Group 33:
Team Members:
Goutham Chadalla Manjunath
Mohit Mahesh Vaswani
Rohan Rajput
Yash Satish Lande

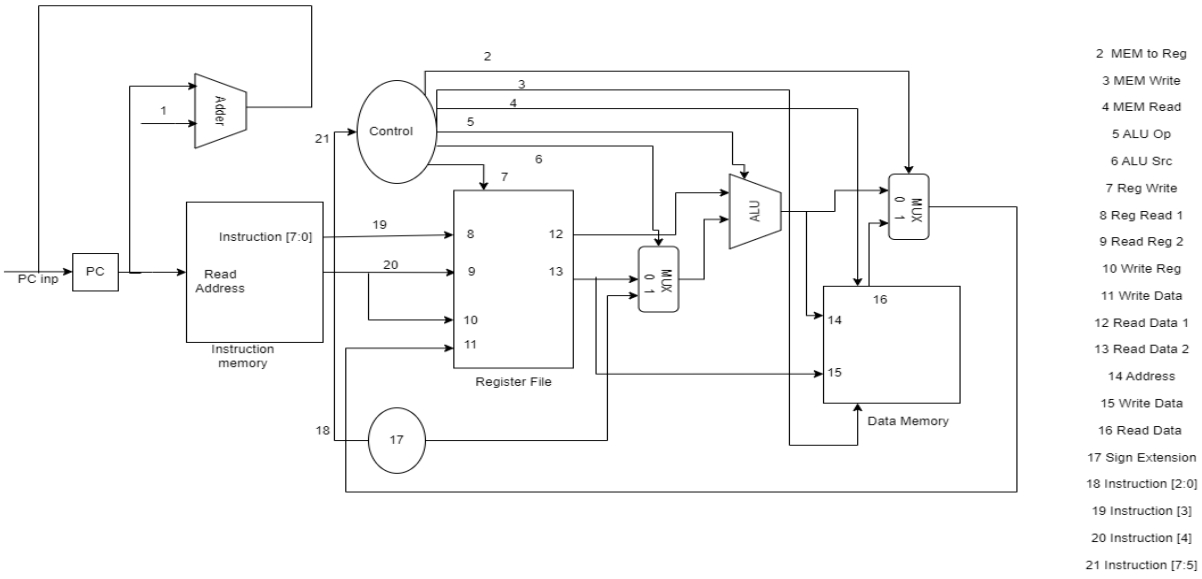
Schematic of our Datapath and Control Path

We have implemented the following instructions:

Instruction	Opcode (in binary)	Type
add	000	R-type
addi	100	I-type
sw	101	I-type
lw	110	I-type
sll	111	R-type

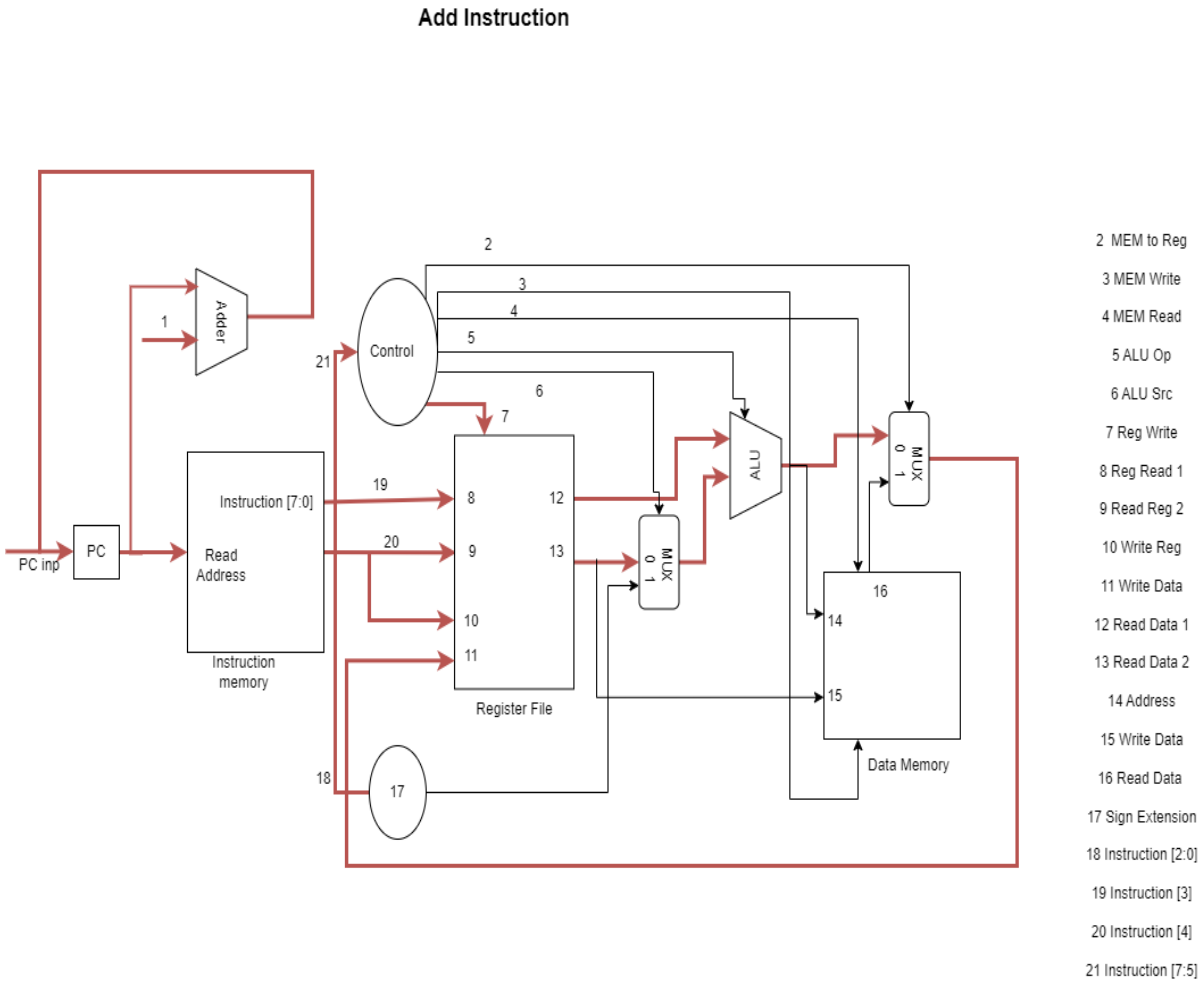
Here is the basic design of our processor:

Basic Diagram

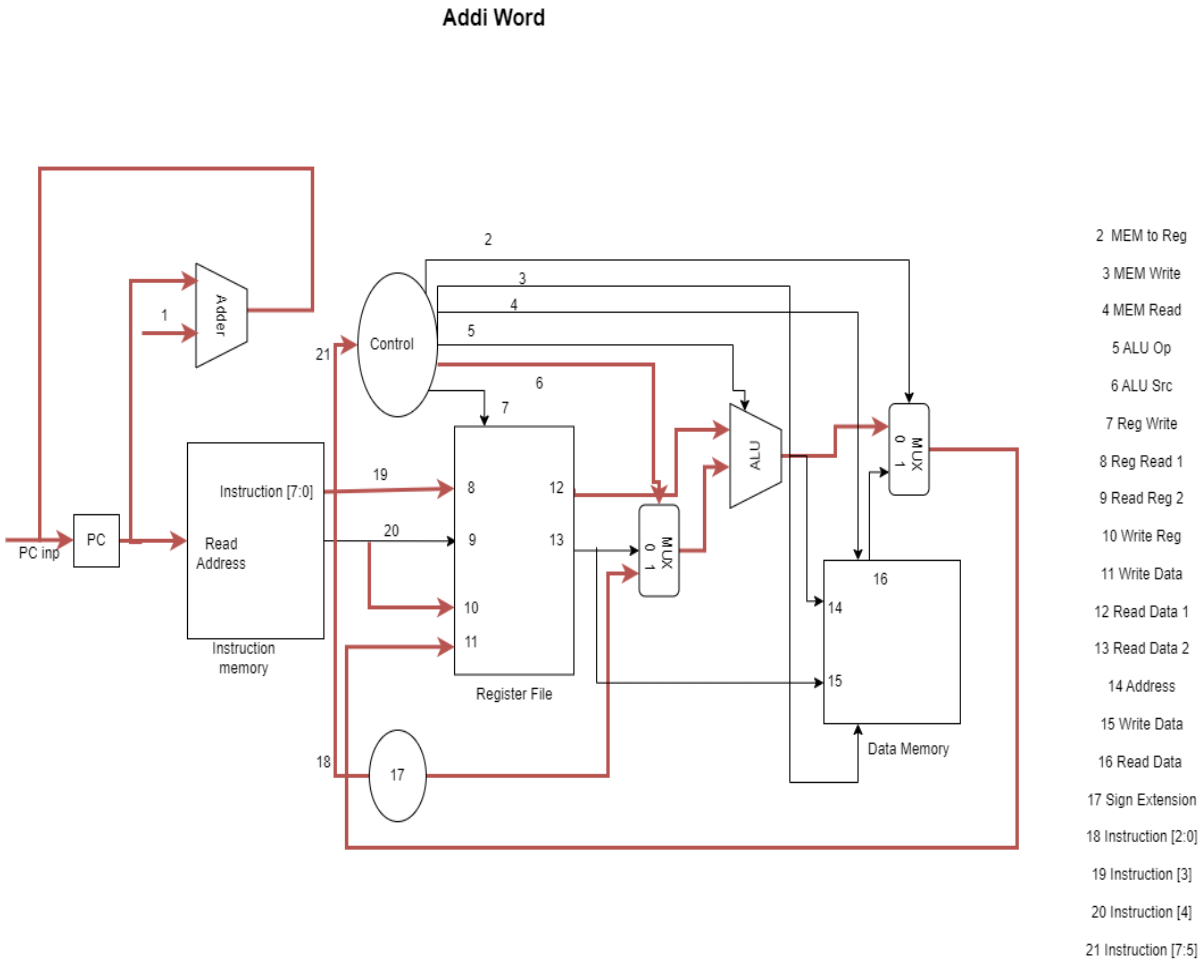


Here is the basic flow of our processor design:
In any instruction first the pc will send the instruction memory location, moving forward instruction memory will basically break down the instruction into Opcode, rt/rd(Read Data 2) , rs(Read Data 1), shamt/unused. Moving on we are sending the opcode into our control block in which we decide about the control lines. We send our Register1 and Register2 to the register file. In this file we read the data and store the respective values. Moving on we decide whether we need to use register data or the shifted data(coming from extension block) in our mux. Next we perform the ALU operation based on opcode using the data. Next we go to data memory if the instructions are LW or SW, in the last mux we decide if we need to do writeback or not.

Here is the control path and datapath of add instruction:



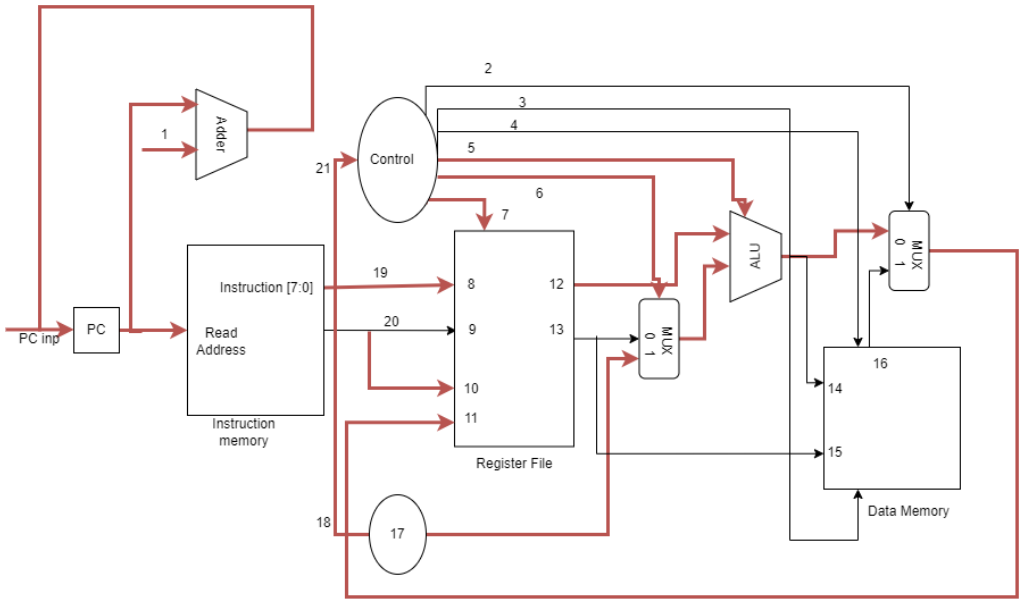
Here is the control path and datapath of addi instruction:



In this instruction we are using the sign extension.

Here is the control path and datapath of sll instruction:

Shift left Diagram



2 MEM to Reg

3 MEM Write

4 MEM Read

5 ALU Op

6 ALU Src

7 Reg Write

8 Reg Read 1

9 Read Reg 2

10 Write Reg

11 Write Data

12 Read Data 1

13 Read Data 2

14 Address

15 Write Data

16 Read Data

17 Sign Extension

18 Instruction [2:0]

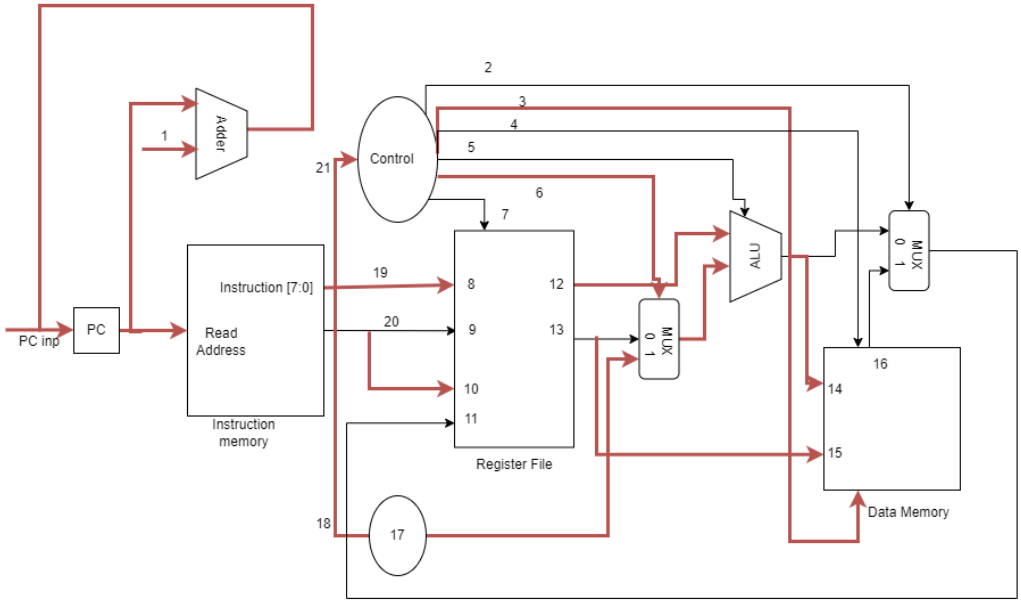
19 Instruction [3]

20 Instruction [4]

21 Instruction [7:5]

Here is the control path and datapath of sw instruction:

Store word



2 MEM to Reg

3 MEM Write

4 MEM Read

5 ALU Src

6ALU op

7 Reg Write

8 Reg Read 1

9 Read Reg 2

10 Write Reg

11 Write Data

12 Read Data 1

13 Read Data 2

14 Address

15 Write Data

16 Read Data

17 Sign Extension

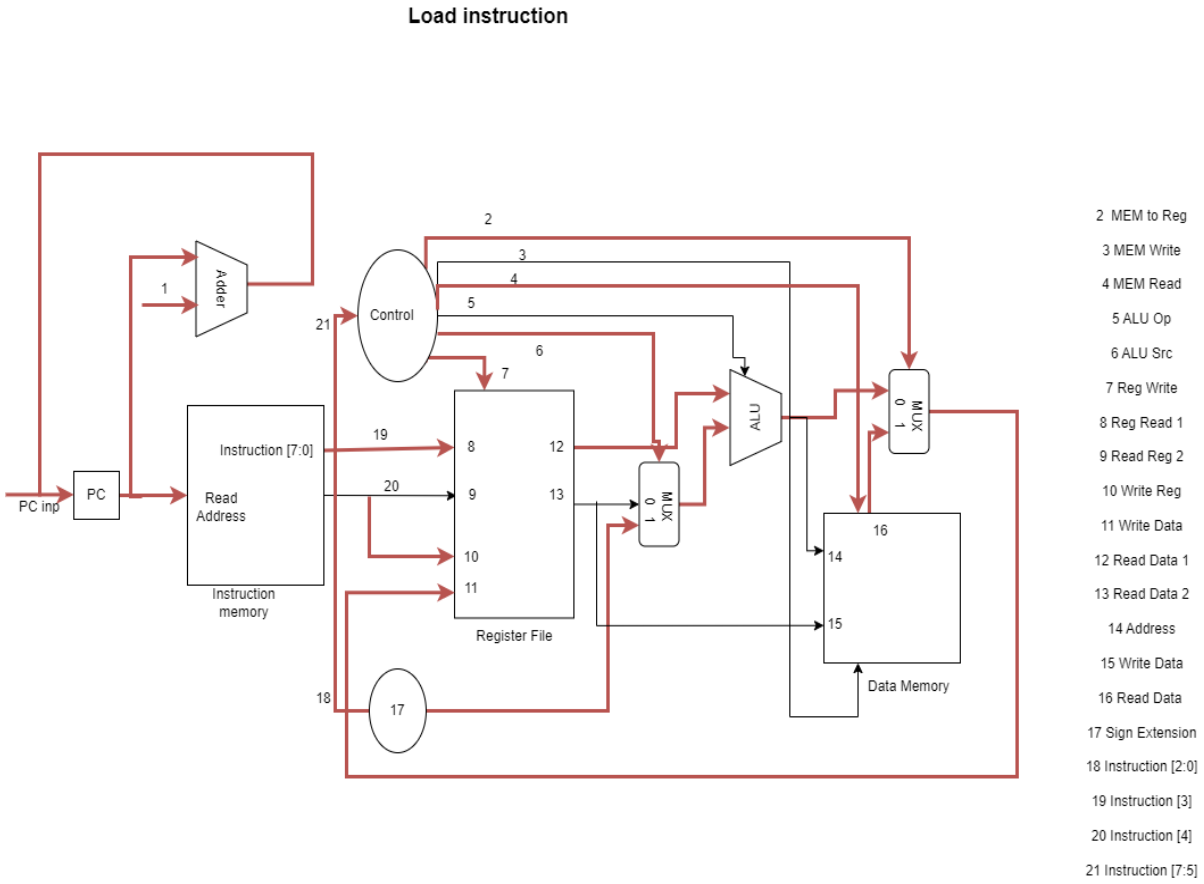
18 Instruction [2:0]

19 Instruction [3]

20 Instruction [4]

21 Instruction [7:5]

Here is the control path and datapath of lw instruction:



Short Description of each module:

PC: On each positive edge of the clock, if reset is enabled, we reset the program counter to 0 else we output the incoming program counter value.
PCAdder: On each positive clock edge we increase the program counter value by 1.
Instruction Memory: We have initialized 16 memory locations in our instruction memory module, and we have initialized the first 7 values of the memory by hardcoding the values. Based on the program counter value we fetch the instruction memory values corresponding to the PC value.
Control Unit: Based on the instruction memory opcode we are setting the control signals for specific instructions. memRead: control signal to enable the data memory read operation (0 -> read disabled, 1-> read enabled) memToReg: multiplexer select line for writing back multiplexer (0-> write back Alu output, 1-> write back data memory output) aluOp: control signal to decide which alu operation to perform in the Alu unit (0-> add operation, 1-> shift left operation) memWrite: control signal to enable the data memory write operation (0 -> write disabled, 1-> write enabled) aluSrc: multiplexer select line for ALU multiplexer (0-> take readData2(basically

value of register 2), 1-> take output of sign extension unit) regWrite: if this control signal is enabled it will write back the data to the register file

Register File: on the positive edge of clock, if reset is enabled it will set the register values to 1 and 5 respectively. Else if regWrite is enabled it will write back the write data in the destination register.

Sign Extension Unit: it extends the value of the input from 3 bits to 8 bits unsigned value.

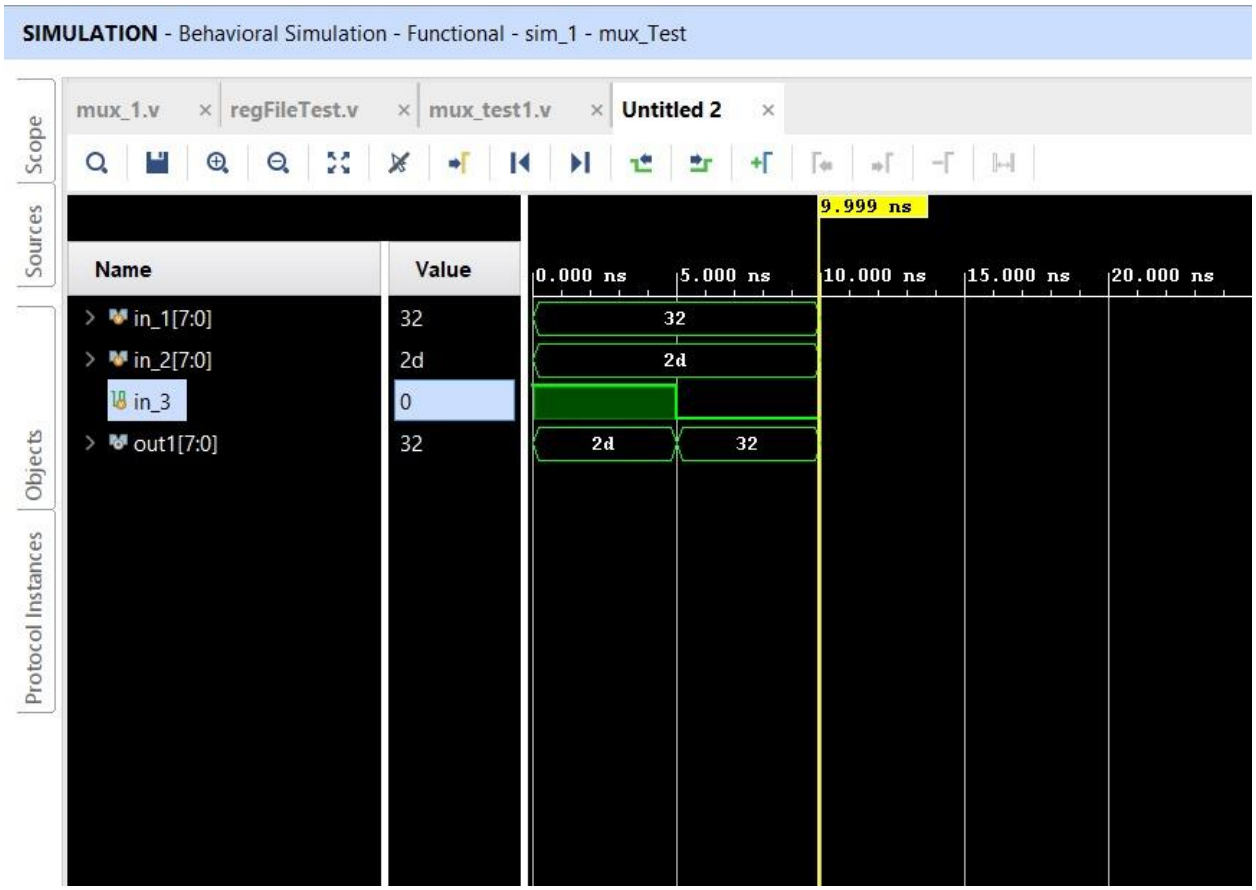
Multiplexer: based on the select line it outputs either input 1 or input2. If the select line is 0 it outputs input 1 else input 2.

Alu Unit: based on aluOp signal from control unit, either Alu will perform add operation or shift operation. If AluOp is 0 it will perform add operation else if AluOp is 1 it will perform shift left operation.

Data Memory: We have initialized 256 data memory locations in our data memory module, and we have initialized the first 16 values of the data memory by hardcoding the values. Based on control signals, memWrite and memRead we will either read data from the data memory or write it to the memory location.

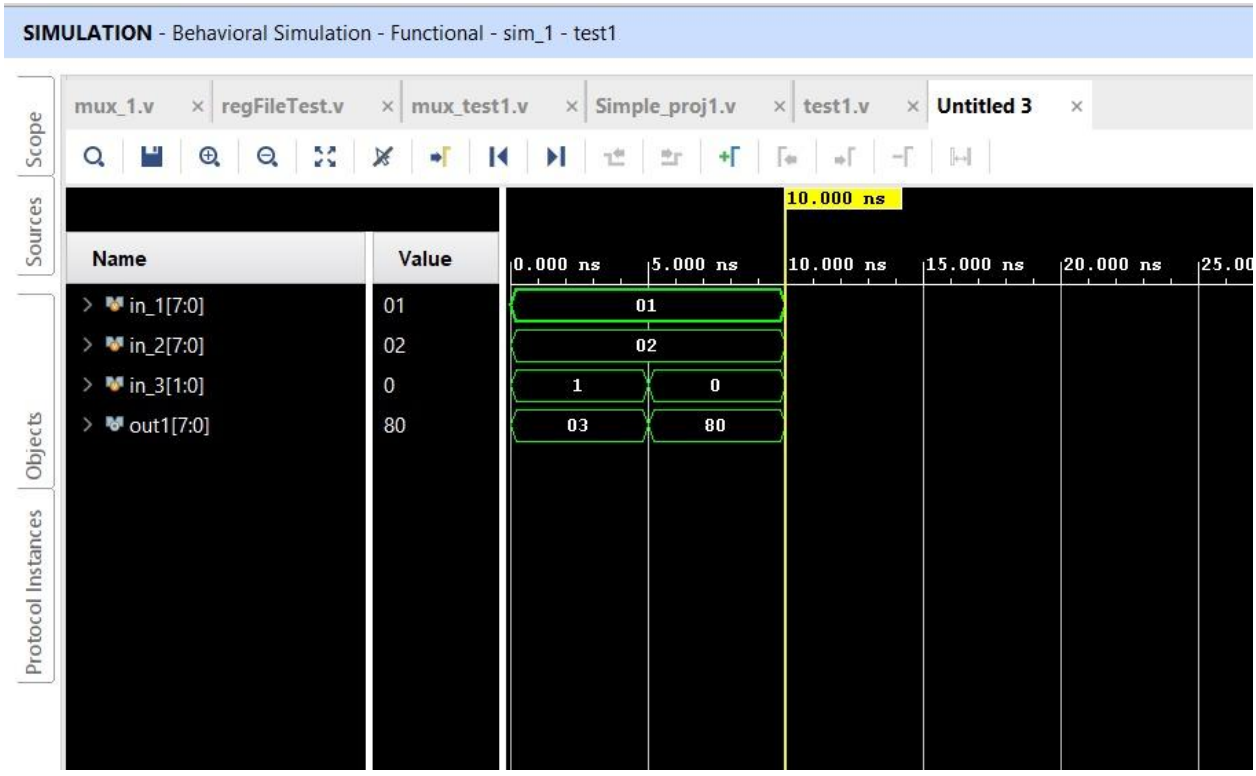
Simulation Results of Each Components:

Multiplexer:



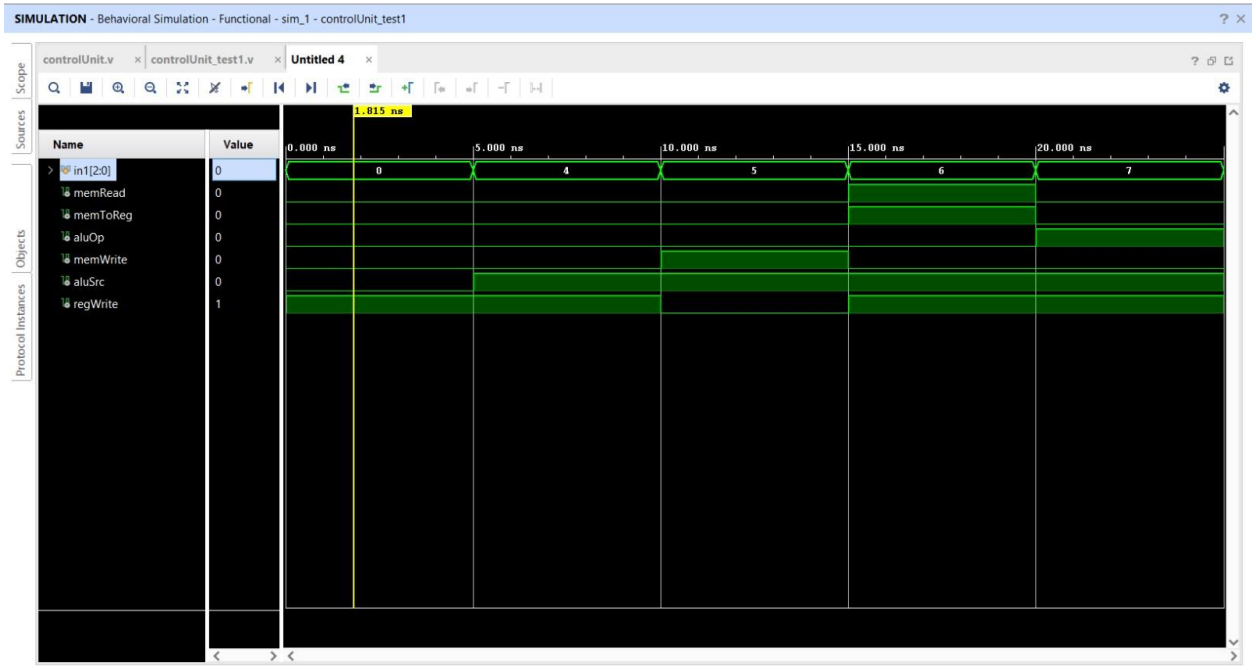
In test of Mux we have taken Input_1 as 50 and Input_2 as 45 , which are 8 bits each and Mux_select_line is 1 so we get output as 50 and for the same inputs the Mux_select_line is 0 so we get output as in_2 i.e 45.

ALU:



In ALU operation, we have Taken 3 Inputs that is input1,input2 and input3 where input1 is 1 and input_2 is 2 and flag which is either 1 or 0. Firstly, we have instruction input_1 = 1 and Input_2 = 2 and flag is 1 so according to our ALU block when Flag is 1 we do ADD operation so 1+2 = 3 and we can see this in simulation block. For the same inputs when flag is 0 we do shift left operation on input 1 i.e 00000001 --> 10000000 and output is shown in above screenshot simulation.

Control Block:



In the Control Unit, we are taking one input that is Opcode of 3 bits where according to testbench we get following data as follows:

1)For input 0 -> Regwrite as 1, Everything else will be 0.

2)For input 4 -> AluSrc as 1
Regwrite as 1, Everything else will be 0.

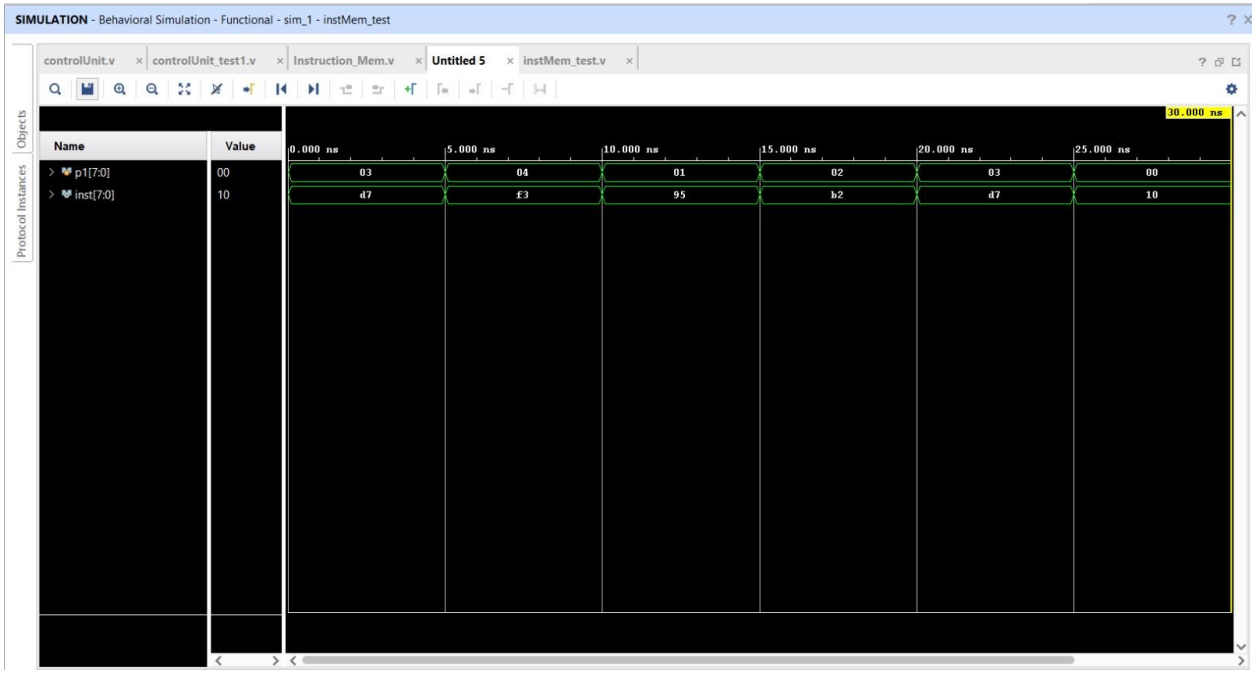
3)For input 5 -> MemWrite as 1
AluSrc as 1, Everything else will be 0.

4)For input 6 -> MemWrite as 1
MemtoReg as 1
AluSrc as 1
Regwrite as 1, Everything else will be 0.

5)For Input 7 -> Aluop as 1
AluSrc as 1
Regwrite as 1, Everything else will be 0.

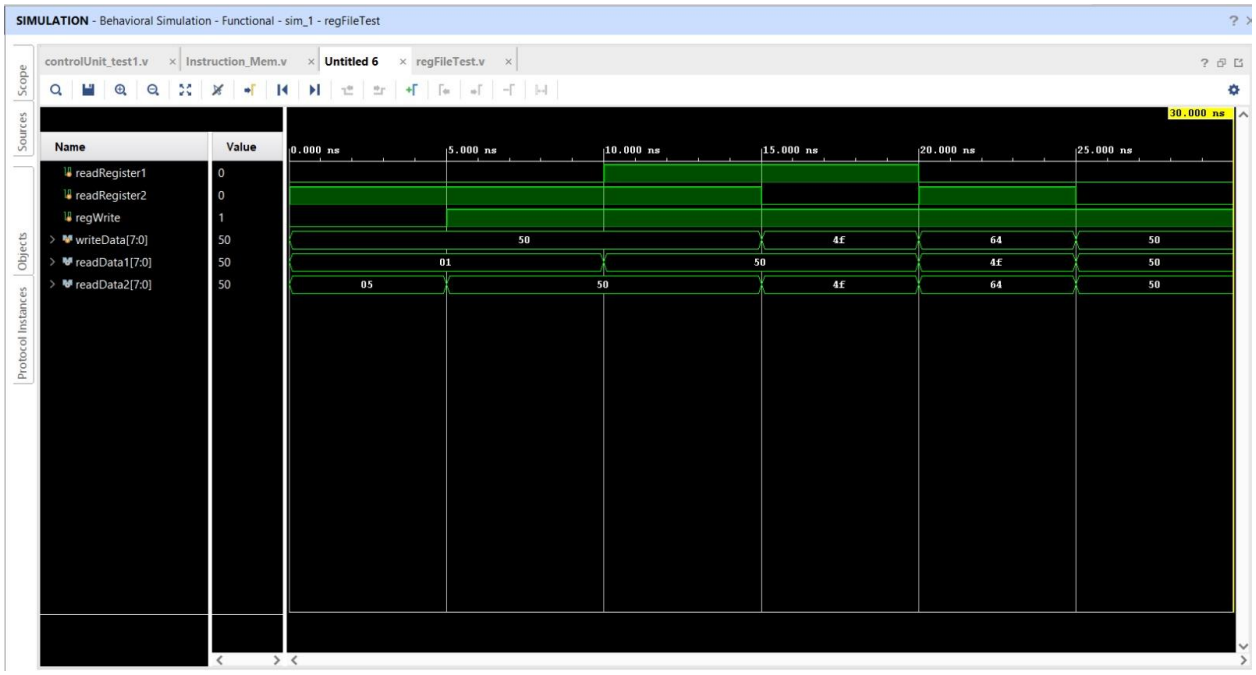
The output is simulated in the waveform and which validates as mention in Control unit file.

Instruction Memory:



In Instruction memory, we have One input that is pcOut and we output the instruction so first input is 0 and we get the instruction 0 that is -> 00010000 ADD instruction. For, instruction 1 that is -> 10110010 ADDI instruction .For, instruction 2 that is ->10110010 sw instruction. For, instruction 3 that is ->11010111 lw instruction. For instruction 4 that is -> 11110011 sll instruction. All the outputs can be verified in simulation screenshot above.

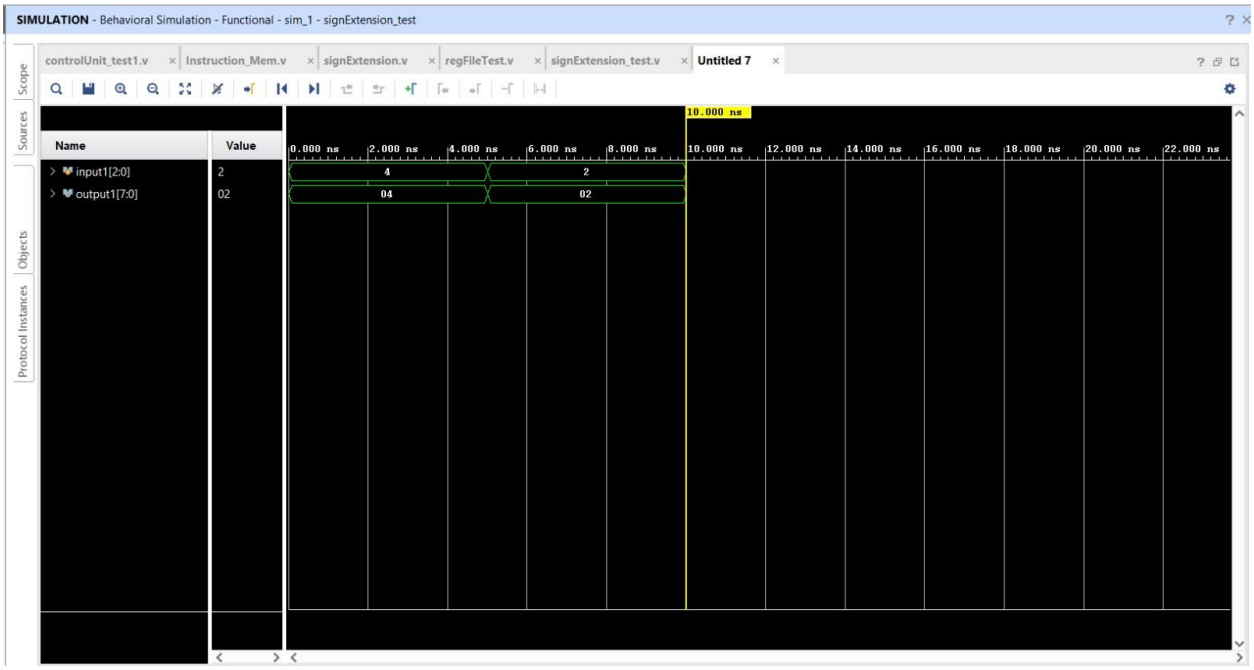
Register File:



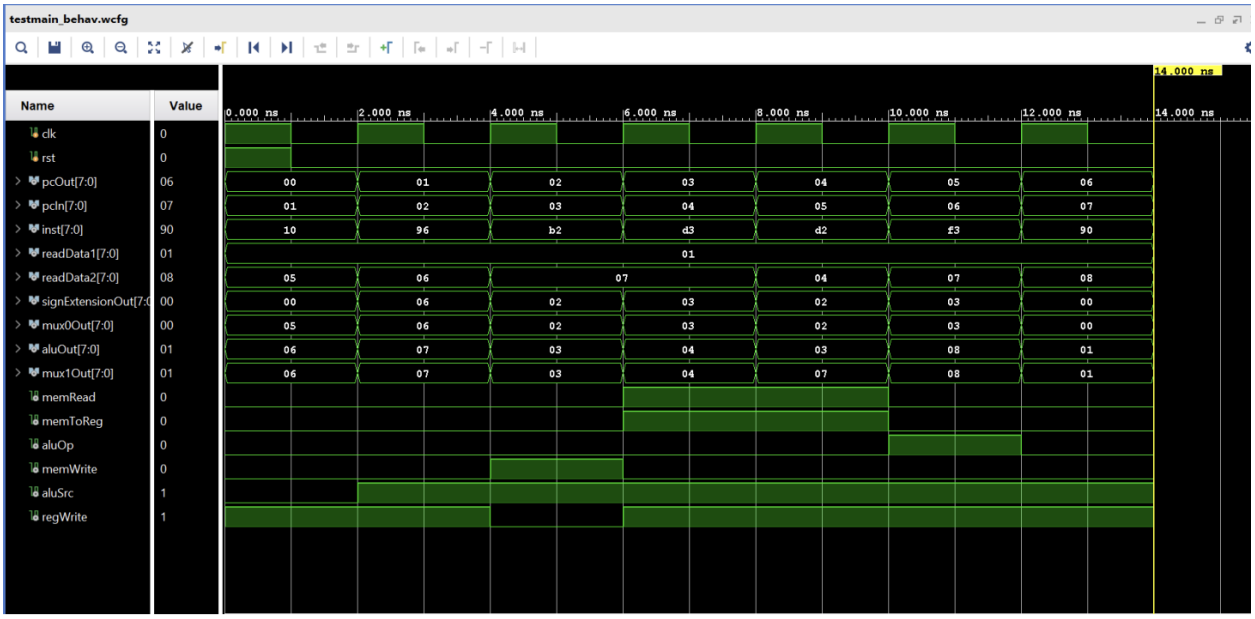
First input is reading register(rs), second input is reading as well as destination register(rt/rd).

In the register file initially register1 stores 01 and register2 stores 05 in first instruction first input is 1st register and second input is 2nd register and which basically means we would be reading from both and writing in 2nd register, initially regwrite is 0 hence no data will be written. In 2nd instruction first input is 1st register and second input is 2nd register which means if regwrite is 1 we should write data to 2nd register, in this case it is 1 so we write 50(data in write data) to register 2. In the 3rd cycle both inputs are 1 so basically destination is 1st register so we write the data 50 as regwrite is 1. Similarly in 4th cycle the first input is 2nd register and second input is 1st register so we write 4f data which is write data for that cycle in 1st register. In the 5th cycle the first input is 1st register, second input is 2nd register so we write 64 in the 2nd register which is basically the destination for that cycle. In the last cycle both inputs are 1st registers hence we write 50 which is basically the write data for that cycle in the register1.

Sign Extension block:



Final Waveform of the processor:



Description:

For the whole processor we are taking only 2 inputs clock and rest, rest all the lines are output. Initially Register0(readData1) stores 01 and Register1(readData2) stores 05, Register1 is our destination register as well in all the instructions. In first cycle our instruction is 00010000 which is basically add instruction telling us to store the value in Register1. So 01 + 05 which is 06 will be stored in Register1 in writeback stage.

In next clock cycle we have ‘addi’ instruction: 10010110. In addi we have set immediate value to 6 which is added to the value stored in Register0, value stored is 1, and the result: 06+01=07 is written back to Register1.
In next clock cycle we have ‘sw’ instruction: 10110010. In sw we have set immediate value to 2 which is added to the value stored in Register0, value stored is 1, and the result: 02+01=03 gives the index of data memory where data stored in Register1 is written at the data memory index 03.
In next clock cycle we have ‘lw’ instruction: 11010011. In lw we have set immediate value to 3 which is added to the value stored in Register0, value stored is 1, and the result: 03+01=04 gives the index of data memory. The data stored at index 04 in data memory is 4. Now this data is written back to Register1 and hence, value in Register1 become 04 as shown in the waveform above.
In next clock cycle we have used another ‘lw’ instruction: 11010010. In lw we have set immediate value to 2 which is added to the value stored in Register0, value stored is 1, and the result: 02+01=03 gives the index of data memory. The data stored at index 03 in data memory is 7. Now this data is written back to Register1 and hence, value in Register1 become 07 as shown in the waveform above.
In next clock cycle we have used ‘sll’ instruction: 11110011. In sll we have set shamt value to 3 which left shifted the value stored in Register0 by 3, the value becomes 08. The value 08 is written back to Register1 and hence, value in Register1 become 08 as shown in the waveform above.
In next clock cycle we have used ‘addi’ instruction: 10010000. In addi we have set immediate value to 0 which is added to the value stored in Register0, value stored is 1, and the result: 00+01=01 is written back to Register1 and hence, value in Register1 become 01.

Hardware Simulation:

Link to video: LINK
Screenshots: LINK

Work Distribution:

Initially we divided ourselves into two sub groups and implemented the different modules as well as test bench for that module as follows:
Goutham/Mohit: ALU, Register File, MUX, Instruction Memory
Rohan/Yash: Control Block, Program Counter/Adder , Data Memory, Sign Extension

We then merged and designed the dataflow together as it required us to critically think about each instruction.

Moving forward we implemented the hardware part and worked on the report simultaneously together, there is no individual work distribution because we worked as a team to grow forward to our goal.