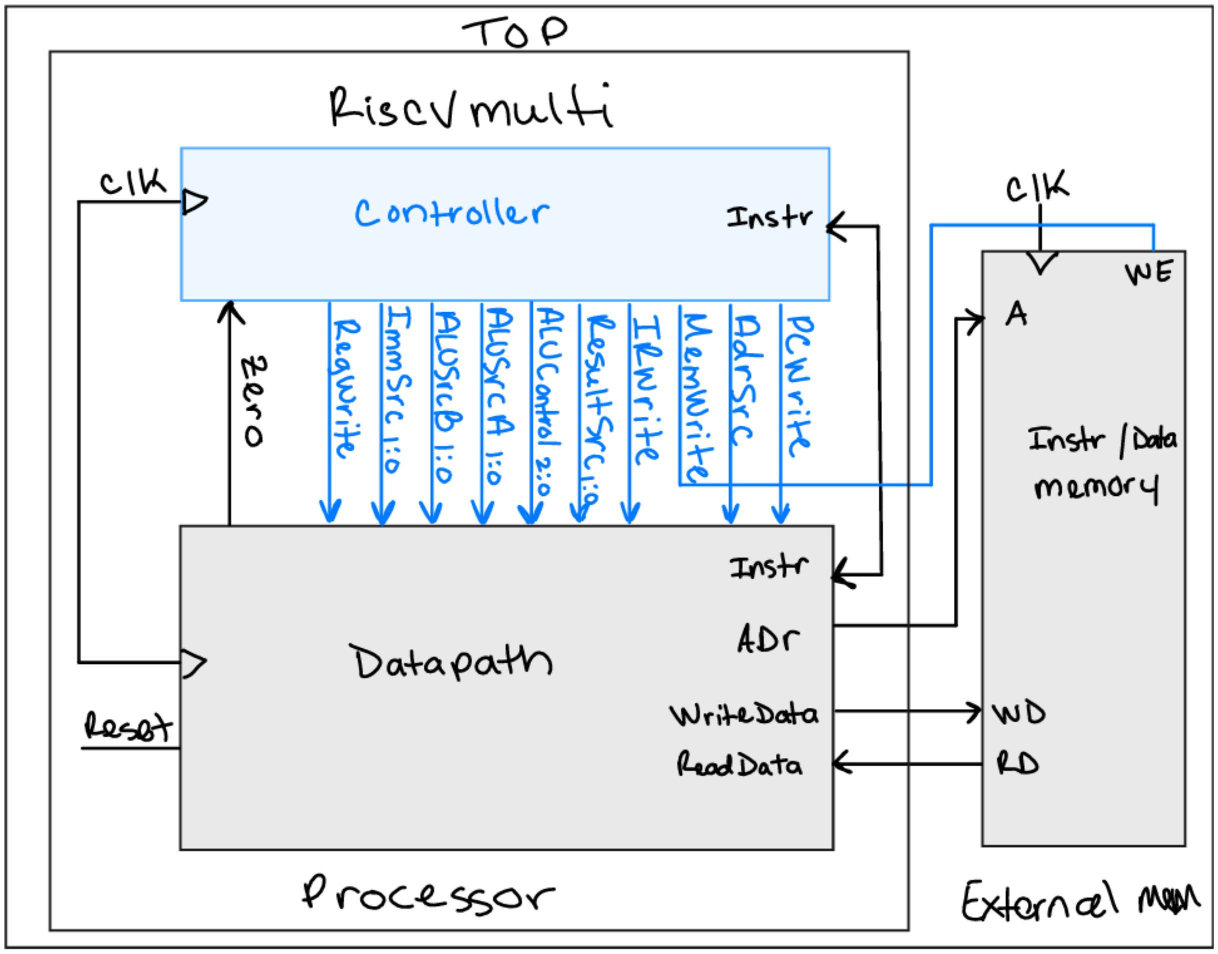
Class:	**CPE300L - Digital System Architecture and Design Lab**		Semester:	**Fall 2025**
Points		Document author:	**Darryll Mckoy**	
		Author's email:	**mckoyd1@unlv.nevada.edu**	
		Document topic:	**Postlab 9**	
Instructor's comments:				

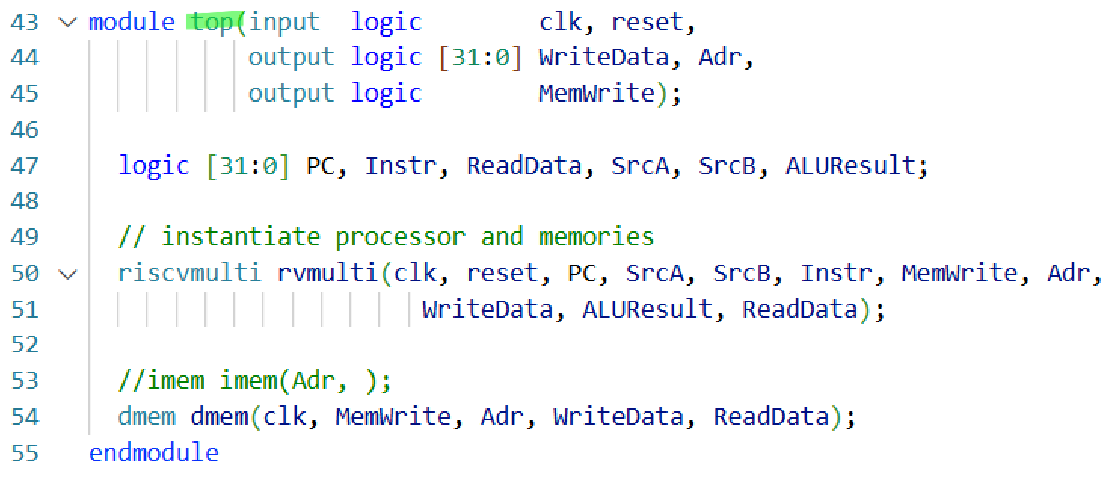
1. Time required to complete this lab

This lab took around 12 hours.

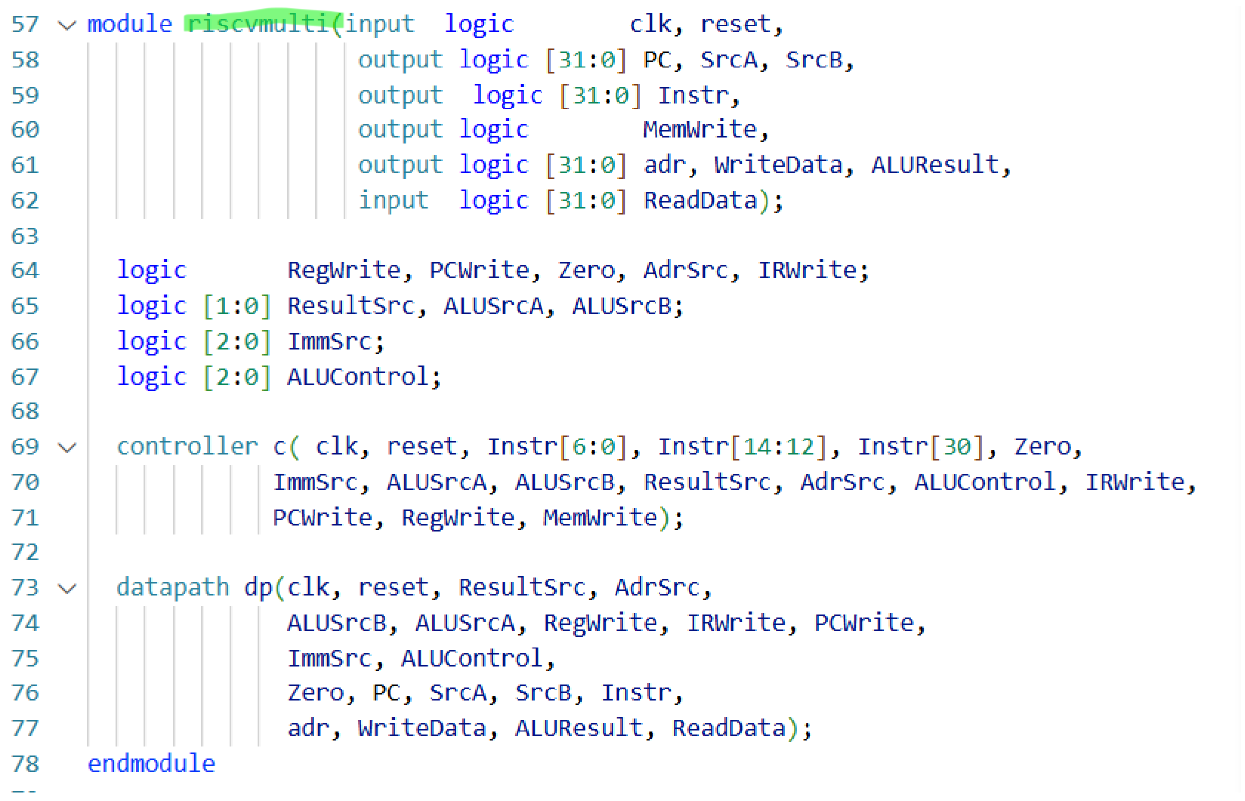
2. Multicycle processor box diagram



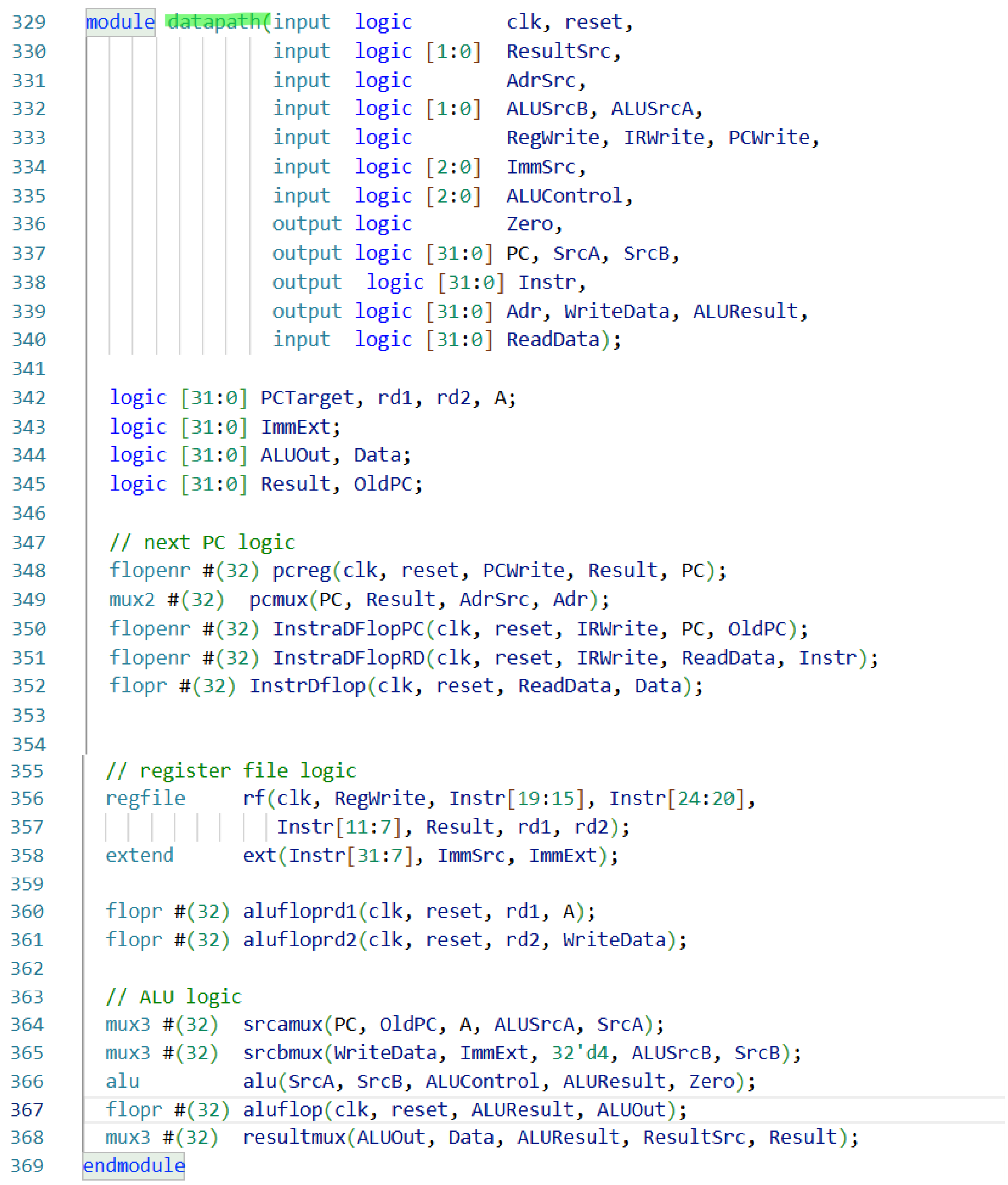
3. Hierarchical SystemVerilog code of top-level processor module (and submodules)



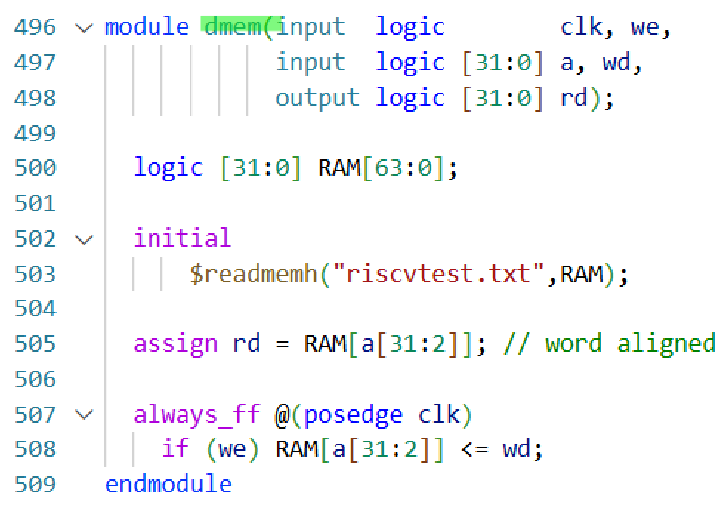
```
module top(input  logic        clk, reset,
           output logic [31:0] WriteData, Adr,
           output logic        MemWrite);

  logic [31:0] PC, Instr, ReadData, SrcA, SrcB, ALUResult;

  // instantiate processor and memories
  riscvmulti rvmulti(clk, reset, PC, SrcA, SrcB, Instr, MemWrite, Adr,
                     WriteData, ALUResult, ReadData);

  //imem imem(Adr, );
  dmem dmem(clk, MemWrite, Adr, WriteData, ReadData);
endmodule
```



```
module riscvmulti(input  logic        clk, reset,
                  output logic [31:0] PC, SrcA, SrcB,
                  output  logic [31:0] Instr,
                  output logic        MemWrite,
                  output logic [31:0] adr, WriteData, ALUResult,
                  input  logic [31:0] ReadData);

  logic       RegWrite, PCWrite, Zero, AdrSrc, IRWrite;
  logic [1:0] ResultSrc, ALUSrcA, ALUSrcB;
  logic [2:0] ImmSrc;
  logic [2:0] ALUControl;

  controller c( clk, reset, Instr[6:0], Instr[14:12], Instr[30], Zero,
               ImmSrc, ALUSrcA, ALUSrcB, ResultSrc, AdrSrc, ALUControl, IRWrite,
               PCWrite, RegWrite, MemWrite);

  datapath dp(clk, reset, ResultSrc, AdrSrc,
               ALUSrcB, ALUSrcA, RegWrite, IRWrite, PCWrite,
               ImmSrc, ALUControl,
               Zero, PC, SrcA, SrcB, Instr,
               adr, WriteData, ALUResult, ReadData);
endmodule
```



```
module datapath(input  logic        clk, reset,
                input  logic [1:0]  ResultSrc,
                input  logic        AdrSrc,
                input  logic [1:0]  ALUSrcB, ALUSrcA,
                input  logic        RegWrite, IRWrite, PCWrite,
                input  logic [2:0]  ImmSrc,
                input  logic [2:0]  ALUControl,
                output logic        Zero,
                output logic [31:0] PC, SrcA, SrcB,
                output  logic [31:0] Instr,
                output logic [31:0] Adr, WriteData, ALUResult,
                input  logic [31:0] ReadData);

  logic [31:0] PCTarget, rd1, rd2, A;
  logic [31:0] ImmExt;
  logic [31:0] ALUOut, Data;
  logic [31:0] Result, OldPC;

  // next PC logic
  flopenr #(32) pcreg(clk, reset, PCWrite, Result, PC);
  mux2 #(32)  pcmux(PC, Result, AdrSrc, Adr);
  flopenr #(32) InstraDFlopPC(clk, reset, IRWrite, PC, OldPC);
  flopenr #(32) InstraDFlopRD(clk, reset, IRWrite, ReadData, Instr);
  flopr #(32) InstrDflop(clk, reset, ReadData, Data);


  // register file logic
  regfile     rf(clk, RegWrite, Instr[19:15], Instr[24:20],
                 Instr[11:7], Result, rd1, rd2);
  extend      ext(Instr[31:7], ImmSrc, ImmExt);

  flopr #(32) alufloprd1(clk, reset, rd1, A);
  flopr #(32) alufloprd2(clk, reset, rd2, WriteData);

  // ALU logic
  mux3 #(32)  srcamux(PC, OldPC, A, ALUSrcA, SrcA);
  mux3 #(32)  srcbmux(WriteData, ImmExt, 32'd4, ALUSrcB, SrcB);
  alu         alu(SrcA, SrcB, ALUControl, ALUResult, Zero);
  flopr #(32) aluflop(clk, reset, ALUResult, ALUOut);
  mux3 #(32)  resultmux(ALUOut, Data, ALUResult, ResultSrc, Result);
endmodule
```



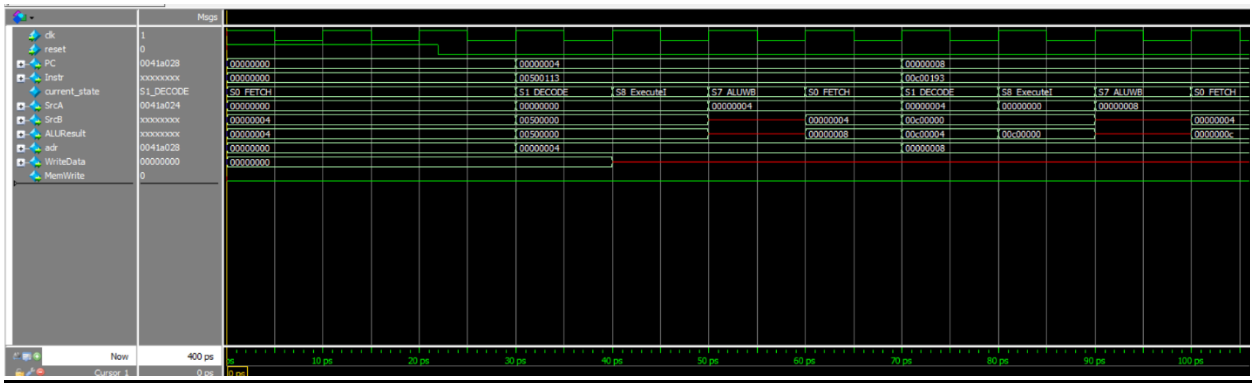
```
module dmem(input  logic        clk, we,
            input  logic [31:0] a, wd,
            output logic [31:0] rd);

  logic [31:0] RAM[63:0];

  initial
      $readmemh("riscvtest.txt",RAM);

  assign rd = RAM[a[31:2]]; // word aligned

  always_ff @(posedge clk)
    if (we) RAM[a[31:2]] <= wd;
endmodule
```

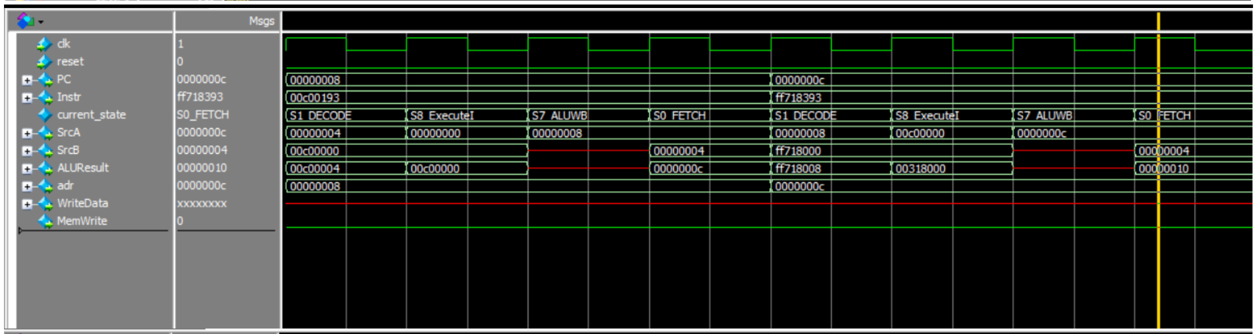
<u>GitHub link to full code:</u>
https://github.com/mckoyd1/CPE300L/blob/main/CPE300L_PostLab9/riscvmulti.sv

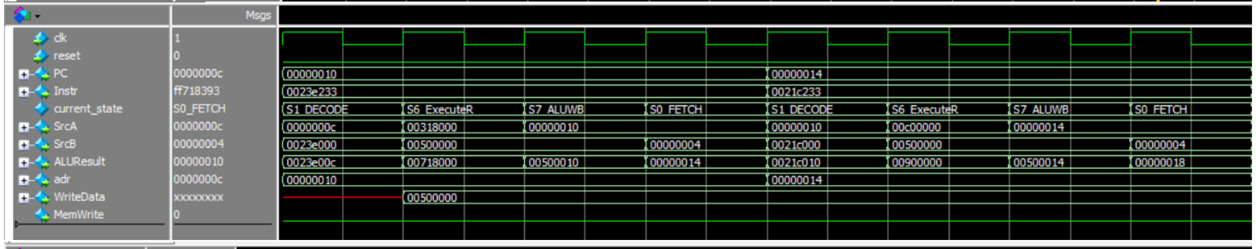
4. Table 1 showing key signals for at least the first three instructions

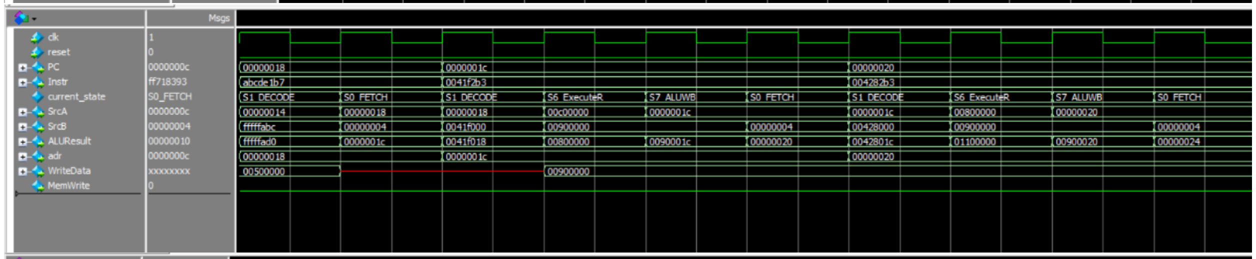
Step	PC	Instr	State	Result	Result Notes
1	00	00000000	S0: Fetch	00000004	
2	04	00500113	S1: Decode	00000005	
2	04	“ “	S8: ExecuteI	00000005	
2	04	“ “	S7: ALUWB	X	
2	04	“ “	S0: Fetch	00000008	
3	08	00c00193	S1: Decode	00c00000	
3	08	“ “	S8: ExecuteI	00c00000	
3	08	“ “	S7: ALUWB	X	
3	08	“ “	S0: Fetch	00000004	
4	0c	FF718393	S1: Decode	Ff718008	
4	0c	“ “	S8: ExecuteI	00318000	
4	0c	“ “	S7: ALUWB	X	
4	0c	“ “	S0: Fetch	00000010	

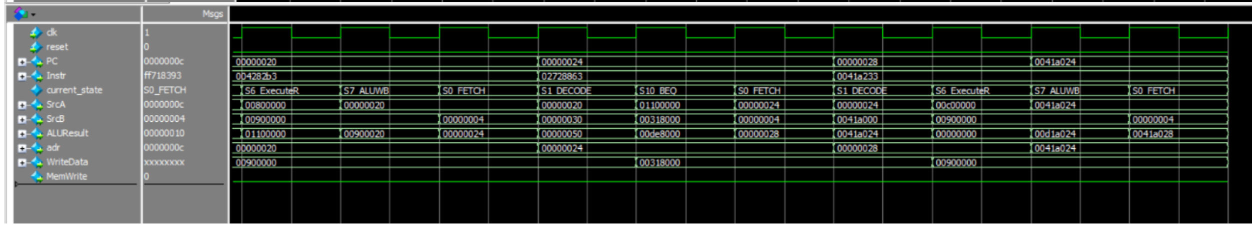
5. Simulation Waveforms


Msgs
clk 1
reset 0
PC 0041a028
Instr xxxxxxxx
current_state S1_DECODE
SrcA 0041a024
SrcB xxxxxxxx
ALUResult xxxxxxxx
adr 0041a028
WriteData 00000000
MemWrite 0
Now 400 ps
Cursor 1 0 ps


Msgs
clk 1
reset 0
PC 0000000c
Instr ff718393
current_state S0_FETCH
SrcA 0000000c
SrcB 00000004
ALUResult 00000010
adr 0000000c
WriteData xxxxxxxx
MemWrite 0


Msgs
clk 1
reset 0
PC 0000000c
Instr ff718393
current_state S0_FETCH
SrcA 0000000c
SrcB 00000004
ALUResult 00000010
adr 0000000c
WriteData xxxxxxxx
MemWrite 0


Msgs
clk 1
reset 0
PC 0000000c
Instr ff718393
current_state S0_FETCH
SrcA 0000000c
SrcB 00000004
ALUResult 00000010
adr 0000000c
WriteData xxxxxxxx
MemWrite 0


Msgs
clk 1
reset 0
PC 0000000c
Instr ff718393
current_state S0_FETCH
SrcA 0000000c
SrcB 00000004
ALUResult 00000010
adr 0000000c
WriteData xxxxxxxx
MemWrite 0