Lab Report 9

Vparik6

1. How many hours have you spent for this lab entirely?
   - I spent around total of around 9-10 hours.

2. Which activity takes the most significant amount of time?
   - Figuring out Datapath was quite challenging for me. Although, I was able to achieve it.
   - Also, filling out the table was time consuming.

3. A completed copy of Table 5

(Note: See attached excel file for better resolution)

Cycle	Reset	PC	Instr	FSM) state	SrcA	SrcB	ALUResult	Zero	Control Word
1	1	00	0	0	00	04	04	00	5010
2	0	04	di 20020005	1	04	14	18(X)	00	0030
3	0	04	di 20020005	9	00	05	05	00	0564
4	0	04	di 20020005	10	04	00	05	00	0C24
5	0	04	di 20020005	0	08	04	04	00	5010
6	0	08	di 2003000c	1	08	04	38	00	0030
7	0	08	di 2003000c	9	00	0c	0c	00	0564
8	0	08	di 2003000c	10	00	0c	0c	00	0C24
9	0	08	di 2003000c	0	08	04	0c	00	5010
10	0	0c	addi 2067ffff7	1	0c	fffffffdc	fffffffe8	00	0030
11	0	0c	addi 2067ffff7	9	0c	ffffffff7	3	00	0564
12	0	0c	addi 2067ffff7	10	0c	ffffffff7	03	00	0C24
13	0	0c	addi 2067ffff7	0	0c	04	10	00	5010
14	0	10	or 00c22025	1	10	8094	80a4	00	0030
15	0	10	or 00c22025	6	03	05	07	00	0582
16	0	10	or 00c22025	7	03	05	07	00	0D42
17	0	10	or 00c22025	0	10	04	14	00	5010
18	0	14	sd 00642824	1	14	a090	a0a5	00	0030
19	0	14	sd 00642824	6	0c	07	04	00	0582
20	0	14	sd 00642824	7	0c	07	04	00	0D42
21	0	14	sd 00642824	0	14	04	18	00	5010
22	0	18	dd 00a42820	1	18	a080	a098	00	0030
23	0	18	dd 00a42820	6	04	07	000b	00	0582
24	0	18	dd 00a42820	7	04	07	000b	00	0D42
25	0	18	dd 00a42820	0	18	04	1c	00	5010
26	0	1c	beq 10a7000a	1	1c	28	44	00	0030
27	0	1c	beq 10a7000a	8	0b	03	08	00	0745
28	0	1c	beq 10a7000a	0	1c	04	20	00	5010
29	0	20	slt 0064202a	1	20	80a8	80c8	00	0030
30	0	20	slt 0064202a	6	0c	07	00	01	0582
31	0	20	slt 0064202a	7	0c	07	00	01	0D42
32	0	20	slt 0064202a	0	20	04	24	00	5010
33	0	24	beq 10800001	1	24	04	28	00	0030
34	0	24	beq 10800001	8	00	00	00	01	0745
35	0	28	beq 10800001	0	28	04	2c	00	5010
36	0	2c	slt 00c2202a	1	2c	80a8	80d4	00	0030
37	0	2c	slt 00c2202a	6	03	05	01	00	0582
38	0	2c	slt 00c2202a	7	03	05	01	00	0D42
39	0	2c	slt 00c2202a	0	2c	04	30	00	5010
40	0	30	dd 00853820	1	30	e080	e0b0	00	0030
41	0	30	dd 00853820	6	01	0b	0c	00	0582
42	0	30	dd 00853820	7	01	0b	0c	00	0D42
43	0	30	dd 00853820	0	30	04	34	00	5010
44	0	34	ub 00e23822	1	34	e088	e0bc	00	0030
45	0	34	ub 00e23822	6	0c	05	07	00	0582
46	0	34	ub 00e23822	7	0c	05	07	00	0D42
47	0	34	ub 00e23822	0	34	04	38	00	5010
48	0	38	sw ac670044	1	38	110	148	00	0030
49	0	38	sw ac670044	2	0c	44	50	00	0420
50	0	38	sw ac670044	5	0c	44	50	00	25A0
51	0	38	sw ac670044	0	38	04	3c	00	5010
52	0	3c	lw 8c020050	1	3c	140	17c	00	0030
53	0	3c	lw 8c020050	2	00	50	50	00	0420
54	0	3c	lw 8c020050	3	00	50	50	00	0520
55	0	3c	lw 8c020050	4	00	50	50	00	0DA0
56	0	3c	lw 8c020050	0	3c	04	40	00	5010
57	0	40	J 08000011	1	40	44	84	00	0030
58	0	40	J 08000011	11	00	11	11	00	4428
59	0	44	J 08000011	0	44	04	48	00	5010
60	0	48	sw ac020054	1	48	150	198	00	0030
61	0	48	sw ac020054	2	00	54	54	00	0420
62	0	48	sw ac020054	5	00	54	54	00	25A0

4. SystemVerilog code of both control and Datapath.

Controller.sv

```
module controller(input logic clk, reset,
                  input logic [5:0] op, funct,
                  input logic zero,
                  output logic pcen, memwrite, irwrite, regwrite,
                  output logic alusrca, iord, memtoreg, regdst,
                  output logic [1:0] alusrcb, pcsrc,
                  output logic [2:0] alucontrol,
                  output logic [3:0] state);

    logic [1:0] aluop;
    logic branch, pcwrite;

    // Main Decoder and ALU Decoder subunits.
    maindec md(clk, reset, op,
               pcwrite, memwrite, irwrite, regwrite,
               alusrca, branch, iord, memtoreg, regdst,
               alusrcb, pcsrc, aluop, state);
    aludec ad(funct, aluop, alucontrol);

     // ADD CODE HERE
     // Add combinational logic (i.e. an assign statement)
     // to produce the PCEn signal (pcen) from the branch,
     // zero, and pcwrite signals

    assign pcen = (branch && zero) || (pcwrite);

endmodule

//MAINDEC:
module maindec(input logic clk, reset,
               input logic [5:0] op,
               output logic pcwrite, memwrite, irwrite, regwrite,
               output logic alusrca, branch, iord, memtoreg, regdst,
               output logic [1:0] alusrcb, pcsrc,
               output logic [1:0] aluop,
               output logic [3:0] state);

    parameter FETCH   = 4'b0000; // State 0
    parameter DECODE  = 4'b0001; // State 1
    parameter MEMADR  = 4'b0010; // State 2
    parameter MEMRD   = 4'b0011; // State 3
    parameter MEMWB   = 4'b0100; // State 4
    parameter MEMWR   = 4'b0101; // State 5
    parameter RTYPEEX = 4'b0110; // State 6
    parameter RTYPEWB = 4'b0111; // State 7
    parameter BEQEX   = 4'b1000; // State 8
    parameter ADDIEX  = 4'b1001; // State 9
    parameter ADDIWB  = 4'b1010; // state 10
    parameter JEX     = 4'b1011; // State 11
    parameter LW      = 6'b100011; // Opcode for lw
    parameter SW      = 6'b101011; // Opcode for sw
    parameter RTYPE   = 6'b000000; // Opcode for R-type
    parameter BEQ     = 6'b000100; // Opcode for beq
    parameter ADDI    = 6'b001000; // Opcode for addi
    parameter J       = 6'b000010; // Opcode for j

    logic [3:0] nextstate;
    logic [14:0] controls;

       // state register

       always_ff @(posedge clk or posedge reset)
          if(reset) state <= FETCH;
          else state <= nextstate;
```

```
71          always_comb
72             case(state)
73                FETCH:      nextstate = DECODE;
74                DECODE:
75                   case(op)
76                      LW:         nextstate = MEMADR;
77                      SW:         nextstate = MEMADR;
78                      RTYPE:      nextstate = RTYPEEX;
79                      BEQ:        nextstate = BEQEX;
80                      ADDI:       nextstate = ADDIEX;
81                      J:          nextstate = JEX;
82                      default:    nextstate = 4'bx; // should never happen
83                   endcase
84
85                // Add code here
86                MEMADR:
87                   case(op)
88                      LW: nextstate = MEMRD;
89                      SW: nextstate = MEMWR;
90                      default: nextstate= 4'bx;
91                   endcase
92
93                MEMRD:   nextstate = MEMWB;
94                MEMWB:   nextstate = FETCH;
95                MEMWR:   nextstate = FETCH;
96                RTYPEEX: nextstate = RTYPEWB;
97                RTYPEWB: nextstate = FETCH;
98                BEQEX:   nextstate = FETCH;
99                ADDIEX:  nextstate = ADDIWB;
100               ADDIWB:  nextstate = FETCH ;
101               JEX:     nextstate = FETCH;
102               default: nextstate = 4'bx; // should never happen
103            endcase
104
105         // output logic
106         assign {pcwrite, memwrite, irwrite, regwrite,
107                 alusrca, branch, iord, memtoreg, regdst,
108                 alusrcb, pcsrc, aluop} = controls;
109
110         // ADD CODE HERE
111         // Finish entering the output logic below. The
112         // output logic for the first two states, S0 and S1,
113         // have been completed for you.
114
115         always_comb
116            case(state)
117               FETCH:  controls = 15'h5010;
118               DECODE: controls = 15'h0030;
119
120               // your code goes here
121               MEMADR:  controls = 15'h0420;
122               MEMRD:   controls = 15'h0520;
123               MEMWB:   controls = 15'h0da0;
124               MEMWR:   controls = 15'h25a0;
125               RTYPEEX: controls = 15'h0582;
126               RTYPEWB: controls = 15'h0d42;
127               BEQEX:   controls = 15'h0745;
128               ADDIEX:  controls = 15'h0564;
129               ADDIWB:  controls = 15'h0c24;
130               JEX:     controls = 15'h4428;
131               default: controls = 15'hxxxx; // should never happen
132            endcase
133   endmodule
134
```

```
135  // ALUDEC:
136  module aludec(input logic [5:0] funct,
137                input logic [1:0] aluop,
138                output logic [2:0] alucontrol);
139
140              // ADD CODE HERE
141              // Complete the design for the ALU Decoder.
142              // Your design goes here. Remember that this is a combinational
143              // module.
144              // Remember that you may also reuse any code from previous labs.
145              always_comb
146                  case(aluop)
147                    2'b00: alucontrol = 3'b010;  // add
148                    2'b01: alucontrol = 3'b110;  // sub
149                    2'b11: alucontrol = 3'b001;  // OR
150                    default:
151                          case(funct)          // RTYPE
152                              6'b100000: alucontrol = 3'b010; // ADD
153                              6'b100010: alucontrol = 3'b110; // SUB
154                              6'b100100: alucontrol = 3'b000; // AND
155                              6'b100101: alucontrol = 3'b001; // OR
156                              6'b101010: alucontrol = 3'b111; // SLT
157                              default:   alucontrol = 3'bxxx; // ???
158                          endcase
159                endcase
160  endmodule
161
```

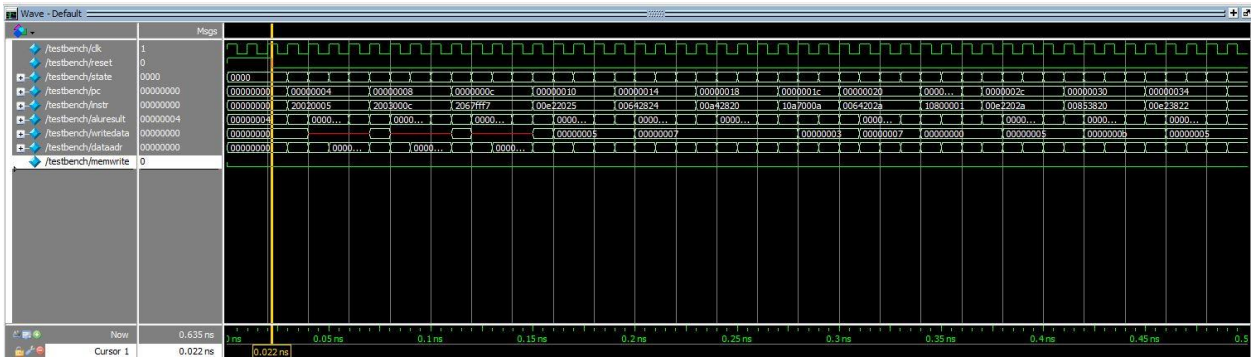
Datapath.sv

```
4   // it is composed of instances of its sub-modules. For example,
5   // the instruction register is instantiated as a 32-bit flopenr.
6   // The other submodules are likewise instantiated.
7   module datapath(input logic  clk, reset,
8                   input logic  pcen, irwrite, regwrite,
9                   input logic  alusrca, iord, memtoreg, regdst,
10                  input logic  [1:0] alusrcb, pcsrc,
11                  input logic  [2:0] alucontrol,
12                  output logic [5:0] op, funct,
13                  output logic zero,
14                  output logic [31:0] adr, writedata, pc, instr, aluresult,
15                  input logic  [31:0] readdata);
16
17      // Below are the internal signals of the datapath module.
18      logic [4:0]  writereg;
19      logic [31:0] pcnext;
20      logic [31:0] data, srca, srcb;
21      logic [31:0] a,b,A,B;
22      logic [31:0] aluout, result;
23      logic [31:0] signimm; // the sign-extended immediate
24      logic [31:0] signimmsh; // the sign-extended immediate shifted left by 2
25      //logic [31:0] wd3, rd1, rd2;
26
27      // op and funct fields to controller
28      assign op = instr[31:26];
29      assign funct = instr[5:0];
30      assign writedata = B;
31
32      // Your datapath hardware goes below. Instantiate each of the submodules
33      // that you need. Remember that alu's, mux's and various other
34      // versions of parameterizable modules are available in textbook 7.6
35      // Here, parameterizable 3:1 and 4:1 muxes are provided below for your use.
36      // Remember to give your instantiated modules applicable names
37      // such as pcreg (PC register), wdmux (Write Data Mux), etc.
38      // so it's easier to understand.
39      // ADD CODE HERE
40      // datapath
41
42       // next PC logic
43    flopenr #(32) pcreg(clk, reset, pcen, pcnext, pc);
44    flopenr #(32) instrReg(clk, reset, irwrite, readdata, instr);
45    flopr   #(32) dataReg(clk, reset, readdata, data);
46    flopr   #(32) rega(clk, reset, a, A);
47    flopr   #(32) regb(clk, reset, b, B);
48    flopr   #(32) aluReg(clk, reset, aluresult, aluout);
49
50    sl2           immsh(signimm, signimmsh);
51    signext       se(instr[15:0], signimm);
52    //zeroext       ze(instr[15:0], zeroimm);
53
54    mux2 #(32)  srcamux(pc, A, alusrca, srca);
55    mux2 #(32)  pcmux2(pc, aluout, iord, adr);
56    mux2 #(5)   wrmux(instr[20:16], instr[15:11], regdst, writereg);
57    mux2 #(32)  resmux(aluout, data, memtoreg, result);
58    mux3 #(32)  pcmux3(aluresult, aluout, {pc[31:28], instr[25:0], 2'b00},
59                       pcsrc, pcnext);
60
61    // register file logic
62    regfile     rf(clk, regwrite, instr[25:21], instr[20:16], writereg, result, a, b);
63
64     // ALU logic
65    mux4 #(32)  srcbmux(B, 32'd4, signimm, signimmsh, alusrcb, srcb);
66    alu         alu(.A(srca), .B(srcb), .F(alucontrol), .Y(aluresult), .zero(zero));
67  endmodule
68
```

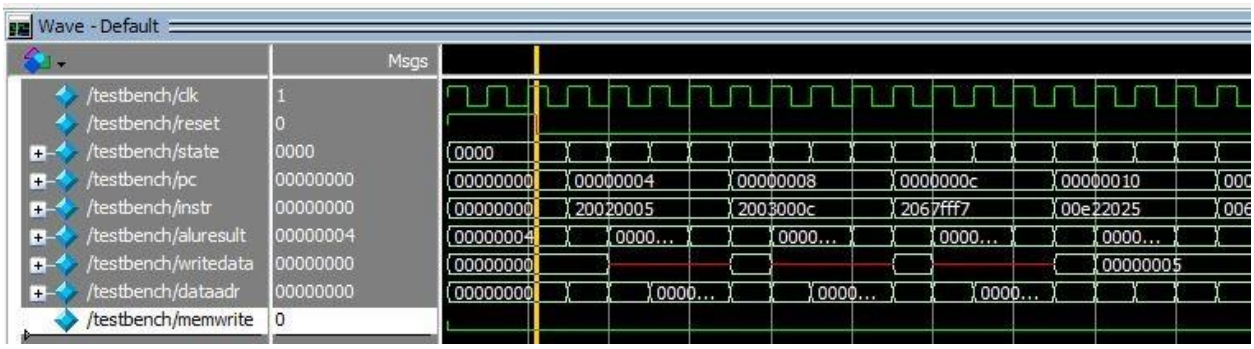
5. Simulation waveforms

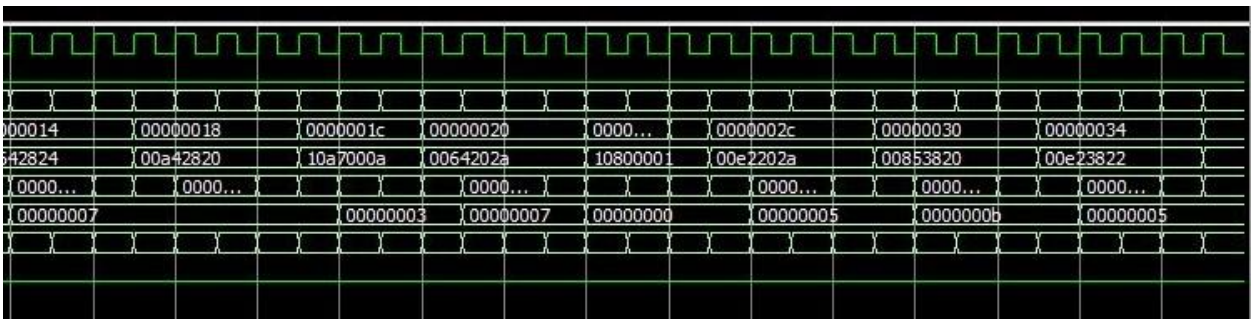
Q) Do the results match your expectations?

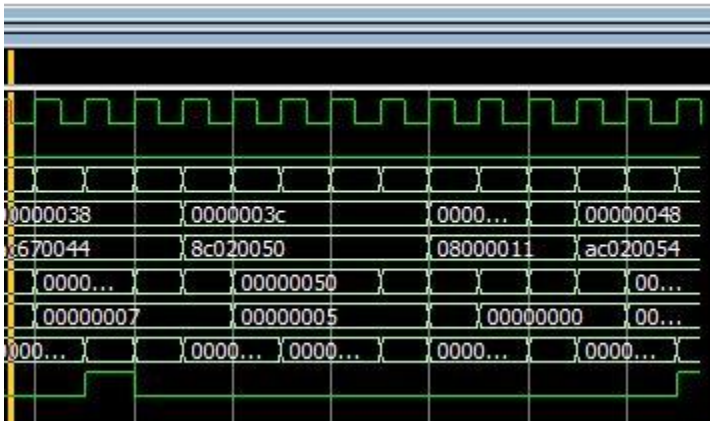
A) Yes, as I expected in table 5. My results were matching precisely.



Full waveform (snippets)

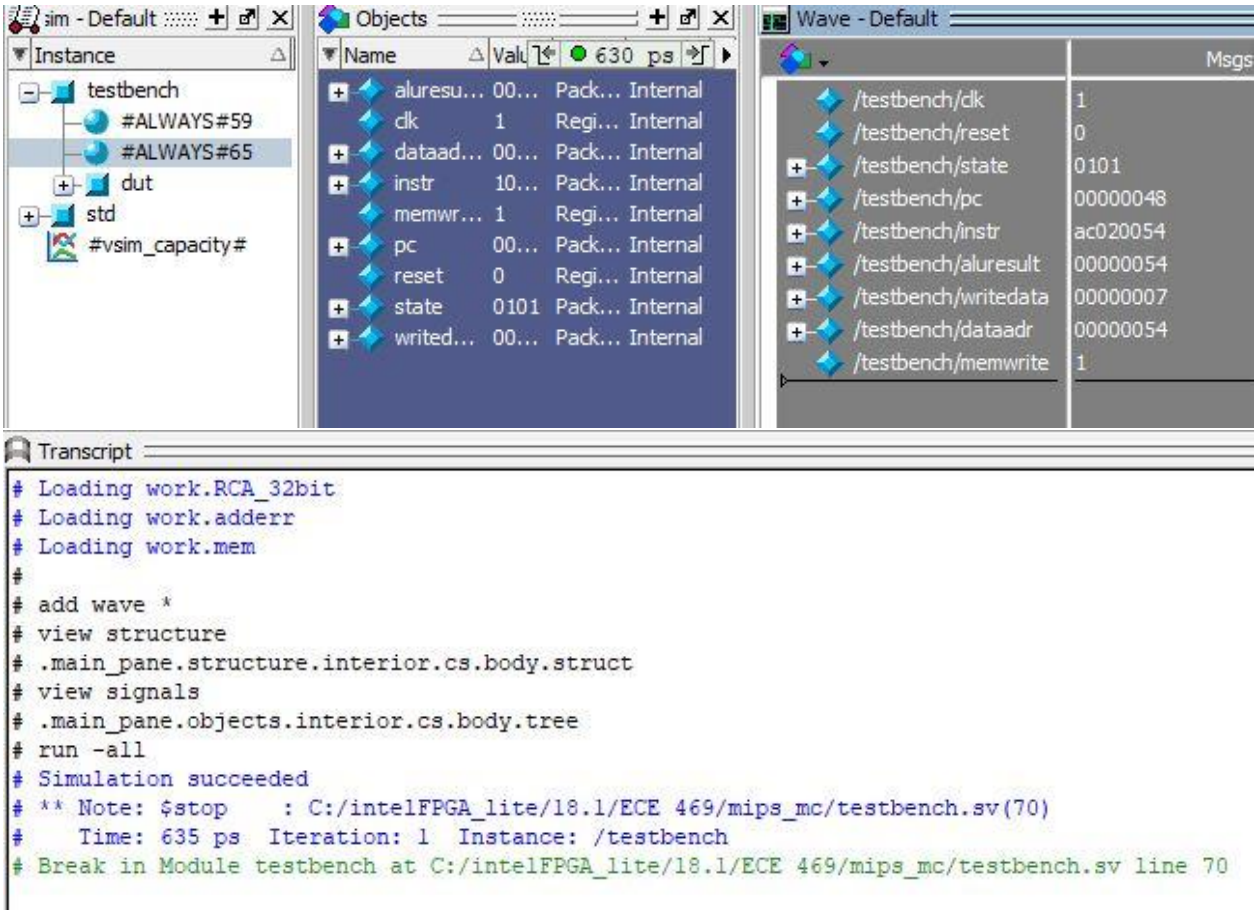






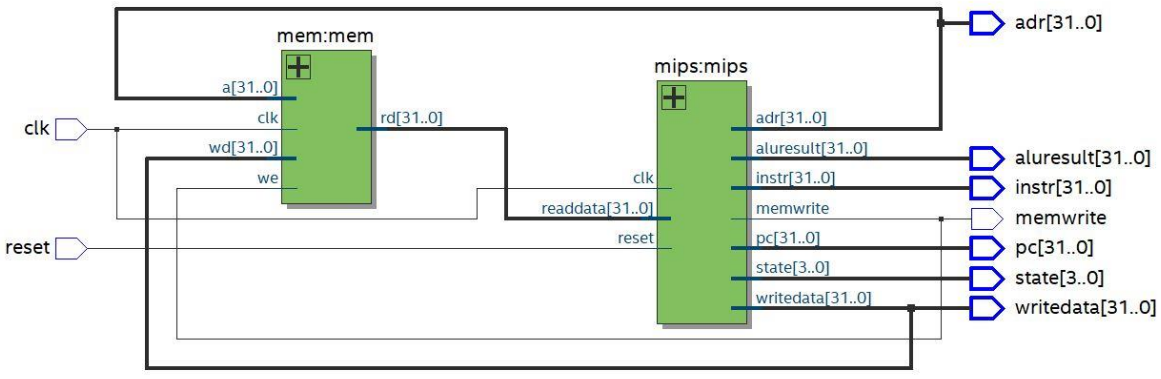
Q) Does the program indicate Simulation Succeeded?

A) Yes, as you can see in screenshot below my simulation was succeeded.

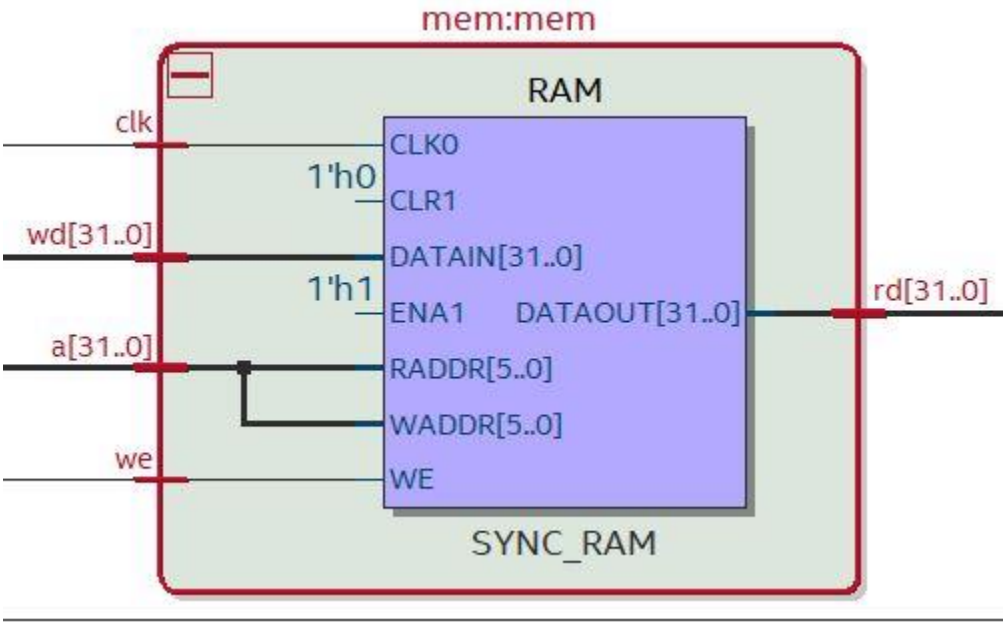


6) RTL view from Quartus' compilation result of your MIPS processor.

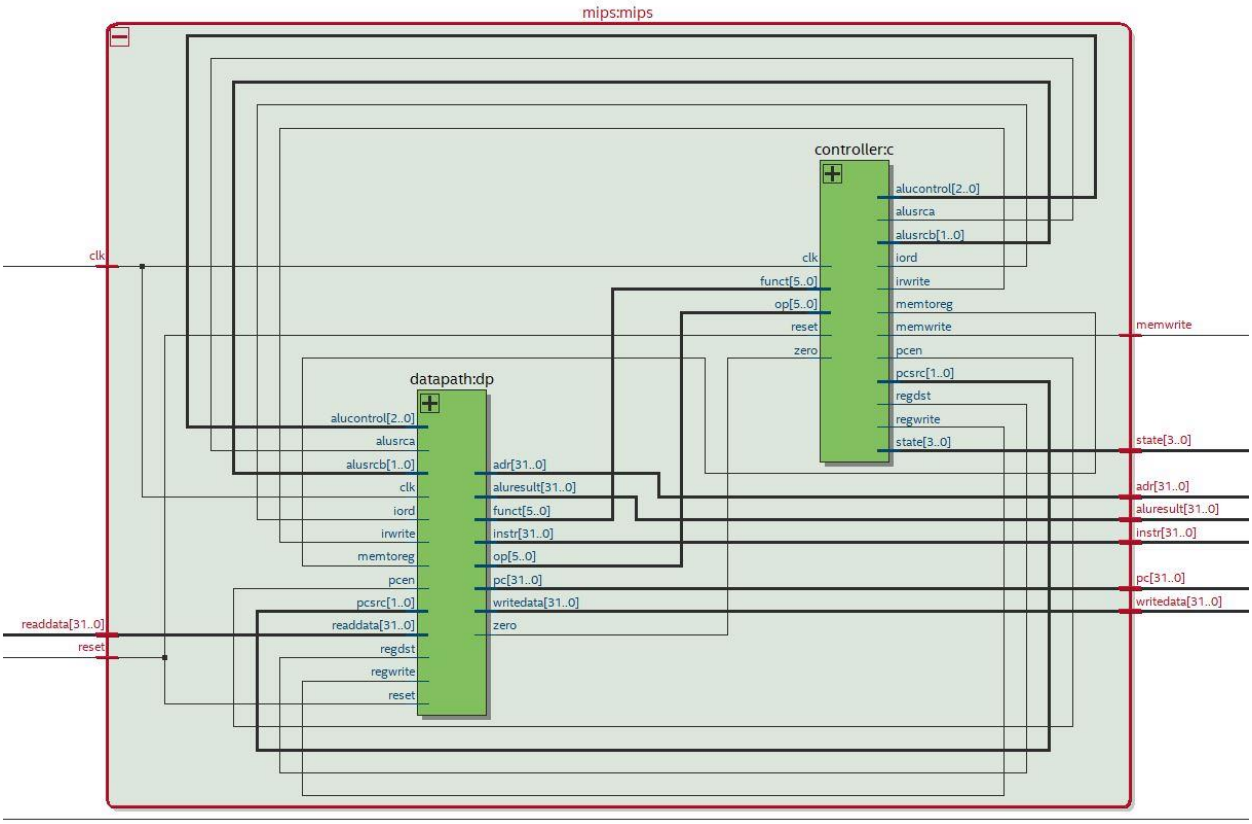
Top Level Module



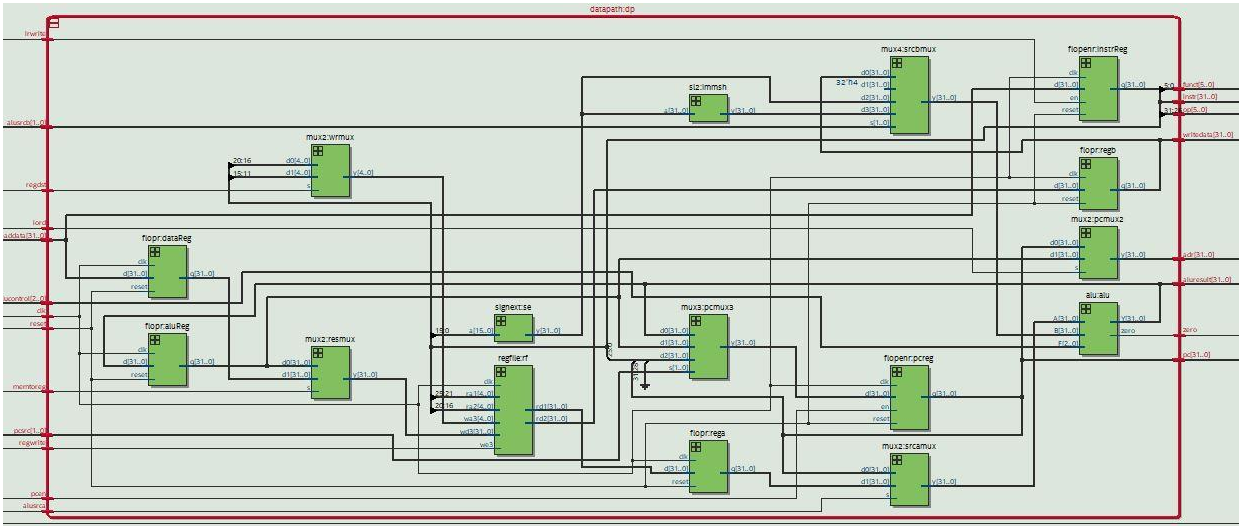
Mem Module



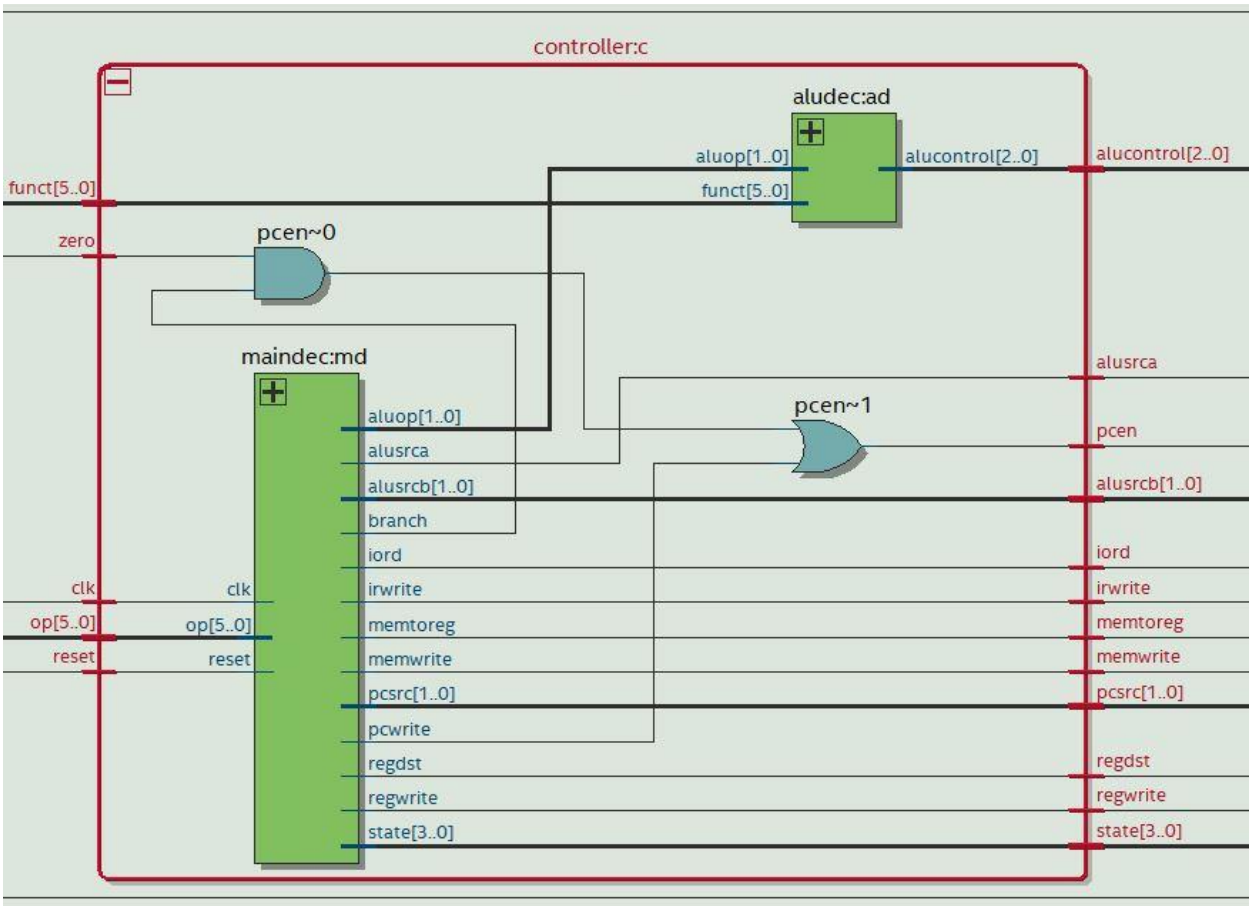
MIPS



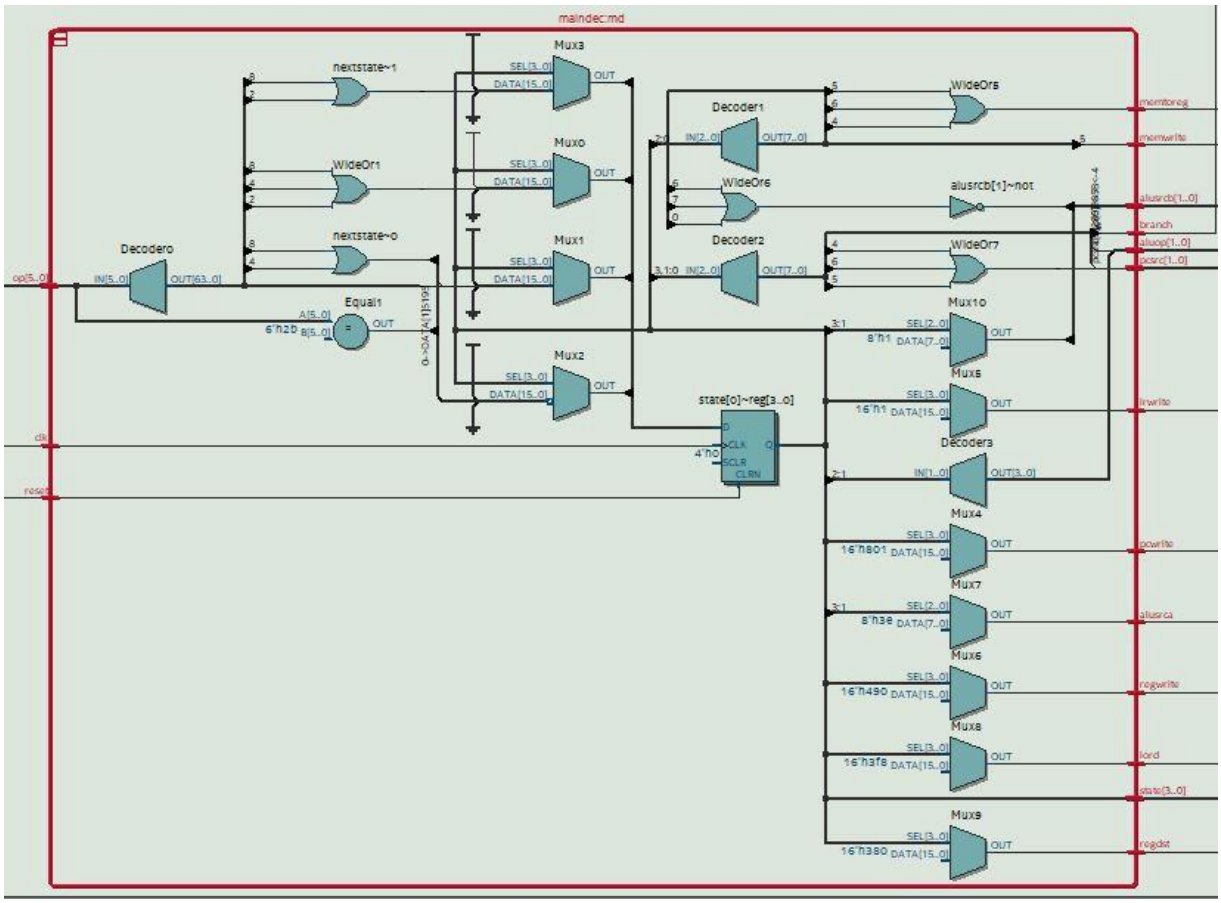
Datapath



Controller



MainDec



AluDec