```
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. ////////////////////// load & store instructions
30.
31. `define storereg       5'b01101   //////store content of register in data memory
32. `define storedin       5'b01110   ////// store content of din bus in data memory
33. `define senddout       5'b01111   /////send data from DM to dout bus
34. `define sendreg        5'b10001   ////// send data from DM to register
35.
36.
37.
38. module top(
39. input clk,sys_rst,
40. input [15:0] din,
41. output reg [15:0] dout
42. );
43.
44. ////////////////adding program and data memory
45. reg [31:0] inst_mem [15:0]; ////program memory
46. reg [15:0] data_mem [15:0]; ////data memory
47.
48.
49.
50.
51.
52. reg [31:0] IR;            ////// instruction register  <--ir[31:27]--><--ir[26:22]--><--
    ir[21:17]--><--ir[16]--><--ir[15:11]--><--ir[10:0]-->
53.                          //////fields                 <---  oper  --><--   rdest --><--
    rsrc1 --><--modesel--><--  rsrc2 --><--unused  -->
54.                          //////fields                 <---  oper  --><--   rdest --><--
    rsrc1 --><--modesel--><--  immediate_date      -->   2^15
55.
56. reg [15:0] GPR [31:0] ;   ///////general purpose register gpr[0] ..... gpr[31]
57.
58.
```

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

```
120.
121.       `ror : begin
122.             if(`imm_mode)
123.               GPR[`rdst]  = GPR[`rsrc1] | `isrc;
124.            else
125.              GPR[`rdst]   = GPR[`rsrc1] | GPR[`rsrc2];
126.       end
127.
128.       ////////////////////////////////////////////////////////////bitwise and
129.
130.       `rand : begin
131.             if(`imm_mode)
132.               GPR[`rdst]  = GPR[`rsrc1] & `isrc;
133.            else
134.              GPR[`rdst]   = GPR[`rsrc1] & GPR[`rsrc2];
135.       end
136.
137.       ////////////////////////////////////////////////////////// bitwise xor
138.
139.       `rxor : begin
140.             if(`imm_mode)
141.               GPR[`rdst]  = GPR[`rsrc1] ^ `isrc;
142.            else
143.              GPR[`rdst]   = GPR[`rsrc1] ^ GPR[`rsrc2];
144.       end
145.
146.       ////////////////////////////////////////////////////////// bitwise xnor
147.
148.       `rxnor : begin
149.             if(`imm_mode)
150.               GPR[`rdst]  = GPR[`rsrc1] ~^ `isrc;
151.            else
152.               GPR[`rdst]   = GPR[`rsrc1] ~^ GPR[`rsrc2];
153.       end
154.
155.       ////////////////////////////////////////////////////////// bitwisw nand
156.
157.       `rnand : begin
158.             if(`imm_mode)
159.               GPR[`rdst]  = ~(GPR[`rsrc1] & `isrc);
160.            else
161.              GPR[`rdst]   = ~(GPR[`rsrc1] & GPR[`rsrc2]);
162.       end
163.
164.       ////////////////////////////////////////////////////////////bitwise nor
165.
166.       `rnor : begin
167.             if(`imm_mode)
168.               GPR[`rdst]  = ~(GPR[`rsrc1] | `isrc);
169.            else
170.              GPR[`rdst]   = ~(GPR[`rsrc1] | GPR[`rsrc2]);
171.       end
172.
173.       ////////////////////////////////////////////////////////////not
174.
175.       `rnot : begin
176.             if(`imm_mode)
177.               GPR[`rdst]  = ~(`isrc);
178.            else
179.               GPR[`rdst]   = ~(GPR[`rsrc1]);
180.       end
```

```
181.
182.       ////////////////////////////////////////////////////////
183.
184.       `storedin: begin
185.          data_mem[`isrc] = din;
186.       end
187.
188.       ////////////////////////////////////////////////////////////
189.
190.       `storereg: begin
191.          data_mem[`isrc] = GPR[`rsrc1];
192.       end
193.
194.       ////////////////////////////////////////////////////////////
195.
196.
197.       `senddout: begin
198.          dout  = data_mem[`isrc];
199.       end
200.
201.       ////////////////////////////////////////////////////////////
202.
203.       `sendreg: begin
204.         GPR[`rdst] =  data_mem[`isrc];
205.       end
206.
207.       ////////////////////////////////////////////////////////////
208.       endcase
209.       end
210.       endtask
211.
212.
213.
214.       ///////////////////////logic for condition flag
215.       reg sign = 0, zero = 0, overflow = 0, carry = 0;
216.       reg [16:0] temp_sum;
217.
218.       task decode_condflag();
219.       begin
220.
221.       /////////////////sign bit
222.       if(`oper_type == `mul)
223.         sign = SGPR[15];
224.       else
225.         sign = GPR[`rdst][15];
226.
227.       ///////////////carry bit
228.
229.       if(`oper_type == `add)
230.          begin
231.             if(`imm_mode)
232.                begin
233.                temp_sum = GPR[`rsrc1] + `isrc;
234.                carry    = temp_sum[16];
235.                end
236.             else
237.                begin
238.                temp_sum = GPR[`rsrc1] + GPR[`rsrc2];
239.                carry    = temp_sum[16];
240.                end   end
241.          else
```

```
242.           begin
243.               carry  = 1'b0;
244.           end
245.
246.       ////////////////////// zero bit
247.
248.        zero =  ( ~(|GPR[`rdst]) | ~(|SGPR[15:0]) )  ;
249.
250.
251.       ///////////////////////overflow bit
252.
253.       if(`oper_type == `add)
254.            begin
255.              if(`imm_mode)
256.                overflow = ( (~GPR[`rsrc1][15] & ~IR[15] & GPR[`rdst][15] ) | (GPR[`rsrc1][15]
    & IR[15] & ~GPR[`rdst][15]) );
257.              else
258.                overflow = ( (~GPR[`rsrc1][15] & ~GPR[`rsrc2][15] & GPR[`rdst][15]) |
    (GPR[`rsrc1][15] & GPR[`rsrc2][15] & ~GPR[`rdst][15]));
259.            end
260.         else if(`oper_type == `sub)
261.           begin
262.              if(`imm_mode)
263.                overflow = ( (~GPR[`rsrc1][15] & IR[15] & GPR[`rdst][15] ) | (GPR[`rsrc1][15]
    & ~IR[15] & ~GPR[`rdst][15]) );
264.              else
265.                overflow = ( (~GPR[`rsrc1][15] & GPR[`rsrc2][15] & GPR[`rdst][15]) |
    (GPR[`rsrc1][15] & ~GPR[`rsrc2][15] & ~GPR[`rdst][15]));
266.           end
267.         else
268.            begin
269.            overflow = 1'b0;
270.            end
271.
272.       end
273.       endtask
274.
275.
276.
277.
278.
279.       ////////////////////////////////////////////////////////////////////////////////
    //////////////////////
280.
281.       ////////////////////////////////////////////
282.       ///////////reading program
283.
284.       initial begin
285.       $readmemb("C:/Users/kumar/proc_part2/proc_part2.srcs/sources_1/new/inst_data.mem",inst_
    mem);
286.       end
287.
288.       ////////////////////////////////////////////////
289.       //////////reading instructions one after another
290.       reg [2:0] count = 0;
291.       integer PC = 0;
292.
293.       always@(posedge clk)
294.       begin
295.         if(sys_rst)
296.          begin
```

```
            count <= 0;
            PC    <= 0;
          end
          else
          begin
            if(count < 4)
            begin
            count <= count + 1;
            end
            else
            begin
            count <= 0;
            PC    <= PC + 1;
            end
        end
      end
      ////////////////////////////////////////////////
      /////////reading instructions

      always@(*)
      begin
      if(sys_rst == 1'b1)
      IR = 0;
      else
      begin
      IR = inst_mem[PC];
      decode_inst();
      decode_condflag();
      end
      end

      ////////////////////////////////////////////////


      endmodule
```