`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 22:41:45 05/18/2018

// Design Name:

// Module Name: abcd

// Project Name:

// Target Devices:

// Tool versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module processor(output[7:0] instruction,input clk ,input rst,input inter,output zflag,output overflow);

wire load\_r0,load\_r1,load\_r2,load ,load\_r3,load\_reg\_y,load\_reg\_z,load\_pc,inc\_pc,write,load\_ir,

load\_add\_reg,read,load\_sp,inc\_sp,dec\_sp;

wire[2:0] sel\_bus\_mux1;

wire[1:0] sel\_bus\_mux2;

wire slow\_clk;

//wire zero,overflow;

wire [7:0] addraddress,bus1,wmem;

slow\_clk slowclk(clk,rst,slow\_clk);

controlunit cu(.load\_r0(load\_r0),.load\_r1(load\_r1),.load\_r2(load\_r2),.load\_r3(load\_r3)

,.load\_pc(load\_pc),.inc\_pc(inc\_pc),.sel\_bus1(sel\_bus\_mux1),.load\_ir(load\_ir),.load\_addr(load\_add\_reg),

.load\_regy(load\_reg\_y),.load\_regz(load\_reg\_z),.sel\_bus2(sel\_bus\_mux2)

,.clk(slow\_clk),.rst(rst),.alu\_zflag(zflag),.alu\_overflow(overflow),.instruction(instruction),.write(write)

,.read(read),.load\_sp(load\_sp),.inc\_sp(inc\_sp),.dec\_sp(dec\_sp),.interrupt(inter));

memory mem(.address(addraddress),.read(read),.write(write),.data\_in(bus1),.out(wmem),.rst(rst),.clk(slow\_clk));

abcd datapath(.load0(load\_r0),.load1(load\_r1),.load2(load\_r2),.load3(load\_r3),.loadpc(load\_pc)

,.inc(inc\_pc),.sel1(sel\_bus\_mux1),.loadIR(load\_ir),.loadaddr(load\_add\_reg),.loady(load\_reg\_y)

,.load\_reg\_z(load\_reg\_z),.sel2(sel\_bus\_mux2),.clk(slow\_clk),.rst(rst),.mainout({zflag,overflow}),.wIR(instruction)

,.addraddress(addraddress),.wmem(wmem),.wbus1(bus1),.load\_sp(load\_sp),.inc\_sp(inc\_sp)

,.dec\_sp(dec\_sp));

endmodule

module register( input load,input rst,input clk,input [7:0]inp1,output reg [7:0]out);

always@(posedge clk or posedge rst)

begin

if(rst)

out<=8'd0;

else if(load==1'b1)

out<=inp1;

else

out<=out;

end

endmodule

module bus1\_mux(input [7:0]inp0,input [7:0]inp1,input [7:0]inp2,input [7:0]inp3,input [7:0]inp4

,input [7:0]inp5,output reg [7:0]out,input [2:0]sel);

always@(\*)

begin

if(sel==3'b000)

out<=inp0;

else if(sel==3'b001)

out<=inp1;

else if(sel==3'b010)

out<=inp2;

else if(sel==3'b011)

out<=inp3;

else if(sel==3'b100)

out<=inp4;

else if(sel==3'b101)

out<=inp5;

else

out<= 0;

end

endmodule

module bus2\_mux(input [7:0] inp0,input [7:0] inp1,input [7:0] inp2,output reg [7:0] out, input [1:0]sel);

always@(\*)

begin

if(sel==2'b00)

out<=inp0;

else if(sel==2'b01)

out<=inp1;

else if(sel==2'b10)

out<=inp2;

else

out<=0;

end

endmodule

module pc(input load,input [7:0]inp1,output reg[7:0]out,input inc,input rst,input clk);

always@(posedge clk or posedge rst)

begin

if(rst)

out<=8'd0;

else if(inc)

out<=out+1'b1;

else if(load==1'b1)

out<=inp1;

else

out<=out;

end

endmodule

module alu(input [7:0] inp1,input [7:0] inp2,input [2:0]opcode,output reg[7:0] out,

output reg overflow,output reg alu\_zero\_flag);

always@(\*)

begin

if(opcode==3'b000)

{overflow ,out}<=inp1+inp2;

else if(opcode==3'b001)

{overflow,out}<=inp1-inp2;

else if(opcode==3'b010)

{overflow,out}<=inp1\*inp2;

// else if(opcode==3'b011)

//{overflow,out}<=inp1/inp2;

else if(opcode==3'b100)

{overflow ,out}<=inp1&inp2;

else if(opcode==3'b101)

{overflow ,out}<=inp1|inp2;

else

{overflow ,out}<=0;

end

always@(\*)

begin

if(out==8'd0)

alu\_zero\_flag<=1;

else

alu\_zero\_flag<=0;

end

endmodule

module memory(input [7:0] address,input [7:0] data\_in,input read,

input write,output reg[7:0] out,input clk, input rst);

integer i;

reg [7:0] mem [255:0];

always@(posedge clk or posedge rst)

begin

if(rst)

begin

for(i=0;i<=255;i=i+1)

mem[i]<=8'd0;

end

else if(read)

out<=mem[address];

else if(write)

mem[address] <= data\_in;

else

out<= out;

end

endmodule

module regz( input rst,input [1:0] inp1,input clk,input load,output reg[1:0] out);

always@(posedge clk or posedge rst)

begin

if(rst)

out<=2'b0;

else if (load)

out<=inp1;

else

out<=out;

end

endmodule

module controlunit(output reg load\_r0,output reg load\_r1,output reg load\_r2,output reg load\_r3

,output reg load\_pc,output reg inc\_pc,output reg [2:0] sel\_bus1,output reg load\_ir,output reg load\_addr

,output reg load\_regy,output reg load\_regz,output reg [1:0]sel\_bus2,output reg write,output reg read

,input clk,input rst,input alu\_zflag,input alu\_overflow,input [7:0] instruction,output reg load\_sp

,output reg inc\_sp,output reg dec\_sp,input interrupt);

reg[3:0] p\_state;

reg[3:0] n\_state;

parameter fetch1\_alu=4'd0;

parameter fetch2\_alu=4'd1;

parameter decode=4'd2;

parameter execute1\_alu=4'd3;

parameter execute2\_alu=4'd4;

parameter execute3\_flag=4'd5;

parameter write\_back=4'd6;

parameter execute1\_int=4'd7;

parameter execute2\_int=4'd8;

parameter execute3\_int=4'd9;

parameter execute4\_int=4'd10;

parameter execute5\_int=4'd11;

always@(posedge clk or posedge rst)

begin

if(rst)

p\_state<=fetch1\_alu;

else

p\_state<=n\_state;

end // end always clk

always@(posedge clk or posedge rst)

begin

if(rst)

begin

load\_r0<=0;

load\_r1<=0;

load\_r2<=0;

load\_r3<=0;

load\_pc<=0;

inc\_pc<=0;

sel\_bus1<=3'd0;

load\_ir<=0;

load\_addr<=0;

load\_regy<=0;

load\_regz<=0;

sel\_bus2<=2'd0;

write<=0;

read<=0;

n\_state<=3'd0;

inc\_sp<=0;

dec\_sp<=0;

end // end if rst

else

begin

case(p\_state)

fetch1\_alu:

begin

if(interrupt)

n\_state<=execute1\_int;

else

begin

load\_sp<=0;

inc\_sp<=0;

dec\_sp<=0;

write<=0;

load\_pc<=1;

sel\_bus1<=3'd4;

sel\_bus2<=2'd1;

load\_addr<=1;

read<=1;

sel\_bus2<=2'd2;

//load\_ir<=1;

//load\_pc<=1;

inc\_pc<=1;

n\_state<=fetch2\_alu;

end //end else

end //end fetch1

fetch2\_alu:

begin

//load\_sp<=0;

//inc\_sp<=0;

//dec\_sp<=0;

//write<=0;

sel\_bus2<=2'd1;

read<=0;

load\_addr<=0;

load\_ir<=1;

//sel\_bus1<=3'd4;

load\_pc<=0;

//sel\_bus2<=2'd2;

//load\_ir<=1;

//load\_pc<=1;

//inc\_pc<=1;

n\_state<=decode;

end //end fetch2

decode:

begin

//load\_pc<=0;

load\_ir<=0;

inc\_pc<=0;

case(instruction[3:0])

4'd0:n\_state<=execute1\_alu;

4'd1:n\_state<=execute1\_alu;

4'd2:n\_state<=execute1\_alu;

4'd3:n\_state<=execute1\_alu;

4'd4:n\_state<=execute1\_alu;

4'd5:n\_state<=execute1\_alu;

4'd6:n\_state<=execute1\_alu;

4'd7:n\_state<=execute1\_alu;

4'd8:n\_state<=execute1\_alu;

4'd9:n\_state<=execute1\_alu;

4'd10:n\_state<=execute1\_alu;

4'd11:n\_state<=execute1\_alu;

4'd12:n\_state<=execute1\_alu;

4'd13:n\_state<=execute1\_alu;

4'd14:n\_state<=execute1\_alu;

4'd15:n\_state<=execute1\_alu;

default:n\_state<=n\_state;

endcase

end //end decode

execute1\_alu: /////////////// from execute 1

begin

case(instruction[5:4])

2'd0:begin

load\_r0<=1;

load\_r1<=0;

load\_r2<=0;

load\_r3<=0;

sel\_bus1<=3'd0;

sel\_bus2<=2'd1;

load\_regy<=1;

n\_state<=execute2\_alu;

end //end 2'd0

2'd1:begin

load\_r0<=0;

load\_r1<=1;

load\_r2<=0;

load\_r3<=0;

sel\_bus1<=3'd1;

sel\_bus2<=2'd1;

load\_regy<=1;

n\_state<=execute2\_alu;

end // end 2'd1

2'd2:begin

load\_r0<=0;

load\_r1<=0;

load\_r2<=1;

load\_r3<=0;

sel\_bus1<=3'd2;

sel\_bus2<=2'd1;

load\_regy<=1;

n\_state<=execute2\_alu;

end // end 2'd2

2'd3:begin

load\_r0<=0;

load\_r1<=0;

load\_r2<=0;

load\_r3<=1;

sel\_bus1<=3'd3;

sel\_bus2<=2'd1;

load\_regy<=1;

n\_state<=execute2\_alu;

end //end 2'd3

default:n\_state<=n\_state;

endcase

end // end execute1\_alu

execute2\_alu:

begin

load\_regy<=0;

//load\_regz<=1;

case(instruction[7:6])

2'd0:begin

load\_r0<=1;

load\_r1<=0;

load\_r2<=0;

load\_r3<=0;

sel\_bus1<=3'd0;

sel\_bus2<=2'd0;

load\_regz<=1;

n\_state<=execute3\_flag;

end // end 2'd0

2'd1:begin

load\_r0<=0;

load\_r1<=1;

load\_r2<=0;

load\_r3<=0;

sel\_bus1<=3'd1;

sel\_bus2<=2'd0;

load\_regz<=1;

n\_state<=execute3\_flag;

end // end 2'd1

2'd2:begin

load\_r0<=0;

load\_r1<=0;

load\_r2<=1;

load\_r3<=0;

sel\_bus1<=3'd2;

sel\_bus2<=2'd0;

load\_regz<=1;

n\_state<=execute3\_flag;

end // end 2'd2

2'd3:begin

load\_r0<=0;

load\_r1<=0;

load\_r2<=0;

load\_r3<=1;

sel\_bus1<=3'd3;

sel\_bus2<=2'd0;

load\_regz<=1;

n\_state<=execute3\_flag;

end // end 2'd3

default:n\_state<=n\_state;

endcase

end // end execute2\_alu

execute3\_flag:

begin

if((alu\_zflag==1'b1)||(alu\_overflow==1'b1))

load\_regz<=1;

n\_state<=write\_back;

end

write\_back:

begin

load\_regz<=0;

load\_addr<=1;

write<=1;

n\_state<=fetch1\_alu;

end

execute1\_int:

begin

load\_sp<=1;

sel\_bus1<=3'd5;

sel\_bus2<=2'd1;

load\_addr<=1;

n\_state<=execute2\_int;

end

execute2\_int:

begin

//load\_sp<=1;

sel\_bus1<=3'd0;

//sel\_bus2<=2'd1;

write<=1;

inc\_pc<=1;

//load\_addr<=1;

n\_state<=execute3\_int;

end

execute3\_int:

begin

inc\_pc<=1;

write<=0;

//load\_sp<=1;

sel\_bus1<=3'd5;

sel\_bus2<=2'd1;

load\_addr<=1;

n\_state<=execute4\_int;

end

execute4\_int:

begin

//load\_sp<=1;

sel\_bus1<=3'd0;

//sel\_bus2<=2'd1;

write<=1;

dec\_sp<=1;

//inc\_pc<=1;

//load\_addr<=1;

n\_state<=execute5\_int;

end

execute5\_int:

begin

dec\_sp<=0;

write<=0;

sel\_bus1<=3'd5;

sel\_bus2<=2'd1;

load\_addr<=1;

//write<=1;

//inc\_pc<=1;

n\_state<=fetch1\_alu;

end

//endcase

endcase

end

end //end else

endmodule

module stack\_pointer(input [7:0] inp1,input inc\_sp,input dec\_sp,input clk,input rst,

input load\_sp,output reg[7:0] out);

always@(posedge clk or posedge rst)

begin

if(rst)

out<=8'd0;

else if(inc\_sp)

out<=inc\_sp+8'd1;

else if(dec\_sp)

out<=inc\_sp-8'd1;

else if(load\_sp)

out<=inp1;

else

out<=out;

end

endmodule

module slow\_clk(clk,rst,slowclk);

input clk,rst;

output reg slowclk;

reg [25:0] count;

always @ (posedge clk or posedge rst)

begin

if(rst)

begin

count<=0;

slowclk<=0;

end

else

begin

count<=count+1'd1;

if(count==5000000)

begin

slowclk<=~slowclk;

count<=0;

end

end

end

endmodule

module abcd( input rst,input clk,input inc,input loadpc,input load0,input load1,

input load2,input load3, input[2:0] sel1,input load\_reg\_z,input loady,

input [1:0] sel2,input loadaddr, input loadIR ,output [1:0] mainout,

output [7:0] wIR,output [7:0] addraddress,input [7:0] wmem ,output [7:0] wbus1,

input load\_sp,input inc\_sp,input dec\_sp);

wire [7:0] w0,w1,w2,w3,wpc,wbus2,waddr,walu,wregy,wsp;

wire alu\_overflow,alu\_zflag;

register register0(.inp1(wbus2),.rst(rst),.clk(clk),.load(load0),.out(w0));

register register1(.inp1(wbus2),.rst(rst),.clk(clk),.load(load1),.out(w1));

register register2(.inp1(wbus2),.rst(rst),.clk(clk),.load(load2),.out(w2));

register register3(.inp1(wbus2),.rst(rst),.clk(clk),.load(load3),.out(w3));

pc pc1(.inp1(wbus2),.rst(rst),.clk(clk),.load(loadpc),.out(wpc),.inc(inc));

stack\_pointer sp(.inp1(wbus2),.inc\_sp(inc\_sp),.dec\_sp(dec\_sp),.clk(clk),.rst(rst),.out(wsp),.load\_sp(load\_sp));

bus1\_mux mux1(.inp0(w0),.inp1(w1),.inp2(w2),.inp3(w3),.inp4(wpc),.inp5(wsp),.sel(sel1),.out(wbus1));

bus2\_mux mux2(.inp1(wbus1),.sel(sel2),.out(wbus2),.inp0(walu),.inp2(wmem));

register addr(.inp1(wbus2),.rst(rst),.clk(clk),.load(loadaddr),.out(addraddress));//addraddress

register IR(.inp1(wbus2),.rst(rst),.clk(clk),.load(loadIR),.out(wIR));

register regy(.inp1(wbus2),.rst(rst),.clk(clk),.load(loady),.out(wregy));

//memory mem(.address(waddr),.data\_in(wbus1),.read(read),.write(write),.clk(clk),.rst(rst),.out(wmem));

regz registerz(.rst(rst),.clk(clk),.inp1({alu\_overflow,alu\_zflag}),.load(load\_reg\_z),.out(mainout));

alu aluu(.inp1(wregy),.inp2(wbus1),.opcode(wIR [2:0]),.overflow(alu\_overflow),.alu\_zero\_flag(alu\_zflag),.out(walu));

endmodule

simulation

`timescale 1ns / 1ps

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

// Company:

// Engineer:

//

// Create Date: 10:40:30 05/22/2018

// Design Name: processor

// Module Name: C:/.Xilinx/abcd/abcd\_testt.v

// Project Name: abcd

// Target Device:

// Tool versions:

// Description:

//

// Verilog Test Fixture created by ISE for module: processor

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

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

module abcd\_testt;

// Inputs

reg clk;

reg rst;

reg inter;

// Outputs

wire [7:0] instruction;

wire zflag;

wire overflow;

// Instantiate the Unit Under Test (UUT)

processor uut (

.instruction(instruction),

.clk(clk),

.rst(rst),

.inter(inter),

.zflag(zflag),

.overflow(overflow)

);

initial begin

// Initialize Inputs

clk = 0;

rst = 1;

inter = 0;

#50 rst=0;

#100 inter=1;

#100 inter=0;

// Add stimulus here

end

always

#5 clk=~clk;

Endmodule

Ucf file

NET "clk" LOC="L15";

NET "rst" LOC="A10";

NET "inter" LOC="D14";

NET "clk" CLOCK\_DEDICATED\_ROUTE=FALSE;

NET "instruction<7>"LOC="N12";

NET "instruction<6>"LOC="P16";

NET "instruction<5>"LOC="D4";

NET "instruction<4>"LOC="M13";

NET "instruction<3>"LOC="L14";

NET "instruction<2>"LOC="N14";

NET "instruction<1>"LOC="M14";

NET "instruction<0>"LOC="U18";

NET "zflag" LOC="E4";

NET "overflow"LOC="T5";