

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

1. 표지와 목차.....	1~2
2. 결과 화면.....	3
3. 모듈 설명.....	4~7
4. 코드.....	8~18

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 2000 ps, PC = 0, RF[0, 1, 2, 3, 4, 7] is: 0, 0, 35, 0, 0, 0
ID:4143, at time 3000 ps, PC = 1, RF[0, 1, 2, 3, 4, 7] is: 0, 0, 35, 9, 0, 0
ID:4143, at time 4000 ps, PC = 2, RF[0, 1, 2, 3, 4, 7] is: 0, 0, 35, 9, 44, 0
ID:4143, at time 5000 ps, PC = 3, RF[0, 1, 2, 3, 4, 7] is: 0, 0, 35, 9, 44, 0
ID:4143, at time 6000 ps, PC = 4, RF[0, 1, 2, 3, 4, 7] is: 0, 0, 35, 9, 44, 0
ID:4143, at time 7000 ps, PC = 7, RF[0, 1, 2, 3, 4, 7] is: 0, 0, 35, 9, 26, 0
ID:4143, at time 8000 ps, PC = 8, RF[0, 1, 2, 3, 4, 7] is: 0, 0, 35, 9, 43, 0
ID:4143, at time 9000 ps, PC = 9, RF[0, 1, 2, 3, 4, 7] is: 0, 0, 35, 9, 43, 10
ID:4143, at time 10000 ps, PC = 13, RF[0, 1, 2, 3, 4, 7] is: 0, 0, 35, 9, 3, 10
ID:4143, at time 11000 ps, PC = 14, RF[0, 1, 2, 3, 4, 7] is: 0, 0, 35, 9, 3, 10
ID:4143, at time 12000 ps, PC = 10, RF[0, 1, 2, 3, 4, 7] is: 0, 0, 35, 9, 5, 10
ID:4143, at time 13000 ps, PC = 11, RF[0, 1, 2, 3, 4, 7] is: 0, 0, 35, 9, 5, 10
ID:4143, at time 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.

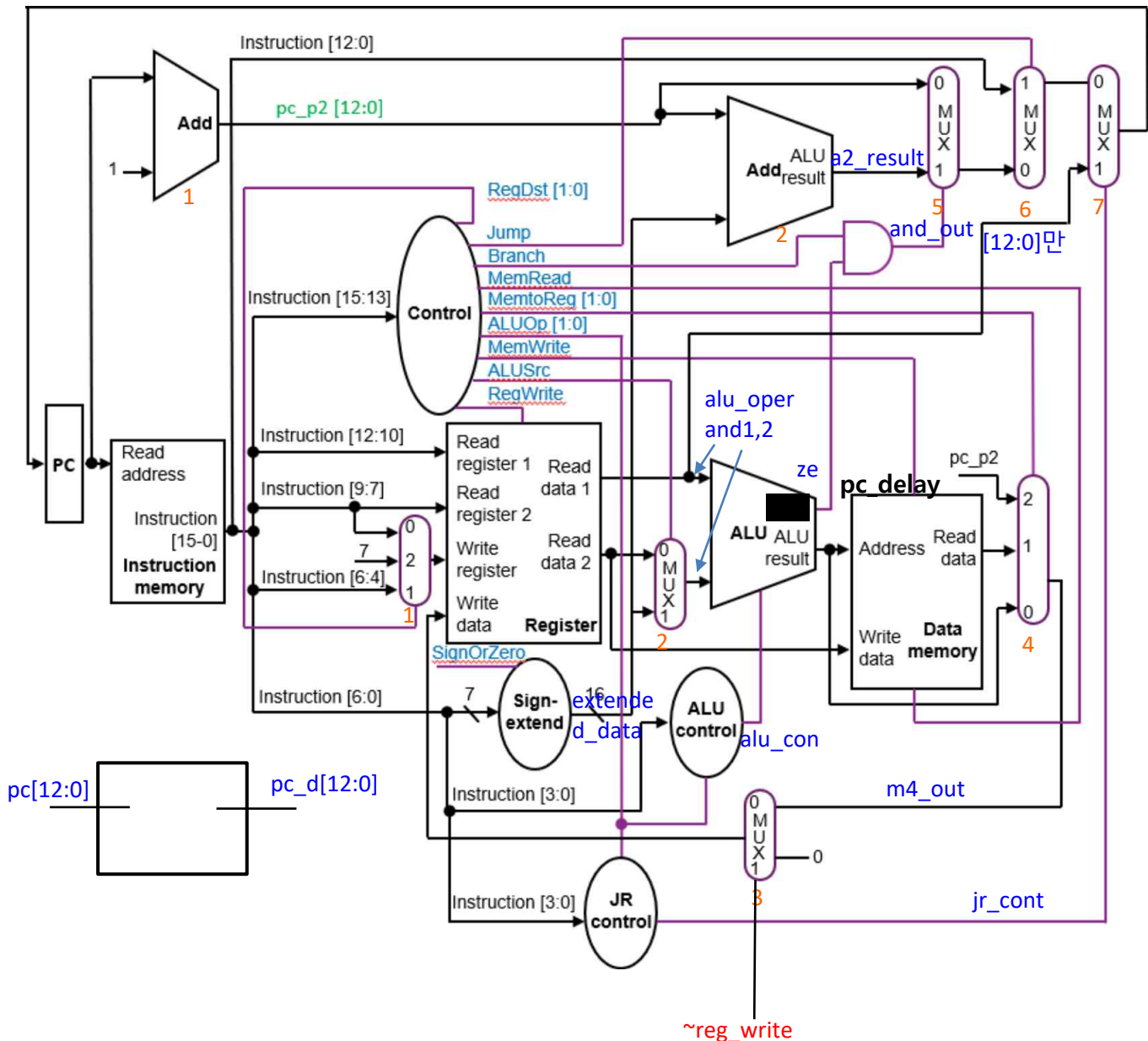
3. 모듈 설명

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 불친 이름들을 파란색으로 표시했습니다. mux말에는 몇번 mux인지 적어놨으며, 따로 추가로 표기 안 한것들은 기존에 이름대로 씁니다.

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 띄우기 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 triggered되어야 한다.(7번째)빼고는 다 만족했습니다.

1. 탑 모듈은 하위 모듈로만 구성

```

1  module top_module(pc_d,r0,r1,r2,r3,r4,r7,clk,rst);
2      output [12:0] pc_d;
3      output [15:0] r0,r1,r2,r3,r4,r7;
4
5      input clk;
6      input rst;
7
8      wire [15:0] instruction;
9      wire [15:0] read_data1,read_data2;
10     wire [15:0] write_data;
11     wire [15:0] read_data;
12     wire [15:0] m4_out;
13     wire [15:0] extended_data;
14     wire [15:0] alu_operand2;
15     wire [15:0] alu_result;
16     wire [12:0] a2_result;
17     wire [12:0] pc;
18     wire [12:0] m5_out,m6_out;
19     wire [12:0] pc_p2;
20     wire [2:0] write_r;
21     wire [2:0] alu_con;
22     wire [1:0] reg_dst,mem_to_reg,alu_op;
23     wire jump,branch,mem_read,mem_write,alu_src,reg_write;
24     wire ze;
25     wire and_out;
26     wire jr_cont;
27
28     instruction_memory im(instruction,pc,rst);
29     regisier registers(r0,r1,r2,r3,r4,r7,read_data1,read_da
30     control_con(reg_dst,jump,branch,mem_read,mem_to_reg,al
31     sign_extend ext(extended_data,instruction[6:0]);
32     add_1 a1(pc_p2,pc);
33     jr_control jrc(jr_cont,instruction[3:0],reg_dst);
34     add_2 a2(a2_result,pc_p2,extended_data[12:0]);
35     alu_control alu_c(alu_con,alu_op,instruction[3:0]);
36     alu alu(alu_result,ze,alu_con,read_data1,alu_operand2)
37     andmod andmo(and_out,branch,ze);
38     data_memory dm(read_data,alu_result,mem_read,mem_write
39     pc_delay pcd(pc_d,clk,pc);
40
41     mux_1 m1(write_r,instruction[9:7],instruction[6:4],reg
42     mux_2 m2(alu_operand2,read_data2,extended_data,alu_src
43     mux_3 m3(write_data,m4_out,~reg_write);
44     mux_4 m4(m4_out,alu_result,read_data,{3'b000,pc_p2},me
45     mux_5 m5(m5_out,pc_p2,a2_result,and_out);
46     mux_6 m6(m6_out,m5_out,instruction[12:0],jump);
47     mux_7 m7(pc,clk,m6_out,read_data1[12:0],rst,jr_cont);
48
49     endmodule

```

2. adder와 mux는 parameter를 써서 설계

<pre> 243 module add_1(out,in); 244 245 localparam length=13; 246 output reg[length-1:0] out; 247 248 input [length-1:0] in; 249 250 always @(*) begin 251 out=in+1; 252 end 253 endmodule </pre>	<pre> 393 module mux_1(out,in1,in2,sel); 394 395 localparam length=3; 396 397 output reg[length-1:0] out; 398 399 input [length-1:0] in1; 400 input [length-1:0] in2; 401 input [1:0] sel; 402 403 always @(*) begin 404 case(sel) 405 0: out=in1; 406 1: out=in2; 407 2: out=7; 408 endcase 409 end 410 endmodule </pre>
---	---

둘 다 길이를 localparameter를 써서 했습니다.

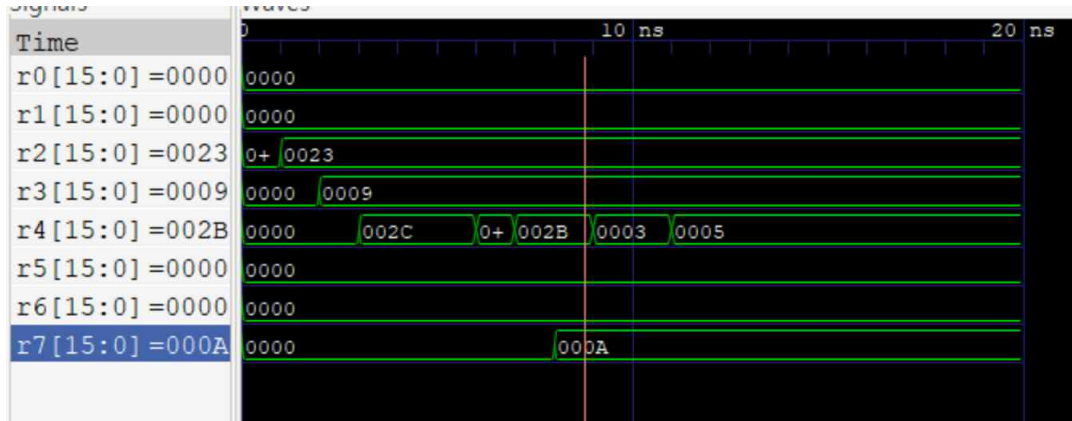
3. 메모리를 가장 편리한 방법대로 초기화

<pre> 60 always @(*) begin 61 if(rst==0) begin 62 instruction_set[0]<=16'b1001100100000001; 63 instruction_set[1]<=16'b1001100110000010; 64 instruction_set[2]<=16'b0000100111000000; 65 instruction_set[3]<=16'b0000100110010100; 66 instruction_set[4]<=16'b1100010000000010; 67 instruction_set[5]<=16'b0000100111000010; 68 instruction_set[6]<=16'b0100000000001000; 69 instruction_set[7]<=16'b0000100111000001; 70 instruction_set[8]<=16'b0000100111000011; 71 instruction_set[9]<=16'b0110000000001101; 72 instruction_set[10]<=16'b1111001000000010; 73 instruction_set[11]<=16'b1011101000000011; 74 instruction_set[12]<=16'b0100000000001111; 75 instruction_set[13]<=16'b0000100111000110; 76 instruction_set[14]<=16'b0001110000001000; 77 instruction_set[15]<=16'b0000000000000000; 78 instruction_set[16]<=16'b0000000000000000; 79 instruction_set[17]<=16'b0000000000000000; 80 instruction_set[18]<=16'b0000000000000000; 81 instruction_set[19]<=16'b0000000000000000; 82 instruction_set[20]<=16'b0000000000000000; 83 instruction_set[21]<=16'b0000000000000000; 84 end </pre>	<pre> 345 always @(*) begin 346 if(rst==0) begin 347 data_set[0]=16'b0000000000000000; 348 data_set[1]=16'b00000000000100011; 349 data_set[2]=16'b00000000000001001; 350 data_set[3]=16'b00000000000110001; 351 data_set[4]=16'b00000000011001001; 352 data_set[5]=16'b0000000000111100; 353 data_set[6]=16'b0000000011011011; 354 data_set[7]=16'b0000000000000111; 355 data_set[8]=16'b1110000100101000; 356 data_set[9]=16'b0101001111000101; 357 data_set[10]=16'b1001011110001001; 358 data_set[11]=16'b1101001111011111; 359 data_set[12]=16'b101100110010001; 360 data_set[13]=16'b000010001100101; 361 data_set[14]=16'b0001110001010110; 362 data_set[15]=16'b0001001011100100; 363 data_set[16]=16'b0110100000111001; 364 data_set[17]=16'b111100000101011; 365 data_set[18]=16'b0011110101111001; 366 data_set[19]=16'b1011000001110001; 367 data_set[20]=16'b0010000111100110; 368 data_set[21]=16'b1101000011001010; 369 data_set[22]=16'b0111011100001110; 370 data_set[23]=16'b111110110111001; 371 end 372 end </pre>
--	---

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

```
1 `timescale 1ns/100ps
2
3 module misc();
4     reg clk;
5     reg rst;
6
7     wire[12:0] current_inst;
8     wire[15:0] r0,r1,r2,r3,r4,r7;
9
10    top_module top(current_inst,r0,r1,r2,r3,r4,r7,clk,rst);
11
12    always begin
13        #0.5 clk = ~clk;
14    end
15
16    always begin
17        #1
18        $write("ID:4143, at time");
19        $write($time);
20        $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);
21    end
22    initial begin
23        rst=0;
24        clk=1;
25        #1 rst=1;
26    end
27
28    initial begin
29        $dumpfile("output.vcd");
30        $dumpvars(0);
31        #20 $write("The final result of $s3 in memory is: %d\n",r4);
32        $finish;
33    end
34
35 endmodule
```


4-2 p3.v

1) top module

```

1 module top_module(pc_d,r0,r1,r2,r3,r4,r7,clk,rst);
2     output [12:0] pc_d;
3     output [15:0] r0,r1,r2,r3,r4,r7;
4
5     input clk;
6     input rst;
7
8     wire [15:0] instruction;
9     wire [15:0] read_data1,read_data2;
10    wire [15:0] write_data;
11    wire [15:0] read_data;
12    wire [15:0] m4_out;
13    wire [15:0] extended_data;
14    wire [15:0] alu_operand2;
15    wire [15:0] alu_result;
16    wire [12:0] a2_result;
17    wire [12:0] pc;
18    wire [12:0] m5_out,m6_out;
19    wire [12:0] pc_p2;
20    wire [2:0] write_r;
21    wire [2:0] alu_con;
22    wire [1:0] reg_dst,mem_to_reg,alu_op;
23    wire jump,branch,mem_read,mem_write,alu_src,reg_write;
24    wire ze;
25    wire and_out;
26    wire jr_cont;
27
28    instruction_memory im(instruction,pc,rst);
29    regisier registers(r0,r1,r2,r3,r4,r7,read_data1,read_data2,clk,instruction[12:10],instruction[9:7],reg_write,rst,write_data,write_r);
30    control con(reg_dst,jump,branch,mem_read,mem_to_reg,alu_op,mem_write,alu_src,reg_write,instruction[15:13]);
31    sign_extend ext(extended_data,instruction[6:0]);
32    add_1 a1(pc_p2,pc);
33    jr_control jrc(jr_cont,instruction[3:0],reg_dst);
34    add_2 a2(a2_result,pc_p2,extended_data[12:0]);
35    alu_control alu_c(alu_con,alu_op,instruction[3:0]);
36    alu alu(alu_result,ze,alu_con,read_data1,alu_operand2);
37    andmod andmo(and_out,branch,ze);
38    data_memory dm(read_data,alu_result,mem_read,mem_write,rst,read_data2);
39    pc_delay pcd(pc_d,clk,pc);
40
41    mux_1 m1(write_r,instruction[9:7],instruction[6:4],reg_dst);
42    mux_2 m2(alu_operand2,read_data2,extended_data,alu_src);
43    mux_3 m3(write_data,m4_out,~reg_write);
44    mux_4 m4(m4_out,alu_result,read_data,{3'b000,pc_p2},mem_to_reg);
45    mux_5 m5(m5_out,pc_p2,a2_result,and_out);
46    mux_6 m6(m6_out,m5_out,instruction[12:0],jump);
47    mux_7 m7(pc,clk,m6_out,read_data1[12:0],rst,jr_cont);
48
49 endmodule

```

2)그 이외

```

51 module instruction_memory(instruction,pc,rst);
52
53     output reg[15:0] instruction;
54
55     input[12:0] pc;
56     input rst;
57
58     reg[15:0] instruction_set[22:0];
59
60     always @(*) begin
61         if(rst==0) begin
62             instruction_set[0]<=16'b1001100100000001;
63             instruction_set[1]<=16'b1001100110000010;
64             instruction_set[2]<=16'b0000100111000000;
65             instruction_set[3]<=16'b0000100110010100;
66             instruction_set[4]<=16'b1100010000000010;
67             instruction_set[5]<=16'b0000100111000010;
68             instruction_set[6]<=16'b0100000000001000;
69             instruction_set[7]<=16'b0000100111000001;
70             instruction_set[8]<=16'b0000100111000011;
71             instruction_set[9]<=16'b0110000000001101;
72             instruction_set[10]<=16'b1111001000000010;
73             instruction_set[11]<=16'b1011101000000011;
74             instruction_set[12]<=16'b0100000000001111;
75             instruction_set[13]<=16'b0000100111000110;
76             instruction_set[14]<=16'b0001110000001000;
77             instruction_set[15]<=16'b0000000000000000;
78             instruction_set[16]<=16'b0000000000000000;
79             instruction_set[17]<=16'b0000000000000000;
80             instruction_set[18]<=16'b0000000000000000;
81             instruction_set[19]<=16'b0000000000000000;
82             instruction_set[20]<=16'b0000000000000000;
83             instruction_set[21]<=16'b0000000000000000;
84         end
85
86         instruction=instruction_set[pc];
87     end
88 endmodule
89
90 module resgiser(r0,r1,r2,r3,r4,r7,read_data1,read_data2,clk,read_r1,read_r2,reg_write,rst,write_data,write_r);
91     output [15:0] read_data1,read_data2;
92     output [15:0] r0,r1,r2,r3,r4,r7;
93
94     input [2:0] read_r1,read_r2;
95     input rst,reg_write;
96     input [15:0] write_data;
97     input [2:0] write_r;
98     input clk;
99
100     reg [15:0] register[7:0];
101     wire [15:0] r5,r6;
102
103     always @(posedge clk) begin
104         if(rst==0) begin
105             register[0]<=0;
106             register[1]<=0;
107             register[2]<=0;
108             register[3]<=0;
109             register[4]<=0;
110             register[5]<=0;
111             register[6]<=0;
112             register[7]<=0;
113         end
114
115         if(reg_write==1&&write_r!=0) begin //0일때 쓰이는 일이 없도록
116             register[write_r]<=write_data;
117         end
118     end
119

```

```

120
121     assign read_data1=register[read_r1];
122     assign read_data2=register[read_r2];
123     assign r0=register[0];
124     assign r1=register[1];
125     assign r2=register[2];
126     assign r2=register[2];
127     assign r3=register[3];
128     assign r4=register[4];
129     assign r5=register[5];
130     assign r6=register[6];
131     assign r7=register[7];
132 endmodule
134 module control(reg_dst,jump,branch,mem_read,mem_to_reg,alu_op,mem_write,alu_src,reg_write,instruction
135     output reg jump,branch,mem_read,mem_write,alu_src,reg_write;
136     output reg [1:0] reg_dst,mem_to_reg,alu_op;
137
138     input [2:0] instruction;
139     always @(instruction) begin
140         case(instruction)
141             0: begin //R-type
142                 reg_dst=1;
143                 alu_src=0;
144                 mem_to_reg=0;
145                 reg_write=1;
146                 mem_read=0;
147                 mem_write=0;
148                 branch=0;
149                 alu_op=0;
150                 jump=0;
151             end
152             1: begin //slti
153                 reg_dst=0;
154                 alu_src=1;
155                 mem_to_reg=0;
156                 reg_write=1;
157                 mem_read=0;
158                 mem_write=0;
159                 branch=0;
160                 alu_op=2;
161                 jump=0;
162             end
163             2: begin //j
164                 reg_dst=0;
165                 alu_src=0;
166                 mem_to_reg=0;
167                 reg_write=0;
168                 mem_read=0;
169                 mem_write=0;
170                 branch=0;
171                 alu_op=0;
172                 jump=1;
173             end
174         endcase
175     end

```

```

174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232

```

```

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;
end
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

```

```

232
233 ● module sign_extend(out,in);
234     output reg [15:0] out;
235
236     input [6:0] in;
237     always @(*) begin
238         if(in[6]==1) out={9'b11111111,in};
239         else out={9'b00000000,in};
240     end
241 endmodule
242
243 ● module add_1(out,in);
244
245     localparam length=13;
246     output reg[length-1:0] out;
247
248     input [length-1:0] in;
249
250     always @(*) begin
251         out=in+1;
252     end
253 endmodule
254
255 ● module jr_control(out,in,reg_dst);
256     output reg out;
257
258     input[3:0] in;
259     input[1:0] reg_dst;
260     always @(*) begin
261         if(in==8&&reg_dst==1) //이것을 만족할때는 jr뿐이므로 alu_op대신 reg_dst를 썼습니다.
262             out=1; // (그렇지 않으면, j나jal가 주소 값 끝이 8이면 오작동할 것입니다.)
263         else
264             out=0;
265     end
266 endmodule
267
268 ● module add_2(out,in1,in2);
269
270     localparam length = 13;
271
272     output[length-1:0] out;
273
274     input [length-1:0] in1,in2;
275
276     assign out=in1+in2;
277 endmodule
278

```



```

278
279 module alu_control(out,alu_op,in);
280     output reg [2:0] out;
281
282     input[1:0] alu_op;
283     input[3:0] in;
284
285     always @(*) begin
286         case (alu_op)
287             0: begin //R-type
288                 if (in==8) out=7;
289                 else out =in[2:0];
290             end
291             1: out=1; //beq-> sub수행해야 하므로 1
292             2: out=4; //slti-> slt수행해야 하므로 4
293             3: out=0; //addi-> add수행해야 하므로 0
294         endcase
295     end
296 endmodule
297
299 module alu(alu_result,ze,alu_con,alu_operand1,alu_operand2);
300
301     output reg ze;
302     output reg [15:0] alu_result;
303
304     input[2:0] alu_con;
305     input[15:0] alu_operand1,alu_operand2;
306
307     always @(*) begin
308         case(alu_con)
309             0:alu_result=alu_operand1+alu_operand2; //add
310             1:alu_result=alu_operand1-alu_operand2; //sub
311             2:alu_result=alu_operand1&alu_operand2; //and
312             3:alu_result=alu_operand1|alu_operand2; //or
313             4: begin //slt
314                 if(alu_operand1<alu_operand2) alu_result=1;
315                 else alu_result=0;
316             end
317             5:alu_result=alu_operand1*alu_operand2; //mul
318             6:alu_result=alu_operand1/alu_operand2; //div
319             7: ; //jr
320         endcase
321         if(alu_result==0) ze=1;
322         else ze=0;
323     end
324
325 endmodule
326
327 module andmod(out,in1,in2);
328     output out;
329
330     input in1,in2;
331
332     assign out=in1*in2; //1비트라 이렇게 해도 됩니다.
333 endmodule
334

```

```

335 module data_memory(read_data,address,mem_read,mem_write,rst,write_data_dm);
336     output reg [15:0] read_data;
337
338     input [15:0] address;
339     input mem_read,mem_write;
340     input [15:0] write_data_dm;
341     input rst;
342
343     reg[15:0] data_set[23:0];
344
345     always @(*) begin
346         if(rst==0) begin
347             data_set[0]=16'b0000000000000000;
348             data_set[1]=16'b0000000000100011;
349             data_set[2]=16'b0000000000001001;
350             data_set[3]=16'b0000000000110001;
351             data_set[4]=16'b0000000011001001;
352             data_set[5]=16'b000000000111100;
353             data_set[6]=16'b0000000011011011;
354             data_set[7]=16'b0000000000000111;
355             data_set[8]=16'b1110000100101000;
356             data_set[9]=16'b0101001111000101;
357             data_set[10]=16'b1001011110001001;
358             data_set[11]=16'b1101001111011111;
359             data_set[12]=16'b1011100110010001;
360             data_set[13]=16'b0000010001100101;
361             data_set[14]=16'b0001110001010110;
362             data_set[15]=16'b0001001011100100;
363             data_set[16]=16'b0110100000111001;
364             data_set[17]=16'b1111100000101011;
365             data_set[18]=16'b0011110101111001;
366             data_set[19]=16'b1011000001110001;
367             data_set[20]=16'b0010000111100110;
368             data_set[21]=16'b1101000011001010;
369             data_set[22]=16'b0111011100001110;
370             data_set[23]=16'b111110110111001;
371         end
372
373         if (mem_write==1) begin
374             data_set[address] = write_data_dm;
375         end
376         else if (mem_read==1) begin
377             read_data=data_set[address];
378         end
379     end
380 endmodule
381
382 module pc_delay(out,clk,in);
383     output reg[12:0] out;
384
385     input[12:0] in;
386     input clk;
387
388     always @(posedge clk) begin
389         out=in;
390     end
391 endmodule

```

```

393 module mux_1(out,in1,in2,sel);
394
395     localparam length=3;
396
397     output reg[length-1:0] out;
398
399     input [length-1:0] in1;
400     input [length-1:0] in2;
401     input[1:0] sel;
402
403     always @(*) begin
404         case(sel)
405             0: out=in1;
406             1: out=in2;
407             2: out=7;
408         endcase
409     end
410 endmodule
411
412 module mux_2(out,in1,in2,sel);
413
414     localparam length = 16;
415
416     output reg [length-1:0] out;
417
418     input [length-1:0] in1;
419     input [length-1:0] in2;
420     input sel;
421
422     always @(*) begin
423         case(sel)
424             0: out=in1;
425             1: out=in2;
426         endcase
427     end
428 endmodule
429
430 module mux_3(out,in1,sel);
431
432     localparam length = 16;
433
434     output reg [length-1:0] out;
435
436     input [length-1:0] in1;
437     input sel;
438
439     always @(*) begin
440         case(sel)
441             0: out=in1;
442             1: out=16'b0;
443         endcase
444     end
445 endmodule

```



```

446 module mux_4(out,in1,in2,in3,sel);
447
448     localparam length = 16;
449
450     output reg [length-1:0] out;
451
452     input [length-1:0] in1;
453     input [length-1:0] in2;
454     input [length-1:0] in3;
455     input [1:0] sel;
456
457     always @(*) begin
458         case(sel)
459             0: out=in1;
460             1: out=in2;
461             2: out=in3;
462         endcase
463     end
464 endmodule
465
466 module mux_5(out,in1,in2,sel);
467
468     localparam length = 13;
469
470     output reg [length-1:0] out;
471
472     input [length-1:0] in1;
473     input [length-1:0] in2;
474     input sel;
475
476     always @(*) begin
477         case(sel)
478             0: out=in1;
479             1: out=in2;
480         endcase
481     end
482 endmodule
483
484 module mux_6(out,in1,in2,sel);
485
486     localparam length = 13;
487
488     output reg [length-1:0] out;
489
490     input [length-1:0] in1;
491     input [length-1:0] in2;
492     input sel;
493
494     always @(*) begin
495         case(sel)
496             0: out=in1;
497             1: out=in2;
498         endcase
499     end
500 endmodule
501
502 module mux_7(out,clk,in1,in2,rst,sel);
503
504     localparam length = 13;

```

```
505
506     output reg [length-1:0] out;
507
508     input [length-1:0] in1;
509     input [length-1:0] in2;
510     input sel;
511     input rst;
512     input clk;
513
514     always @(posedge clk) begin
515         if(rst==0)
516             out<=0;
517         else begin
518             case(sel)
519                 0: out<=in1;
520                 1: out<=in2;
521             endcase
522         end
523     end
524 endmodule
```