CSE 417 3rd Homework.

Submission deadline: 8 January 2017, @12pm.

Presentations: 16-17 January (together with the final exam presentations).

1) A verilog program will be sent to you. You will modify this program to build a CPU board with 2 pushbuttons and 8 7-segment displays. You must also add support for a counter.

2) Add push, pop, call and ret instructions to the verilog code.

3) Modify the verilog program to see the I/O devices at the following addresses:

pushbutton 1: A00

pushbutton 2: A01 (you have to add this)

first 4 seven segment displays: C00

second 4 seven segment display: C01 (you have to add this)

counter: D00. (You have to add this)

Note that pushbuttons and counter are your input devices, seven segment displays are your output devices.

4) counter is an 16-bit register which increments at (approximately) every second. To construct it, first generate an (approximately) 1Hz clock signal clk1 from the 50 MHz clock of the FPGA board, then increment the counter with that clk1.

```
always @(posedge clk1)
    counter <= counter+1;
```

As mentioned, the CPU must be able to read the value of this counter from the address D00 at any given time, i.e.;

```
ldi 2, 0x0d00
ld 2,2
```

5) Your assembly code (which works on top of your hardware) must perform as follows:

----When pushbutton 1 is pressed, the value in the first 4 seven segment registers will be INCREMENTED by 2 (this value must start with 0000 and must never increment beyond FFFF) and the value of the counter at the time of key press must be read and displayed at the second 4 seven segment displays. The display must only increment once for each pressing of the button.

----When pushbutton 2 is pressed, the value in the first 4 seven segment registers will be DECREMENTED by 2 (this value must never decrement below 0000) and the value of the counter at the time of key press must be read and displayed at the second 4 seven segment displays. The display must only decrement once for each pressing of the button.

In your code you MUST use call and ret instructions.

NOTES:

1) When using $readmemh, load the whole memory, i.e., if your memory is 512 words you must load a 512-word image. Otherwise, your image will not load.

2) Your final hardware/software must work under the 50 MHz clock (In the current code, the clock is connected to the pushbutton).

3) If you have any questions, please contact me at crom.baran@gmail.com

The code

```
module ilk_cpu(grounds, display, clk, pushbutton, iobutton);
        input pushbutton, iobutton;
        output  [3:0] grounds;
        output  [6:0] display;
        input     clk;
        //memory map is defined here
        localparam      BEGINMEM=12'h000,
                        ENDMEM=12'h0ff,
                        KEYPAD=12'h900,
                        SEVENSEG=12'hb00;
        // cpu's input-output pins
        wire      [15:0] pc,pc1;
        reg       [15:0] memory [0:127];
        wire      [15:0] data_out;
        reg       [15:0] data_in;
        wire      [11:0] address;
        wire      memwt;

        // input-output devices
        wire      [15:0] datafrombutton;
```

```
reg       [15:0] ss7;

integer i;


//assign pc1=data_in;

//instantiation of cpu

cpu  cpu1(.clk(clk),  .outdata(pc),  .data_out(data_out),  .data_in(data_in),  .address(address),
.memwt(memwt));

//instantiation ofoutput device=  monitor

monitor monitor1( .grounds(grounds), .display(display), .clk(clk),  .info(ss7));

//instantiation of an input device =button

button button1(.iobutton(iobutton), .out(datafrombutton));


//multiplexer for cpu input

always @*

        if ( (BEGINMEM<=address) && (address<=ENDMEM) )

                data_in=memory[address];

        else if ( (KEYPAD==address) )

                data_in=datafrombutton;

        else

                data_in=16'hf345;


//multiplexer for cpu output

always @(posedge clk) //data output port of the cpu

        if (memwt)

                if ( (BEGINMEM<=address) && (address<=ENDMEM) )

                        memory[address]<=data_out;

                else if ( SEVENSEG==address)

                        ss7<=data_out;
```

```
initial begin
                        memory[0]=16'h1000; //ldi 0 0x900
                        memory[1]=16'h0900;
                        memory[2]=16'h2001; //ld 1 0
                        memory[3]=16'h7049; //add 0 0 0, to set zero flag
                        memory[4]=16'h4ffb; //jz -5
                        memory[5]=16'h7e9b; //inc 3
                        memory[6]=16'h1004; //ldi 4, 0xb00
                        memory[7]=16'h0b00;
                        memory[8]=16'h30e0; //st 4 3
                        memory[9]=16'h5ff6; //jmp -10
                        ss7=16'b0;
                        for (i=10;i<127;i=i+1)
                        begin
                                memory[i]=16'h0;
                        end
//                      $readmemh("prog.txt", memory);
                end

endmodule

module cpu( clk, outdata, data_out, data_in, address, memwt  );
        input clk;
        output [15:0] outdata, data_out;
        input [15:0] data_in;
        output reg [11:0] address;
        output wire memwt;

        localparam      FETCH=4'b0000,
```

```
                                LDI=4'b0001,
                                LD=4'b0010,
                                ST=4'b00011,
                                JZ=4'b0100,
                                JMP=4'b0101,
                                ALU=4'b0111;
reg [11:0] pc;
reg [11:0] ir;
reg [4:0] state;
reg [15:0] regbank [7:0];
reg [15:0] result;
wire zeroresult;

assign outdata={memwt, 3'b0, address};
//assign outdata=regbank[6];

always @(posedge clk)
        case(state)
                FETCH:
                begin
                        state<=data_in[15:12];
                        ir<=data_in[11:0];
                        pc<=pc+1;
                end

                LDI:
                begin
                        state<=FETCH;
                        // regbank[ ir[2:0] ] <= memory[pc];
```

```
                regbank[ ir[2:0] ] <= data_in;
                pc<=pc+1;
        end

        LD:
        begin
                state<=FETCH;
                if (ir[2:0]!=3'h6)
//                              regbank[ir[2:0]] <= memory[regbank[ ir[5:3] ] [11:0]];
                                regbank[ir[2:0]] <= data_in;
        end


        ST:
        begin
                state<=FETCH;
//              memory[ regbank[ir[5:3]][11:0] ] <= regbank[ ir[8:6] ];
        end

        JZ:
        begin
                if (regbank[6][0])
                        pc <= pc+ir;
                state<=FETCH;
        end

        JMP:
        begin
                pc <= pc+ir;
```

```
                state<=FETCH;
        end

        ALU:
        begin
                state<=FETCH;
                regbank[ir[2:0]]<=result;
                regbank[6][0]<=zeroresult;
        end
endcase

always @*   //address bus of the cpu
        case (state)
                FETCH: address =pc;
                LDI:   address= pc;
                LD:    address=regbank[ ir[5:3] ][11:0];
                ST:    address=regbank[ ir[5:3] ][11:0];
                default: address=pc;
        endcase

assign data_out = regbank[ ir[8:6] ];
assign memwt=(state==ST);

always @*
        case (ir[11:9])
                3'h0: result = regbank[ir[8:6]]+regbank[ir[5:3]];
                3'h1: result = regbank[ir[8:6]]-regbank[ir[5:3]];
                3'h2: result = regbank[ir[8:6]]&regbank[ir[5:3]];
                3'h3: result = regbank[ir[8:6]]|regbank[ir[5:3]];
```

```
                3'h4: result = regbank[ir[8:6]]^regbank[ir[5:3]];
                3'h7: case (ir[8:6])
                                3'h0: result = !regbank[ir[5:3]];
                                3'h1: result = regbank[ir[5:3]];
                                3'h2: result = regbank[ir[5:3]]+1;
                                3'h3: result = regbank[ir[5:3]]-1;
                                default: result=16'h0000;
                        endcase
                default: result=16'h0000;
        endcase

    assign zeroresult = ~|result;
    initial begin
                state=FETCH;
    end
endmodule

module monitor( grounds, display, clk, info);
    output reg [3:0] grounds;
    output reg [6:0] display;
    input clk;
    input [15:0] info;
    //reg [15:0] info;
    reg [3:0] data [3:0] ;
    reg [1:0] count;
    reg [25:0] clk1;

    always @*
        begin
```

```
                data[3]=info[15:12];
                data[2]=info[11:8];
                data[1]=info[7:4];
                data[0]=info[3:0];
        end

always @(posedge clk1[15]) //25 slow //19 wavy //15 perfect
begin
        grounds<={grounds[2:0],grounds[3]};
        count<=count+1;
end

always @(posedge clk)
        clk1<=clk1+1;

always @(*)
        case(data[count])
                0:display=7'b1111110; //starts with led a, ends with led g
                1:display=7'b0110000;
                2:display=7'b1101101;
                3:display=7'b1111001;
                4:display=7'b0110011;
                5:display=7'b1011011;
                6:display=7'b1011111;
                7:display=7'b1110000;
                8:display=7'b1111111;
                9:display=7'b1111011;
                'ha:display=7'b1110111;
                'hb:display=7'b0011111;
```

```
                'hc:display=7'b1001110;
                'hd:display=7'b0111101;
                'he:display=7'b1001111;
                'hf:display=7'b1000111;
                default display=7'b1111111;
        endcase

        initial begin
                data[0]=4'h8;
                data[1]=4'h9;
                data[2]=4'ha;
                data[3]=4'hb;
                grounds=4'b1110;
                clk1=0;
        end

endmodule

module button(iobutton, out);
        input iobutton; //button input, coming from pin assignment
        output reg [15:0] out; //16-bit data to be sent to cpu .Only bit 0 matters.
        always @*
                out[0]=~iobutton;
        initial begin
                out=16'h0000;
        end
endmodule
```