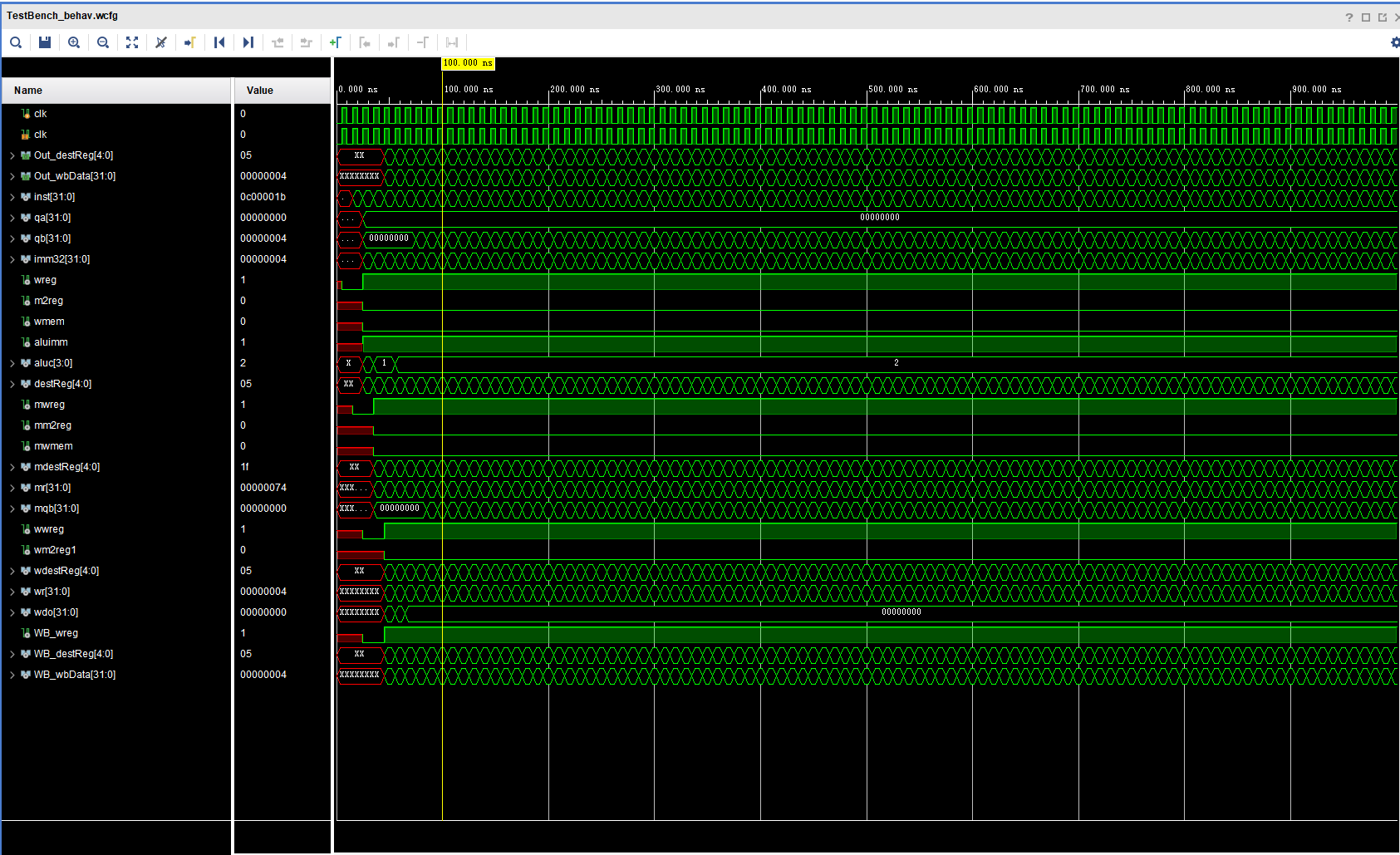
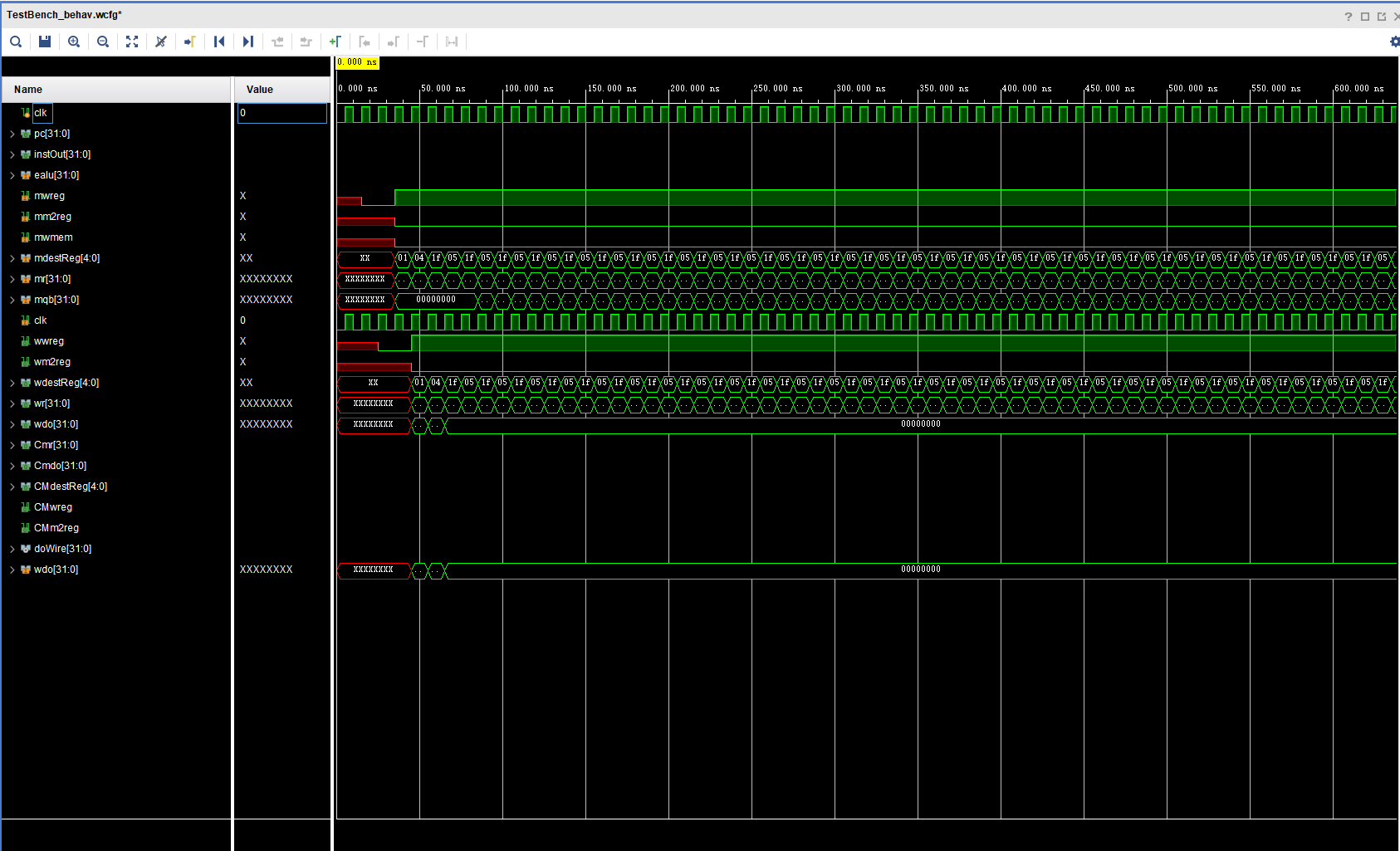
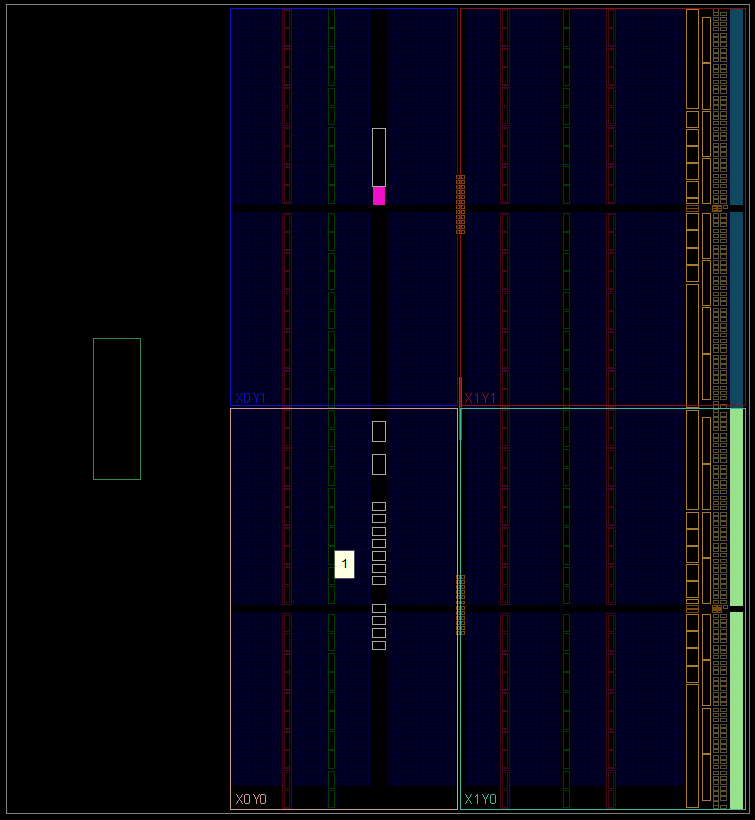
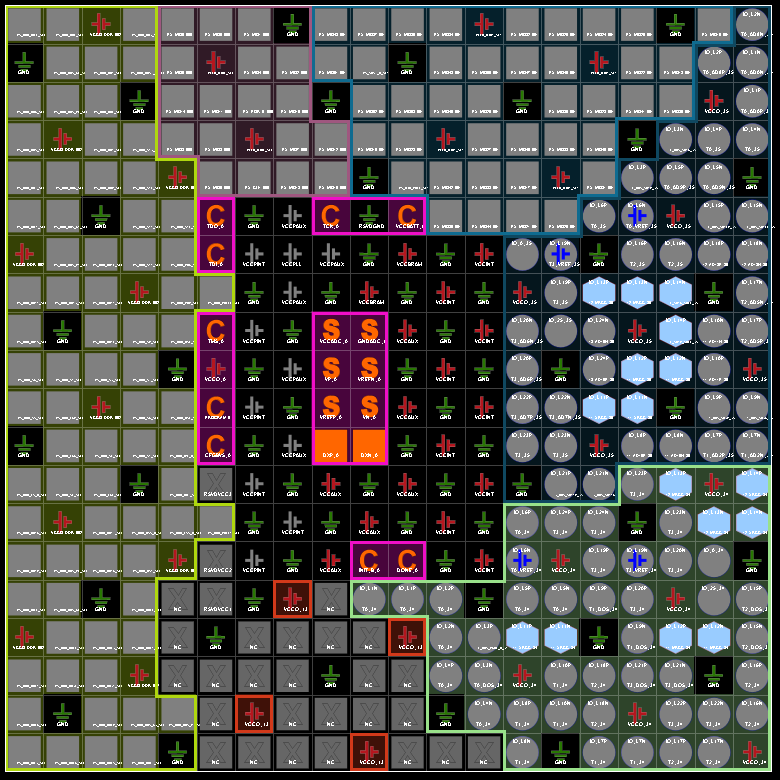
CMPEN 331 Lab 5

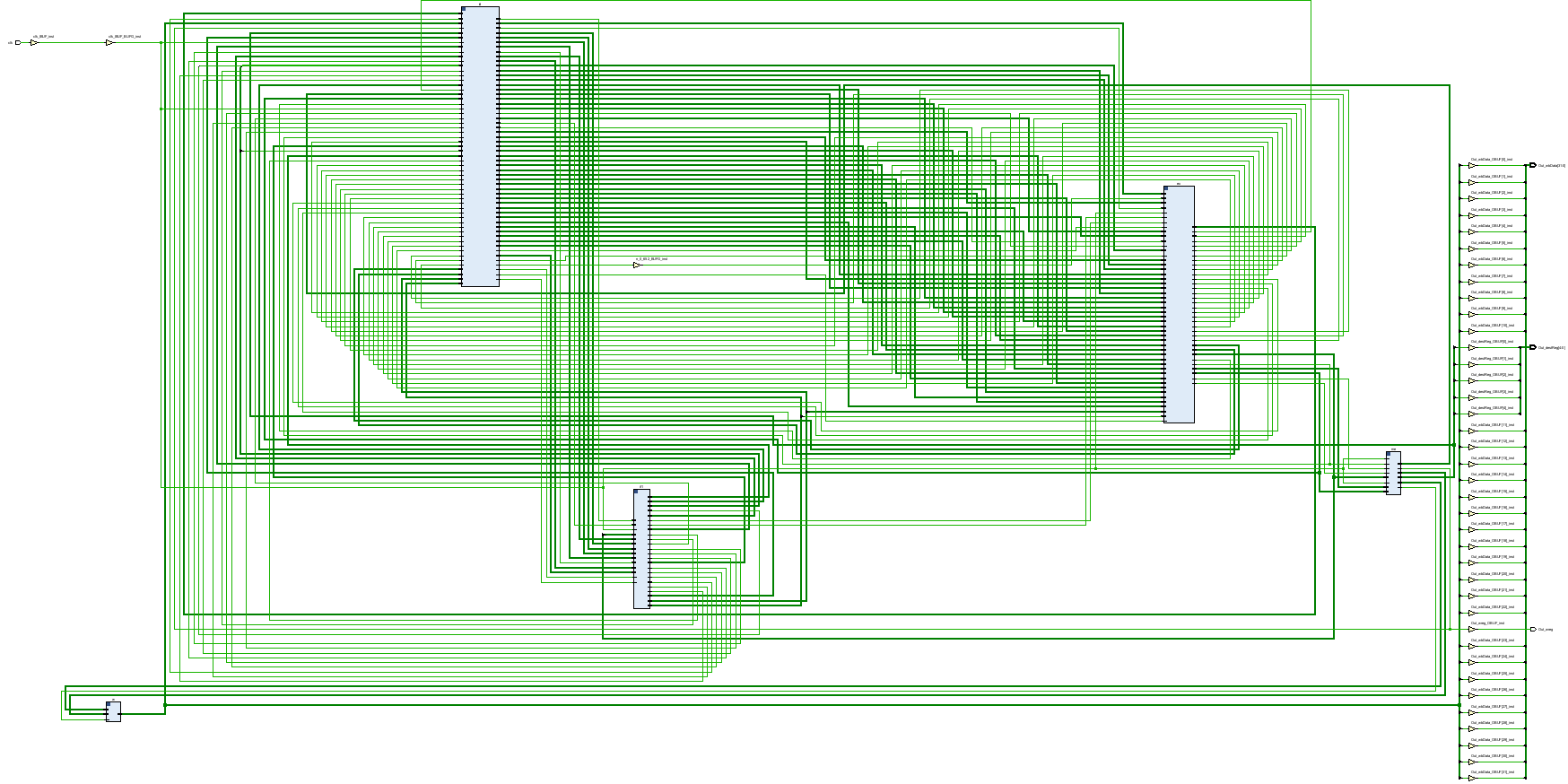
Shi Qiu











`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create: 2022/04/10 13:18:23

// Design Name:

// Module Name:

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module DataPath(

input clk,

output Out\_wreg,

output [4:0] Out\_destReg,

output [31:0] Out\_wbData

);

wire [31:0] inst;

wire [31:0] qa;

wire [31:0] qb;

wire [31:0] imm32;

wire wreg;

wire m2reg;

wire wmem;

wire aluimm;

wire [3:0] aluc;

wire [4:0] destReg;

wire mwreg;

wire mm2reg;

wire mwmem;

wire [4:0] mdestReg;

wire [31:0] mr, mqb;

wire wwreg;

wire wm2reg1;

wire [4:0] wdestReg;

wire [31:0] wr;

wire [31:0] wdo;

wire WB\_wreg;

wire [4:0] WB\_destReg;

wire [31:0] WB\_wbData;

wire [1:0] pcsrc;

wire [31:0] ra, jpcFull, bpc, dpc;

wire wpcir;

wire [4:0] CmdestReg;

wire Cmm2Reg, Cmwreg;

wire [4:0] CedestReg;

wire Cem2Reg, Cewreg;

wire [31:0] dpc4, Dr, Dmr, Dmdo;

wire ejal, eshift;

wire [31:0] epc4;

InstFetch if1(clk, wpcir, bpc, ra, jpcFull, pcsrc, inst, dpc);

InstDecode id(inst, WB\_wreg, WB\_destReg, WB\_wbData, clk, CmdestReg, Cmm2Reg, Cmwreg, CedestReg, Cem2Reg, Cewreg, dpc, Dr, Dmr, Dmdo,

wreg, m2reg, wmem, aluc, aluimm, destReg, qa, qb, imm32, wpcir, pcsrc, jpcFull, bpc, ra, ejal, eshift, epc4);

Exe ex(wreg, m2reg, wmem, aluc, aluimm, destReg, qa, qb, imm32, clk, ejal, eshift, epc4,

mwreg, mm2reg, mwmem, mdestReg, mr, mqb, Dr, CedestReg, Cewreg, Cem2Reg);

Mem me(mwreg, mm2reg, mwmem, mdestReg, mr, mqb, clk, wwreg, wm2reg1, wdestReg, wr, wdo, Dmr, Dmdo, CmdestReg, Cmwreg, Cmm2Reg);

WB w(wwreg, wm2reg1, wdestReg, wr, wdo, WB\_wreg, WB\_destReg, WB\_wbData);

assign Out\_wreg = WB\_wreg;

assign Out\_destReg = WB\_destReg;

assign Out\_wbData = WB\_wbData;

endmodule

`timescale 1ns/1ps

module pc(

input [31:0] nextPC,

input clk,

input wpcir,

output reg [31:0] pc

);

initial pc = 100;

always @(posedge clk) begin

if (wpcir) pc <= nextPC;

end

endmodule

module IFIDreg(

input [31:0] instOut,

input clk,

input wpcir,

input [31:0] pc4,

output reg [31:0] dinstOut,

output reg [31:0] dpc4

);

always @ (posedge clk) begin

if (wpcir) begin

dinstOut <= instOut;

dpc4 <= pc4;

end

end

endmodule

module pcAdd(

input [31:0] pc,

output [31:0] pc4

);

assign pc4 = pc + 32'd4;

endmodule

module pcAdd(

input [31:0] pc,

output [31:0] pc4

);

assign pc4 = pc + 32'd4;

endmodule

module instMem(

input [31:0] pc,

output reg [31:0] instOut

);

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

initial begin

//lw $2, 00($1)lw $2, 00($1) # $2 ?? memory[$1+00]; load x[0]

// lw $3, 04($1) # $3 ?? memory[$1+04]; load x[1]

//lw $4, 08($1) # $4 ?? memory[$1+08]; load x[2]

//lw $5, 12($1) # $5 ?? memory[$1+12]; load x[3]

//add $6, $2, $10

memory[25] = 32'h3c010000; // (00) main: lui $1, 0

memory[26] = 32'h34240050; // (04) ori $4, $1, 80

memory[27] = 32'h0c00001b; // (08) call: jal sum

memory[28] = 32'h20050004; // (0c) dslot1: addi $5, $0, 4

memory[29] = 32'hac820000; // (10) return: sw $2, 0($4)

memory[30] = 32'h8c890000; // (14) lw $9, 0($4)

memory[31] = 32'h01244022; // (18) sub $8, $9, $4

memory[32] = 32'h20050003; // (1c) addi $5, $0, 3

memory[33] = 32'h20a5ffff; // (20) loop2: addi $5, $5, -1

memory[34] = 32'h34a8ffff; // (24) ori $8, $5, 0xffff

memory[35] = 32'h39085555; // (28) xori $8, $8, 0x5555

memory[36] = 32'h2009ffff; // (2c) addi $9, $0, -1

memory[37] = 32'h312affff; // (30) andi $10,$9,0xffff

memory[38] = 32'h01493025; // (34) or $6, $10, $9

memory[39] = 32'h01494026; // (38) xor $8, $10, $9

memory[40] = 32'h01463824; // (3c) and $7, $10, $6

memory[41] = 32'h10a00003; // (40) beq $5, $0, shift

memory[42] = 32'h00000000; // (44) dslot2: nop

memory[43] = 32'h08000008; // (48) j loop2

memory[44] = 32'h00000000; // (4c) dslot3: nop

memory[45] = 32'h2005ffff; // (50) shift: addi $5, $0, -1

memory[46] = 32'h000543c0; // (54) sll $8, $5, 15

memory[47] = 32'h00084400; // (58) sll $8, $8, 16

memory[48] = 32'h00084403; // (5c) sra $8, $8, 16

memory[49] = 32'h000843c2; // (60) srl $8, $8, 15

memory[50] = 32'h08000019; // (64) finish: j finish

memory[51] = 32'h00000000; // (68) dslot4: nop

memory[52] = 32'h00004020; // (6c) sum: add $8, $0, $0

memory[53] = 32'h8c890000; // (70) loop: lw $9, 0($4)

memory[54] = 32'h01094020; // (74) stall: add $8, $8, $9

memory[55] = 32'h20a5ffff; // (78) addi $5, $5, -1

memory[56] = 32'h14a0fffc; // (7c) bne $5, $0, loop

memory[57] = 32'h20840004; // (80) dslot5: addi $4, $4, 4

memory[58] = 32'h03e00008; // (84) jr $31

memory[59] = 32'h00081000; // (88) dslot6: sll $2, $8, 0

end

always @ (\*) begin

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

end

endmodule

module pcMUX(

input [31:0] pc4,

input [31:0] bpc,

input [31:0] ra,

input [31:0] jpcFull,

input [1:0] pcsrc,

output [31:0] nextPC

);

assign nextPC = (pcsrc == 2'd0)? pc4 : ((pcsrc == 2'd1)? bpc : ((pcsrc == 2'd2)? ra : jpcFull));

endmodule

module InstFetch(

input clk,

input wpcir,

input [31:0] bpc,

input [31:0] ra,

input [31:0] jpcFull,

input [1:0] pcsrc,

output [31:0] dinstOut,

output [31:0] dpc4

);

wire [31:0] wpc;

wire [31:0] wnextPC;

wire [31:0] winstOut;

wire [31:0] pc4;

wire [31:0] wpc4;

pc pc1(wnextPC, clk, wpcir, wpc);

pcAdd pa(wpc, wpc4);

pcMUX pcm(wpc4, bpc, ra, jpcFull, pcsrc, wnextPC);

instMem inmem(wpc, winstOut);

IFIDreg IFID(winstOut, clk, wpcir, wpc4, dinstOut, dpc4);

endmodule

module Control(

input [5:0] op,

input [5:0] func,

input [4:0] rs, rt, mdestReg,

input mm2Reg, mwreg,

input [4:0] edestReg,

input em2Reg, ewreg,

input RsrtEqu,

output reg wreg,

output reg m2reg,

output reg wmem,

output reg [3:0] aluc,

output reg aluImm,

output reg regrt,

output reg [1:0] fwd\_A, fwd\_B,

output reg wpcir,

output reg jal,

output reg shift,

output reg sext,

output reg [1:0] pcsrc

);

reg regUsage;

initial begin

wreg <= 0;

pcsrc <= 0;

end

always @(\*) begin

if (ewreg && em2Reg && (edestReg != 0) && regUsage && (edestReg == rs || edestReg == rt))

begin

wreg <= 0;

wmem <= 0;

wpcir <= 0;

end

else wpcir <= 1;

if (ewreg & (edestReg != 0) & (edestReg == rs) & ~em2Reg) fwd\_A <= 1;

else if (mwreg & (mdestReg != 0) & (mdestReg == rs) & ~mm2Reg) fwd\_A <= 2;

else if (mwreg & (mdestReg != 0) & (mdestReg == rs) & mm2Reg) fwd\_A <= 3;

else fwd\_A <= 0;

if (ewreg & (edestReg != 0) & (edestReg == rt) & ~em2Reg) fwd\_B <= 1;

else if (mwreg & (mdestReg != 0) & (mdestReg == rt) & ~mm2Reg) fwd\_B <= 2;

else if (mwreg & (mdestReg != 0) & (mdestReg == rt) & mm2Reg) fwd\_B <= 3;

else fwd\_B <= 0;

case (op)

6'b000000: //for different cases

begin

case (func)

6'b100000:

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0010;

aluImm <= 0;

regrt <= 0;

jal <= 0;

shift <= 0;

sext <= 1;

pcsrc <= 2'd0;

regUsage <= 1;

end

6'b100010:

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0110;

aluImm <= 0;

regrt <= 0;

jal <= 0;

shift <= 0;

sext <= 1;

pcsrc <= 2'd0;

regUsage <= 1;

end

6'b100100:

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0000;

aluImm <= 0;

regrt <= 0;

jal <= 0;

shift <= 0;

sext <= 1;

pcsrc <= 2'd0;

regUsage <= 1;

end

6'b100101:

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0001;

aluImm <= 0;

regrt <= 0;

jal <= 0;

shift <= 0;

sext <= 1;

pcsrc <= 2'd0;

regUsage <= 1;

end

6'b100110:

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0111;

aluImm <= 0;

regrt <= 0;

jal <= 0;

shift <= 0;

sext <= 1;

pcsrc <= 2'd0;

regUsage <= 1;

end

6'b000000:

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b1010;

aluImm <= 0;

regrt <= 0;

jal <= 0;

shift <= 1;

sext <= 1;

pcsrc <= 2'd0;

regUsage <= 1;

end

6'b000010:

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b1100;

aluImm <= 0;

regrt <= 0;

jal <= 0;

shift <= 1;

sext <= 1;

pcsrc <= 2'd0;

regUsage <= 1;

end

6'b000010:

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b1011;

aluImm <= 0;

regrt <= 0;

jal <= 0;

shift <= 1;

sext <= 1;

pcsrc <= 2'd0;

regUsage <= 1;

end

endcase

end

6'b100011:

begin

wreg <= 1;

m2reg <= 1;

wmem <= 0;

aluc <= 4'b0010;

aluImm <= 1;

regrt <= 1;

jal <= 0;

shift <= 0;

sext <= 1;

pcsrc <= 2'd0;

regUsage <= 1;

end

6'b101011:

begin

wreg <= 0;

m2reg <= 0;

wmem <= 1;

aluc <= 4'b0010;

aluImm <= 1;

regrt <= 1;

jal <= 0;

shift <= 0;

sext <= 1;

pcsrc <= 2'd0;

regUsage <= 1;

end

6'b001000:

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0010;

aluImm <= 1;

regrt <= 1;

jal <= 0;

shift <= 0;

sext <= 0;

pcsrc <= 2'd0;

regUsage <= 1;

end

6'b001100:

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0000;

aluImm <= 1;

regrt <= 1;

jal <= 0;

shift <= 0;

sext <= 0;

pcsrc <= 2'd0;

regUsage <= 1;

end

6'b001101:

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0001;

aluImm <= 1;

regrt <= 1;

jal <= 0;

shift <= 0;

sext <= 0;

pcsrc <= 2'd0;

regUsage <= 1;

end

6'b001110:

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b0111;

aluImm <= 1;

regrt <= 1;

jal <= 0;

shift <= 0;

sext <= 0;

pcsrc <= 2'd0;

regUsage <= 1;

end

6'b000100:

begin

if (RsrtEqu) pcsrc <= 2'd1;

else pcsrc <= 2'd0;

jal <= 0;

shift <= 0;

sext <= 1;

regUsage <= 1;

end

6'b000101:

begin

if (RsrtEqu) pcsrc <= 2'd0;

else pcsrc <= 2'd1;

jal <= 0;

shift <= 0;

sext <= 1;

regUsage <= 1;

end

6'b001111:

begin

wreg <= 1;

m2reg <= 0;

wmem <= 0;

aluc <= 4'b1110;

aluImm <= 1;

regrt <= 1;

jal <= 0;

shift <= 0;

sext <= 0;

pcsrc <= 2'd0;

regUsage <= 0;

end

6'b000010:

begin

pcsrc <= 2'd3;

jal <= 0;

shift <= 0;

sext <= 1;

wreg <= 0;

wmem <= 0;

regUsage <= 0;

end

6'b000011:

begin

jal <= 1;

pcsrc <= 2'd3;

shift <= 0;

sext <= 1;

wreg <= 1;

wmem <= 0;

regUsage <= 0;

m2reg <= 0;

end

6'b000011:

begin

pcsrc <= 2'd2;

jal <= 0;

shift <= 0;

sext <= 1;

wreg <= 0;

wmem <= 0;

regUsage <= 0;

end

endcase

end

endmodule

module forwardMux(

input [1:0] fwd,

input [31:0] regOut, r, mr, mdo,

output reg [31:0] q

);

always @ (\*) begin

case (fwd)

2'd0: q <= regOut;

2'd1: q <= r;

2'd2: q <= mr;

2'd3: q <= mdo;

endcase

end

endmodule

module jump(

input [25:0] addr,

input [31:0] dcp4,

output [31:0] jpcFull

);

wire [31:0] jpc;

wire [29:0] temp;

assign temp = {4'd0, addr};

assign jpc = temp << 2;

assign jpcFull = {dcp4[31:28], jpc[27:0]};

endmodule

module Branch(

input [15:0] imm,

input [31:0] dpc4,

output [31:0] bpc

);

assign bpc = {{{16'd0}, imm} << 2} + dpc4;

endmodule

module equCheck(

input [31:0] qa, qb,

output rsrtequ

);

assign rsrtequ = (qa == qb) ? 1'b1 : 1'b0;

endmodule

module regRTMux(

input [4:0] rt,

input [4:0] rd,

input regrt,

output reg [4:0] destReg

);

always @ (\*) begin

if (~regrt) destReg <= rd;

else if (regrt) destReg <= rt;

end

endmodule

module regFile(

input [4:0] rs,

input [4:0] rt,

input wwreg,

input [4:0] wdestReg,

input [31:0] wbData,

input clk,

output reg [31:0] qa,

output reg [31:0] qb

);

reg [31:0] registers [0:31];

initial

//begin

// for i in 0 to 31 loop

// registers[i] = 32'd0;

//end loop

//end

begin

registers[0] = 32'd0;

registers[1] = 32'd0;

registers[2] = 32'd0;

registers[3] = 32'd0;

registers[4] = 32'd0;

registers[5] = 32'd0;

registers[6] = 32'd0;

registers[7] = 32'd0;

registers[8] = 32'd0;

registers[9] = 32'd0;

registers[10] = 32'd0;

registers[11] = 32'd0;

registers[12] = 32'd0;

registers[13] = 32'd0;

registers[14] = 32'd0;

registers[15] = 32'd0;

registers[16] = 32'd0;

registers[17] = 32'd0;

registers[18] = 32'd0;

registers[19] = 32'd0;

registers[20] = 32'd0;

registers[21] = 32'd0;

registers[22] = 32'd0;

registers[23] = 32'd0;

registers[24] = 32'd0;

registers[25] = 32'd0;

registers[26] = 32'd0;

registers[27] = 32'd0;

registers[28] = 32'd0;

registers[29] = 32'd0;

registers[30] = 32'd0;

registers[31] = 32'd0;

end

always @ (\*) begin

qa = registers[rs];

qb = registers[rt];

end

always @ (negedge clk) begin

if (wwreg) begin

registers[wdestReg] <= wbData;

end

end

endmodule

module ImmExt(

input [15:0] imm,

input sext,

output reg [31:0] imm32

);

always @ (\*) begin

if (sext) imm32 <= {{17{imm[15]}}, imm[14:0]};

else imm32 <= {{17'd0}, imm[14:0]};

end

endmodule

module IDEXEreg(

input wreg,

input m2reg,

input wmem,

input [3:0] aluc,

input aluimm,

input [4:0] destReg,

input [31:0] qa,

input [31:0] qb,

input [31:0] imm32,

input clk,

input jal, shift,

input [31:0] dpc4,

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,

output reg ejal, eshift,

output reg [31:0] epc4

);

always @ (posedge clk) begin

ewreg <= wreg;

em2reg <= m2reg;

ewmem <= wmem;

ealuc <= aluc;

ealuimm <= aluimm;

edestReg <= destReg;

eqa <= qa;

eqb <= qb;

eimm32 <= imm32;

ejal <= jal;

eshift <= shift;

epc4 <= dpc4;

end

endmodule

module InstDecode(

input [31:0] dinstOut,

input wwreg,

input [4:0] wdestReg,

input [31:0] wbData,

input clk,

input [4:0] mdestReg,

input mm2Reg, mwreg,

input [4:0] edestReg,

input em2Reg, ewreg,

input [31:0] dpc4, r, mr, mdo,

output owreg,

output om2reg,

output owmem,

output [3:0] oaluc,

output oaluimm,

output [4:0] odestReg,

output [31:0] oqa,

output [31:0] oqb,

output [31:0] oimm32,

output wpcir,

output [1:0] pcsrc,

output [31:0] jpcFull, bpc,

output [31:0] ra,

output ejal, eshift,

output [31:0] epc4

);

wire wreg;

wire wm2reg;

wire wwmem;

wire [3:0] waluc;

wire waluimm;

wire [4:0] destReg;

wire [31:0] wqa;

wire [31:0] wqb;

wire [31:0] wimm32;

wire wrtMUX;

wire [1:0] fwd\_A, fwd\_B;

wire [31:0] Fqa, Fqb;

wire jal, shift, sext;

wire RsrtEqu;

assign ra = Fqa;

Control c(dinstOut[31:26], dinstOut[5:0], dinstOut[25:21], dinstOut[20:16], mdestReg, mm2Reg, mwreg, edestReg, em2Reg, ewreg, RsrtEqu,

wreg, wm2reg, wwmem, waluc, waluimm, wrtMUX, fwd\_A, fwd\_B, wpcir, jal, shift, sext, pcsrc);

forwardMux fwdA(fwd\_A, wqa, r, mr, mdo, Fqa);

forwardMux fwdB(fwd\_B, wqb, r, mr, mdo, Fqb);

jump j(dinstOut[25:0], dpc4, jpcFull);

Branch b(dinstOut[16:0], dpc4, bpc);

equCheck equ(Fqa, Fqb, RsrtEqu);

regRTMux rtMux(dinstOut[20:16], dinstOut[15:11], wrtMUX, destReg);

regFile rf(dinstOut[25:21], dinstOut[20:16], wwreg, wdestReg, wbData, clk, wqa, wqb);

ImmExt extend(dinstOut[15:0], sext, wimm32);

IDEXEreg idexe1(wreg, wm2reg, wwmem, waluc, waluimm, destReg, Fqa, Fqb, wimm32, clk, jal, shift, dpc4,

owreg, om2reg, owmem, oaluc, oaluimm, odestReg, oqa, oqb, oimm32, ejal, eshift, epc4);

endmodule

module aluMUX (

input [31:0] eqb,

input [31:0] eimm,

input ealuimm,

output reg [31:0] b

);

always @ (\*)

begin

if (ealuimm) b <= eimm;

else b <= eqb;

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; //xor

4'b1010: r <= b << eqa[10:6]; //sll

4'b1100: r <= b >> eqa[10:6]; //srl

4'b1011: r <= b >>> eqa[10:6]; //sra

4'b1110: r <= {b[31:16], 15'd0};

endcase

end

endmodule

module exemem(

input ewreg,

input em2reg,

input ewmem,

input [4:0] edestReg,

input [31:0] r,

input [31:0] eqb,

input clk,

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 clk) begin

mwreg <= ewreg;

mm2reg <= em2reg;

mwmem <= ewmem;

mdestReg <= edestReg;

mr <= r;

mqb <= eqb;

end

endmodule

module jal(

input [31:0] epc4, ealu,

input ejal,

output [31:0] er

);

assign er = (ejal) ? (epc4 + 32'd4) : ealu;

endmodule

module shiftMux(

input [31:0] sa, eqa,

input shift,

output [31:0] a

);

assign a = (shift) ? sa : eqa;

endmodule

module f(

input jal,

input [4:0] destReg,

output [4:0] edestReg

);

assign edestReg = (jal) ? 5'd31 : destReg;

endmodule

module Exe(

input ewreg,

input em2reg,

input ewmem,

input [3:0] ealuc,

input ealuimm,

input [4:0] edestReg,

input [31:0] eqa,

input [31:0] eqb,

input [31:0] eimm32,

input clk,

input ejal, eshift,

input [31:0] epc4,

output mwreg,

output mm2reg,

output mwmem,

output [4:0] mdestReg,

output [31:0] mr,

output [31:0] mqb,

output [31:0] er,

output [4:0] CEdestReg,

output CEwreg, CEm2reg

);

wire [31:0] rWire;

wire [31:0] bWire;

wire [31:0] aluOut, a;

wire [4:0] fdestReg;

assign CEdestReg = fdestReg;

assign CEwreg = ewreg;

assign CEm2reg = em2reg;

assign er = rWire;

aluMUX almu(eqb, eimm32, ealuimm, bWire);

shiftMux sft(eimm32, eqa, eshift, a);

Alu alu(a, bWire, ealuc, aluOut);

jal jl(epc4, aluOut, ejal, rWire);

f ff(ejal, edestReg, fdestReg);

exemem em(ewreg, em2reg, ewmem, fdestReg, rWire, eqb, clk, mwreg, mm2reg, mwmem, mdestReg, mr, mqb);

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create: 2022/04/10 13:18:23

// Design Name:

// Module Name:

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module Main\_Mem(

input [31:0] mr,

input [31:0] mqb,

input mwmem,

input clk,

output reg [31:0] mdo

);

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

integer i;

initial begin

// mem[0] = 32'hA00000AA;

// mem[1] = 32'h10000011;

// mem[2] = 32'h20000022;

// mem[3] = 32'h30000033;

// mem[4] = 32'h40000044;

// mem[5] = 32'h50000055;

// mem[6] = 32'h60000066;

// mem[7] = 32'h70000077;

// mem[8] = 32'h80000088;

// mem[9] = 32'h90000099;

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

mem[i] = 0;

mem[5'h14] = 32'h000000a3;

mem[5'h15] = 32'h00000027;

mem[5'h16] = 32'h00000079;

mem[5'h17] = 32'h00000115;

end

always @ (\*) begin

mdo <= mem[mr[7:2]];

end

always @ (negedge clk) begin

if (mwmem) begin

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 clk,

output reg wwreg,

output reg wm2reg,

output reg [4:0] wdestReg,

output reg [31:0] wr,

output reg [31:0] wdo

);

always @ (posedge clk) begin

wwreg <= mwreg;

wm2reg <=mm2reg;

wdestReg <= mdestReg;

wr <= mr;

wdo <= mdo;

end

endmodule

module Mem(

input mwreg,

input mm2reg,

input mwmem,

input [4:0] mdestReg,

input [31:0] mr,

input [31:0] mqb,

input clk,

output wwreg,

output wm2reg,

output [4:0] wdestReg,

output [31:0] wr,

output [31:0] wdo,

output [31:0] Cmr, Cmdo,

output [4:0] CMdestReg,

output CMwreg, CMm2reg

);

wire [31:0] doWire;

assign Cmr = mr;

assign Cmdo = doWire;

assign CMdestReg = mdestReg;

assign CMwreg = mwreg;

assign CMm2reg = mm2reg;

Main\_Mem m(mr, mqb, mwmem, clk, doWire);

memwb mw(mwreg, mm2reg, mdestReg, mr, doWire, clk, wwreg, wm2reg, wdestReg, wr, wdo);

endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create : 2022/04/10 13:18:23

// Design Name:

// Module Name:

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module WbMUX(

input wm2reg,

input [31:0] wr,

input [31:0] wdo,

output [31:0] wbData

);

assign wbData = wm2reg ? wdo : wr;

//always @ (\*) begin

//if (wm2reg)

//begin

// assign wbData = wdo;

//end

//else

//begin

// assign wbData = wr;

//end

//end

endmodule

module WB(

input wwreg,

input wm2reg,

input [4:0] wdestReg,

input [31:0] wr,

input [31:0] wdo,

output wreg,

output [4:0] destReg,

output [31:0] wbData

);

assign wreg = wwreg;

assign destReg = wdestReg;

WbMUX wbm(wm2reg, wr, wdo, wbData);

Endmodule

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 2022/04/10 12:09:47

// Design Name:

// Module Name: TestBench

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module TestBench();//迷茫

reg clk;

DataPath dp (clk);

initial begin

clk = 0;

end

always begin

#5;

clk = ~clk;

end

endmodule