**COMPEN 311: Final Project**

**Topic Final Project**

**Class:** CompEn 331 section 2(22081—UP---P-CMPEN---331------002)

**Name** : Ayush Tiwari

**Abstract**

In this project I started with taking the code from lab 5 and using the zyboboard device for the lab. Then the first few additions include the two multiplexers forwardA and forward, both had cases where we either passed through values of qa (or qb), alu\_out, malu\_out, or data\_memory\_out depending on values of fwda (and fwdb) received from control unit. Then I added new inputs to the control unit- rs, rt, mrn, mm2reg, mwreg, ern, em2reg, and ewreg. And for the two new outputs of control unit, namely fwda and fwdb, I created if-cases which decided their values depending on whether rs || rt==em1\_out, AND rs || rt==mm1\_out. If rs ==em1\_out , then I set fwda=01, and if rs ==mm1\_out , then fwda=10, else fwda=00. And similar for rt and fwdb in place of rs and fwda respectively. Lastly, I just added the new initialization for the reg\_file and new instructions into the instruction memory.

**Module code**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 12/17/2020 03:39:41 AM

// Design Name:

// Module Name: PC

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module PC(

input wire clk,

input wire [7:0] pc\_in,

output reg [7:0] pc\_out

);

// initial begin

// pc\_out <= 100;

// end

// assign pc\_out <= 8'b100;

initial begin

pc\_out <= 8'd100;

end

always @(posedge clk) begin

pc\_out <= pc\_in;

end

endmodule

module adder(

input wire [7:0] pc,

output reg [7:0] next\_pc

);

initial begin

next\_pc <= 8'd100;

end

always @(pc) begin

next\_pc <= pc+ 8'd4;

end

endmodule

module IM(

input wire [7:0] pc,

output reg [31:0] instr

);

reg [31:0] im [0:511];

// im[100] = 32'b000000 00001 00010 00011 00000 100000

// im[104] = 32'b000000 01001 00011 00100 00000 100010

// im[108] = 32'b000000 00011 01001 00101 00000 100101

// im[112] = 32'b000000 00011 01001 00110 00000 100110

// im[116] = 32'b000000 00011 01001 00111 00000 100100

// im[100] = 32'b00000000001000100001100000100000

// im[104] = 32'b00000001001000110010000000100010

// im[108] = 32'b00000000011010010010100000100101

// im[112] = 32'b00000000011010010011000000100110

// im[116] = 32'b00000000011010010011100000100100

initial begin

im[8'd100] = 32'h221820; // Add

im[8'd104] = 32'h1232022; // Sub

im[8'd108] = 32'h692825; // Or

im[8'd112] = 32'h693026; // XOR

im[8'd116] = 32'h693824; // AND

instr <= 32'h0;

end

always@(pc) begin

instr <= im[pc];

end

endmodule

module IF\_ID(

input wire clk,

input wire [31:0] val\_in,

output reg [31:0] val\_out

);

always @(posedge clk) begin

val\_out <= val\_in;

end

endmodule

module control\_unit(

input wire [5:0] op,

input wire [5:0] func,

input wire [4:0] rs,

input wire [4:0] rt,

input wire [4:0] mm1\_out,

input wire mm2reg,

input wire mwreg,

input wire [4:0] em1\_out,

input wire em2reg,

input wire ewreg,

output reg wreg,

output reg m2reg,

output reg wmem,

output reg [3:0] aluc,

output reg aluimm,

output reg [1:0] fwd\_b,

output reg [1:0] fwd\_a,

output reg regrt

);

initial begin

fwd\_a <= 2'b00;

fwd\_b <= 2'b00;

end

always@(op, func) begin

if(op==6'b100011) begin // we will know that its lw

wreg <= 1;

m2reg <= 1;

wmem <= 0;

aluc <= 4'b0010;

aluimm <= 1;

regrt <= 1;

end

else if(op==6'b000000)begin // this will be an r-type fiunc

// deciding which r-type function it is

//

if(func==6'b100000)begin // add

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0010;

aluimm <= 0;

regrt <= 0;

end

else if(func==6'b100010)begin // sub

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0110;

aluimm <= 0;

regrt <= 0;

end

else if(func==6'b100100)begin // aNd

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0000;

aluimm <= 0;

regrt <= 0;

end

else if(func==6'b100101)begin // or

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0001;

aluimm <= 0;

regrt <= 0;

end

else if(func==6'b100110)begin // Xor

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b1001; // if chagnge this, the change ALU as welll // MAKE SURE ABOUT THIS BEFORE SUBMitting

aluimm <= 0;

regrt <= 0;

end

end

// input (a) for ALU

if(rs == em1\_out)begin

fwd\_a <=2'b01; // send alu\_out to output of fwd\_a

end

else if(rs == mm1\_out) begin

fwd\_a <=2'b10; // send Malu\_out to output of fwd\_a

end

else

fwd\_a <=2'b00;

//input (b) for ALU

if(rt == em1\_out)begin

fwd\_b <= 2'b01; // send alu\_out to output of fwd\_b

end

else if(rt == mm1\_out)begin

fwd\_b <= 2'b10; // send Malu\_out to output of fwd\_b

end

else

fwd\_b <= 2'b00;

// if(rt == em1\_out )begin

// end

end

endmodule

module mux\_1(

input wire regrt,

input wire [4:0] rd,

input wire [4:0] rt,

output reg [4:0] m1\_out

);

always @(\*) begin

if(regrt== 1'b1) begin

m1\_out <= rt;

end

else

m1\_out <= rd;

end

endmodule

module reg\_file(

input wire clk,

input wire wwreg,

input wire [4:0] rs,

input wire [4:0] rt,

input wire[4:0] wm1\_out,

input wire [31:0] m3\_out,

output reg [31:0] qa,

output reg [31:0] qb

);

reg [31:0] reg\_f [0:31];

integer i;

initial begin

// for(i=0;i<=31;i=i+1) begin

// reg\_f[i] <= 32'b0;

// end

reg\_f[0] = 32'h00000000;

reg\_f[1] = 32'ha00000aa;

reg\_f[2] = 32'h10000011;

reg\_f[3] = 32'h20000022;

reg\_f[4] = 32'h30000033;

reg\_f[5] = 32'h40000044;

reg\_f[6] = 32'h50000055;

reg\_f[7] = 32'h60000066;

reg\_f[8] = 32'h70000077;

reg\_f[9] = 32'h80000088;

reg\_f[10] = 32'h90000099;

end

// always@(rs, rt) begin

always@(posedge clk) begin

// for(i=0;i<=31;i=i+1) begin

// reg\_f[i] <= 32'b0;

// end

//READ

//

qa <= reg\_f[rs];

qb <= reg\_f[rt];

end

always@(negedge clk) begin

// WRITE

//

if(wwreg==1) begin

reg\_f[wm1\_out]<= m3\_out;

end

end

endmodule

module mux\_forw\_a(

input wire [1:0] fwd\_a,

input wire [31:0] qa,

input wire [31:0] alu\_out,

input wire [31:0] malu\_out,

input wire [31:0] data\_mem\_out,

output reg [31:0] mfwa\_out

);

always @(\*)begin

if(fwd\_a== 2'b00) begin

mfwa\_out<= qa;

end

else if(fwd\_a== 2'b01) begin

mfwa\_out<= alu\_out;

end

else if(fwd\_a== 2'b10) begin

mfwa\_out<= malu\_out;

end

else if(fwd\_a== 2'b11) begin

mfwa\_out<= data\_mem\_out;

end

end

endmodule

module mux\_forw\_b(

input wire [1:0] fwd\_b,

input wire [31:0] qb,

input wire [31:0] alu\_out,

input wire [31:0] malu\_out,

input wire [31:0] data\_mem\_out,

output reg [31:0] mfwb\_out

);

always @(\*)begin

if(fwd\_b== 2'b00) begin

mfwb\_out<= qb;

end

else if(fwd\_b== 2'b01) begin

mfwb\_out<= alu\_out;

end

else if(fwd\_b== 2'b10) begin

mfwb\_out<= malu\_out;

end

else if(fwd\_b== 2'b11) begin

mfwb\_out<= data\_mem\_out;

end

end

endmodule

module sign\_extender(

input wire [15:0] instr\_const,

output reg [31:0] se\_out

);

always @(instr\_const) begin

se\_out <= {{16{instr\_const[15]}} , instr\_const};

end

endmodule

module ID\_EXE(

input wire clk,

input wire wreg,

input wire m2reg,

input wire wmem,

input wire [3:0]aluc,

input wire aluimm,

input wire [4:0] m1\_out,

input wire [31:0] qa,

input wire [31:0]qb,

input wire [31:0]se\_out,

output reg ewreg,

output reg em2reg,

output reg ewmem,

output reg [3:0] ealuc,

output reg ealuimm,

output reg [4:0] em1\_out,

output reg [31:0]eqa,

output reg [31:0]eqb,

output reg [31:0]ese\_out

);

always@(posedge clk) begin

ewreg <= wreg;

em2reg <= m2reg;

ewmem <= wmem;

ealuc <= aluc;

ealuimm <= aluimm;

em1\_out <= m1\_out;

eqa <= qa;

eqb <= qb;

ese\_out <= se\_out;

end

endmodule

module mux\_2(

input wire ealuimm,

input wire [31:0] eqb,

input wire [31:0] ese\_out,

output reg [31:0] m2\_out

);

always@(\*)begin

if(ealuimm==0)begin

m2\_out<= eqb;

end

else

m2\_out <= ese\_out;

end

endmodule

module ALU(

input wire [3 :0] ealuc,

input wire [31:0] eqa,

input wire [31:0] m2\_out,

output reg [31:0] alu\_out

);

always@(\*)begin

if(ealuc==4'b0000) //AND

alu\_out<= eqa & m2\_out;

else if(ealuc==4'b0001) //OR

alu\_out<= eqa | m2\_out;

else if(ealuc==4'b0010) // add

alu\_out<= eqa + m2\_out;

else if(ealuc==4'b0110) // sub

alu\_out<= eqa - m2\_out;

else if(ealuc==4'b1001) // xor

alu\_out<= eqa ^ m2\_out;

end

endmodule

module EXE\_MEM(

input wire clk,

input wire ewreg,

input wire em2reg,

input wire ewmem,

input wire [4:0] em1\_out,

input wire [31:0] alu\_out,

input wire [31:0] eqb,

output reg mwreg,

output reg mm2reg,

output reg mwmem,

output reg [4:0] mm1\_out,

output reg [31:0] malu\_out,

output reg [31:0] mqb

);

always@(posedge clk)begin

mwreg <= ewreg;

mm2reg <= em2reg;

mwmem <= ewmem ;

mm1\_out <= em1\_out;

malu\_out <= alu\_out ;

mqb <= eqb;

end

endmodule

module data\_mem(

input wire mwmem,

input wire [31:0]malu\_out,

input wire [31:0]mqb,

output reg [31:0]do

);

reg [31:0] data\_m [0:511];

initial begin

data\_m[0] = 32'ha00000aa;

data\_m[4] = 32'h10000011;

data\_m[8] = 32'h20000022;

data\_m[12] = 32'h30000033;

data\_m[16] = 32'h40000044;

data\_m[20] = 32'h50000055;

data\_m[24] = 32'h60000066;

data\_m[28] = 32'h70000077;

data\_m[32] = 32'h80000088;

data\_m[36] = 32'h90000099;

// data\_m[40] = 32'h8c030004;

end// use

always@(\*) begin

if(mwmem==0)begin // means we are reading from the data memory

do <= data\_m[malu\_out];

end

else if(mwmem==1) begin

// data\_m[]<=;

// do <= data\_m[mqb];

data\_m[malu\_out]=mqb;

end

end

endmodule

module MEM\_WB(

input wire clk,

input wire mwreg,

input wire mm2reg,

input wire[4:0] mm1\_out,

input wire [31:0] malu\_out,

input wire [31:0] data\_mem\_out,

output reg wwreg,

output reg wm2reg,

output reg [4:0] wm1\_out,

output reg [31:0] walu\_out,

output reg [31:0] w\_data\_mem\_out

);

always@(posedge clk) begin

wwreg <= mwreg;

wm2reg <=mm2reg;

wm1\_out <= mm1\_out ;

walu\_out <= malu\_out;

w\_data\_mem\_out <= data\_mem\_out;

end

endmodule

module mux\_3(

wire wm2reg,

wire [31:0] walu\_out,

wire [31:0] w\_data\_mem\_out,

output reg [31:0] m3\_out

);

always@(wm2reg or walu\_out or w\_data\_mem\_out) begin

if(wm2reg==0)begin

m3\_out<= walu\_out;

end

else begin

m3\_out <= w\_data\_mem\_out;

end

end

endmodule

**Testbench**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 12/17/2020 03:42:05 AM

// Design Name:

// Module Name: testbench

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module testbench();

reg clk;

wire [7:0] pc;

wire [7:0] next\_pc;

wire [31:0] im\_out;

wire [31:0] instr;

wire wreg;

wire m2reg;

wire wmem;

wire [3:0] aluc;

wire aluimm;

wire [1:0] fwd\_a;

wire [1:0] fwd\_b;

wire regrt;

wire [31:0] mfwa\_out;

wire [31:0] mfwb\_out;

wire [4:0] m1\_out;

wire [31:0] qa;

wire [31:0] qb;

wire [31:0] se\_out;

wire ewreg;

wire em2reg;

wire ewmem;

wire [3:0] ealuc;

wire ealuimm;

wire [4:0] em1\_out;

wire [31:0]eqa;

wire [31:0]eqb;

wire [31:0]ese\_out;

wire [31:0] m2\_out;

wire [31:0] alu\_out;

wire mwreg;

wire mm2reg;

wire mwmem;

wire [4:0] mm1\_out;

wire [31:0] malu\_out;

wire [31:0] mqb;

wire [31:0] data\_mem\_out;

wire wwreg;

wire wm2reg;

wire [4:0] wm1\_out;

wire [31:0] walu\_out;

wire [31:0] w\_data\_mem\_out;

wire [31:0] m3\_out;

//for IF stage

PC pc\_tb(clk, next\_pc, pc);

adder adder\_tb( pc, next\_pc);

IM im\_tb(pc, im\_out);

IF\_ID ifid\_tb(clk, im\_out, instr);

//for ID stage

control\_unit cu\_tb( instr[31:26],instr[5:0], instr[25:21],instr[20:16], mm1\_out, mm2reg, mwreg, em1\_out , em2reg, ewreg , wreg, m2reg, wmem, aluc, aluimm, fwd\_b,fwd\_a,regrt);

mux\_1 mux1\_tb(regrt,instr[15:11], instr[20:16], m1\_out);

reg\_file regfile\_tb(~clk,wwreg, instr[25:21], instr[20:16],wm1\_out,m3\_out, qa, qb);

mux\_forw\_a muxfa\_tb(fwd\_a,qa, alu\_out, malu\_out, data\_mem\_out, mfwa\_out );

mux\_forw\_b muxfb\_tb(fwd\_b,qb, alu\_out, malu\_out, data\_mem\_out, mfwb\_out );

sign\_extender se\_tb(instr[15:0], se\_out); //takes in the 16 bit constant and makes it 32 bits

ID\_EXE idexe\_tb(clk, wreg, m2reg, wmem, aluc, aluimm, m1\_out, mfwa\_out, mfwb\_out , se\_out , ewreg ,em2reg, ewmem, ealuc, ealuimm,em1\_out, eqa, eqb, ese\_out);

// for EXE stage

mux\_2 mux2\_tb(ealuimm,eqb,ese\_out, m2\_out);

ALU alu\_tb(ealuc, /\*eqa\*/ eqa, m2\_out, alu\_out);

EXE\_MEM exemem\_tb(clk, ewreg, em2reg, ewmem, em1\_out,alu\_out, eqb , mwreg, mm2reg, mwmem, mm1\_out,malu\_out, mqb);

// for MEM stage

data\_mem dm\_tb(mwmem, malu\_out, mqb, data\_mem\_out);

MEM\_WB memwb\_tb(clk,mwreg, mm2reg, mm1\_out, malu\_out, data\_mem\_out, wwreg, wm2reg, wm1\_out, walu\_out, w\_data\_mem\_out );

mux\_3 mux3\_tb(wm2reg,walu\_out,w\_data\_mem\_out,m3\_out);

always #5 clk = ~clk;

initial begin

clk = 0;

// pc= 8'b100;

// next\_pc= 8'b100;

// ewreg <= 0 ;

// em2reg <= 0;

// ewmem <= 0;

// ealuc <= 4'b0;

// ealuimm<=0;

// em1\_out<= 5'b0;

// eqa <=32'b0;

// eqb<= 32'b0;

// ese\_out<= 32'b0;

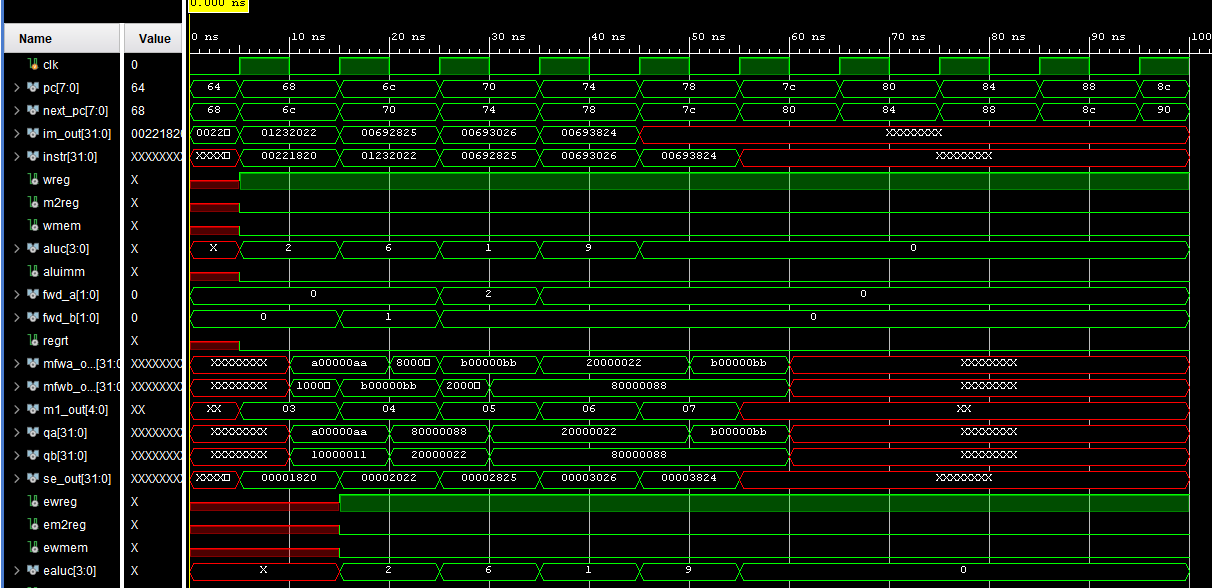
#100;

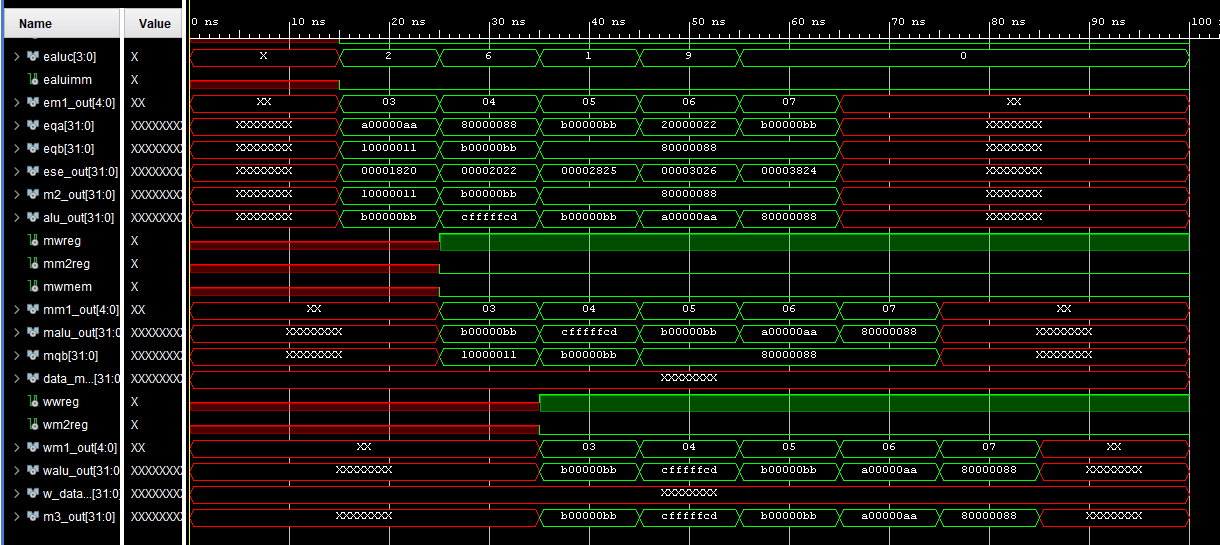
$stop;

end

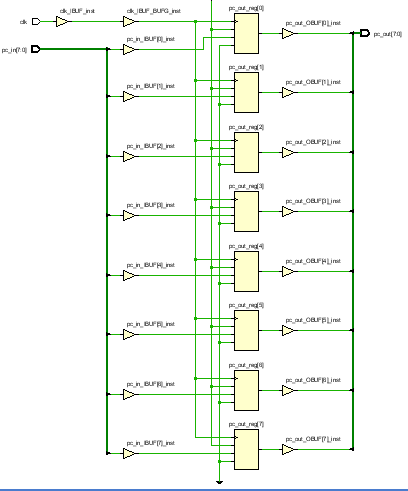
endmodule

**Waveform**

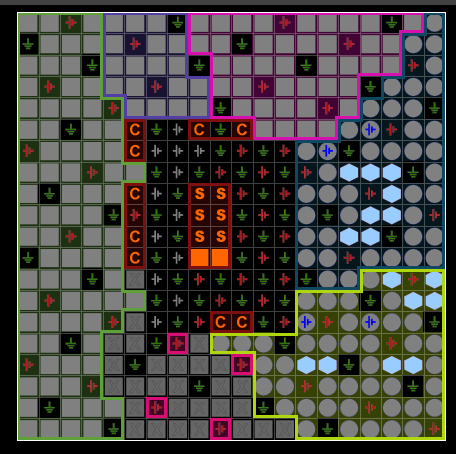




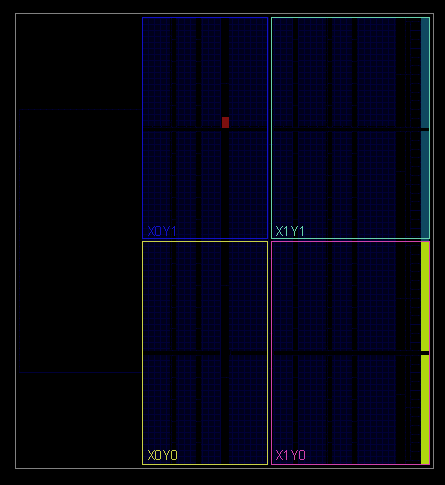
**Schematic**



**IO Planning**



**Floorplanning**



**Bit stream generated successfully**

