Lab 5

CMPEN331

Tyler Rossi

7/30/24

a.

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 07/09/2024 09:31:39 PM

// Design Name:

// Module Name: MAIN

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module InstructionMemory(

input wire [31:0] PC,

output reg [31:0] instOut

);

// Creates memory with 64 different positions and 32 bits each

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

// Initializes intructions

initial begin

memory[25] = 32'h8c220000;

memory[26] = 32'h8c230004;

memory[27] = 32'h8c240008;

memory[28] = 32'h8c25000c;

memory[29] = 32'h00251820;

end

always @(\*) begin

instOut = memory[PC[7:2]];

end

endmodule

module ProgramCounter(

input wire [31:0] nextPC,

input wire clk,

output reg [31:0] PC

);

// Initializes PC to 100

initial begin

PC = 32'd100;

end

always @(posedge clk) begin

PC = nextPC;

end

endmodule

module PCadder(

input wire [31:0] PC,

output reg [31:0] nextPC

);

wire [31:0] four = 32'd4;

always @(\*) begin

nextPC = PC + four;

end

endmodule

module IFID(

input wire [31:0] instOut,

input wire clk,

output reg [31:0] dinstOut

);

always @(posedge clk) begin

dinstOut = instOut;

end

endmodule

module ControlUnit(

input wire [5:0] op,

input wire [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

// Default assignments

wreg = 1'b0;

m2reg = 1'b0;

wmem = 1'b0;

aluc = 4'b0001;

aluimm = 1'b0;

regrt = 1'b0;

case(op)

6'b000000: begin // R-Type

case(func)

6'b100000: begin // Add Instruction

aluc = 4'b0000;

wreg = 1'b1;

m2reg = 1'b0;

wmem = 1'b0;

aluimm = 1'b0;

regrt = 1'b0;

end

// 6'b100010: begin // Sub Instruction

// end

endcase

end

6'b100011: begin // LW

aluc = 4'b0010;

wreg = 1'b1;

m2reg = 1'b1;

wmem = 1'b0;

aluimm = 1'b1;

regrt = 1'b1;

end

endcase

end

endmodule

module RegrtMult(

input wire [4:0] rt,

input wire [4:0] rd,

input wire regrt,

output reg [4:0] destReg

);

always @(\*) begin

case(regrt)

1'b0: destReg = rd;

1'b1: destReg = rt;

endcase

end

endmodule

module RegisterFile(

input wire [4:0] rs,

input wire [4:0] rt,

input wire [4:0] wdestReg,

input wire [31:0] wbData,

input wire wwreg,

input wire clk,

output reg [31:0] qa,

output reg [31:0] qb

);

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

integer i;

// Initialize registers to zero

initial begin

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

registers[i] = 32'b0;

end

end

always @(\*) begin

qa = registers[rs];

qb = registers[rt];

end

always @(negedge clk) begin

case(wwreg)

1'b1:

begin

registers[wdestReg] = wbData;

end

endcase

end

endmodule

module ImmediateExtender(

input wire [15:0] imm,

output reg [31:0] imm32

);

wire [15:0] oneExt = 16'b1111111111111111;

wire [15:0] zeroExt = 16'b0000000000000000;

always @(\*) begin

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

end

endmodule

module IDEXE(

input wire wreg,

input wire m2reg,

input wire wmem,

input wire [3:0] aluc,

input wire aluimm,

input wire [4:0] destReg,

input wire [31:0] qa,

input wire [31:0] qb,

input wire [31:0] imm32,

input wire clk,

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 clk) begin

ewreg = wreg;

em2reg = m2reg;

ewmem = wmem;

ealuc = aluc;

ealuimm = aluimm;

edestReg = destReg;

eqa = qa;

eqb = qb;

eimm32 = imm32;

$display("imm: %b", eimm32);

end

endmodule

module AluMux(

input wire [31:0] eqb,

input wire [31:0] eimm32,

input wire ealuimm,

output reg [31:0] b

);

always @(\*)

begin

if(ealuimm == 1'b0)

begin

b = eqb;

end

if(ealuimm == 1'b1)

begin

b = eimm32;

end

end

endmodule

module Alu(

input wire [31:0] eqa,

input wire [31:0] b,

input wire [3:0] ealuc,

output reg [31:0] r

);

always @(\*)

begin

case(ealuc)

//add

4'b0010: begin

r = eqa + b;

end

//subtract

4'b0110: begin

r = eqa - b;

end

//and

4'b0000: begin

r = eqa & b;

end

//or

4'b0001: begin

r = eqa | b;

end

endcase

end

endmodule

module DataMemory(

input wire [31:0] mr,

input wire [31:0] mqb,

input wire mwmem,

input wire clk,

output reg [31:0] mdo

);

// Creates memory with 64 different positions and 32 bits each

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

// Initializes first 10 words into memory

initial begin

dmemory[0] = 32'hA00000AA;

dmemory[1] = 32'h10000011;

dmemory[2] = 32'h20000022;

dmemory[3] = 32'h30000033;

dmemory[4] = 32'h40000044;

dmemory[5] = 32'h50000055;

dmemory[6] = 32'h60000066;

dmemory[7] = 32'h70000077;

dmemory[8] = 32'h80000088;

dmemory[9] = 32'h90000099;

end

always @(\*) begin

mdo = dmemory[mr[7:2]];

end

always @(negedge clk)

begin

if(mwmem == 1'b1)

begin

dmemory[mr[7:2]] = mqb;

end

end

endmodule

module EXEMEM(

input wire ewreg,

input wire em2reg,

input wire ewmem,

input wire [4:0] edestReg,

input wire [31:0] r,

input wire [31:0] eqb,

input wire 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 MEMWB(

input wire mwreg,

input wire mm2reg,

input wire [4:0] mdestReg,

input wire [31:0] mr,

input wire [31:0] mdo,

input wire 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 WbMux(

input wire [31:0] wr,

input wire [31:0] wdo,

input wire wm2reg,

output reg [31:0] wbData

);

always @(\*)

begin

case(wm2reg)

1'b0: begin

wbData = wr;

end

1'b1: begin

wbData = wdo;

end

endcase

end

endmodule

module Datapath(

input wire clk,

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

);

// Wires for interconnections

wire [31:0] nextPc;

wire [31:0] instOut;

wire wreg, m2reg, wmem, aluimm, regrt;

wire [3:0] aluc;

wire [4:0] rs, rt, rd, destReg;

wire [31:0] qa, qb, imm32;

wire [5:0] op, func;

wire [15:0] imm;

wire [31:0] b, r, mdo, wbData;

// Program Counter

ProgramCounter pc\_dp (

.nextPC(nextPc),

.clk(clk),

.PC(pc)

);

// PC Adder

PCadder pcadder\_dp (

.PC(pc),

.nextPC(nextPc)

);

// Instruction Memory

InstructionMemory im\_dp (

.PC(pc),

.instOut(instOut)

);

// IF/ID Register

IFID ifidreg\_dp (

.instOut(instOut),

.clk(clk),

.dinstOut(dinstOut)

);

// Control Unit

assign op = dinstOut[31:26];

assign func = dinstOut[5:0];

ControlUnit cu\_dp (

.op(op),

.func(func),

.wreg(wreg),

.m2reg(m2reg),

.wmem(wmem),

.aluc(aluc),

.aluimm(aluimm),

.regrt(regrt)

);

// Register file inputs

assign rs = dinstOut[25:21];

assign rt = dinstOut[20:16];

assign rd = dinstOut[15:11];

// Register destination mux

RegrtMult regrtmux\_dp (

.rt(rt),

.rd(rd),

.regrt(regrt),

.destReg(destReg)

);

// Register File

RegisterFile rf\_dp (

.rs(rs),

.rt(rt),

.wdestReg(wdestReg),

.wbData(wbData),

.wwreg(wwreg),

.clk(clk),

.qa(qa),

.qb(qb)

);

// Immediate Extender

assign imm = dinstOut[15:0];

ImmediateExtender immex\_dp (

.imm(imm),

.imm32(imm32)

);

// ID/EXE Register

IDEXE idexereg\_dp (

.wreg(wreg),

.m2reg(m2reg),

.wmem(wmem),

.aluc(aluc),

.aluimm(aluimm),

.destReg(destReg),

.qa(qa),

.qb(qb),

.imm32(imm32),

.clk(clk),

.ewreg(ewreg),

.em2reg(em2reg),

.ewmem(ewmem),

.ealuc(ealuc),

.ealuimm(ealuimm),

.edestReg(edestReg),

.eqa(eqa),

.eqb(eqb),

.eimm32(eimm32)

);

// ALU Mux

AluMux alumux\_dp (

.eqb(eqb),

.eimm32(eimm32),

.ealuimm(ealuimm),

.b(b)

);

// ALU

Alu alu\_dp (

.eqa(eqa),

.b(b),

.ealuc(ealuc),

.r(r)

);

// EXE/MEM Register

EXEMEM exemem\_dp (

.ewreg(ewreg),

.em2reg(em2reg),

.ewmem(ewmem),

.edestReg(edestReg),

.r(r),

.eqb(eqb),

.clk(clk),

.mwreg(mwreg),

.mm2reg(mm2reg),

.mwmem(mwmem),

.mdestReg(mdestReg),

.mr(mr),

.mqb(mqb)

);

// Data Memory

DataMemory dm\_dp (

.mr(mr),

.mqb(mqb),

.mwmem(mwmem),

.clk(clk),

.mdo(mdo)

);

// MEM/WB Register

MEMWB memwb\_dp (

.mwreg(mwreg),

.mm2reg(mm2reg),

.mdestReg(mdestReg),

.mr(mr),

.mdo(mdo),

.clk(clk),

.wwreg(wwreg),

.wm2reg(wm2reg),

.wdestReg(wdestReg),

.wr(wr),

.wdo(wdo)

);

// WB Mux

WbMux wb\_mux\_dp (

.wr(wr),

.wdo(wdo),

.wm2reg(wm2reg),

.wbData(wbData)

);

Endmodule

b.

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 07/11/2024 03:11:37 PM

// 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\_tb;

initial clk\_tb = 1'b0;

wire [31:0] pc, dinstOut;

wire ewreg, em2reg, ewmem, ealuimm;

wire [3:0] ealuc;

wire [4:0] edestReg;

wire [31:0] eqa, eqb, eimm32;

wire mwreg, mm2reg, mwmem;

wire [4:0] mdestReg;

wire [31:0] mr, mqb;

wire wwreg, wm2reg;

wire [4:0] wdestReg;

wire [31:0] wr, wdo;

Datapath dp\_tb(

.clk(clk\_tb),

.pc(pc),

.dinstOut(dinstOut),

.ewreg(ewreg),

.em2reg(em2reg),

.ewmem(ewmem),

.ealuc(ealuc),

.ealuimm(ealuimm),

.edestReg(edestReg),

.eqa(eqa),

.eqb(eqb),

.eimm32(eimm32),

.mwreg(mwreg),

.mm2reg(mm2reg),

.mwmem(mwmem),

.mdestReg(mdestReg),

.mr(mr),

.mqb(mqb),

.wwreg(wwreg),

.wm2reg(wm2reg),

.wdestReg(wdestReg),

.wr(wr),

.wdo(wdo)

);

always

begin

#5 clk\_tb = ~clk\_tb;

end

endmodule

c.

A screenshot of a computer program

Description automatically generated

d.

A green and black line drawing

Description automatically generatedA computer diagram of a circuit board

Description automatically generated with medium confidence

e.

A screen shot of a computer screen

Description automatically generated