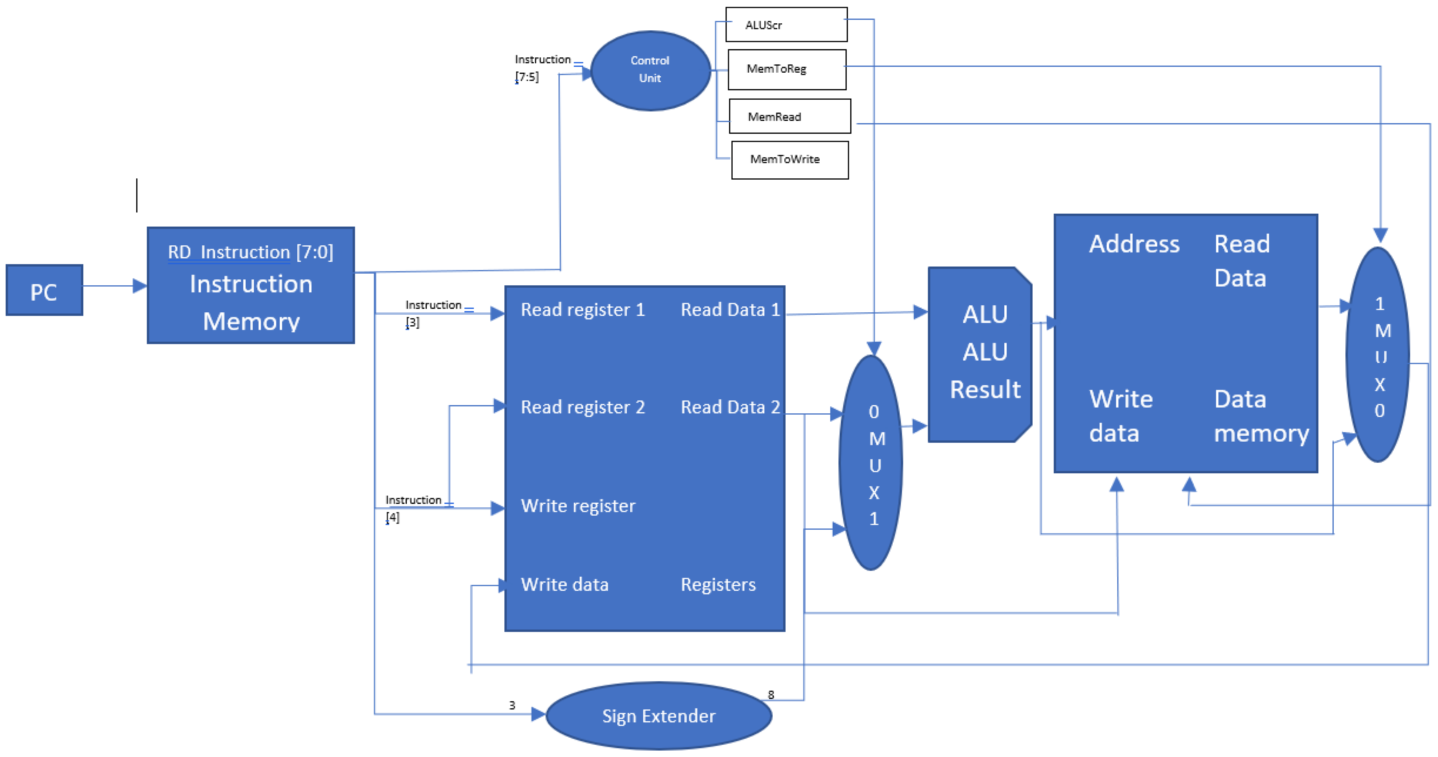
Programming Assignment 1 Report - DESIGN OF AN 8-BIT NON-PIPELINED PROCESSOR USING VERILOG

CSE490
Kyle Shuttleworth
Rishit Parihar

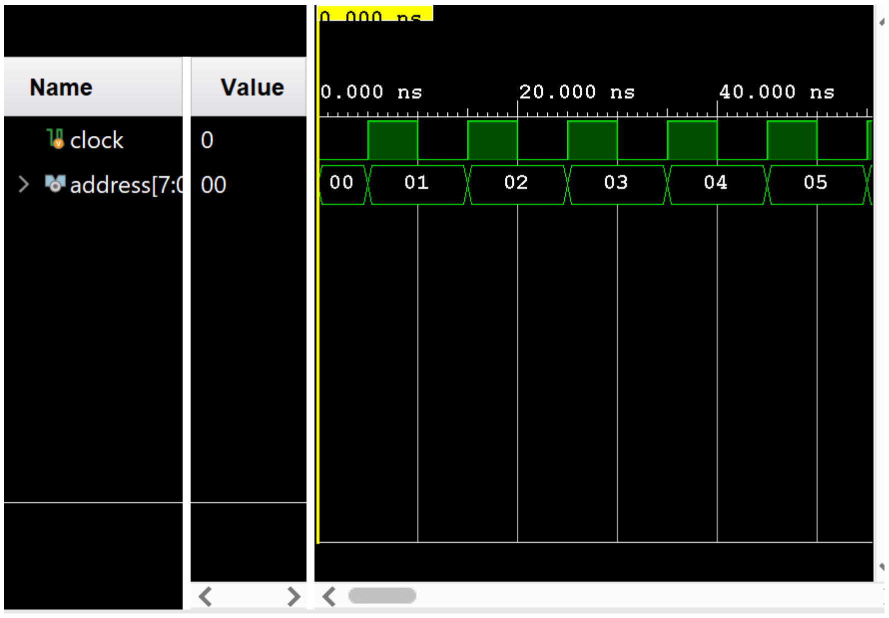
GitHub: https://github.com/kyleshut/Design-of-an-8-bit-non-pipelined-processor-using-verilog

Schematic:


Instruction [7:5]
Control Unit
ALUScr
MemToReg
MemRead
MemToWrite
PC
RD Instruction [7:0]
Instruction Memory
Instruction [3]
Instruction [4]
Read register 1
Read register 2
Write register
Write data
Read Data 1
Read Data 2
Registers
0 MUX 1
ALU
ALU Result
Address
Read Data
Write data
Data memory
1 MUX 0
3
8
Sign Extender

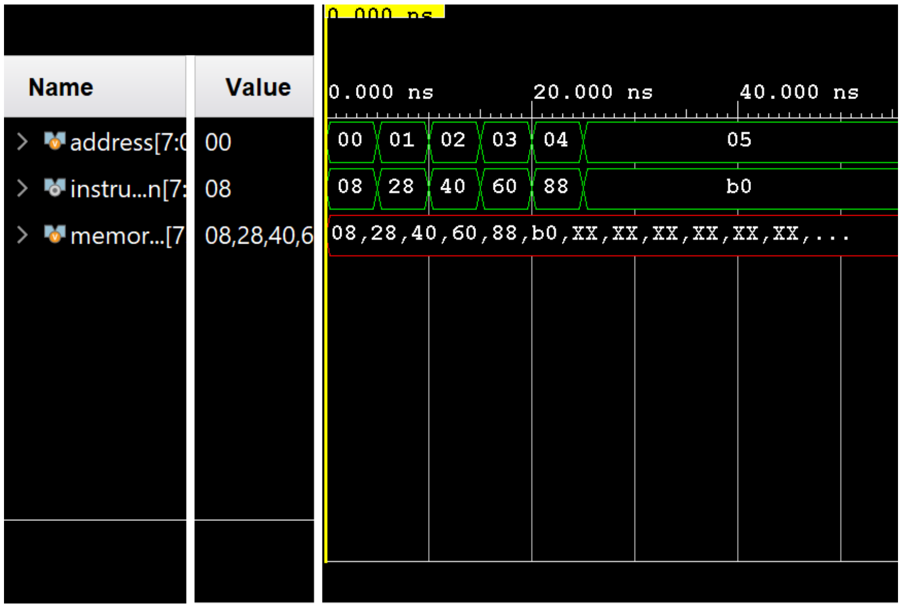
Program Counter: The program counter increments the address to the next instruction that will be executed.

Simulation Results: As you can see in the simulation below the address is incremented by 1 at every posedge which is what we expect the program counter to do.



Instruction Memory: The Instruction Memory is the memory where instructions are fetched. It takes in the address passed by the program counter and retrieves the instruction that was stored at that address and then outputs that instruction

Simulation Results: I initialized the instruction memory to hold the 6 instructions given to us in the project handout. As you can see in the simulation below everytime the address is incremented the instruction memory outputs the instruction at that address in the instruction memory.



Name	Value	Waveform
address[7:0	00	00, 01, 02, 03, 04, 05
instru...n[7:	08	08, 28, 40, 60, 88, b0
memor...[7	08,28,40,6	08,28,40,60,88,b0,XX,XX,XX,XX,XX,XX,...

Register File: The Register file contains a set of registers and those registers can be read or written to.

Simulation Results: In the simulation below at every posedge data from data write data is written into the write register and initial inputs for the first clock cycle in test bench are as follows:
clock = 0;
registers[0] = 8'b00001000; // 8
registers[1] = 8'b00001001; // 9
read_reg1 = 0; // $t0
read_reg2 = 1; // $t1
write_reg = read_reg1; $t0
write_data = 8'b00000101; // 5

After delay i turn clock to 1 and as you can see by simulation below:

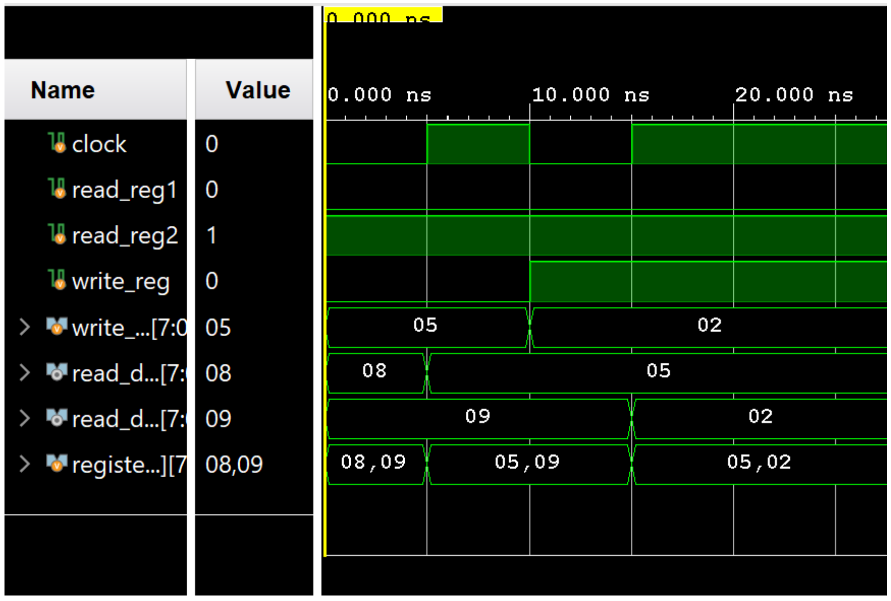
The value of write_data was written into the registers at the write_reg index.

After delay I then turn clock back to 0 and initialize inputs

clock = 0;
write_reg = read_reg2;
write_data = 8'b00000010;

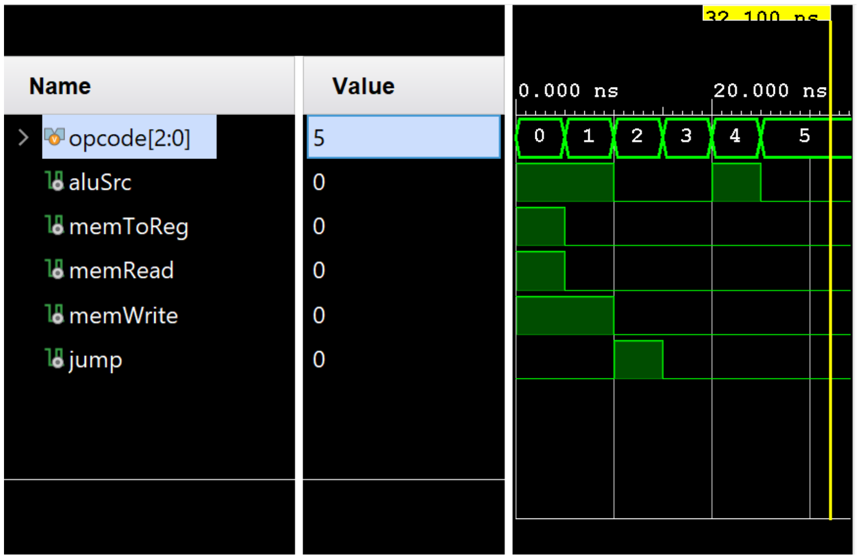
After delay i turn clock to 1 and as you can see by simulation below:

The value of write was written into the registers at the write_reg index.



Control: The control unit is the brains of the datapath. It directs the datapath on what operations it should take for different instructions based on the opcode that is passed as an input to the control.

Simulation:

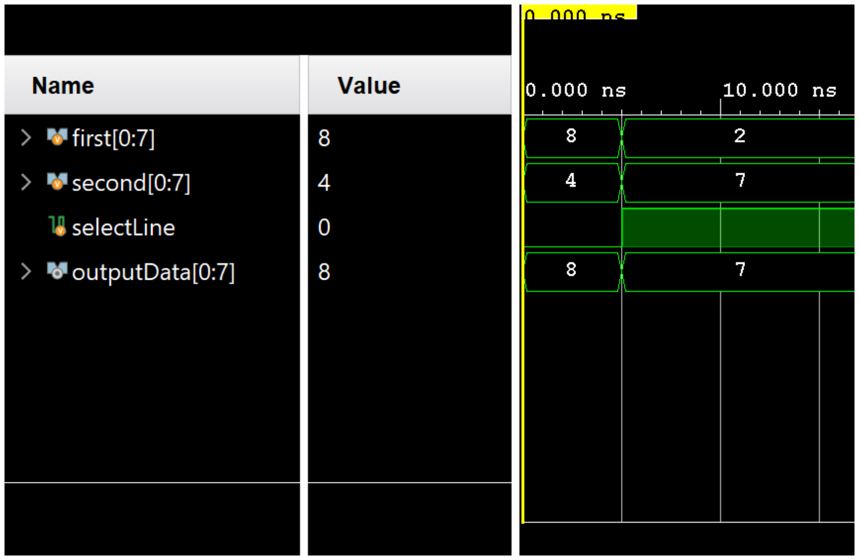


Name	Value
opcode[2:0]	5
aluSrc	0
memToReg	0
memRead	0
memWrite	0
jump	0

As you can see by the simulation as we pass in different opcode values the signals will change to the designated signal values for that specific opcode

MUX: the mux unit in our project is 2 to 1. Therefore mux take's in 2 inputs and then outputs one of them depending on the value of a signal.

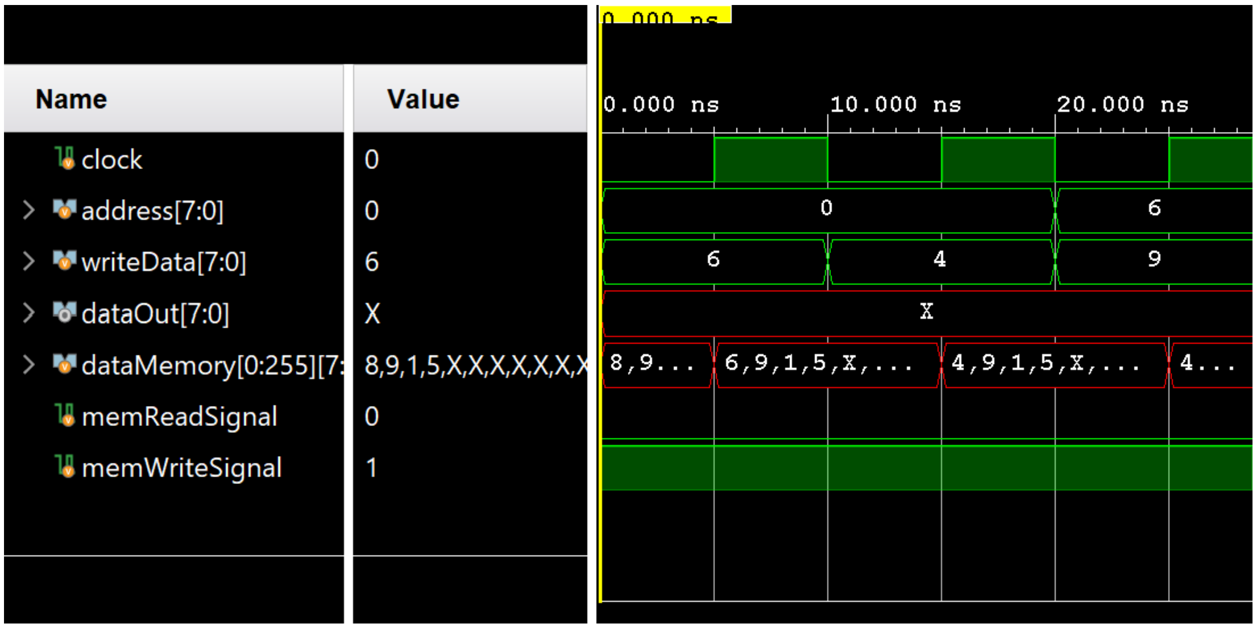
simulation :



Name	Value
first[0:7]	8
second[0:7]	4
selectLine	0
outputData[0:7]	8

As you can see by the simulation above mux takes in two inputs in the first clock cycle 8 and 4. It outputs 8 since the signal is 0. In the second clock cycle mux takes in two inputs in the second clock cycle 2 and 7. It outputs 7 since signal is 7.

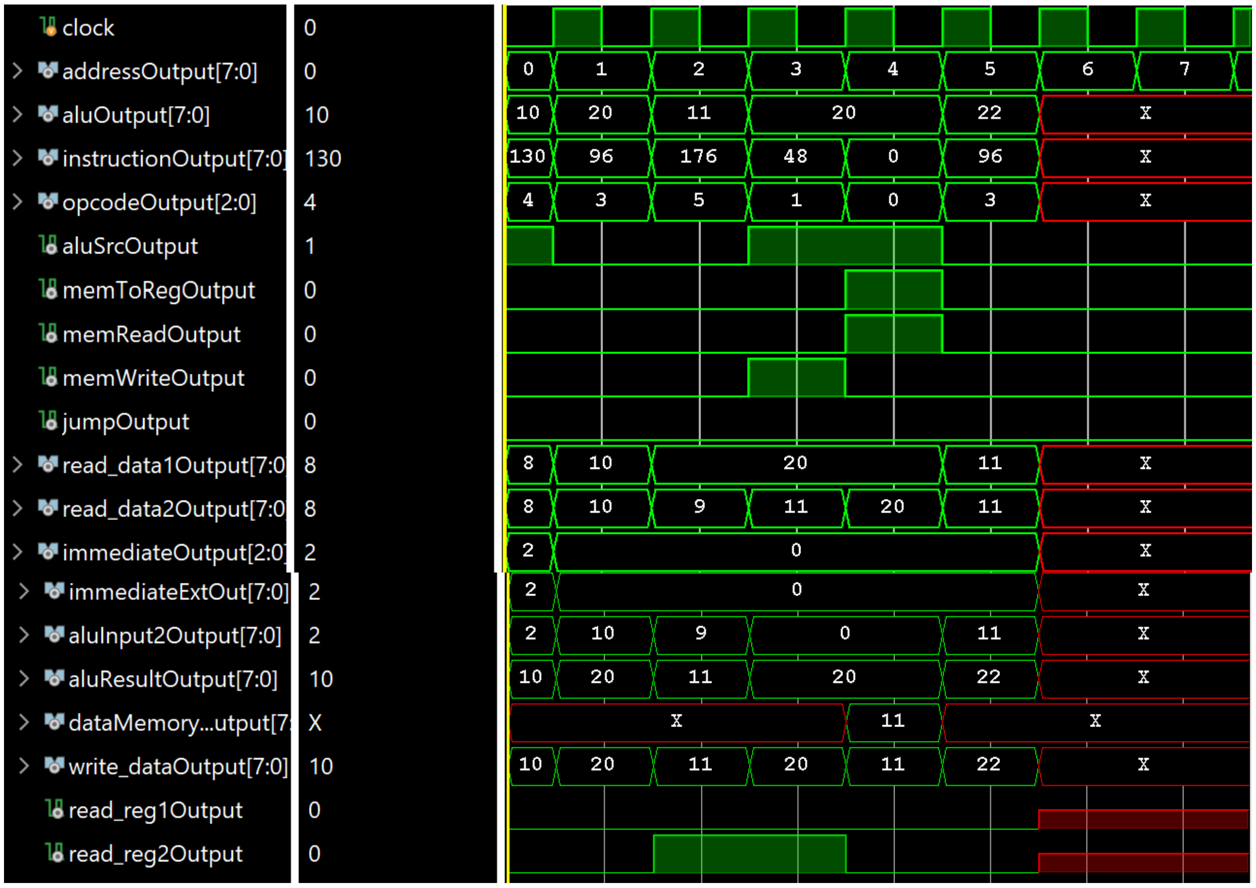
Data Memory: The data memory is a component that allows you to store the value of a register in the data memory or read a value from the data memory into a register.



Name	Value
clock	0
address[7:0]	0
writeData[7:0]	6
dataOut[7:0]	X
dataMemory[0:255][7:	8,9,1,5,X,X,X,X,X,X,X
memReadSignal	0
memWriteSignal	1

As you can see by the simulation we are able to write the value that is in writeData into the DataMemory.

Processor: The processor is the component that executes instructions. It is able to do this since the processor is the component that connects all of the other components. Since the processor is able to run simple instructions like add, sub, addi, lw, sw we can run instructions one after another to create a program.



clock 0
addressOutput[7:0] 0 — 0 1 2 3 4 5 6 7
aluOutput[7:0] 10 — 10 20 11 20 22 X
instructionOutput[7:0] 130 — 130 96 176 48 0 96 X
opcodeOutput[2:0] 4 — 4 3 5 1 0 3 X
aluSrcOutput 1
memToRegOutput 0
memReadOutput 0
memWriteOutput 0
jumpOutput 0
read_data1Output[7:0] 8 — 8 10 20 11 X
read_data2Output[7:0] 8 — 8 10 9 11 20 11 X
immediateOutput[2:0] 2 — 2 0 X
immediateExtOut[7:0] 2 — 2 0 X
aluInput2Output[7:0] 2 — 2 10 9 0 11 X
aluResultOutput[7:0] 10 — 10 20 11 20 22 X
dataMemory...utput[7 X — X 11 X
write_dataOutput[7:0] 10 — 10 20 11 20 11 22 X
read_reg1Output 0
read_reg2Output 0

The instruction sequence loaded into instruction memory is:
Initially I set r0 = 8, r1 = 9

Addi \$0, \$0, 2 // r0 = 10, r1 = 9
Add \$0, \$0, \$0 // r0 = 20, r1 = 9
Sub \$1, \$0, \$1 // r0 = 20, r1 = 11
Sw \$1, 0(\$0) // r0 = 20, r1 = 11, d[20] = 11
Lw \$0, 0(\$0)// r0 = d[20], r1 = 11
Add \$0, \$0, \$0 // r0 = 22, r1 = 11

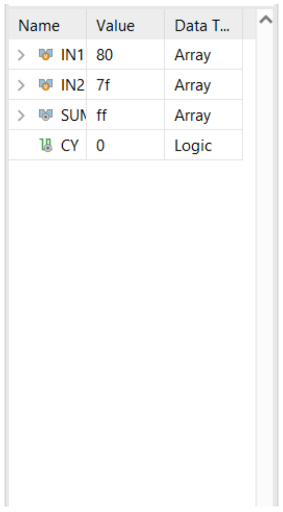
memory[0] = 8'b10000010;
memory[1] = 8'b01100000;

```
memory[2] = 8'b10110000;
memory[3] = 8'b00110000;
memory[4] = 8'b00000000;
memory[5] = 8'b01100000;
```

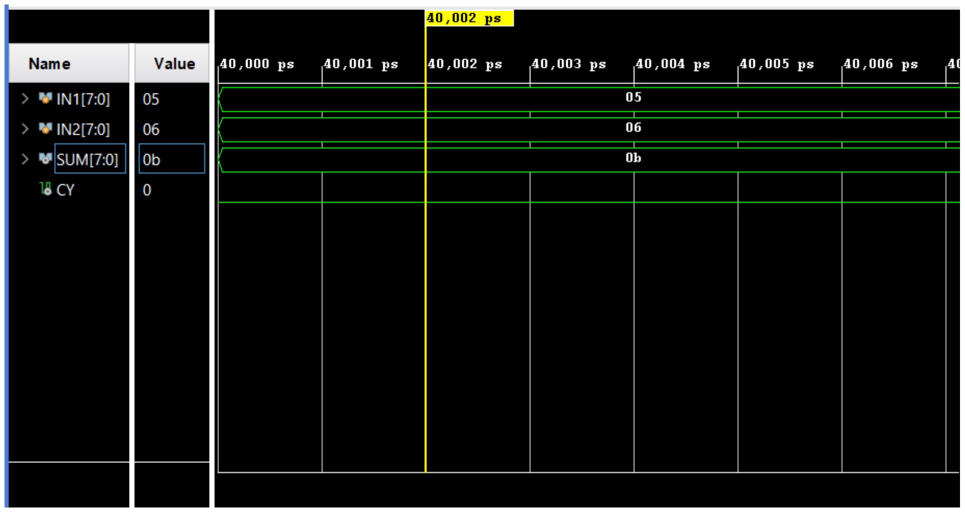
As you can see by the simulation above, the processor / components are working correctly since the alu result we get in the last cycle we have an instruction is 22. Based on our instruction input and initialized register values this is what we expect.

Adder

The adder adds any two unsigned 8 bit integers while accounting for the carryout



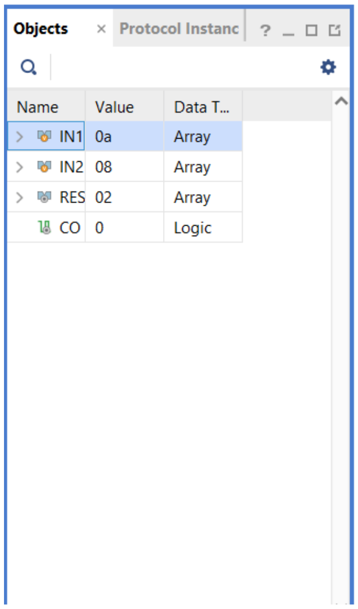
Name	Value	Data T...
IN1	80	Array
IN2	7f	Array
SUN	ff	Array
CY	0	Logic



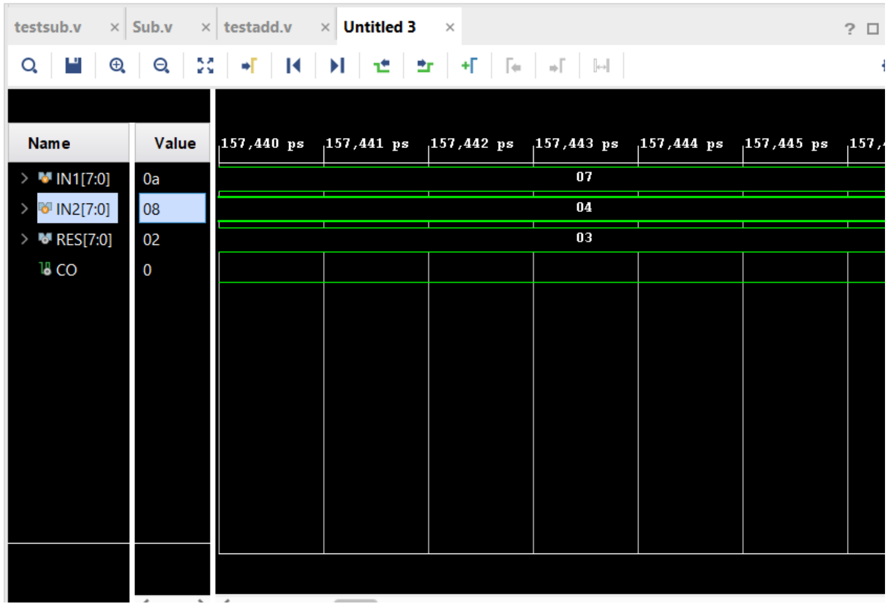
Name	Value
IN1[7:0]	05
IN2[7:0]	06
SUM[7:0]	0b
CY	0

SUB

The sub, subtracts any two unsigned 8 bit integers while accounting for the carryout



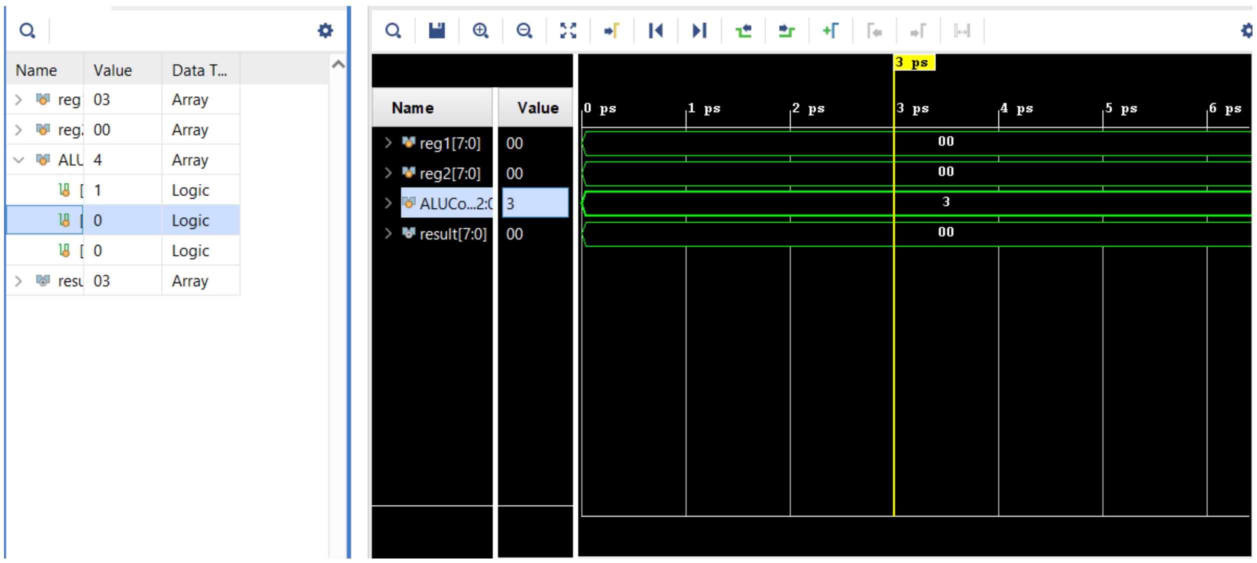
Name	Value	Data T...
IN1	0a	Array
IN2	08	Array
RES	02	Array
CO	0	Logic



Name	Value
IN1[7:0]	0a
IN2[7:0]	08
RES[7:0]	02
CO	0

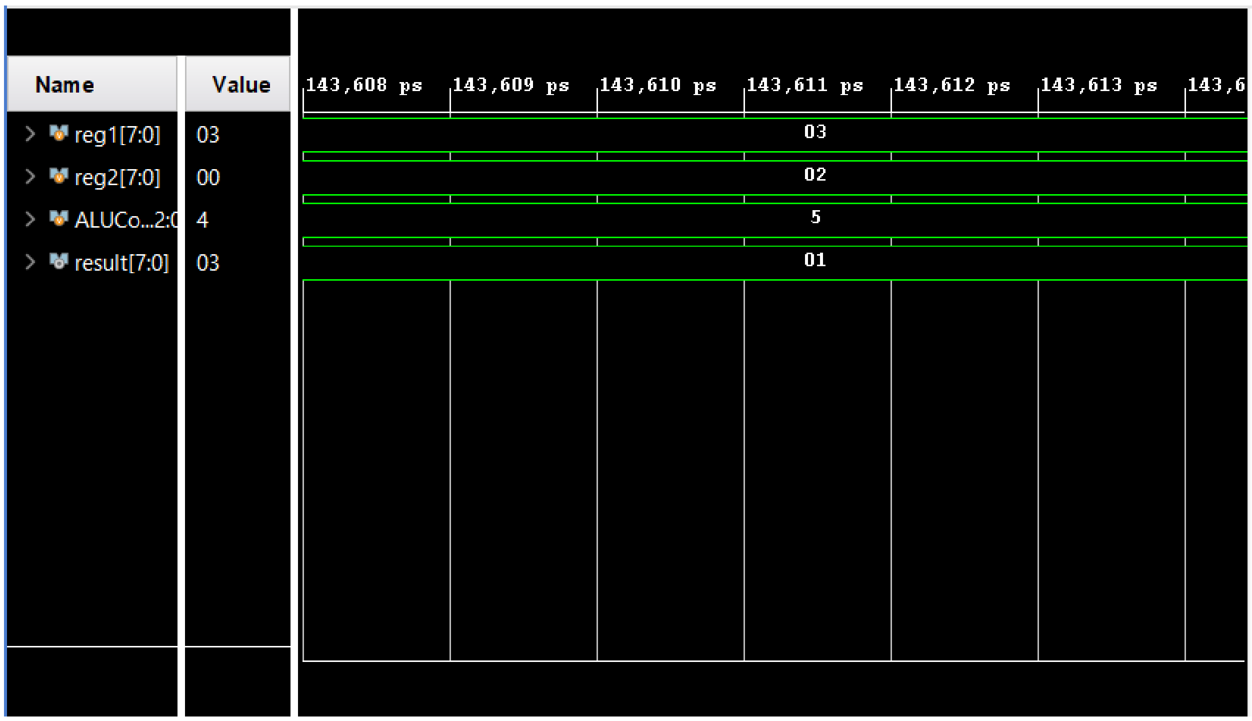
ALU

Description: The Alu is component responsible for carrying out logic and arithmetic calculations.

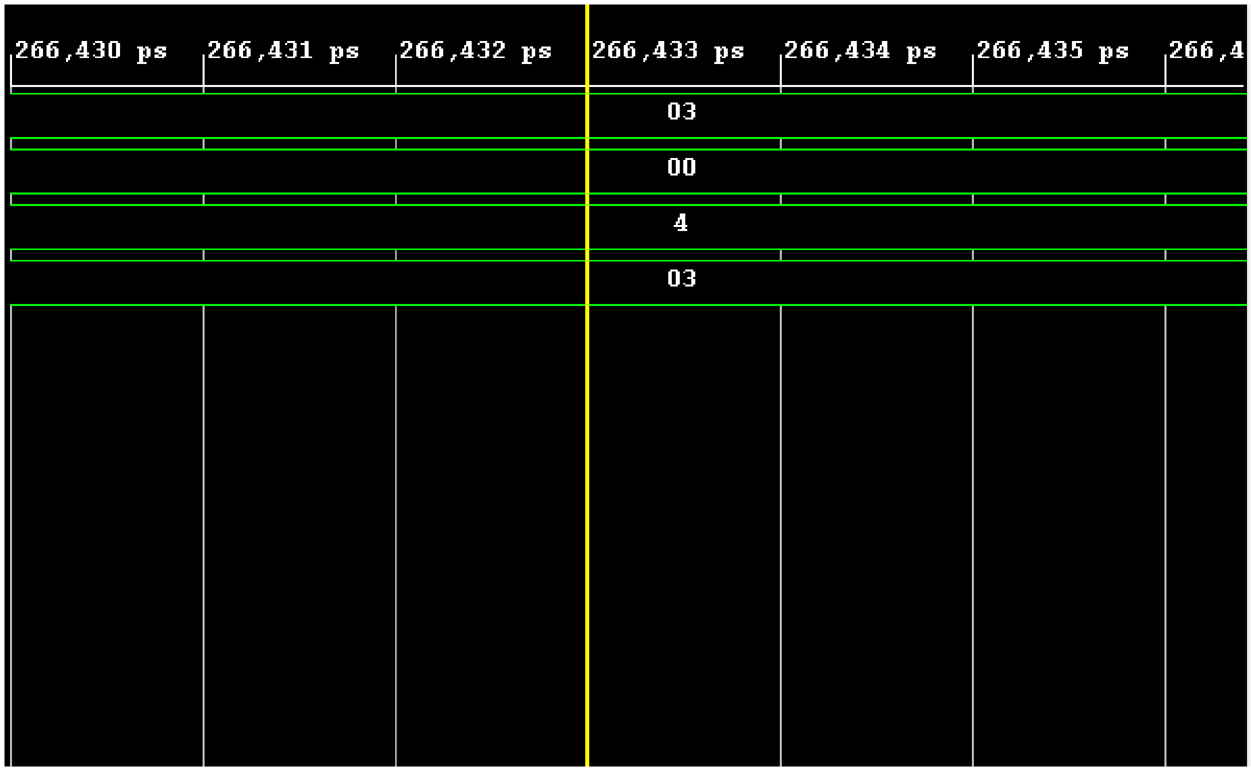


Name	Value	Data T...
reg	03	Array
reg:	00	Array
ALU	4	Array
[	1	Logic
[	0	Logic
[	0	Logic
resu	03	Array

Name	Value
reg1[7:0]	00
reg2[7:0]	00
ALUCo...2:0	3
result[7:0]	00

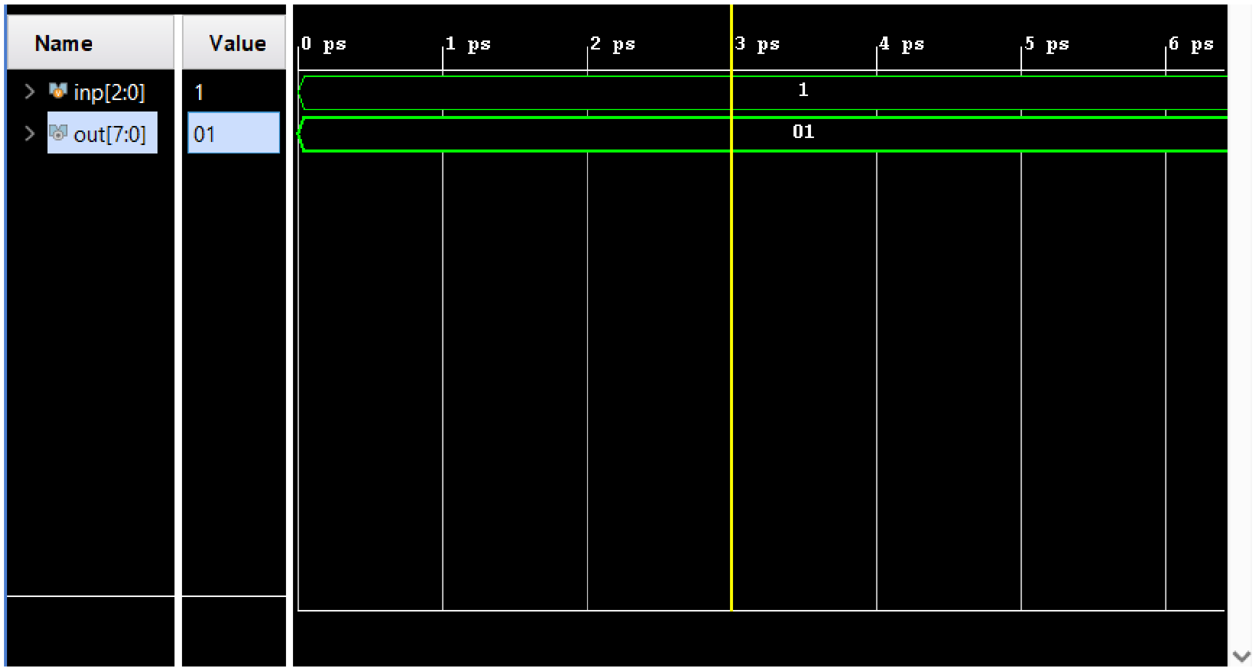


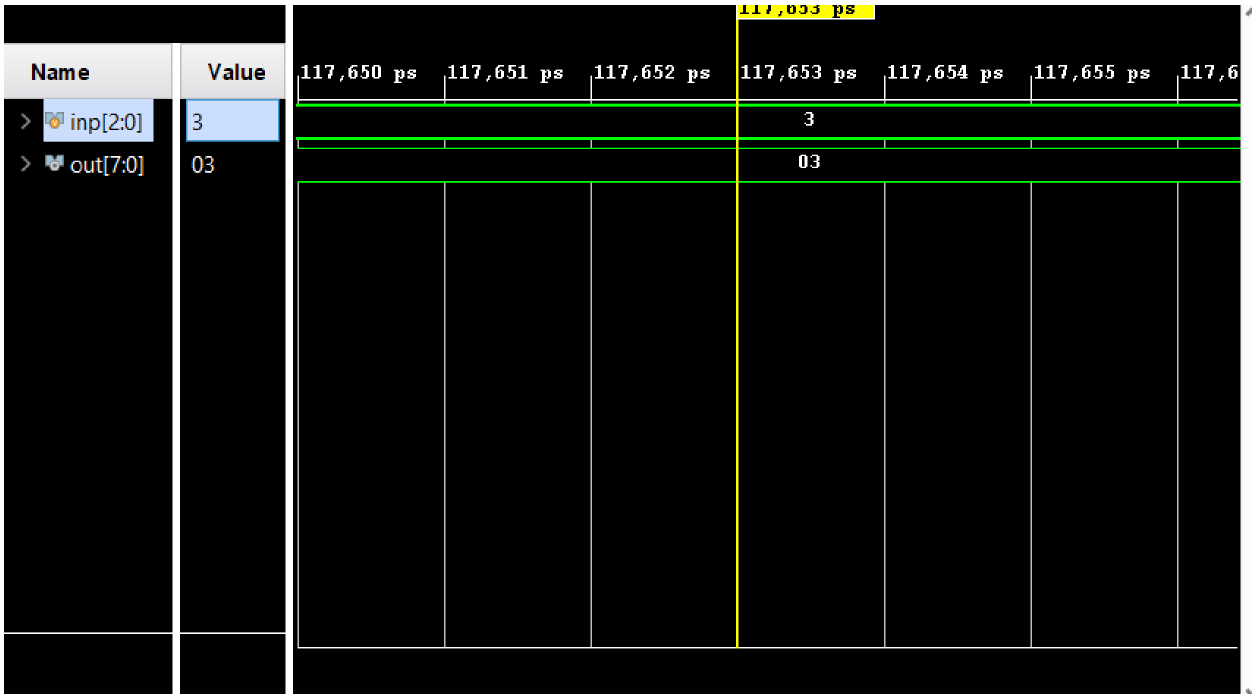
Name	Value
reg1[7:0]	03
reg2[7:0]	00
ALUCo...2:0	4
result[7:0]	03



SIGN Extend

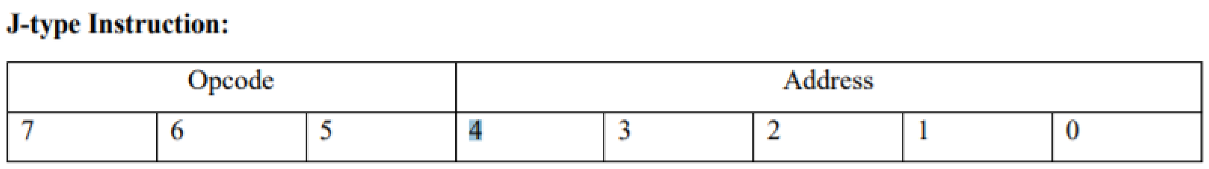
sign extend takes in a 3 bit input and extends it to 8 bits


Name
Value
inp[2:0]
1
out[7:0]
01
0 ps
1 ps
2 ps
3 ps
4 ps
5 ps
6 ps
1
01


117,653 ps
Name
Value
inp[2:0]
3
out[7:0]
03
117,650 ps
117,651 ps
117,652 ps
117,653 ps
117,654 ps
117,655 ps
117,6
3
03

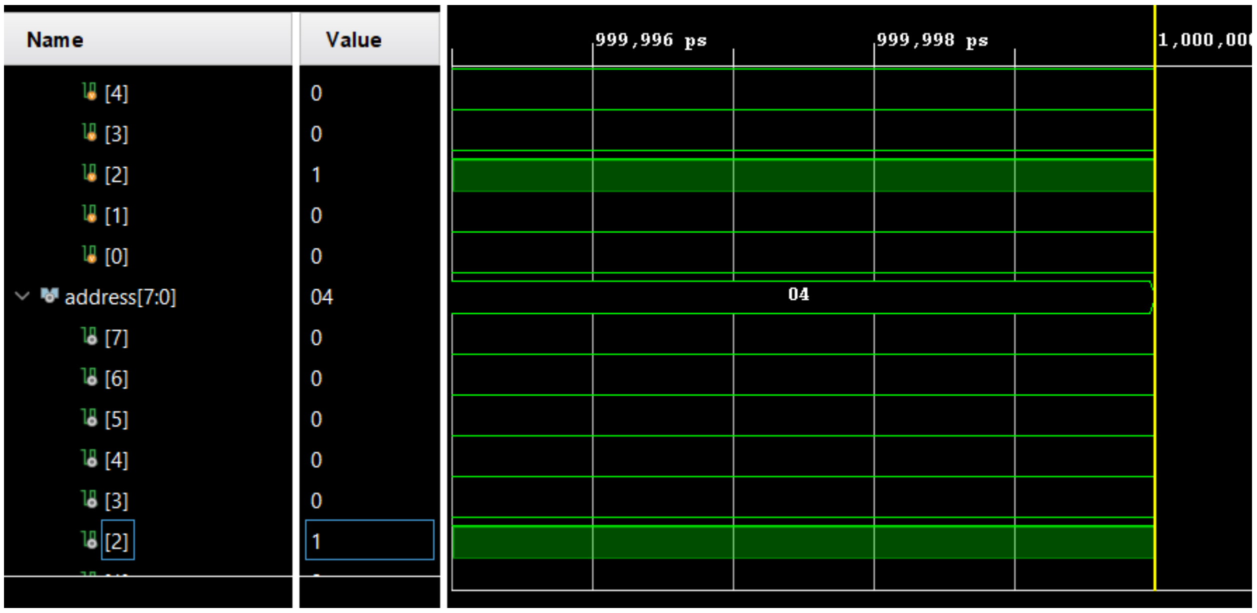
JUMP

As jump is a J-type instruction in which the first 5 bits will be the address for the target, we take the first 4 bits from the input and sign extend them to be 8bits and output the extended address



J-type Instruction:

Opcode			Address				
7	6	5	4	3	2	1	0



Name	Value
[4]	0
[3]	0
[2]	1
[1]	0
[0]	0
address[7:0]	04
[7]	0
[6]	0
[5]	0
[4]	0
[3]	0
[2]	1

Work Log:

Kyle:
Program Counter: implementation + testbench/simulation + report
Instruction Memory: implementation + testbench/simulation + report
Register File: implementation + testbench/simulation + report
Control: implementation + testbench/simulation + report
Mux: implementation + testbench/simulation + report
Data Memory: implementation + testbench/simulation + report
Processor: implemented data path for add Instruction
Processor: implemented data path for sub Instruction
Processor: implemented data path for addi instruction
Processor implemented data path for sw instruction
Processor implemented data path for lw instruction
Processor testbench/simulation
Processor report
Schematic
Fixed signext

Rishit
ALU: implementation + testbench/simulation + report
Adder: implementation + testbench/simulation + report
SUB: implementation + testbench/simulation + report
SIGN EXT: implementation + testbench/simulation + report
JUMP: implementation + testbench/simulation + report
Schematic

References:

https://marceluda.github.io/rp_dummy/EEOF2018/Verilog_Cheat_Sheet.pdf

https://www.youtube.com/watch?v=-Kdbzax9EOQ