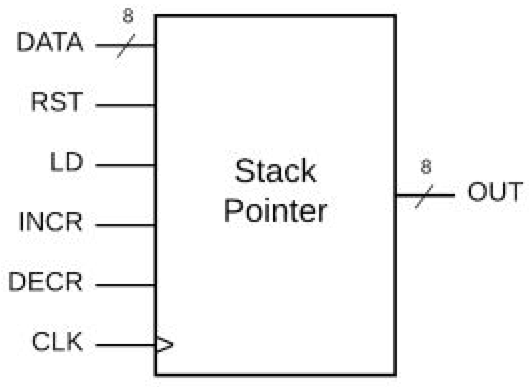
Michael Hegglin

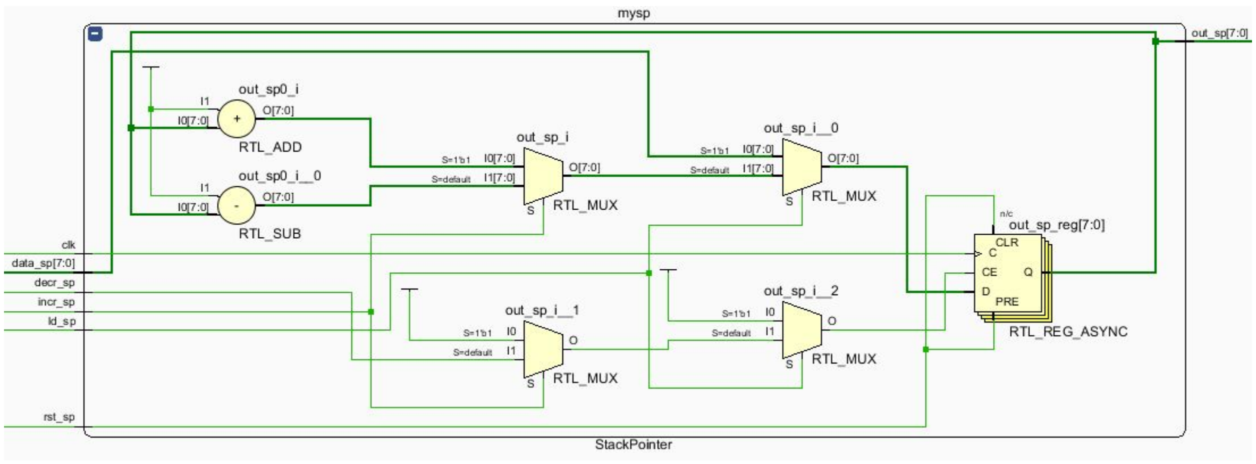
Black Box Diagram:



Behavioral Description:

The stack pointer keeps track of the address in the Scratch RAM that is the top of the stack. The stack pointer is used as an input into the Scratch RAM for pushing data onto and popping data off of the stack. As data is pushed on top of the stack, the address is decremented, and as data is popped off the top of the stack, the address is incremented. The stack pointer can also be reset or loaded. All operations of the stack pointer occur synchronously.

Structural Design:



Source Code:

Michael Hegglin

```
module StackPointer(data_sp, rst_sp, ld_sp, incr_sp, decr_sp, clk,
out_sp);

    input [7:0] data_sp;
    input rst_sp, ld_sp, incr_sp, decr_sp, clk;
    output reg [7:0] out_sp;

    always @(posedge rst_sp, posedge clk)
    begin
        if (rst_sp == 1)       // asynch reset
           out_sp <= 'hF9;
        else if (ld_sp == 1)   // load new value
           out_sp <= data_sp;
        else if (incr_sp == 1)   // count up (increment)
           out_sp <= out_sp + 1;
        else if (decr_sp == 1)   // count down (decrement)
           out_sp <= out_sp - 1;
    end

Endmodule


`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////
/////////////
// Company:
// Engineer:
//
// Create Date: 02/12/2019 01:38:58 PM
// Design Name:
// Module Name: realControl
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////
```

```
/////////////


module realControl(c_flag, z_flag, clk, reset, in_port, ophi, oplo,
pc_reset, sc_reset, pc_inc, alu_opy_sel, flg_c_ld, flg_z_ld, io_strb,
pc_ld, rf_wr,
 pc_mux_sel, rf_wr_sel, alu_sel, scr_we, flag_c_clr, pc_sel,
flag_c_set,
rst_sp, ld_sp, incr_sp, decr_sp, scr_addr_sel, scr_data_sel);

    input clk, reset, c_flag, z_flag;
    input [7:0] in_port;
    input [4:0] ophi;
    input [1:0] oplo;


    parameter RESET = 2'b00;
    parameter FETCH = 2'b01;
    parameter EXECUTE = 2'b10;

    logic [1:0] ps, ns;
    output logic [1:0] pc_mux_sel, rf_wr_sel, pc_sel;
    output logic flag_c_set, flag_c_clr, pc_reset, sc_reset, pc_inc;
    output logic scr_data_sel, alu_opy_sel, flg_c_ld, flg_z_ld,
io_strb, pc_ld, rf_wr, scr_we, rst_sp, ld_sp, incr_sp, decr_sp;
    output logic [3:0] alu_sel, scr_addr_sel;

    always_ff @(posedge clk)
        if(reset)
            ps <= RESET;
        else
            ps <= ns;

    always_comb begin
        pc_reset <= 1'b0;
        sc_reset <= 'b0;
        pc_inc <= 'b0;
        alu_opy_sel <= 'b0;
        flg_c_ld <= 'b0;
        flag_c_clr <= 'b0;
        flag_c_set <= 'b0;
        flg_z_ld <= 'b0;
        io_strb <= 'b0;
```

```
pc_ld <= 'b0;
rf_wr <= 'b0;
scr_we <= 'b0;
alu_sel <= 'b0;
rf_wr_sel <= 2'b00;
pc_mux_sel <= 'b0;
pc_sel <= 'b0;
rst_sp <= 'b0;
ld_sp <= 'b0;
incr_sp <= 'b0;
decr_sp <= 'b0;
scr_addr_sel <= 'b0;
scr_data_sel <= 'b0;

case(ps)
    RESET: begin
        pc_reset <= 1'b1;
        sc_reset <= 1'b1;
        rst_sp <= 1'b1;
        ns <= FETCH;
    end

    FETCH: begin
        pc_inc <= 1'b1;
        ns <= EXECUTE;
    end

    EXECUTE: begin
        ns <= FETCH;
        pc_inc <= 1'b0;

        case(ophi[4])
            1'b0: begin //
                case({ophi, oplo})
                    7'b0010000: begin //BRN
                        pc_ld <= 1'b1;
                    end

                    7'b0010001: begin // call
                        scr_we <= 1'b1;
                        pc_ld <= 1'b1;
                        scr_addr_sel <= 2'b11;
                        decr_sp <= 1'b1;
        scr_data_sel <= 1'b1;
    end

    7'b0100110: begin // pop
        rf_wr <= 1'b1;
        rf_wr_sel <= 2'b01;
        incr_sp <= 1'b1;
        scr_addr_sel <= 2'b10;
    end

    7'b0100101: begin // push
        scr_we <= 1'b1;
        rf_wr_sel <= 2'b01;
        decr_sp <= 1'b1;
        scr_addr_sel <= 2'b11;
    end

    7'b0110010: begin // ret
        incr_sp <= 1'b1;
        scr_addr_sel <= 2'b10;
        pc_mux_sel <= 2'b01;
        pc_ld <= 1'b1;
    end

    7'b0101000: begin // wsp
        ld_sp <= 1'b1;
    end

    7'b0001011: begin // st
        scr_addr_sel <= 2'b00;
        scr_we <= 1'b1;
    end

    7'b0001010: begin // ld
        rf_wr <= 1'b1;
        rf_wr_sel <= 2'b01;
    end

    7'b0000100: begin //ADD
        rf_wr <= 1'b1;
        flg_z_ld <= 1'b1;
        flg_c_ld <= 1'b1;
    end
```

```
7'b00000101: begin //ADDC
    rf_wr <= 1'b1;
    alu_sel <= 4'b0001;
    flg_z_ld <= 1'b1;
    flg_c_ld <= 1'b1;
end

7'b00000000: begin //AND
    rf_wr <= 1'b1;
    flg_z_ld <= 1'b1;
    flag_c_clr <= 1'b1;
    alu_sel <= 4'b0101;
end

7'b01001000: begin //ASR
    rf_wr <= 1'b1;
    flg_z_ld <= 1'b1;
    flg_c_ld <= 1'b1;
    alu_sel <= 4'b1101;
end

7'b0010101: begin //BRCC
    pc_sel <= 2'b00;
    pc_ld <= ~c_flag;
end

7'b0010100: begin //BRCS
    pc_sel <= 2'b00;
    pc_ld <= c_flag;
end

7'b0010010: begin //BREQ
    pc_sel <= 2'b00;
    pc_ld <= z_flag;
end

7'b0010011: begin //BRNE
    pc_sel <= 2'b00;
    pc_ld <= ~z_flag;
end

7'b0110000: begin //CLC
```

```
    flag_c_clr <= 1'b1;
end

7'b0001000: begin //CMP
    alu_sel <= 4'b0100;
    flg_c_ld <= 1'b1;
    flg_z_ld <= 1'b1;
end

7'b0100000: begin //LSL
    rf_wr <= 1'b1;
    alu_sel <= 4'b1001;
    flg_c_ld <= 1'b1;
    flg_z_ld <= 1'b1;
end

7'b0100001: begin //LSR
    rf_wr <= 1'b1;
    alu_sel <= 4'b1010;
    flg_c_ld <= 1'b1;
    flg_z_ld <= 1'b1;
end

7'b0000001: begin //OR
    rf_wr <= 1'b1;
    alu_sel <= 4'b0110;
    flag_c_clr <= 1'b1;
    flg_z_ld <= 1'b1;
end

7'b0100010: begin //ROL
    rf_wr <= 1'b1;
    alu_sel <= 4'b1011;
    flg_c_ld <= 1'b1;
    flg_z_ld <= 1'b1;
end

7'b0100011: begin //ROR
    rf_wr <= 1'b1;
    alu_sel <= 4'b1100;
    flg_c_ld <= 1'b1;
    flg_z_ld <= 1'b1;
end
```

```
7'b0110001: begin //SEC
    flag_c_set <= 1'b1;
end

7'b0000110: begin //SUB
    rf_wr <= 1'b1;
    alu_sel <= 4'b0010;
    flg_c_ld <= 1'b1;
    flg_z_ld <= 1'b1;
end

7'b0000111: begin //SUBC
    rf_wr <= 1'b1;
    alu_sel <= 4'b0011;
    flg_c_ld <= 1'b1;
    flg_z_ld <= 1'b1;
end

7'b0000011: begin //TEST
    rf_wr <= 1'b1;
    alu_sel <= 4'b1000;
    flag_c_clr <= 1'b1;
    flg_z_ld <= 1'b1;
end

7'b0000010: begin //exor 2 registers
    rf_wr <= 1'b1;
    rf_wr_sel <= 2'b00;
    alu_sel <= 4'b0111;
    alu_opy_sel <= 1'b0;
    flg_c_ld <= 1'b1;
    flg_z_ld <= 1'b1;
end

7'b0001001: begin //mov with regs
    rf_wr <= 1'b1;
    rf_wr_sel <= 2'b00;
    alu_sel <= 4'b1110;
    alu_opy_sel <= 1'b0;
    flg_c_ld <= 1'b0;
    flg_z_ld <= 1'b0;
end
```

```
        default: ns <= FETCH;
    endcase
end

1'b1:
    case(ophi)
        5'b10010: begin //exor with an immediate
            rf_wr <= 1'b1;
            rf_wr_sel <= 2'b00;
            alu_sel <= 4'b0111;
            alu_opy_sel <= 1'b1;
            flg_c_ld <= 1'b1;
            flg_z_ld <= 1'b1;
        end

        7'b11101: begin // st
            scr_addr_sel <= 2'b01;
            scr_we <= 1'b1;
        end

        7'b11100: begin // ld
            rf_wr <= 1'b1;
            rf_wr_sel <= 2'b01;
            scr_addr_sel <= 2'b01;
        end

        5'b10100: begin //ADD
            rf_wr <= 1'b1;
            flg_z_ld <= 1'b1;
            flg_c_ld <= 1'b1;
            alu_opy_sel <= 1'b1;
        end

        5'b10101: begin //ADDC
            rf_wr <= 1'b1;
            alu_sel <= 4'b0001;
            flg_z_ld <= 1'b1;
            flg_c_ld <= 1'b1;
            alu_opy_sel <= 1'b1;
        end

        5'b10000: begin //AND
        rf_wr <= 1'b1;
        flg_z_ld <= 1'b1;
        flag_c_clr <= 1'b1;
        alu_sel <= 4'b0101;
        alu_opy_sel <= 1'b1;
    end

    5'b11000: begin //CMP
        alu_sel <= 4'b0100;
        flg_c_ld <= 1'b1;
        flg_z_ld <= 1'b1;
        alu_opy_sel <= 1'b1;
    end

    5'b10001: begin //OR
        rf_wr <= 1'b1;
        alu_sel <= 4'b0110;
        flag_c_clr <= 1'b1;
        flg_z_ld <= 1'b1;
        alu_opy_sel <= 1'b1;
    end

    5'b10110: begin //SUB
        rf_wr <= 1'b1;
        alu_sel <= 4'b0010;
        flg_c_ld <= 1'b1;
        flg_z_ld <= 1'b1;
        alu_opy_sel <= 1'b1;
    end

    5'b10111: begin //SUBC
        rf_wr <= 1'b1;
        alu_sel <= 4'b0011;
        flg_c_ld <= 1'b1;
        flg_z_ld <= 1'b1;
        alu_opy_sel <= 1'b1;
    end

    7'b0000011: begin //TEST
        rf_wr <= 1'b1;
        alu_sel <= 4'b1000;
        flag_c_clr <= 1'b1;
        flg_z_ld <= 1'b1;
```

```
                                alu_opy_sel <= 1'b1;
                            end

                            5'b11001: begin //in
                                rf_wr <= 1'b1;
                                rf_wr_sel <= 2'b11;
                                alu_sel <= 4'b0000;
                                alu_opy_sel <= 1'b0;
                                flg_c_ld <= 1'b0;
                                flg_z_ld <= 1'b0;
                            end

                            5'b11011: begin //mov with immediate
                                rf_wr <= 1'b1;
                                rf_wr_sel <= 2'b00;
                                alu_sel <= 4'b1110;
                                alu_opy_sel <= 1'b1;
                                flg_c_ld <= 1'b0;
                                flg_z_ld <= 1'b0;
                            end



                            5'b11010: begin //out
                                io_strb <= 1'b1;
                            end

                            default: ns <= FETCH;
                        endcase

                    default: ns <= FETCH;
                endcase
            end

             default: ns <= RESET;
        endcase
    end

endmodule
```