module tb\_avion\_cpu;

parameter TEST\_CASE = 2;

parameter ADDRESS\_WIDTH = 6;

parameter DATA\_WIDTH = 10;

reg clk;

reg rst;

initial begin

clk =1;

end

wire [ADDRESS\_WIDTH-1:0] addr\_toRAM;

wire [DATA\_WIDTH-1:0] data\_toRAM, data\_fromRAM;

wire [ADDRESS\_WIDTH-1:0] pCounter;

wire wrEn;

always clk = #5 !clk;

reg error;

integer i;

initial begin

rst = 1;

error = 0;

#100;

rst <= #1 0;

#5000;

if(TEST\_CASE == 1)

memCheck(52,15);

else if(TEST\_CASE == 2)

memCheck(52,50);

else if(TEST\_CASE == 3)

memCheck(52,50);

#500;

end

avion\_cpu #(

ADDRESS\_WIDTH,

DATA\_WIDTH

) avion\_cpu\_Inst(

.clk(clk),

.rst(rst),

.MDRIn(data\_toRAM),

.RAMWr(wrEn),

.MAR(addr\_toRAM),

.MDROut(data\_fromRAM),

.PC(pCounter)

);

blram #(ADDRESS\_WIDTH, 64, TEST\_CASE) blram(

.clk(clk),

.rst(rst),

.i\_we(wrEn),

.i\_addr(addr\_toRAM),

.i\_ram\_data\_in(data\_toRAM),

.o\_ram\_data\_out(data\_fromRAM)

);

task memCheck;

input [31:0] memLocation, expectedValue;

begin

if(blram.memory[memLocation] != expectedValue[9:0]) begin

error = 1;

end

end

endtask

endmodule

module blram(clk, rst, i\_we, i\_addr, i\_ram\_data\_in, o\_ram\_data\_out);

parameter SIZE = 6;

parameter DEPTH = 64;

parameter TEST\_CASE = 1;

input clk;

input rst;

input i\_we;

input [SIZE-1:0] i\_addr;

input [9:0] i\_ram\_data\_in;

output reg [9:0] o\_ram\_data\_out;

reg [9:0] memory[0:DEPTH-1];

always @(posedge clk) begin

o\_ram\_data\_out <= #1 memory[i\_addr[SIZE-1:0]];

if (i\_we)

memory[i\_addr[SIZE-1:0]] <= #1 i\_ram\_data\_in;

end

initial begin

if(TEST\_CASE == 1) begin

memory[0] = 10'b0000110010; // LOD 50, (ACC = \*50), Hex = 32

memory[1] = 10'b0010110011; // ADD 51, ACC = ACC + (\*51), Hex = B3

memory[2] = 10'b0001110100; // STO 52, (\*52) = ACC, Hex = 74

memory[3] = 10'b1001000000; // Halt, Hex = 240

memory[50] = 10'b0000000101; // Hex = 5

memory[51] = 10'b0000001010; // Hex = A

end else if(TEST\_CASE == 2) begin

memory[0] = 10'b0000110010; // LOD 50, (ACC = \*50), Hex = 32

memory[1] = 10'b0100110011; // MUL 51, ACC = ACC \* (\*51), Hex = 133

memory[2] = 10'b0001110100; // STO 52, (\*52) = ACC, Hex = 74

memory[3] = 10'b1001000000; // Halt, Hex = 240

memory[50] = 10'b0000000101; // Hex = 5

memory[51] = 10'b0000001010; // Hex = A

end else if(TEST\_CASE == 3) begin

memory[0]= 10'b0000110011; // LOD 51, ACC = \*51, Hex = 33

memory[1]= 10'b0011110001; // SUB 49, ACC = ACC - \*49, Hex = F1

memory[2]= 10'b0111001010; // JMZ 10,

memory[3]= 10'b0000110000; // LOD 48,

memory[4]= 10'b0010110010; // ADD 50,

memory[5]= 10'b0001110000; // STO 48,

memory[6]= 10'b0000110001; // LOD 49, ACC = i, Hex = 31

memory[7]= 10'b0010101110; // ADD 46, ACC = i + 1, Hex = AE

memory[8]= 10'b0001110001; // STO 49, i = i + 1, Hex = 71

memory[9]= 10'b0110000000; // JMP 0,180

memory[10]= 10'b0000110000; // LOD 48, ACC = temp, Hex = 30

memory[11]= 10'b0001110100; // STO 52, \*52 = ACC, Hex = 74

memory[12]= 10'b1001000000;// HLT

memory[46]= 10'b1; // 1 number

memory[48]= 10'b0; // Hex = 0, temp

memory[49]= 10'b0; // Hex = 0,

memory[50]= 10'b0000000101; // Hex = 5

memory[51]= 10'b0000001010; // Hex = A

end

end

endmodule

module avion\_cpu #(

parameter ADDRESS\_WIDTH = 6,

parameter DATA\_WIDTH = 10

)(

input clk,

input rst,

output reg [DATA\_WIDTH-1:0] MDRIn,

output reg RAMWr,

output reg [ADDRESS\_WIDTH-1:0] MAR,

input [DATA\_WIDTH-1:0] MDROut,

output reg [5:0] PC

);

reg [DATA\_WIDTH - 1:0] IR, IRNext;

reg [5:0] PCNext;

reg [9:0] ACC, ACCNext;

reg [2:0] state, stateNext;

always@(posedge clk) begin

state <= #1 stateNext;

PC <= #1 PCNext;

IR <= #1 IRNext;

ACC <= #1 ACCNext;

end

always@(\*) begin

stateNext = state;

PCNext = PC;

IRNext = IR;

ACCNext = ACC;

MAR = 0;

RAMWr = 0;

MDRIn = 0;

if(rst) begin

stateNext = 0;

PCNext = 0;

MAR = 0;

RAMWr = 0;

IRNext = 0;

ACCNext = 0;

MDRIn = 0;

end else begin

case(state)

0: begin

MAR = PC;

RAMWr = 0;

stateNext = state + 1;

end

1: begin

IRNext = MDROut;

PCNext = PC + 1;

stateNext = state + 1;

end

2: begin

if(IR[9:6] < 6) begin

//elimdeki komutun en büyük 4 biti yani operasyon kodumun bulunduğu 4 bit 6'dan küçükse

//yani yapacağım iş bellekten sayı okumayı gerektiriyorsa

MAR = IR[5:0]; //adresin kendisini ramden okumaya başladım çünkü bu komutların bellekten alınan veriyi kullanacağını biliyorum

stateNext = 3; //durum 3'e atladım

end else if(IR[9:6] == 6) begin // 6 jump komutu

stateNext = 0;

PCNext = IR[5:0]; //program counter'a 6 biti atadım

end else if(IR[9:6] == 7) begin

stateNext = 0;

if(ACC == 0) begin

PCNext = IR[5:0];

end

end else if(IR[9:6] == 8) begin // nop, no operation

stateNext = 0;

end else if(IR[9:6] == 9) begin // halt (boş) operasyon

stateNext = 4;

end

end

3: begin

stateNext = 0;

RAMWr = 0;

MAR = 0;

if(IR[9:6] == 0) begin

ACCNext = MDROut;

end else if(IR[9:6] == 1) begin

MAR = IR[5:0];

RAMWr =1;

MDRIn = ACC;

end else if(IR[9:6] == 2) begin

ACCNext = MDROut + ACC;

end else if(IR[9:6] == 3) begin

ACCNext = ACC - MDROut;

end else if(IR[9:6] == 4) begin

ACCNext = ACC \* MDROut;

end else if(IR[9:6] == 5) begin

ACCNext = ACC / MDROut;

end

end

4: begin

// state 4 scope is null

end

endcase

end

end

endmodule

module top (

input clk, // 10Hz

input rst,

input [15:0] switches,

input btnU,

input btnD,

input btnL,

input btnR,

input btnM,

output reg [15:0] leds,

output reg [7:0] ss3, ss2, ss1, ss0,

output reg red, green, blue

);

tb\_avion\_cpu tb\_avion\_cpuInst();

endmodule