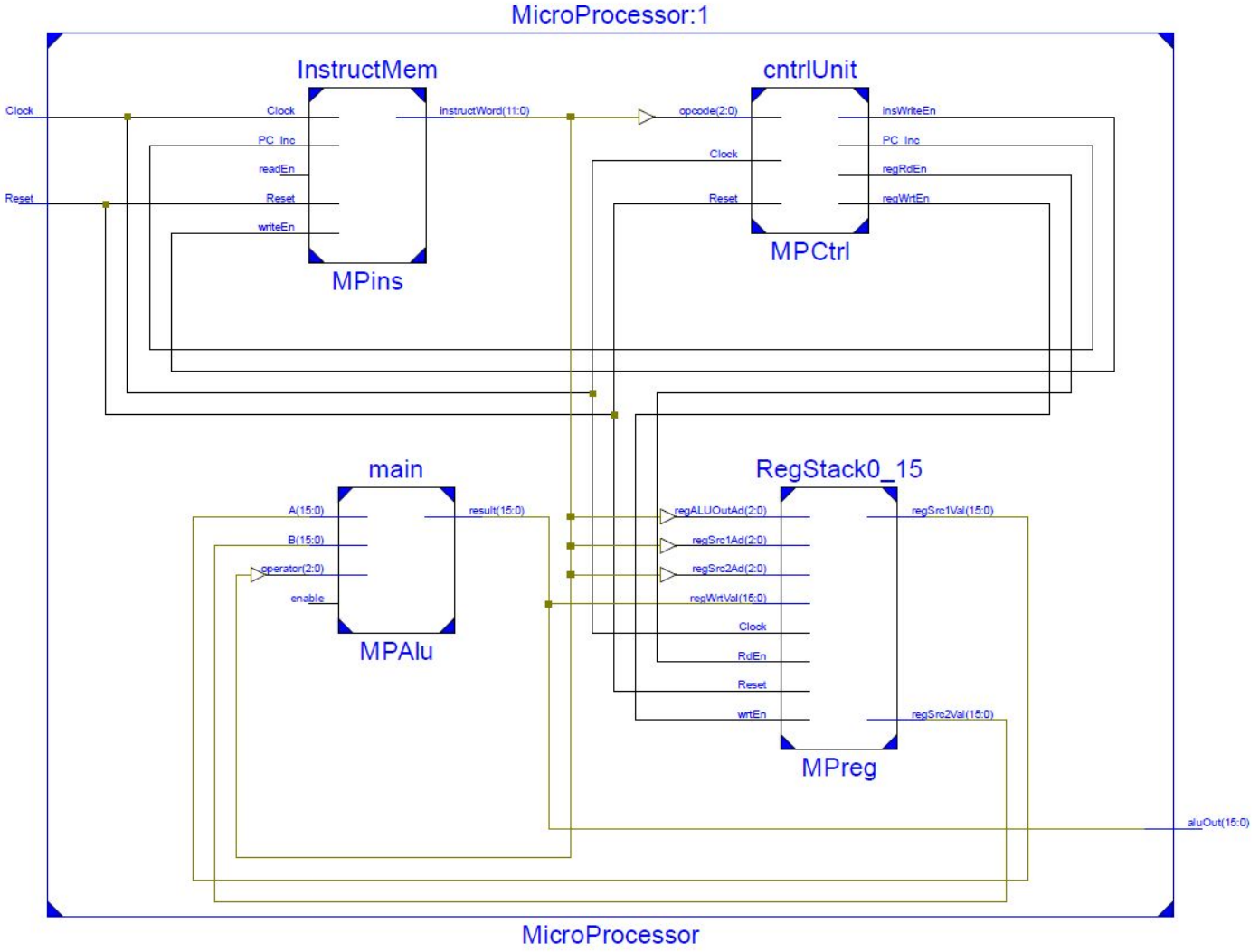
- The 12 bit instruction is divided into four parts each containing 3 bits.
- The first three bits (MSB) are further divided in two parts- one containing the first bit and the other containing next two bits.
- The first bit represents whether we have to stall the PC or not.
- The next two bits represents the operation needed to be performed.
- The 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.

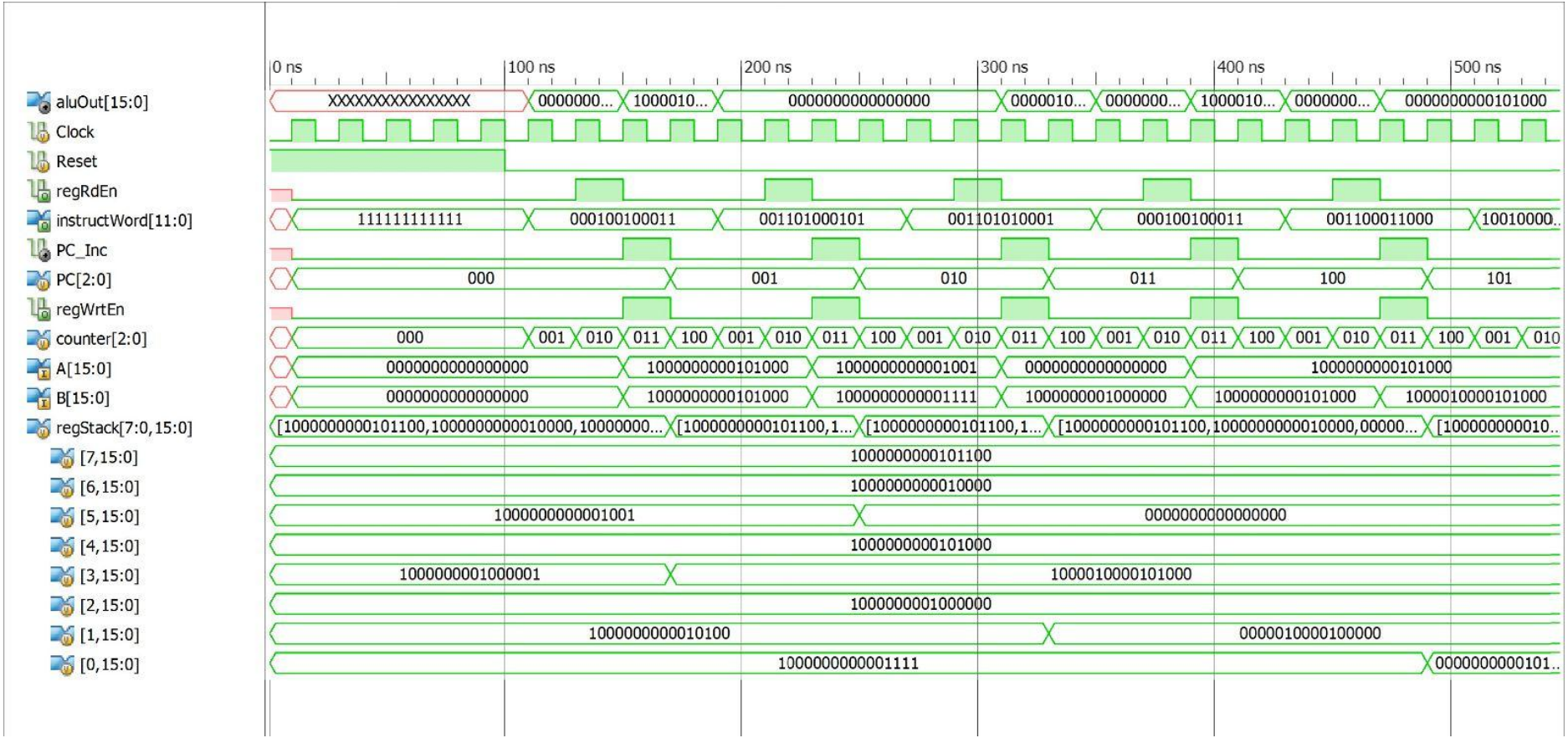
Program Counter (PC)

- PC is a 3 bit register in IR which stores the address of the instruction which is being performed.
- PC increases its value by 1 when PC_inc (Enable) is set to 1.
- PC can also jump from one address to another address but in our program it is always being increased by 1.
- PC stops increasing when we have an instruction that has the most significant bit in the Instruction.

Microprocessor

RTL Schematic





Microprocessor Simulation

Learnings

- Learnt IEEE standard of representing data.
- Different algorithms like boots, restoring.
- Got an idea of Pipeline processor.
- Hands on experience of making a processor from scratch.
- Got to learn how to debug effectively.

Thank You