UNIVERSITY OF NEVADA LAS VEGAS. DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING LABORATORIES.

Class:	CPE 300L - 1001	Semester:	Fall 2021

Points		Document author:	Dylan Flores
		Author's email:	flored16@unlv.nevada.edu
		Document topic:	Final Project

Instructor's comments:

1. Control Word Table

Test Code 1:

ADDRESS	Label	Instruction	MACHINE CODE	Comment
0		LDAC 0	10	$AC = 55
1		INAC	A0	$AC = 56
2		JPNZ 4	74	Taken
3		JUMP 0	50	Not taken
4		INAC	A0	$AC = 57
5		MVAC	30	$R = 57
6		ADD	80	$AC = 57 + 57 = 114
7		STAC 2	22	M[2] = 114
8		NOP	00	No operation

Test Code 2:

ADDRESS	Label	Instruction	MACHINE CODE	Comment
0		CLAC	B0	$AC = 0
1		INAC	A0	$AC = 1
2		MVAC	31	$R = 1
3		NOT	F0	$AC = 254
4		XOR	E1	$AC = 254 ^ 1 = 255
5		JMPZ 10	6A	Not taken
6		STAC 4	24	M[4] = 255
7		NOP	00	$AC = 255
8		NOP	00	No operation

2. Machine for Test Codes

Machine Code memfile.dat 1:

10
a0
74
50
a0
30
80
22
00

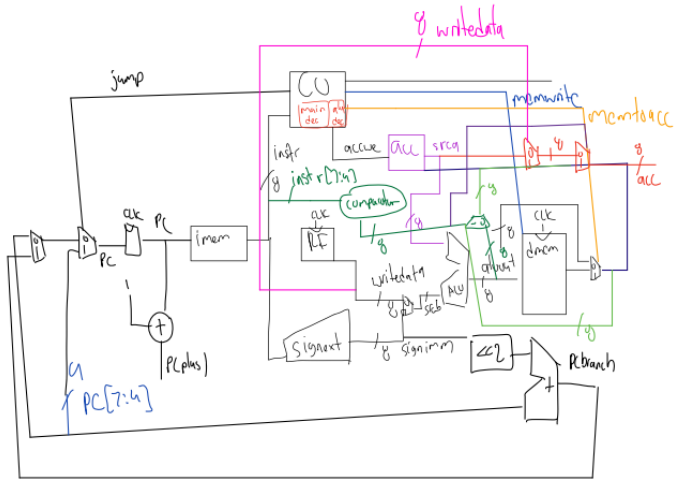
Machine Code memfile.dat 2:

b0
a0
31
f0
e1
6a
24
00
00

3. Deliveries

Experiment 1:

Block Diagram:



Experiment 2:

a) Verilog Code

imem:

```
module imem (input [5:0] a, output [7:0] rd);
reg [7:0] RAM[63:0]; // limited memory
initial
begin
$readmemh ("memfile.dat",RAM);
end
assign rd = RAM[a]; // word aligned
endmodule
```

maindec:

```
module maindec (input [7:0] instr, output accwe, memtoacc, memwrite, output branch, alusrc,
output regdst, regwrite, output jump, output [1:0] aluop, output regtoacc, input acc_reg_is_zero);

reg [10:0] controls;
reg Z;

assign {accwe, regwrite, regdst, alusrc, branch, memwrite, memtoacc, jump, aluop, regtoacc}  = controls;
always @ (*) begin
    Z = ~acc_reg_is_zero;
    casex(instr)
        8'b0: controls <= 11'b0; // NOP
        8'b0001xxxx: controls <= 11'b10010010_00_0; // AC = M[T]
        8'b0010xxxx: controls <= 11'b00010100_00_0; // M[T] = AC
        8'b0011xxxx: controls <= 11'b01000000_00_0; // R = AC
        8'b0100xxxx: controls <= 11'b10000000_00_1; // AC = R
        8'b0101xxxx: controls <= 11'b00000001_00_0; // J to memory addr
        8'b0110xxxx: if(Z == 1) controls <= 11'b00000001_00_0; // JMPZ z == 1, jump to addr
                     else controls <= 11'b0;
        8'b0111xxxx: if(Z == 0) controls <= 11'b00000001_00_0; // JPNZ z == 0 jump to addr
                     else controls <= 11'b0;
        8'b1000xxxx: controls <= 11'b10010000_00_0; // ADD
        8'b1001xxxx: controls <= 11'b10010000_00_0; // SUB
        8'b1010xxxx: controls <= 11'b10010000_00_0; // INAC
        8'b1011xxxx: controls <= 11'b10010000_00_0; // CLAC
        8'b1100xxxx: controls <= 11'b10010000_00_0; // AND
        8'b1101xxxx: controls <= 11'b10010000_00_0; // OR
        8'b1110xxxx: controls <= 11'b10010000_00_0; // XOR
        8'b1111xxxx: controls <= 11'b10010000_00_0; // NOT
        default: controls <= 11'bx;
    endcase

end

endmodule
```

mips:

```
module mips (input clk, reset,
output [7:0] pc,
input [7:0] instr, output[7:0] acc,
output memwrite,
output [7:0] aluout, writedata,
input [7:0] readdata);
wire accwe, memtoacc, branch, alusrc, regdst, regwrite, jump, regtoacc;
wire [3:0] alucontrol;
wire acc_reg_is_zero;
controller c(instr, zero, accwe, memtoacc, memwrite, pcsrc,
alusrc, regdst, regwrite, jump, regtoacc, alucontrol, acc_reg_is_zero);
datapath dp(clk, reset, accwe, memtoacc, pcsrc,
alusrc, regdst, regwrite, jump, regtoacc,
alucontrol,
zero, pc, instr, acc,
aluout, writedata, readdata, acc_reg_is_zero);
endmodule
```

mux2:

```
module mux2 # (parameter WIDTH = 8)
(input [WIDTH-1:0] d0, d1, input s,
output [WIDTH-1:0] y);
assign y = s ? d1 : d0;
endmodule
```

regfile:

```
module regfile (input clk, input we3,
input [3:0] ra1, ra2, wa3,
input [7:0] wd3,
output [7:0] rd1, rd2);
reg [7:0] rf;

always @ (posedge clk)
    if (we3) rf <= wd3;
assign rd1 = rf;
assign rd2 = rf;
endmodule
```

signext:

```
module signext (input [7:0] a,
output [7:0] y);
assign y = a;
endmodule
```

sl2:

```
module sl2 (input [7:0] a, output [7:0] y);
// shift left by 2
assign y = {a[5:0], 2'b00};
endmodule
```

top:

```
module top (input clk, reset, output [7:0] writedata, dataadr, pc_display, instr_display, acc_display, output memwrite);
wire [7:0] pc, instr, readdata, acc;
assign pc_display = pc;
assign instr_display = instr;
assign acc_display = acc;

// instantiate processor and memories
mips mips_dut (clk, reset, pc, instr, acc, memwrite, dataadr,
writedata, readdata);
imem imem_dut (pc[5:0], instr);
dmem dmem_dut (clk, memwrite, instr[3:0], acc, readdata);
endmodule
```

accumulator:

```
module accumulator(input [7:0] result,
                   input clk, we, clr,
                   output reg [7:0] accout);

    always @(posedge clk) begin
        if(clr == 1)
            accout <= 8'b0;
        else if(we == 1)
            accout <= result;
    end
endmodule
```

adder:

```
module adder (input [7:0] a, b, output [7:0] y);
assign y = a + b;
endmodule
```

alu part 1:

See below.

```
module alu (a,b,sel, out, zero);
    input [7:0] a,b;
    input [3:0] sel;
    output reg [7:0] out;
    output reg zero;

  initial
  begin
  out = 0;
  zero =1'b0;
  end
    always @ (*)
        case(sel)
            4'b0000: begin
                out = a;
                if (out == 0)
                    zero = 1;
                else
                    zero = 0;
            end
            4'b1000: begin
                out = a + b;
                if (out == 0)
                    zero = 1;
                else
                    zero = 0;
            end
            4'b1001: begin
                out = a - b;
                if (out == 0)
                    zero = 1;
                else
                    zero = 0;
            end
        4'b1010: begin
            out = a + 1;
            if (out == 0)
                zero = 1;
            else
            zero = 0;
        end
        4'b1011: begin
            out = 0;
            zero = 1;
        end
        4'b1100: begin
            out = a & b;
            if (out == 0)
                zero = 1;
```

alu part 2:

```
            else
                zero = 0;
        end
        4'b1101: begin
            out = a | b;
            if (out == 0)
                zero = 1;
            else
                zero = 0;
    end
        4'b1110: begin
            out = a ^ b;
            if (out == 0)
                zero = 1;
            else
                zero = 0;
        end
        4'b1111: begin
            out = ~a;
            if (out == 0)
                zero = 1;
            else
                zero = 0;
        end
    endcase
endmodule
```

aludec:
See below.

```
module aludec (input [7:0] instr,
input [1:0] aluop,
output reg [3:0] alucontrol);
always @ (*) begin
    casex(instr)
    8'b0: alucontrol <= 4'b0000; // NOP
    8'b1000xxxx: alucontrol <= 4'b1000; // ADD
    8'b1001xxxx: alucontrol <= 4'b1001; // SUB
    8'b1010xxxx: alucontrol <= 4'b1010; // INAC
    8'b1011xxxx: alucontrol <= 4'b1011; // CLAC
    8'b1100xxxx: alucontrol <= 4'b1100; // AND
    8'b1101xxxx: alucontrol <= 4'b1101; // OR
    8'b1110xxxx: alucontrol <= 4'b1110; // XOR
    8'b1111xxxx: alucontrol <= 4'b1111; // NOT
    default: alucontrol <= 4'bxxxx; // ???
    endcase
end
endmodule
```

controller:

```
module controller (input [7:0] instr,
input zero,
output accwe, memtoacc, memwrite,
output pcsrc, alusrc,
output regdst, regwrite,
output jump, regtoacc,
output [3:0] alucontrol,
input acc_reg_is_zero);
wire [1:0] aluop;
wire branch;


maindec md(instr, accwe, memtoacc, memwrite, branch,
alusrc, regdst, regwrite, jump,
aluop, regtoacc, acc_reg_is_zero);
aludec ad (instr, aluop, alucontrol);

assign pcsrc = branch & zero;
endmodule
```

flopr:

```
module flopr # (parameter WIDTH = 8)
(input clk, reset,
input [WIDTH-1:0] d,
output reg [WIDTH-1:0] q);
always @ (posedge clk, posedge reset)
if (reset) q <= 0;
else q <= d;
endmodule
```

dmem:
See below.

```
module dmem (input clk, we,
input [3:0] a,
input [7:0] wd,
output [7:0] rd);

reg [7:0] RAM[15:0];
assign rd = RAM[a];

initial begin
    RAM[0] <= 55;
    RAM[1] <= 29;
    RAM[2] <= 0;
    RAM[3] <= 0;
    RAM[4] <= 0;
    RAM[5] <= 0;
    RAM[6] <= 0;
    RAM[7] <= 0;
    RAM[8] <= 0;
    RAM[9] <= 0;
    RAM[10] <= 0;
    RAM[11] <= 0;
    RAM[12] <= 0;
    RAM[13] <= 0;
    RAM[14] <= 0;
    RAM[15] <= 0;
    // RAM[15:0] = 0;
end
always @ (posedge clk)
if (we)
    RAM[a] <= wd;
endmodule
```

datapath:

```
module datapath (input clk, reset,
input accwe,
input memtoacc, pcsrc,
input alusrc, regdst,
input regwrite, jump, regtoacc,
input [3:0] alucontrol,
output zero,
output [7:0] pc,
input [15:0] instr, output[7:0] acc,
output [7:0] aluout, writedata,
input [7:0] readdata,
output acc_reg_is_zero);

wire [3:0] writereg;
wire [7:0] pcnext, pcnextbr, pcplus1, pcbranch;
wire [7:0] signimm, signimmsh;
wire [7:0] srca, srcb, srcc;
wire [7:0] result, srcb_result, acc_result, acc_ld_result, acc_final;

wire accwedata;
assign accwedata = (instr[7:4] == 4'b0001) ? 1 : 0; // LDAC

assign acc = srca;
mux2 #(8) regtoa(aluout, writedata, regtoacc, acc_result);
mux2 #(8) ldactoacc(aluout, readdata, accwedata, acc_ld_result);
assign acc_final = (accwedata == 0) ? acc_result : acc_ld_result;
accumulator accumulator(acc_final, clk, accwe, reset, srca);

assign acc_reg_is_zero = srca[7] | srca[6] | srca[5] | srca[4] | srca[3] | srca[2] | srca[1] | srca[0];
// next PC logic
flopr #(8) pcreg(clk, reset, pcnext, pc);
adder pcadd1 (pc, 8'b1, pcplus1);
sl2 immsh(signimm, signimmsh);
adder pcadd2(pcplus1, signimmsh, pcbranch);
mux2 #(8) pcbrmux(pcplus1, pcbranch, pcsrc, pcnextbr);
mux2 #(8) pcmux(pcnextbr, {pcplus1[7:4], instr[3:0]},jump, pcnext);
// register file logic
regfile rf(clk, regwrite, instr[3:0],
instr[3:0], writereg, srca, writedata, srcc);
mux2 #(4) wrmux(instr[3:0], instr[3:0],regdst, writereg);
mux2 #(8) resmux(aluout, readdata, memtoacc, result);
signext se(instr, signimm);
// ALU logic
mux2 #(8) srcbmux(result, instr, alusrc, srcb_result);
alu alu(srca, writedata, alucontrol, aluout, zero);

endmodule
```

dmem for Test Code 2:

```
module dmem (input clk, we,
input [3:0] a,
input [7:0] wd,
output [7:0] rd);

reg [7:0] RAM[15:0];
assign rd = RAM[a];

initial begin
    RAM[0] <= 0;
    RAM[1] <= 0;
    RAM[2] <= 0;
    RAM[3] <= 0;
    RAM[4] <= 0;
    RAM[5] <= 0;
    RAM[6] <= 0;
    RAM[7] <= 0;
    RAM[8] <= 0;
    RAM[9] <= 0;
    RAM[10] <= 0;
    RAM[11] <= 0;
    RAM[12] <= 0;
    RAM[13] <= 0;
    RAM[14] <= 0;
    RAM[15] <= 0;
end
always @ (posedge clk)
if (we)
    RAM[a] <= wd;
endmodule
```

All Addresses initialized to 0.

b) Testbench for Test Code 1:

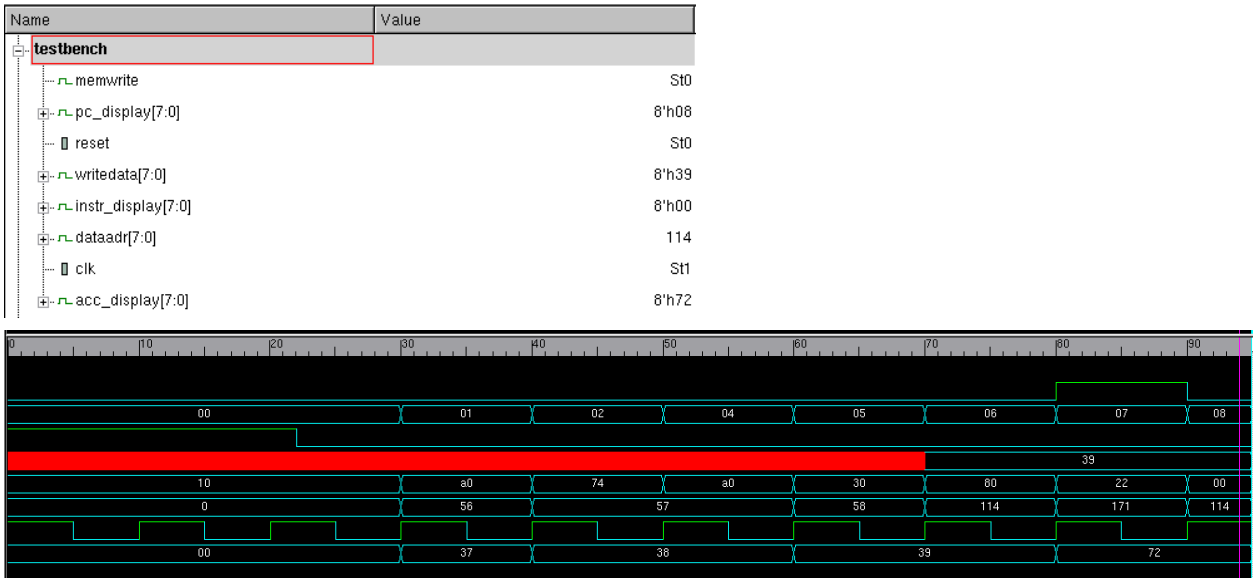
```
module testbench();
reg clk;
reg reset;
wire [7:0] writedata, dataadr, pc_display, instr_display, acc_display;
wire memwrite;
// instantiate device to be tested
top dut (clk, reset, writedata, dataadr, pc_display, instr_display, acc_display, memwrite);
// generate clock to sequence tests
// initialize test
initial
begin
reset <= 1; # 22; reset <= 0;
end
always
begin
clk <= 1;
 # 5;
 clk <= 0;
 # 5; // clock duration
end
// check results
always @ (negedge clk)
begin
if (acc_display == 114 && writedata == 57 && dataadr == 114) begin
$display ("Simulation succeeded");
$stop;
end
end
endmodule
```

Testbench for Test Code 2:

```
module testbench();
reg clk;
reg reset;
wire [7:0] writedata, dataadr, pc_display, instr_display, acc_display;
wire memwrite;
// instantiate device to be tested
top dut (clk, reset, writedata, dataadr, pc_display, instr_display, acc_display, memwrite);
// generate clock to sequence tests
// initialize test
initial
begin
reset <= 1; # 22; reset <= 0;
end
always
begin
clk <= 1;
 # 5;
 clk <= 0;
 # 5; // clock duration
end
// check results
always @ (negedge clk)
begin
if (acc_display == 255 && writedata == 1 && dataadr == 255) begin
$display ("Simulation succeeded");
$stop;
end
end
endmodule
```

c)

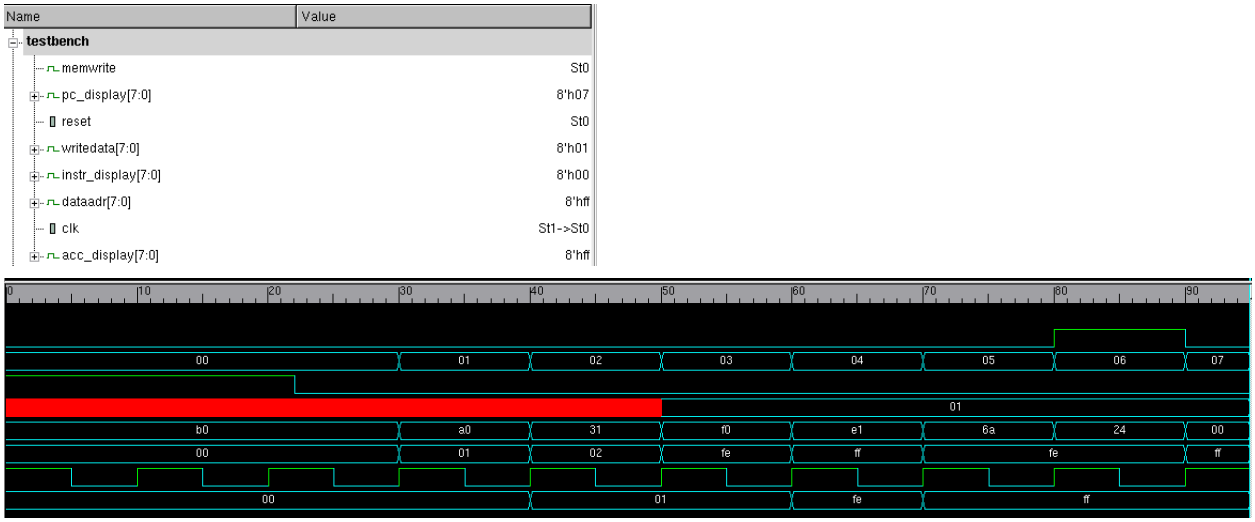
Test Code 1 Waveform:



Console Sim:

```
Simulation succeeded
```

Test Code 2 Waveform:



Console Sim:

```
Simulation succeeded
```

d) Video Delivery

Test Code 1 Link to Video:
https://drive.google.com/file/d/1lEhljz75PPeXYCl4Avg_o381FwMLwYDX/view?usp=sharing

Test Code 2 Link to Video:
https://drive.google.com/file/d/1Y3gNmOcY66FVxy0rkSkeSTIf-be3dI0m/view?usp=sharing

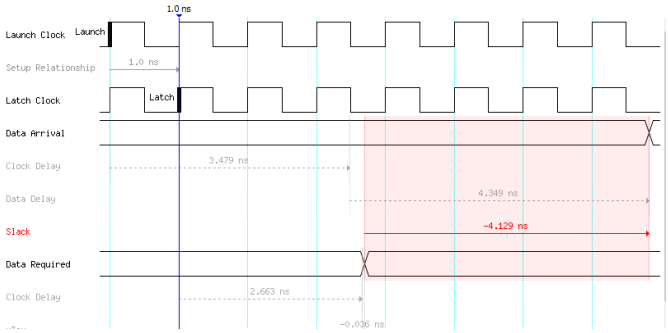
Experiment 3:

a)

Test Code 1:

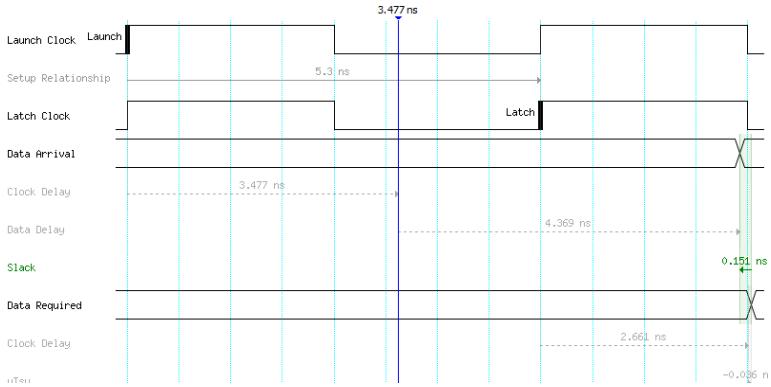
Before timing constraint:

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-4.129	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[1]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[2]	clk	clk	1.000	-0.816	4.349
2	-4.129	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[1]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[4]	clk	clk	1.000	-0.816	4.349
3	-4.092	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[2]	clk	clk	1.000	-0.816	4.312
4	-4.092	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[4]	clk	clk	1.000	-0.816	4.312
5	-4.039	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[2]	clk	clk	1.000	-0.816	4.259
6	-4.039	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[4]	clk	clk	1.000	-0.816	4.259
7	-4.029	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[5]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[2]	clk	clk	1.000	-0.818	4.247
8	-4.029	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[5]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[4]	clk	clk	1.000	-0.818	4.247
9	-3.979	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[1]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[6]	clk	clk	1.000	-0.818	4.197
10	-3.974	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[1]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[0]	clk	clk	1.000	-0.818	4.192



After timing constraint set to **5.2 ns**:

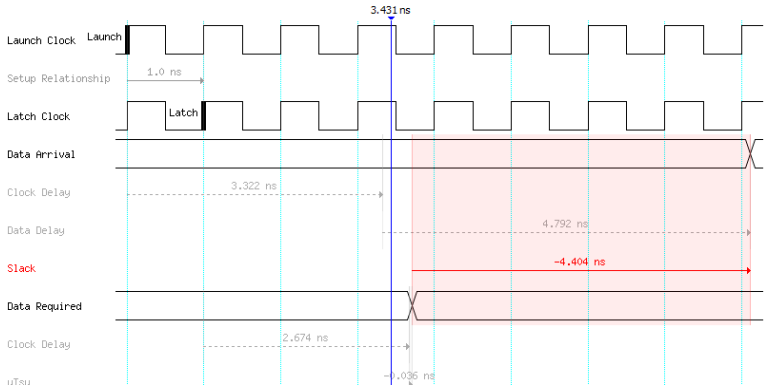
	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	0.151	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[2]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[3]	clk	clk	5.300	-0.816	4.369
2	0.156	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[2]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[5]	clk	clk	5.300	-0.816	4.364
3	0.161	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[2]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[7]	clk	clk	5.300	-0.816	4.359
4	0.202	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[2]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[2]	clk	clk	5.300	-0.814	4.320
5	0.202	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[2]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[4]	clk	clk	5.300	-0.814	4.320
6	0.228	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[4]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[2]	clk	clk	5.300	-0.817	4.291
7	0.228	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[4]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[4]	clk	clk	5.300	-0.817	4.291
8	0.259	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[2]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[4]	clk	clk	5.300	-0.814	4.263
9	0.282	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[4]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[6]	clk	clk	5.300	-0.819	4.235
10	0.289	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[4]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[1]	clk	clk	5.300	-0.819	4.228



Test Code 2:

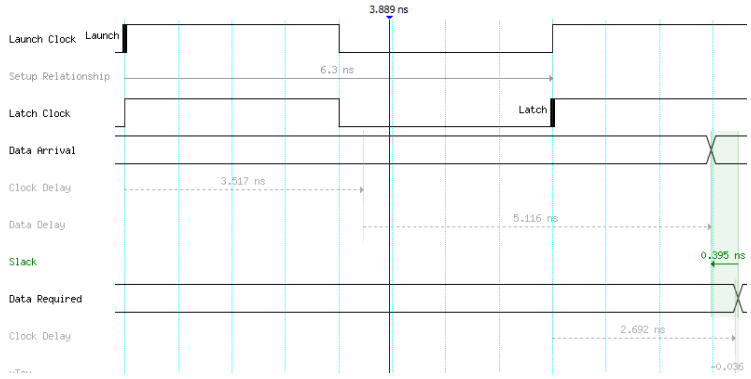
Before timing constraint:

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-4.404	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[1]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[7]	clk	clk	1.000	-0.648	4.792
2	-4.401	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[1]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[2]	clk	clk	1.000	-0.648	4.789
3	-4.324	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[7]	clk	clk	1.000	-0.648	4.712
4	-4.322	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[1]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[4]	clk	clk	1.000	-0.649	4.709
5	-4.321	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[2]	clk	clk	1.000	-0.648	4.709
6	-4.308	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[1]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[5]	clk	clk	1.000	-0.648	4.696
7	-4.282	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[1]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[2]	clk	clk	1.000	-0.648	4.670
8	-4.278	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[1]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[7]	clk	clk	1.000	-0.648	4.666
9	-4.271	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[1]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[6]	clk	clk	1.000	-0.649	4.658
10	-4.270	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[1]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[7]	clk	clk	1.000	-0.648	4.658



After timing constraint set to **6.3 ns**:

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	0.395	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[7]	clk	clk	6.300	-0.825	5.116
2	0.406	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|dmem:dmem_dut|RAM~3	clk	clk	6.300	-0.823	5.107
3	0.412	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[2]	clk	clk	6.300	-0.825	5.099
4	0.483	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[7]	clk	clk	6.300	-0.825	5.028
5	0.485	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|dmem:dmem_dut|RAM~3	clk	clk	6.300	-0.823	5.028
6	0.500	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[1]	clk	clk	6.300	-0.825	5.011
7	0.500	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[2]	clk	clk	6.300	-0.825	5.011
8	0.515	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[4]	top:comb_3|mips:mips_dut|datapath:dp|accumulator:accumulator|accout[0]	clk	clk	6.300	-0.827	4.994
9	0.544	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|dmem:dmem_dut|RAM~86	clk	clk	6.300	-0.823	4.969
10	0.544	top:comb_3|mips:mips_dut|datapath:dp|flopr:pcreg|q[3]	top:comb_3|dmem:dmem_dut|RAM~80	clk	clk	6.300	-0.823	4.969



b)

Tc = tpcq_PC + 2tmem + tRFread + tmux + tRFsetup = 30 + 2*250 + 150 + 25 + 200 + 20 = 925 ps

Time to complete Test Code 1 = 8 * 1 cycle * 925 ps = **7.4 ns**
Time to complete Test Code 2 = 9 * 1 cycle * 925 ps = **8.325 ns**

4. Conclusions & Summary

Upon implementing the bits of the datapath, I had errors for the pcmux where the counter did not properly count through the next address so the jump/branch instructions were not working. The fix to this was to implement the least significant byte into the pcmux with the jump in the pcmux module so the pc value would not get lost and can jump based on the immediate address (instr[3:0]) to that address. Another error that took up a lot of time was not noticing that aluop took up two bits so my controls register was off by one bit and everything was not working properly in the dut. Also, it was not possible for me to

load a value from dmem (data memory) into the accumulator register and my fix was to establish a mux for the acc_result register to equal the readdata output from dmem or to continue its previous value making it unaffected in the datapath and this readdata value is only implemented into the accumulator register when instr[7:4] = 4'b0001 or from the control word table, LDAC. All in all, each module performs a certain function and can only maintain 8 bits or one byte and must be consistent throughout the entirety of the modules. Afterwards, the accumulator register is essentially another regfile but only obtains one register and since R and AC are the only two registers used in this SMP microprocessor, we only know about storing in one reg. The output in each register remains until cleared or changed otherwise. The main decoder module tells the datapath and control module what to do for each instruction and the alu performs the arithmetic/logic functions for the inputs implemented from memory or the registers themselves and is consistent to perform each machine code successfully in each single clock cycle.

5. Zip files submitted