```
1.  `timescale 1ns / 1ps
2.
3.  ////////////fields of IR
4.  `define oper_type IR[31:27]
5.  `define rdst      IR[26:22]
6.  `define rsrc1     IR[21:17]
7.  `define imm_mode  IR[16]
8.  `define rsrc2     IR[15:11]
9.  `define isrc      IR[15:0]
10.
11.
12. ////////////////arithmetic operation
13. `define movsgpr        5'b00000
14. `define mov            5'b00001
15. `define add            5'b00010
16. `define sub            5'b00011
17. `define mul            5'b00100
18.
19. ////////////////logical operations : and or xor xnor nand nor not
20.
21. `define ror            5'b00101
22. `define rand           5'b00110
23. `define rxor           5'b00111
24. `define rxnor          5'b01000
25. `define rnand          5'b01001
26. `define rnor           5'b01010
27. `define rnot           5'b01011
28.
29.
30. module top();
31.
32.
33.
34.
35.
36.
37. reg [31:0] IR;            ////// instruction register  <--ir[31:27]--><--ir[26:22]--><--
    ir[21:17]--><--ir[16]--><--ir[15:11]--><--ir[10:0]-->
38.                          //////fields                 <---  oper  --><--   rdest --><--
    rsrc1 --><--modesel--><--  rsrc2 --><--unused  -->
39.                          //////fields                 <---  oper  --><--   rdest --><--
    rsrc1 --><--modesel--><--  immediate_date      -->
40.
41. reg [15:0] GPR [31:0] ;   ///////general purpose register gpr[0] ..... gpr[31]
42.
43.
44.
45. reg [15:0] SGPR ;      ///// msb of multiplication --> special register
46.
47. reg [31:0] mul_res;
48.
49.
50.
51. always@(*)
52. begin
53. case(`oper_type)
54. ////////////////////////////////
55. `movsgpr: begin
56.
57.    GPR[`rdst] = SGPR;
58.
```

```
59. end
60.
61. ////////////////////////////////
62. `mov : begin
63.    if(`imm_mode)
64.         GPR[`rdst]  = `isrc;
65.    else
66.        GPR[`rdst]   = GPR[`rsrc1];
67. end
68.
69. ////////////////////////////////////////////////
70.
71. `add : begin
72.       if(`imm_mode)
73.         GPR[`rdst]   = GPR[`rsrc1] + `isrc;
74.      else
75.         GPR[`rdst]   = GPR[`rsrc1] + GPR[`rsrc2];
76. end
77.
78. ////////////////////////////////////////////////////////
79.
80. `sub : begin
81.       if(`imm_mode)
82.         GPR[`rdst]  = GPR[`rsrc1] - `isrc;
83.      else
84.        GPR[`rdst]   = GPR[`rsrc1] - GPR[`rsrc2];
85. end
86.
87. ////////////////////////////////////////////////////////////
88.
89. `mul : begin
90.       if(`imm_mode)
91.         mul_res   = GPR[`rsrc1] * `isrc;
92.      else
93.         mul_res   = GPR[`rsrc1] * GPR[`rsrc2];
94.
95.      GPR[`rdst]   =  mul_res[15:0];
96.      SGPR         =  mul_res[31:16];
97. end
98.
99. ////////////////////////////////////////////////////////////// bitwise or
100.
101.       `ror : begin
102.            if(`imm_mode)
103.              GPR[`rdst]  = GPR[`rsrc1] | `isrc;
104.           else
105.             GPR[`rdst]   = GPR[`rsrc1] | GPR[`rsrc2];
106.       end
107.
108.       ////////////////////////////////////////////////////////////bitwise and
109.
110.       `rand : begin
111.            if(`imm_mode)
112.              GPR[`rdst]  = GPR[`rsrc1] & `isrc;
113.           else
114.             GPR[`rdst]   = GPR[`rsrc1] & GPR[`rsrc2];
115.       end
116.
117.       ////////////////////////////////////////////////////////// bitwise xor
118.
119.       `rxor : begin
```

```
120.              if(`imm_mode)
121.                GPR[`rdst]  = GPR[`rsrc1] ^ `isrc;
122.             else
123.               GPR[`rdst]   = GPR[`rsrc1] ^ GPR[`rsrc2];
124.       end
125.
126.       ////////////////////////////////////////////////////////// bitwise xnor
127.
128.       `rxnor : begin
129.             if(`imm_mode)
130.               GPR[`rdst]  = GPR[`rsrc1] ~^ `isrc;
131.             else
132.               GPR[`rdst]   = GPR[`rsrc1] ~^ GPR[`rsrc2];
133.       end
134.
135.       ////////////////////////////////////////////////////////// bitwisw nand
136.
137.       `rnand : begin
138.             if(`imm_mode)
139.               GPR[`rdst]  = ~(GPR[`rsrc1] & `isrc);
140.             else
141.               GPR[`rdst]   = ~(GPR[`rsrc1] & GPR[`rsrc2]);
142.       end
143.
144.       ////////////////////////////////////////////////////////////bitwise nor
145.
146.       `rnor : begin
147.             if(`imm_mode)
148.               GPR[`rdst]  = ~(GPR[`rsrc1] | `isrc);
149.             else
150.               GPR[`rdst]   = ~(GPR[`rsrc1] | GPR[`rsrc2]);
151.       end
152.
153.       ////////////////////////////////////////////////////////////not
154.
155.       `rnot : begin
156.             if(`imm_mode)
157.               GPR[`rdst]  = ~(`isrc);
158.             else
159.               GPR[`rdst]   = ~(GPR[`rsrc1]);
160.       end
161.
162.       //////////////////////////////////////////////////////////
163.
164.       endcase
165.       end
166.
167.
168.
169.       ///////////////////////logic for condition flag
170.       reg sign = 0, zero = 0, overflow = 0, carry = 0;
171.       reg [16:0] temp_sum;
172.
173.       always@(*)
174.       begin
175.
176.       /////////////////sign bit
177.       if(`oper_type == `mul)
178.         sign = SGPR[15];
179.       else
180.         sign = GPR[`rdst][15];
```

```
181.
182.       ///////////////carry bit
183.
184.       if(`oper_type == `add)
185.          begin
186.             if(`imm_mode)
187.                begin
188.                temp_sum = GPR[`rsrc1] + `isrc;
189.                carry    = temp_sum[16];
190.                end
191.             else
192.                begin
193.                temp_sum = GPR[`rsrc1] + GPR[`rsrc2];
194.                carry    = temp_sum[16];
195.                end   end
196.          else
197.           begin
198.               carry  = 1'b0;
199.           end
200.
201.       ////////////////////// zero bit
202.       if(`oper_type == `mul)
203.         zero =  ~((|SGPR[15:0]) | (|GPR[`rdst]));
204.       else
205.         zero =  ~(|GPR[`rdst]);
206.
207.
208.       ///////////////////////overflow bit
209.
210.       if(`oper_type == `add)
211.            begin
212.              if(`imm_mode)
213.                overflow = ( (~GPR[`rsrc1][15] & ~IR[15] & GPR[`rdst][15] ) | (GPR[`rsrc1][15]
    & IR[15] & ~GPR[`rdst][15]) );
214.              else
215.                overflow = ( (~GPR[`rsrc1][15] & ~GPR[`rsrc2][15] & GPR[`rdst][15]) |
    (GPR[`rsrc1][15] & GPR[`rsrc2][15] & ~GPR[`rdst][15]));
216.            end
217.         else if(`oper_type == `sub)
218.           begin
219.              if(`imm_mode)
220.                overflow = ( (~GPR[`rsrc1][15] & IR[15] & GPR[`rdst][15] ) | (GPR[`rsrc1][15]
    & ~IR[15] & ~GPR[`rdst][15]) );
221.              else
222.                overflow = ( (~GPR[`rsrc1][15] & GPR[`rsrc2][15] & GPR[`rdst][15]) |
    (GPR[`rsrc1][15] & ~GPR[`rsrc2][15] & ~GPR[`rdst][15]));
223.           end
224.         else
225.            begin
226.            overflow = 1'b0;
227.            end
228.
229.       end
230.
231.
232.
233.       endmodule
234.
235.       ////////////////////////////////////////////////////////////////////////////////
```