2022 Spring COMP311: Logic Circuit Design

Final Project

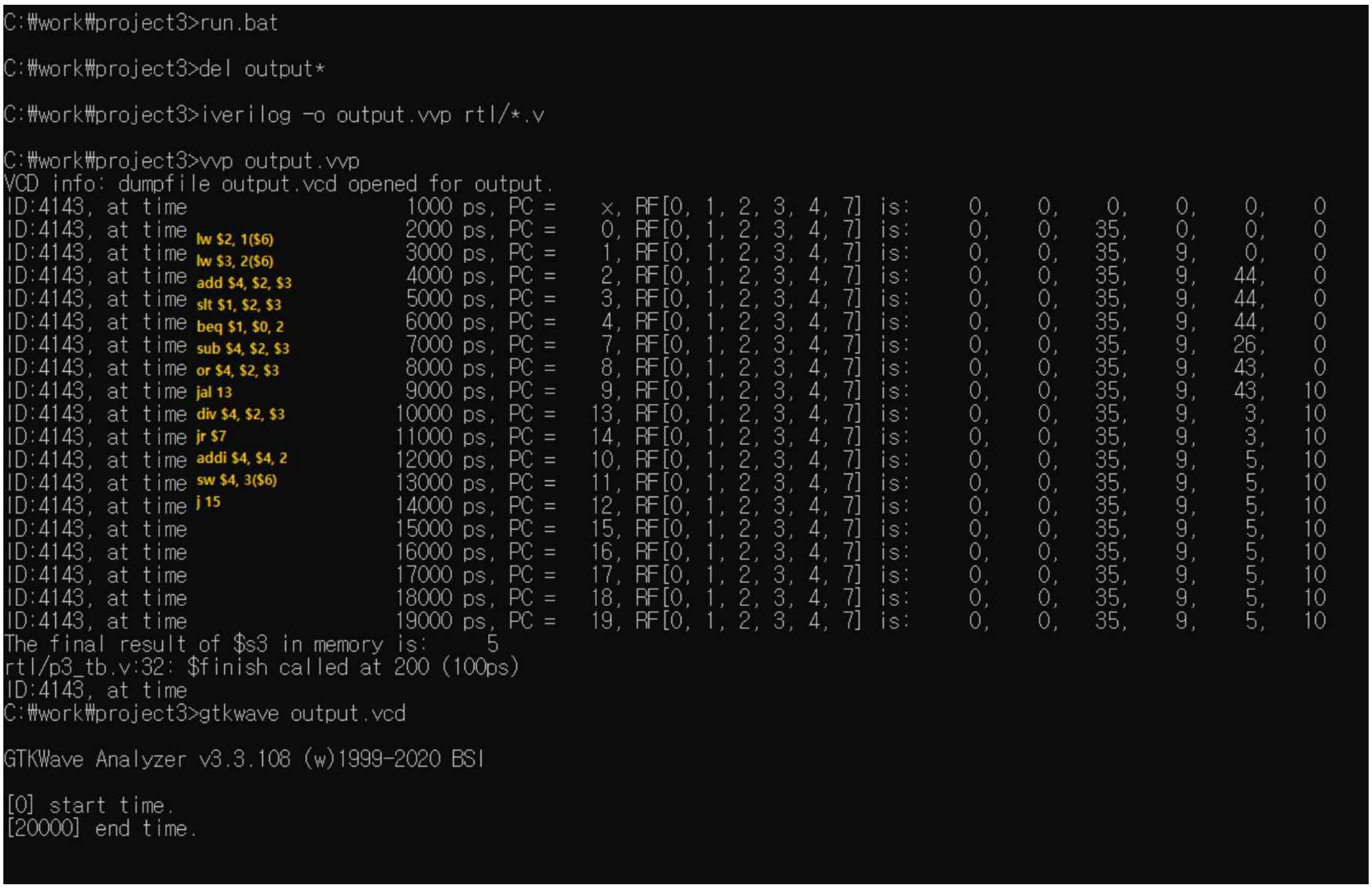
Student ID / Name: 2018114143 권명섭

Final Project check list (each 10 points)

	Answers
1. First value of $s1, $s2	($s1,2): 35,9
2. First value of $s3	44
3. Final $s3 value	5
4. What are the instructions implemented, but not used?	mul,slti
5. What is the final PC # processed?	#12
6. What are the PC #s that have not been processed?	#5,6
7. Parameterized MUX design	Y
8. load memory with instructions	Y
9. Implemented result stored in the register after PC 0000	Y
10. Implemented result after beq	Y
11. Implemented result after first sw is done	Y
12. When run is complete: (1) final $s3 value,	Y
13. When run is complete: (2) registers have intended results	Y

목차

2.결과 화면



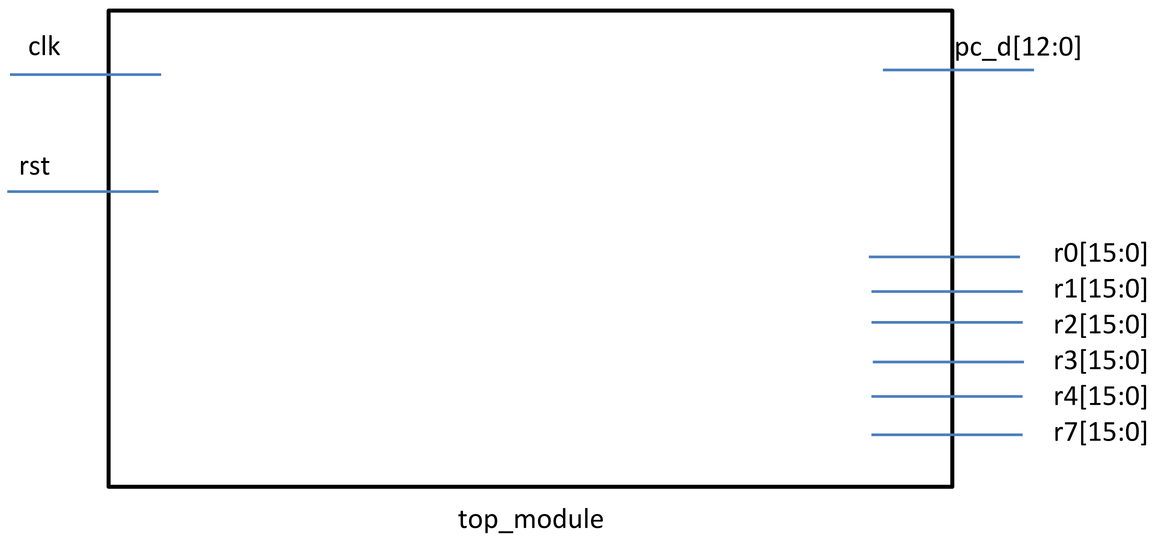
```
C:₩work₩project3>run.bat

C:₩work₩project3>del output*

C:₩work₩project3>iverilog -o output.vvp rtl/*.v

C:₩work₩project3>vvp output.vvp
VCD info: dumpfile output.vcd opened for output.
ID:4143, at time                     1000 ps, PC =    x, RF[0, 1, 2, 3, 4, 7] is:     0,    0,    0,    0,    0,    0
ID:4143, at time lw $2, 1($6)        2000 ps, PC =    0, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    0,    0,    0
ID:4143, at time lw $3, 2($6)        3000 ps, PC =    1, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,    0,    0
ID:4143, at time add $4, $2, $3      4000 ps, PC =    2, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,   44,    0
ID:4143, at time slt $1, $2, $3      5000 ps, PC =    3, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,   44,    0
ID:4143, at time beq $1, $0, 2       6000 ps, PC =    4, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,   44,    0
ID:4143, at time sub $4, $2, $3      7000 ps, PC =    7, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,   26,    0
ID:4143, at time or $4, $2, $3       8000 ps, PC =    8, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,   43,    0
ID:4143, at time jal 13              9000 ps, PC =    9, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,   43,   10
ID:4143, at time div $4, $2, $3     10000 ps, PC =   13, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,    3,   10
ID:4143, at time jr $7              11000 ps, PC =   14, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,    3,   10
ID:4143, at time addi $4, $4, 2     12000 ps, PC =   10, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,    5,   10
ID:4143, at time sw $4, 3($6)       13000 ps, PC =   11, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,    5,   10
ID:4143, at time j 15               14000 ps, PC =   12, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,    5,   10
ID:4143, at time                    15000 ps, PC =   15, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,    5,   10
ID:4143, at time                    16000 ps, PC =   16, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,    5,   10
ID:4143, at time                    17000 ps, PC =   17, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,    5,   10
ID:4143, at time                    18000 ps, PC =   18, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,    5,   10
ID:4143, at time                    19000 ps, PC =   19, RF[0, 1, 2, 3, 4, 7] is:     0,    0,   35,    9,    5,   10
The final result of $s3 in memory is:     5
rtl/p3_tb.v:32: $finish called at 200 (100ps)
ID:4143, at time
C:₩work₩project3>gtkwave output.vcd

GTKWave Analyzer v3.3.108 (w)1999-2020 BSI

[0] start time.
[20000] end time.
```

노란색 글자는 그림판으로 추가한 것이며, 해당 pc의 코드를 나타냅니다.

제 손코딩과 똑같은 결과를 보였습니다. 해당하는 순간에 실행되는 pc를 보여주며, 그 실행이 된 후의 레지스터0~3,4,7의 값입니다. 마지막 $s3(r4)도 예상한 결과인 5가 나옴을 알 수 있습니다.

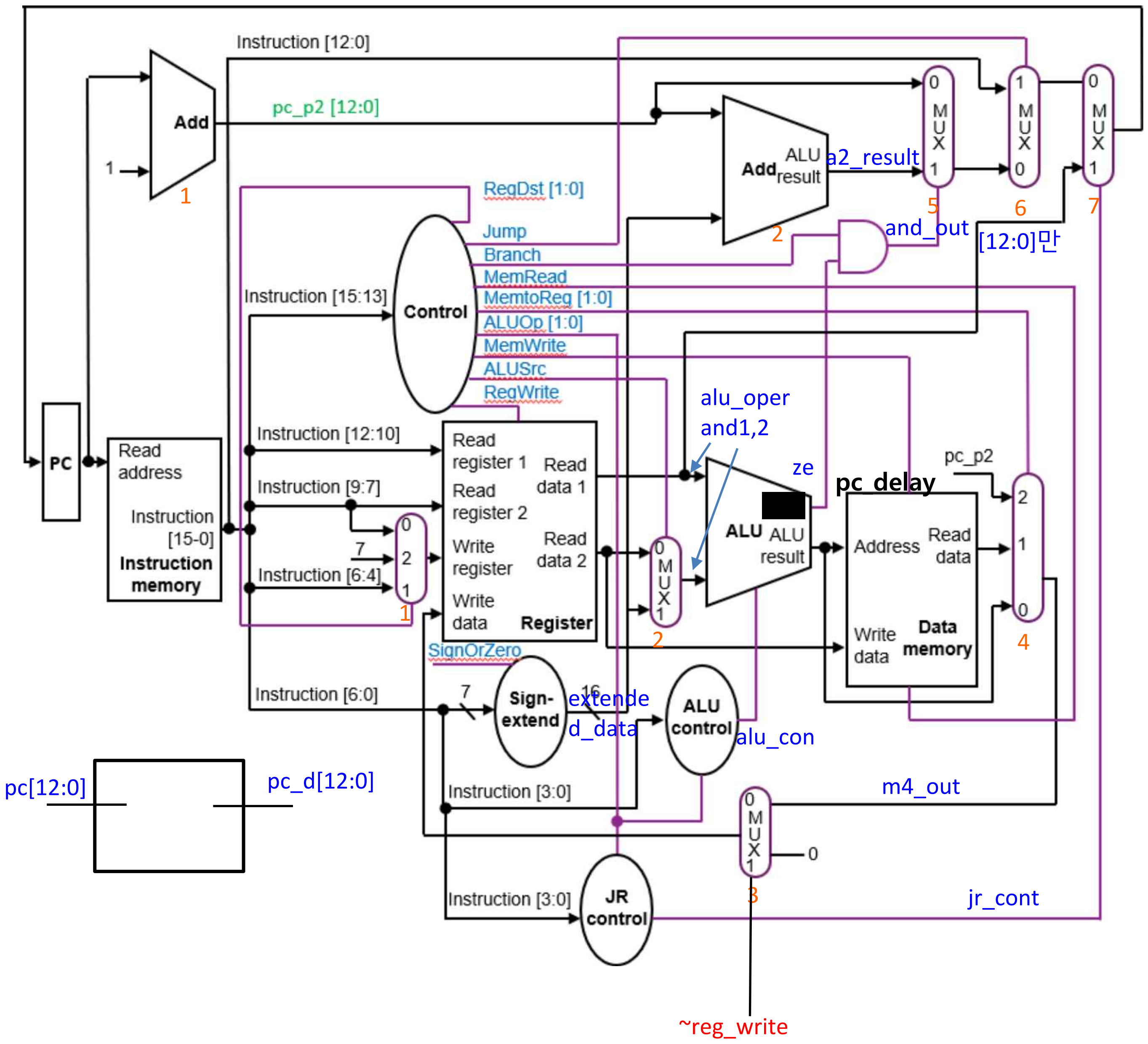
3. 모듈 설명

3-1 top module



모든 하위 모듈들이 속해있는 top 모듈입니다. tb에서는 이 모듈만 호출하며, 이 모듈이 다른 모든 모듈을 가지고 있습니다. cmd창에 호출을 위해서 레지스터 값들이 output으로 있습니다. rst은 0ns에서 1ns에서만 0입니다.

3-2 top module 내부

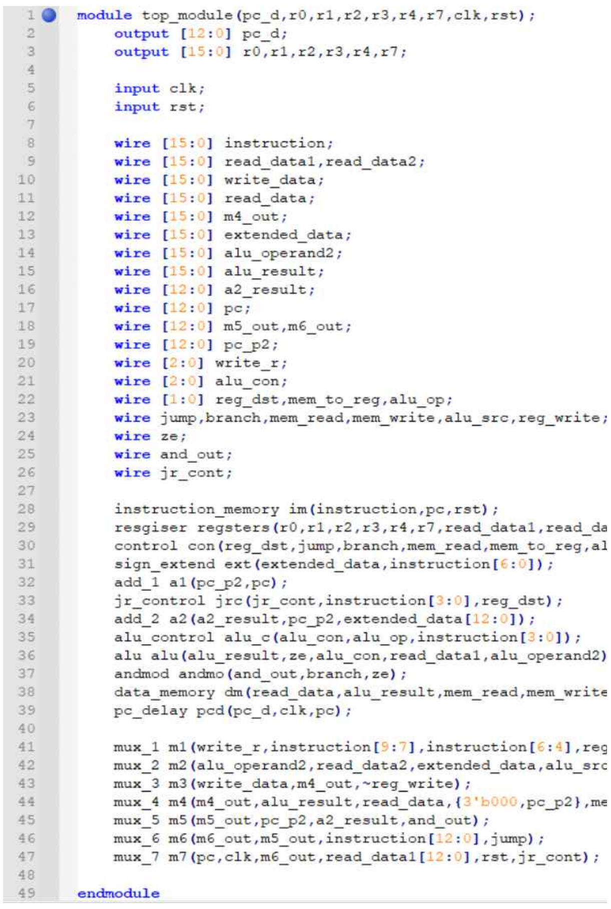


이름이 없는 선들이 있어서 제가 붙힌 이름들을 파란색으로 표시했습니다. mux밑에는 몇번 mux인지 적어놨으며, 따로 추가로 표기 안 한것들은 기존에 이름대로 썼습니다.
3번 mux에 sel이 뭔지 안 나와있어서 제가 생각한 결과 ~reg_write를 쓰면 write_data를 안쓸때 0이 되므로 그렇게 했습니다.
register와 7번 mux는 posedge clk에서 동작하며, control은 instruction이 바뀔때 동작하고, 나머지는 항상 동작합니다.(always@(*)) 이렇게 하면 instruction을 실행하고 난 후 다음pc가 pc에 바로 반영됩니다. 이 때, tb의 출력이 "실행한 pc, 그 instruction후 레지스터"로 뛰우기 위해 이전 pc를 보내야 해서 pc_delay모듈을 만들었습니다.
instruction memory ,data memory,resister module들은 모듈 내부에서 if문을 이용해 reset이 0일때 초기화하도록 했습니다. rst이 0일때 7번 mux는 출력(pc)을 0을 내게 해서 pc를 0으로 초기화 했습니다.
각각의 모듈들은 알려진 기능을 그대로 수행하므로 제가 설명을 할필요는 없다고 생각해서 뺐습니다.

3-3 design conditions 충족

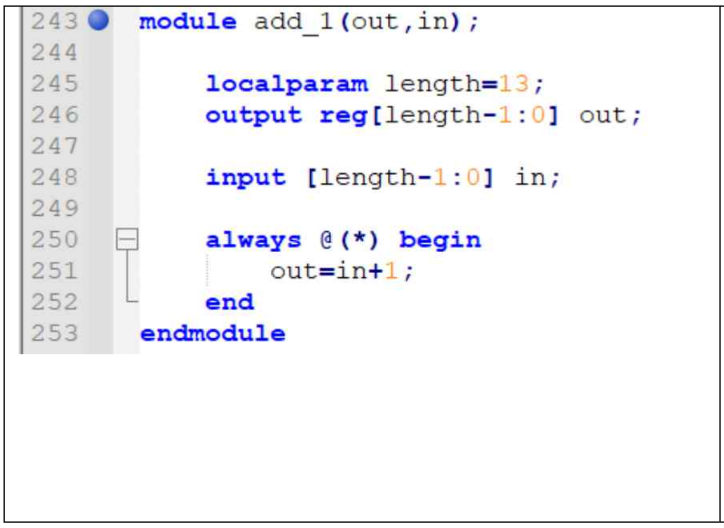
과제에서 9.design conditions를 만족해야 한다고 되어 있어서 그것을 만족하는지를 설명하겠습니다. synthesizable(4번째)과 레지스터와 FFs,memory는 posedge triggered되야 한다.(7번째)빼고는 다 만족했습니다.

1. 탑 모듈은 하위 모듈로만 구성

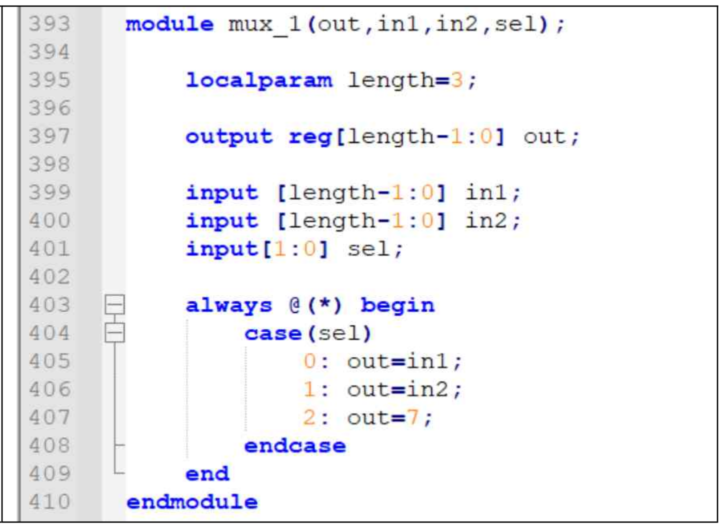


```verilog
module top_module(pc_d,r0,r1,r2,r3,r4,r7,clk,rst);
    output [12:0] pc_d;
    output [15:0] r0,r1,r2,r3,r4,r7;

    input clk;
    input rst;

    wire [15:0] instruction;
    wire [15:0] read_data1,read_data2;
    wire [15:0] write_data;
    wire [15:0] read_data;
    wire [15:0] m4_out;
    wire [15:0] extended_data;
    wire [15:0] alu_operand2;
    wire [15:0] alu_result;
    wire [12:0] a2_result;
    wire [12:0] pc;
    wire [12:0] m5_out,m6_out;
    wire [12:0] pc_p2;
    wire [2:0] write_r;
    wire [2:0] alu_con;
    wire [1:0] reg_dst,mem_to_reg,alu_op;
    wire jump,branch,mem_read,mem_write,alu_src,reg_write;
    wire ze;
    wire and_out;
    wire jr_cont;

    instruction_memory im(instruction,pc,rst);
    resgiser regsters(r0,r1,r2,r3,r4,r7,read_data1,read_da
    control con(reg_dst,jump,branch,mem_read,mem_to_reg,al
    sign_extend ext(extended_data,instruction[6:0]);
    add_1 a1(pc_p2,pc);
    jr_control jrc(jr_cont,instruction[3:0],reg_dst);
    add_2 a2(a2_result,pc_p2,extended_data[12:0]);
    alu_control alu_c(alu_con,alu_op,instruction[3:0]);
    alu alu(alu_result,ze,alu_con,read_data1,alu_operand2)
    andmod andmo(and_out,branch,ze);
    data_memory dm(read_data,alu_result,mem_read,mem_write
    pc_delay pcd(pc_d,clk,pc);

    mux_1 m1(write_r,instruction[9:7],instruction[6:4],reg
    mux_2 m2(alu_operand2,read_data2,extended_data,alu_src
    mux_3 m3(write_data,m4_out,~reg_write);
    mux_4 m4(m4_out,alu_result,read_data,{3'b000,pc_p2},me
    mux_5 m5(m5_out,pc_p2,a2_result,and_out);
    mux_6 m6(m6_out,m5_out,instruction[12:0],jump);
    mux_7 m7(pc,clk,m6_out,read_data1[12:0],rst,jr_cont);

endmodule
```

2. adder와 mux는 parameter를 써서 설계



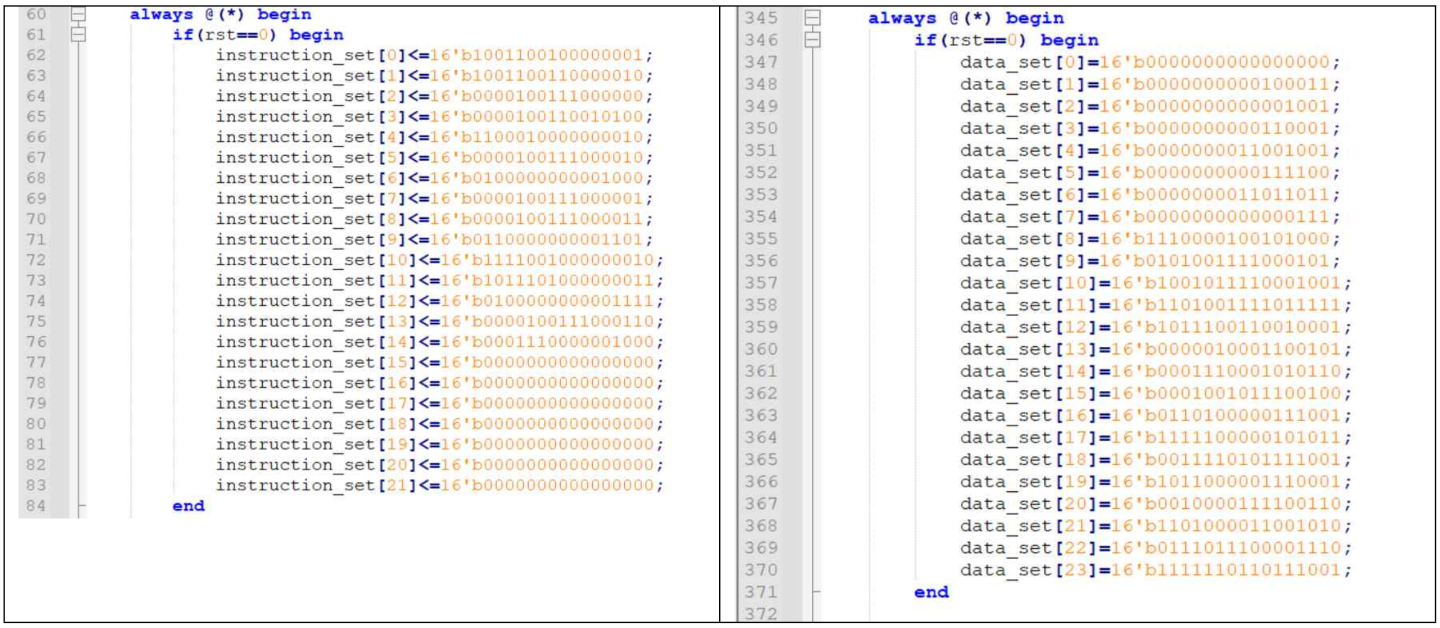
```verilog
module add_1(out,in);

    localparam length=13;
    output reg[length-1:0] out;

    input [length-1:0] in;

    always @(*) begin
        out=in+1;
    end
endmodule
```



```verilog
module mux_1(out,in1,in2,sel);

    localparam length=3;

    output reg[length-1:0] out;

    input [length-1:0] in1;
    input [length-1:0] in2;
    input[1:0] sel;

    always @(*) begin
        case(sel)
            0: out=in1;
            1: out=in2;
            2: out=7;
        endcase
    end
endmodule
```

둘 다 길이를 localparameter를 써서 했습니다.

3. 메모리를 가장 편리한 방법대로 초기화



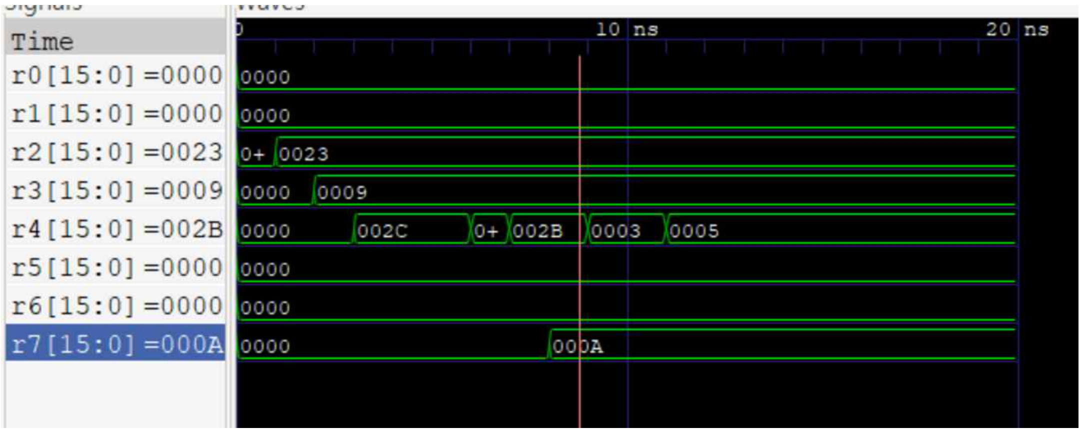
```
    always @(*) begin
        if(rst==0) begin
            instruction_set[0]<=16'b1001100100000001;
            instruction_set[1]<=16'b1001100110000010;
            instruction_set[2]<=16'b0000100111000000;
            instruction_set[3]<=16'b0000100110010100;
            instruction_set[4]<=16'b1100010000000010;
            instruction_set[5]<=16'b0000100111000010;
            instruction_set[6]<=16'b0100000000001000;
            instruction_set[7]<=16'b0000100111000001;
            instruction_set[8]<=16'b0000100111000011;
            instruction_set[9]<=16'b0110000000001101;
            instruction_set[10]<=16'b1111001000000010;
            instruction_set[11]<=16'b1011101000000011;
            instruction_set[12]<=16'b0100000000001111;
            instruction_set[13]<=16'b0000100111000110;
            instruction_set[14]<=16'b0001110000001000;
            instruction_set[15]<=16'b0000000000000000;
            instruction_set[16]<=16'b0000000000000000;
            instruction_set[17]<=16'b0000000000000000;
            instruction_set[18]<=16'b0000000000000000;
            instruction_set[19]<=16'b0000000000000000;
            instruction_set[20]<=16'b0000000000000000;
            instruction_set[21]<=16'b0000000000000000;
        end
```

```
    always @(*) begin
        if(rst==0) begin
            data_set[0]=16'b0000000000000000;
            data_set[1]=16'b0000000000100011;
            data_set[2]=16'b0000000000001001;
            data_set[3]=16'b0000000000110001;
            data_set[4]=16'b0000000011001001;
            data_set[5]=16'b0000000000111100;
            data_set[6]=16'b0000000011011011;
            data_set[7]=16'b0000000000000111;
            data_set[8]=16'b1110000100101000;
            data_set[9]=16'b0101001111000101;
            data_set[10]=16'b1001011110001001;
            data_set[11]=16'b1101001111011111;
            data_set[12]=16'b1011100110010001;
            data_set[13]=16'b0000010001100101;
            data_set[14]=16'b0001110001010110;
            data_set[15]=16'b0001001011100100;
            data_set[16]=16'b0110100000111001;
            data_set[17]=16'b1111100000101011;
            data_set[18]=16'b0011110101111001;
            data_set[19]=16'b1011000001110001;
            data_set[20]=16'b0010000111100110;
            data_set[21]=16'b1101000011001010;
            data_set[22]=16'b0111011100001110;
            data_set[23]=16'b1111110110111001;
        end
```

위에서 말했듯이 모듈내부에서 reset이 0일때 초기화 했습니다.

5. pc는 13bit며 1씩 증가 - 13bit이며, branch나 jump가 아니면 1씩 증가합니다.

6. $0, $6은 상수 0이며, $ra, $7은 0



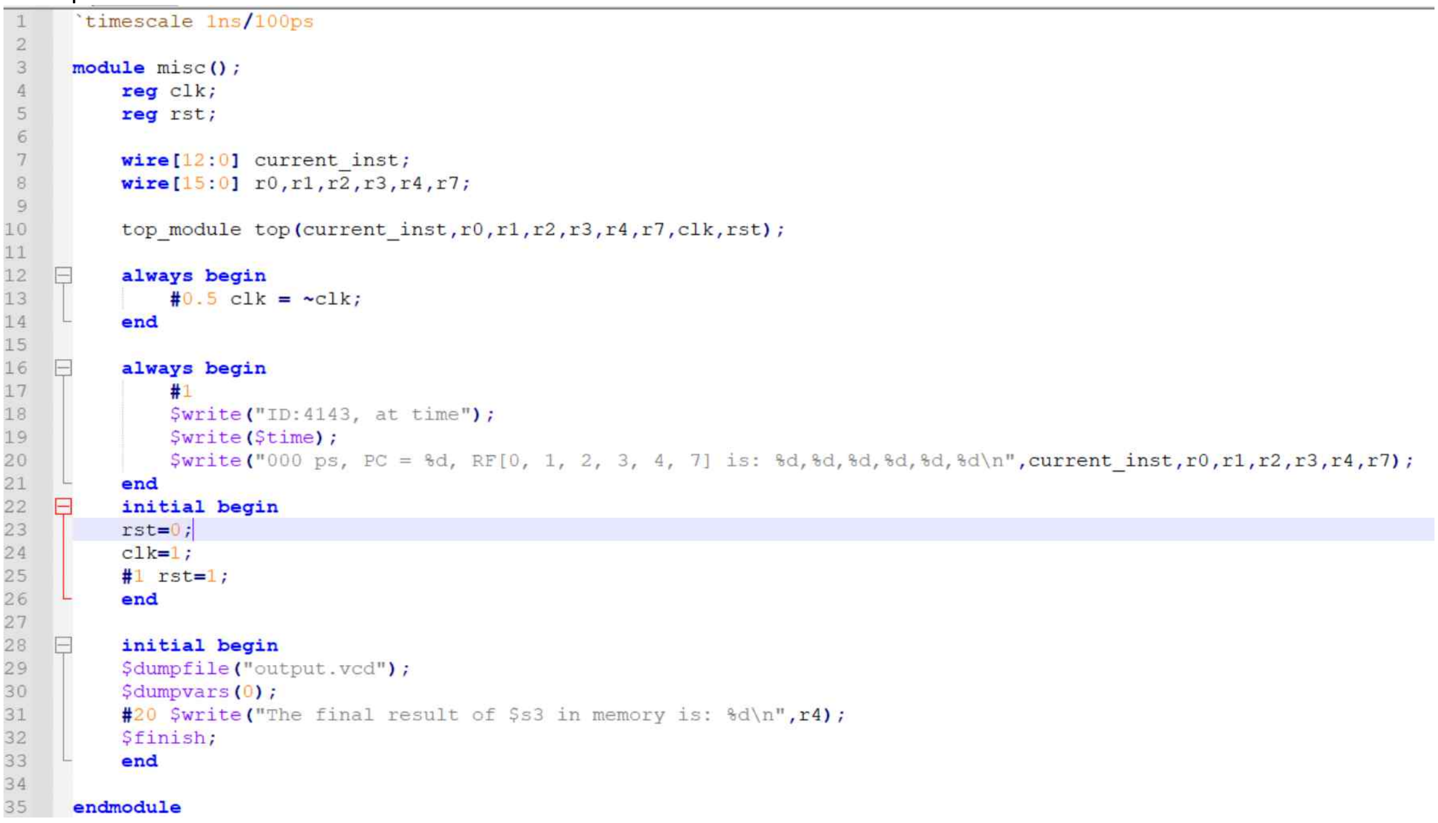
벡터 어레이는 따로 확인 할 수 없어서 아웃풋이 안나오는 r5,6도 wire 선언을 해서 결과를 띄운 것입니다.(코드에 반영되어 있습니다.) 보면 알수있듯이 맨 처음 모두 0으로 초기화됐으며, r0, r6은 상수 0입니다.

8. 모든 순차 요소는 reset=0일때 초기화- 메모리들과 레지스터,7번 mux모두 rst=0일때 초기화됩니다.

9. 인풋:rst,clk, 아웃풋: current_instr 필요 : 그것들이 있으며, 추가로 출력을 위해 레지스터들도 아웃풋으로 설정했습니다.

4. 코드

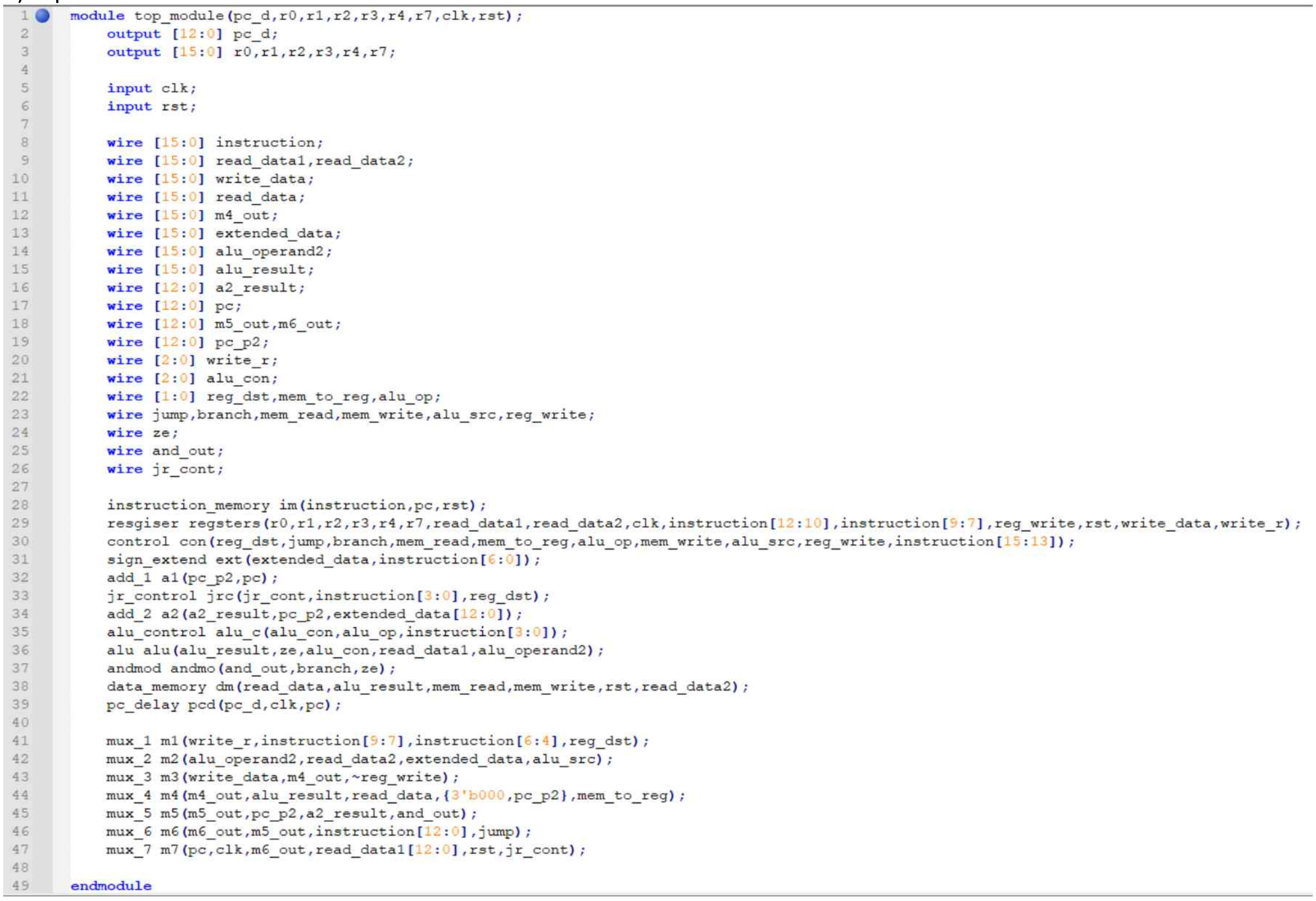
4-1 p3.tb.v



```verilog
`timescale 1ns/100ps

module misc();
    reg clk;
    reg rst;

    wire[12:0] current_inst;
    wire[15:0] r0,r1,r2,r3,r4,r7;

    top_module top(current_inst,r0,r1,r2,r3,r4,r7,clk,rst);

    always begin
        #0.5 clk = ~clk;
    end

    always begin
        #1
        $write("ID:4143, at time");
        $write($time);
        $write("000 ps, PC = %d, RF[0, 1, 2, 3, 4, 7] is: %d,%d,%d,%d,%d,%d\n",current_inst,r0,r1,r2,r3,r4,r7);
    end
    initial begin
    rst=0;
    clk=1;
    #1 rst=1;
    end

    initial begin
    $dumpfile("output.vcd");
    $dumpvars(0);
    #20 $write("The final result of $s3 in memory is: %d\n",r4);
    $finish;
    end

endmodule
```

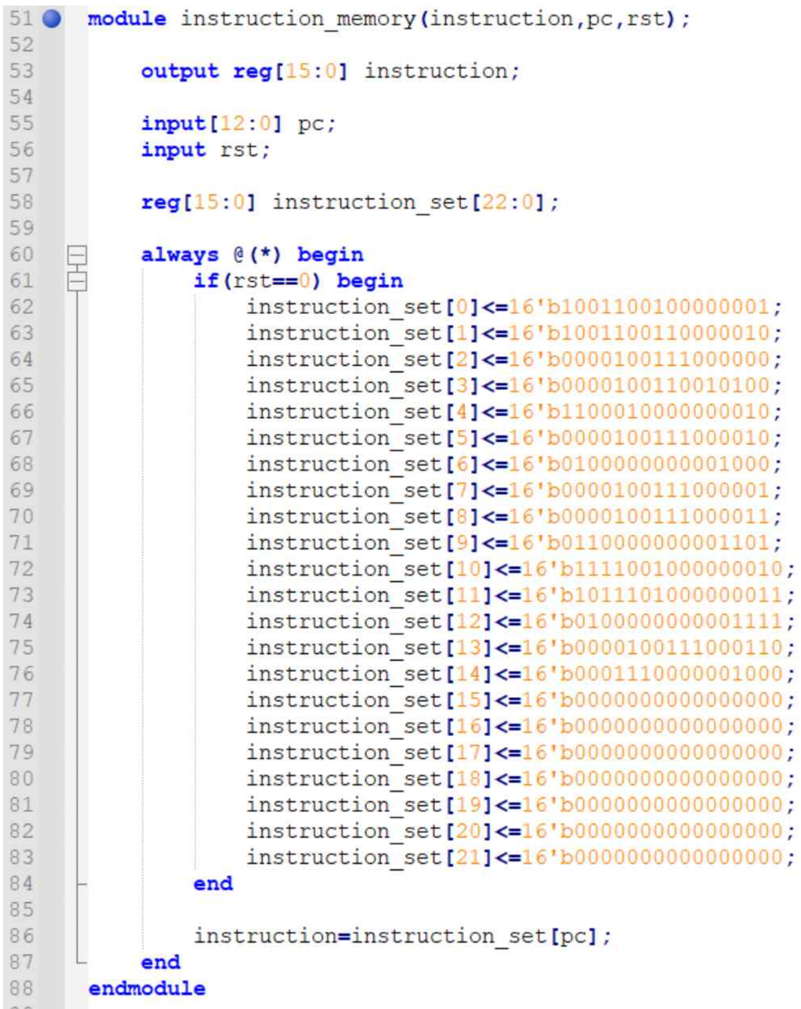
4-2 p3.v

1) top module

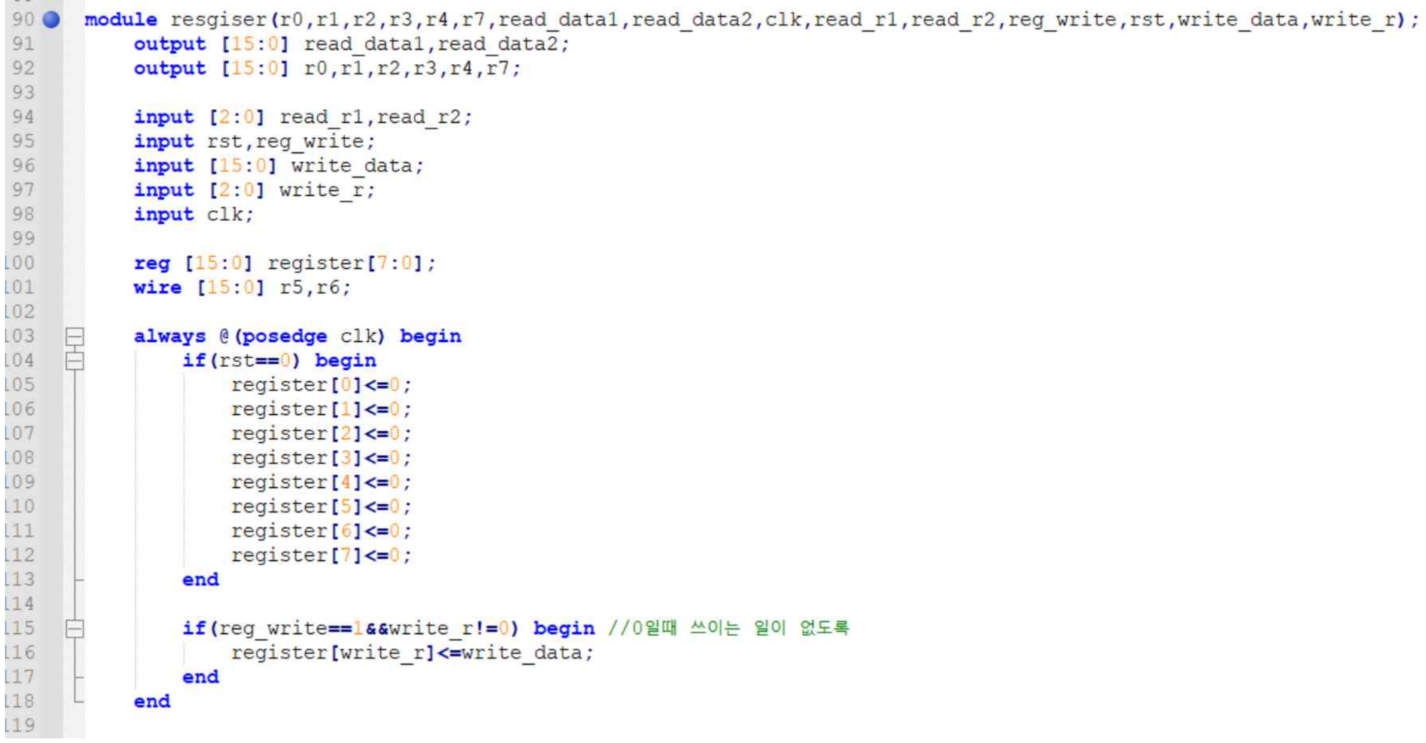


```verilog
module top_module(pc_d,r0,r1,r2,r3,r4,r7,clk,rst);
    output [12:0] pc_d;
    output [15:0] r0,r1,r2,r3,r4,r7;

    input clk;
    input rst;

    wire [15:0] instruction;
    wire [15:0] read_data1,read_data2;
    wire [15:0] write_data;
    wire [15:0] read_data;
    wire [15:0] m4_out;
    wire [15:0] extended_data;
    wire [15:0] alu_operand2;
    wire [15:0] alu_result;
    wire [12:0] a2_result;
    wire [12:0] pc;
    wire [12:0] m5_out,m6_out;
    wire [12:0] pc_p2;
    wire [2:0] write_r;
    wire [2:0] alu_con;
    wire [1:0] reg_dst,mem_to_reg,alu_op;
    wire jump,branch,mem_read,mem_write,alu_src,reg_write;
    wire ze;
    wire and_out;
    wire jr_cont;

    instruction_memory im(instruction,pc,rst);
    resgiser regsters(r0,r1,r2,r3,r4,r7,read_data1,read_data2,clk,instruction[12:10],instruction[9:7],reg_write,rst,write_data,write_r);
    control con(reg_dst,jump,branch,mem_read,mem_to_reg,alu_op,mem_write,alu_src,reg_write,instruction[15:13]);
    sign_extend ext(extended_data,instruction[6:0]);
    add_1 a1(pc_p2,pc);
    jr_control jrc(jr_cont,instruction[3:0],reg_dst);
    add_2 a2(a2_result,pc_p2,extended_data[12:0]);
    alu_control alu_c(alu_con,alu_op,instruction[3:0]);
    alu alu(alu_result,ze,alu_con,read_data1,alu_operand2);
    andmod andmo(and_out,branch,ze);
    data_memory dm(read_data,alu_result,mem_read,mem_write,rst,read_data2);
    pc_delay pcd(pc_d,clk,pc);

    mux_1 m1(write_r,instruction[9:7],instruction[6:4],reg_dst);
    mux_2 m2(alu_operand2,read_data2,extended_data,alu_src);
    mux_3 m3(write_data,m4_out,~reg_write);
    mux_4 m4(m4_out,alu_result,read_data,{3'b000,pc_p2},mem_to_reg);
    mux_5 m5(m5_out,pc_p2,a2_result,and_out);
    mux_6 m6(m6_out,m5_out,instruction[12:0],jump);
    mux_7 m7(pc,clk,m6_out,read_data1[12:0],rst,jr_cont);

endmodule
```

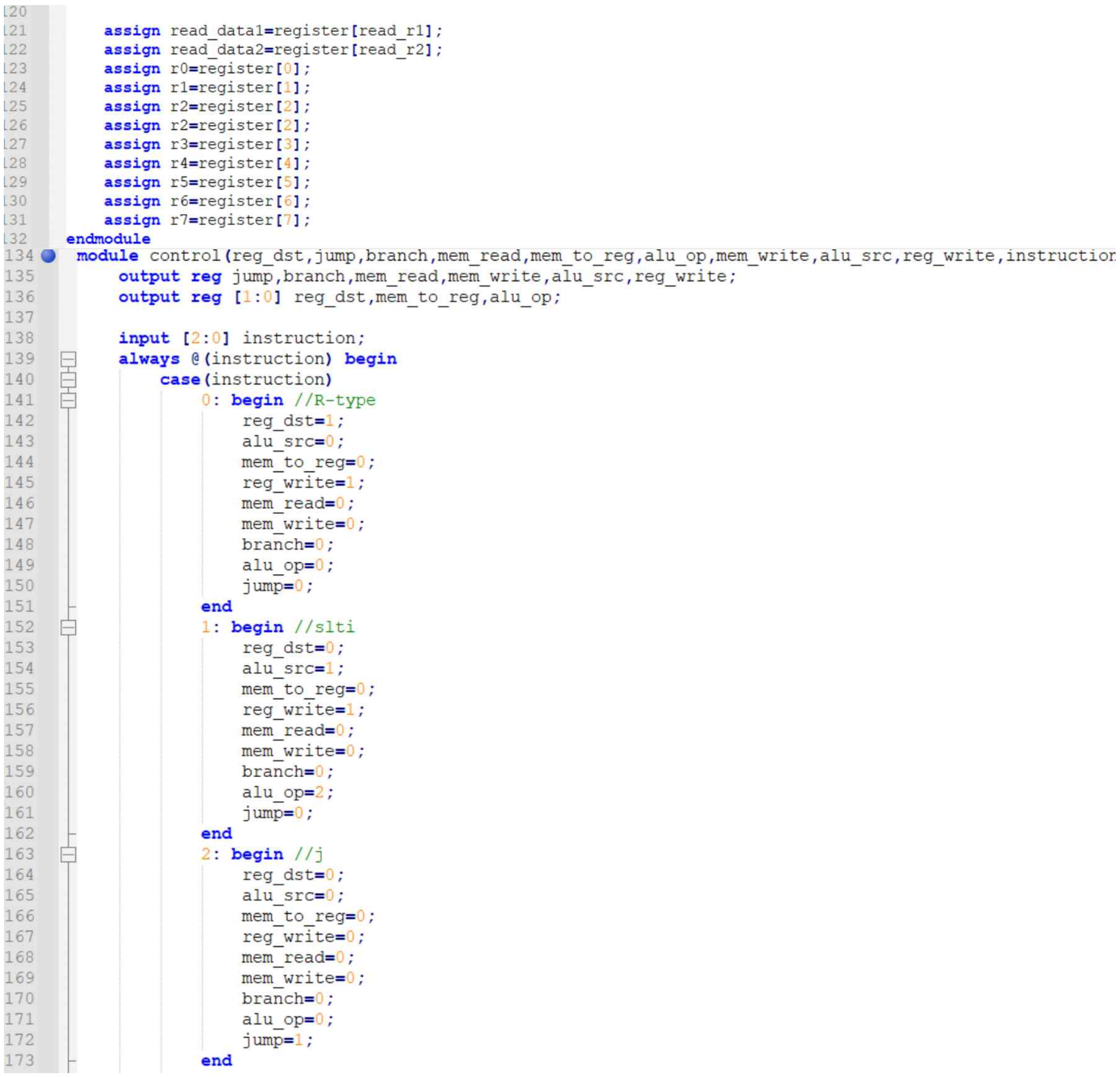
2)그 이외



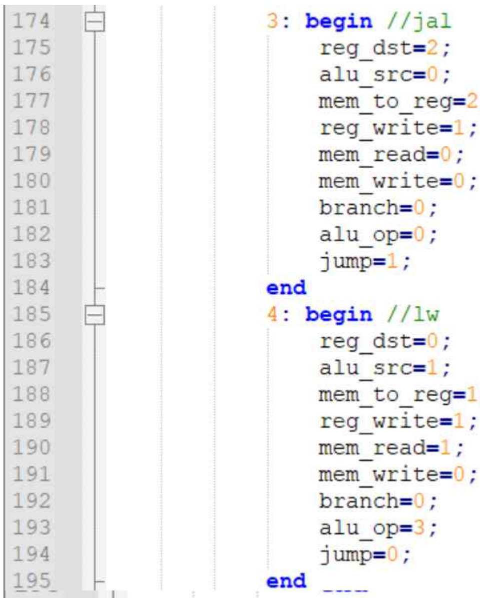
```verilog
module instruction_memory(instruction,pc,rst);

    output reg[15:0] instruction;

    input[12:0] pc;
    input rst;

    reg[15:0] instruction_set[22:0];

    always @(*) begin
        if(rst==0) begin
            instruction_set[0]<=16'b1001100100000001;
            instruction_set[1]<=16'b1001100110000010;
            instruction_set[2]<=16'b0000100111000000;
            instruction_set[3]<=16'b0000100110010100;
            instruction_set[4]<=16'b1100010000000010;
            instruction_set[5]<=16'b0000100111000010;
            instruction_set[6]<=16'b0100000000001000;
            instruction_set[7]<=16'b0000100111000001;
            instruction_set[8]<=16'b0000100111000011;
            instruction_set[9]<=16'b0110000000001101;
            instruction_set[10]<=16'b1111001000000010;
            instruction_set[11]<=16'b1011101000000011;
            instruction_set[12]<=16'b0100000000001111;
            instruction_set[13]<=16'b0000100111000110;
            instruction_set[14]<=16'b0001110000001000;
            instruction_set[15]<=16'b0000000000000000;
            instruction_set[16]<=16'b0000000000000000;
            instruction_set[17]<=16'b0000000000000000;
            instruction_set[18]<=16'b0000000000000000;
            instruction_set[19]<=16'b0000000000000000;
            instruction_set[20]<=16'b0000000000000000;
            instruction_set[21]<=16'b0000000000000000;
        end

        instruction=instruction_set[pc];
    end
endmodule
```



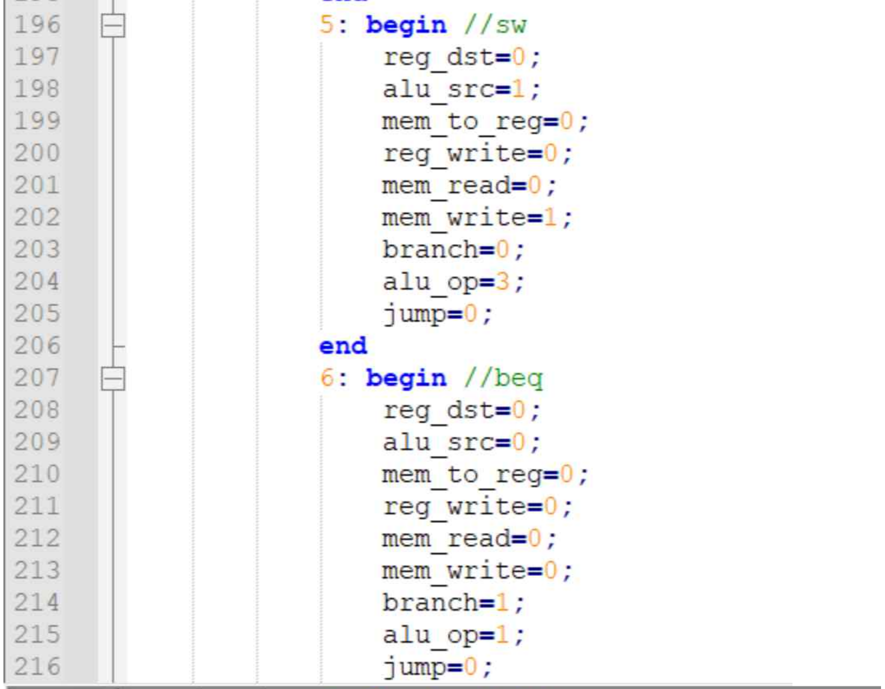
```verilog
module resgiser(r0,r1,r2,r3,r4,r7,read_data1,read_data2,clk,read_r1,read_r2,reg_write,rst,write_data,write_r);
    output [15:0] read_data1,read_data2;
    output [15:0] r0,r1,r2,r3,r4,r7;

    input [2:0] read_r1,read_r2;
    input rst,reg_write;
    input [15:0] write_data;
    input [2:0] write_r;
    input clk;

    reg [15:0] register[7:0];
    wire [15:0] r5,r6;

    always @(posedge clk) begin
        if(rst==0) begin
            register[0]<=0;
            register[1]<=0;
            register[2]<=0;
            register[3]<=0;
            register[4]<=0;
            register[5]<=0;
            register[6]<=0;
            register[7]<=0;
        end

        if(reg_write==1&&write_r!=0) begin //0일때 쓰이는 일이 없도록
            register[write_r]<=write_data;
        end
    end

```



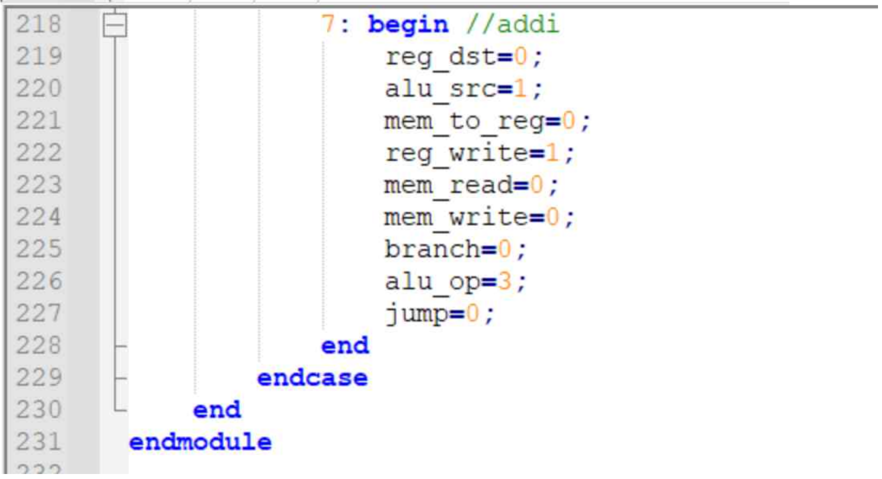
```
    assign read_data1=register[read_r1];
    assign read_data2=register[read_r2];
    assign r0=register[0];
    assign r1=register[1];
    assign r2=register[2];
    assign r2=register[2];
    assign r3=register[3];
    assign r4=register[4];
    assign r5=register[5];
    assign r6=register[6];
    assign r7=register[7];
endmodule
module control(reg_dst,jump,branch,mem_read,mem_to_reg,alu_op,mem_write,alu_src,reg_write,instructio
    output reg jump,branch,mem_read,mem_write,alu_src,reg_write;
    output reg [1:0] reg_dst,mem_to_reg,alu_op;

    input [2:0] instruction;
    always @(instruction) begin
        case(instruction)
            0: begin //R-type
                reg_dst=1;
                alu_src=0;
                mem_to_reg=0;
                reg_write=1;
                mem_read=0;
                mem_write=0;
                branch=0;
                alu_op=0;
                jump=0;
            end
            1: begin //slti
                reg_dst=0;
                alu_src=1;
                mem_to_reg=0;
                reg_write=1;
                mem_read=0;
                mem_write=0;
                branch=0;
                alu_op=2;
                jump=0;
            end
            2: begin //j
                reg_dst=0;
                alu_src=0;
                mem_to_reg=0;
                reg_write=0;
                mem_read=0;
                mem_write=0;
                branch=0;
                alu_op=0;
                jump=1;
            end
```



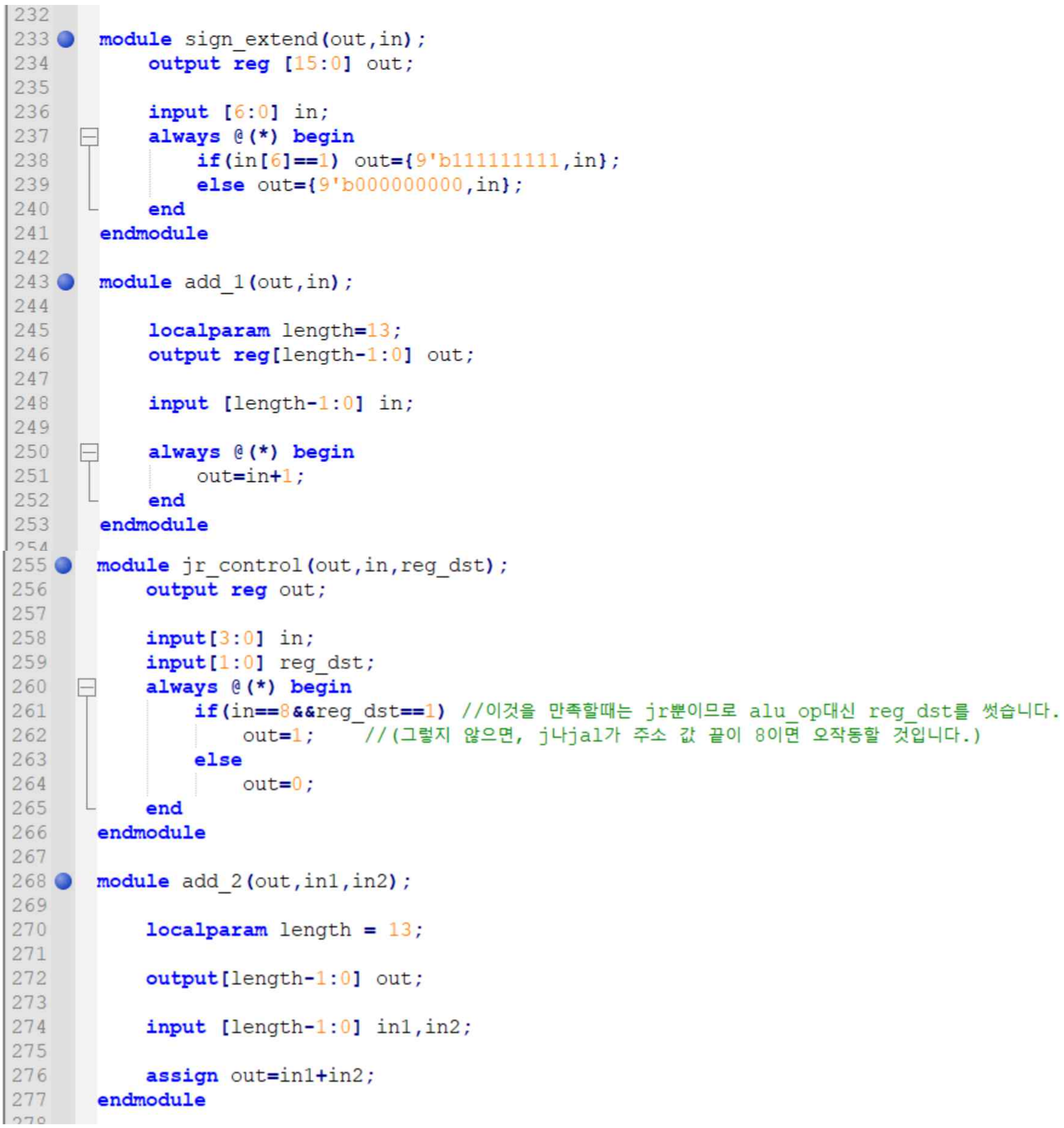
```
            3: begin //jal
                reg_dst=2;
                alu_src=0;
                mem_to_reg=2;
                reg_write=1;
                mem_read=0;
                mem_write=0;
                branch=0;
                alu_op=0;
                jump=1;
            end
            4: begin //lw
                reg_dst=0;
                alu_src=1;
                mem_to_reg=1;
                reg_write=1;
                mem_read=1;
                mem_write=0;
                branch=0;
                alu_op=3;
                jump=0;
            end
```



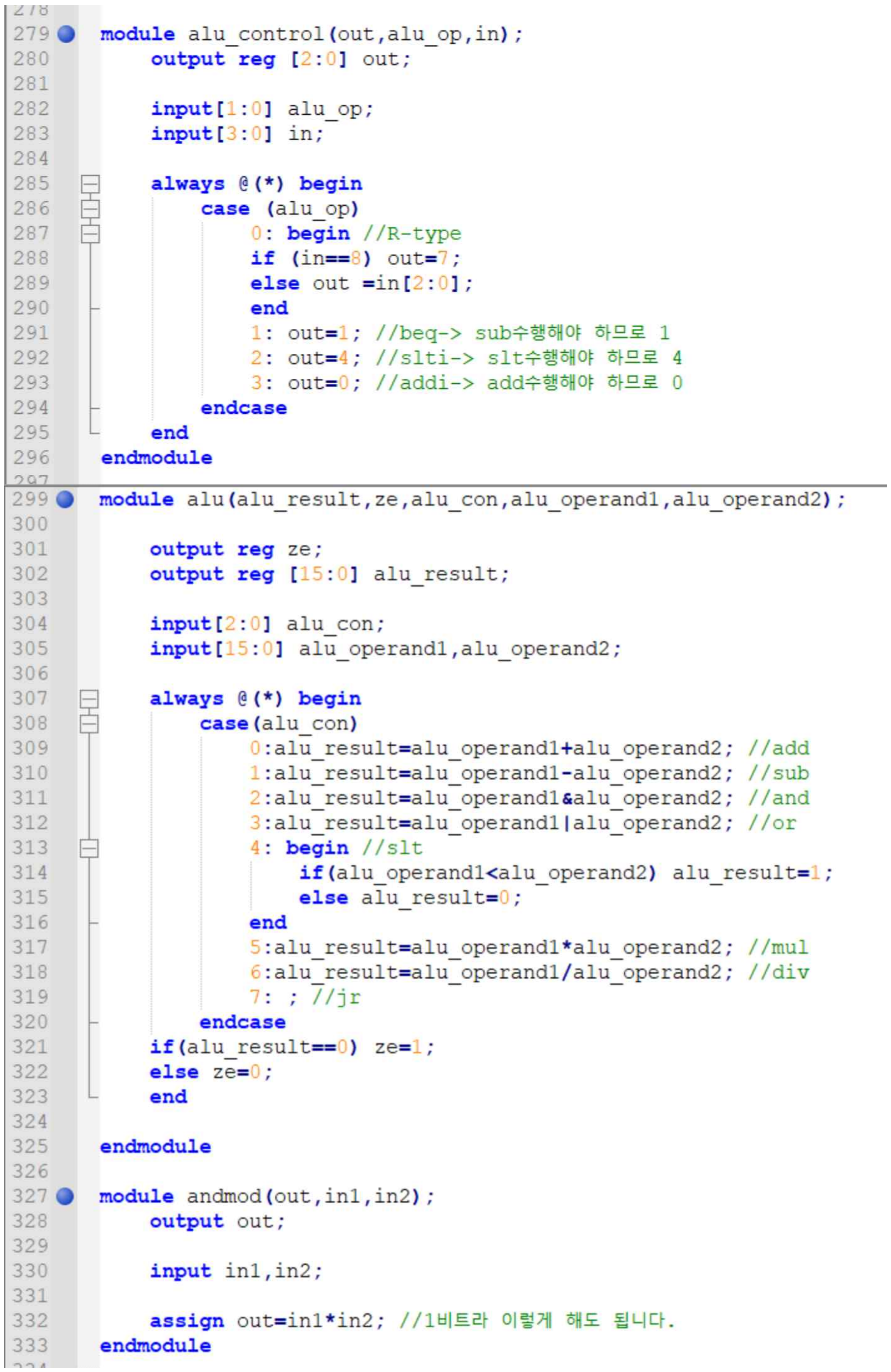
```
            5: begin //sw
                reg_dst=0;
                alu_src=1;
                mem_to_reg=0;
                reg_write=0;
                mem_read=0;
                mem_write=1;
                branch=0;
                alu_op=3;
                jump=0;
            end
            6: begin //beq
                reg_dst=0;
                alu_src=0;
                mem_to_reg=0;
                reg_write=0;
                mem_read=0;
                mem_write=0;
                branch=1;
                alu_op=1;
                jump=0;
```



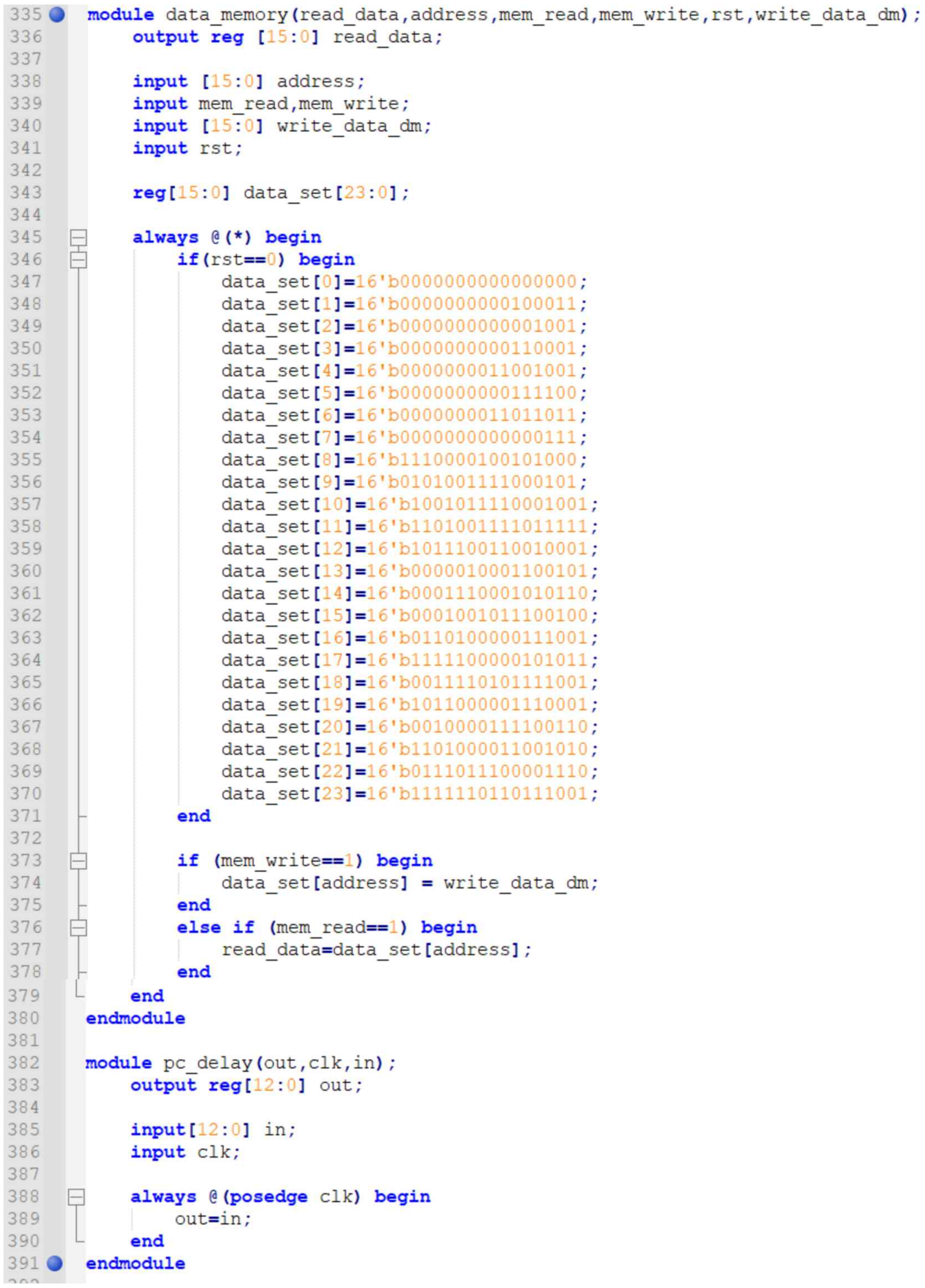
```
            7: begin //addi
                reg_dst=0;
                alu_src=1;
                mem_to_reg=0;
                reg_write=1;
                mem_read=0;
                mem_write=0;
                branch=0;
                alu_op=3;
                jump=0;
            end
        endcase
    end
endmodule
```



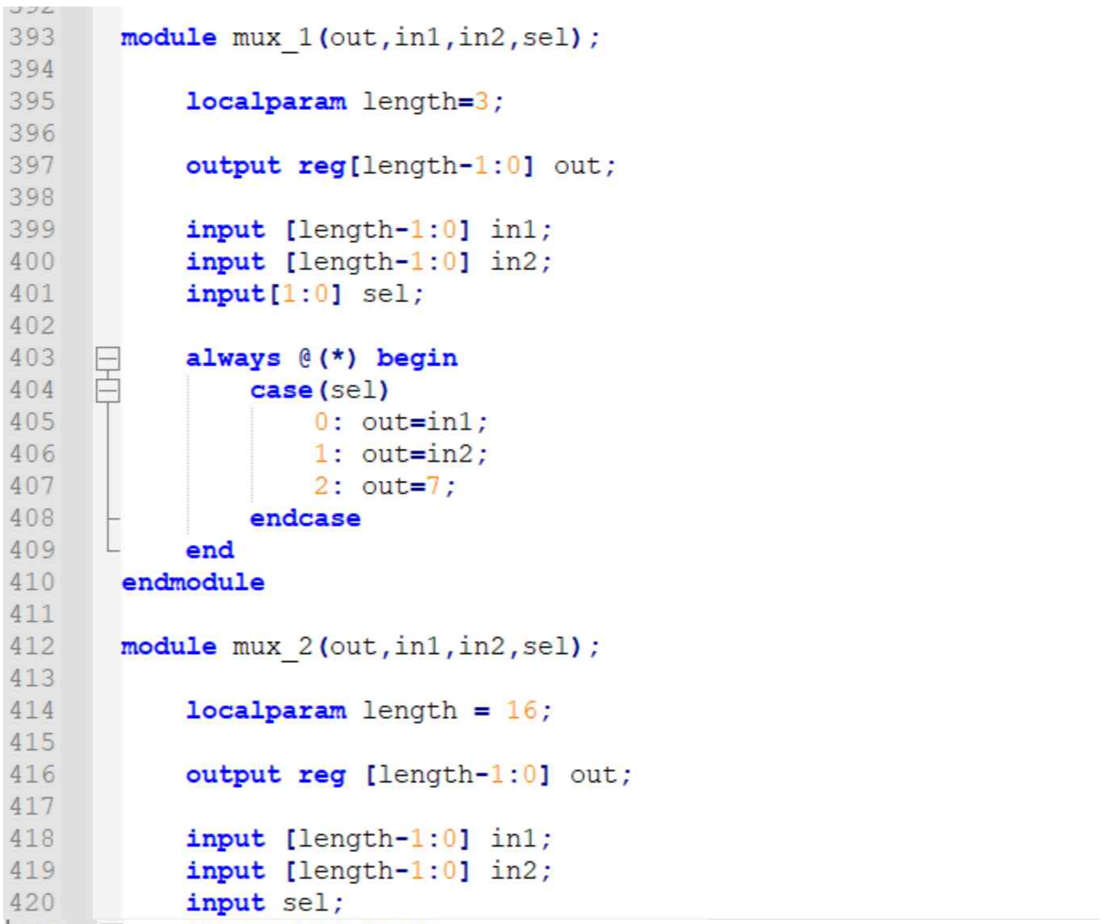
```verilog
module sign_extend(out,in);
    output reg [15:0] out;

    input [6:0] in;
    always @(*) begin
        if(in[6]==1) out={9'b111111111,in};
        else out={9'b000000000,in};
    end
endmodule

module add_1(out,in);

    localparam length=13;
    output reg[length-1:0] out;

    input [length-1:0] in;

    always @(*) begin
        out=in+1;
    end
endmodule

module jr_control(out,in,reg_dst);
    output reg out;

    input[3:0] in;
    input[1:0] reg_dst;
    always @(*) begin
        if(in==8&&reg_dst==1) //이것을 만족할때는 jr뿐이므로 alu_op대신 reg_dst를 썼습니다.
            out=1;    //(그렇지 않으면, j나jal가 주소 값 끝이 8이면 오작동할 것입니다.)
        else
            out=0;
    end
endmodule

module add_2(out,in1,in2);

    localparam length = 13;

    output[length-1:0] out;

    input [length-1:0] in1,in2;

    assign out=in1+in2;
endmodule
```



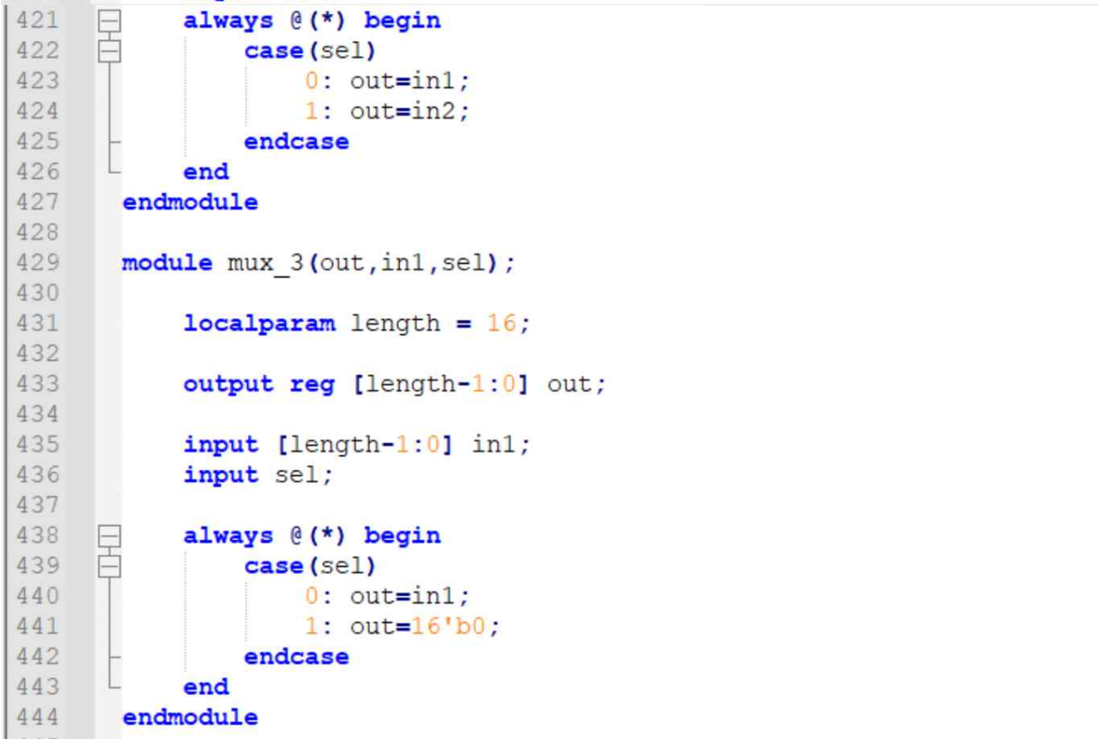
```
module alu_control(out,alu_op,in);
    output reg [2:0] out;

    input[1:0] alu_op;
    input[3:0] in;

    always @(*) begin
        case (alu_op)
            0: begin //R-type
            if (in==8) out=7;
            else out =in[2:0];
            end
            1: out=1; //beq-> sub수행해야 하므로 1
            2: out=4; //slti-> slt수행해야 하므로 4
            3: out=0; //addi-> add수행해야 하므로 0
        endcase
    end
endmodule

module alu(alu_result,ze,alu_con,alu_operand1,alu_operand2);

    output reg ze;
    output reg [15:0] alu_result;

    input[2:0] alu_con;
    input[15:0] alu_operand1,alu_operand2;

    always @(*) begin
        case(alu_con)
            0:alu_result=alu_operand1+alu_operand2; //add
            1:alu_result=alu_operand1-alu_operand2; //sub
            2:alu_result=alu_operand1&alu_operand2; //and
            3:alu_result=alu_operand1|alu_operand2; //or
            4: begin //slt
                if(alu_operand1<alu_operand2) alu_result=1;
                else alu_result=0;
            end
            5:alu_result=alu_operand1*alu_operand2; //mul
            6:alu_result=alu_operand1/alu_operand2; //div
            7: ; //jr
        endcase
    if(alu_result==0) ze=1;
    else ze=0;
    end

endmodule

module andmod(out,in1,in2);
    output out;

    input in1,in2;

    assign out=in1*in2; //1비트라 이렇게 해도 됩니다.
endmodule
```



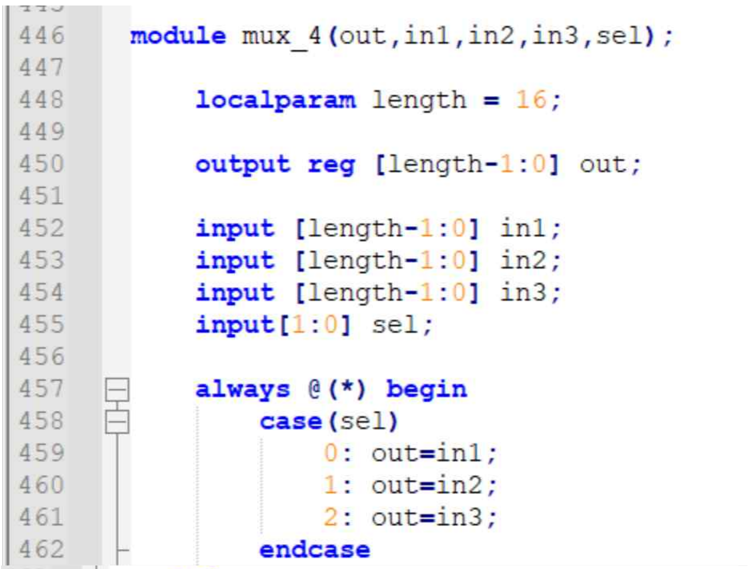
```
module data_memory(read_data,address,mem_read,mem_write,rst,write_data_dm);
    output reg [15:0] read_data;

    input [15:0] address;
    input mem_read,mem_write;
    input [15:0] write_data_dm;
    input rst;

    reg[15:0] data_set[23:0];

    always @(*) begin
        if(rst==0) begin
            data_set[0]=16'b0000000000000000;
            data_set[1]=16'b0000000000100011;
            data_set[2]=16'b0000000000001001;
            data_set[3]=16'b0000000000110001;
            data_set[4]=16'b0000000011001001;
            data_set[5]=16'b0000000000111100;
            data_set[6]=16'b0000000011011011;
            data_set[7]=16'b0000000000000111;
            data_set[8]=16'b1110000100101000;
            data_set[9]=16'b0101001111000101;
            data_set[10]=16'b1001011110001001;
            data_set[11]=16'b1101001111011111;
            data_set[12]=16'b1011100110010001;
            data_set[13]=16'b0000010001100101;
            data_set[14]=16'b0001110001010110;
            data_set[15]=16'b0001001011100100;
            data_set[16]=16'b0110100000111001;
            data_set[17]=16'b1111100000101011;
            data_set[18]=16'b0011110101111001;
            data_set[19]=16'b1011000001110001;
            data_set[20]=16'b0010000111100110;
            data_set[21]=16'b1101000011001010;
            data_set[22]=16'b0111011100001110;
            data_set[23]=16'b1111110110111001;
        end

        if (mem_write==1) begin
            data_set[address] = write_data_dm;
        end
        else if (mem_read==1) begin
            read_data=data_set[address];
        end
    end
endmodule

module pc_delay(out,clk,in);
    output reg[12:0] out;

    input[12:0] in;
    input clk;

    always @(posedge clk) begin
        out=in;
    end
endmodule
```



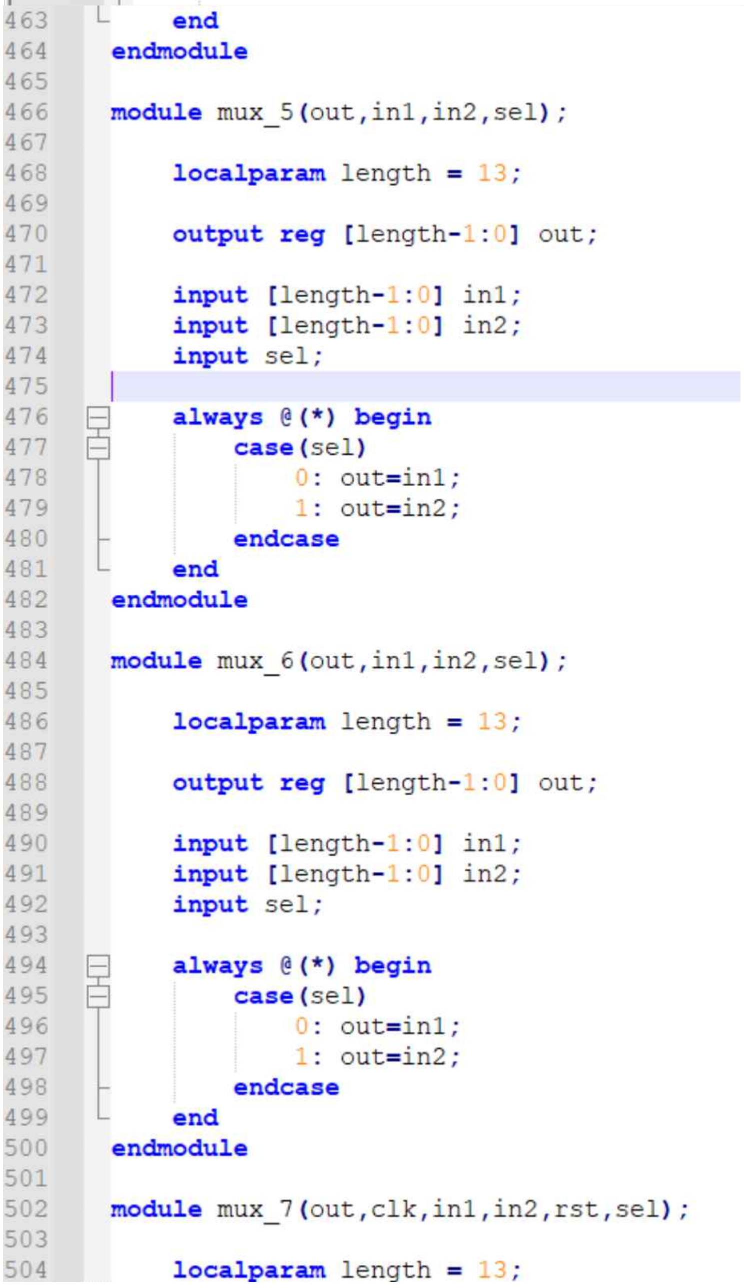
```
module mux_1(out,in1,in2,sel);

    localparam length=3;

    output reg[length-1:0] out;

    input [length-1:0] in1;
    input [length-1:0] in2;
    input[1:0] sel;

    always @(*) begin
        case(sel)
            0: out=in1;
            1: out=in2;
            2: out=7;
        endcase
    end
endmodule

module mux_2(out,in1,in2,sel);

    localparam length = 16;

    output reg [length-1:0] out;

    input [length-1:0] in1;
    input [length-1:0] in2;
    input sel;
```



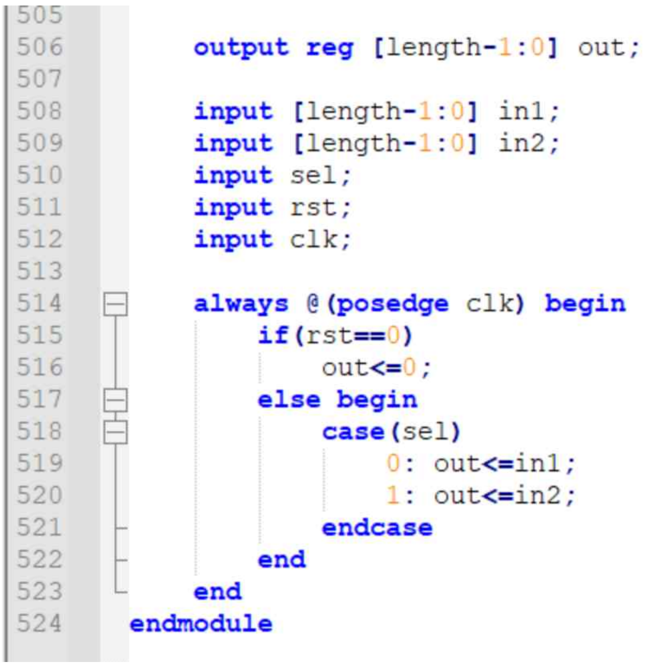
```
    always @(*) begin
        case(sel)
            0: out=in1;
            1: out=in2;
        endcase
    end
endmodule

module mux_3(out,in1,sel);

    localparam length = 16;

    output reg [length-1:0] out;

    input [length-1:0] in1;
    input sel;

    always @(*) begin
        case(sel)
            0: out=in1;
            1: out=16'b0;
        endcase
    end
endmodule
```



```verilog
module mux_4(out,in1,in2,in3,sel);

    localparam length = 16;

    output reg [length-1:0] out;

    input [length-1:0] in1;
    input [length-1:0] in2;
    input [length-1:0] in3;
    input[1:0] sel;

    always @(*) begin
        case(sel)
            0: out=in1;
            1: out=in2;
            2: out=in3;
        endcase
```



```verilog
    end
endmodule

module mux_5(out,in1,in2,sel);

    localparam length = 13;

    output reg [length-1:0] out;

    input [length-1:0] in1;
    input [length-1:0] in2;
    input sel;

    always @(*) begin
        case(sel)
            0: out=in1;
            1: out=in2;
        endcase
    end
endmodule

module mux_6(out,in1,in2,sel);

    localparam length = 13;

    output reg [length-1:0] out;

    input [length-1:0] in1;
    input [length-1:0] in2;
    input sel;

    always @(*) begin
        case(sel)
            0: out=in1;
            1: out=in2;
        endcase
    end
endmodule

module mux_7(out,clk,in1,in2,rst,sel);

    localparam length = 13;
```



```
    output reg [length-1:0] out;

    input [length-1:0] in1;
    input [length-1:0] in2;
    input sel;
    input rst;
    input clk;

    always @(posedge clk) begin
        if(rst==0)
            out<=0;
        else begin
            case(sel)
                0: out<=in1;
                1: out<=in2;
            endcase
        end
    end
endmodule
```