//-------------------------------------------------------

// mipsmulti.v

// Last Modified by: Hadil Mustafa 3/28/15

//

// Multicycle MIPS processor

//------------------------------------------------

module mips(input clk, reset,

output [31:0] adr, writedata,

output memwrite,

input [31:0] readdata);

wire zero, pcen, irwrite, regwrite,

alusrca, iord, memtoreg, regdst;

wire [1:0] alusrcb, pcsrc;

wire [2:0] alucontrol;

wire [5:0] op, funct;

controller c(clk, reset, op, funct, zero,

pcen, memwrite, irwrite, regwrite,

alusrca, iord, memtoreg, regdst,

alusrcb, pcsrc, alucontrol);

datapath dp(clk, reset,

pcen, irwrite, regwrite,

alusrca, iord, memtoreg, regdst,

alusrcb, pcsrc, alucontrol,

op, funct, zero,

adr, writedata, readdata);

endmodule

module controller(input clk, reset,

input [5:0] op, funct,

input zero,

output pcen, memwrite, irwrite, regwrite,

output alusrca, iord, memtoreg, regdst,

output [1:0] alusrcb, pcsrc,

output [2:0] alucontrol);

wire [1:0] aluop;

wire branch, pcwrite;

// Main Decoder and ALU Decoder subunits.

maindec md(clk, reset, op,

pcwrite, memwrite, irwrite, regwrite,

alusrca, branch, iord, memtoreg, regdst,

alusrcb, pcsrc, aluop);

aludec ad(funct, aluop, alucontrol);

// ADD CODE HERE

// Add combinational logic (i.e. an assign statement)

// to produce the PCEn signal (pcen) from the branch,

// zero, and pcwrite signals

endmodule

module maindec(input clk, reset,

input [5:0] op,

output pcwrite, memwrite, irwrite, regwrite,

output alusrca, branch, iord, memtoreg, regdst,

output [1:0] alusrcb, pcsrc,

output [1:0] aluop);

parameter FETCH = 4'b0000; // State 0

parameter DECODE = 4'b0001; // State 1

parameter MEMADR = 4'b0010; // State 2

parameter MEMRD = 4'b0011; // State 3

parameter MEMWB = 4'b0100; // State 4

parameter MEMWR = 4'b0101; // State 5

parameter RTYPEEX = 4'b0110; // State 6

parameter RTYPEWB = 4'b0111; // State 7

parameter BEQEX = 4'b1000; // State 8

parameter ADDIEX = 4'b1001; // State 9

parameter ADDIWB = 4'b1010; // state 10

parameter JEX = 4'b1011; // State 11

parameter LW = 6'b100011; // Opcode for lw

parameter SW = 6'b101011; // Opcode for sw

parameter RTYPE = 6'b000000; // Opcode for R-type

parameter BEQ = 6'b000100; // Opcode for beq

parameter ADDI = 6'b001000; // Opcode for addi

parameter J = 6'b000010; // Opcode for j

reg [3:0] state, nextstate;

reg [14:0] controls;

// state register

always @(posedge clk or posedge reset)

if(reset) state <= FETCH;

else state <= nextstate;

// ADD CODE HERE

// Finish entering the next state logic below.

// the first two states, FETCH and DECODE, are completed for you.

// next state logic

Always @ (\*)

case(state)

FETCH: nextstate <= DECODE;

DECODE: case(op)

LW: nextstate <= MEMADR;

SW: nextstate <= MEMADR;

RTYPE: nextstate <= RTYPEEX;

BEQ: nextstate <= BEQEX;

ADDI: nextstate <= ADDIEX;

J: nextstate <= JEX;

default: nextstate <= 4'bx; // should never happen

endcase

// Add code here

MEMADR:

MEMRD:

MEMWB:

MEMWR:

RTYPEEX:

RTYPEWB:

BEQEX:

ADDIEX:

ADDIWB:

JEX:

default: nextstate <= 4'bx; // should never happen

endcase

// output logic

assign {pcwrite, memwrite, irwrite, regwrite,

alusrca, branch, iord, memtoreg, regdst,

alusrcb, pcsrc, aluop} = controls;

// ADD CODE HERE

// Finish entering the output logic below. We've entered the

// output logic for the first two states, S0 and S1, for you.

Always @(\*)

case(state)

FETCH: controls <= 15'h5010;

DECODE: controls <= 15'h0030;

// your code goes here

default: controls <= 15'hxxxx; // should never happen

endcase

endmodule

module aludec(input [5:0] funct,

input [1:0] aluop,

output [2:0] alucontrol);

// ADD CODE HERE

// Complete the design for the ALU Decoder.

// Your design goes here. Remember that this is a combinational

// module.

// Remember that you may also reuse any code from previous labs.

endmodule

**// Complete the datapath module below for Lab 7.**

**// You do not need to complete this module for Lab 6**

// The datapath unit is a structural verilog module. That is,

// it is composed of instances of its sub-modules. For example,

// the instruction register is instantiated as a 32-bit flopenr.

// The other submodules are likewise instantiated.

// The datapath module is for lab 7.

module datapath(input clk, reset,

input pcen, irwrite, regwrite,

input alusrca, iord, memtoreg, regdst,

input [1:0] alusrcb, pcsrc,

input [2:0] alucontrol,

output [5:0] op, funct,

output zero,

output [31:0] adr, writedata,

input [31:0] readdata);

// Below are the internal signals of the datapath module.

wire [4:0] writereg;

wire [31:0] pcnext, pc;

wire [31:0] instr, data, srca, srcb;

wire [31:0] a;

wire [31:0] aluresult, aluout;

wire [31:0] signimm; // the sign-extended immediate

wire [31:0] signimmsh; // the sign-extended immediate shifted left by 2

wire [31:0] wd3, rd1, rd2;

// op and funct fields to controller

assign op = instr[31:26];

assign funct = instr[5:0];

// Your datapath hardware goes below. Instantiate each of the submodules

// that you need. Remember that alu's, mux's and various other

// versions of parameterizable modules are available in mipsparts.v

// from Lab 5. You'll likely want to include this verilog file in your

// simulation.

// I’ve included parameterizable 3:1 and 4:1 muxes below for your use.

// Remember to give your instantiated modules applicable names

// such as pcreg (PC register), wdmux (Write Data Mux), etc.

// so it's easier to understand.

// ADD CODE HERE

// datapath

endmodule

module mux3 #(parameter WIDTH = 8)

(input [WIDTH-1:0] d0, d1, d2,

input [1:0] s,

output [WIDTH-1:0] y);

assign #1 y = s[1] ? d2 : (s[0] ? d1 : d0);

endmodule

module mux4 #(parameter WIDTH = 8)

(input [WIDTH-1:0] d0, d1, d2, d3,

input [1:0] s,

output [WIDTH-1:0] y);

always\_comb

case(s)

2'b00: y <= d0;

2'b01: y <= d1;

2'b10: y <= d2;

2'b11: y <= d3;

endcase

endmodule