```
/**
 * op code
 */
// I-style
‘define OP_ADDI 6’d1
‘define OP_LUI  6’d3
‘define OP_ANDI 6’d4
‘define OP_ORI  6’d5
‘define OP_XORI 6’d6
‘define OP_LW   6’d16
‘define OP_LH   6’d18
‘define OP_LB   6’d20
‘define OP_SW   6’d24
‘define OP_SH   6’d26
‘define OP_SB   6’d28
‘define OP_BEQ  6’d32
‘define OP_BNE  6’d33
‘define OP_BLT  6’d34
‘define OP_BLE  6’d35

// A-style
‘define OP_J    6’d40
‘define OP_JAL  6’d41

// R-style
‘define OP_R    6’d0           // use aux
‘define OP_JR   6’d42


/**
 * alu control bits (R-style aux[4:0])
 */
‘define ALU_ADD 5’d0
‘define ALU_SUB 5’d2
‘define ALU_AND 5’d8
‘define ALU_OR  5’d9
‘define ALU_XOR 5’d10
‘define ALU_NOR 5’d11
‘define ALU_SLL 5’d16
‘define ALU_SRL 5’d17
‘define ALU_SRA 5’d18


/**
 * memory access bits
 */
‘define WORD     2’b11
‘define HALFWORD 2’b10
‘define BYTE     2’b01
‘define MEM_NONE 2’b00


/**
 * forwarding bits
 */
‘define FORWARD_NONE 2’b00
‘define FORWARD_MEM  2’b10
‘define FORWARD_WB   2’b01
```

```verilog
module IF(input clk_i,
          input n_rst_i,
          input ID_stall_i,
          input [31:0] MEM_pc_branched_i,
          input MEM_do_branch_i,
          output reg [31:0] IFID_pc_o,
          output reg [31:0] IFID_ir_o);

    reg [31:0] _pc;
    reg [31:0] _ins_mem[0:255];

    initial $readmemb("ins_mem.bnr", _ins_mem);

    wire [31:0] _next_pc;
    wire [31:0] _next_pc_plus4;
    wire [31:0] _ins;

    assign _next_pc = next_pc(IFID_pc_o,
                              MEM_pc_branched_i,
                              MEM_do_branch_i);
    assign _next_pc_plus4 = _next_pc + 4;
    assign _ins = _ins_mem[_next_pc >> 2];

    always @(negedge n_rst_i or posedge clk_i) begin
        if (~n_rst_i) begin
            _pc <= 0;
            IFID_pc_o <= 0;
            IFID_ir_o <= 32'hxxxxxxxx;
        end else if (clk_i) begin
            _pc <= _next_pc;
            if (~ID_stall_i) begin
                IFID_pc_o <= _next_pc_plus4;
                IFID_ir_o <= _ins;
            end
        end
    end

    function [31:0] next_pc;
        input [31:0] pc_plus4;
        input [31:0] pc_branched;
        input do_branch;
        if (do_branch) begin
            next_pc = pc_branched;
        end else begin
            next_pc = pc_plus4;
        end
    endfunction

endmodule
```

```verilog
‘include "header.v"

module ID(input clk_i,
          input n_rst_i,
          input [31:0] IFID_pc_i,
          input [31:0] IFID_ir_i,
          input MEM_do_branch_i,
          input [4:0] WB_reg_write_address_i,
          input [31:0] WB_reg_write_data_i,
          input WB_ctrl_reg_write_i,
          output reg [31:0] IDEX_pc_o,
          output reg [31:0] IDEX_ir_o,
          output reg [31:0] IDEX_a_o,
          output reg [31:0] IDEX_b_o,
          output reg IDEX_ctrl_reg_dst_o,
          output reg IDEX_ctrl_alu_src_o,
          output reg IDEX_ctrl_branch_o,
          output reg [1:0] IDEX_ctrl_mem_read_o,  // word, half-word, byte
          output reg [1:0] IDEX_ctrl_mem_write_o, // word, half-word, byte
          output reg IDEX_ctrl_reg_write_o,
          output reg IDEX_ctrl_mem_to_reg_o,
          output ID_stall_o);

    wire [4:0] _rs;
    wire [4:0] _rt;
    wire [31:0] _reg_read_data1;
    wire [31:0] _reg_read_data2;

    assign _rs = IFID_ir_i[25:21];
    assign _rt = IFID_ir_i[20:16];

    register register(.clk_i(clk_i),
                      .n_rst_i(n_rst_i),
                      .read_address1_i(_rs),
                      .read_address2_i(_rt),
                      .write_address_i(WB_reg_write_address_i),
                      .write_data_i(WB_reg_write_data_i),
                      .ctrl_reg_write_i(WB_ctrl_reg_write_i),
                      .read_data1_o(_reg_read_data1),
                      .read_data2_o(_reg_read_data2));

    wire _ctrl_reg_dst;
    wire _ctrl_alu_src;
    wire _ctrl_branch;
    wire [1:0] _ctrl_mem_read;
    wire [1:0] _ctrl_mem_write;
    wire _ctrl_reg_write;
    wire _ctrl_mem_to_reg;

    control_unit control_unit(.ir_i(IFID_ir_i),
                              .reg_dst_o(_ctrl_reg_dst),
                              .alu_src_o(_ctrl_alu_src),
                              .branch_o(_ctrl_branch),
                              .mem_read_o(_ctrl_mem_read),
                              .mem_write_o(_ctrl_mem_write),
                              .reg_write_o(_ctrl_reg_write),
                              .mem_to_reg_o(_ctrl_mem_to_reg));

    hazard_unit hazard_unit(.EX_ctrl_mem_read_i(IDEX_ctrl_mem_read_o),
                            .EX_rt_i(IDEX_ir_o[20:16]),
                            .ID_rs_i(_rs),
                            .ID_rt_i(_rt),
                            .stall_o(ID_stall_o));

    always @(negedge n_rst_i or posedge clk_i) begin
        if (~n_rst_i) begin
            IDEX_pc_o <= 0;
            IDEX_ir_o <= 0;
            IDEX_a_o <= 0;
            IDEX_b_o <= 0;
            IDEX_ctrl_reg_dst_o <= 0;
            IDEX_ctrl_alu_src_o <= 0;
            IDEX_ctrl_branch_o <= 0;
            IDEX_ctrl_mem_read_o <= 0;
            IDEX_ctrl_mem_write_o <= 0;
            IDEX_ctrl_reg_write_o <= 0;
            IDEX_ctrl_mem_to_reg_o <= 0;
        end else if (clk_i) begin
            IDEX_pc_o <= IFID_pc_i;
            IDEX_ir_o <= IFID_ir_i;
            IDEX_a_o <= _reg_read_data1;
            IDEX_b_o <= _reg_read_data2;
            if (ID_stall_o || MEM_do_branch_i) begin
                IDEX_ctrl_reg_dst_o <= 0;
                IDEX_ctrl_alu_src_o <= 0;
                IDEX_ctrl_branch_o <= 0;
                IDEX_ctrl_mem_read_o <= 0;
                IDEX_ctrl_mem_write_o <= 0;
                IDEX_ctrl_reg_write_o <= 0;
                IDEX_ctrl_mem_to_reg_o <= 0;
            end else begin
                IDEX_ctrl_reg_dst_o <= _ctrl_reg_dst;
                IDEX_ctrl_alu_src_o <= _ctrl_alu_src;
                IDEX_ctrl_branch_o <= _ctrl_branch;
                IDEX_ctrl_mem_read_o <= _ctrl_mem_read;
                IDEX_ctrl_mem_write_o <= _ctrl_mem_write;
                IDEX_ctrl_reg_write_o <= _ctrl_reg_write;
                IDEX_ctrl_mem_to_reg_o <= _ctrl_mem_to_reg;
            end
        end
    end

endmodule

module register(input clk_i,
                input n_rst_i,
                input [4:0] read_address1_i,
                input [4:0] read_address2_i,
                input [4:0] write_address_i,
                input [31:0] write_data_i,
                input ctrl_reg_write_i,
                output [31:0] read_data1_o,
                output [31:0] read_data2_o);

    reg [31:0] _regs[0:31];

    assign read_data1_o = _regs[read_address1_i];
    assign read_data2_o = _regs[read_address2_i];

    always @(negedge n_rst_i) begin
        _regs[0] <= 0;
    end

    always @(negedge clk_i) begin
        if (ctrl_reg_write_i) begin
            _regs[write_address_i] <= write_data_i;
        end
    end
```

```verilog

endmodule

module control_unit(input [31:0] ir_i,
                    output reg_dst_o,
                    output alu_src_o,
                    output branch_o,
                    output [1:0] mem_read_o,
                    output [1:0] mem_write_o,
                    output reg_write_o,
                    output mem_to_reg_o);

    wire [5:0] _op;
    assign _op = ir_i[31:26];

    assign reg_dst_o = (_op == `OP_R);
    assign alu_src_o = alu_src(_op);
    assign branch_o = branch(_op);
    assign mem_read_o = mem_read(_op);
    assign mem_write_o = mem_write(_op);
    assign reg_write_o = reg_write(_op);
    assign mem_to_reg_o = mem_to_reg(_op);

    function alu_src;
        input [5:0] op;
        case (op)
            `OP_ADDI, `OP_ANDI, `OP_ORI, `OP_XORI, `OP_LW, `OP_LH, `OP_LB, `OP_SW, `OP_SH, `OP_SB:
              alu_src = 1;
            default: alu_src = 0;
        endcase
    endfunction

    function branch;
        input [5:0] op;
        case (op)
            `OP_BEQ, `OP_BNE, `OP_BLT, `OP_BLE, `OP_J, `OP_JAL, `OP_JR:
              branch = 1;
            default: branch = 0;
        endcase
    endfunction

    function [1:0] mem_read;
        input [5:0] op;
        case (op)
            `OP_LW: mem_read = `WORD;
            `OP_LH: mem_read = `HALFWORD;
            `OP_LB: mem_read = `BYTE;
            default: mem_read = `MEM_NONE;
        endcase
    endfunction

    function [1:0] mem_write;
        input [5:0] op;
        case (op)
            `OP_SW: mem_write = `WORD;
            `OP_SH: mem_write = `HALFWORD;
            `OP_SB: mem_write = `BYTE;
            default: mem_write = `MEM_NONE;
        endcase
    endfunction

    function reg_write;
        input [5:0] op;
        case (op)
            `OP_R, `OP_ADDI, `OP_LUI, `OP_ANDI, `OP_ORI, `OP_XORI, `OP_LW, `OP_LH, `OP_LB, `OP_JAL:
              reg_write = 1;
            default: reg_write = 0;
        endcase
    endfunction

    function mem_to_reg;
        input [5:0] op;
        case (op)
            `OP_LW, `OP_LH, `OP_LB: mem_to_reg = 1;
            default: mem_to_reg = 0;
        endcase
    endfunction

endmodule

module hazard_unit(input [1:0] EX_ctrl_mem_read_i,
                   input [4:0] EX_rt_i,
                   input [4:0] ID_rs_i,
                   input [4:0] ID_rt_i,
                   output stall_o);

    assign stall_o = EX_ctrl_mem_read_i != `MEM_NONE
                     && (EX_rt_i == ID_rs_i || EX_rt_i == ID_rt_i);

endmodule
```

EX.v

```verilog
‘include "header.v"

module EX(input clk_i,
          input n_rst_i,
          input [31:0] IDEX_pc_i,
          input [31:0] IDEX_ir_i,
          input [31:0] IDEX_a_i,
          input [31:0] IDEX_b_i,
          input IDEX_ctrl_reg_dst_i,
          input IDEX_ctrl_alu_src_i,
          input IDEX_ctrl_branch_i,
          input [1:0] IDEX_ctrl_mem_read_i,
          input [1:0] IDEX_ctrl_mem_write_i,
          input IDEX_ctrl_reg_write_i,
          input IDEX_ctrl_mem_to_reg_i,
          input MEM_do_branch_i,
          input [4:0] WB_reg_write_address_i,
          input [31:0] WB_reg_write_data_i,
          input WB_ctrl_reg_write_i,
          output reg [31:0] EXMEM_pc_branched_o,
          output reg [31:0] EXMEM_alu_o,
          output reg EXMEM_alu_do_branch_o,
          output reg [31:0] EXMEM_b_o,
          output reg [4:0] EXMEM_reg_write_address_o,
          output reg EXMEM_ctrl_branch_o,
          output reg [1:0] EXMEM_ctrl_mem_read_o,
          output reg [1:0] EXMEM_ctrl_mem_write_o,
          output reg EXMEM_ctrl_reg_write_o,
          output reg EXMEM_ctrl_mem_to_reg_o);

    wire [5:0] _op;
    wire [4:0] _rs;
    wire [4:0] _rt;
    wire [4:0] _rd;
    wire [4:0] _shift;
    wire [4:0] _aux;
    wire [31:0] _imm_dpl;
    wire [25:0] _addr;

    assign _op = IDEX_ir_i[31:26];
    assign _rs = IDEX_ir_i[25:21];
    assign _rt = IDEX_ir_i[20:16];
    assign _rd = IDEX_ir_i[15:11];
    assign _shift = IDEX_ir_i[10:6];
    assign _aux = IDEX_ir_i[4:0];
    assign _imm_dpl = {{16{IDEX_ir_i[15]}}, IDEX_ir_i[15:0]};
    assign _addr = IDEX_ir_i[25:0];

    wire [31:0] _a;
    wire [31:0] _b;
    wire [31:0] _alu_b;
    wire [31:0] _alu;
    wire _alu_zero;
    wire _alu_sign;
    wire _alu_do_branch;

    assign _a = forward_mux(IDEX_a_i,
                            EXMEM_alu_o,
                            WB_reg_write_data_i,
                            _forward_a);
    assign _b = forward_mux(IDEX_b_i,
                            EXMEM_alu_o,
                            WB_reg_write_data_i,
                            _forward_b);
    assign _alu_b = alu_b(_b,
                          _imm_dpl,
                          IDEX_ctrl_alu_src_i);

    assign _alu = (_op == ‘OP_LUI) ? (_imm_dpl << 16) :
                  (_op == ‘OP_JAL) ? IDEX_pc_i :
                  alu(alu_ctrl(_op, _aux),
                      _a,
                      _alu_b,
                      _shift);
    assign _alu_zero = ~|_alu;
    assign _alu_sign = _alu[31];
    assign _alu_do_branch = alu_do_branch(_op,
                                          _alu_zero,
                                          _alu_sign);

    wire [31:0] _pc_branched;
    assign _pc_branched = pc_branched(_op,
                                      IDEX_pc_i,
                                      _imm_dpl,
                                      _addr,
                                      _a);

    wire [4:0] _reg_write_address;
    assign _reg_write_address = (_op == ‘OP_JAL) ? 5’d31 :
                                (IDEX_ctrl_reg_dst_i) ? _rd : _rt;

    wire [1:0] _forward_a;
    wire [1:0] _forward_b;
    forwarding_unit forwarding_unit(.rs_i(_rs),
                                    .rt_i(_rt),
                                    .MEM_reg_write_address_i(EXMEM_reg_write_address_o),
                                    .MEM_ctrl_reg_write_i(EXMEM_ctrl_reg_write_o),
                                    .WB_reg_write_address_i(WB_reg_write_address_i),
                                    .WB_ctrl_reg_write_i(WB_ctrl_reg_write_i),
                                    .forward_a_o(_forward_a),
                                    .forward_b_o(_forward_b));

    always @(negedge n_rst_i or posedge clk_i) begin
        if (~n_rst_i) begin
            EXMEM_pc_branched_o <= 0;
            EXMEM_alu_o <= 0;
            EXMEM_alu_do_branch_o <= 0;
            EXMEM_b_o <= 0;
            EXMEM_reg_write_address_o <= 0;
            EXMEM_ctrl_branch_o <= 0;
            EXMEM_ctrl_mem_read_o <= 0;
            EXMEM_ctrl_mem_write_o <= 0;
            EXMEM_ctrl_reg_write_o <= 0;
            EXMEM_ctrl_mem_to_reg_o <= 0;
        end else if (clk_i) begin
            EXMEM_pc_branched_o <= _pc_branched;
            EXMEM_alu_o <= _alu;
            EXMEM_alu_do_branch_o <= _alu_do_branch;
            EXMEM_b_o <= _b;
            EXMEM_reg_write_address_o <= _reg_write_address;
            if (MEM_do_branch_i) begin
                EXMEM_ctrl_branch_o <= 0;
                EXMEM_ctrl_mem_read_o <= 0;
                EXMEM_ctrl_mem_write_o <= 0;
                EXMEM_ctrl_reg_write_o <= 0;
                EXMEM_ctrl_mem_to_reg_o <= 0;
            end else begin
                EXMEM_ctrl_branch_o <= IDEX_ctrl_branch_i;
                EXMEM_ctrl_mem_read_o <= IDEX_ctrl_mem_read_i;
                EXMEM_ctrl_mem_write_o <= IDEX_ctrl_mem_write_i;
                EXMEM_ctrl_reg_write_o <= IDEX_ctrl_reg_write_i;
                EXMEM_ctrl_mem_to_reg_o <= IDEX_ctrl_mem_to_reg_i;
            end
        end
    end

    function [31:0] pc_branched;
        input [5:0] op;
        input [31:0] pc;
        input [31:0] imm_dpl;
        input [25:0] addr;
        input [31:0] a;
        case (op)
            ‘OP_BEQ, ‘OP_BNE, ‘OP_BLT, ‘OP_BLE: pc_branched = pc + (imm_dpl << 2);
            ‘OP_J, ‘OP_JAL: pc_branched = {4’b0, addr << 2};
            ‘OP_JR: pc_branched = a;
            default: pc_branched = pc;
        endcase
    endfunction

    function [4:0] alu_ctrl;
        input [5:0] op;
        input [4:0] aux;
        case (op)
            ‘OP_R:    alu_ctrl = aux;
            ‘OP_ADDI: alu_ctrl = ‘ALU_ADD;
            ‘OP_ANDI: alu_ctrl = ‘ALU_AND;
            ‘OP_ORI:  alu_ctrl = ‘ALU_OR;
            ‘OP_XORI: alu_ctrl = ‘ALU_XOR;
            ‘OP_BEQ, ‘OP_BNE, ‘OP_BLT, ‘OP_BLE: alu_ctrl = ‘ALU_SUB;
            ‘OP_LW, ‘OP_LH, ‘OP_LB, ‘OP_SW, ‘OP_SH, ‘OP_SB: alu_ctrl = ‘ALU_ADD;
            default:  alu_ctrl = 5’h1f;
        endcase
    endfunction

    function [31:0] forward_mux;
        input [31:0] EX_data;
        input [31:0] MEM_reg_write_data;
        input [31:0] WB_reg_write_data;
        input [1:0] forward;
        case (forward)
            ‘FORWARD_MEM: forward_mux = MEM_reg_write_data;
            ‘FORWARD_WB:  forward_mux = WB_reg_write_data;
            default:      forward_mux = EX_data;
        endcase
    endfunction

    function [31:0] alu_b;
        input [31:0] IDEX_b;
        input [31:0] imm_dpl;
        input ctrl_alu_src;
        if (ctrl_alu_src) begin
            alu_b = imm_dpl;
        end else begin
            alu_b = IDEX_b;
        end
    endfunction

    function [31:0] alu;
        input [4:0] ctrl;
        input [31:0] a;
        input [31:0] b;
        input [4:0] shift;
        case (ctrl)
            ‘ALU_ADD: alu = a + b;
            ‘ALU_SUB: alu = a - b;
            ‘ALU_AND: alu = a & b;
            ‘ALU_OR:  alu = a | b;
            ‘ALU_XOR: alu = a ^ b;
            ‘ALU_NOR: alu = ~(a | b);
            ‘ALU_SLL: alu = a << shift;
            ‘ALU_SRL: alu = a >> shift;
            ‘ALU_SRA: alu = {{32{a[31]}}, a} >> shift;
            default:  alu = 32’hffffffff;
        endcase
    endfunction

    function alu_do_branch;
        input [5:0] op;
        input alu_zero;
        input alu_sign;
        case (op)
            ‘OP_BEQ: alu_do_branch = alu_zero;
            ‘OP_BNE: alu_do_branch = ~alu_zero;
            ‘OP_BLT: alu_do_branch = alu_sign;
            ‘OP_BLE: alu_do_branch = alu_zero | alu_sign;
            ‘OP_J, ‘OP_JAL, ‘OP_JR: alu_do_branch = 1;
            default: alu_do_branch = 0;
        endcase
    endfunction

endmodule

module forwarding_unit(input [4:0] rs_i,
                       input [4:0] rt_i,
                       input [4:0] MEM_reg_write_address_i,
                       input MEM_ctrl_reg_write_i,
                       input [4:0] WB_reg_write_address_i,
                       input WB_ctrl_reg_write_i,
                       output [1:0] forward_a_o,
                       output [1:0] forward_b_o);

    assign forward_a_o = forward_a(rs_i,
                                   MEM_reg_write_address_i,
                                   MEM_ctrl_reg_write_i,
                                   WB_reg_write_address_i,
                                   WB_ctrl_reg_write_i);

    assign forward_b_o = forward_b(rt_i,
                                   MEM_reg_write_address_i,
                                   MEM_ctrl_reg_write_i,
                                   WB_reg_write_address_i,
                                   WB_ctrl_reg_write_i);

    function [1:0] forward_a;
        input [4:0] rs;
        input [4:0] MEM_reg_write_address;
        input MEM_ctrl_reg_write;
        input [4:0] WB_reg_write_address;
        input WB_ctrl_reg_write;
        if (MEM_ctrl_reg_write
            && MEM_reg_write_address != 0
            && MEM_reg_write_address == rs) begin
            forward_a = ‘FORWARD_MEM;
        end else if (WB_ctrl_reg_write
                     && WB_reg_write_address != 0
                     && WB_reg_write_address == rs) begin
            forward_a = `FORWARD_WB;
        end else begin
            forward_a = `FORWARD_NONE;
        end
    endfunction

    function [1:0] forward_b;
        input [4:0] rt;
        input [4:0] MEM_reg_write_address;
        input MEM_ctrl_reg_write;
        input [4:0] WB_reg_write_address;
        input WB_ctrl_reg_write;
        if (MEM_ctrl_reg_write
            && MEM_reg_write_address != 0
            && MEM_reg_write_address == rt) begin
            forward_b = `FORWARD_MEM;
        end else if (WB_ctrl_reg_write
                     && WB_reg_write_address != 0
                     && WB_reg_write_address == rt) begin
            forward_b = `FORWARD_WB;
        end else begin
            forward_b = `FORWARD_NONE;
        end
    endfunction

endmodule
```

```verilog
‘include "header.v"

module MEM(input clk_i,
           input n_rst_i,
           input [31:0] EXMEM_pc_branched_i,
           input [31:0] EXMEM_alu_i,
           input EXMEM_alu_do_branch_i,
           input [31:0] EXMEM_b_i,
           input [4:0] EXMEM_reg_write_address_i,
           input EXMEM_ctrl_branch_i,
           input [1:0] EXMEM_ctrl_mem_read_i,
           input [1:0] EXMEM_ctrl_mem_write_i,
           input EXMEM_ctrl_reg_write_i,
           input EXMEM_ctrl_mem_to_reg_i,
           output reg [31:0] MEMWB_mem_o,
           output reg [31:0] MEMWB_alu_o,
           output reg [4:0] MEMWB_reg_write_address_o,
           output reg MEMWB_ctrl_reg_write_o,
           output reg MEMWB_ctrl_mem_to_reg_o,
           output [31:0] MEM_pc_branched_o,
           output MEM_do_branch_o);

    assign MEM_pc_branched_o = EXMEM_pc_branched_i;
    assign MEM_do_branch_o = (EXMEM_ctrl_branch_i & EXMEM_alu_do_branch_i);

    wire [31:0] _mem_read_data;

    memory memory(.clk_i(clk_i),
                  .address_i(EXMEM_alu_i[7:0]),
                  .write_data_i(EXMEM_b_i),
                  .ctrl_mem_read_i(EXMEM_ctrl_mem_read_i),
                  .ctrl_mem_write_i(EXMEM_ctrl_mem_write_i),
                  .read_data_o(_mem_read_data));

    always @(negedge n_rst_i or posedge clk_i) begin
        if (~n_rst_i) begin
            MEMWB_mem_o <= 0;
            MEMWB_alu_o <= 0;
            MEMWB_reg_write_address_o <= 0;
            MEMWB_ctrl_reg_write_o <= 0;
            MEMWB_ctrl_mem_to_reg_o <= 0;
        end else if (clk_i) begin
            MEMWB_mem_o <= _mem_read_data;
            MEMWB_alu_o <= EXMEM_alu_i;
            MEMWB_reg_write_address_o <= EXMEM_reg_write_address_i;
            MEMWB_ctrl_reg_write_o <= EXMEM_ctrl_reg_write_i;
            MEMWB_ctrl_mem_to_reg_o <= EXMEM_ctrl_mem_to_reg_i;
        end
    end

endmodule

module memory(input clk_i,
              input [7:0] address_i,
              input [31:0] write_data_i,
              input [1:0] ctrl_mem_read_i,
              input [1:0] ctrl_mem_write_i,
              output [31:0] read_data_o);

    reg [31:0] _mem[0:255];

    assign read_data_o = read(_mem[address_i], ctrl_mem_read_i);

    always @(negedge clk_i) begin
        case (ctrl_mem_write_i)
            ‘WORD: _mem[address_i] <= write_data_i;
            ‘HALFWORD: _mem[address_i][15:0] <= write_data_i[15:0];
            ‘BYTE: _mem[address_i][7:0] <= write_data_i[7:0];
        endcase
    end

    function [31:0] read;
        input [31:0] data;
        input [1:0] ctrl;
        case (ctrl)
            ‘WORD: read = data;
            ‘HALFWORD: read = {{16{data[15]}}, data[15:0]};
            ‘BYTE: read = {{24{data[7]}}, data[7:0]};
            default: read = 0;
        endcase
    endfunction

endmodule
```

```verilog
module WB(input clk_i,
          input n_rst_i,
          input [31:0] MEMWB_mem_i,
          input [31:0] MEMWB_alu_i,
          input [4:0] MEMWB_reg_write_address_i,
          input MEMWB_ctrl_reg_write_i,
          input MEMWB_ctrl_mem_to_reg_i,
          output [4:0] WB_reg_write_address_o,
          output [31:0] WB_reg_write_data_o,
          output WB_ctrl_reg_write_o);

    assign WB_reg_write_address_o = MEMWB_reg_write_address_i;
    assign WB_reg_write_data_o = (MEMWB_ctrl_mem_to_reg_i) ? MEMWB_mem_i : MEMWB_alu_i;
    assign WB_ctrl_reg_write_o = MEMWB_ctrl_reg_write_i;

endmodule
```

```verilog
module processor;
    reg clk;
    reg n_rst;

    initial begin
        clk <= 0;
        n_rst <= 1;
        #10 n_rst <= 0;
        #10 n_rst <= 1;
        #10000 $stop;
    end

    always #50 clk = ~clk;

    // IF
    wire [31:0] IFID_pc;
    wire [31:0] IFID_ir;

    // ID
    wire [31:0] IDEX_pc;
    wire [31:0] IDEX_ir;
    wire [31:0] IDEX_a;
    wire [31:0] IDEX_b;
    wire IDEX_ctrl_reg_dst;
    wire IDEX_ctrl_alu_src;
    wire IDEX_ctrl_branch;
    wire [1:0] IDEX_ctrl_mem_read;
    wire [1:0] IDEX_ctrl_mem_write;
    wire IDEX_ctrl_reg_write;
    wire IDEX_ctrl_mem_to_reg;
    wire ID_stall;

    // EX
    wire [31:0] EXMEM_pc_branched;
    wire [31:0] EXMEM_alu;
    wire EXMEM_alu_do_branch;
    wire [31:0] EXMEM_b;
    wire [4:0] EXMEM_reg_write_address;
    wire EXMEM_ctrl_branch;
    wire [1:0] EXMEM_ctrl_mem_read;
    wire [1:0] EXMEM_ctrl_mem_write;
    wire EXMEM_ctrl_reg_write;
    wire EXMEM_ctrl_mem_to_reg;

    // MEM
    wire [31:0] MEMWB_mem;
    wire [31:0] MEMWB_alu;
    wire [4:0] MEMWB_reg_write_address;
    wire MEMWB_ctrl_reg_write;
    wire MEMWB_ctrl_mem_to_reg;
    wire [31:0] MEM_pc_branched;
    wire MEM_do_branch;

    // WB
    wire [4:0] WB_reg_write_address;
    wire [31:0] WB_reg_write_data;
    wire WB_ctrl_reg_write;

    IF IF(.clk_i(clk),
          .n_rst_i(n_rst),
          .ID_stall_i(ID_stall),
          .MEM_pc_branched_i(MEM_pc_branched),
          .MEM_do_branch_i(MEM_do_branch),
          .IFID_pc_o(IFID_pc),
          .IFID_ir_o(IFID_ir));

    ID ID(.clk_i(clk),
          .n_rst_i(n_rst),
          .IFID_pc_i(IFID_pc),
          .IFID_ir_i(IFID_ir),
          .MEM_do_branch_i(MEM_do_branch),
          .WB_reg_write_address_i(WB_reg_write_address),
          .WB_reg_write_data_i(WB_reg_write_data),
          .WB_ctrl_reg_write_i(WB_ctrl_reg_write),
          .IDEX_pc_o(IDEX_pc),
          .IDEX_ir_o(IDEX_ir),
          .IDEX_a_o(IDEX_a),
          .IDEX_b_o(IDEX_b),
          .IDEX_ctrl_reg_dst_o(IDEX_ctrl_reg_dst),
          .IDEX_ctrl_alu_src_o(IDEX_ctrl_alu_src),
          .IDEX_ctrl_branch_o(IDEX_ctrl_branch),
          .IDEX_ctrl_mem_read_o(IDEX_ctrl_mem_read),
          .IDEX_ctrl_mem_write_o(IDEX_ctrl_mem_write),
          .IDEX_ctrl_reg_write_o(IDEX_ctrl_reg_write),
          .IDEX_ctrl_mem_to_reg_o(IDEX_ctrl_mem_to_reg),
          .ID_stall_o(ID_stall));

    EX EX(.clk_i(clk),
          .n_rst_i(n_rst),
          .IDEX_pc_i(IDEX_pc),
          .IDEX_ir_i(IDEX_ir),
          .IDEX_a_i(IDEX_a),
          .IDEX_b_i(IDEX_b),
          .IDEX_ctrl_reg_dst_i(IDEX_ctrl_reg_dst),
          .IDEX_ctrl_alu_src_i(IDEX_ctrl_alu_src),
          .IDEX_ctrl_branch_i(IDEX_ctrl_branch),
          .IDEX_ctrl_mem_read_i(IDEX_ctrl_mem_read),
          .IDEX_ctrl_mem_write_i(IDEX_ctrl_mem_write),
          .IDEX_ctrl_reg_write_i(IDEX_ctrl_reg_write),
          .IDEX_ctrl_mem_to_reg_i(IDEX_ctrl_mem_to_reg),
          .MEM_do_branch_i(MEM_do_branch),
          .WB_reg_write_address_i(WB_reg_write_address),
          .WB_reg_write_data_i(WB_reg_write_data),
          .WB_ctrl_reg_write_i(WB_ctrl_reg_write),
          .EXMEM_pc_branched_o(EXMEM_pc_branched),
          .EXMEM_alu_o(EXMEM_alu),
          .EXMEM_alu_do_branch_o(EXMEM_alu_do_branch),
          .EXMEM_b_o(EXMEM_b),
          .EXMEM_reg_write_address_o(EXMEM_reg_write_address),
          .EXMEM_ctrl_branch_o(EXMEM_ctrl_branch),
          .EXMEM_ctrl_mem_read_o(EXMEM_ctrl_mem_read),
          .EXMEM_ctrl_mem_write_o(EXMEM_ctrl_mem_write),
          .EXMEM_ctrl_reg_write_o(EXMEM_ctrl_reg_write),
          .EXMEM_ctrl_mem_to_reg_o(EXMEM_ctrl_mem_to_reg));

    MEM MEM(.clk_i(clk),
            .n_rst_i(n_rst),
            .EXMEM_pc_branched_i(EXMEM_pc_branched),
            .EXMEM_alu_i(EXMEM_alu),
            .EXMEM_alu_do_branch_i(EXMEM_alu_do_branch),
            .EXMEM_b_i(EXMEM_b),
            .EXMEM_reg_write_address_i(EXMEM_reg_write_address),
            .EXMEM_ctrl_branch_i(EXMEM_ctrl_branch),
            .EXMEM_ctrl_mem_read_i(EXMEM_ctrl_mem_read),
            .EXMEM_ctrl_mem_write_i(EXMEM_ctrl_mem_write),
            .EXMEM_ctrl_reg_write_i(EXMEM_ctrl_reg_write),
            .EXMEM_ctrl_mem_to_reg_i(EXMEM_ctrl_mem_to_reg),
            .MEMWB_mem_o(MEMWB_mem),
            .MEMWB_alu_o(MEMWB_alu),
            .MEMWB_reg_write_address_o(MEMWB_reg_write_address),
            .MEMWB_ctrl_reg_write_o(MEMWB_ctrl_reg_write),
            .MEMWB_ctrl_mem_to_reg_o(MEMWB_ctrl_mem_to_reg),
            .MEM_pc_branched_o(MEM_pc_branched),
            .MEM_do_branch_o(MEM_do_branch));

    WB WB(.clk_i(clk),
          .n_rst_i(n_rst),
          .MEMWB_mem_i(MEMWB_mem),
          .MEMWB_alu_i(MEMWB_alu),
          .MEMWB_reg_write_address_i(MEMWB_reg_write_address),
          .MEMWB_ctrl_reg_write_i(MEMWB_ctrl_reg_write),
          .MEMWB_ctrl_mem_to_reg_i(MEMWB_ctrl_mem_to_reg),
          .WB_reg_write_address_o(WB_reg_write_address),
          .WB_reg_write_data_o(WB_reg_write_data),
          .WB_ctrl_reg_write_o(WB_ctrl_reg_write));

endmodule
```