# State Diagram

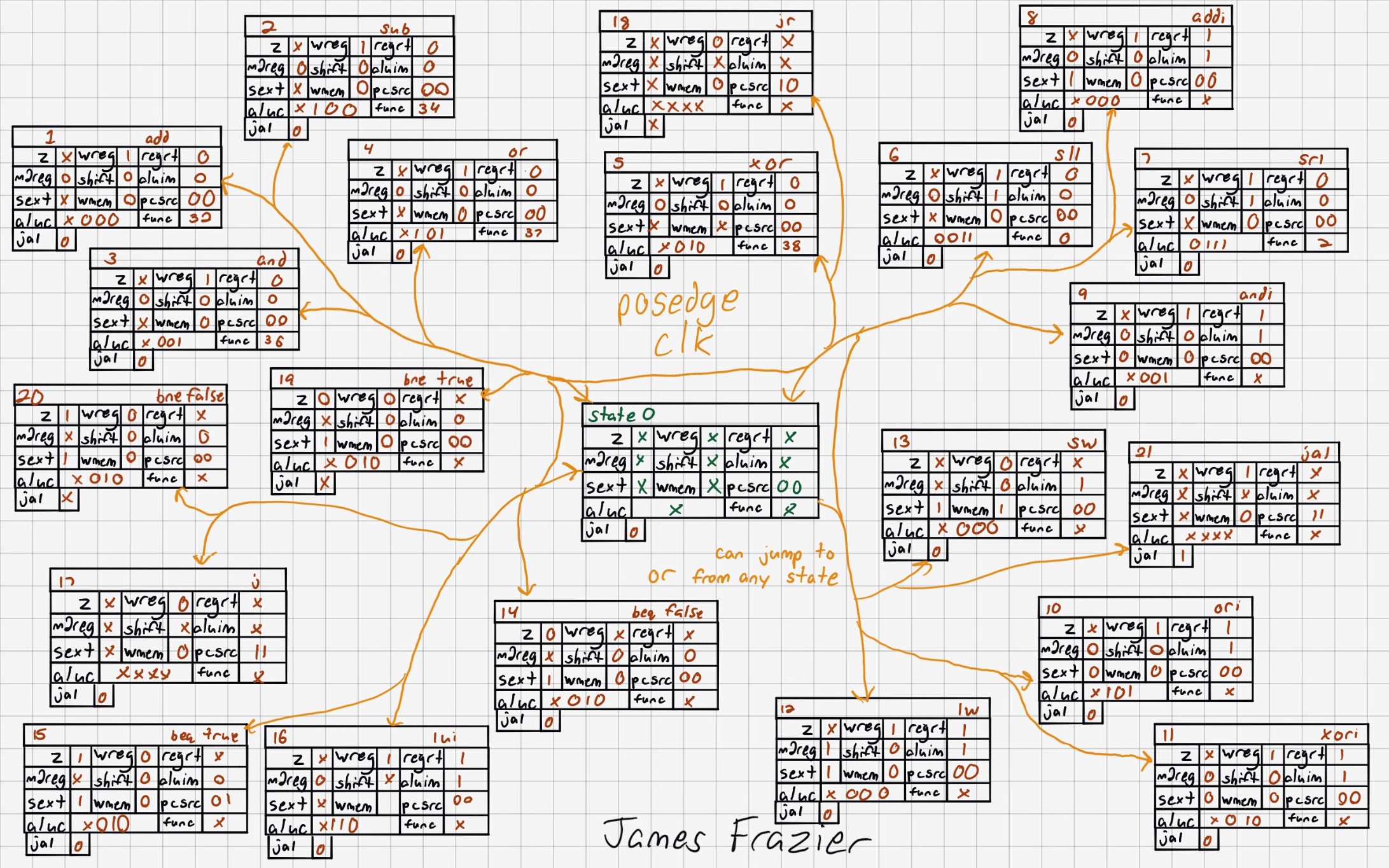
LAB 7

jwf5426@psu.edu | JWF5426

Due: Friday, December 9th @ 7:00 AM Handed In: Saturday, December 10th @ 5:35 AM

James Frazier

2016



# Verilog Code

## Test Bench

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: CPU

//

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

module CPU(

input clk,

input rst,

output reg bogus

);

wire [31:0] pc;

wire [5:0] op;

wire [4:0] rs\_addressMSB;

wire [4:0] rt\_addressmiddleONE;

wire [4:0] rd\_addressmiddleTWO\_immMSB;

wire [4:0] sa\_addressmiddleTHREE\_immMIDDLE;

wire [5:0] func\_addressLSB\_immLSB;

wire [4:0] wn\_in;

wire z;

wire [3:0] aluc;

wire [1:0] prsrc;

wire shift;

wire wreg;

wire sext;

wire wmem;

wire m2reg;

wire aluim;

wire regrt;

wire branch;

wire jal;

wire [31:0] qa;

wire [31:0] A;

wire [31:0] qb;

wire [31:0] B;

wire [31:0] aluout;

wire [31:0] memout;

wire [31:0] pcMuxOut;

wire [31:0] dataMuxOut;

wire [31:0] pcPlusFour;

wire [31:0] branchAddr;

wire [31:0] imm\_extend;

wire [31:0] imm\_extend\_shifted;

wire [31:0] addr\_shifted;

dataMux myDataMux({jal, m2reg}, aluout, memout, pcPlusFour, dataMuxOut);

data\_memory myData\_Memory(wmem, rst, aluout, qb, memout);

shift\_two\_left mySecond\_shift\_two\_left({6'b000000, rs\_addressMSB, rt\_addressmiddleONE, rd\_addressmiddleTWO\_immMSB, sa\_addressmiddleTHREE\_immMIDDLE, func\_addressLSB\_immLSB}, addr\_shifted);

adder mySecondAdder(pcPlusFour, imm\_extend\_shifted, branchAddr);

shift\_two\_left myFirst\_shift\_two\_left(imm\_extend, imm\_extend\_shifted);

ALU\_A\_Mux myALU\_A\_Mux(shift, qa, sa\_addressmiddleTHREE\_immMIDDLE, A);

ALU\_B\_Mux myALU\_B\_Mux(aluim, qb, imm\_extend, B);

sign\_extend mySign\_Extend(sext, {rd\_addressmiddleTWO\_immMSB, sa\_addressmiddleTHREE\_immMIDDLE, func\_addressLSB\_immLSB}, imm\_extend);

pcMux myPcMux(prsrc, pcPlusFour, branchAddr, qa, {pcPlusFour[31:28], addr\_shifted[27:0]}, pcMuxOut);

WN\_Input\_Mux myWN\_Input\_Mux(rd\_addressmiddleTWO\_immMSB, rt\_addressmiddleONE, {jal, regrt}, wn\_in);

adder myFirstAdder(32'd4, pc, pcPlusFour);

program\_counter myPC(/\*1, \*/pcMuxOut, /\*0, 16'd0,\*/ clk, rst, pc);

instruction\_memory myIM(pc, op, rs\_addressMSB, rt\_addressmiddleONE, rd\_addressmiddleTWO\_immMSB, sa\_addressmiddleTHREE\_immMIDDLE, func\_addressLSB\_immLSB);

ALUControl myALUControl(op, func\_addressLSB\_immLSB, z, aluc, prsrc, shift, wreg, sext, wmem, m2reg, aluim, regrt, branch, jal);

regfile myRegFile(rs\_addressMSB, rt\_addressmiddleONE, dataMuxOut, wn\_in, wreg, clk, rst, qa, qb);

MIPSALU myMIPSALU(aluc, branch, A, B, aluout, z);

Endmodule

## Design Code

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: dataMux

//

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

module dataMux(

input [1:0] switch,

input [31:0] ALU\_Result,

input [31:0] MEM\_Result,

input [31:0] PCPlusFour\_Result,

output reg [31:0] dataMux\_out

);

always @(\*)

case(switch)

2'b00: dataMux\_out = ALU\_Result;

2'b01: dataMux\_out = MEM\_Result;

2'b1x: dataMux\_out = PCPlusFour\_Result;

endcase

endmodule

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: data\_memory

//

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

module data\_memory(

input wmem,

input clrn,

input [31:0] address,

input [31:0] write\_data,

output [31:0] read\_data

);

reg [7:0] data [0:100]; // 100 8-bit data

assign read\_data = {data[address], data[address + 1], data[address + 2], data[address + 3]};

integer i;

always @(address or clrn) begin

if (clrn)

for(i = 0; i <= 70; i = i + 1)

data[i] <= 0;

else

if (wmem) begin

data[address] <= write\_data[31:24];

data[address + 1] <= write\_data[23:16];

data[address + 2] <= write\_data[15:8];

data[address + 3] <= write\_data[7:0];

$display("Wrote %h at address %d", write\_data, address);

end

end

endmodule

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: shift\_two\_left

//

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

module shift\_two\_left(

input [31:0] in,

output [31:0] out\_shifted

);

assign out\_shifted = in << 2;

endmodule

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: adder

//

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

module adder (

input [31:0] A,

input [31:0] B,

output [31:0] sum);

assign sum = A + B;

endmodule

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: shift\_two\_left

//

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

module shift\_two\_left(

input [31:0] in,

output [31:0] out\_shifted

);

assign out\_shifted = in << 2;

endmodule

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: shiftMux

//

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

module ALU\_A\_Mux(

input shift,

input [31:0] qa,

input [4:0] sa,

output reg [31:0] A

);

always @(\*)

if(shift == 1'b1)

A = {27'd0, sa};

else

A = qa;

Endmodule

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: ALU\_B\_Mux

//

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

module ALU\_B\_Mux(

input switch,

input [31:0] qb,

input [31:0] imm\_exten,

output reg [31:0] B

);

always @(\*)

if(switch == 1'b1)

B = imm\_exten;

else

B = qb;

Endmodule

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: sign\_extend

//

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

module sign\_extend(

input switch,

input [15:0] imm,

output reg [31:0] imm\_output

);

always @(\*)

if(switch == 1'b1 && imm[15] == 1'b1)

imm\_output = {16'b1111111111111111, imm};

else

imm\_output = {16'b0000000000000000, imm};

endmodule

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: pcMux

//

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

module pcMux(

input [1:0] switch,

input [31:0] PcPlusFour,

input [31:0] BranchAddr,

input [31:0] RegAddr,

input [31:0] JumpAddr,

output reg [31:0] pcMuxOut

);

always @(\*)

if(switch == 2'd1)

pcMuxOut = BranchAddr;

else if(switch == 2'd2)

pcMuxOut = RegAddr;

else if(switch == 2'd3)

pcMuxOut = JumpAddr;

else

pcMuxOut = PcPlusFour;

endmodule

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: WN\_Input\_Mux

//

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

module WN\_Input\_Mux(

input [4:0] rd,

input [4:0] rt,

input [1:0] switch,

output reg [4:0] wn

);

always @(\*)

case(switch)

2'b1x: wn = 5'd31;

2'b00: wn = rd;

2'b01: wn = rt;

endcase

endmodule

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: adder

//

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

module adder (

input [31:0] A,

input [31:0] B,

output [31:0] sum);

assign sum = A + B;

endmodule

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: program\_counter

//

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

module program\_counter(

input [31:0] update,

// input [31:0] update\_by,

// input branch,

// input [15:0] branch\_offset,

input clk,

input rst,

output reg [31:0] pc

);

// parameter INCREMENT\_AMOUNT = 32'd4;

initial begin

pc = 32'd0;

end

always @(posedge clk or posedge rst)

begin

if (rst)

pc <= 0;

else /\*if (update)

if (branch)

pc <= pc + {16'd0,branch\_offset};

else\*/

pc <= update/\*\_byINCREMENT\_AMOUNT\*/;

end

endmodule

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: instruction\_memory

//

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

module instruction\_memory(

input [31:0] read\_address,

output [5:0] op,

output [4:0] rs\_addressMSB,

output [4:0] rt\_addressmiddleONE,

output [4:0] rd\_addressmiddleTWO\_immMSB,

output [4:0] sa\_addressmiddleTHREE\_immMIDDLE,

output [5:0] func\_addressLSB\_immLSB

);

reg [31:0] instruction [0:2]; // 3 instructions

reg [31:0] current\_instruction;

initial begin

instruction[0] = 32'h0c000002;

instruction[1] = 32'h14a00000;

instruction[2] = 32'h03e00008;

end

assign op = instruction[read\_address / 4][31:26];

assign rs\_addressMSB = instruction[read\_address / 4][25:21];

assign rt\_addressmiddleONE = instruction[read\_address / 4][20:16];

assign rd\_addressmiddleTWO\_immMSB = instruction[read\_address / 4][15:11];

assign sa\_addressmiddleTHREE\_immMIDDLE = instruction[read\_address / 4][10:6];

assign func\_addressLSB\_immLSB = instruction[read\_address / 4][5:0];

always @(\*)

current\_instruction = instruction[read\_address / 4];

endmodule

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: ALUControl

//

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

module ALUControl (

input [5:0] ALUOp,

input [5:0] FuncCode,

input z,

output reg [3:0] ALUCtl,

output reg [1:0] prsrc,

output reg shift,

output reg wreg,

output reg sext,

output reg wmem,

output reg m2reg,

output reg aluimm,

output reg regrt,

output reg branch,

output reg jal

);

always @(\*) begin

if(ALUOp == 6'b000000) begin

case (FuncCode)

32: begin

ALUCtl <= 4'bx000; // add

prsrc <= 2'b00;

shift <= 1'b0;

wreg <= 1'b1;

sext <= 1'bx;

wmem <= 1'b0;

m2reg <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

branch <= 1'b0;

jal <= 1'b0;

end

34: begin

ALUCtl <= 4'bx100;; // sub

prsrc <= 2'b00;

shift <= 1'b0;

wreg <= 1'b1;

sext <= 1'bx;

wmem <= 1'b0;

m2reg <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

branch <= 1'b0;

jal <= 1'b0;

end

36: begin // and

ALUCtl <= 4'bx001;

prsrc <= 2'b00;

shift <= 1'b0;

wreg <= 1'b1;

sext <= 1'bx;

wmem <= 1'b0;

m2reg <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

branch <= 1'b0;

jal <= 1'b0;

end

37: begin // or

ALUCtl <= 4'bx101;

prsrc <= 2'b00;

shift <= 1'b0;

wreg <= 1'b1;

sext <= 1'bx;

wmem <= 1'b0;

m2reg <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

branch <= 1'b0;

jal <= 1'b0;

end

38: begin

ALUCtl <= 4'bx010; // xor

prsrc <= 2'b00;

shift <= 1'b0;

wreg <= 1'b1;

sext <= 1'bx;

wmem <= 1'b0;

m2reg <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

branch <= 1'b0;

jal <= 1'b0;

end

0: begin

ALUCtl <= 4'b0011; // sll

prsrc <= 2'b00;

shift <= 1'b1;

wreg <= 1'b1;

sext <= 1'bx;

wmem <= 1'b0;

m2reg <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

branch <= 1'b0;

jal <= 1'b0;

end

2: begin

ALUCtl <= 4'b0111; // srl

prsrc <= 2'b00;

shift <= 1'b1;

wreg <= 1'b1;

sext <= 1'bx;

wmem <= 1'b0;

m2reg <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

branch <= 1'b0;

jal <= 1'b0;

end

/\*3: begin

ALUCtl <= 4'b1111; // sra

prsrc <= 2'b00;

shift <= 1'b1;

wreg <= 1'b1;

sext <= 1'bx;

wmem <= 1'b0;

m2reg <= 1'b0;

aluimm <= 1'b0;

regrt <= 1'b0;

end\*/

8: begin

ALUCtl <= 4'bxxxx; // jr

prsrc <= 2'b10;

shift <= 1'bx;

wreg <= 1'b0;

sext <= 1'bx;

wmem <= 1'b0;

m2reg <= 1'bx;

aluimm <= 1'bx;

regrt <= 1'bx;

branch <= 1'b0;

jal <= 1'bx;

end

default: begin

ALUCtl <= 4'dx; // should not happen.

prsrc <= 2'bxx;

shift <= 1'bx;

wreg <= 1'bx;

sext <= 1'bx;

wmem <= 1'bx;

m2reg <= 1'bx;

aluimm <= 1'bx;

regrt <= 1'bx;

branch <= 1'bx;

jal <= 1'b0;

end

endcase

end

else begin

case (ALUOp)

8: begin

ALUCtl <= 4'bx000; // addi

prsrc <= 2'b00;

shift <= 1'b0;

wreg <= 1'b1;

sext <= 1'b1;

wmem <= 1'b0;

m2reg <= 1'b0;

aluimm <= 1'b1;

regrt <= 1'b1;

branch <= 1'b0;

jal <= 1'b0;

end

12: begin

ALUCtl <= 4'bx001;; // andi

prsrc <= 2'b00;

shift <= 1'b0;

wreg <= 1'b1;

sext <= 1'b0;

wmem <= 1'b0;

m2reg <= 1'b0;

aluimm <= 1'b1;

regrt <= 1'b1;

branch <= 1'b0;

jal <= 1'b0;

end

13: begin // ori

ALUCtl <= 4'bx101;

prsrc <= 2'b00;

shift <= 1'b0;

wreg <= 1'b1;

sext <= 1'b0;

wmem <= 1'b0;

m2reg <= 1'b0;

aluimm <= 1'b1;

regrt <= 1'b1;

branch <= 1'b0;

jal <= 1'b0;

end

14: begin // xori

ALUCtl <= 4'bx010;

prsrc <= 2'b00;

shift <= 1'b0;

wreg <= 1'b1;

sext <= 1'b0;

wmem <= 1'b0;

m2reg <= 1'b0;

aluimm <= 1'b1;

regrt <= 1'b1;

branch <= 1'b0;

jal <= 1'b0;

end

35: begin

ALUCtl <= 4'bx000; // lw

prsrc <= 2'b00;

shift <= 1'b0;

wreg <= 1'b1;

sext <= 1'b1;

wmem <= 1'b0;

m2reg <= 1'b1;

aluimm <= 1'b1;

regrt <= 1'b1;

branch <= 1'b0;

jal <= 1'b0;

end

43: begin

ALUCtl <= 4'bx000; // sw

prsrc <= 2'b00;

shift <= 1'b0;

wreg <= 1'b0;

sext <= 1'b1;

wmem <= 1'b1;

m2reg <= 1'bx;

aluimm <= 1'b1;

regrt <= 1'bx;

branch <= 1'b0;

jal <= 1'b0;

end

4: begin

ALUCtl <= 4'bx010; // beq

shift <= 1'b0;

wreg <= 1'b0;

sext <= 1'b1;

wmem <= 1'b0;

m2reg <= 1'bx;

aluimm <= 1'b0;

regrt <= 1'bx;

branch <= 1'b1;

jal <= 1'b0;

end

15: begin

ALUCtl <= 4'bx110; // lui

prsrc <= 2'b00;

shift <= 1'bx;

wreg <= 1'b1;

sext <= 1'bx;

wmem <= 1'b0;

m2reg <= 1'b0;

aluimm <= 1'b1;

regrt <= 1'b1;

branch <= 1'b0;

jal <= 1'b0;

end

2: begin

ALUCtl <= 4'bxxxx; // j

prsrc <= 2'b11;

shift <= 1'bx;

wreg <= 1'b0;

sext <= 1'bx;

wmem <= 1'b0;

m2reg <= 1'bx;

aluimm <= 1'bx;

regrt <= 1'bx;

branch <= 1'b0;

jal <= 1'b0;

end

5: begin

ALUCtl <= 4'bx010; // bne

shift <= 1'b0;

wreg <= 1'b0;

sext <= 1'b1;

wmem <= 1'b0;

m2reg <= 1'bx;

aluimm <= 1'b0;

regrt <= 1'bx;

branch <= 1'b1;

jal <= 1'b0;

end

3: begin

ALUCtl <= 4'bxxxx; // jal

prsrc <= 2'b11;

shift <= 1'bx;

wreg <= 1'b1;

sext <= 1'bx;

wmem <= 1'b0;

m2reg <= 1'bx;

aluimm <= 1'bx;

regrt <= 1'bx;

branch <= 1'b0;

jal <= 1'b1;

end

default: begin

ALUCtl <= 4'bxxxx; // should not happen

prsrc <= 2'bxx;

shift <= 1'bx;

wreg <= 1'bx;

sext <= 1'bx;

wmem <= 1'bx;

m2reg <= 1'bx;

aluimm <= 1'bx;

regrt <= 1'bx;

branch <= 1'bx;

jal <= 1'bx;

end

endcase

end

end

always @(z)

if(ALUOp == 6'd4)

case (z)

1'b0: prsrc <= 2'b00;

1'b1: prsrc <= 2'b01;

endcase

else if(ALUOp == 6'd5)

case (z)

1'b0: prsrc <= 2'b01;

1'b1: prsrc <= 2'b00;

endcase

endmodule

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: regfile

//

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

module regfile (rna,rnb,d,wn,we,clk,clrn,qa,qb); // 32x32 regfile

input [31:0] d; // data of write port

input [4:0] rna; // reg # of read port A

input [4:0] rnb; // reg # of read port B

input [4:0] wn; // reg # of write port

input we; // write enable

input clk, clrn; // clock and reset

output [31:0] qa, qb; // read ports A and B

reg [31:0] register [1:32]; // 31 32-bit registers

assign qa = (rna == 0)? 0 : register[rna]; // read port A

assign qb = (rnb == 0)? 0 : register[rnb]; // read port B

initial begin

// register[0] = 32'h00000000;

register[1] = 32'hA00000AA;

register[2] = 32'h10000011;

register[3] = 32'h20000022;

register[4] = 32'h30000033;

register[5] = 32'h40000044;

register[6] = 32'h50000055;

register[7] = 32'h60000066;

register[8] = 32'h70000077;

register[9] = 32'h80000088;

register[10] = 32'h90000099;

end

integer i;

always @(posedge clk or posedge clrn) begin // write port

if (clrn) begin

for(i = 0; i <= 10; i = i + 1)

register[i] <= 0;

end

else

if ((wn != 0) && we) begin // not reg[0] & enabled

register[wn] <= d; // write d to reg[wn]

$display("register[%d] <= %h", wn, register[wn]);

end

end

endmodule

`timescale 1ns / 1ps

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

// Company: Penn State Univeristy

// Engineer: James W. Frazier

//

// Create Date: 11/09/2016 06:18:02 PM

// Design Name:

// Module Name: MIPSALU

//

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

module MIPSALU (ALUctl, branch, A, B, ALUOut, Zero);

input [3:0] ALUctl;

input branch;

input [31:0] A,B;

output reg [31:0] ALUOut;

output reg Zero;

// assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0

always @(ALUctl, A, B)

begin //reevaluate if these change

case (ALUctl)

4'bx000: begin // add addi lw sw

ALUOut <= A + B;

Zero <= 1'bx;

end

4'bx100: begin // sub

ALUOut <= A - B;

Zero <= 1'bx;

end

4'bx001: begin // and andi

ALUOut <= A & B;

Zero <= 1'bx;

end

4'bx101: begin // or ori

ALUOut <= A | B;

Zero <= 1'bx;

end

4'bx010: begin // xor xori beq bne

ALUOut <= A ^ B;

if(branch)

if((A ^ B) == 32'd0)

Zero <= 1'b1;

else

Zero <= 1'b0;

else

Zero <= 1'bx;

end

4'b0011: begin // sll

ALUOut <= B << A;

Zero <= 1'bx;

end

4'b0111: begin // srl

ALUOut <= B >> A;

Zero <= 1'bx;

end

4'bx110: begin // lui

ALUOut <= B << 16;

Zero <= 1'bx;

end

default: begin // jr j jal

ALUOut <= 32'dx;

Zero <= 1'bx;

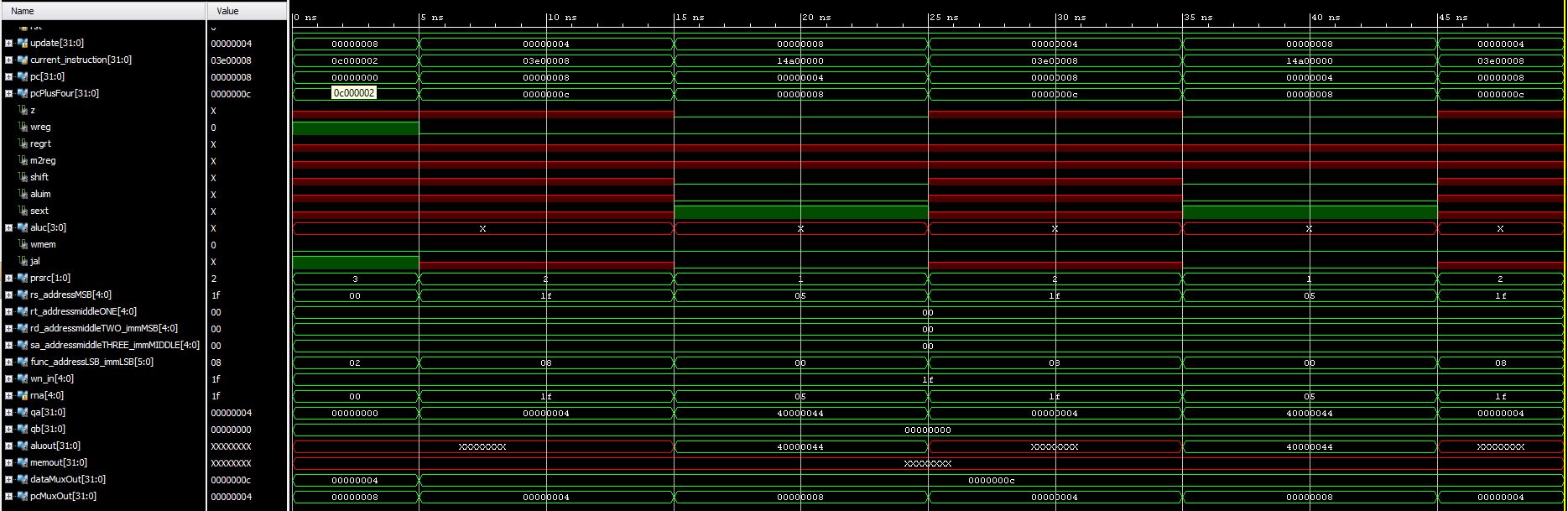
end

endcase

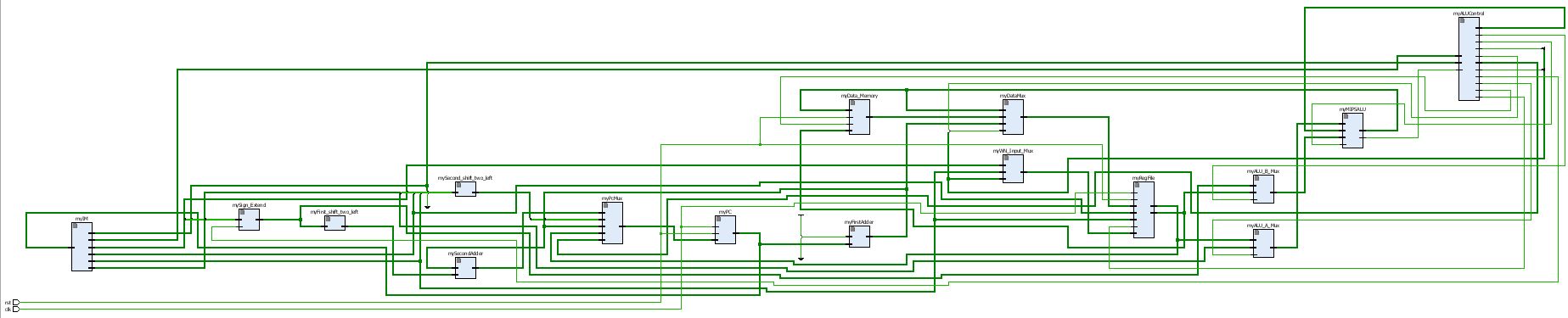
end

endmodule

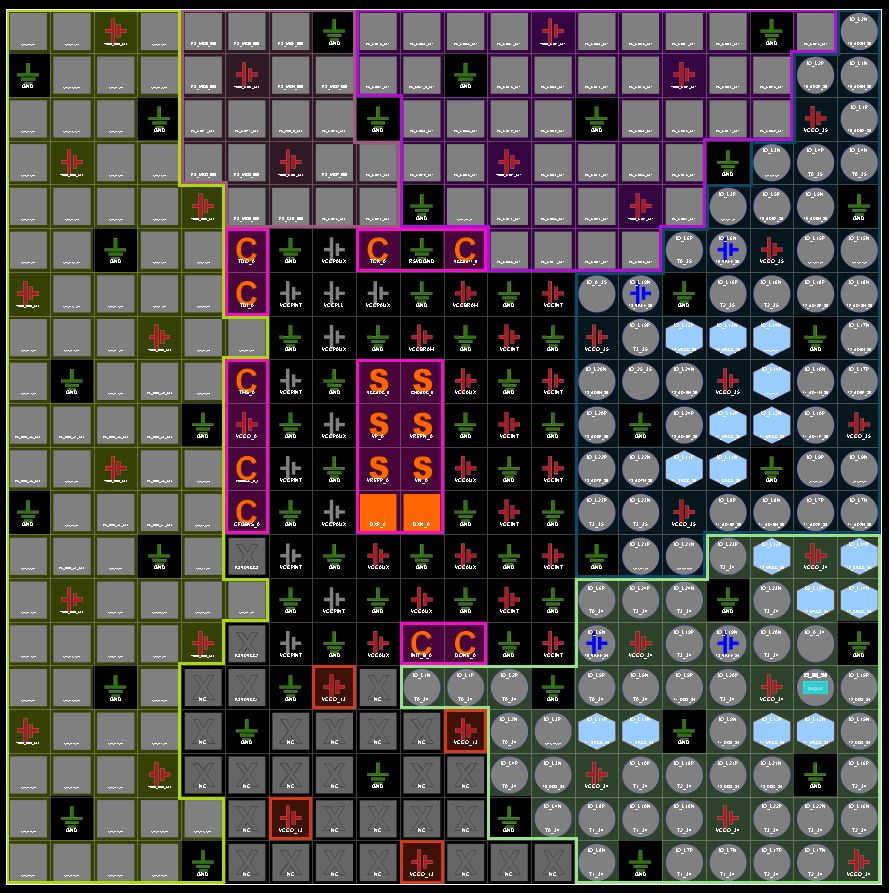
# Waveforms



# Design Schematic



# I/O Planning



# Floor Planning

# 