

```
1: /**
2:  * op code
3:  */
4: // I-style
5: `define OP_ADDI 6'd1
6: `define OP_LUI 6'd3
7: `define OP_ANDI 6'd4
8: `define OP_ORI 6'd5
9: `define OP_XORI 6'd6
10: `define OP_LW 6'd16
11: `define OP_LH 6'd18
12: `define OP_LB 6'd20
13: `define OP_SW 6'd24
14: `define OP_SH 6'd26
15: `define OP_SB 6'd28
16: `define OP_BEQ 6'd32
17: `define OP_BNE 6'd33
18: `define OP_BLT 6'd34
19: `define OP_BLE 6'd35
20:
21: // A-style
22: `define OP_J 6'd40
23: `define OP_JAL 6'd41
24:
25: // R-style
26: `define OP_R 6'd0 // use aux
27: `define OP_JR 6'd42
28:
29:
30: /**
31:  * alu control bits (R-style aux[4:0])
32:  */
33: `define ALU_ADD 5'd0
34: `define ALU_SUB 5'd2
35: `define ALU_AND 5'd8
36: `define ALU_OR 5'd9
37: `define ALU_XOR 5'd10
38: `define ALU_NOR 5'd11
39: `define ALU_SLL 5'd16
40: `define ALU_SRL 5'd17
41: `define ALU_SRA 5'd18
42:
43:
44: /**
45:  * memory access bits
46:  */
47: `define WORD 2'b11
48: `define HALFWORD 2'b10
49: `define BYTE 2'b01
50: `define MEM_NONE 2'b00
51:
52:
53: /**
54:  * forwarding bits
55:  */
56: `define FORWARD_NONE 2'b00
57: `define FORWARD_MEM 2'b10
58: `define FORWARD_WB 2'b01
```

```
1: module IF(input clk_i,
2:            input n_rst_i,
3:            input ID_stall_i,
4:            input [31:0] MEM_pc_branched_i,
5:            input MEM_do_branch_i,
6:            output reg [31:0] IFID_pc_o,
7:            output reg [31:0] IFID_ir_o);
8:
9:    reg [31:0] _pc;
10:   reg [31:0] _ins_mem[0:255];
11:
12:   initial $readmemb("ins_mem.bnr", _ins_mem);
13:
14:   wire [31:0] _next_pc;
15:   wire [31:0] _next_pc_plus4;
16:   wire [31:0] _ins;
17:
18:   assign _next_pc = next_pc(IFID_pc_o,
19:                             MEM_pc_branched_i,
20:                             MEM_do_branch_i);
21:   assign _next_pc_plus4 = _next_pc + 4;
22:   assign _ins = _ins_mem[_next_pc >> 2];
23:
24:   always @(negedge n_rst_i or posedge clk_i) begin
25:       if (~n_rst_i) begin
26:           _pc <= 0;
27:           IFID_pc_o <= 0;
28:           IFID_ir_o <= 32'hxxxxxxxx;
29:       end else if (clk_i) begin
30:           _pc <= _next_pc;
31:           if (~ID_stall_i) begin
32:               IFID_pc_o <= _next_pc_plus4;
33:               IFID_ir_o <= _ins;
34:           end
35:       end
36:   end
37:
38:   function [31:0] next_pc;
39:       input [31:0] pc_plus4;
40:       input [31:0] pc_branched;
41:       input do_branch;
42:       if (do_branch) begin
43:           next_pc = pc_branched;
44:       end else begin
45:           next_pc = pc_plus4;
46:       end
47:   endfunction
48:
49: endmodule
```

```

1: `include "header.v"
2:
3: module ID(input clk_i,
4:           input n_rst_i,
5:           input [31:0] IFID_pc_i,
6:           input [31:0] IFID_ir_i,
7:           input MEM_do_branch_i,
8:           input [4:0] WB_reg_write_address_i,
9:           input [31:0] WB_reg_write_data_i,
10:          input WB_ctrl_reg_write_i,
11:          output reg [31:0] IDEX_pc_o,
12:          output reg [31:0] IDEX_ir_o,
13:          output reg [31:0] IDEX_a_o,
14:          output reg [31:0] IDEX_b_o,
15:          output reg IDEX_ctrl_reg_dst_o,
16:          output reg IDEX_ctrl_alu_src_o,
17:          output reg IDEX_ctrl_branch_o,
18:          output reg [1:0] IDEX_ctrl_mem_read_o, // word, half-word, byte
19:          output reg [1:0] IDEX_ctrl_mem_write_o, // word, half-word, byte
20:          output reg IDEX_ctrl_reg_write_o,
21:          output reg IDEX_ctrl_mem_to_reg_o,
22:          output ID_stall_o);
23:
24:   wire [4:0] _rs;
25:   wire [4:0] _rt;
26:   wire [31:0] _reg_read_data1;
27:   wire [31:0] _reg_read_data2;
28:
29:   assign _rs = IFID_ir_i[25:21];
30:   assign _rt = IFID_ir_i[20:16];
31:
32:   register register(.clk_i(clk_i),
33:                   .n_rst_i(n_rst_i),
34:                   .read_address1_i(_rs),
35:                   .read_address2_i(_rt),
36:                   .write_address_i(WB_reg_write_address_i),
37:                   .write_data_i(WB_reg_write_data_i),
38:                   .ctrl_reg_write_i(WB_ctrl_reg_write_i),
39:                   .read_data1_o(_reg_read_data1),
40:                   .read_data2_o(_reg_read_data2));
41:
42:   wire _ctrl_reg_dst;
43:   wire _ctrl_alu_src;
44:   wire _ctrl_branch;
45:   wire [1:0] _ctrl_mem_read;
46:   wire [1:0] _ctrl_mem_write;
47:   wire _ctrl_reg_write;
48:   wire _ctrl_mem_to_reg;
49:
50:   control_unit control_unit(.ir_i(IFID_ir_i),
51:                           .reg_dst_o(_ctrl_reg_dst),
52:                           .alu_src_o(_ctrl_alu_src),
53:                           .branch_o(_ctrl_branch),
54:                           .mem_read_o(_ctrl_mem_read),
55:                           .mem_write_o(_ctrl_mem_write),
56:                           .reg_write_o(_ctrl_reg_write),
57:                           .mem_to_reg_o(_ctrl_mem_to_reg));
58:
59:   hazard_unit hazard_unit(.EX_ctrl_mem_read_i(IDEX_ctrl_mem_read_o),
60:                          .EX_rt_i(IDEX_ir_o[20:16]),
61:                          .ID_rs_i(_rs),
62:                          .ID_rt_i(_rt),
63:                          .stall_o(ID_stall_o));
64:

```

```

65:   always @(negedge n_rst_i or posedge clk_i) begin
66:       if (~n_rst_i) begin
67:           IDEX_pc_o <= 0;
68:           IDEX_ir_o <= 0;
69:           IDEX_a_o <= 0;
70:           IDEX_b_o <= 0;
71:           IDEX_ctrl_reg_dst_o <= 0;
72:           IDEX_ctrl_alu_src_o <= 0;
73:           IDEX_ctrl_branch_o <= 0;
74:           IDEX_ctrl_mem_read_o <= 0;
75:           IDEX_ctrl_mem_write_o <= 0;
76:           IDEX_ctrl_reg_write_o <= 0;
77:           IDEX_ctrl_mem_to_reg_o <= 0;
78:       end else if (clk_i) begin
79:           IDEX_pc_o <= IFID_pc_i;
80:           IDEX_ir_o <= IFID_ir_i;
81:           IDEX_a_o <= _reg_read_data1;
82:           IDEX_b_o <= _reg_read_data2;
83:           if (ID_stall_o || MEM_do_branch_i) begin
84:               IDEX_ctrl_reg_dst_o <= 0;
85:               IDEX_ctrl_alu_src_o <= 0;
86:               IDEX_ctrl_branch_o <= 0;
87:               IDEX_ctrl_mem_read_o <= 0;
88:               IDEX_ctrl_mem_write_o <= 0;
89:               IDEX_ctrl_reg_write_o <= 0;
90:               IDEX_ctrl_mem_to_reg_o <= 0;
91:           end else begin
92:               IDEX_ctrl_reg_dst_o <= _ctrl_reg_dst;
93:               IDEX_ctrl_alu_src_o <= _ctrl_alu_src;
94:               IDEX_ctrl_branch_o <= _ctrl_branch;
95:               IDEX_ctrl_mem_read_o <= _ctrl_mem_read;
96:               IDEX_ctrl_mem_write_o <= _ctrl_mem_write;
97:               IDEX_ctrl_reg_write_o <= _ctrl_reg_write;
98:               IDEX_ctrl_mem_to_reg_o <= _ctrl_mem_to_reg;
99:           end
100:       end
101:   end
102: endmodule
103:
104: module register(input clk_i,
105:               input n_rst_i,
106:               input [4:0] read_address1_i,
107:               input [4:0] read_address2_i,
108:               input [4:0] write_address_i,
109:               input [31:0] write_data_i,
110:               input ctrl_reg_write_i,
111:               output [31:0] read_data1_o,
112:               output [31:0] read_data2_o);
113:
114:
115:   reg [31:0] _regs[0:31];
116:
117:   assign read_data1_o = _regs[read_address1_i];
118:   assign read_data2_o = _regs[read_address2_i];
119:
120:   always @(negedge n_rst_i) begin
121:       _regs[0] <= 0;
122:   end
123:
124:   always @(negedge clk_i) begin
125:       if (ctrl_reg_write_i) begin
126:           _regs[write_address_i] <= write_data_i;
127:       end
128:   end

```

```
129:
130: endmodule
131:
132: module control_unit(input [31:0] ir_i,
133:                    output reg_dst_o,
134:                    output alu_src_o,
135:                    output branch_o,
136:                    output [1:0] mem_read_o,
137:                    output [1:0] mem_write_o,
138:                    output reg_write_o,
139:                    output mem_to_reg_o);
140:
141: wire [5:0] _op;
142: assign _op = ir_i[31:26];
143:
144: assign reg_dst_o = (_op == 'OP_R);
145: assign alu_src_o = alu_src(_op);
146: assign branch_o = branch(_op);
147: assign mem_read_o = mem_read(_op);
148: assign mem_write_o = mem_write(_op);
149: assign reg_write_o = reg_write(_op);
150: assign mem_to_reg_o = mem_to_reg(_op);
151:
152: function alu_src;
153:   input [5:0] op;
154:   case (op)
155:     'OP_ADDI, 'OP_ANDI, 'OP_ORI, 'OP_XORI, 'OP_LW, 'OP_LH, 'OP_LB, 'OP_SW,
'OP_SH, 'OP_SB:
156:       alu_src = 1;
157:     default: alu_src = 0;
158:   endcase
159: endfunction
160:
161: function branch;
162:   input [5:0] op;
163:   case (op)
164:     'OP_BEQ, 'OP_BNE, 'OP_BLT, 'OP_BLE, 'OP_J, 'OP_JAL, 'OP_JR:
165:       branch = 1;
166:     default: branch = 0;
167:   endcase
168: endfunction
169:
170: function [1:0] mem_read;
171:   input [5:0] op;
172:   case (op)
173:     'OP_LW: mem_read = 'WORD;
174:     'OP_LH: mem_read = 'HALFWORD;
175:     'OP_LB: mem_read = 'BYTE;
176:     default: mem_read = 'MEM_NONE;
177:   endcase
178: endfunction
179:
180: function [1:0] mem_write;
181:   input [5:0] op;
182:   case (op)
183:     'OP_SW: mem_write = 'WORD;
184:     'OP_SH: mem_write = 'HALFWORD;
185:     'OP_SB: mem_write = 'BYTE;
186:     default: mem_write = 'MEM_NONE;
187:   endcase
188: endfunction
189:
190: function reg_write;
191:   input [5:0] op;
```

```
192:   case (op)
193:     'OP_R, 'OP_ADDI, 'OP_LUI, 'OP_ANDI, 'OP_ORI, 'OP_XORI, 'OP_LW, 'OP_LH,
'OP_LB, 'OP_JAL:
194:       reg_write = 1;
195:     default: reg_write = 0;
196:   endcase
197: endfunction
198:
199: function mem_to_reg;
200:   input [5:0] op;
201:   case (op)
202:     'OP_LW, 'OP_LH, 'OP_LB: mem_to_reg = 1;
203:     default: mem_to_reg = 0;
204:   endcase
205: endfunction
206:
207: endmodule
208:
209: module hazard_unit(input [1:0] EX_ctrl_mem_read_i,
210:                  input [4:0] EX_rt_i,
211:                  input [4:0] ID_rs_i,
212:                  input [4:0] ID_rt_i,
213:                  output stall_o);
214:
215:   assign stall_o = EX_ctrl_mem_read_i != 'MEM_NONE
216:                   && (EX_rt_i == ID_rs_i || EX_rt_i == ID_rt_i);
217:
218: endmodule
```

```

1: `include "header.v"
2:
3: module EX(input clk_i,
4:            input n_rst_i,
5:            input [31:0] IDEX_pc_i,
6:            input [31:0] IDEX_ir_i,
7:            input [31:0] IDEX_a_i,
8:            input [31:0] IDEX_b_i,
9:            input IDEX_ctrl_reg_dst_i,
10:           input IDEX_ctrl_alu_src_i,
11:           input IDEX_ctrl_branch_i,
12:           input [1:0] IDEX_ctrl_mem_read_i,
13:           input [1:0] IDEX_ctrl_mem_write_i,
14:           input IDEX_ctrl_reg_write_i,
15:           input IDEX_ctrl_mem_to_reg_i,
16:           input MEM_do_branch_i,
17:           input [4:0] WB_reg_write_address_i,
18:           input [31:0] WB_reg_write_data_i,
19:           input WB_ctrl_reg_write_i,
20:           output reg [31:0] EXMEM_pc_branched_o,
21:           output reg [31:0] EXMEM_alu_o,
22:           output reg EXMEM_alu_do_branch_o,
23:           output reg [31:0] EXMEM_b_o,
24:           output reg [4:0] EXMEM_reg_write_address_o,
25:           output reg EXMEM_ctrl_branch_o,
26:           output reg [1:0] EXMEM_ctrl_mem_read_o,
27:           output reg [1:0] EXMEM_ctrl_mem_write_o,
28:           output reg EXMEM_ctrl_reg_write_o,
29:           output reg EXMEM_ctrl_mem_to_reg_o);
30:
31: wire [5:0] _op;
32: wire [4:0] _rs;
33: wire [4:0] _rt;
34: wire [4:0] _rd;
35: wire [4:0] _shift;
36: wire [4:0] _aux;
37: wire [31:0] _imm_dpl;
38: wire [25:0] _addr;
39:
40: assign _op = IDEX_ir_i[31:26];
41: assign _rs = IDEX_ir_i[25:21];
42: assign _rt = IDEX_ir_i[20:16];
43: assign _rd = IDEX_ir_i[15:11];
44: assign _shift = IDEX_ir_i[10:6];
45: assign _aux = IDEX_ir_i[4:0];
46: assign _imm_dpl = {{16{IDEX_ir_i[15]}}}, IDEX_ir_i[15:0];
47: assign _addr = IDEX_ir_i[25:0];
48:
49: wire [31:0] _a;
50: wire [31:0] _b;
51: wire [31:0] _alu_b;
52: wire [31:0] _alu;
53: wire _alu_zero;
54: wire _alu_sign;
55: wire _alu_do_branch;
56:
57: assign _a = forward_mux(IDEX_a_i,
58:                        EXMEM_alu_o,
59:                        WB_reg_write_data_i,
60:                        _forward_a);
61: assign _b = forward_mux(IDEX_b_i,
62:                        EXMEM_alu_o,
63:                        WB_reg_write_data_i,
64:                        _forward_b);

```

```

65: assign _alu_b = alu_b(_b,
66:                    _imm_dpl,
67:                    IDEX_ctrl_alu_src_i);
68:
69: assign _alu = (_op == `OP_LUI) ? (_imm_dpl << 16) :
70:              (_op == `OP_JAL) ? IDEX_pc_i :
71:              alu(alu_ctrl(_op, _aux),
72:                 _a,
73:                 _alu_b,
74:                 _shift);
75: assign _alu_zero = ~|_alu;
76: assign _alu_sign = _alu[31];
77: assign _alu_do_branch = alu_do_branch(_op,
78:                                       _alu_zero,
79:                                       _alu_sign);
80:
81: wire [31:0] _pc_branched;
82: assign _pc_branched = pc_branched(_op,
83:                                  IDEX_pc_i,
84:                                  _imm_dpl,
85:                                  _addr,
86:                                  _a);
87:
88: wire [4:0] _reg_write_address;
89: assign _reg_write_address = (_op == `OP_JAL) ? 5'd31 :
90:                             (IDEX_ctrl_reg_dst_i) ? _rd : _rt;
91:
92: wire [1:0] _forward_a;
93: wire [1:0] _forward_b;
94: forwarding_unit forwarding_unit(.rs_i(_rs),
95:                                .rt_i(_rt),
96:                                .MEM_reg_write_address_i(EXMEM_reg_write_addre
97:                                                         ss_o),
98:                                .MEM_ctrl_reg_write_i(EXMEM_ctrl_reg_write_o),
99:                                .WB_reg_write_address_i(WB_reg_write_address_i
100:                                                         ),
101:                                .WB_ctrl_reg_write_i(WB_ctrl_reg_write_i),
102:                                .forward_a_o(_forward_a),
103:                                .forward_b_o(_forward_b));
104:
105: always @(negedge n_rst_i or posedge clk_i) begin
106:   if (~n_rst_i) begin
107:     EXMEM_pc_branched_o <= 0;
108:     EXMEM_alu_o <= 0;
109:     EXMEM_alu_do_branch_o <= 0;
110:     EXMEM_b_o <= 0;
111:     EXMEM_reg_write_address_o <= 0;
112:     EXMEM_ctrl_branch_o <= 0;
113:     EXMEM_ctrl_mem_read_o <= 0;
114:     EXMEM_ctrl_mem_write_o <= 0;
115:     EXMEM_ctrl_reg_write_o <= 0;
116:     EXMEM_ctrl_mem_to_reg_o <= 0;
117:   end else if (clk_i) begin
118:     EXMEM_pc_branched_o <= _pc_branched;
119:     EXMEM_alu_o <= _alu;
120:     EXMEM_alu_do_branch_o <= _alu_do_branch;
121:     EXMEM_b_o <= _b;
122:     EXMEM_reg_write_address_o <= _reg_write_address;
123:     if (MEM_do_branch_i) begin
124:       EXMEM_ctrl_branch_o <= 0;
125:       EXMEM_ctrl_mem_read_o <= 0;
126:       EXMEM_ctrl_mem_write_o <= 0;
127:       EXMEM_ctrl_reg_write_o <= 0;
128:       EXMEM_ctrl_mem_to_reg_o <= 0;

```

```

127:         end else begin
128:             EXMEM_ctrl_branch_o <= IDEX_ctrl_branch_i;
129:             EXMEM_ctrl_mem_read_o <= IDEX_ctrl_mem_read_i;
130:             EXMEM_ctrl_mem_write_o <= IDEX_ctrl_mem_write_i;
131:             EXMEM_ctrl_reg_write_o <= IDEX_ctrl_reg_write_i;
132:             EXMEM_ctrl_mem_to_reg_o <= IDEX_ctrl_mem_to_reg_i;
133:         end
134:     end
135: end
136:
137: function [31:0] pc_branched;
138:     input [5:0] op;
139:     input [31:0] pc;
140:     input [31:0] imm_dpl;
141:     input [25:0] addr;
142:     input [31:0] a;
143:     case (op)
144:         'OP_BEQ, 'OP_BNE, 'OP_BLT, 'OP_BLE: pc_branched = pc + (imm_dpl << 2);
145:         'OP_J, 'OP_JAL: pc_branched = {4'b0, addr << 2};
146:         'OP_JR: pc_branched = a;
147:         default: pc_branched = pc;
148:     endcase
149: endfunction
150:
151: function [4:0] alu_ctrl;
152:     input [5:0] op;
153:     input [4:0] aux;
154:     case (op)
155:         'OP_R: alu_ctrl = aux;
156:         'OP_ADDI: alu_ctrl = 'ALU_ADD;
157:         'OP_ANDI: alu_ctrl = 'ALU_AND;
158:         'OP_ORI: alu_ctrl = 'ALU_OR;
159:         'OP_XORI: alu_ctrl = 'ALU_XOR;
160:         'OP_BEQ, 'OP_BNE, 'OP_BLT, 'OP_BLE: alu_ctrl = 'ALU_SUB;
161:         'OP_LW, 'OP_LH, 'OP_LB, 'OP_SW, 'OP_SH, 'OP_SB: alu_ctrl = 'ALU_ADD;
162:         default: alu_ctrl = 5'h1f;
163:     endcase
164: endfunction
165:
166: function [31:0] forward_mux;
167:     input [31:0] EX_data;
168:     input [31:0] MEM_reg_write_data;
169:     input [31:0] WB_reg_write_data;
170:     input [1:0] forward;
171:     case (forward)
172:         'FORWARD_MEM: forward_mux = MEM_reg_write_data;
173:         'FORWARD_WB: forward_mux = WB_reg_write_data;
174:         default: forward_mux = EX_data;
175:     endcase
176: endfunction
177:
178: function [31:0] alu_b;
179:     input [31:0] IDEX_b;
180:     input [31:0] imm_dpl;
181:     input ctrl_alu_src;
182:     if (ctrl_alu_src) begin
183:         alu_b = imm_dpl;
184:     end else begin
185:         alu_b = IDEX_b;
186:     end
187: endfunction
188:
189: function [31:0] alu;
190:     input [4:0] ctrl;

```

```

191:     input [31:0] a;
192:     input [31:0] b;
193:     input [4:0] shift;
194:     case (ctrl)
195:         'ALU_ADD: alu = a + b;
196:         'ALU_SUB: alu = a - b;
197:         'ALU_AND: alu = a & b;
198:         'ALU_OR: alu = a | b;
199:         'ALU_XOR: alu = a ^ b;
200:         'ALU_NOR: alu = ~(a | b);
201:         'ALU_SLL: alu = a << shift;
202:         'ALU_SRL: alu = a >> shift;
203:         'ALU_SRA: alu = {{32{a[31]}}, a} >> shift;
204:         default: alu = 32'hffffffff;
205:     endcase
206: endfunction
207:
208: function alu_do_branch;
209:     input [5:0] op;
210:     input alu_zero;
211:     input alu_sign;
212:     case (op)
213:         'OP_BEQ: alu_do_branch = alu_zero;
214:         'OP_BNE: alu_do_branch = ~alu_zero;
215:         'OP_BLT: alu_do_branch = alu_sign;
216:         'OP_BLE: alu_do_branch = alu_zero | alu_sign;
217:         'OP_J, 'OP_JAL, 'OP_JR: alu_do_branch = 1;
218:         default: alu_do_branch = 0;
219:     endcase
220: endfunction
221:
222: endmodule
223:
224: module forwarding_unit(input [4:0] rs_i,
225:                       input [4:0] rt_i,
226:                       input [4:0] MEM_reg_write_address_i,
227:                       input MEM_ctrl_reg_write_i,
228:                       input [4:0] WB_reg_write_address_i,
229:                       input WB_ctrl_reg_write_i,
230:                       output [1:0] forward_a_o,
231:                       output [1:0] forward_b_o);
232:
233:     assign forward_a_o = forward_a(rs_i,
234:                                     MEM_reg_write_address_i,
235:                                     MEM_ctrl_reg_write_i,
236:                                     WB_reg_write_address_i,
237:                                     WB_ctrl_reg_write_i);
238:
239:     assign forward_b_o = forward_b(rt_i,
240:                                     MEM_reg_write_address_i,
241:                                     MEM_ctrl_reg_write_i,
242:                                     WB_reg_write_address_i,
243:                                     WB_ctrl_reg_write_i);
244:
245:     function [1:0] forward_a;
246:         input [4:0] rs;
247:         input [4:0] MEM_reg_write_address;
248:         input MEM_ctrl_reg_write;
249:         input [4:0] WB_reg_write_address;
250:         input WB_ctrl_reg_write;
251:         if (MEM_ctrl_reg_write
252:             && MEM_reg_write_address != 0
253:             && MEM_reg_write_address == rs) begin
254:             forward_a = 'FORWARD_MEM;

```

```
255:         end else if (WB_ctrl_reg_write
256:                       && WB_reg_write_address != 0
257:                       && WB_reg_write_address == rs) begin
258:             forward_a = 'FORWARD_WB;
259:         end else begin
260:             forward_a = 'FORWARD_NONE;
261:         end
262:     endfunction
263:
264:     function [1:0] forward_b;
265:         input [4:0] rt;
266:         input [4:0] MEM_reg_write_address;
267:         input MEM_ctrl_reg_write;
268:         input [4:0] WB_reg_write_address;
269:         input WB_ctrl_reg_write;
270:         if (MEM_ctrl_reg_write
271:           && MEM_reg_write_address != 0
272:           && MEM_reg_write_address == rt) begin
273:             forward_b = 'FORWARD_MEM;
274:         end else if (WB_ctrl_reg_write
275:           && WB_reg_write_address != 0
276:           && WB_reg_write_address == rt) begin
277:             forward_b = 'FORWARD_WB;
278:         end else begin
279:             forward_b = 'FORWARD_NONE;
280:         end
281:     endfunction
282:
283: endmodule
```

```

1: `include "header.v"
2:
3: module MEM(input clk_i,
4:             input n_rst_i,
5:             input [31:0] EXMEM_pc_branched_i,
6:             input [31:0] EXMEM_alu_i,
7:             input EXMEM_alu_do_branch_i,
8:             input [31:0] EXMEM_b_i,
9:             input [4:0] EXMEM_reg_write_address_i,
10:            input EXMEM_ctrl_branch_i,
11:            input [1:0] EXMEM_ctrl_mem_read_i,
12:            input [1:0] EXMEM_ctrl_mem_write_i,
13:            input EXMEM_ctrl_reg_write_i,
14:            input EXMEM_ctrl_mem_to_reg_i,
15:            output reg [31:0] MEMWB_mem_o,
16:            output reg [31:0] MEMWB_alu_o,
17:            output reg [4:0] MEMWB_reg_write_address_o,
18:            output reg MEMWB_ctrl_reg_write_o,
19:            output reg MEMWB_ctrl_mem_to_reg_o,
20:            output [31:0] MEM_pc_branched_o,
21:            output MEM_do_branch_o);
22:
23: assign MEM_pc_branched_o = EXMEM_pc_branched_i;
24: assign MEM_do_branch_o = (EXMEM_ctrl_branch_i & EXMEM_alu_do_branch_i);
25:
26: wire [31:0] _mem_read_data;
27:
28: memory memory(.clk_i(clk_i),
29:               .address_i(EXMEM_alu_i[7:0]),
30:               .write_data_i(EXMEM_b_i),
31:               .ctrl_mem_read_i(EXMEM_ctrl_mem_read_i),
32:               .ctrl_mem_write_i(EXMEM_ctrl_mem_write_i),
33:               .read_data_o(_mem_read_data));
34:
35: always @(negedge n_rst_i or posedge clk_i) begin
36:     if (~n_rst_i) begin
37:         MEMWB_mem_o <= 0;
38:         MEMWB_alu_o <= 0;
39:         MEMWB_reg_write_address_o <= 0;
40:         MEMWB_ctrl_reg_write_o <= 0;
41:         MEMWB_ctrl_mem_to_reg_o <= 0;
42:     end else if (clk_i) begin
43:         MEMWB_mem_o <= _mem_read_data;
44:         MEMWB_alu_o <= EXMEM_alu_i;
45:         MEMWB_reg_write_address_o <= EXMEM_reg_write_address_i;
46:         MEMWB_ctrl_reg_write_o <= EXMEM_ctrl_reg_write_i;
47:         MEMWB_ctrl_mem_to_reg_o <= EXMEM_ctrl_mem_to_reg_i;
48:     end
49: end
50:
51: endmodule
52:
53: module memory(input clk_i,
54:               input [7:0] address_i,
55:               input [31:0] write_data_i,
56:               input [1:0] ctrl_mem_read_i,
57:               input [1:0] ctrl_mem_write_i,
58:               output [31:0] read_data_o);
59:
60: reg [31:0] _mem[0:255];
61:
62: assign read_data_o = read(_mem[address_i], ctrl_mem_read_i);
63:
64: always @(negedge clk_i) begin

```

```

65:     case (ctrl_mem_write_i)
66:         `WORD: _mem[address_i] <= write_data_i;
67:         `HALFWORD: _mem[address_i][15:0] <= write_data_i[15:0];
68:         `BYTE: _mem[address_i][7:0] <= write_data_i[7:0];
69:     endcase
70: end
71:
72: function [31:0] read;
73:     input [31:0] data;
74:     input [1:0] ctrl;
75:     case (ctrl)
76:         `WORD: read = data;
77:         `HALFWORD: read = {{16{data[15]}}, data[15:0]};
78:         `BYTE: read = {{24{data[7]}}, data[7:0]};
79:         default: read = 0;
80:     endcase
81: endfunction
82:
83: endmodule

```



```
1: module WB(input clk_i,
2:            input n_rst_i,
3:            input [31:0] MEMWB_mem_i,
4:            input [31:0] MEMWB_alu_i,
5:            input [4:0] MEMWB_reg_write_address_i,
6:            input MEMWB_ctrl_reg_write_i,
7:            input MEMWB_ctrl_mem_to_reg_i,
8:            output [4:0] WB_reg_write_address_o,
9:            output [31:0] WB_reg_write_data_o,
10:           output WB_ctrl_reg_write_o);
11:
12:   assign WB_reg_write_address_o = MEMWB_reg_write_address_i;
13:   assign WB_reg_write_data_o = (MEMWB_ctrl_mem_to_reg_i) ? MEMWB_a
lu_i;
14:   assign WB_ctrl_reg_write_o = MEMWB_ctrl_reg_write_i;
15:
16: endmodule
```

```
1: module processor;
2:   reg clk;
3:   reg n_rst;
4:
5:   initial begin
6:     clk <= 0;
7:     n_rst <= 1;
8:     #10 n_rst <= 0;
9:     #10 n_rst <= 1;
10:    #10000 $stop;
11:   end
12:
13:   always #50 clk = ~clk;
14:
15:   // IF
16:   wire [31:0] IFID_pc;
17:   wire [31:0] IFID_ir;
18:
19:   // ID
20:   wire [31:0] IDEX_pc;
21:   wire [31:0] IDEX_ir;
22:   wire [31:0] IDEX_a;
23:   wire [31:0] IDEX_b;
24:   wire IDEX_ctrl_reg_dst;
25:   wire IDEX_ctrl_alu_src;
26:   wire IDEX_ctrl_branch;
27:   wire [1:0] IDEX_ctrl_mem_read;
28:   wire [1:0] IDEX_ctrl_mem_write;
29:   wire IDEX_ctrl_reg_write;
30:   wire IDEX_ctrl_mem_to_reg;
31:   wire ID_stall;
32:
33:   // EX
34:   wire [31:0] EXMEM_pc_branched;
35:   wire [31:0] EXMEM_alu;
36:   wire EXMEM_alu_do_branch;
37:   wire [31:0] EXMEM_b;
38:   wire [4:0] EXMEM_reg_write_address;
39:   wire EXMEM_ctrl_branch;
40:   wire [1:0] EXMEM_ctrl_mem_read;
41:   wire [1:0] EXMEM_ctrl_mem_write;
42:   wire EXMEM_ctrl_reg_write;
43:   wire EXMEM_ctrl_mem_to_reg;
44:
45:   // MEM
46:   wire [31:0] MEMWB_mem;
47:   wire [31:0] MEMWB_alu;
48:   wire [4:0] MEMWB_reg_write_address;
49:   wire MEMWB_ctrl_reg_write;
50:   wire MEMWB_ctrl_mem_to_reg;
51:   wire [31:0] MEM_pc_branched;
52:   wire MEM_do_branch;
53:
54:   // WB
55:   wire [4:0] WB_reg_write_address;
56:   wire [31:0] WB_reg_write_data;
57:   wire WB_ctrl_reg_write;
58:
59:   IF IF(.clk_i(clk),
60:     .n_rst_i(n_rst),
61:     .ID_stall_i(ID_stall),
62:     .MEM_pc_branched_i(MEM_pc_branched),
63:     .MEM_do_branch_i(MEM_do_branch),
64:     .IFID_pc_o(IFID_pc),
```

```
65:     .IFID_ir_o(IFID_ir));
66:
67:   ID ID(.clk_i(clk),
68:     .n_rst_i(n_rst),
69:     .IFID_pc_i(IFID_pc),
70:     .IFID_ir_i(IFID_ir),
71:     .MEM_do_branch_i(MEM_do_branch),
72:     .WB_reg_write_address_i(WB_reg_write_address),
73:     .WB_reg_write_data_i(WB_reg_write_data),
74:     .WB_ctrl_reg_write_i(WB_ctrl_reg_write),
75:     .IDEX_pc_o(IDEX_pc),
76:     .IDEX_ir_o(IDEX_ir),
77:     .IDEX_a_o(IDEX_a),
78:     .IDEX_b_o(IDEX_b),
79:     .IDEX_ctrl_reg_dst_o(IDEX_ctrl_reg_dst),
80:     .IDEX_ctrl_alu_src_o(IDEX_ctrl_alu_src),
81:     .IDEX_ctrl_branch_o(IDEX_ctrl_branch),
82:     .IDEX_ctrl_mem_read_o(IDEX_ctrl_mem_read),
83:     .IDEX_ctrl_mem_write_o(IDEX_ctrl_mem_write),
84:     .IDEX_ctrl_reg_write_o(IDEX_ctrl_reg_write),
85:     .IDEX_ctrl_mem_to_reg_o(IDEX_ctrl_mem_to_reg),
86:     .ID_stall_o(ID_stall));
87:
88:   EX EX(.clk_i(clk),
89:     .n_rst_i(n_rst),
90:     .IDEX_pc_i(IDEX_pc),
91:     .IDEX_ir_i(IDEX_ir),
92:     .IDEX_a_i(IDEX_a),
93:     .IDEX_b_i(IDEX_b),
94:     .IDEX_ctrl_reg_dst_i(IDEX_ctrl_reg_dst),
95:     .IDEX_ctrl_alu_src_i(IDEX_ctrl_alu_src),
96:     .IDEX_ctrl_branch_i(IDEX_ctrl_branch),
97:     .IDEX_ctrl_mem_read_i(IDEX_ctrl_mem_read),
98:     .IDEX_ctrl_mem_write_i(IDEX_ctrl_mem_write),
99:     .IDEX_ctrl_reg_write_i(IDEX_ctrl_reg_write),
100:    .IDEX_ctrl_mem_to_reg_i(IDEX_ctrl_mem_to_reg),
101:    .MEM_do_branch_i(MEM_do_branch),
102:    .WB_reg_write_address_i(WB_reg_write_address),
103:    .WB_reg_write_data_i(WB_reg_write_data),
104:    .WB_ctrl_reg_write_i(WB_ctrl_reg_write),
105:    .EXMEM_pc_branched_o(EXMEM_pc_branched),
106:    .EXMEM_alu_o(EXMEM_alu),
107:    .EXMEM_alu_do_branch_o(EXMEM_alu_do_branch),
108:    .EXMEM_b_o(EXMEM_b),
109:    .EXMEM_reg_write_address_o(EXMEM_reg_write_address),
110:    .EXMEM_ctrl_branch_o(EXMEM_ctrl_branch),
111:    .EXMEM_ctrl_mem_read_o(EXMEM_ctrl_mem_read),
112:    .EXMEM_ctrl_mem_write_o(EXMEM_ctrl_mem_write),
113:    .EXMEM_ctrl_reg_write_o(EXMEM_ctrl_reg_write),
114:    .EXMEM_ctrl_mem_to_reg_o(EXMEM_ctrl_mem_to_reg));
115:
116:   MEM MEM(.clk_i(clk),
117:     .n_rst_i(n_rst),
118:     .EXMEM_pc_branched_i(EXMEM_pc_branched),
119:     .EXMEM_alu_i(EXMEM_alu),
120:     .EXMEM_alu_do_branch_i(EXMEM_alu_do_branch),
121:     .EXMEM_b_i(EXMEM_b),
122:     .EXMEM_reg_write_address_i(EXMEM_reg_write_address),
123:     .EXMEM_ctrl_branch_i(EXMEM_ctrl_branch),
124:     .EXMEM_ctrl_mem_read_i(EXMEM_ctrl_mem_read),
125:     .EXMEM_ctrl_mem_write_i(EXMEM_ctrl_mem_write),
126:     .EXMEM_ctrl_reg_write_i(EXMEM_ctrl_reg_write),
127:     .EXMEM_ctrl_mem_to_reg_i(EXMEM_ctrl_mem_to_reg),
128:     .MEMWB_mem_o(MEMWB_mem),
```

```
129:         .MEMWB_alu_o(MEMWB_alu),
130:         .MEMWB_reg_write_address_o(MEMWB_reg_write_address),
131:         .MEMWB_ctrl_reg_write_o(MEMWB_ctrl_reg_write),
132:         .MEMWB_ctrl_mem_to_reg_o(MEMWB_ctrl_mem_to_reg),
133:         .MEM_pc_branched_o(MEM_pc_branched),
134:         .MEM_do_branch_o(MEM_do_branch));
135:
136:     WB WB(.clk_i(clk),
137:         .n_rst_i(n_rst),
138:         .MEMWB_mem_i(MEMWB_mem),
139:         .MEMWB_alu_i(MEMWB_alu),
140:         .MEMWB_reg_write_address_i(MEMWB_reg_write_address),
141:         .MEMWB_ctrl_reg_write_i(MEMWB_ctrl_reg_write),
142:         .MEMWB_ctrl_mem_to_reg_i(MEMWB_ctrl_mem_to_reg),
143:         .WB_reg_write_address_o(WB_reg_write_address),
144:         .WB_reg_write_data_o(WB_reg_write_data),
145:         .WB_ctrl_reg_write_o(WB_ctrl_reg_write));
146:
147: endmodule
```