ECE 469
Project 3 Part A
Tamanna Ravi Rupani
665679988
Niranjani Venkatesan
675456032

Workload report: How many hours have you spent on this part? Which activity takes the most significant amount of time? This will not affect your grade (unless omitted).

Topic	No of hours
Understanding and studying the FSM and Control Word	1.5
Understanding Multi-Cycle MIPS	2
Execution of Controller in SystemVerilog and debugging errors	2
Execution of Datapath in SystemVerilog and debugging errors	2
Quartus Part	0.5
Report Writing	1.5
Total	9.5

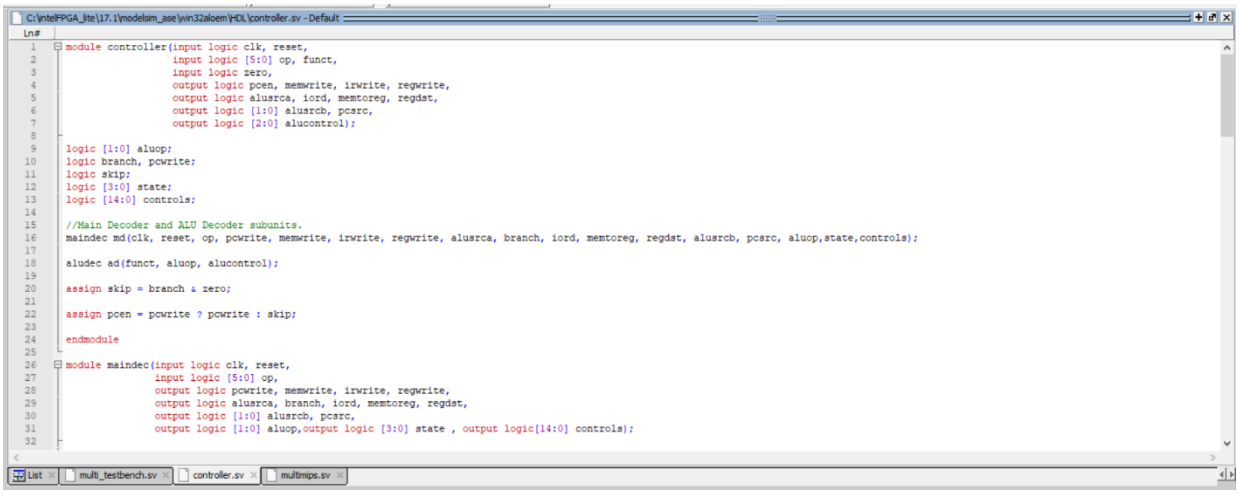
Understanding the Multicycle MIPS, writing the code accordingly took longer time

A completed Main Decoder output table (Table 4).

State (Name)	PCWrite	MemWrite	IRWrite	RegWrite	ALUSrcA	Branch	IorD	MemtoReg	RegDst	ALUSrcB[1:0]	PCSrc[1:0]	ALUOp[1:0]	FSM Control Word
0 (Fetch)	1	0	1	0	0	0	0	0	0	01	00	00	0x5010
1 (Decode)	0	0	0	0	0	0	0	0	0	11	00	00	0x0030
2 (MemAdr)	0	0	0	0	1	0	0	0	0	10	00	00	0x0420
3 (MemRd)	0	0	0	0	0	0	1	0	0	00	00	00	0x0100
4 (MemWB)	0	0	0	1	0	0	0	1	0	00	00	00	0x0880
5 (MemWr)	0	1	0	0	0	0	1	0	0	00	00	00	0x2100
6 (RtypeEx)	0	0	0	0	1	0	0	0	0	00	00	10	0x0402
7 (RtypeWB)	0	0	0	1	0	0	0	0	1	00	00	00	0x0840
8 (BeqEx)	0	0	0	0	1	1	0	0	0	00	01	01	0x0605
9 (AddiEx)	0	0	0	0	1	0	0	0	0	10	00	00	0x0420
10 (AddiWB)	0	0	0	1	0	0	0	0	0	00	00	00	0x0800
11 (JEx)	1	0	0	0	0	0	0	0	0	00	10	00	0x4008

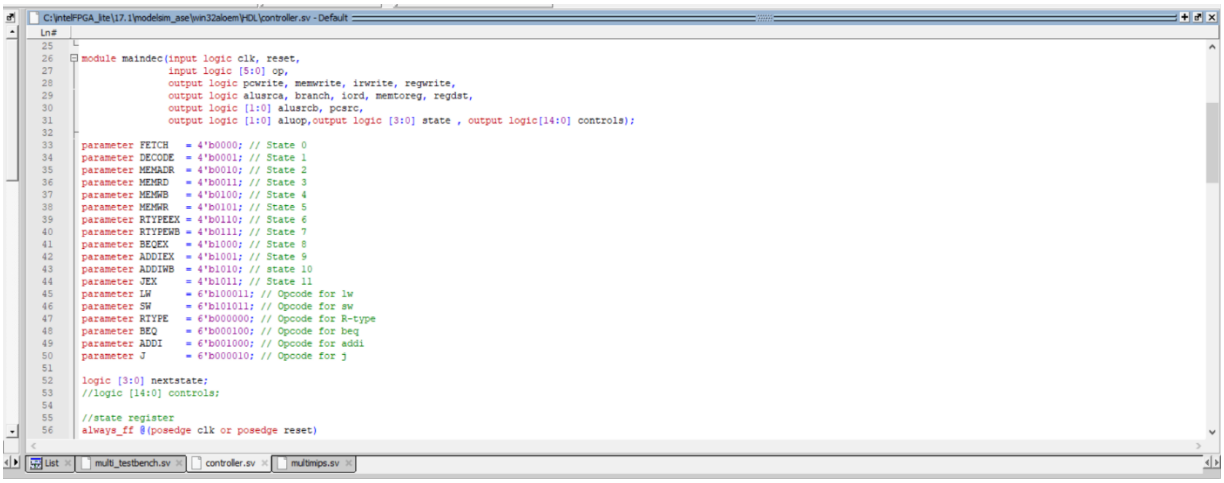
The SystemVerilog for your controller, maindec, and aludec modules

Controller:

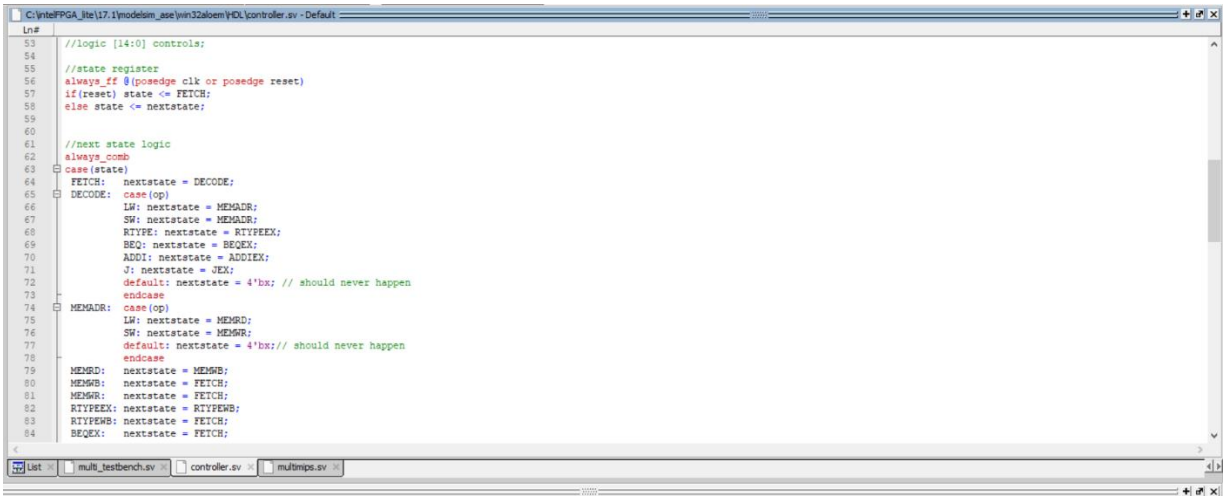


```
module controller(input logic clk, reset,
                  input logic [5:0] op, funct,
                  input logic zero,
                  output logic pcen, memwrite, irwrite, regwrite,
                  output logic alusrca, iord, memtoreg, regdst,
                  output logic [1:0] alusrcb,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

logic [1:0] aluop;
logic branch, pcwrite;
logic skip;
logic [3:0] state;
logic [14:0] controls;

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

aludec ad(funct, aluop, alucontrol);

assign skip = branch & zero;

assign pcen = pcwrite ? pcwrite : skip;

endmodule
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 , output logic[14:0] controls);
```

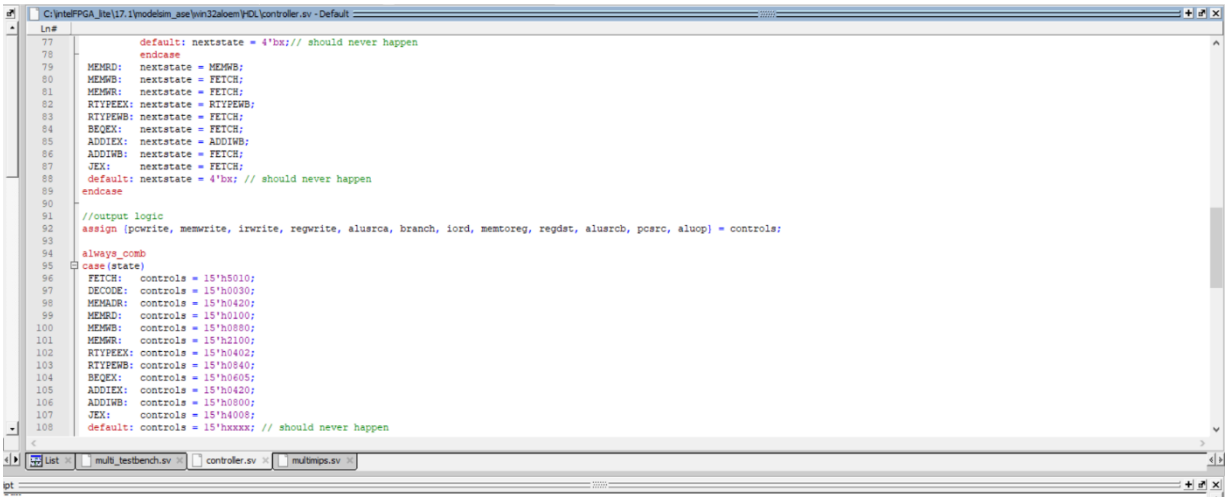
MainDec:



```
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 , output logic[14:0] controls);

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)
```

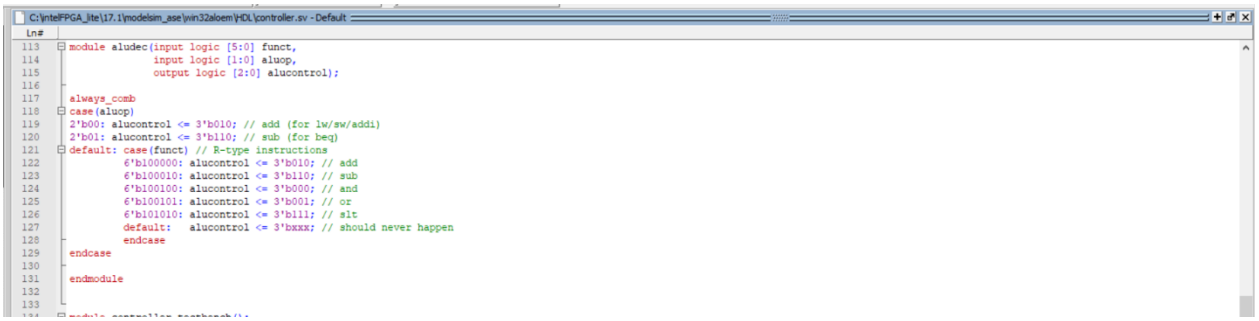


```
//logic [14:0] controls;

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


//next state logic
always_comb
case(state)
 FETCH:   nextstate = DECODE;
 DECODE:  case(op)
          LW: nextstate = MEMADR;
          SW: nextstate = MEMADR;
          RTYPE: nextstate = RTYPEEX;
          BEQ: nextstate = BEQEX;
          ADDI: nextstate = ADDIEX;
          J: nextstate = JEX;
          default: nextstate = 4'bx; // should never happen
          endcase
 MEMADR:  case(op)
          LW: nextstate = MEMRD;
          SW: nextstate = MEMWR;
          default: nextstate = 4'bx;// should never happen
          endcase
 MEMRD:   nextstate = MEMWB;
 MEMWB:   nextstate = FETCH;
 MEMWR:   nextstate = FETCH;
 RTYPEEX: nextstate = RTYPEWB;
 RTYPEWB: nextstate = FETCH;
 BEQEX:   nextstate = FETCH;
```



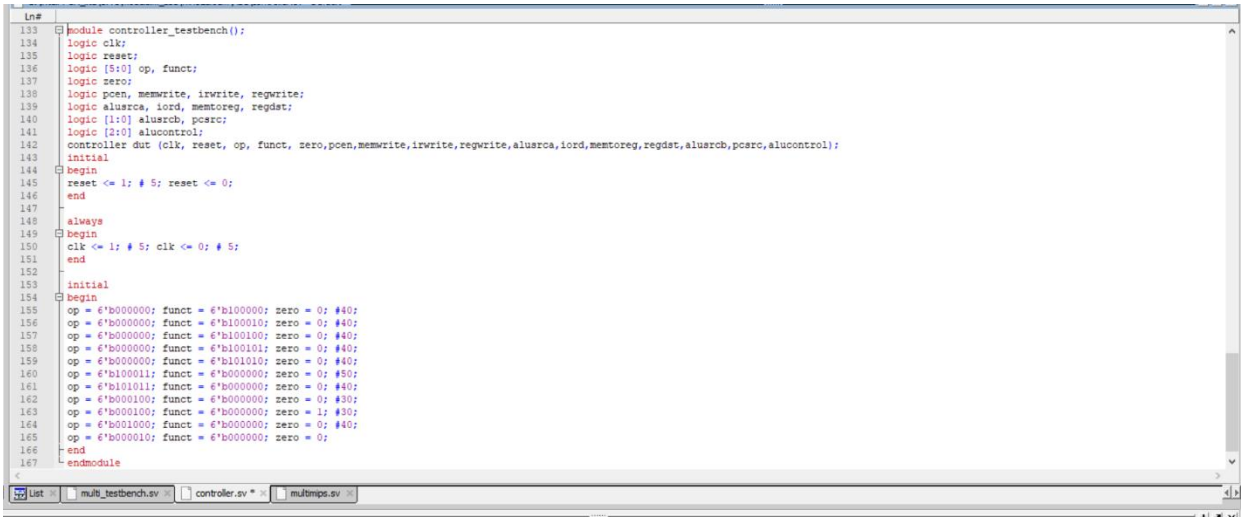
```
C:\intelFPGA_lite\17.1\modelsim_ase\win32aloem\HDL\controller.sv - Default
Ln#
77          default: nextstate = 4'bx;// should never happen
78          endcase
79    MEMRD:   nextstate = MEMWB;
80    MEMWB:   nextstate = FETCH;
81    MEMWR:   nextstate = FETCH;
82    RTYPEEX: nextstate = RTYPEWB;
83    RTYPEWB: nextstate = FETCH;
84    BEQEX:   nextstate = FETCH;
85    ADDIEX:  nextstate = ADDIWB;
86    ADDIWB:  nextstate = FETCH;
87    JEX:     nextstate = FETCH;
88    default: nextstate = 4'bx; // should never happen
89  endcase
90
91  //output logic
92  assign {pcwrite, memwrite, irwrite, regwrite, alusrca, branch, iord, memtoreg, regdst, alusrcb, pcsrc, aluop} = controls;
93
94  always_comb
95  case(state)
96    FETCH:   controls = 15'h5010;
97    DECODE:  controls = 15'h0030;
98    MEMADR:  controls = 15'h0420;
99    MEMRD:   controls = 15'h0100;
100   MEMWB:   controls = 15'h0880;
101   MEMWR:   controls = 15'h2100;
102   RTYPEEX: controls = 15'h0402;
103   RTYPEWB: controls = 15'h0840;
104   BEQEX:   controls = 15'h0605;
105   ADDIEX:  controls = 15'h0420;
106   ADDIWB:  controls = 15'h0800;
107   JEX:     controls = 15'h4008;
108   default: controls = 15'hxxxxx; // should never happen
List | multi_testbench.sv | controller.sv | multimips.sv
```

AluDec:



```
C:\intelFPGA_lite\17.1\modelsim_ase\win32aloem\HDL\controller.sv - Default
Ln#
113 module aludec(input logic [5:0] funct,
114               input logic [1:0] aluop,
115               output logic [2:0] alucontrol);
116
117 always_comb
118 case(aluop)
119 2'b00: alucontrol <= 3'b010; // add (for lw/sw/addi)
120 2'b01: alucontrol <= 3'b110; // sub (for beq)
121 default: case(funct) // R-type instructions
122          6'b100000: alucontrol <= 3'b010; // add
123          6'b100010: alucontrol <= 3'b110; // sub
124          6'b100100: alucontrol <= 3'b000; // and
125          6'b100101: alucontrol <= 3'b001; // or
126          6'b101010: alucontrol <= 3'b111; // slt
127          default:   alucontrol <= 3'bxxx; // should never happen
128          endcase
129 endcase
130
131 endmodule
132
133
```

Your controllertest testbench.



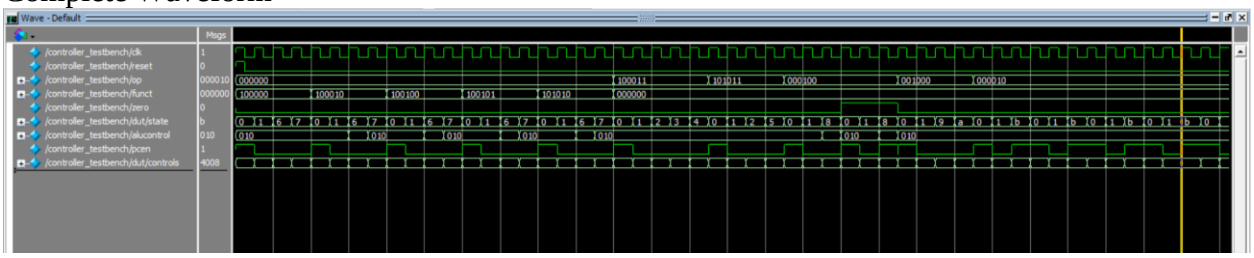
```
Ln#
133  module controller_testbench();
134  logic clk;
135  logic reset;
136  logic [5:0] op, funct;
137  logic zero;
138  logic pcen, memwrite, irwrite, regwrite;
139  logic alusrca, iord, memtoreg, regdst;
140  logic [1:0] alusrcb, pcsrc;
141  logic [2:0] alucontrol;
142  controller dut (clk, reset, op, funct, zero,pcen,memwrite,irwrite,regwrite,alusrca,iord,memtoreg,regdst,alusrcb,pcsrc,alucontrol);
143  initial
144  begin
145  reset <= 1; # 5; reset <= 0;
146  end
147
148  always
149  begin
150  clk <= 1; # 5; clk <= 0; # 5;
151  end
152
153  initial
154  begin
155  op = 6'b000000; funct = 6'b100000; zero = 0; #40;
156  op = 6'b000000; funct = 6'b100010; zero = 0; #40;
157  op = 6'b000000; funct = 6'b100100; zero = 0; #40;
158  op = 6'b000000; funct = 6'b100101; zero = 0; #40;
159  op = 6'b000000; funct = 6'b101010; zero = 0; #40;
160  op = 6'b100011; funct = 6'b000000; zero = 0; #50;
161  op = 6'b101011; funct = 6'b000000; zero = 0; #40;
162  op = 6'b000100; funct = 6'b000000; zero = 0; #30;
163  op = 6'b000100; funct = 6'b000000; zero = 1; #30;
164  op = 6'b001000; funct = 6'b000000; zero = 0; #40;
165  op = 6'b000010; funct = 6'b000000; zero = 0;
166  end
167  endmodule
```

List | multi_testbench.sv | controller.sv * | multimips.sv

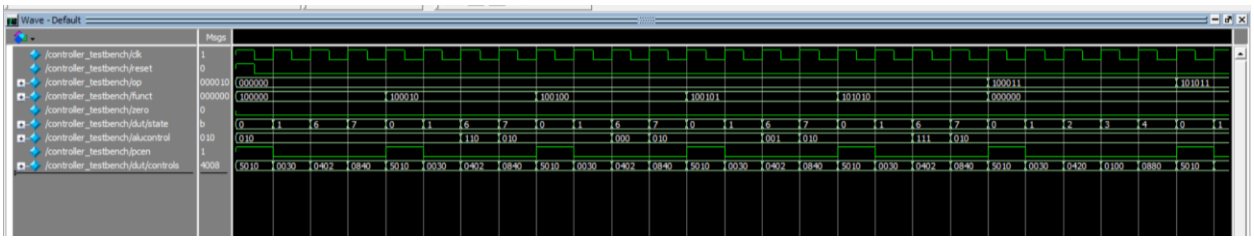
Simulation waveforms of the controller module showing (in the given order): *CLK, Reset, OP, Funct, Zero,* the *state* (this is an internal registered signal), *ALUControl, PCEn*, and the entire control word (i.e. the 4-nibble word you entered in Table 4) demonstrating each instruction (including taken and non-taken branches). Display all signals in hexadecimal. Does it match your expectations?

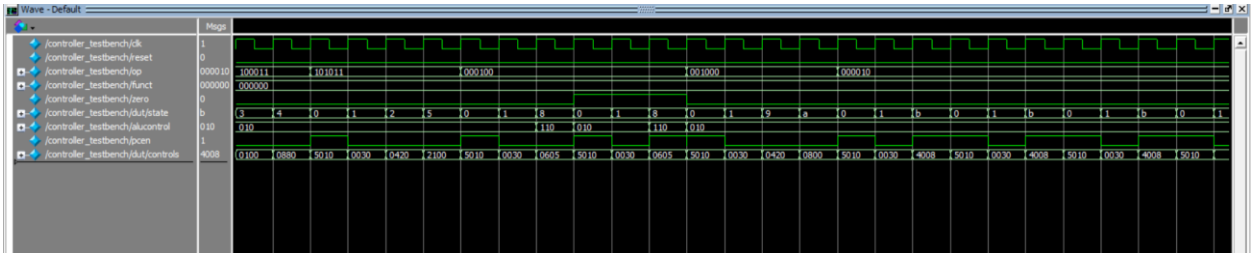
Yes, it matches to our expectations and we have received expected outputs.

Complete Waveform



Zoomed in Waveform





A completed copy of Table 5 indicating the expected outcome of running the test program.

Cycle	Reset	PC	Instr	(FSM) State	SrcA	Src B	AluResult	Zero	FSM Control Word
1	1	00	0	0	0	04	04	0	5010
2	0	04	addi 20020005	1	04	x	x	0	0030
3	0	04	addi 20020005	9	00	05	05	0	0420
4	0	04	addi 20020005	10	x	x	x	0	0800
5	0	04	addi 20020005	0	04	04	08	0	5010
6	0	08	addi 2003000c	1	08	x	x	0	0030
7	0	08	addi 2003000c	9	00	0C	0C	0	0420
8	0	08	addi 2003000c	10	x	x	x	0	0800
9	0	08	addi 2003000c	0	08	04	0C	0	5010
10	0	0C	addi 2067fff7	1	0C	x	x	0	0030
11	0	0C	addi 2067fff7	9	0C	fff7	03	0	0420
12	0	0C	addi 2067fff7	10	x	x	x	0	0800
13	0	0C	addi 2067fff7	0	0C	04	10	0	5010
14	0	10	or 00e22025	1	10	x	x	0	0030
15	0	10	or 00e22025	6	03	05	07	0	0402
16	0	10	or 00e22025	7	x	x	x	0	0840
17	0	10	or 00e22025	0	10	04	14	0	5010
18	0	14	and 00642824	1	14	x	x	0	0030
19	0	14	and 00642824	6	0C	07	04	0	0402
20	0	14	and 00642824	7	x	x	x	0	0840
21	0	14	and 00642824	0	14	04	18	0	5010
22	0	18	add 00a42820	1	18	x	x	0	0030
23	0	18	add 00a42820	6	04	07	0B	0	0402
24	0	18	add 00a42820	7	x	x	x	0	0840
25	0	18	add 00a42820	0	18	04	1C	0	5010
26	0	1C	beq 10a7000a	1	1C	x	x	0	0030
27	0	1C	beq 10a7000a	8	0B	03	08	0	0605
28	0	1C	beq 10a7000a	0	1C	04	20	0	5010
29	0	20	slt 0064202a	1	20	x	x	0	0030
30	0	20	slt 0064202a	6	1C	07	0	1	0402
31	0	20	slt 0064202a	7	x	x	x	0	0840
32	0	20	slt 0064202a	0	20	04	24	0	5010
33	0	24	beq 10800001	1	24	x	x	0	0030
34	0	24	beq 10800001	8	00	00	00	1	0605
35	0	24	beq 10800001	0	24	08	2C	0	5010
36	0	2C	slt 00e2202a	1	2C	x	x	0	0030
37	0	2C	slt 00e2202a	6	03	05	01	0	0402
38	0	2C	slt 00e2202a	7	x	x	x	0	0840

39	0	2C	slt 00e2202a	0	28	04	30	0	5010
40	0	30	add 00853820	1	30	x	x	0	0030
41	0	30	add 00853820	6	01	0B	0C	0	0402
42	0	30	add 00853820	7	x	x	x	0	0840
43	0	30	add 00853820	0	30	04	34	0	5010
44	0	34	sub 00e23822	1	34	x	x	0	0030
45	0	34	sub 00e23823	6	0C	05	07	0	0402
46	0	34	sub 00e23824	7	x	x	x	0	0840
47	0	34	sub 00e23825	0	34	04	38	0	5010
48	0	38	sw ac670044	1	38	x	x	0	0030
49	0	38	sw ac670044	2	0C	44	50	0	0420
50	0	38	sw ac670044	5	x	x	x	0	2100
51	0	38	sw ac670044	0	38	04	3C	0	5010
52	0	3C	lw 8c020050	1	3C	x	x	x	0030
53	0	3C	lw 8c020050	2	00	50	50	0	0420
54	0	3C	lw 8c020050	3	x	x	x	0	0100
55	0	3C	lw 8c020050	4	x	x	x	0	0880
56	0	3C	lw 8c020050	0	3C	04	40	0	5010
57	0	40	j 08000011	1	40	x	x	x	0030
58	0	40	j 08000011	11	40	x	x	x	4008
59	0	40	j 08000011	0	44	04	48	0	5010
60	0	48	sw ac020054	1	48	x	x	x	0030
61	0	48	sw ac020054	2	00	54	54	0	0420
62	0	48	sw ac020054	5	x	x	x	0	2100

SystemVerilog code of the datapath

```
module datapath(input  logic clk, reset,
input  logic pcen, irwrite, regwrite,
input  logic alusrca, iord, memtoreg, regdst,
input  logic [1:0]  alusrcb, pcsrc,
input  logic [2:0]  alucontrol,
output logic [5:0]  op, funct,
output logic zero,
output logic [31:0] adr, writedata,
input  logic [31:0] readdata);

//Internal signals of the datapath module.

logic [4:0]  writereg;
logic [31:0] pcnext, pc;
logic [31:0] instr, data, srca, srcb;
logic [31:0] a;
logic [31:0] aluresult, aluout;
logic [31:0] signimm;   // the sign-extended immediate
logic [31:0] signimmsh; // the sign-extended immediate shifted left by 2
logic [31:0] wd3, rd1, rd2;
logic [31:0] jsignimm;

//op and funct fields to controller
assign op = instr[31:26];
assign funct = instr[5:0];

flopr #(32) pcreg(clk, reset,pcen, pcnext, pc);

mux2 #(32) adrchoose(pc, aluout, iord,  adr);
```

```
flopr #(32) instrchoose(clk, reset,irwrite, readdata, instr);

flop #(32) datachoose(clk, reset, readdata, data);

mux2 #(5) wrmux(instr[20:16], instr[15:11], regdst, writereg);

mux2 #(32) resmux(aluout, data, memtoreg, wd3);

regfile rf(clk, regwrite, instr[25:21], instr[20:16], writereg, wd3, rd1,rd2);

flop2 #(32) srcreg(clk, reset, rd1, rd2, a,writedata);

mux2 #(32) srcachoice(pc, a, alusrca,  srca);

signext se(instr[15:0], signimm);

sl2 immsh(signimm, signimmsh);

mux4  #(32) srcbchoice (writedata,32'b100,signimm,signimmsh,alusrcb,srcb);

aluproj alu(srca, srcb, alucontrol, aluresult, zero);

flop #(32) aluchoose(clk, reset, aluresult, aluout);

jsignim jsi(instr,jsignimm);

mux3 aluchoice(aluresult,aluout,jsignimm,pcsrc,pcnext);

endmodule
```

```
//----------------------------Register File-----------------------------------------
module regfile(input logic clk,
input logic we3,
input logic [4:0] ra1, ra2, wa3,
input logic [31:0] wd3,
output logic [31:0] rd1, rd2);

logic [31:0] rf[31:0];

always_ff @(posedge clk)
if (we3)
rf[wa3] <= wd3;

assign rd1=(ra1 !=0) ? rf[ra1] : 0;
assign rd2=(ra2 !=0) ? rf[ra2] : 0;

endmodule
//-------------------------------------------------------------------------------
//-------------------------------Alu-------------------------------------------------
module aluproj(input logic [31:0] a, b,
input logic [2:0] F,
output logic [31:0] Y,
output logic zero);
logic [31:0] S , B;

assign B = F[2]? ~b:b;

assign S = a + B + F[2];
```

```
always_comb
case(F[1:0])
2'b00 : Y = a & B;
2'b01 : Y = a | B;
2'b10 : Y = S;
2'b11 : Y = {31'b0,S[31]};
endcase


always_comb
if ( Y === 32'b0)
zero = 1;
else zero =0;

endmodule
//-------------------------------------------------------------------------------------
//----------------------To store values into register-----------------------------------
module flop #(parameter WIDTH=8) (input logic clk, reset,
input logic [WIDTH-1:0] d,
output logic [WIDTH-1:0] q);

always_ff @(posedge clk, posedge reset)
if (reset) q <= 0;
else  q <= d;

endmodule
//-------------------------------------------------------------------------------------
//--------------------------------Register to save A and B values-----------------------
module flop2 #(parameter WIDTH=8) (input logic clk, reset,
input logic [WIDTH-1:0] d1, d2,
```

```
output logic [WIDTH-1:0] q1,q2);

always_ff @(posedge clk, posedge reset)
if (reset)
begin
q1 <= 0;
q2 <= 0;
end
else
begin
q1 <= d1;
q2 <= d2;
end

endmodule
//-------------------------------------------------------------------------------------
//---------------------- To store pc value only when en is 1-------------------------------
module flopr #(parameter WIDTH=8) (input logic clk, reset, pcen,
input logic [WIDTH-1:0] d,
output logic [WIDTH-1:0] q);

always_ff @(posedge clk, posedge reset)
if (reset) q <= 0;
else if (pcen) q <= d;

endmodule
//-------------------------------------------------------------------------------------
//---------------Sign extend for immediate values---------------------------------------
module signext(input logic [15:0] a,
output logic [31:0] y);
```

```
assign y={{16{a[15]}}, a};

endmodule
//------------------------------------------------------------------------------
//----------------Left shift for beq instruction----------------------------------
module sl2(input logic [31:0] a,
output logic [31:0] y);

assign y={a[29:0], 2'b00};

endmodule
//------------------------------------------------------------------------------
//----------------Left shift for Jump instruction---------------------------------
module jsignim(input logic [31:0] a,
output logic [31:0] y);

assign y={a[31:28], {a[25:0], 2'b00}};

endmodule
//------------------------------------------------------------------------------
//-------------------------------MUX for 2 choices--------------------------------
module mux2 #(parameter WIDTH=8) (input logic [WIDTH-1:0] d0, d1,
input logic s,
output logic [WIDTH-1:0] y);

assign y=s ? d1 : d0;

endmodule
//------------------------------------------------------------------------------
```

```
//--------------------------------MUX for 3 choices-----------------------------------
module mux3 #(parameter WIDTH = 8) (input  logic [WIDTH-1:0] d0, d1, d2,
input  logic [1:0] s,
output logic [WIDTH-1:0] y);

assign #1 y = s[1] ? d2 : (s[0] ? d1 : d0);
endmodule
//-------------------------------------------------------------------------------
//----------------------Mux for 4 choices-------------------------------------------
module mux4 #(parameter WIDTH = 8) (input  logic [WIDTH-1:0] d0, d1, d2, d3,
input  logic [1:0] s,
output logic [WIDTH-1:0] y);
always_comb
case(s)
2'b00: y = d0;
2'b01: y = d1;
2'b10: y = d2;
2'b11: y = d3;
endcase
endmodule
//-------------------------------------------------------------------------------
```

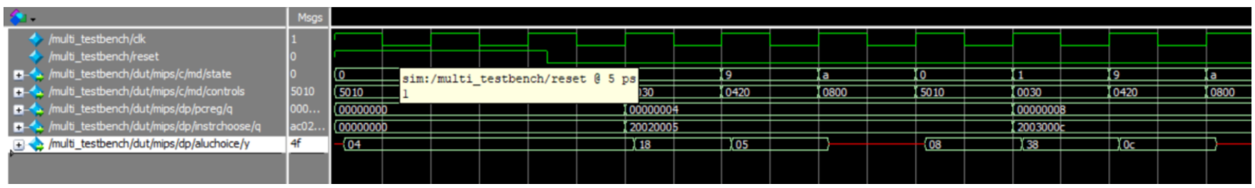
Simulation waveforms of the processor showing *CLK, Reset, state, PC, instr, and ALUResult* in this order while running the test program. As always, output the values in he (or decimal if that is more readable) and make sure they are readable. Do the results match your expectations? Does the program indicate Simulation Succeeded?

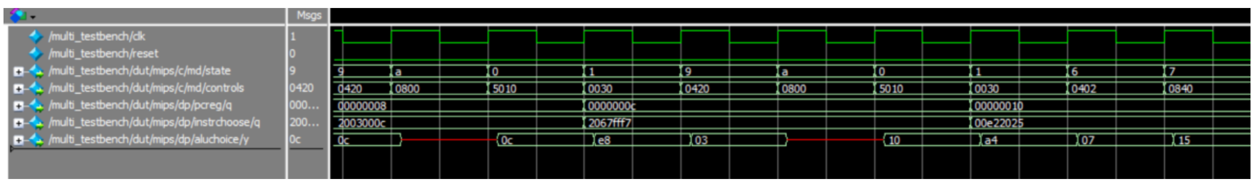
Note: pcreg/q = PC

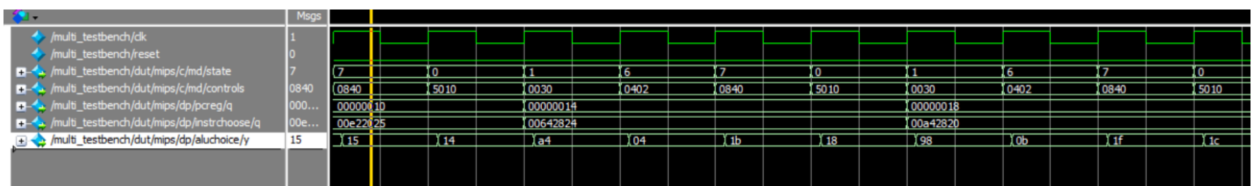
Instrchoose/q = instruction

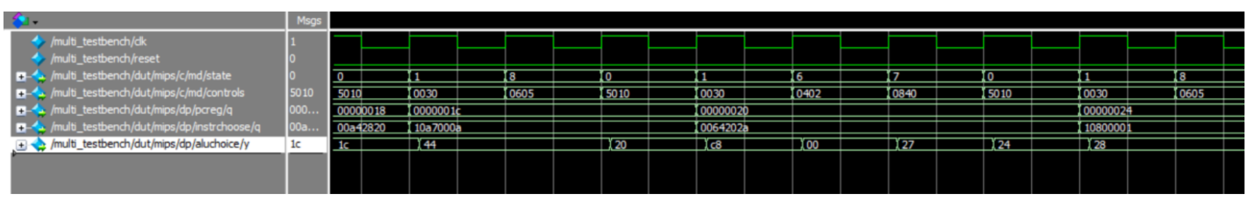
Aluchoice/y = aluout

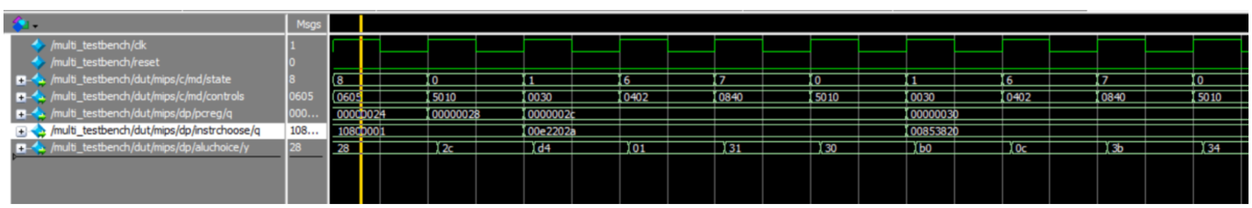
Yes, it matches to our expectations and we have received expected outputs in the waveform as expected from the table 5.

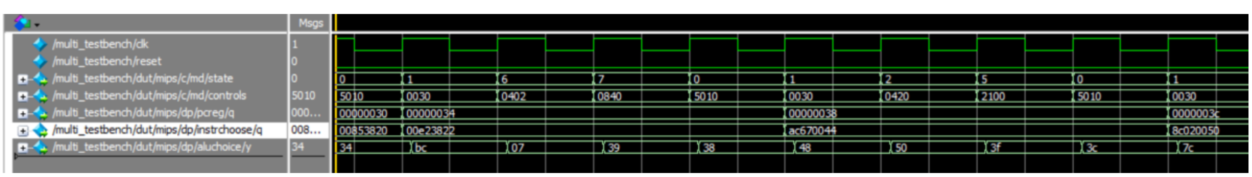


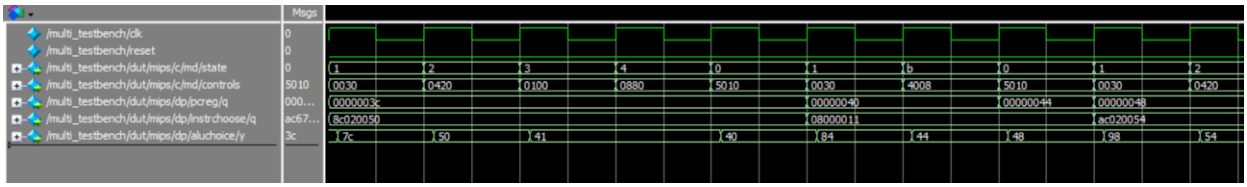


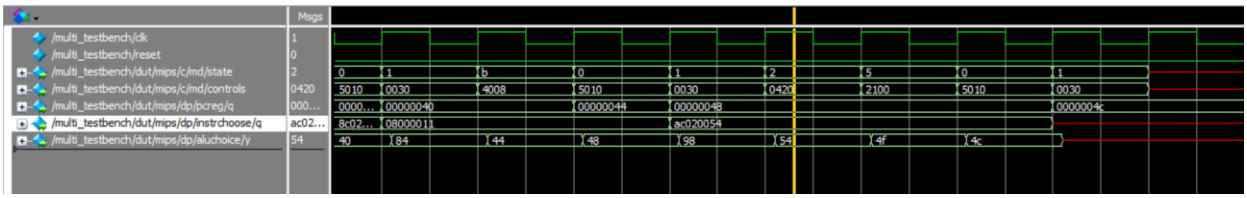




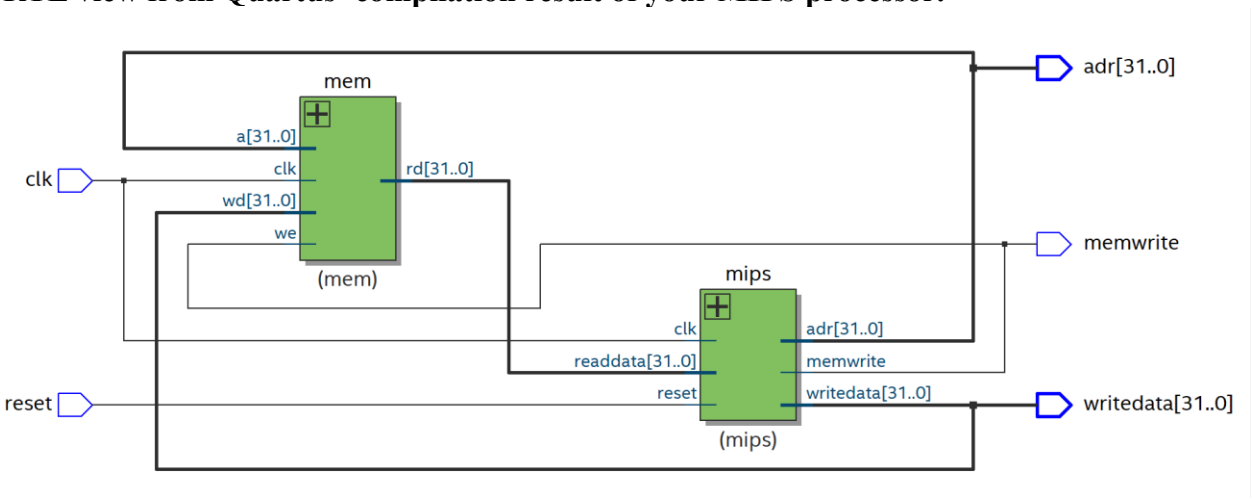


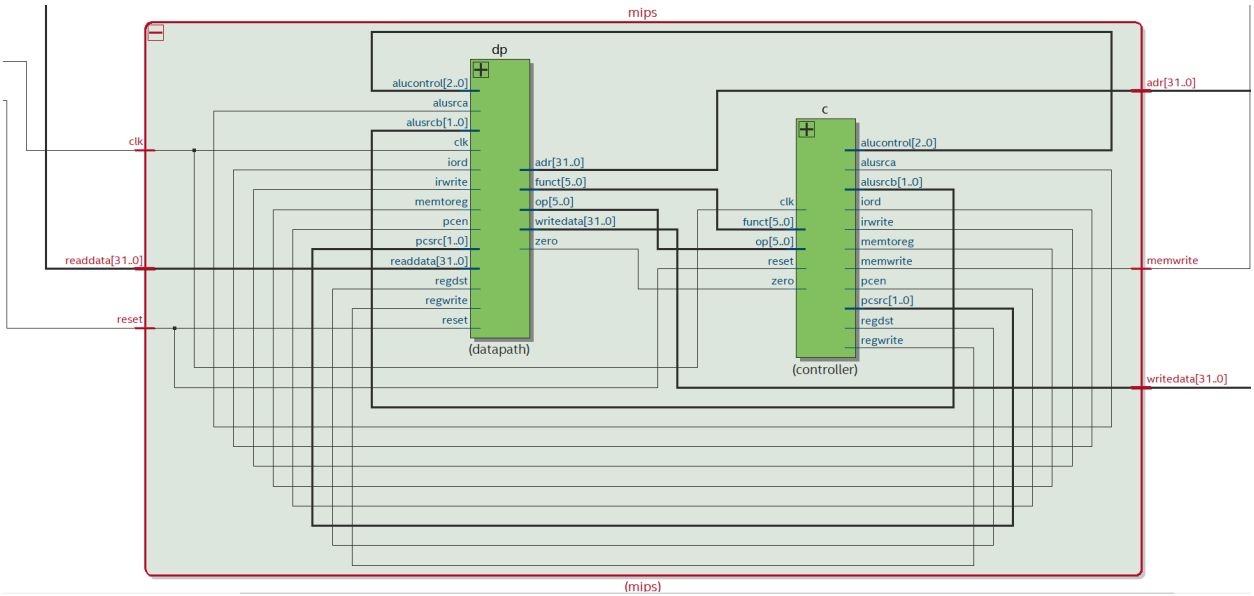



Msgs
/multi_testbench/clk
/multi_testbench/reset
/multi_testbench/dut/mips/c/md/state
/multi_testbench/dut/mips/c/md/controls
/multi_testbench/dut/mips/dp/pcreg/q
/multi_testbench/dut/mips/dp/instrchoose/q
/multi_testbench/dut/mips/dp/aluchoice/y
0
0
0
5010
000...
ac67...
3c
1 2 3 4 0 1 b 0 1 2
0030 0420 0100 0880 5010 0030 4008 5010 0030 0420
0000003c 00000040 00000044 00000048
8c020050 08000011 ac020054
7c 50 41 40 84 44 48 98 54

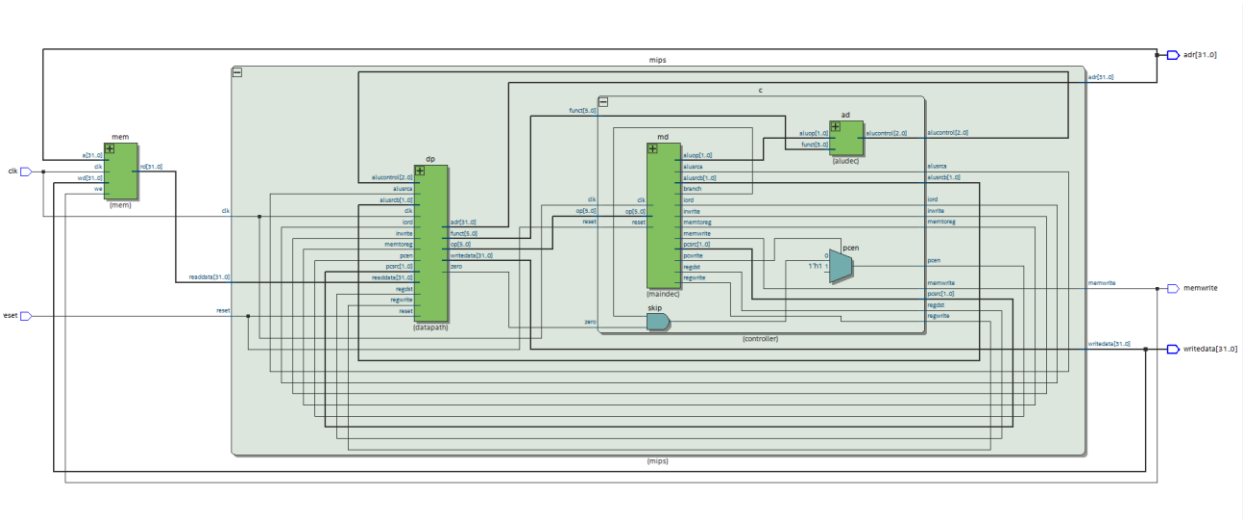

Msgs
/multi_testbench/clk
/multi_testbench/reset
/multi_testbench/dut/mips/c/md/state
/multi_testbench/dut/mips/c/md/controls
/multi_testbench/dut/mips/dp/pcreg/q
/multi_testbench/dut/mips/dp/instrchoose/q
/multi_testbench/dut/mips/dp/aluchoice/y
1
0
2
0420
000...
ac02...
54
0 1 b 0 1 2 5 0 1
5010 0030 4008 5010 0030 0420 2100 5010 0030
0000... 00000040 00000044 00000048 0000004c
8c02... 08000011 ac020054
40 84 44 48 98 54 4f 4c

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

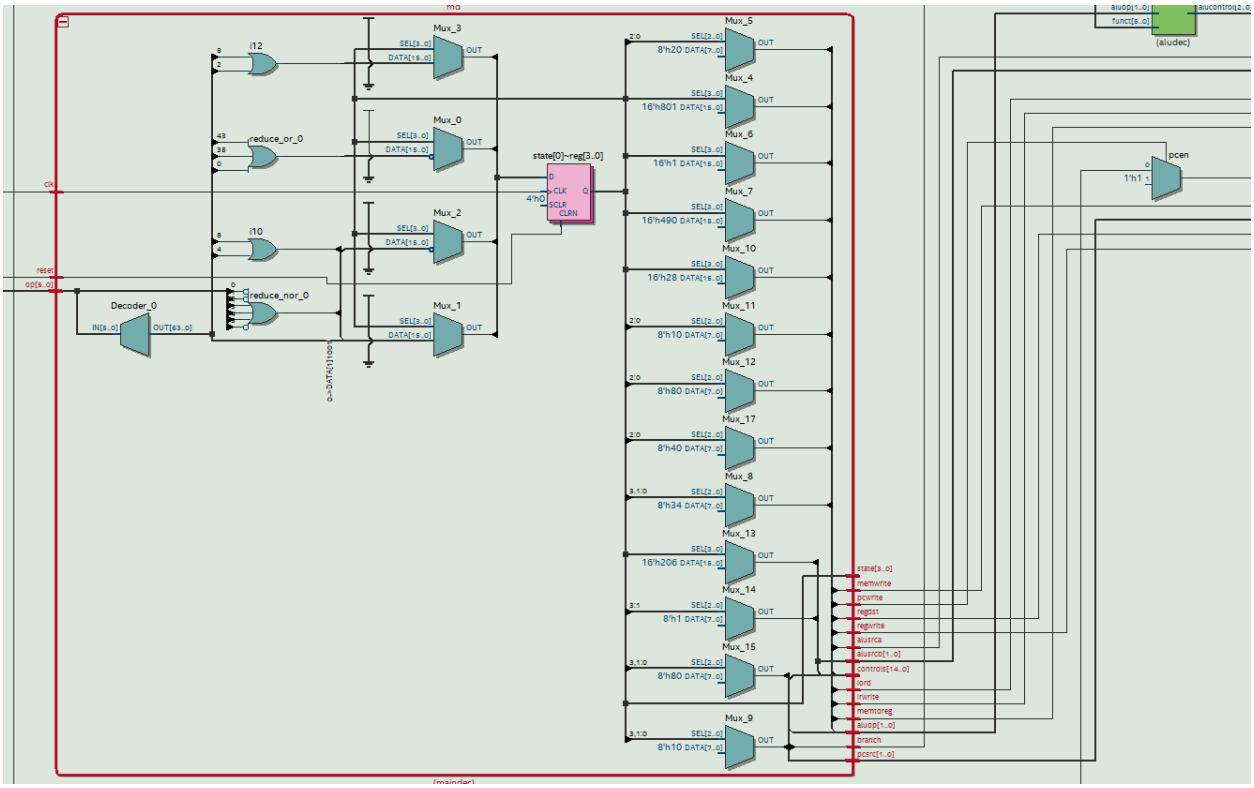




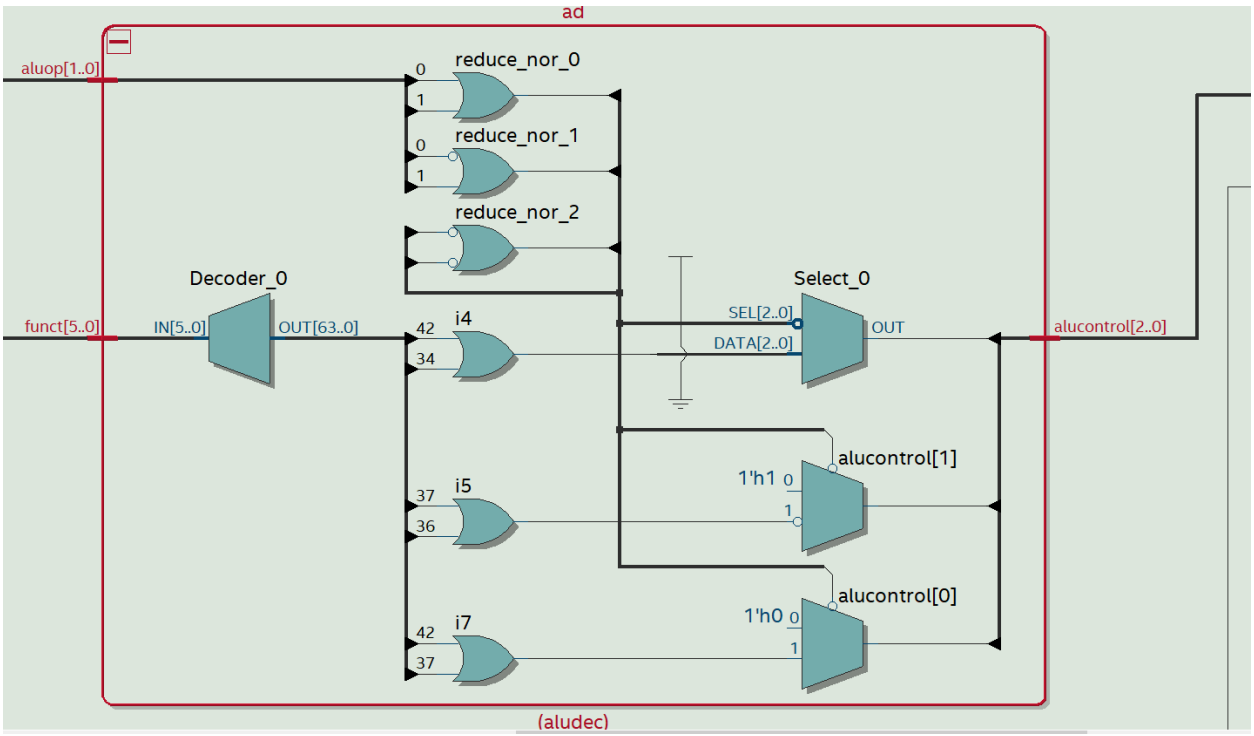
Controller



Maindec



Aludec



Datapath