CMPEN 331 Summer 2022

Lab5

Fatma ALwasmi

module pc\_counter(

input [31:0] nextPc,

input clock,

output reg [31:0] pc

);

initial

begin

pc = 100;

end

always @(posedge clock)

begin

pc <= nextPc;

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module inst\_mem(

input [31:0] pc,

output reg [31:0] instOut

);

reg [31:0] memory [0:63];

initial

begin

memory[25] = {6'b100011, 5'd1, 5'd2, 16'd0}; //100:lw $2, 00($1)

memory[26] = {6'b100011, 5'd1, 5'd3, 16'd4};

memory[27] = {6'b100011, 5'd1, 5'd4, 16'd8};

memory[28] = {6'b100011, 5'd1, 5'd5, 16'd12};

memory[29] = {6'b000000, 5'd2, 5'd10, 5'd6, 5'd0, 6'd100100};

end

always@(\*)

begin

instOut <= memory[pc[7:2]];

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module pc\_adder(

input [31:0] pc,

output reg [31:0] nextPc

);

always @(\*)

begin

nextPc <= pc + 32'd4;

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module ifid\_reg(

input [31:0] instOut,

input clock,

output reg [31:0] distOut

);

always @(posedge clock)

begin

distOut <= instOut;

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module control\_unit(

input [5:0] op,

input [5:0] func,

output reg wreg,

output reg m2reg,

output reg wmem,

output reg [3:0] aluc,

output reg aluimm,

output reg regrt

);

always @(\*)

begin

wreg <=0;

m2reg <=0;

wmem <= 0;

aluc <=4'b0000;

aluimm <= 0;

regrt <=0;

case(op)

6'b000000: //r-types

begin

case(func)

6'b100000: //add instruction

begin

aluc <= 4'b0010;

wreg <=1;

end

endcase

end

6'b100011: //lw

begin

aluc <= 4'b0010;

wreg <= 1;

m2reg <= 1;//set values for control sig for LW

aluimm <=1;

regrt <= 1;

end

endcase

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module regrt(

input [4:0] rt,

input [4:0] rd,

input regrt,

output reg [4:0] destreg

);

always @(\*)

begin

if (regrt == 0)

begin

destreg <=rd;

end

else if (regrt == 1)

begin

destreg <= rt;

end

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module register\_file(

input [4:0] rs,

input [4:0] rt,

input [4:0] wdestreg,

input [31:0] wbdata,

input wwreg,

input clk,

output reg [31:0] qa,

output reg [31:0] qb

);

reg [31:0] register [0:31];

integer i;

initial

begin

for(i=0; i<32; i=i+1)

begin

register[i] = 32'b0;

end

end

always @(\*)

begin

qa = register[rs];

qb = register[rt];

if (clk == 0 && wwreg == 1'b1)

begin

register[wdestreg] <=wbdata;

end

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module immediate\_extender(

input [15:0] imm,

output reg [31:0] imm32

);

always@(\*)

begin

imm32 <= {{16{imm[15]}},imm};

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module idexe\_reg(

input clock,

input wreg,

input m2reg,

input wmem,

input [3:0] aluc,

input aluimm,

input [5:0] destReg,

input [31:0]qa,

input [31:0] qb,

input [31:0] imm32,

output reg ewreg,

output reg em2reg,

output reg ewmem,

output reg [3:0] ealuc,

output reg ealuimm,

output reg [4:0]edestReg,

output reg [31:0] eqa,

output reg [31:0]eqb,

output reg [31:0]eimm32

);

always @(posedge clock)

begin

ewreg <= wreg;

em2reg <= m2reg;

ewmem <= wmem;

ealuc <= aluc;

ealuimm <= aluimm;

edestReg <= destReg;

eqa <= qa;

eqb <= qb;

eimm32 <=imm32;

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module alu\_mux(

input[31:0] eqb,

input [31:0] eimm32,

input ealuimm,

output reg [31:0] b

);

always@(\*)

begin

case(ealuimm)

0: b <= eqb;

1'b1: b <= eimm32;

endcase

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module alu(

input [31:0] eqa,

input [31:0] b,

input [3:0] ealuc,

output reg [31:0] r

);

always@(\*) begin

case(ealuc)

4'b0000: r<= eqa & b;

4'b0001: r<= eqa | b;

4'b0010: r<= eqa + b;

4'b0110: r<= eqa - b;

4'b0111: r<= eqa < b? 1:0;

4'b1100: r<= ~(eqa | b);

endcase

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module exemem(

input ewreg,

input em2reg,

input ewmem,

input [4:0] edestreg,

input [31:0] r,

input [31:0] eqb,

input clock,

output reg mwreg,

output reg mm2reg,

output reg mwmem,

output reg [4:0] mdestreg,

output reg [31:0] mr,

output reg [31:0] mqb

);

always@(posedge clock)

begin

mwreg<=ewreg;

mm2reg<=em2reg;

mwmem<=ewmem;

mdestreg<=edestreg;

mr<=r;

mqb <=eqb;

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module data\_memory(

input [31:0] mr,

input [31:0] mqb,

input mwmem,

input clock,

output reg [31:0] mdo

);

reg [31:0] data\_mem [0:63];

initial

begin

data\_mem[0] <= 32'hA00000AA;

data\_mem[1] <= 32'h10000011;

data\_mem[2] <= 32'h20000022;

data\_mem[3] <= 32'h30000033;

data\_mem[4] <= 32'h40000044;

data\_mem[5] <= 32'h50000055;

data\_mem[6] <= 32'h60000066;

data\_mem[7] <= 32'h70000077;

data\_mem[8] <= 32'h80000088;

data\_mem[9] <= 32'h90000099;

end

always@(\*)

begin

mdo = data\_mem[mr[7:2]];

if(clock == 0 && mwmem ==1'b1)

begin

data\_mem[mr[7:2]] = mqb;

end

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module memwb(

input mwreg,

input mm2reg,

input [4:0] mdestreg,

input [31:0] mr,

input [31:0] mdo,

input clock,

output reg wwreg,

output reg wm2reg,

output reg [4:0] wdestreg,

output reg [31:0] wr,

output reg [31:0] wdo

);

always@(posedge clock)

begin

wwreg<=mwreg;

wm2reg<=mm2reg;

wdestreg<=mdestreg;

wr<=mr;

wdo<=mdo;

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module wbmux(

input [31:0] wr,

input [31:0] wdo,

input wm2reg,

output reg [31:0] wbdata

);

always@(\*)

begin

case(wm2reg)

1'b0: wbdata <= wr;

1'b1: wbdata <= wdo;

endcase

end

endmodule

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

module labv(

input clock,

output wire [31:0] pc,

output wire [31:0] dinstOut,

output wire ewreg,

output wire em2reg,

output wire ewmem,

output wire [3:0] ealuc,

output wire ealuimm,

output wire [4:0]edestReg,

output wire [31:0] eqa,

output wire [31:0]eqb,

output wire [31:0]eimm32,

output wire mwreg,

output wire mm2reg,

output wire mwmem,

output wire [4:0] mdestreg,

output wire [31:0] mr,

output wire [31:0] mqb,

output wire wwreg,

output wire wm2reg,

output wire [4:0] wdestreg,

output wire [31:0] wr,

output wire [31:0] wdo

);

wire [31:0] const;

wire [5:0] op = dinstOut[31:26];

wire [5:0] func = dinstOut[5:0];

wire [4:0]rt = dinstOut[20:16];

wire [4:0] rd = dinstOut[15:11];

wire [4:0] rs = dinstOut[25:21];

wire [15:0] imm = dinstOut[15:0];

wire [31:0] nextPc;

wire [31:0] instOut;

wire wreg;

wire m2reg;

wire wmem;

wire [3:0] aluc;

wire aluimm;

wire regrt;

wire [4:0] destReg;

wire [31:0] qa;

wire [31:0] qb;

wire [31:0] b;

wire [31:0] r;

wire [31:0] mdo;

wire [31:0] wbdata;

wire [31:0] imm32;

pc\_adder adder(pc,nextPc);

pc\_counter pcCounter(nextPc,clock, pc);

inst\_mem instMem(pc,instOut);

ifid\_reg ifid(instOut,clock,dinstOut);

control\_unit controlUnit(op, func, wreg, m2reg, wmem, aluc, aluimm,regrt);

regrt reg\_rt(rt, rd,regrt, destReg);

register\_file reg\_file(rs, rt, wdestreg, wbdata, wwreg, clock, qa, qb);

immediate\_extender immediateExtender(imm,imm32);

idexe\_reg idexe(clock, wreg, m2reg, wmem, aluc, aluimm, destReg, qa, qb, imm32, ewreg, em2reg, ewmem, ealuc, ealuimm, edestReg, eqa, eqb, eimm32);

alu\_mux aluMux(eqb, eimm32, ealuimm, b);

alu ALU(eqa, b, ealuc, r);

exemem EXEMEM(ewreg, em2reg, ewmem, edestReg, r, eqb, clock, mwreg, mm2reg, mwmem, mdestreg, mr, mqb);

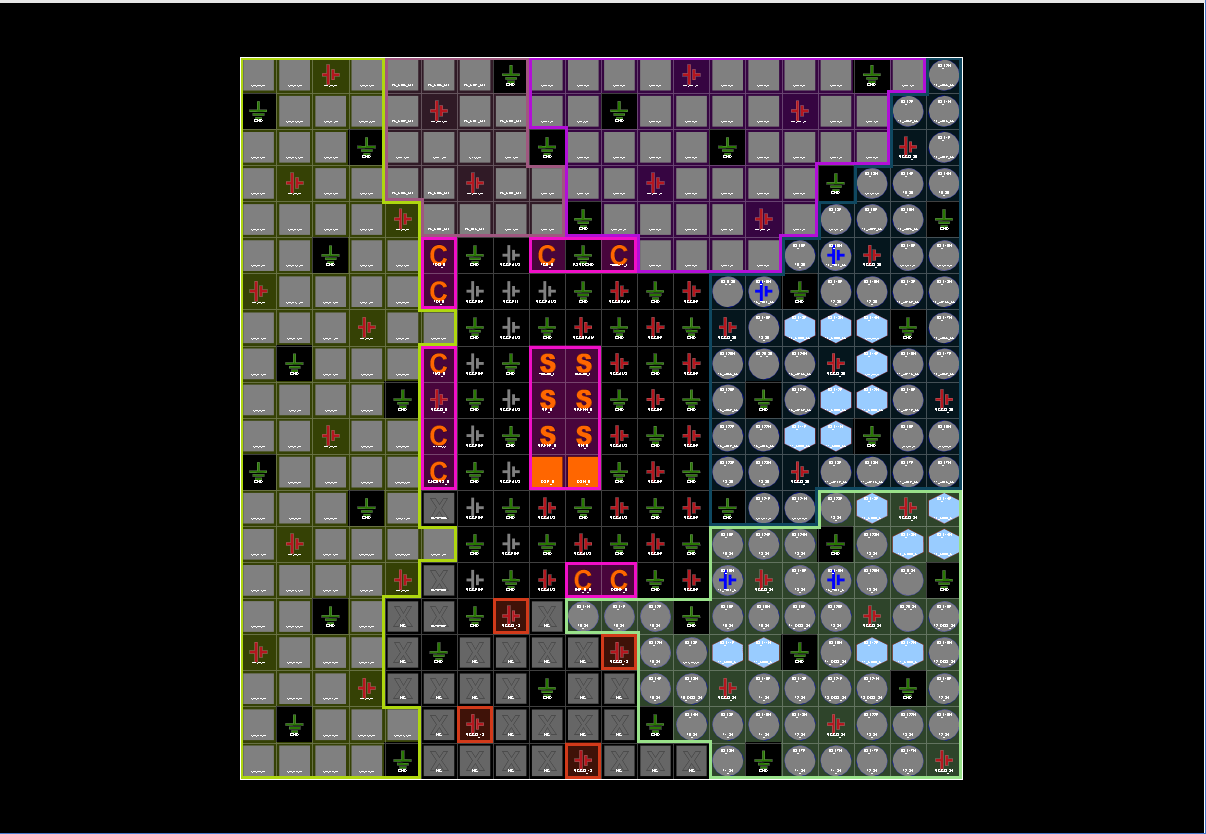
data\_memory dataMem(mr, mqb, mwmem, clock, mdo);

memwb MEMWB(mwreg, mm2reg, mdestreg, mr, mdo, clock, wwreg, wm2reg, wdestreg, wr, wdo);

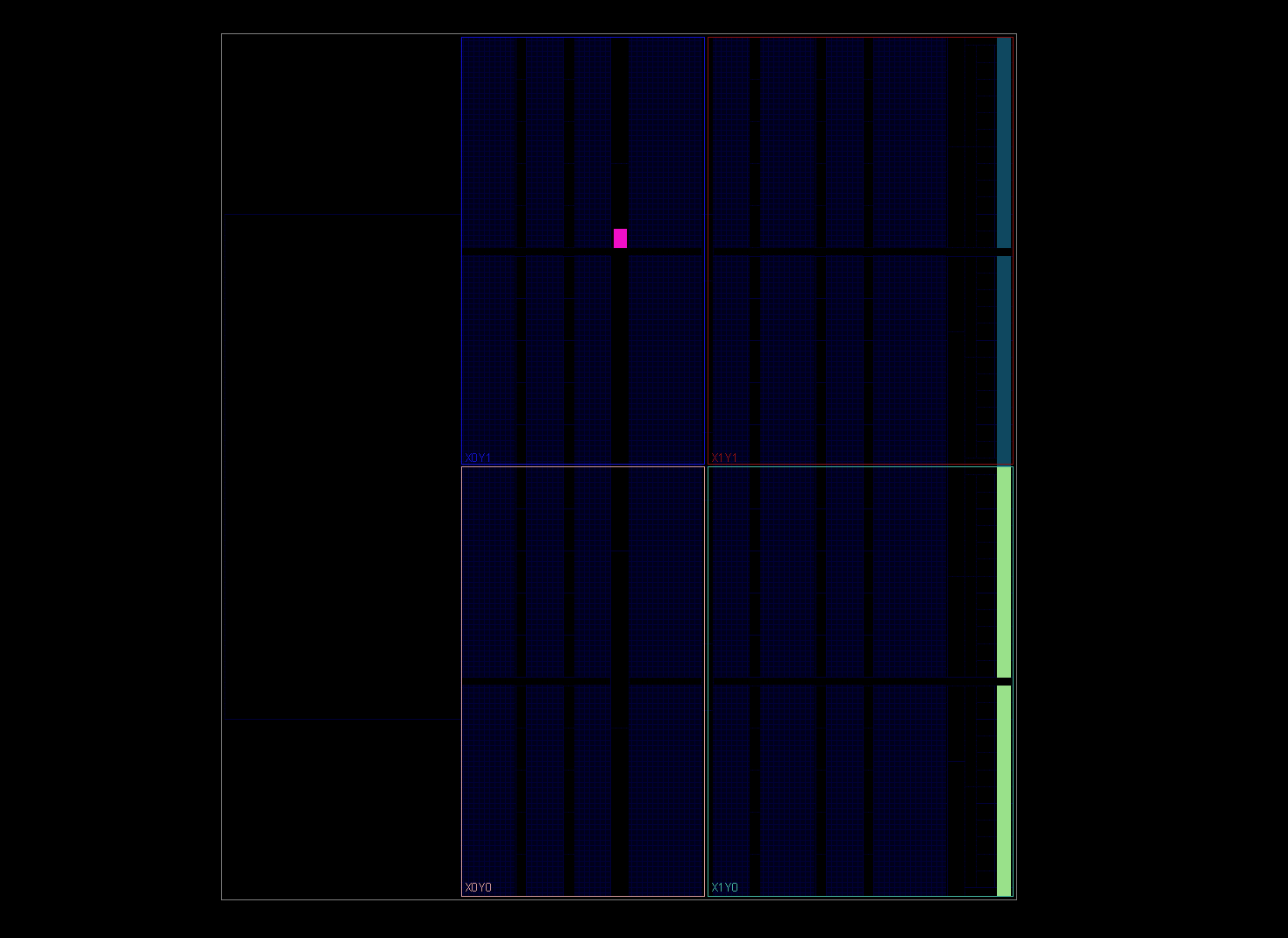
wbmux wbMux(wr, wdo, wm2reg, wbdata);

endmodule

**i/o planning**

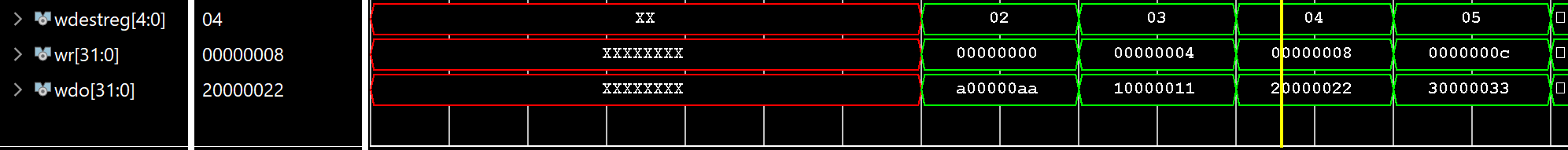
****

**Floor Panning**

****

**Waveform**

****

****