FPGA and HDL EE5193

Final Project -- Reduced Instruction Set Computer (RISC) to be implemented in the FPGA

Student Name: Chao-Jia Liu (Peter)
Student ID: hdj697
Instructor: Savithra Eratne, Ph.D.

Problem Statement:

1. Develop a Reduced Instruction Set Computer to be implemented in the FPGA.
2. Develop each box in the diagram as a block or module
3. Your controller will have 16-bit IR value and RF_Rp_zero as inputs; Identify your outputs for each opcode (left most 4-bits in the instruction)
4. Design four steps execute process per instruction
   I. Fetch (instruction at memory location PC into IR)
   II. Decode
   III. Execute
   IV. Update PC (to the next memory location)
5. Instructions to be demonstrated:
   I. LW 5 201 1001_0101_1100_1001
   II. LW 6 202 1001_0110_1100_1010
   III. ADD 7 5 6 0000_0111_0101_0110
   IV. SW 7 203 1010_0111_1100_1011
   V. LI 8 250 1000_1000_1111_1010
   VI. SUB 4 8 5 0001_0100_1000_0101
   VII. SW 4 204 1010_0100_1100_1100
   VIII. SRA 3 7 ~~0110_0010_0111_0000~~ → 0111_0011_0111_0000
   IX. XOR 2 3 4 0100_0010_0011_0100
   X. SW 2 205 1010_0010_1100_1101
6. Store suitable initial values at memory locations 201 and 202
7. Output: Show the values in memory locations 203, 204 and 205 in 7-segment display once all instructions are executed.
8. The report should include the cover page, the problem statement, explanation of your approach including the reuse of previously developed modules, a block diagram, ASM-D charts, Verilog codes used, the waveform screen-prints, one photo of the synthesis on the board.
9. State problems encountered at simulation and synthesize how they were resolved.

1. Opcode for SRA should be 0111.
2. "3" in binary should be 0011.

Explanation of Approach:

The project report will first discuss how I designed the processor, followed by a block diagram based on the professor's provided design. I added some extra units and wires for operations like LI, JMP, and JR because they need to send the resource address directly between the Register File and Program Counter, so those extra hardware units are mandatory. Next, the report includes an ASM-D chart and comprehensive Verilog code for the final project, including the RISC processor and other hardware designs for the FPGA seven-segment display. Testbench and simulation results for RISC processor and the top module for final project are also provided. At the end, I included a picture of the implementation result on the FPGA board and a video in the Canvas submission.

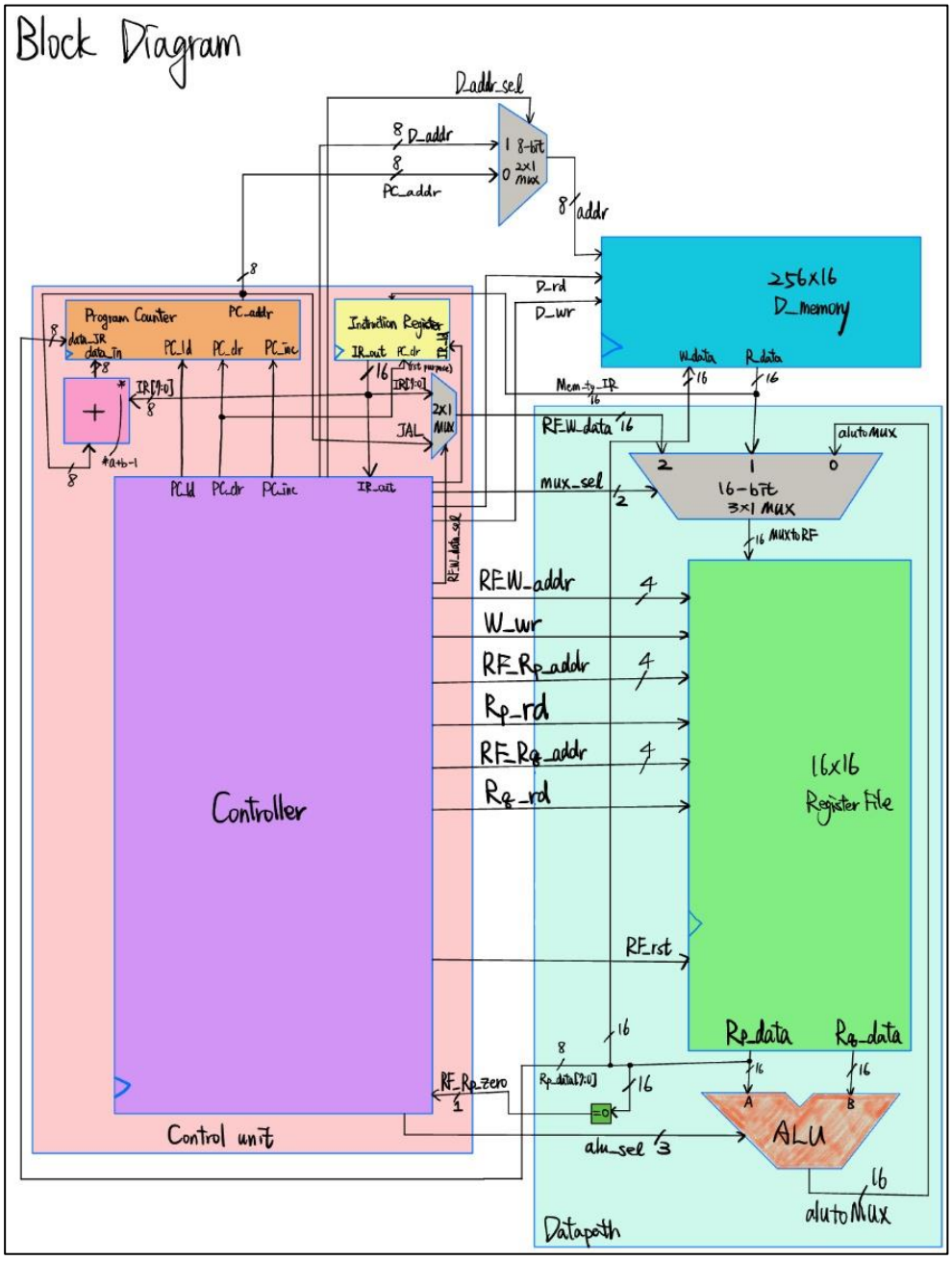
This is the first time I designed a RISC processor with a control unit. Unlike other basic application units, we have to consider the clock cycle, states, and different executions happening each time. I first looked at the block diagram to see what we need in this processor, then looked at the ISA to understand what each opcode

instruction means and their operations. I followed the design in the book to help me understand more about each unit's functionality.

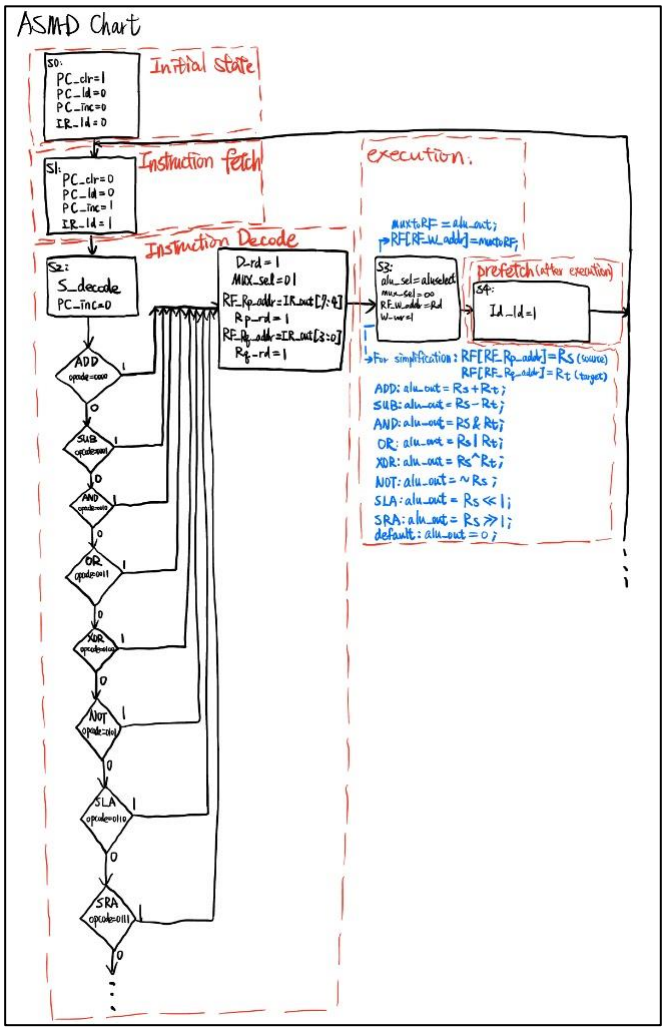
The first and most challenging part was designing the controller. Since this was my first time designing a controller for a processor, I drew a state diagram to follow during the design process. However, when the controller was finally designed, it didn't work as I imagined. Understanding how data flows was also hard to visualize. Therefore, I used a monitor in the testbench to check each data in every clock cycle. Finally, after reviewing the simulation results, I understood how data flows in the processor and what signals from the controller need to be sent in each state.

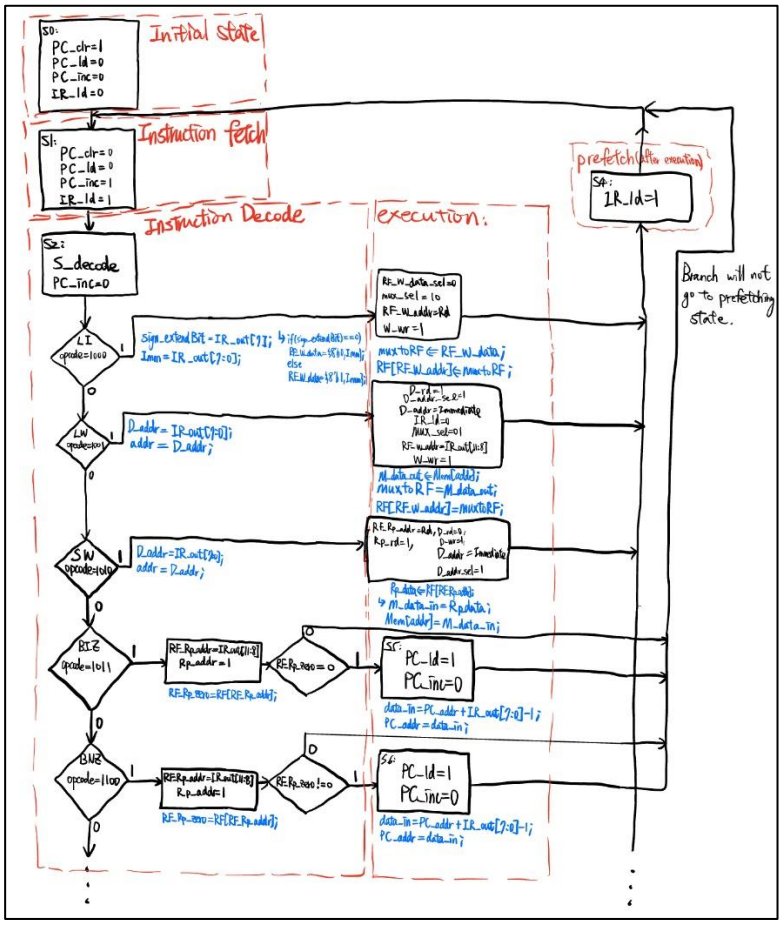
Designing this processor was a great learning opportunity. It helped me understand more about how the control unit works, how different units connect, and, most importantly, how to design a processor from scratch that can be loaded into an FPGA and perform the expected operations.

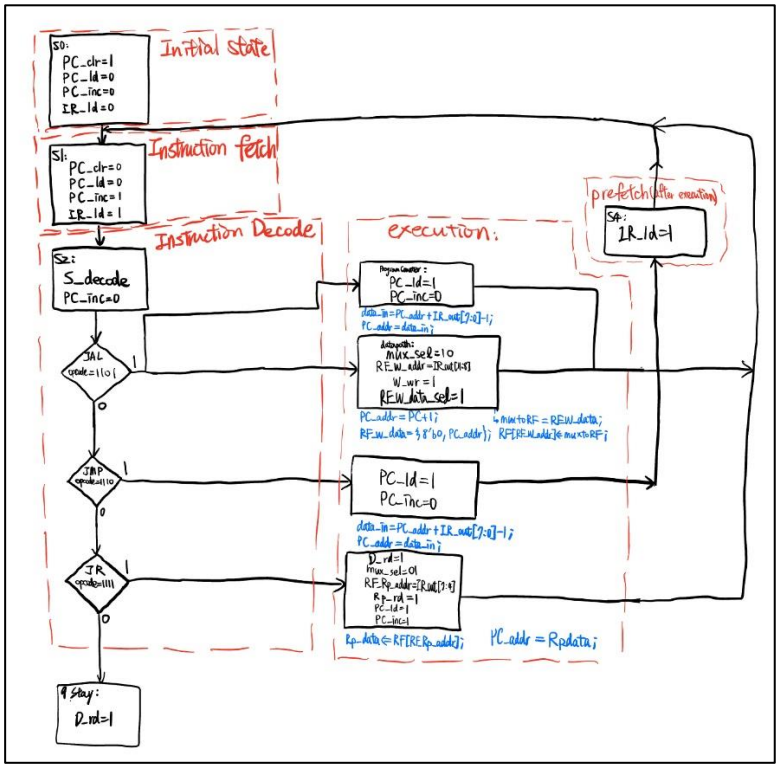
Block Diagram:



ASM-D Chart:

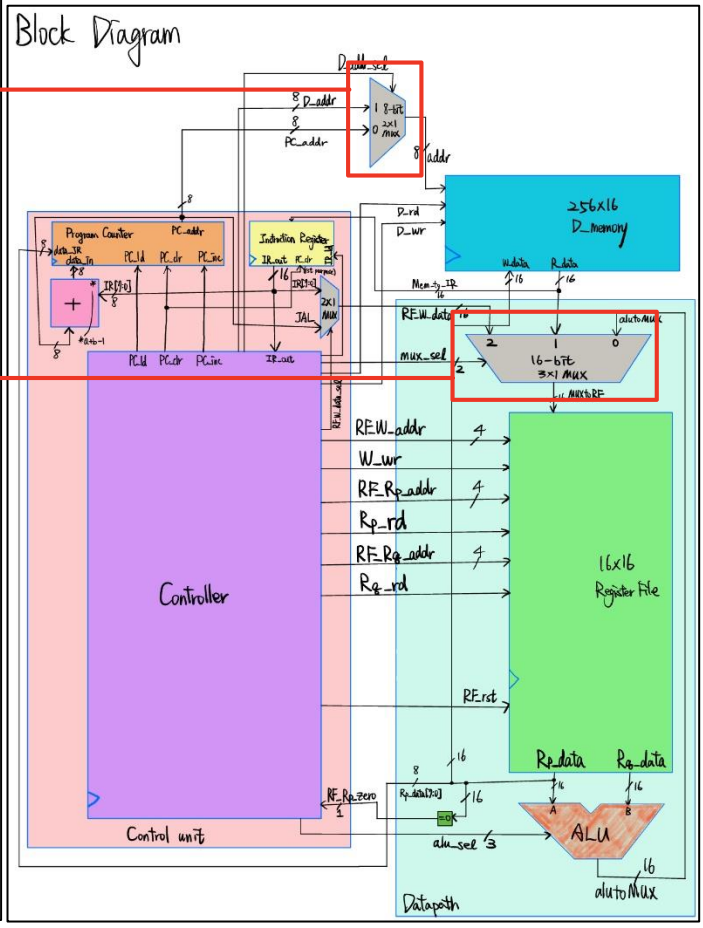

ASMD Chart
Initial state
Instruction fetch
Instruction Decode
execution:
prefetch (after execution)
S0:
PC_clr=1
PC_ld=0
PC_inc=0
IR_ld=0
S1:
PC_clr=0
PC_ld=0
PC_inc=1
IR_ld=1
S2:
S_decode
PC_inc=0
D_rd=1
MUX_sel=01
RF_Rp_addr=IR_out[7:4]
Rp_rd=1
RF_Rq_addr=IR_out[3:0]
Rq_rd=1
S3:
alu_sel=alu selected
mux_sel=00
RF_W_addr=Rd
W_wr=1
S4:
Id_ld=1
ADD opcode=0000
SUB opcode=0001
AND opcode=0010
OR opcode=0011
XOR opcode=0100
NOT opcode=0101
SLA opcode=0110
SRA opcode=0111
mux to RF = alu_out;
RF[RF_W_addr]=mux to RF;
For simplification: RF[RF_Rp_addr]=Rs (source)
RF[RF_Rq_addr]=Rt (target)
ADD: alu_out = Rs + Rt;
SUB: alu_out = Rs - Rt;
AND: alu_out = Rs & Rt;
OR: alu_out = Rs | Rt;
XOR: alu_out = Rs ^ Rt;
NOT: alu_out = ~ Rs ;
SLA: alu_out = Rs << 1;
SRA: alu_out = Rs >> 1;
default: alu_out = 0;


Initial state
Instruction fetch
Instruction Decode
execution:
prefetch (after execution)
S0:
PC_clr=1
PC_ld=0
PC_inc=0
IR_ld=0
S1:
PC_clr=0
PC_ld=0
PC_inc=1
IR_ld=1
S2:
S_decode
PC_inc=0
S4:
IR_ld=1
LI opcode=1000
LW opcode=1001
SW opcode=1010
BRZ opcode=1011
BNZ opcode=1100
Sign_extend Imm = IR_out[7];
Imm = IR_out[7:0];
D_addr = IR_out[7:0];
addr = D_addr;
Rp_data = RF[RF_Rp_addr];
M_data_in = Rp_data;
Mem[addr] = M_data_in;
RF_Rp_zero = RF[RF_Rp_addr];
data_in = PC_addr + IR_out[7:0] - 1;
PC_addr = data_in;
PC_ld=1
PC_inc=0
Branch will not go to prefetching state.



Verilog Code:

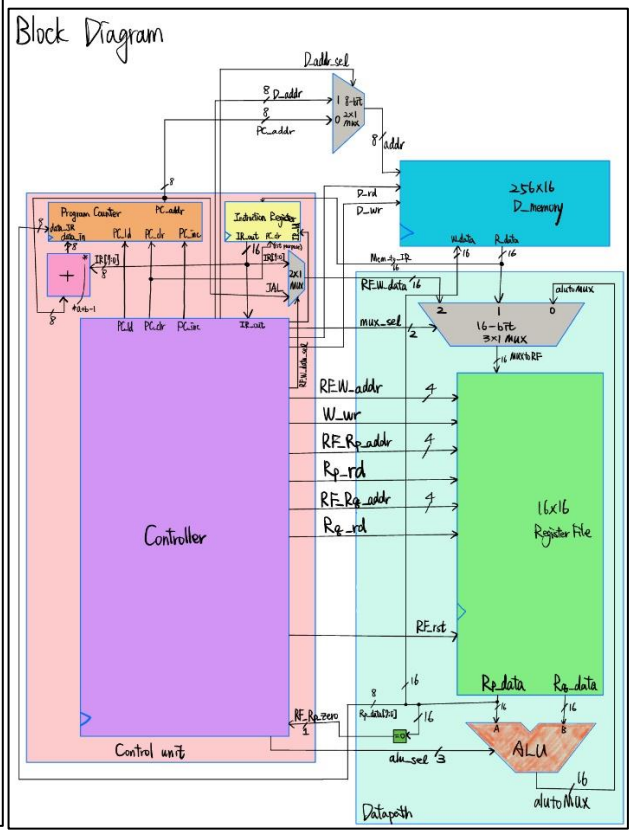
```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: UTSA EE5193
// Engineer: Chao-Jia Liu (Peter)
//////////////////////////////////////////////////////////////////////////////////

module MUX2to1_8b(PC_addr, D_addr, D_addr_sel, addr);
    parameter word_size = 8;
    input [word_size-1:0] PC_addr, D_addr;
    input D_addr_sel;
    output reg [word_size-1:0] addr;
    always@(*) begin
        if(D_addr_sel == 0) addr <= PC_addr;
        else if (D_addr_sel == 1) addr <= D_addr;
    end
endmodule

module MUX3to1_16b(data_0, data_1, data_2, sel, mux_out);
    parameter word_size = 16;
    parameter select_size = 2;
    input [word_size-1:0] data_0, data_1, data_2;
    input [select_size-1:0] sel;
    output reg [word_size-1:0] mux_out;

    always@(*) begin
        case(sel)
            2'b00: mux_out <= data_0;
            2'b01: mux_out <= data_1;
            2'b10: mux_out <= data_2;
            2'b11: mux_out <= 16'bX;
        endcase
    end
endmodule
```



```
35  module RegisterUnit(data_out, data_in, wr, rp_rd, rq_rd, wr_addr_sel, rp_addr_sel, rq_addr_sel, clk, rst);
36      parameter word_size = 16;
37      parameter address_size = 4;
38      output reg [word_size-1:0] data_out;
39      input [word_size-1:0] data_in;
40      input wr_addr_sel, rp_addr_sel, rq_addr_sel;
41      input wr, rp_rd, rq_rd, clk, rst;
42      reg [word_size-1:0] data_inner;
43
44      always @(posedge clk or posedge rst) begin
45          if (rst == 1) begin
46              data_inner <= 0;
47              data_out <= 0;
48          end else if (wr && wr_addr_sel) begin
49              data_inner <= data_in;
50          end
51      end
52
53      always @(*) begin
54          if (rp_rd && rp_addr_sel) begin
55              data_out <= data_inner;
56          end else if (rq_rd && rq_addr_sel) begin
57              data_out <= data_inner;
58          end else begin
59              data_out <= data_out;
60          end
61      end
62  endmodule
63
```

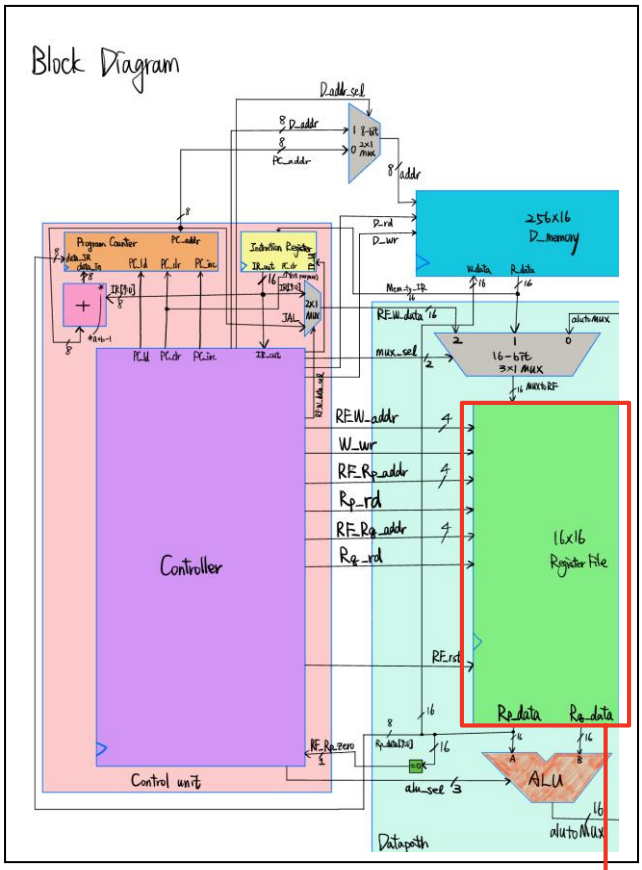
Being used in the 16X16 Register File

```
64  module decoder1to16(address_out, address_in);
65      input [3:0] address_in;
66      output reg [15:0] address_out;
67          always@(*) begin
68              case(address_in)
69                  4'b0000: address_out <= 16'b0000000000000001;
70                  4'b0001: address_out <= 16'b0000000000000010;
71                  4'b0010: address_out <= 16'b0000000000000100;
72                  4'b0011: address_out <= 16'b0000000000001000;
73                  4'b0100: address_out <= 16'b0000000000010000;
74                  4'b0101: address_out <= 16'b0000000000100000;
75                  4'b0110: address_out <= 16'b0000000001000000;
76                  4'b0111: address_out <= 16'b0000000010000000;
77                  4'b1000: address_out <= 16'b0000000100000000;
78                  4'b1001: address_out <= 16'b0000001000000000;
79                  4'b1010: address_out <= 16'b0000010000000000;
80                  4'b1011: address_out <= 16'b0000100000000000;
81                  4'b1100: address_out <= 16'b0001000000000000;
82                  4'b1101: address_out <= 16'b0010000000000000;
83                  4'b1110: address_out <= 16'b0100000000000000;
84                  4'b1111: address_out <= 16'b1000000000000000;
85              endcase
86          end
87  endmodule
88
```



Being used in the 16X16 Register File

```
module selector16to2(r0out, r1out, r2out, r3out, r4out, r5out, r6out, r7out, r8out, r9out,
                     r10out, r11out, r12out, r13out, r14out, r15out, Rp_select, Rq_select, Rp_data, Rq_data);
    parameter word_size = 16;
    parameter address = 4;
    input [word_size-1:0] r0out, r1out, r2out, r3out, r4out, r5out, r6out, r7out, r8out, r9out,
                          r10out, r11out, r12out, r13out, r14out, r15out;
    input [address-1:0] Rp_select, Rq_select;
    output reg [word_size-1:0] Rp_data, Rq_data;
    always@(*) begin
        case (Rp_select)
            4'b0000: Rp_data <= r0out;
            4'b0001: Rp_data <= r1out;
            4'b0010: Rp_data <= r2out;
            4'b0011: Rp_data <= r3out;
            4'b0100: Rp_data <= r4out;
            4'b0101: Rp_data <= r5out;
            4'b0110: Rp_data <= r6out;
            4'b0111: Rp_data <= r7out;
            4'b1000: Rp_data <= r8out;
            4'b1001: Rp_data <= r9out;
            4'b1010: Rp_data <= r10out;
            4'b1011: Rp_data <= r11out;
            4'b1100: Rp_data <= r12out;
            4'b1101: Rp_data <= r13out;
            4'b1110: Rp_data <= r14out;
            4'b1111: Rp_data <= r15out;
        endcase
    end
```

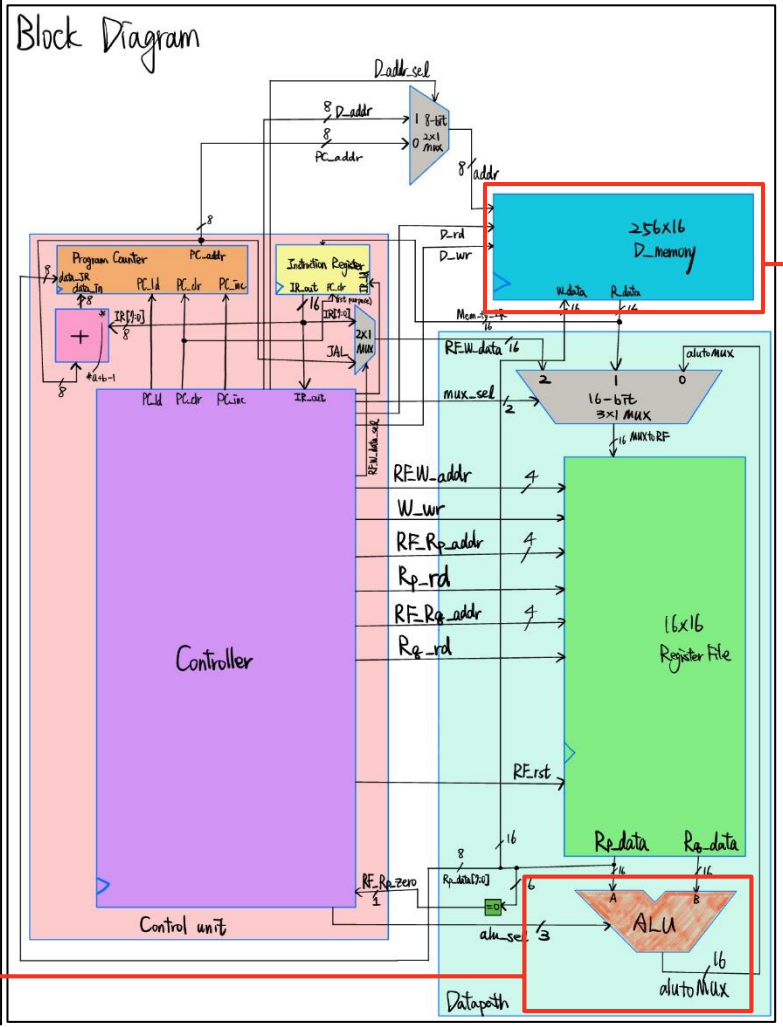



```
    always@(*) begin
        case (Rq_select)
            4'b0000: Rq_data <= r0out;
            4'b0001: Rq_data <= r1out;
            4'b0010: Rq_data <= r2out;
            4'b0011: Rq_data <= r3out;
            4'b0100: Rq_data <= r4out;
            4'b0101: Rq_data <= r5out;
            4'b0110: Rq_data <= r6out;
            4'b0111: Rq_data <= r7out;
            4'b1000: Rq_data <= r8out;
            4'b1001: Rq_data <= r9out;
            4'b1010: Rq_data <= r10out;
            4'b1011: Rq_data <= r11out;
            4'b1100: Rq_data <= r12out;
            4'b1101: Rq_data <= r13out;
            4'b1110: Rq_data <= r14out;
            4'b1111: Rq_data <= r15out;
        endcase
    end
endmodule
```

```
module RegisterFile16x16(Rp_data, Rq_data, W_data, W_addr, W_wr, Rp_addr, Rp_rd, Rq_addr, Rq_rd, clk, rst);
    parameter word_size = 16;
    parameter address = 4;
    output wire [word_size-1:0] Rp_data, Rq_data;
    input [word_size-1:0] W_data;
    input [address-1:0] W_addr, Rp_addr, Rq_addr;
    input W_wr, Rp_rd, Rq_rd, clk, rst;
    wire [word_size-1:0] data_out0, data_out1, data_out2, data_out3, data_out4, data_out5,
                         data_out6, data_out7, data_out8, data_out9, data_out10, data_out11,
                         data_out12, data_out13, data_out14, data_out15;
    wire [word_size-1:0] write_address_select, rp_address_select, rq_address_select;
    reg [15:0] Inner_Register; // keep the data in case, simulation verification purpose

    always@(W_data) begin
        Inner_Register <= W_data;
    end
```

```
    decoder1to16 Write_address_decoder(.address_out(write_address_select),.address_in(W_addr));
    decoder1to16 Rp_address_decoder(.address_out(rp_address_select),.address_in(Rp_addr));
    decoder1to16 Rq_address_decoder(.address_out(rq_address_select),.address_in(Rq_addr));
    RegisterUnit R0(.data_out(data_out0), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[0]), .rp_addr_sel(rp_address_select[0]), .rq_addr_sel(rq_address_select[0]), .clk(clk), .rst(rst));
    RegisterUnit R1(.data_out(data_out1), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[1]), .rp_addr_sel(rp_address_select[1]), .rq_addr_sel(rq_address_select[1]), .clk(clk), .rst(rst));
    RegisterUnit R2(.data_out(data_out2), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[2]), .rp_addr_sel(rp_address_select[2]), .rq_addr_sel(rq_address_select[2]), .clk(clk), .rst(rst));
    RegisterUnit R3(.data_out(data_out3), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[3]), .rp_addr_sel(rp_address_select[3]), .rq_addr_sel(rq_address_select[3]), .clk(clk), .rst(rst));
    RegisterUnit R4(.data_out(data_out4), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[4]), .rp_addr_sel(rp_address_select[4]), .rq_addr_sel(rq_address_select[4]), .clk(clk), .rst(rst));
    RegisterUnit R5(.data_out(data_out5), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[5]), .rp_addr_sel(rp_address_select[5]), .rq_addr_sel(rq_address_select[5]), .clk(clk), .rst(rst));
    RegisterUnit R6(.data_out(data_out6), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[6]), .rp_addr_sel(rp_address_select[6]), .rq_addr_sel(rq_address_select[6]), .clk(clk), .rst(rst));
    RegisterUnit R7(.data_out(data_out7), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[7]), .rp_addr_sel(rp_address_select[7]), .rq_addr_sel(rq_address_select[7]), .clk(clk), .rst(rst));
    RegisterUnit R8(.data_out(data_out8), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[8]), .rp_addr_sel(rp_address_select[8]), .rq_addr_sel(rq_address_select[8]), .clk(clk), .rst(rst));
    RegisterUnit R9(.data_out(data_out9), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[9]), .rp_addr_sel(rp_address_select[9]), .rq_addr_sel(rq_address_select[9]), .clk(clk), .rst(rst));
    RegisterUnit R10(.data_out(data_out10), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[10]), .rp_addr_sel(rp_address_select[10]), .rq_addr_sel(rq_address_select[10]), .clk(clk), .rst(rst));
    RegisterUnit R11(.data_out(data_out11), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[11]), .rp_addr_sel(rp_address_select[11]), .rq_addr_sel(rq_address_select[11]), .clk(clk), .rst(rst));
    RegisterUnit R12(.data_out(data_out12), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[12]), .rp_addr_sel(rp_address_select[12]), .rq_addr_sel(rq_address_select[12]), .clk(clk), .rst(rst));
    RegisterUnit R13(.data_out(data_out13), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[13]), .rp_addr_sel(rp_address_select[13]), .rq_addr_sel(rq_address_select[13]), .clk(clk), .rst(rst));
    RegisterUnit R14(.data_out(data_out14), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[14]), .rp_addr_sel(rp_address_select[14]), .rq_addr_sel(rq_address_select[14]), .clk(clk), .rst(rst));
    RegisterUnit R15(.data_out(data_out15), .data_in(W_data), .wr(W_wr), .rp_rd(Rp_rd), .rq_rd(Rq_rd), .wr_addr_sel(write_address_select[15]), .rp_addr_sel(rp_address_select[15]), .rq_addr_sel(rq_address_select[15]), .clk(clk), .rst(rst));
    selector16to2 memory_output_selector(.r0out(data_out0),.r1out(data_out1),.r2out(data_out2),.r3out(data_out3),.r4out(data_out4),
                .r5out(data_out5),.r6out(data_out6),.r7out(data_out7),.r8out(data_out8),.r9out(data_out9),
                .r10out(data_out10),.r11out(data_out11),.r12out(data_out12),.r13out(data_out13),.r14out(data_out14),
                .r15out(data_out15),.Rp_select(Rp_addr),.Rq_select(Rq_addr),.Rp_data(Rp_data),.Rq_data(Rq_data));
endmodule
```

```verilog
module ALU(A, B, sel, alu_out);
    parameter word_size = 16;
    parameter op_size = 4;
    //operation code definition:
    parameter ADD = 4'b0000;
    parameter SUB = 4'b0001;
    parameter AND = 4'b0010;
    parameter OR  = 4'b0011;
    parameter XOR = 4'b0100;
    parameter NOT = 4'b0101;
    parameter SLA = 4'b0110;
    parameter SRA = 4'b0111;
    input [word_size-1:0] A, B;
    input [op_size-2:0] sel;
    output reg [word_size-1:0] alu_out;

     always@(*) begin
        case(sel)
            ADD: alu_out <= A + B;
            SUB: alu_out <= A - B;
            AND: alu_out <= A & B;
            OR : alu_out <= A | B;
            XOR: alu_out <= A ^ B;
            NOT: alu_out <= ~A;
            SLA: alu_out <= A << 1;
            SRA: alu_out <= A >> 1;
            default alu_out <= 0;
        endcase
     end
endmodule
```

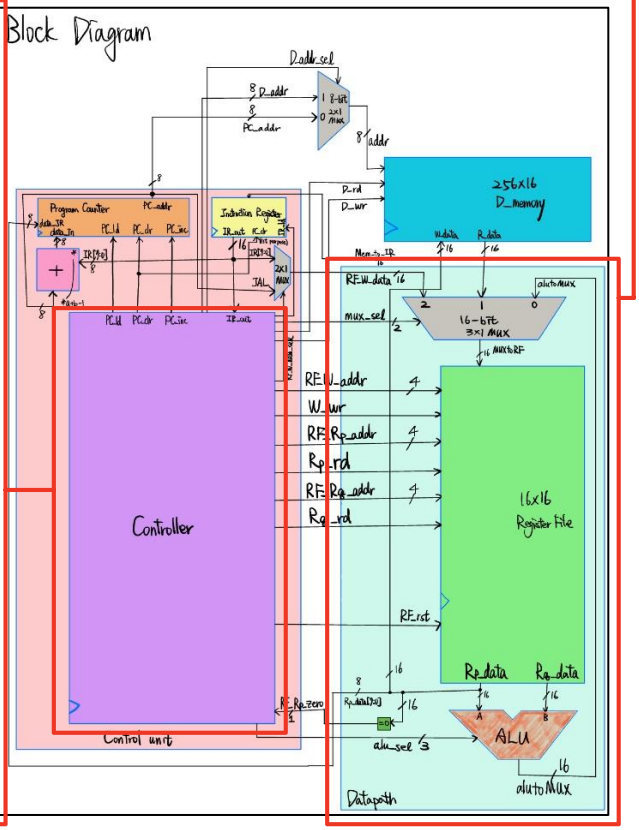



```verilog
module D_memory(R_data, W_data, addr, clk, rd, wr);
    parameter address_size = 8;
    parameter word_size = 16;
    parameter memory_size = 256;
    output reg [word_size-1:0] R_data;
    input [word_size-1:0] W_data;
    input [address_size-1:0] addr;
    input clk, rd, wr;
    reg [word_size-1:0] memory [memory_size-1:0];

    // Initialize memory with specific values at specific addresses
    initial begin
        memory[0] = 16'b1001010111001001;
        memory[1] = 16'b1001011011001010;
        memory[2] = 16'b0000011101010110;
        memory[3] = 16'b1010011111001011;
        memory[4] = 16'b1000100011111010;
        memory[5] = 16'b0001010010000101;
        memory[6] = 16'b1010010011001100;
        memory[7] = 16'b0111001101110000;
        memory[8] = 16'b0100001000110100;
        memory[9] = 16'b1010001011001101;
        memory[201] = 16'h1111; // Initial value for location 201
        memory[202] = 16'h2222; // Initial value for location 202
    end

    always @(posedge clk) begin
        if (wr) begin
            memory[addr] <= W_data;
        end
    end
    always @(*) begin
        R_data = memory[addr];
    end
endmodule
```

Initial Value

```
module Datapath(M_data_in, RF_RP_zero, data_JR, M_data_out, RF_W_data, mux_sel, W_addr, W_wr, Rp_addr, Rp_rd, Rq_addr, Rq_rd, alu_sel, clk, rst);
    parameter word_size1 = 8;
    parameter word_size2 = 16;
    parameter mux_select_size = 2;
    parameter op_size = 4;
    parameter address = 4;
    output RF_RP_zero;
    output [word_size2-1:0] M_data_in;
    output [word_size1-1:0] data_JR;
    input [word_size2-1:0] RF_W_data;
    input [word_size2-1:0] M_data_out;
    wire  [word_size2-1:0] alutomux31;
    input [mux_select_size-1:0] mux_sel;
    wire  [word_size2-1:0] muxtoRF;
    input [address-1:0] W_addr, Rp_addr, Rq_addr;
    wire  [word_size2-1:0] Rp_RFtoalu, Rq_RFtoalu;
    input W_wr, Rp_rd, Rq_rd, clk, rst;
    input [op_size-2:0] alu_sel;

    MUX3to1_16b M1(.data_0(alutomux31), .data_1(M_data_out), .data_2(RF_W_data), .sel(mux_sel), .mux_out(muxtoRF));
    RegisterFile16x16 R1(.Rp_data(Rp_RFtoalu), .Rq_data(Rq_RFtoalu), .W_data(muxtoRF), .W_addr(W_addr), .W_wr(W_wr),
                         .Rp_addr(Rp_addr), .Rp_rd(Rp_rd), .Rq_addr(Rq_addr), .Rq_rd(Rq_rd), .clk(clk), .rst(rst));
    assign M_data_in = Rp_RFtoalu;

    assign data_JR = Rp_RFtoalu[7:0];

    assign RF_RP_zero = ~|Rp_RFtoalu; // reduced OR (unary OR) operation for comparison

    ALU alu1(.A(Rp_RFtoalu), .B(Rq_RFtoalu), .sel(alu_sel), .alu_out(alutomux31));
endmodule
```

```
module Controller(IR_out, RF_Rp_zero, PC_ld, PC_clr, PC_inc,
                  D_addr_sel, D_addr, IR_ld, mux_sel, D_rd, D_wr,
                  RF_W_data_sel, RF_W_addr, W_wr, RF_Rp_addr, Rp_rd,
                  RF_Rq_addr, Rq_rd, RF_rst, alu_sel, clk, rst);
    parameter Instruction_size = 16;
    parameter S_initial = 0, S_fetch = 1, S_decode = 2;
    parameter S_executeALU = 3, S_after_execution = 4;
    parameter S_executeBIZ = 5, S_executeBNZ = 6, S_executeJAL = 7, S_afterJR = 8, S_stay = 9;
    parameter ADD = 0, SUB = 1, AND = 2, OR = 3;
    parameter XOR = 4, NOT = 5, SLA = 6, SRA = 7;
    parameter LI = 8, LW = 9, SW = 10, BIZ = 11;
    parameter BNZ = 12, JAL = 13, JMP = 14, JR = 15;
    input [Instruction_size-1:0] IR_out;
    input RF_Rp_zero;
    output reg PC_ld, PC_clr, PC_inc, D_addr_sel;
    output reg [7:0] D_addr;
    output reg IR_ld;
    output reg [1:0] mux_sel;
    output reg D_rd, D_wr, RF_W_data_sel;
    output reg [3:0] RF_W_addr, RF_Rp_addr, RF_Rq_addr;
    output reg W_wr, Rp_rd, Rq_rd, RF_rst;
    output reg [2:0] alu_sel;
    input clk, rst;
    reg [3:0] state, next_state;
    reg err_flag; // debug simulation
    wire [Instruction_size-1:0] IR_out;
    wire [4:0]opcode = IR_out[15:12];
    wire [4:0] Rd = IR_out[11:8];
    wire [4:0] Rp = IR_out[7:4];
    wire [4:0] Rq = IR_out[3:0];
    wire [7:0] Immediate = IR_out[7:0];
    wire [2:0] aluselect = IR_out[14:12];
```



```
312  always@(posedge clk or posedge rst)begin: State_transitions
313      if(rst == 1) state <= S_initial; else state <= next_state;
314  end
315
316  always@(state or opcode or RF_W_addr or RF_Rp_addr or RF_Rq_addr or RF_Rp_zero)begin
317      PC_ld = 0; PC_clr = 0; PC_inc = 0; D_addr_sel =0;
318      D_addr = 8'b00000000; IR_ld = 0; mux_sel = 2'b00;
319      D_rd = 0; D_wr = 0; RF_W_data_sel = 0;
320      RF_W_addr = Rd; RF_Rp_addr = Rp; RF_Rq_addr = Rq;
321      W_wr = 0; Rp_rd = 0; Rq_rd = 0; RF_rst = 0;
322      alu_sel = 3'b000;
323      err_flag = 0;
324      next_state = state;
325
326      case (state)
327          S_initial:  begin
328                      next_state = S_fetch;
329                      PC_clr = 1; // PC_addr = 8'b00000000
330                      PC_ld = 0;
331                      PC_inc = 0;
332                      IR_ld = 0;
333                      $display("State: S_initial");
334                      end
335          S_fetch  :  begin
336                      next_state = S_decode;
337                      PC_clr = 0;
338                      PC_ld = 0;
339                      PC_inc = 1;
340                      IR_ld = 1;
341                      $display("State: S_fetch");
342                      end
```

```
343              S_decode :  begin
344                          $display("opcode: %b", opcode);
345                          case(opcode)
346                             ADD, SUB, AND, OR, XOR, NOT, SLA, SRA: begin // ALU
347                              next_state = S_executeALU;
348                              D_rd = 1;
349                              mux_sel = 2'b01;
350                              RF_Rp_addr = Rp;
351                              Rp_rd = 1;
352                              RF_Rq_addr = Rq;
353                              Rq_rd = 1;
354                              $display("ALU operation");
355                             end
356                             LI : begin // LI
357                             next_state = S_after_execution;
358                              RF_W_data_sel = 0;
359                              mux_sel = 2'b10;
360                              RF_W_addr = Rd;
361                              W_wr = 1;
362                              $display("LI operation");
363                             end
364                             LW : begin // LW
365                             next_state = S_after_execution;
366                              D_rd = 1;
367                              D_addr_sel = 1;
368                              D_addr = Immediate;
369                              IR_ld = 0;
370                              mux_sel = 2'b01;
371                              RF_W_addr = Rd;
372                              W_wr = 1;
373                              $display("LW operation");
374                             end
```

```
375                             SW : begin // SW
376                             next_state = S_after_execution;
377                              RF_Rp_addr = Rd;
378                              Rp_rd = 1;
379                              D_rd = 0;
380                              D_wr = 1;
381                              D_addr = Immediate;
382                              D_addr_sel = 1;
383                              $display("SW operation");
384                             end
385                             BIZ: begin // BIZ
386                              next_state = S_executeBIZ;
387                              RF_Rp_addr = Rd;
388                              Rp_rd = 1;
389                              $display("BIZ operation");
390                             end
391                             BNZ: begin // BNZ
392                              next_state = S_executeBNZ;
393                              RF_Rp_addr = Rd;
394                              Rp_rd = 1;
395                              $display("BNZ operation");
396                             end
397                             JAL: begin // JAL
398                              next_state = S_after_execution;
399                              RF_W_data_sel = 1;
400                              mux_sel = 2'b10;
401                              RF_W_addr = Rd;
402                              W_wr = 1;
403                              PC_ld = 1;
404                              PC_inc = 0;
405                              $display("JAL operation");
406                             end
```

Controller

```
                        JMP: begin // JMP
                         next_state = S_after_execution;
                         PC_ld = 1;
                         PC_inc = 0;
                         $display("JMP operation");
                        end
                        JR : begin // JR
                         $display("JR operation");
                         D_rd = 1;
                         mux_sel = 2'b01;
                         RF_Rp_addr = Rp;
                         Rp_rd = 1;
                         PC_ld = 1;
                         PC_inc = 1;
                        end
                        default:
                        begin next_state = S_stay;
                        $display("Invalid opcode");
                        end
                    endcase // opcode
                    end
          S_executeALU:begin
                    next_state = S_after_execution;
                    alu_sel = aluselect;
                    mux_sel = 2'b00;
                    RF_W_addr = Rd;
                    W_wr = 1;
                    $display("State: S_afterALU");
                    end
          S_after_execution:begin
                    next_state = S_fetch;
                    IR_ld = 1;
                    $display("State: S_after Execution");
                    end
```

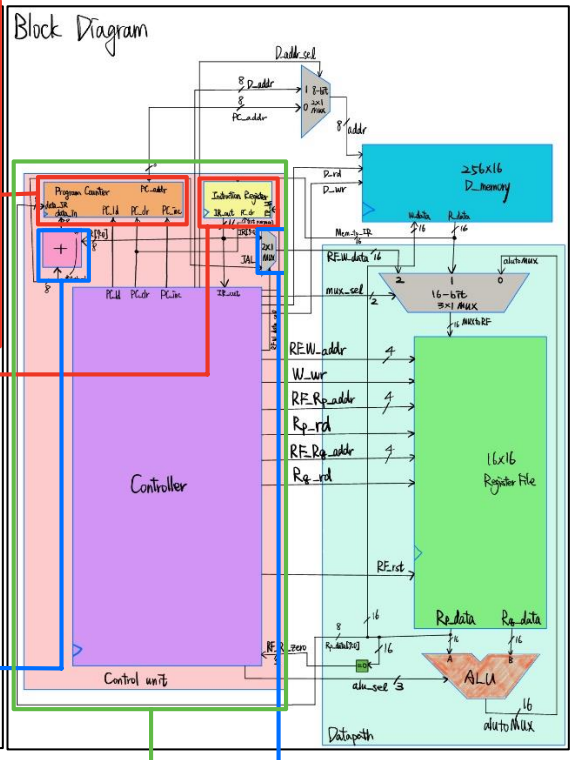
Controller

```
          S_executeBIZ:begin
                    if(RF_Rp_zero == 0)
                      begin
                      next_state = S_fetch;
                      PC_ld = 1;
                      PC_inc = 0;
                      end
                    else begin
                      next_state = S_fetch;
                      end
                      $display("State: S_executeBIZ");
                    end
          S_executeBNZ:begin
                    if(RF_Rp_zero != 0)
                      begin
                      next_state = S_fetch;
                      PC_ld = 1;
                      PC_inc = 0;
                      end
                    else begin
                      next_state = S_fetch;
                      end
                      $display("State: S_executeBNZ");
                    end
          S_stay    :begin
                      D_rd = 1;
                      $display("State: S_stay");
                     end
          default: begin next_state = S_initial;
          $display("State: Default, resetting to S_initial");
          end
        endcase
    end
endmodule
```

```v
module ProgramCounter(PC_addr, data_in, data_JR, PC_ld, PC_inc, PC_clr, clk);
    parameter word_size = 8;
    output reg [word_size-1:0] PC_addr;
    input  [word_size-1:0] data_in, data_JR;
    input  PC_ld, PC_inc, PC_clr, clk;

    always@(posedge clk or posedge PC_clr) begin
        if(PC_clr==1) PC_addr <= 8'b0;
        else if (PC_ld == 1) PC_addr <= data_in;
        else if (PC_inc == 1) PC_addr <= PC_addr + 1;
        else if (PC_ld == 1 && PC_inc == 1) PC_addr <= data_JR;
    end
endmodule
```

```v
module InstructionRegister(data_out, data_in, IR_ld, rst, clk);
    parameter word_size = 16;
    output reg [word_size-1:0] data_out;
    input [word_size-1:0] data_in;
    input IR_ld, rst, clk;
    always@(posedge clk)
        if (IR_ld) data_out <= data_in;
endmodule
```

```v
module Data_in_compute(a, b, out);
    parameter word_size = 8;
    input [word_size-1:0] a, b;
    output [word_size-1:0] out;
    assign out = a + b - 1;
endmodule
```

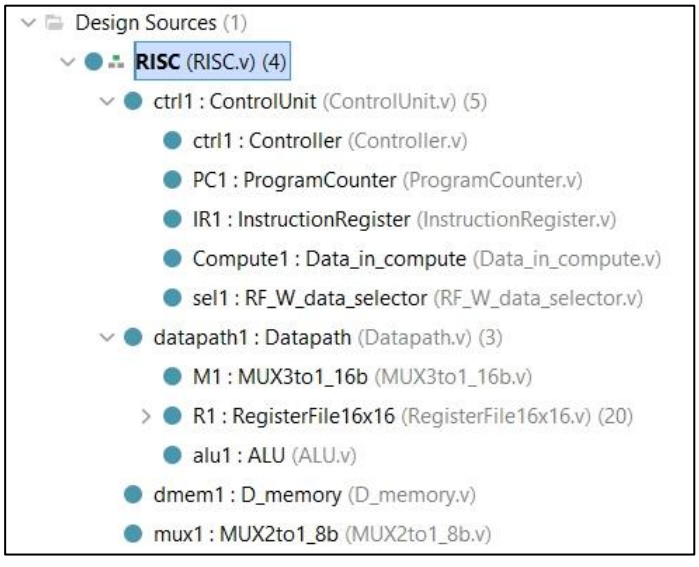


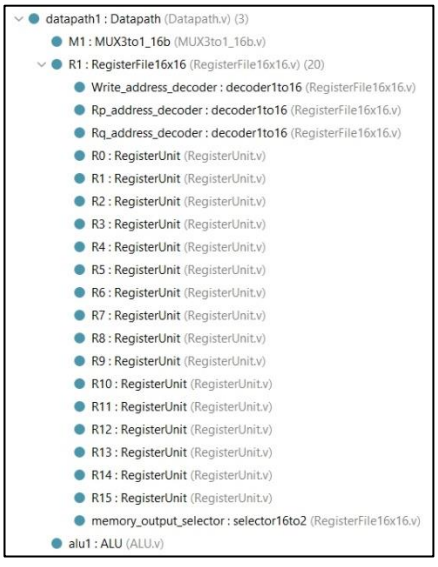

```v
module RF_W_data_selector(data_0, data_1, sign_extendBit, RF_W_data_sel, mux_out);
    parameter in_size = 8;
    parameter out_size = 16;
    input [in_size-1:0] data_0, data_1;
    input RF_W_data_sel, sign_extendBit;
    output reg [out_size-1:0] mux_out;

    always@(*) begin
        case(RF_W_data_sel)
            1'b0:
                if(sign_extendBit == 0)
                    mux_out = {8'b00000000, data_0};
                else if(sign_extendBit == 1)
                    mux_out = {8'b11111111, data_0};
            1'b1: mux_out = {8'b00000000, data_1};
        endcase
    end
endmodule
```

```v
module ControlUnit(RF_Rp_zero, D_addr_sel, D_addr, mux_sel, D_rd, D_wr, RF_W_addr, RF_Rp_addr, RF_Rq_addr,
                   W_wr, Rp_rd, Rq_rd, RF_rst, alu_sel, clk, rst, Mem_to_IR, data_JR, PC_addr, mux_out);
    parameter word_size = 16;
    // Controller
    input RF_Rp_zero;
    output D_addr_sel;
    output [7:0] D_addr;
    output [1:0] mux_sel;
    output D_rd, D_wr;
    output [3:0] RF_W_addr, RF_Rp_addr, RF_Rq_addr;
    output W_wr, Rp_rd, Rq_rd, RF_rst;
    output [2:0] alu_sel;
    input clk, rst;
    // Instruction Register
    input [word_size-1:0] Mem_to_IR;
    // Program Counter
    input [7:0] data_JR;
    output [7:0] PC_addr;
    // RF_W_data_selector
    output [word_size-1:0] mux_out;
    // Controller inner wire
    wire PC_ld, PC_clr, PC_inc, IR_ld, RF_W_data_sel;
    // Instruction Register inner wire
    wire [word_size-1:0] IR_data_out;
    // Program Counter wire
    wire [7:0] data_in;

    Controller ctrl1(.IR_out(IR_data_out), .RF_Rp_zero(RF_Rp_zero), .PC_ld(PC_ld), .PC_clr(PC_clr), .PC_inc(PC_inc),
                     .D_addr_sel(D_addr_sel), .D_addr(D_addr), .IR_ld(IR_ld), .mux_sel(mux_sel), .D_rd(D_rd), .D_wr(D_wr),
                     .RF_W_data_sel(RF_W_data_sel), .RF_W_addr(RF_W_addr), .W_wr(W_wr), .RF_Rp_addr(RF_Rp_addr), .Rp_rd(Rp_rd),
                     .RF_Rq_addr(RF_Rq_addr), .Rq_rd(Rq_rd), .RF_rst(RF_rst), .alu_sel(alu_sel), .clk(clk), .rst(rst));
    ProgramCounter PC1(.PC_addr(PC_addr), .data_in(data_in), .data_JR(data_JR), .PC_ld(PC_ld), .PC_inc(PC_inc), .PC_clr(rst), .clk(clk));
    InstructionRegister IR1(.data_out(IR_data_out), .data_in(Mem_to_IR), .IR_ld(IR_ld), .rst(PC_clr), .clk(clk));
    Data_in_compute Compute1(.a(PC_addr), .b(IR_data_out[7:0]), .out(data_in));
    RF_W_data_selector sel1(.data_0(IR_data_out[7:0]), .data_1(PC_addr), .sign_extendBit(IR_data_out[7]), .RF_W_data_sel(RF_W_data_sel), .mux_out(mux_out));
endmodule
```

```
562  module RISC(clk, rst);
563      parameter instruction_size = 16;
564      parameter address_size = 8;
565      parameter register_address_size = 4;
566      input clk, rst;
567      //wire
568      //Datapath to Control Unit
569      wire RF_Rp_zero;
570      wire [address_size-1:0] data_JR;
571      //Datapath to Memory
572      wire [instruction_size-1:0] Datapath_to_Mem;
573      //Memory to Control Unit
574      wire [instruction_size-1:0] Memory_out;
575      //Contorl Unit to MUX2X1
576      wire D_addr_sel;
577      wire [address_size-1:0] PC_addr, D_addr;
578      //Contorl Unit to D_Memory
579      wire D_rd, D_wr;
580      //Control Unit to Datapath
581      wire [1:0] mux_sel;
582      wire [register_address_size-1:0] RF_W_addr, RF_Rp_addr, RF_Rq_addr;
583      wire W_wr, Rp_rd, Rq_rd;
584      wire [2:0] alu_sel;
585      wire [instruction_size-1:0] mux_out;
586      //MUX2X1 to D_Memory
587      wire [address_size-1:0] addr;
588      ControlUnit ctrl1(.RF_Rp_zero(RF_Rp_zero), .D_addr_sel(D_addr_sel), .D_addr(D_addr), .mux_sel(mux_sel), .D_rd(D_rd), .D_wr(D_wr),
589                        .RF_W_addr(RF_W_addr), .RF_Rp_addr(RF_Rp_addr), .RF_Rq_addr(RF_Rq_addr), .W_wr(W_wr), .Rp_rd(Rp_rd), .Rq_rd(Rq_rd),
590                        .RF_rst(RF_rst), .alu_sel(alu_sel), .clk(clk), .rst(rst), .Mem_to_IR(Memory_out), .data_JR(data_JR), .PC_addr(PC_addr),
591                        .mux_out(mux_out));
592      Datapath datapath1(.M_data_in(Datapath_to_Mem), .RF_RP_zero(RF_Rp_zero), .data_JR(data_JR), .M_data_out(Memory_out), .RF_W_data(mux_out),
593                         .mux_sel(mux_sel), .W_addr(RF_W_addr), .W_wr(W_wr), .Rp_addr(RF_Rp_addr), .Rp_rd(Rp_rd), .Rq_addr(RF_Rq_addr), .Rq_rd(Rq_rd),
594                         .alu_sel(alu_sel), .clk(clk), .rst(RF_rst));
595      D_memory dmem1(.R_data(Memory_out), .W_data(Datapath_to_Mem), .addr(addr), .clk(clk), .rd(D_rd), .wr(D_wr));
596      MUX2to1_8b mux1(.PC_addr(PC_addr), .D_addr(D_addr), .D_addr_sel(D_addr_sel), .addr(addr));
597  endmodule
```

RISC Processor module





Module List

```
599  module bin2bcd( // binary to decimal using double dabble algorithm
600      input [15:0] bin,
601      output reg [19:0] bcd
602      );
603
604      integer i;
605
606      always @(bin) begin
607      bcd = 20'b0;
608
609      for (i = 0; i < 16; i = i + 1) begin
610          // Add 3 to each BCD digit if it is greater than or equal to 5 (Double Dabble algroithm)
611          // compare and add the BCD value
612          if (bcd[3:0] >= 5) bcd[3:0] = bcd[3:0] + 3;
613          if (bcd[7:4] >= 5) bcd[7:4] = bcd[7:4] + 3;
614          if (bcd[11:8] >= 5) bcd[11:8] = bcd[11:8] + 3;
615          if (bcd[15:12] >= 5) bcd[15:12] = bcd[15:12] + 3;
616          if (bcd[19:16] >= 5) bcd[19:16] = bcd[19:16] + 3;
617          // Shift the current BCD value left by one bit
618          bcd = {bcd[18:0], bin[15 - i]};
619          end
620      end
621  endmodule
```

Binary to Decimal Translator

```
module sevensegment1to8( // seven segmant displayment for one LED segment
    input [3:0] in,
    output reg [7:0] seg
);
    always@(in) begin
        case (in)
            4'd0: seg = 8'b11000000; // 0
            4'd1: seg = 8'b11111001; // 1
            4'd2: seg = 8'b10100100; // 2
            4'd3: seg = 8'b10110000; // 3
            4'd4: seg = 8'b10011001; // 4
            4'd5: seg = 8'b10010010; // 5
            4'd6: seg = 8'b10000010; // 6
            4'd7: seg = 8'b11111000; // 7
            4'd8: seg = 8'b10000000; // 8
            4'd9: seg = 8'b10010000; // 9
            default seg = 8'b11111111; // turn off the LED
        endcase
    end
endmodule
```

Seven Segment Value Controller

```
module finalproject_sevensegment(
    input clk_i,
    input [15:0] SW,
    output [15:0] LED,
    output reg [7:0] disp_an_o,
    output reg [7:0] disp_seg_o
    );

    RISC risc1(.clk(clk_i), .rst(SW[0]));

    assign LED = SW;

    reg [32:0] counter_for_mem;
    reg [2:0] select_for_mem;
    reg [15:0] select_for_mem_value;

    always @(posedge clk_i) begin
        if (counter_for_mem != 100000000) begin
            counter_for_mem <= counter_for_mem + 1;
        end
        else begin
            counter_for_mem <= 0;
            select_for_mem <= (select_for_mem < 4)? select_for_mem+1:0; //loop through 0 to 4
        end
    end

    always @(*) begin
        // use case statement
        case (select_for_mem)
            3'd0: begin
                select_for_mem_value <= risc1.dmem1.memory[201];
            end
            3'd1: begin
                select_for_mem_value <= risc1.dmem1.memory[202];
            end
```

Top Module

```
            3'd2: begin
                select_for_mem_value <= risc1.dmem1.memory[203];
            end
            3'd3: begin
                select_for_mem_value <= risc1.dmem1.memory[204];
            end
            3'd4: begin
                select_for_mem_value <= risc1.dmem1.memory[205];
            end
        endcase
    end

    // BCD
    wire [19:0] bcd;
    bin2bcd uut(
        .bin(select_for_mem_value),
        .bcd(bcd)
    );
    // store the value for each LED Segment
    // 5 segment LED will displayed so 5 wire and each are 8-bit
    wire [7:0] seg0;
    wire [7:0] seg1;
    wire [7:0] seg2;
    wire [7:0] seg3;
    wire [7:0] seg4;
    //get the values for each segment (calling 5 times)
    // For seg 0
    sevensegment1to8 LED0(
        .in(bcd[3:0]),
        .seg(seg0)
    );
    //For seg 1
    sevensegment1to8 LED1(
        .in(bcd[7:4]),
        .seg(seg1)
    );
```

```
715         //For seg 2
716         sevensegment1to8 LED2(
717             .in(bcd[11:8]),
718             .seg(seg2)
719         );
720         //For seg 3
721         sevensegment1to8 LED3(
722             .in(bcd[15:12]),
723             .seg(seg3)
724         );
725         //For seg 4
726         sevensegment1to8 LED4(
727             .in(bcd[19:16]),
728             .seg(seg4)
729         );
730
731         //make a 1KHz clock input so the human eyes can not see it switching
732         reg [16:0] counter; // 17-bit counter to divide 100MHDZ Clock into 1KHz
733         reg [2:0] select; // Selector for the mux (3 bits because only need 5)
734         always @(posedge clk_i) begin
735             if (counter != 50000) begin
736                 counter <= counter + 1;
737             end
738             else begin
739                 counter <= 0;
740                 select <= (select < 4)? select+1:0; //loop through 0 to 4
741             end
742         end
743
744         always @(*) begin
745             // use case statement
746             case (select)
747                 3'd0: begin
748                     disp_an_o <= 8'b11111110; //display the AN0
749                     disp_seg_o <= seg0;
750                 end
```

Top Module

```
751                 3'd1: begin
752                     disp_an_o <= 8'b11111101; //display the AN1
753                     disp_seg_o <= seg1;
754                 end
755                 3'd2: begin
756                     disp_an_o <= 8'b11111011; //display the AN2
757                     disp_seg_o <= seg2;
758                 end
759                 3'd3: begin
760                     disp_an_o <= 8'b11110111;  //display the AN3
761                     disp_seg_o <= seg3;
762                 end
763                 3'd4: begin
764                     disp_an_o <= 8'b11101111; //display the AN4
765                     disp_seg_o <= seg4;
766                 end
767                 default: begin
768                     disp_an_o <= 8'b11111111; // all display not used
769                 end
770             endcase
771         end
772     endmodule
```

Testbench and Simulation for *RISC processor*: Result Verification is at the Top Module Part

```
module RISC_tb;
    parameter instruction_size = 16;
    parameter address_size = 8;
    parameter register_address_size = 4;

    reg clk;
    reg rst;
    reg [instruction_size-1:0] Istruc_in;

    RISC risc1 (.clk(clk), .rst(rst));

    always #10 clk = ~clk;

    initial begin
        $monitor("time = %g, clk = %b, PC_addr = %h, IR_out: %h,  Memory_out: %h, Register2 inner value: %h,Register3 inner value:%h, Register4 inner value: %h, Register5 inner value: %h, Register6 inner value: %h, Register7 inner value: %h, Mem_in:%h",
                $time, clk, risc1.ctrl1.PC1.PC_addr, risc1.ctrl1.IR1.data_out, risc1.datapath1.M1.mux_out, risc1.datapath1.R1.R2.data_inner, risc1.datapath1.R1.R3.data_inner, risc1.datapath1.R1.R4.data_inner, risc1.datapath1.R1.R5.data_inner,
                risc1.datapath1.R1.R6.data_inner, risc1.datapath1.R1.R7.data_inner, risc1.datapath1.M_data_in);

        clk = 0; rst = 1;
        #20; rst = 0;
        #980;
        $display("Value at memory location 203: %h", risc1.dmem1.memory[203]);
        $display("Value at memory location 204: %h", risc1.dmem1.memory[204]);
        $display("Value at memory location 205: %h", risc1.dmem1.memory[205]);
        #20;
        $finish;
    end
endmodule
```

Mistype!!! This should be MUX_out (3 to 1 MUX)

```
# run 1000ns
State: Default, resetting to S_initial
State: S_initial
time = 0, clk = 0, PC_addr = xx, IR_out: xxxx,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: xxxx, Register6 inner value: xxxx, Register7 inner value: xxxx, Mem_in:xxxx
time = 10, clk = 1, PC_addr = 00, IR_out: xxxx,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: xxxx, Register6 inner value: xxxx, Register7 inner value: xxxx, Mem_in:xxxx
time = 20, clk = 0, PC_addr = 00, IR_out: xxxx,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: xxxx, Register6 inner value: xxxx, Register7 inner value: xxxx, Mem_in:xxxx
State: S_fetch
State: S_fetch
time = 30, clk = 1, PC_addr = 00, IR_out: 95c9,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: xxxx, Register6 inner value: xxxx, Register7 inner value: xxxx, Mem_in:xxxx
time = 40, clk = 0, PC_addr = 00, IR_out: 95c9,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: xxxx, Register6 inner value: xxxx, Register7 inner value: xxxx, Mem_in:xxxx
opcode: 01001
LW operation
time = 50, clk = 1, PC_addr = 01, IR_out: 95c9,  Memory_out: 1111, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: xxxx, Register6 inner value: xxxx, Register7 inner value: xxxx, Mem_in:xxxx
time = 60, clk = 0, PC_addr = 01, IR_out: 95c9,  Memory_out: 1111, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: xxxx, Register6 inner value: xxxx, Register7 inner value: xxxx, Mem_in:xxxx
State: S_afterLW
time = 70, clk = 1, PC_addr = 01, IR_out: 95c9,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: xxxx, Register7 inner value: xxxx, Mem_in:xxxx
time = 80, clk = 0, PC_addr = 01, IR_out: 95c9,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: xxxx, Register7 inner value: xxxx, Mem_in:xxxx
State: S_fetch
time = 90, clk = 1, PC_addr = 01, IR_out: 96ca,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: xxxx, Register7 inner value: xxxx, Mem_in:xxxx
time = 100, clk = 0, PC_addr = 01, IR_out: 96ca,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: xxxx, Register7 inner value: xxxx, Mem_in:xxxx
opcode: 01001
LW operation
time = 110, clk = 1, PC_addr = 02, IR_out: 96ca,  Memory_out: 2222, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: xxxx, Register7 inner value: xxxx, Mem_in:xxxx
time = 120, clk = 0, PC_addr = 02, IR_out: 96ca,  Memory_out: 2222, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: xxxx, Register7 inner value: xxxx, Mem_in:xxxx
State: S_afterLW
time = 130, clk = 1, PC_addr = 02, IR_out: 96ca,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: xxxx, Mem_in:xxxx
time = 140, clk = 0, PC_addr = 02, IR_out: 96ca,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: xxxx, Mem_in:xxxx
State: S_fetch
State: S_fetch
time = 150, clk = 1, PC_addr = 02, IR_out: 0756,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: xxxx, Mem_in:xxxx
time = 160, clk = 0, PC_addr = 02, IR_out: 0756,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: xxxx, Mem_in:xxxx
```

Mistype!!! This should be MUX_out (3 to 1 MUX)

First instruction loaded into Instruction Register

Decode

Value stored in Register 5

Second instruction loaded into Instruction Register

Decode

Value stored in Register 6

Third instruction loaded into Instruction Register

```
opcode: 00000
ALU operation
opcode: 00000
ALU operation
time = 170, clk = 1, PC_addr = 03, IR_out: 0756,  Memory_out: 3333, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: xxxx, Mem_in:1111
time = 180, clk = 0, PC_addr = 03, IR_out: 0756,  Memory_out: 3333, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: xxxx, Mem_in:1111
State: S_afterALU
time = 190, clk = 1, PC_addr = 03, IR_out: 0756,  Memory_out: 3333, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:1111
time = 200, clk = 0, PC_addr = 03, IR_out: 0756,  Memory_out: 3333, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:1111
State: S_fetch
State: S_fetch
State: S_fetch
time = 210, clk = 1, PC_addr = 03, IR_out: a7cb,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
time = 220, clk = 0, PC_addr = 03, IR_out: a7cb,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
opcode: 01010
SW operation
opcode: 01010
SW operation
time = 230, clk = 1, PC_addr = 04, IR_out: a7cb,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:3333
time = 240, clk = 0, PC_addr = 04, IR_out: a7cb,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:3333
State: S_afterSW
State: S_afterSW
time = 250, clk = 1, PC_addr = 04, IR_out: a7cb,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
time = 260, clk = 0, PC_addr = 04, IR_out: a7cb,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
State: S_fetch
State: S_fetch
time = 270, clk = 1, PC_addr = 04, IR_out: 88fa,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
time = 280, clk = 0, PC_addr = 04, IR_out: 88fa,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
opcode: 01000
LI operation
time = 290, clk = 1, PC_addr = 05, IR_out: 88fa,  Memory_out: fffa, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
time = 300, clk = 0, PC_addr = 05, IR_out: 88fa,  Memory_out: fffa, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
State: S_afterLI
time = 310, clk = 1, PC_addr = 05, IR_out: 88fa,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
time = 320, clk = 0, PC_addr = 05, IR_out: 88fa,  Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
```

Value stored in Register 7

Forth instruction loaded into Instruction Register

Decode

Value stored back to the memory 203

Fifth instruction loaded into Instruction Register

Decode

```
State: S_fetch
State: S_fetch
time = 330, clk = 1, PC_addr = 05, IR_out: 1485, Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
time = 340, clk = 0, PC_addr = 05, IR_out: 1485, Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
opcode: 00001
ALU operation
opcode: 00001
ALU operation
time = 350, clk = 1, PC_addr = 06, IR_out: 1485, Memory_out: eee9, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:ffffa
time = 360, clk = 0, PC_addr = 06, IR_out: 1485, Memory_out: eee9, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: xxxx, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:ffffa
State: S_afterALU
time = 370, clk = 1, PC_addr = 06, IR_out: 1485, Memory_out: 110b, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:ffffa
time = 380, clk = 0, PC_addr = 06, IR_out: 1485, Memory_out: 110b, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:ffffa
State: S_fetch
State: S_fetch
State: S_fetch
time = 390, clk = 1, PC_addr = 06, IR_out: a4cc, Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
time = 400, clk = 0, PC_addr = 06, IR_out: a4cc, Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
opcode: 01010
SW operation
opcode: 01010
SW operation
time = 410, clk = 1, PC_addr = 07, IR_out: a4cc, Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:eee9
time = 420, clk = 0, PC_addr = 07, IR_out: a4cc, Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:eee9
State: S_afterSW
State: S_afterSW
time = 430, clk = 1, PC_addr = 07, IR_out: a4cc, Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
time = 440, clk = 0, PC_addr = 07, IR_out: a4cc, Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
State: S_fetch
State: S_fetch
State: S_fetch
time = 450, clk = 1, PC_addr = 07, IR_out: 7370, Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:3333
time = 460, clk = 0, PC_addr = 07, IR_out: 7370, Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:3333
opcode: 00111
ALU operation
```

Sixth instruction loaded into Instruction Register

Decode

Seventh instruction loaded into Instruction Register

Decode

Value stored back to the memory 204

Eighth instruction loaded into Instruction Register

```
time = 470, clk = 1, PC_addr = 08, IR_out: 7370, Memory_out: 1999, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:3333
time = 480, clk = 0, PC_addr = 08, IR_out: 7370, Memory_out: 1999, Register2 inner value: xxxx,Register3 inner value:xxxx, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:3333
State: S_afterALU
time = 490, clk = 1, PC_addr = 08, IR_out: 7370, Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:3333
time = 500, clk = 0, PC_addr = 08, IR_out: 7370, Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:3333
State: S_fetch
State: S_fetch
State: S_fetch
time = 510, clk = 1, PC_addr = 08, IR_out: 4234, Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
time = 520, clk = 0, PC_addr = 08, IR_out: 4234, Memory_out: xxxx, Register2 inner value: xxxx,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
opcode: 00100
ALU operation
opcode: 00100
ALU operation
time = 530, clk = 1, PC_addr = 09, IR_out: 4234, Memory_out: f770, Register2 inner value: xxxx,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:1999
time = 540, clk = 0, PC_addr = 09, IR_out: 4234, Memory_out: f770, Register2 inner value: xxxx,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:1999
State: S_afterALU
time = 550, clk = 1, PC_addr = 09, IR_out: 4234, Memory_out: 0882, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:1999
time = 560, clk = 0, PC_addr = 09, IR_out: 4234, Memory_out: 0882, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:1999
State: S_fetch
State: S_fetch
State: S_fetch
time = 570, clk = 1, PC_addr = 09, IR_out: a2cd, Memory_out: xxxx, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
time = 580, clk = 0, PC_addr = 09, IR_out: a2cd, Memory_out: xxxx, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
opcode: 01010
SW operation
opcode: 01010
SW operation
time = 590, clk = 1, PC_addr = 0a, IR_out: a2cd, Memory_out: xxxx, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:f770
time = 600, clk = 0, PC_addr = 0a, IR_out: a2cd, Memory_out: xxxx, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:f770
State: S_afterSW
State: S_afterSW
time = 610, clk = 1, PC_addr = 0a, IR_out: a2cd, Memory_out: xxxx, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
time = 620, clk = 0, PC_addr = 0a, IR_out: a2cd, Memory_out: xxxx, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
State: S_fetch
State: S_fetch
```

Ninth instruction loaded into Instruction Register

Decode

Tenth instruction loaded into Instruction Register

Decode

Value stored back to the memory 205

```
time = 630, clk = 1, PC_addr = 0a, IR_out: xxxx, Memory_out: xxxx, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
time = 640, clk = 0, PC_addr = 0a, IR_out: xxxx, Memory_out: xxxx, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
opcode: 0xxxx
Invalid opcode
time = 650, clk = 1, PC_addr = 0b, IR_out: xxxx, Memory_out: xxxx, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
time = 660, clk = 0, PC_addr = 0b, IR_out: xxxx, Memory_out: xxxx, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
State: S_stay
time = 670, clk = 1, PC_addr = 0b, IR_out: xxxx, Memory_out: xxxx, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
time = 680, clk = 0, PC_addr = 0b, IR_out: xxxx, Memory_out: xxxx, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
```

Stay

```
time = 990, clk = 1, PC_addr = 0b, IR_out: xxxx, Memory_out: xxxx, Register2 inner value: f770,Register3 inner value:1999, Register4 inner value: eee9, Register5 inner value: 1111, Register6 inner value: 2222, Register7 inner value: 3333, Mem_in:xxxx
Value at memory location 203: 3333
Value at memory location 204: eee9
Value at memory location 205: f770
```

Final Value in Memory 203, 204, and 205.

```
Value at memory location 203: 3333
Value at memory location 204: eee9
Value at memory location 205: f770
```

Testbench and Simulation for the *Top Module*:

```
module finalproject_sevensegment_tb();
    reg clk_i;
    reg [15:0] SW;
    wire [15:0] LED;
    wire [7:0] disp_an_o;
    wire [7:0] disp_seg_o;

    fp_test3 uut (
        .clk_i(clk_i),
        .SW(SW),
        .LED(LED),
        .disp_an_o(disp_an_o),
        .disp_seg_o(disp_seg_o)
    );

        always begin
        #5 clk_i = ~clk_i;
    end

    initial begin
        clk_i = 0;
        SW = 16'b0;
        SW[0] = 1;
        #10;
        SW[0] = 0;
        #500;
        $display("mem201: %h, mem202: %h, mem203: %h, mem204: %h, mem205: %h",
                uut.risc1.dmem1.memory[201], uut.risc1.dmem1.memory[202], uut.risc1.dmem1.memory[203], uut.risc1.dmem1.memory[204], uut.risc1.dmem1.memory[205]);
        $finish;
    end

    initial begin
        $monitor("mem201: %h, mem202: %h, mem203: %h, mem204: %h, mem205: %h",
                uut.risc1.dmem1.memory[201], uut.risc1.dmem1.memory[202], uut.risc1.dmem1.memory[203], uut.risc1.dmem1.memory[204], uut.risc1.dmem1.memory[205]);
    end
endmodule
```

```
# run 1000ns
State: Default, resetting to S_initial
State: S_initial
mem201: 1111, mem202: 2222, mem203: xxxx, mem204: xxxx, mem205: xxxx
State: S_fetch
opcode: 0xxxx
Invalid opcode
opcode: 01001
LW operation
State: S_after Execution
State: S_fetch
opcode: 01001
LW operation
State: S_after Execution
State: S_fetch
State: S_fetch
opcode: 00000
ALU operation
opcode: 00000
ALU operation
State: S_afterALU
State: S_after Execution
State: S_fetch
State: S_fetch
State: S_fetch
opcode: 01010
SW operation
opcode: 01010
SW operation
State: S_after Execution
State: S_after Execution
mem201: 1111, mem202: 2222, mem203: 3333, mem204: xxxx, mem205: xxxx
```

```
State: S_fetch
State: S_fetch
opcode: 01000
LI operation
State: S_after Execution
State: S_fetch
State: S_fetch
opcode: 00001
ALU operation
opcode: 00001
ALU operation
State: S_afterALU
State: S_after Execution
State: S_fetch
State: S_fetch
State: S_fetch
opcode: 01010
SW operation
opcode: 01010
SW operation
State: S_after Execution
State: S_after Execution
mem201: 1111, mem202: 2222, mem203: 3333, mem204: eee9, mem205: xxxx
```

```
State: S_fetch
State: S_fetch
State: S_fetch
opcode: 00111
ALU operation
State: S_afterALU
State: S_after Execution
State: S_fetch
State: S_fetch
State: S_fetch
opcode: 00100
ALU operation
opcode: 00100
ALU operation
State: S_afterALU
State: S_after Execution
State: S_fetch
State: S_fetch
State: S_fetch
opcode: 01010
SW operation
opcode: 01010
SW operation
State: S_after Execution
State: S_after Execution
mem201: 1111, mem202: 2222, mem203: 3333, mem204: eee9, mem205: f770
State: S_fetch
State: S_fetch
opcode: 0xxxx
Invalid opcode
State: S_stay
mem201: 1111, mem202: 2222, mem203: 3333, mem204: eee9, mem205: f770
$finish called at time : 500 ns : File "C:/Xilinx/project/FPGA_HW/fp_
```

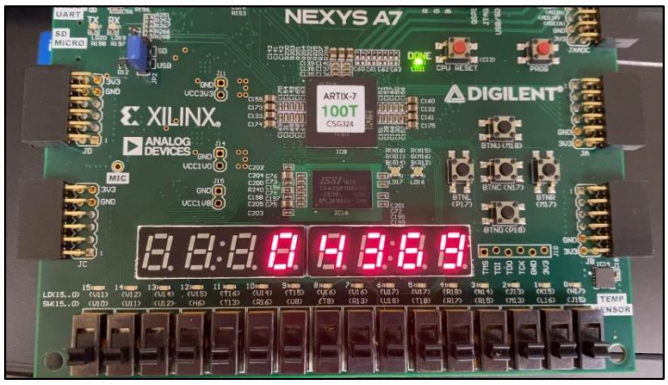
Verification:

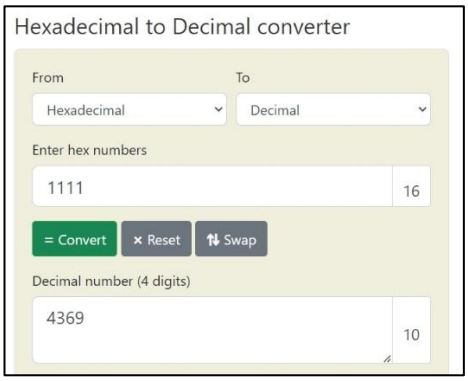
1. LW 5 201 1001_0101_1100_1001
2. LW 6 202 1001_0110_1100_1010
3. ADD 7 5 6 0000_0111_0101_0110
4. SW 7 203 1010_0111_1100_1011
5. LI 8 250 1000_1000_1111_1010
6. SUB 4 8 5 0001_0100_1000_0101
7. SW 4 204 1010_0100_1100_1100
8. SRA 3 7 0111_0011_0111_0000
9. XOR 2 3 4 0100_0010_0011_0100
10. SW 2 205 1010_0010_1100_1101

1. Load memory 201: 1111 (Initial Value).
2. Load memory 202: 2222 (Initial Value).
3. 1111 add 2222 = 3333 and store in Register 7.
4. Store 3333 back to Memory 203.
5. 1111111111111010(sign-extension of 250) being store in Register 8.
6. 1111111111111010-0001000100010001 = 1110111011101001(eee9) and store in Register 4.
7. Store eee9 back to Memory 204.
8. 0011001100110011(3333hex in binary) shift right and became 0001100110011001(1999 in hex) then store to Register 3.
9. 0001100110011001 XOR 1110111011101001 = 1111011101110000(f770 in hex) and store into Register 2.
10. Store f770 back to Memory 205.

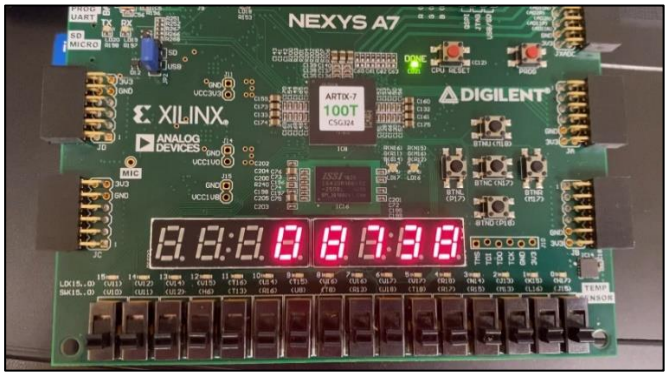
On-board synthesis picture:

Memory [201]: 1111(hex.) → 4369(dec.)

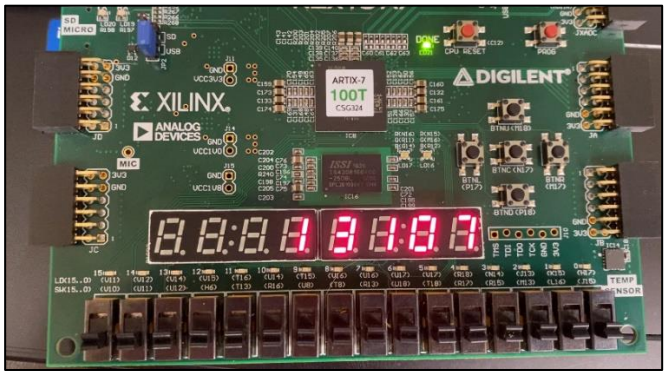




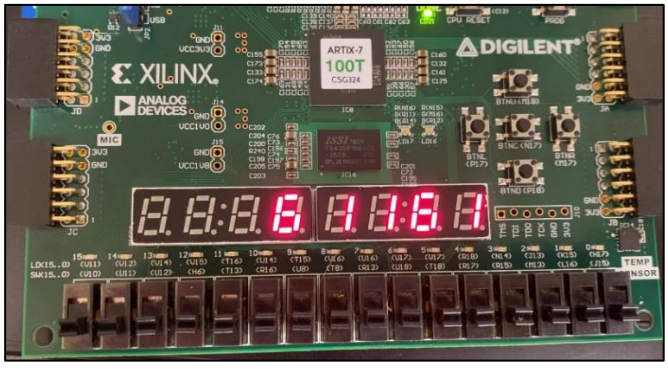
Memory [202]: 2222(hex.) → 8738(dec.)



Memory [203]: 3333(hex.) → 13107(dec.)



Memory [204]: eee9(hex.) → 61161(dec.)



Memory [205]: f770(hex.) → 63344(dec.)