module testbench ();

reg clk;

reg reset;

wire [7:0] inext,dataoutf,ramin,IR;

wire [5:0] ramaddress;

wire memWr,RFwr;

reg [1:0]succuss;

top cpusim(clk,reset,inext,dataoutf,ramin,IR,ramaddress,memWr,RFwr);

// initialize test

initial

begin

succuss <= 2'b00;

reset <= 1; # 22; reset <= 0;

end

always

begin

clk <= 1;

# 5;

clk <= 0;

# 5; // clock duration

end

// check results

always @ (negedge RFwr)

begin

if (IR[2:0] == 3'b000&&ramin == 8'b00000001)

begin

succuss <= succuss + 1'b1;

$display ("R[",IR[2:0],"] = ", ramin);

end

end

always @ (negedge memWr)

begin

if (ramaddress == 6'b001111&&ramin == 8'b00000001)

begin

succuss <= succuss + 1'b1;

$display ("RAM[",ramaddress,"] = ", ramin);

end

else if (ramaddress == 6'b010000&&ramin == 8'b00000010)

begin

succuss <= succuss + 1'b1;

$display ("RAM[",ramaddress,"] = ", ramin);

end

end

always @(\*)

if (succuss == 2'b11)

begin

$display ("Simulation succeeded");

#50

$stop;

end

endmodule

module top(

input clk,

input reset,

input [7:0] inext,

output [7:0] dataoutf,ramin,IR,

output [5:0] ramaddress,

output memWr,RFwr

);

wire [7:0] dataout;//IR,

wire Aeq0,apos,IRload,PCload,MemInst,MRload,Aload,reset1,outen;//,RFwr,memWr

wire [1:0] JMPmux,Shftsel;

wire [2:0] Asel,ALUsel;

assign dataoutf = dataout;

controler conunit(clk,IR,Aeq0,apos,dataout,IRload,JMPmux,PCload,MemInst,

MRload,memWr,Asel,Aload,reset1,RFwr,ALUsel,Shftsel,outen);

datapath datunit(clk,IRload,JMPmux,PCload,MemInst,MRload,memWr,Asel,Aload,

reset,RFwr,ALUsel,Shftsel,outen,inext,IR,Aeq0,apos,dataout,ramin,ramaddress);

endmodule

module datapath (

input clk,

input IRload,

input [1:0] JMPmux,

input PCload,MemInst,MRload,memWr,

input [2:0] Asel,

input Aload,reset,RFwr,

input [2:0]ALUsel,

input [1:0] Shftsel,

input outen,

input [7:0] inext,

output [7:0] IR,

output Aeq0,apos,

output [7:0]dataout, //output reg thing);

output [7:0] ramin,

output [5:0] ramaddress

);

wire [7:0] memout,irout,MR2ram,Accin,Accout,shftout,rfout,aluout,regout;

wire [5:0] PCplus1,relnout,relpout,PCout,PCin,RAMaddr;

IRreg insreg(memout,IRload,reset,clk,irout);

IRreg MAreg(memout,MRload,reset,clk,MR2ram);

reln ad(PCout,irout [2:0],relnout);

relp su(PCout,irout [2:0],relpout);

incer plus1(PCout,PCplus1);

mux4 pcmuxin(PCplus1, memout[5:0], relnout,relpout,JMPmux,PCin );

pcreg pc(PCin,PCload,reset,clk,PCout);

pcmux2 pcmuxout(PCout,MR2ram[5:0],MemInst,RAMaddr);

RAM datainst(Accout,RAMaddr,memWr,clk,memout);

regifile regies(Accout,RFwr,irout[2:0],clk,rfout);

mux5 accmux(shftout,rfout,inext,memout,memout,Asel,Accin); //picture shows two inputs from memout

accumulator accu(Accin,Aload,reset,clk,Accout);

assign Aeq0 = ~|Accout;

assign apos = ~ Accout [7];

assign IR = irout;

assign dataout = regout;

assign ramaddress = RAMaddr;

assign ramin = Accout;

alu alumod(Accout,rfout,ALUsel,aluout);

shifter shiftmod(aluout,Shftsel,shftout);

outreg outer(Accout,outen,reset,clk,regout);

endmodule

module controler (

input clk,

input [7:0] IR,

input Aeq0,apos,

input [7:0]dataout,

output IRload,

output [1:0] JMPmux,

output PCload,MemInst,MRload,memWr,

output [2:0] Asel,

output Aload,reset,RFwr,

output [2:0]ALUsel,

output [1:0] Shftsel,

output outen

);

reg [18:0] controls;

assign {IRload,JMPmux,PCload,MemInst,MRload,memWr,Asel,Aload,reset,RFwr,ALUsel,Shftsel,outen} = controls;

reg resetcount,stop;

wire [1:0] cycle;

initial

begin

stop <=0;

resetcount <=0;

end

count2b counter(clk,resetcount,stop,cycle);

always @ (\*)

begin

case (cycle)

2'b00 :

begin

controls <= 19'b1001000xxx000xxxxx0;

resetcount <=0;

end

2'b01 : //2 byte instructions must update pc

begin

casex (IR)

8'b00010xxx: //LDA

begin

controls <= 19'b0xx0000001100xxxxx0;

resetcount <=1;

end

8'b00100xxx: //STA

begin

controls <= 19'b0xx0000xxx001xxxxx0;

resetcount <=1;

end

8'b00110000: //LDM0

begin

controls <= 19'b0xx0010xxx000xxxxx0;

resetcount <=0;

end

8'b01000000: //STM0

begin

controls <= 19'b0xx0010xxx000xxxxx0;

resetcount <=0;

end

8'b01010000: //LDI

begin

controls <= 19'b0001000100100xxxxx0;

resetcount <=1;

end

// uses alu

8'b10100xxx: //AND

begin

controls <= 19'b0xx0000000100001000;

resetcount <=1;

end

8'b10110xxx: //OR

begin

controls <= 19'b0xx0000000100010000;

resetcount <=1;

end

8'b11000xxx: //ADD

begin

controls <= 19'b0xx0000000100011000;

resetcount <=1;

end

8'b11010xxx: //SUB

begin

controls <= 19'b0xx0000000100100000;

resetcount <=1;

end

8'b11100000: //NOT

begin

controls <= 19'b0xx0000000100101000;

resetcount <=1;

end

8'b11100001: //INC

begin

controls <= 19'b0xx0000000100110000;

resetcount <=1;

end

8'b11100010: //DEC

begin

controls <= 19'b0xx0000000100111000;

resetcount <=1;

end

8'b11100011: //SHFL

begin

controls <= 19'b0xx0000000100000010;

resetcount <=1;

end

8'b11100100: //SHFR

begin

controls <= 19'b0xx0000000100000100;

resetcount <=1;

end

8'b11100101: //ROTR

begin

controls <= 19'b0xx0000000100000110;

resetcount <=1;

end

//j instructions

8'b01100000: //JMP

begin

controls <= 19'b0011000xxx000xxxxx0;

resetcount <=1;

end

8'b0110xxxx: //JMPR

begin

if (IR[3])

controls <= 19'b0101000xxx000xxxxx0;

else

begin

controls <= 19'b0111000xxx000xxxxx0;

resetcount <=1;

end

end

8'b01110000: //JZ

begin

if (Aeq0)

begin

controls <= 19'b0011000xxx000xxxxx0;

resetcount <=1;

end

else

begin

controls <= 19'b0001000xxx000xxxxx0;

resetcount <=1;

end

end

8'b0111xxxx: //JZR

begin

if (Aeq0)

begin

if (IR[3])

controls <= 19'b0101000xxx000xxxxx0;

else

begin

controls <= 19'b0111000xxx000xxxxx0;

resetcount <=1;

end

end

else

begin

controls <= 19'b0001000xxx000xxxxx0;

resetcount <=1;

end

end

8'b10000000: //JNZ

begin

if (!Aeq0)

begin

controls <= 19'b0011000xxx000xxxxx0;

resetcount <=1;

end

else

begin

controls <= 19'b0001000xxx000xxxxx0;

resetcount <=1;

end

end

8'b1000xxxx: //JNZR

begin

if (!Aeq0)

begin

if (IR[3])

controls <= 19'b0101000xxx000xxxxx0;

else

begin

controls <= 19'b0111000xxx000xxxxx0;

resetcount <=1;

end

end

else

begin

controls <= 19'b0001000xxx000xxxxx0;

resetcount <=1;

end

end

8'b10010000: //JP

begin

if (apos)

begin

controls <= 19'b0011000xxx000xxxxx0;

resetcount <=1;

end

else

begin

controls <= 19'b0001000xxx000xxxxx0;

resetcount <=1;

end

end

8'b1001xxxx: //JPR

begin

if (apos)

begin

controls <= 19'b0xx0000000100000110;

resetcount <=1;

end

else

begin

controls <= 19'b0001000xxx000xxxxx0;

resetcount <=1;

end

end

//done so far

//in/out

8'b11110000: //IN

begin

controls <= 19'b0xx0000010100xxxxx0;

resetcount <=1;

end

8'b11110001: //OUT

begin

controls <= 19'b0xx0000xxx000xxxxx1;

resetcount <=1;

end

8'b11110010: //HALT

begin

controls <= 19'b0xx0000xxx010xxxxx0;

resetcount <=1;

stop <=1;

end

8'b00000000: //NOP

begin

controls <= 19'b0xx0000xxx000xxxxx0;

resetcount <=1;

end

//default: not needed

//begin

//controls <= 19'b1001000xxx000xxxxx0;

//resetcount <=1;

// end

endcase

end

2'b10:

begin

case (IR)

8'b00110000: //LDM1

begin

controls <= 19'b0001100011100xxxxx0;

resetcount <=1;

end

8'b01000000: //STM1

begin

controls <= 19'b0001101xxx000xxxxx0;

resetcount <=1;

end

endcase

end

endcase

end

endmodule

module accumulator (D,Wr,clear,clk,q);

input [7:0] D;

input Wr,clear,clk;

output reg [7:0] q;

always @ (posedge clk , posedge clear)

begin

if (clear)

q <= 0;

else if(Wr)

q<=D;

else

q<=q;

end

endmodule

module alu (a,b,sel,y);

input [7:0] a,b;

input [2:0] sel;

output reg [7:0] y;

always @ (\*)

begin

case(sel)

3'b000:

begin

y=a; //pass though

end

3'b001:

begin

y= a & b;

end

3'b010:

begin

y= a | b;

end

3'b011:

begin

y= a + b;

end

3'b100:

begin

y= a - b;

end

3'b101:

begin

y= ~a; // bitwise invert

end

3'b110:

begin

y = a + 1'b1; // inc

end

3'b111:

begin

y = a - 1'b1; // dec

end

endcase

end

endmodule

module count2b (clk,reset,stop,cycle);

input clk,reset,stop;

output reg [1:0]cycle;

initial

cycle <=00;

always @ (posedge clk)

begin

if (!stop)

begin

if (reset)

cycle <= 2'b00;

else

cycle <= cycle + 2'b01;

end

end

endmodule

module incer (in,out);

input [5:0] in;

output [5:0] out;

assign out = in + 1'b1;

endmodule

module IRreg(D,load,clear,clk,Out);

input [7:0] D;

input load,clear,clk;

output reg [7:0] Out;

initial

Out <= 8'b00000000;

always @ (posedge clk, posedge clear)

begin

if (clear)

Out <= 8'b00000000;

else if(load)

Out <= D;

else

Out <= Out;

end

endmodule

module mux4 (a,b,c,d,sel,y);

input [1:0] sel;

input [5:0] a,b,c,d;

output reg [5:0] y;

always @ (\*)

begin

case (sel)

2'b00: begin

y=a;

end

2'b01: begin

y=b;

end

2'b10: begin

y=c;

end

2'b11: begin

y=d;

end

endcase

end

endmodule

module mux5 (a,b,c,d,e,sel,y);

input [2:0] sel;

input [7:0] a,b,c,d,e;

output reg [7:0] y;

always @ (\*)

begin

case (sel)

3'b000: begin

y<=a;

end

3'b001: begin

y<=b;

end

3'b010: begin

y<=c;

end

3'b011: begin

y<=d;

end

3'b100: begin

y<=e;

end

endcase

end

endmodule

module outreg ( D,Wr,clear,clk,qr);

input [7:0] D;

input Wr,clear,clk;

output reg [7:0] qr;

reg [7:0] q;

always @ (posedge clk , posedge clear) begin

if (clear) q = 8'b00000000;

else q<=D;

end

always @ (\*) begin

if (Wr)

qr = q;

else

qr = 8'bzzzzzzzz;

end

endmodule

module pcmux2 (a,b,sel,out);

input [5:0] a,b;

input sel;

output reg [5:0] out;

always @ (\*)

begin

case (sel)

1'b0: begin

out=a;

end

1'b1: begin

out=b;

end

endcase

end

endmodule

module pcreg (D,Load,Clear,clk,out);

input [5:0] D;

input Load,Clear,clk;

output reg [5:0] out;

initial

out <= 6'b0;

always @ (posedge clk , posedge Clear) begin

if (Clear) out <= 0;

else if(Load) out <=D;

else out<=out;

end

endmodule

module RAM (D,addr,WE,clk,Q); //instructions stored here

input [7:0] D;

input [5:0] addr;

input WE,clk;

output [7:0] Q;

reg [7:0] MEM [63:0];

initial

begin

$readmemb ("memfile.dat",MEM);

end

assign Q = MEM [addr];

always @ (negedge clk)

if (WE)

MEM [addr] <= D;

endmodule

module regifile (D,WE,WA,clk,out);

input [7:0] D;

input WE,clk;

input [2:0] WA;

output [7:0] out;

reg [7:0] regmem [7:0];

assign out = regmem [WA];

always @ (posedge clk)

if (WE)

regmem[WA] <= D;

endmodule

module reln(a,b,y);

input [5:0] a;

input [2:0] b;

output [5:0] y;

assign y = a - b;

endmodule

module relp(a,b,y);

input [5:0] a;

input [2:0] b;

output [5:0] y;

assign y = a + b;

endmodule

module shifter (in,sel,out);

input [7:0] in;

input [1:0] sel;

output reg [7:0] out;

always @ (\*) begin

case (sel)

2'b00: //pass though

begin

out = in;

end

2'b01: //SHFL

begin

out = in << 1'b1;

end

2'b10: //SHFR

begin

out = in >> 1'b1;

end

2'b11: //Rotr

begin

out = {in[0], in[7:1]};

end

endcase

end

endmodule