CSE590: COMPUTER ARCHITECTURE

DESIGN OF AN 8-BIT NON-PIPELINED PROCESSOR USING VERILOG

SUBMITTED BY

SAI KALYAN KATTA
50292522

SAHIL SUHAS PATHAK
50289739

ABSTRACT

This project works with designing an 8-bit microprocessor using Verilog HDL by using Structural Verilog modelling.

1. Processor (CPU)

The CPU model is the top-level module which contains the inter-connections between all the above modules and has output ports like PCin, PCout, ReadData1, ReadData2, ReadData, WriteData etc.

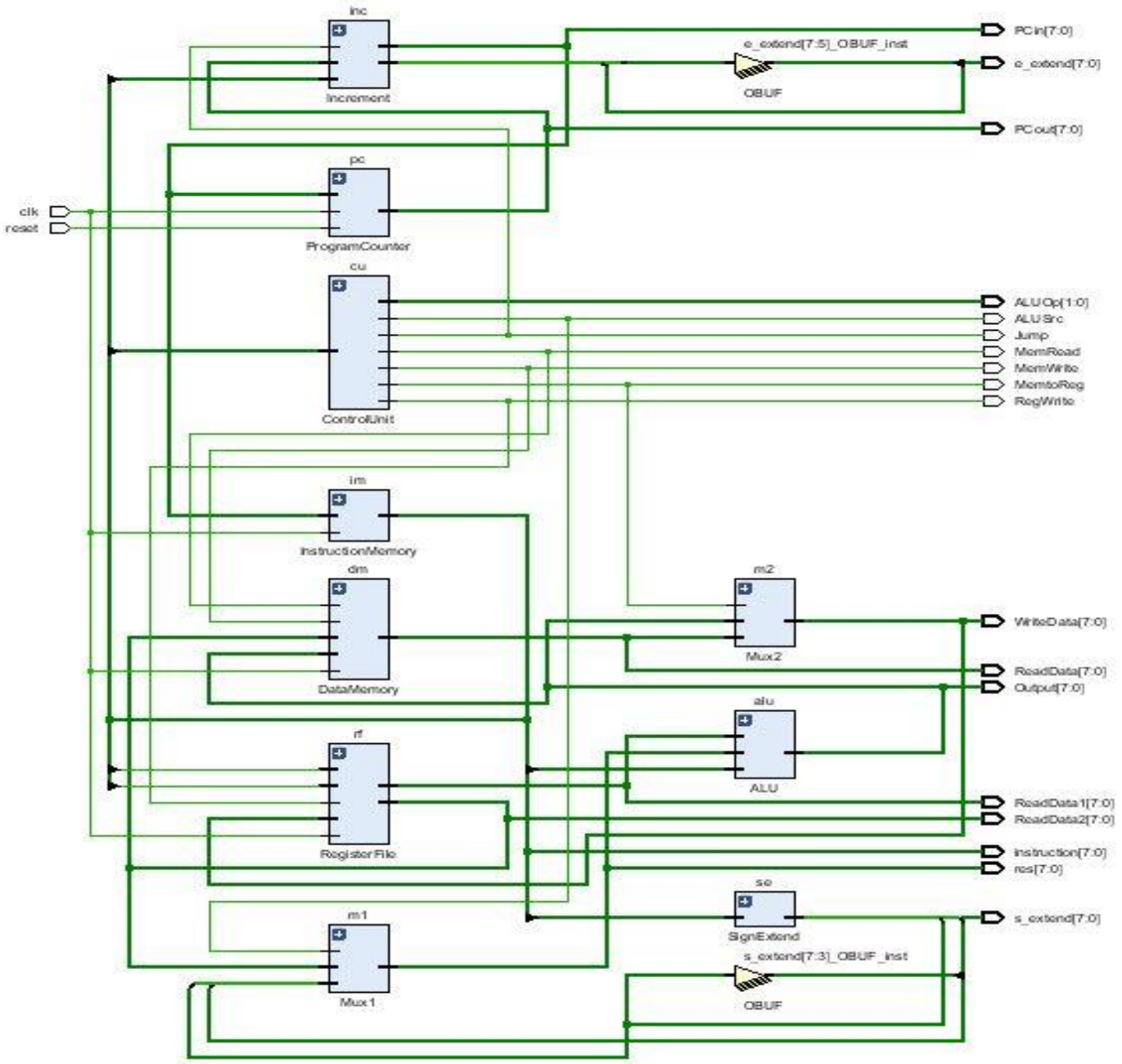

Fig. 1: Schematic Diagram

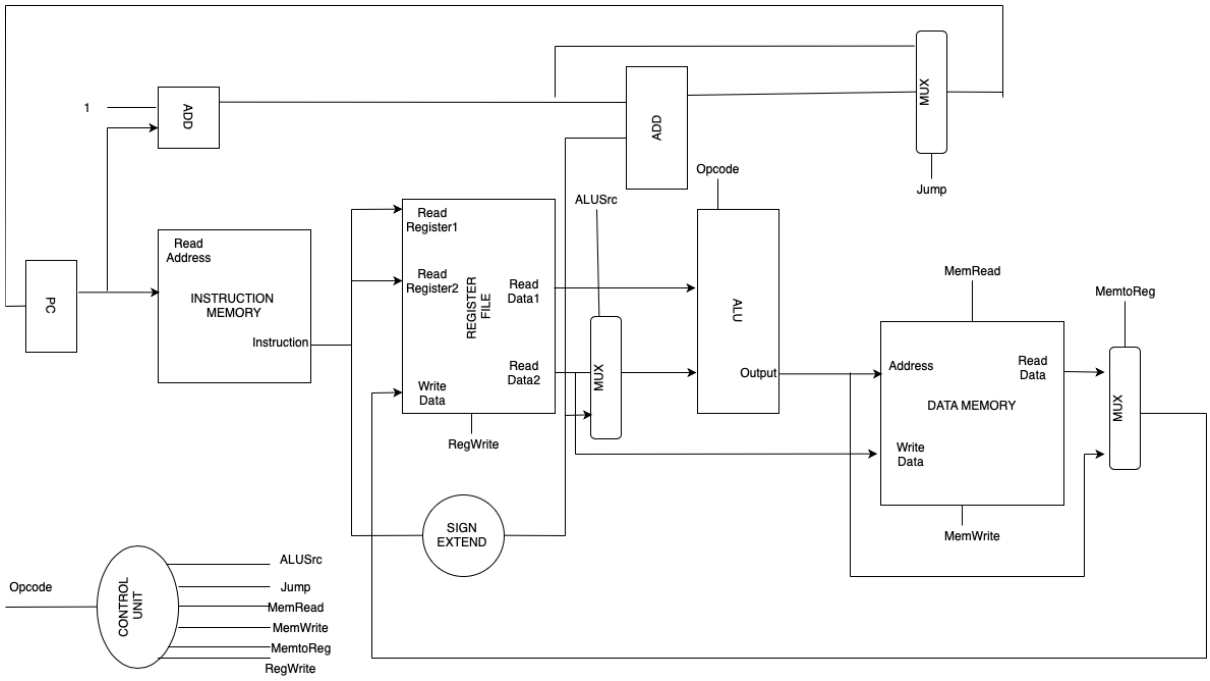

Fig. 2: Block Diagram

1.1 Control Unit

The control unit is the main module in the datapath which controls all the operations that are involved in the course of execution of instructions. The control unit takes the 3-bit opcode as the input and returns the 1 bit signals such as jump, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite and a 2-bit signal, ALUOp as the output. Here we are not the RegDst flag as we have only one destination register instead of two (in case of 32-bit processor), so there is no necessity of choosing between two destinations so that is why we are not considering the multiplexer module between the instruction memory and the Register File.

Instruction	lw	sw	add	addi	sub	jump
ALUSrc	1	1	0	1	0	0
MemtoReg	1	0	0	0	0	0
RegWrite	1	0	1	1	1	0
MemRead	1	0	0	0	0	0
MemWrite	0	1	0	0	0	0
Jump	0	0	0	0	0	1
ALUOp	00	00	10	11	10	00

Table 1: Control unit output

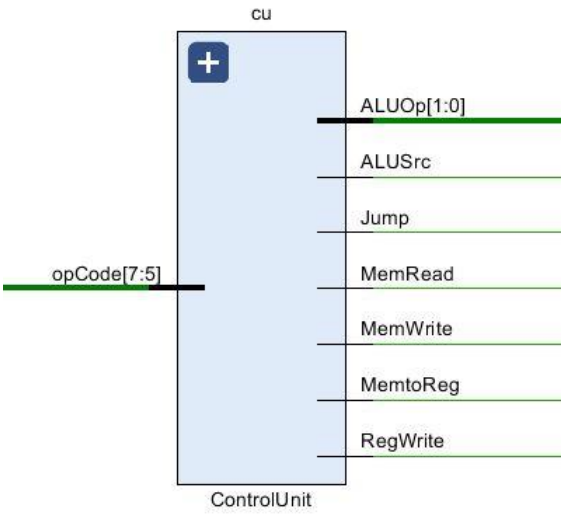

Fig. 1.1: Control Unit

1.2 Register File

The register file module deals with the writing and reading the values from the registers which has the inputs such as the RegWrite, ReadRegister1, ReadRegister2, WriteData, and outputs as ReadData1, ReadData2. We defined a memory array with 3 values in it as 2, 8, 20. In the module, we check for the RegWrite signal and save the value from WriteData to the register memory and we take the values from the register memory and store them into the ReadData1 and ReadData2.

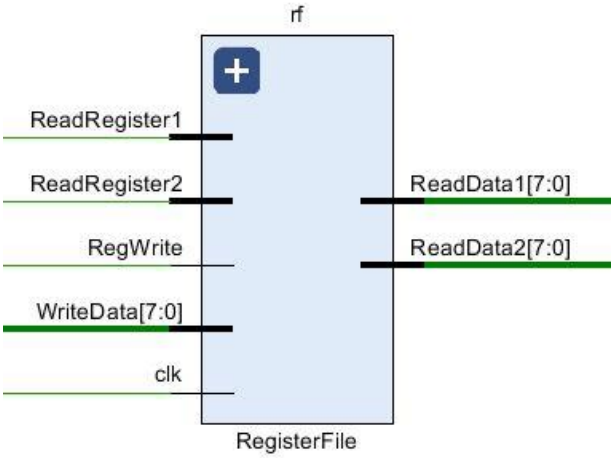

Fig. 1.2: Register File

1.3 Data Memory

The DataMemory module deals with the writing and reading the values from the main memory which has the inputs such as the address, MemWrite, MemRead, WriteData, and output as ReadData. We defined a memory array with 3 values in it as 5, 6,7. In the module, we check for the MemWrite signal and save the value from WriteData to the main memory and we read the values from the memory and store them into the ReadData register which we defined when the MemRead flag is set.

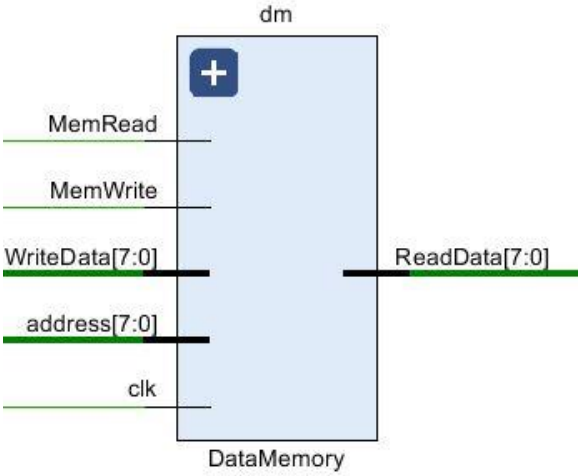

Figure1.3: Data Memory

1.4 Instruction Memory

The Instruction Memory module deals with the fetching of instructions which were saved in an array (instruction memory). This module has address as the input and the instruction is returned as the output.

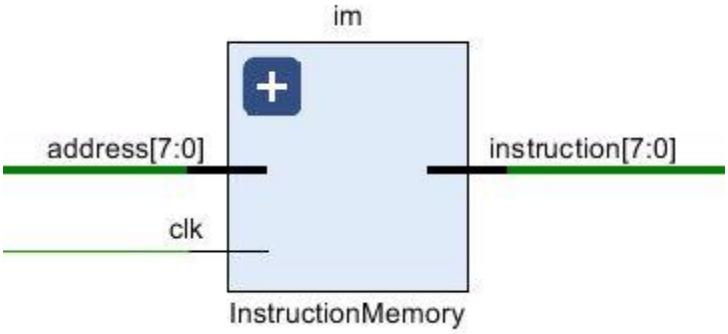

Fig. 1.4: Instruction Memory

1.5 Program Counter

This module performs the computation of PC = PC+1 which is used in pointing to the next instruction.

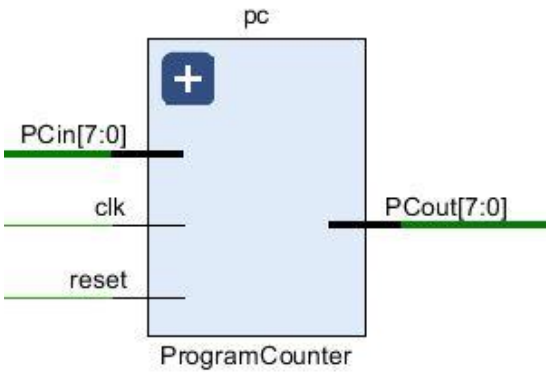

Fig. 1.5: Program Counter

1.6 Sign Extender

The sign extender module is mainly used while executing the I type instructions such as the lw (load word) and sw (store word). This module takes the 3-bit immediate value from the instruction and extends it to the 8-bit value.

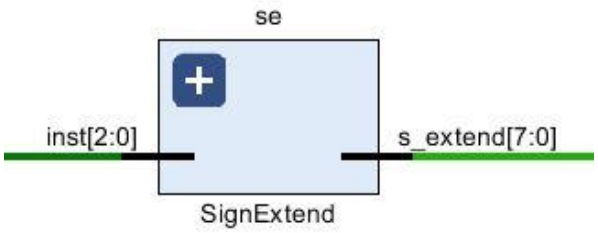

Fig. 1.6: Sign Extender

1.7 Multiplexer 1

The multiplexer 1 has the ReadData2 and the extended immediate value from the sign extender as the inputs and the ALUSrc signal from the control unit as the select bit. So, when the instruction is an I type instruction, the ALUSrc value set to 1 so the immediate value is sent as the output from the multiplexer. In the other cases as the ALUSrc is 0, the ReadData2 value is passed to the ALU from the multiplexer.

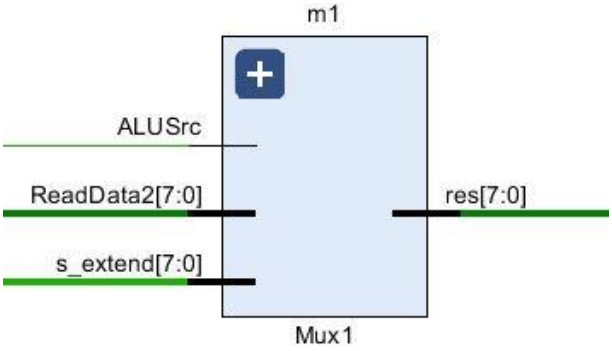

Fig. 1.7: Multiplexer 1

1.8 ALU

ALU is the module where all the arithmetic operations are done. The ALU module takes the opcode, ReadData1 and the output of the multiplexer 1 as the inputs and returns the desired arithmetic result as the output.
When the opcode is 010: result is the addition of both ReadData1 and output from multiplexer 1.
When the opcode is 100: result is the subtraction of multiplexer 1 output from ReadData1.
When the opcode is 000: result is the addition of both ReadData1 and output from multiplexer 1.
When the opcode is 001: result is the addition of both ReadData1 and output from multiplexer 1.
When the opcode is 011: result is the addition of both ReadData1 and output from multiplexer 1.
For the jump instruction (opcode:101) there is no use of the ALU module.

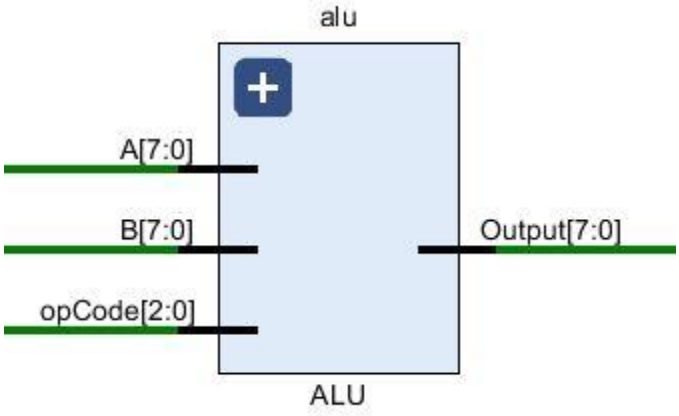

Fig. 1.8: ALU

1.9 Multiplexer 2

The multiplexer 2 has the ReadData from memory and the output result value from the ALU as the inputs and the MemtoReg signal from the control unit as the select bit. So, when the instruction is an lw instruction, the MemtoReg value set to 1 so the ReadData value from the DataMemory is sent as the output from the multiplexer to the Register File as the Write Data. In the other cases as the MemtoReg is 0, the output value from the ALU is passed to the Register File from the multiplexer as the Write Data.

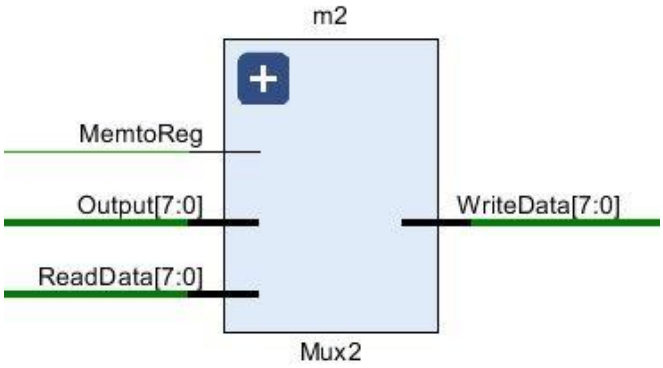

Fig. 1.9: Multiplexer 2

2. Simulation Results

Executing add, sub, addi, lw & sw together in a loop

- The executing starts when PCin is 8'b1. The first instruction is "sub r1, r2"
- Registers r1 and r2, initially have values 8 and 2 respectively.
- As the instruction (10001000) executes, 6 is stored back as result in r1 since we are considering r1 as the destination. This concept is followed for every other instruction.
- The WriteData_Reg value (6) indicates that we need to store the value into r1 as a result.
- The next instruction (01001000) is executed with updated value of r1 as 6 and r2 remaining 2. Since it is an add instruction, the Output (ALU Output) is 8. Again, the WriteData_Reg value (8) indicates that we need to store the value 8 into r1 as a result.
- Similarly, the addition immediate instruction (01101100) is executed. The lower 3 bits are sign extended to 8 bits (00000100). Then the value of r1 and sign extended bits is added. (8 + 4 = 12). Then the value 12 is stored as a result in r1.
- Similarly, load instruction (00001000) executes. The final 3-bits 000 are extended to 8-bit value and then the ALU performs the addition of r1 and the sign extended bits [(12 + 0) = 12]. This value 12 will now be used as an address and DataMemory [12] has a value of 10 which is declared in the DataMemory module. Thus, value 10 will be stored in the RegisterFile (r1).
- When the next instruction store (00101001) is executed, r1 has the value 10 and r2 remains 2. The final 3 (001) bits are sign extended to 8 bits and added to ReadData1 register and the ALU will add them. The Output of the ALU will be considered as an address i.e (10 + 1 = 11) and DataMemory [11] will store the value of the register.

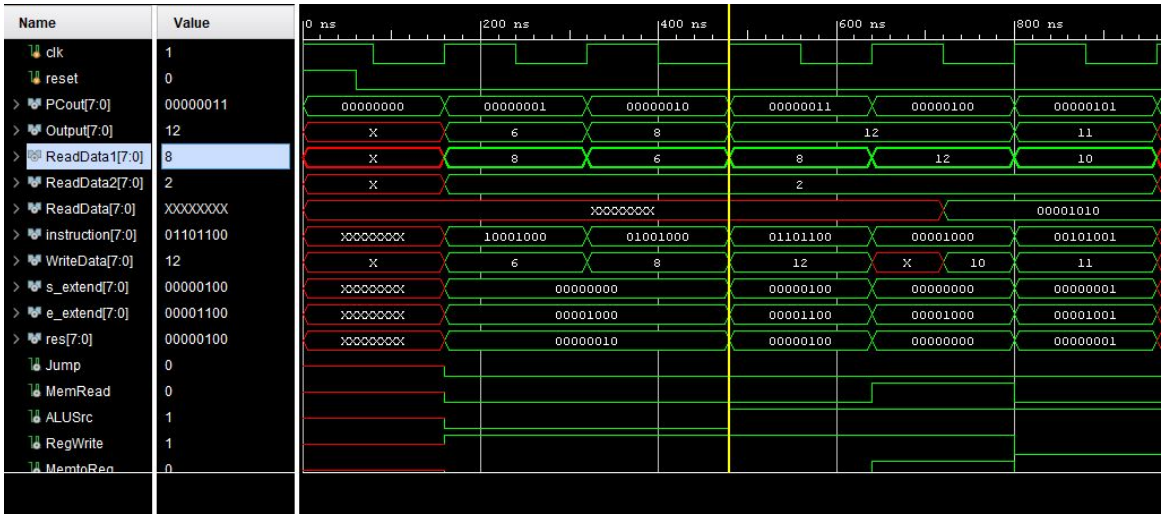

Fig. 2: Results after executing all instructions at once in a loop

2.1 Executing sub instruction

Instruction is (10001000). The two registers have the value 8 and 2 respectively. The result (ALU Output) is 6.

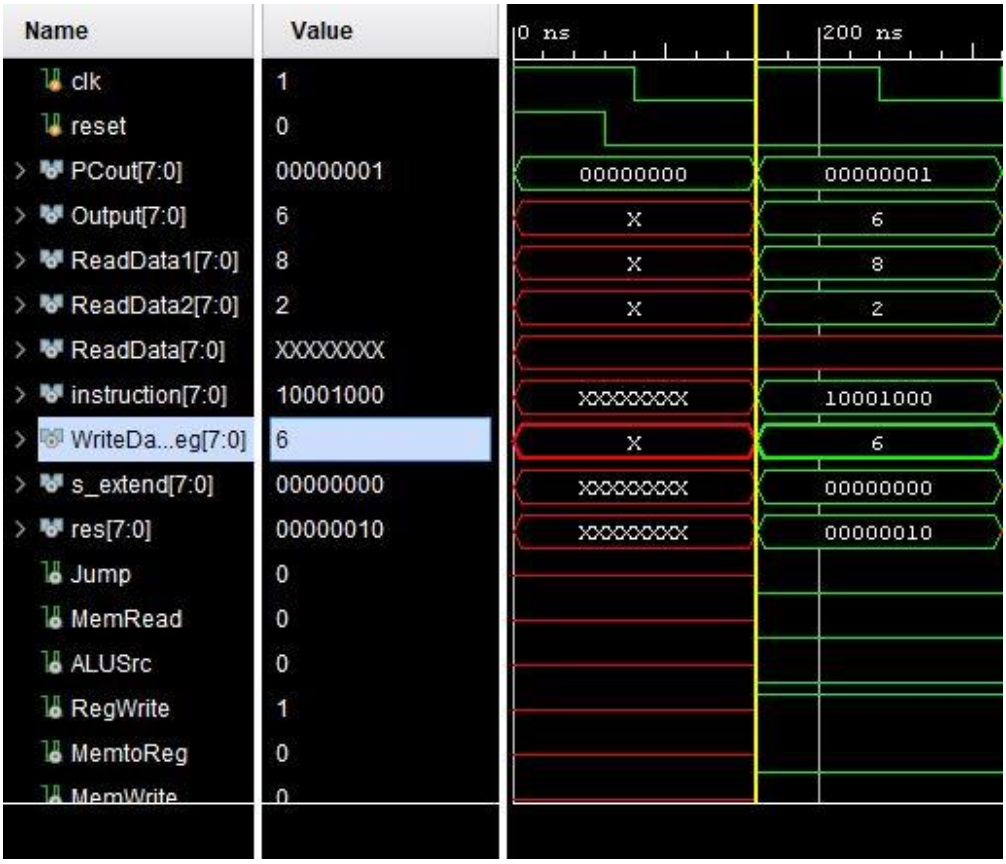

Fig. 3: Results after executing sub instruction

2.2 Executing add instruction

Instruction is (01001000). The two registers have the value 8 and 2 respectively. The result (ALU Output) is 10.

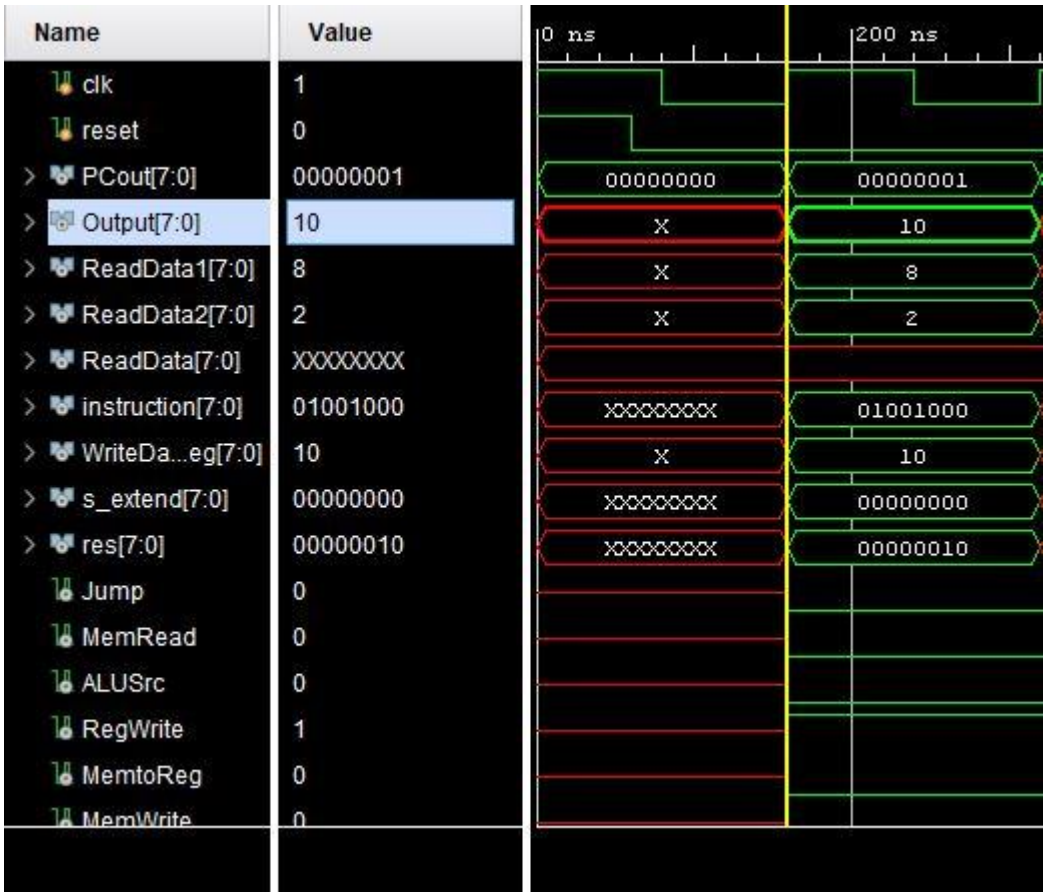

Fig. 4: Results after executing add instruction

2.3 Executing addi instruction

Instruction is (01101100). The register r1 has the value 8. The sign extened lower 3 bits correspond to a value 4. Thus the result is 12.

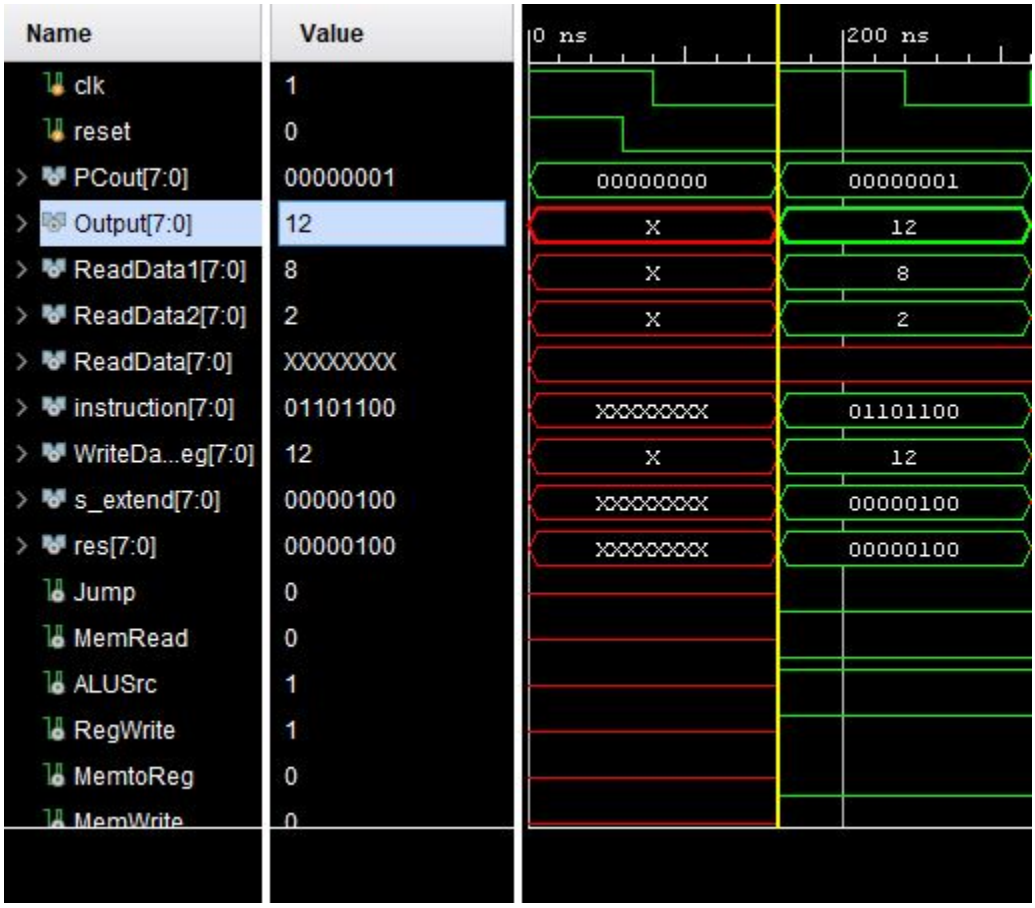

Fig. 5: Results after executing addi instruction

2.4 Executing lw instruction

Instruction is (00001000). The final 3-bits 000 are extended to 8-bit value and then the ALU performs the addition of r1 and the sign extended bits [(8 + 0) = 8]. This value 8 will now be used as an address and DataMemory [8] has a value of 20 which is declared in the DataMemory module. Thus, value 20 will be stored in the RegisterFile (r1) which is indicated by WriteData_Reg. The value that is read from the DataMemory is 20 indicated by ReadData.

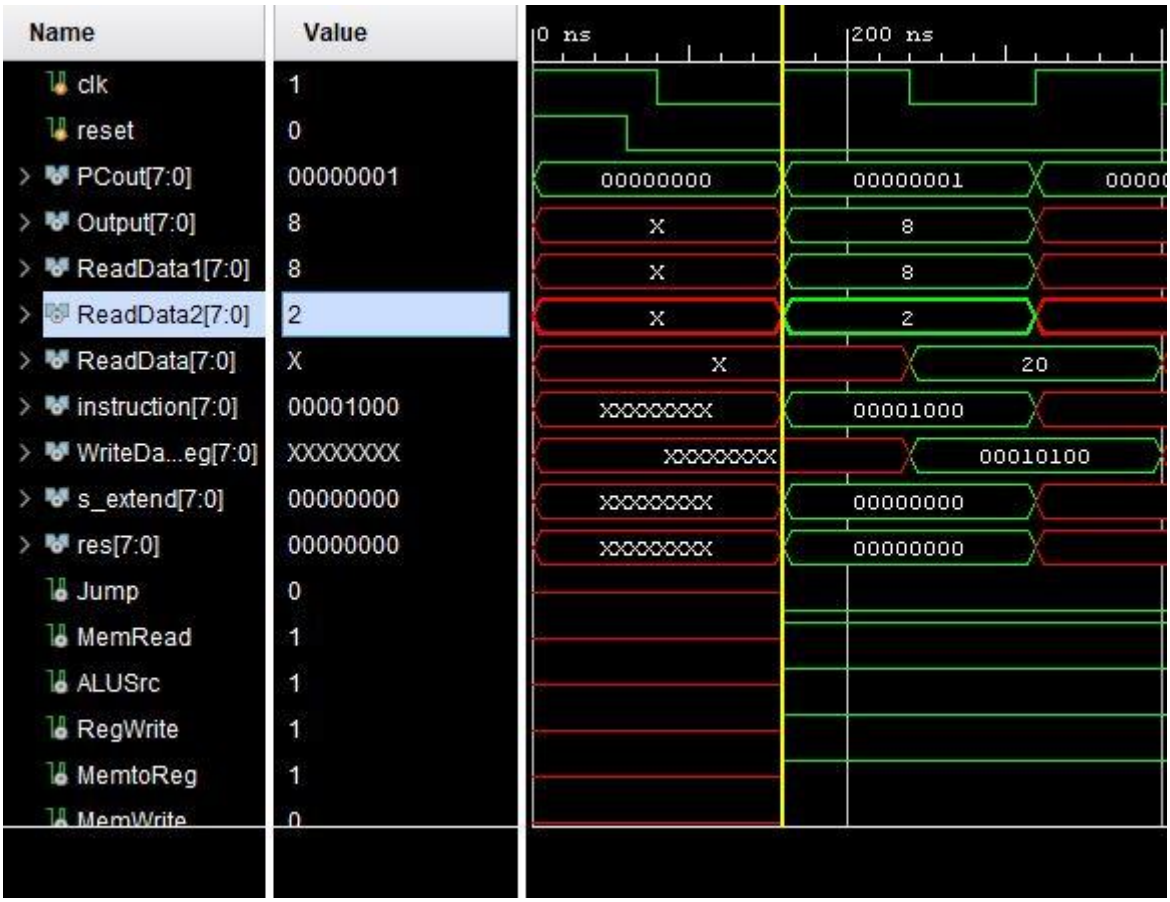

Fig. 6: Results after executing lw instruction

2.5 Executing sw instruction

The instruction is (00101001). r1 has the value 8 and r2 is 2. The final 3 (001) bits are sign extended to 8 bits and added to ReadData1 register and the ALU will add them. The Output of the ALU will be considered as an address i.e (8 + 1 = 9) and DataMemory [9] will store the value of the register.

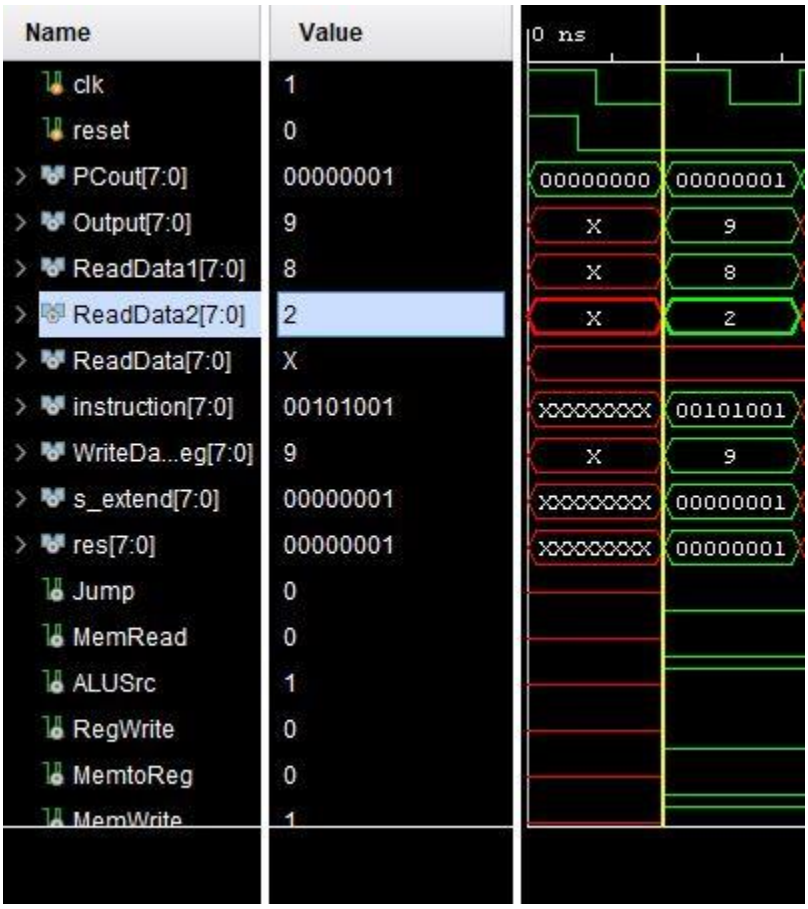

Fig. 7: Results after executing sw instruction

2.6 Executing jmp instruction

The instruction is (1010000). The lower 5 bits are sign extended to 8 bits and the address generated is considered as the next instruction that needs to be executed.

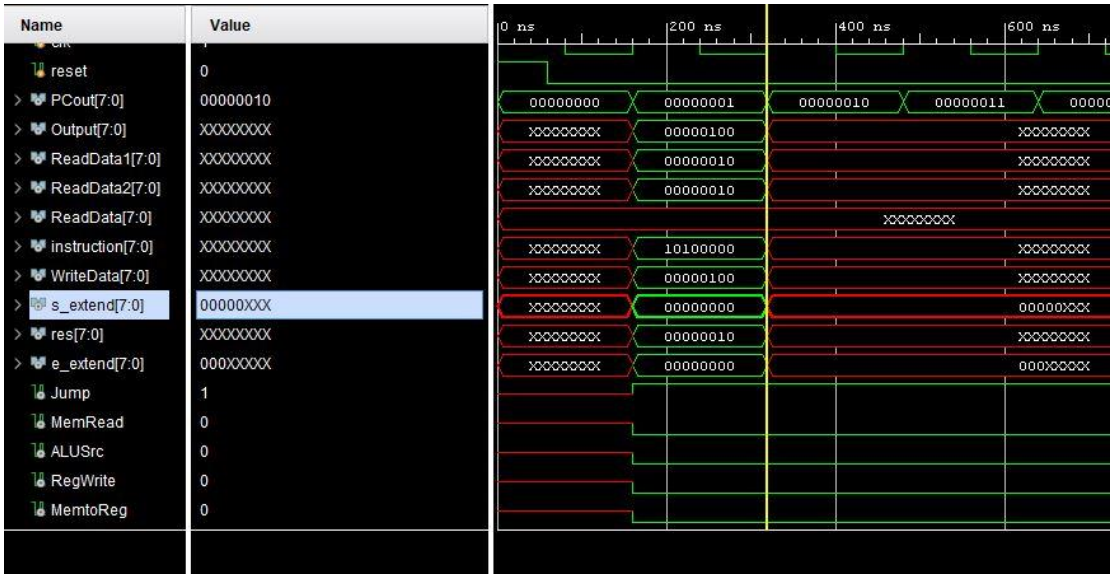

Fig. 8: Results after executing jmp instruction

3 Contribution and References

We basically had equal contribution in all the tasks involved in the course of the project. To be precise the modules that we individually handled were,

Sai Kalyan Katta: Control Unit, Multiplexers, Register File, ALU.
Sahil Suhas Pathak: Data Memory, Sign Extender, Program Counter, Instruction Memory, CPU and Testbench.

Reference #1: Understanding 32-bit Processor Implementation:

https://github.com/DiabloBlood/Single-cycle-MIPS-processor

Reference #2: Understanding the concept behind the instructions:

“Computer Organization and Design – The Hardware/Software Interface” by Patterson and Hennessy, 4th ed.”