Lab 5

PB22030892 刘铠瑜

任务 1：写优先的寄存器堆

1. 写优先寄存器

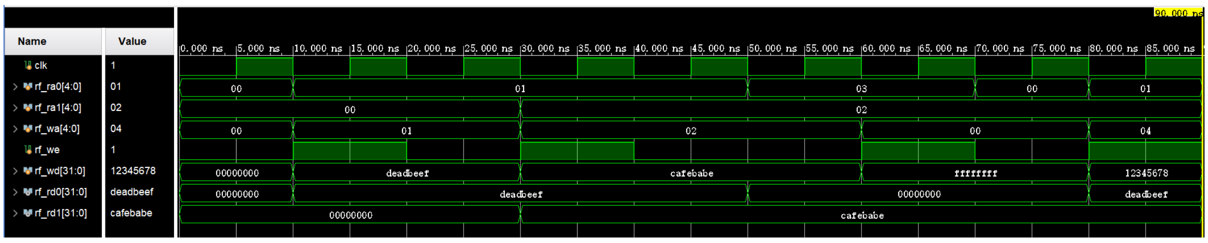
    1. **REGFILE.v**

```
module REGFILE (
    input                   [ 0 : 0]        clk,

    input                   [ 4 : 0]        rf_ra0,     // 读寄存器地址
    input                   [ 4 : 0]        rf_ra1,
    input                   [ 4 : 0]        rf_wa,      // 写寄存器地址
    input                   [ 0 : 0]        rf_we,      // 写使能信号
    input                   [31 : 0]        rf_wd,      // 写数据

    output                  [31 : 0]        rf_rd0,     // 读数据
    output                  [31 : 0]        rf_rd1
);

    reg [31 : 0] reg_file [0 : 31];

    // 用于初始化寄存器
    integer i;
    initial begin
        for (i = 0; i < 32; i = i + 1)
            reg_file[i] = 0;
    end

    always @(posedge clk) begin
        if (rf_we && (rf_wa != 0)) begin
            reg_file[rf_wa] <= rf_wd;
        end
    end

    assign rf_rd0 = ((rf_ra0 == rf_wa) && (rf_wa != 0) && rf_we) ? rf_wd :
reg_file[rf_ra0];
    assign rf_rd1 = ((rf_ra1 == rf_wa) && (rf_wa != 0) && rf_we) ? rf_wd :
reg_file[rf_ra1];
endmodule
```

只有在写入有效`(rf_wa != 0) && rf_we`且读地址和写地址一致`rf_ra0 == rf_wa`时，直接读写数据。其他情况正常读寄存器的值。 2. **REGFILE_tb.v**

```
`timescale 1ns / 1ps

module REGFILE_tb;

    // Inputs
    reg clk;
    reg [4:0] rf_ra0;
    reg [4:0] rf_ra1;
    reg [4:0] rf_wa;
    reg rf_we;
    reg [31:0] rf_wd;

    // Outputs
    wire [31:0] rf_rd0;
    wire [31:0] rf_rd1;

    REGFILE uut (
        .clk(clk),
        .rf_ra0(rf_ra0),
        .rf_ra1(rf_ra1),
        .rf_wa(rf_wa),
        .rf_we(rf_we),
        .rf_wd(rf_wd),
        .rf_rd0(rf_rd0),
        .rf_rd1(rf_rd1)
    );

    // Clock generation
    initial begin
        clk = 0;
        forever #5 clk = ~clk; // 10ns clock period
    end

    initial begin
        // Initialize inputs
        rf_ra0 = 0;
        rf_ra1 = 0;
        rf_wa = 0;
        rf_we = 0;
        rf_wd = 0;

        // Wait for global reset
        #10;

        // Test 1: Write to register 1 and verify read (write priority)
        rf_wa = 5'd1;     // Write address
        rf_wd = 32'hDEADBEEF; // Write data
        rf_we = 1;        // Enable write
        rf_ra0 = 5'd1;    // Read address 0 (same as write address)
        #5;               // Wait for half clock period to test combinational
logic
        $display("Test 1: rf_rd0 = %h (expected: DEADBEEF)", rf_rd0);
        #5;                 // Wait for clock edge
        rf_we = 0;          // Disable write
        #10;                // Wait for read

        // Test 2: Write to register 2 and verify read
        rf_wa = 5'd2;
        rf_wd = 32'hCAFEBABE;
        rf_we = 1;
        rf_ra1 = 5'd2;      // Read address 1 (same as write address)
        #5;                 // Wait for half clock period to test combinational
logic

        #5;                 // Wait for clock edge
        rf_we = 0;
        #10;

        // Test 3: Read from an uninitialized register
        rf_ra0 = 5'd3;      // Read address 0
        #10;

        // Test 4: Write to register 0 (should be ignored)
        rf_wa = 5'd0;
        rf_wd = 32'hFFFFFFFF;
        rf_we = 1;
        #10;
        rf_we = 0;
        rf_ra0 = 5'd0;      // Read address 0
        #10;

        // Test 5: Simultaneous read and write to different registers
        rf_wa = 5'd4;
        rf_wd = 32'h12345678;
        rf_we = 1;
        rf_ra0 = 5'd1;      // Read address 0 (different from write address)
        #10;

        $stop;
    end

endmodule
```

3. **波形图**



2. add x4, x1, x2指令从寄存器中取值时，addi x1, x0, 10已经完成写入操作，所以 x4 = 1 + 2 + 10 = 13。 add x5, x2, x3从寄存器中取值时，addi x2, x0, 20已经完成写入操作，所以 x5 = 2 + 3 + 20 = 25。

任务 2：无冒险流水线

PIPELINE_REG.v

```
module PIPELINE_REG (
    input                          clk,
    input                          rst,
    input                          en,
    input                          flush,
    input                          stall,

    // input
    input             [ 0:0]       commit_in,
    input             [31:0]       pc_in,
    input             [31:0]       pc_add4_in,
    input             [31:0]       inst_in,
    input             [ 4:0]       alu_op_in,
    input             [31:0]       imm_in,
    input             [ 4:0]       rf_wa_in,
    input             [ 0:0]       rf_we_in,
    input             [ 0:0]       alu_src0_sel_in,
    input             [ 0:0]       alu_src1_sel_in,
    input             [ 3:0]       dmem_access_in,
    input             [ 0:0]       dmem_we_in,
    input             [ 1:0]       rf_wd_sel_in,
    input             [ 3:0]       br_type_in,
    input             [31:0]       rf_rd0_in,
    input             [31:0]       rf_rd1_in,
    input             [31:0]       alu_res_in,
    input             [31:0]       dmem_rd_out_in,
    input             [31:0]       dmem_wd_out_in,

    // output
    output reg        [ 0:0]       commit_out,
    output reg        [31:0]       pc_out,
    output reg        [31:0]       pc_add4_out,
    output reg        [31:0]       inst_out,
    output reg        [ 4:0]       alu_op_out,
    output reg        [31:0]       imm_out,
    output reg        [ 4:0]       rf_wa_out,
    output reg        [ 0:0]       rf_we_out,
    output reg        [ 0:0]       alu_src0_sel_out,
    output reg        [ 0:0]       alu_src1_sel_out,
    output reg        [ 3:0]       dmem_access_out,
    output reg        [ 0:0]       dmem_we_out,
    output reg        [ 1:0]       rf_wd_sel_out,
    output reg        [ 3:0]       br_type_out,
    output reg        [31:0]       rf_rd0_out,
    output reg        [31:0]       rf_rd1_out,
    output reg        [31:0]       alu_res_out,
    output reg        [31:0]       dmem_rd_out_out,
    output reg        [31:0]       dmem_wd_out_out
);

    always @(posedge clk) begin
        if (rst) begin
            commit_out        <= 1'b0;
            pc_out            <= 32'h0040_0000;
            pc_add4_out       <= 32'h0040_0004;
            inst_out          <= 32'h0000_0013;  // NOP instruction: addi x0, x0, 0
            alu_op_out        <= 5'b0;
            imm_out           <= 32'b0;
            rf_wa_out         <= 5'b0;
            rf_we_out         <= 1'b0;
            alu_src0_sel_out <= 1'b0;           // use rs1
            alu_src1_sel_out <= 1'b1;           // use imm
            dmem_access_out   <= 4'b0;
            dmem_we_out       <= 1'b0;
            rf_wd_sel_out     <= 2'b0;
            br_type_out       <= 4'b0;
            rf_rd0_out        <= 32'b0;
            rf_rd1_out        <= 32'b0;
            alu_res_out       <= 32'b0;
            dmem_rd_out_out   <= 32'b0;
            dmem_wd_out_out   <= 32'b0;

        end else if (en) begin
            if (flush) begin    // flush 有效时，寄存器被清空
                commit_out        <= 1'b0;
                pc_out            <= 32'h0040_0000;
                pc_add4_out       <= 32'h0040_0004;
                inst_out          <= 32'h0000_0013;  // NOP instruction: addi x0,
x0, 0
                alu_op_out        <= 5'b0;
                imm_out           <= 32'b0;
                rf_wa_out         <= 5'b0;
                rf_we_out         <= 1'b0;
                alu_src0_sel_out <= 1'b0;           // use rs1
                alu_src1_sel_out <= 1'b1;           // use imm
                dmem_access_out   <= 4'b0;
                dmem_we_out       <= 1'b0;
                rf_wd_sel_out     <= 2'b0;
                br_type_out       <= 4'b0;
                rf_rd0_out        <= 32'b0;
                rf_rd1_out        <= 32'b0;
                alu_res_out       <= 32'b0;
                dmem_rd_out_out   <= 32'b0;
                dmem_wd_out_out   <= 32'b0;

            end else if (!stall) begin  // stall 有效时，输出仍保持之前的值不变，而非
接收输入
                commit_out        <= commit_in;
                pc_out            <= pc_in;
                pc_add4_out       <= pc_add4_in;
                inst_out          <= inst_in;
                alu_op_out        <= alu_op_in;
                imm_out          <= imm_in;
                rf_wa_out        <= rf_wa_in;
                rf_we_out        <= rf_we_in;
                alu_src0_sel_out <= alu_src0_sel_in;
                alu_src1_sel_out <= alu_src1_sel_in;
                dmem_access_out  <= dmem_access_in;
                dmem_we_out      <= dmem_we_in;
                rf_wd_sel_out    <= rf_wd_sel_in;
                br_type_out      <= br_type_in;
                rf_rd0_out       <= rf_rd0_in;
                rf_rd1_out       <= rf_rd1_in;
                alu_res_out      <= alu_res_in;
                dmem_rd_out_out  <= dmem_rd_out_in;
                dmem_wd_out_out  <= dmem_wd_out_in;
            end
        end
    end
endmodule
```

CPU.v

```
module CPU (
    input                   [ 0 : 0]            clk,
    input                   [ 0 : 0]            rst,

    input                   [ 0 : 0]            global_en,

/* ------------------------------ Memory (inst) ----------------------------- */
    output                  [31 : 0]            imem_raddr,
    input                   [31 : 0]            imem_rdata,

/* ------------------------------ Memory (data) ----------------------------- */
    input                   [31 : 0]            dmem_rdata,
    output                  [ 0 : 0]            dmem_we,
    output                  [31 : 0]            dmem_addr,
    output                  [31 : 0]            dmem_wdata,

/* ---------------------------------- Debug --------------------------------- */
    output                  [ 0 : 0]            commit,
    output                  [31 : 0]            commit_pc,
    output                  [31 : 0]            commit_inst,
    output                  [ 0 : 0]            commit_halt,
    output                  [ 0 : 0]            commit_reg_we,
    output                  [ 4 : 0]            commit_reg_wa,
    output                  [31 : 0]            commit_reg_wd,
    output                  [ 0 : 0]            commit_dmem_we,
    output                  [31 : 0]            commit_dmem_wa,
    output                  [31 : 0]            commit_dmem_wd,

    input                   [ 4 : 0]            debug_reg_ra,
    output                  [31 : 0]            debug_reg_rd
);
```

```
    `define HALT_INST 32'H00100073

    // Commit
    reg  [ 0 : 0]   commit_reg          ;
    reg  [31 : 0]   commit_pc_reg       ;
    reg  [31 : 0]   commit_inst_reg     ;
    reg  [ 0 : 0]   commit_halt_reg     ;
    reg  [ 0 : 0]   commit_reg_we_reg   ;
    reg  [ 4 : 0]   commit_reg_wa_reg   ;
    reg  [31 : 0]   commit_reg_wd_reg   ;
    reg  [ 0 : 0]   commit_dmem_we_reg  ;
    reg  [31 : 0]   commit_dmem_wa_reg  ;
    reg  [31 : 0]   commit_dmem_wd_reg  ;

    // Commit
    assign commit_if = 1'H1;
    // 这个信号需要经过 IF/ID、ID/EX、EX/MEM、MEM/WB 段间寄存器，最终连接到 commit_reg
上

    always @(posedge clk) begin
        if (rst) begin
            commit_reg          <= 1'H0;
            commit_pc_reg       <= 32'H0;
            commit_inst_reg     <= 32'H0;
            commit_halt_reg     <= 1'H0;
            commit_reg_we_reg   <= 1'H0;
            commit_reg_wa_reg   <= 5'H0;
            commit_reg_wd_reg   <= 32'H0;
            commit_dmem_we_reg  <= 1'H0;
            commit_dmem_wa_reg  <= 32'H0;
            commit_dmem_wd_reg  <= 32'H0;
        end
        else if (global_en) begin
            // 这里右侧的信号都是 MEM/WB 段间寄存器的输出
            commit_reg          <= commit_wb;
            commit_pc_reg       <= pc_wb;
            commit_inst_reg     <= inst_wb;
            commit_halt_reg     <= inst_wb == `HALT_INST;
            commit_reg_we_reg   <= rf_we_wb;
            commit_reg_wa_reg   <= rf_wa_wb;
            commit_reg_wd_reg   <= rf_wd_wb;
            commit_dmem_we_reg  <= dmem_we_wb;
            commit_dmem_wa_reg  <= dmem_addr_wb;
            commit_dmem_wd_reg  <= dmem_wdata_wb;
        end
    end

    assign commit               = commit_reg;
    assign commit_pc            = commit_pc_reg;
    assign commit_inst          = commit_inst_reg;
    assign commit_halt          = commit_halt_reg;
    assign commit_reg_we        = commit_reg_we_reg;
    assign commit_reg_wa        = commit_reg_wa_reg;
```

```verilog
    assign commit_reg_wd        = commit_reg_wd_reg;
    assign commit_dmem_we       = commit_dmem_we_reg;
    assign commit_dmem_wa       = commit_dmem_wa_reg;
    assign commit_dmem_wd       = commit_dmem_wd_reg;

/* ------------------------------ IF Stage ------------------------------ */
    wire [ 0 : 0]   commit_if;
    wire [31 : 0]   pc_if;
    wire [31 : 0]   pc_add4_if;
    wire [31 : 0]   inst_if;

    PC pc (
        .clk        (clk),
        .rst        (rst),
        .en         (global_en),
        .npc        (npc_ex),
        .pc         (pc_if)
    );

    ADD_4 add4 (
        .in         (pc_if),
        .out        (pc_add4_if)
    );

    assign imem_raddr = pc_if;
    assign inst_if    = imem_rdata;

/* ------------------------------ IF/ID  ------------------------------ */
    wire [ 0 : 0]   commit_id;
    wire [31 : 0]   pc_id;
    wire [31 : 0]   pc_add4_id;
    wire [31 : 0]   inst_id;

    PIPELINE_REG if_id (
        .clk                (clk),
        .rst                (rst),
        .en                 (global_en),
        .flush              (commit_ex),
        .stall              (1'b0),

        // input
        .commit_in          (commit_if),
        .pc_in              (pc_if),
        .pc_add4_in         (pc_add4_if),
        .inst_in            (inst_if),
        .alu_op_in          (5'b0),
        .imm_in             (32'b0),
        .rf_wa_in           (5'b0),
        .rf_we_in           (1'b0),
        .alu_src0_sel_in    (1'b0),
        .alu_src1_sel_in    (1'b0),
        .dmem_access_in     (4'b0),
        .dmem_we_in         (1'b0),
        .rf_wd_sel_in       (2'b0),
        .br_type_in          (4'b0),
        .rf_rd0_in           (32'b0),
        .rf_rd1_in           (32'b0),
        .alu_res_in          (32'b0),
        .dmem_rd_out_in      (32'b0),
        .dmem_wd_out_in      (32'b0),

        // output
        .commit_out          (commit_id),
        .pc_out              (pc_id),
        .pc_add4_out         (pc_add4_id),
        .inst_out            (inst_id)
    );

/* ----------------------------- ID Stage ----------------------------- */

    wire [ 4 : 0]   alu_op_id;
    wire [31 : 0]   imm_id;
    wire [ 4 : 0]   rf_ra0_id;
    wire [ 4 : 0]   rf_ra1_id;
    wire [ 4 : 0]   rf_wa_id;
    wire [ 0 : 0]   rf_we_id;
    wire [ 0 : 0]   alu_src0_sel_id;
    wire [ 0 : 0]   alu_src1_sel_id;
    wire [ 3 : 0]   dmem_access_id;
    wire [ 0 : 0]   dmem_we_id;
    wire [ 1 : 0]   rf_wd_sel_id;
    wire [ 3 : 0]   br_type_id;
    wire [31 : 0]   rf_rd0_id;
    wire [31 : 0]   rf_rd1_id;

    DECODER decoder (
        .inst            (inst_id),
        .alu_op          (alu_op_id),
        .imm             (imm_id),
        .rf_ra0          (rf_ra0_id),
        .rf_ra1          (rf_ra1_id),
        .rf_wa           (rf_wa_id),
        .rf_we           (rf_we_id),
        .alu_src0_sel    (alu_src0_sel_id),
        .alu_src1_sel    (alu_src1_sel_id),
        .dmem_access     (dmem_access_id),
        .dmem_we         (dmem_we_id),
        .rf_wd_sel       (rf_wd_sel_id),
        .br_type         (br_type_id)
    );

    REGFILE regfile (
        .clk             (clk),
        .rf_ra0          (rf_ra0_id),
        .rf_ra1          (rf_ra1_id),
        .dbg_rf_ra       (debug_reg_ra),
        .rf_wa           (rf_wa_wb),
        .rf_we           (rf_we_wb),
        .rf_wd          (rf_wd_wb),

        .rf_rd0         (rf_rd0_id),
        .rf_rd1         (rf_rd1_id),
        .dbg_rf_rd      (debug_reg_rd)
    );

/* ------------------------------ ID/EX ------------------------------ */

    wire [ 0 : 0]   commit_ex;
    wire [31 : 0]   pc_ex;
    wire [31 : 0]   pc_add4_ex;
    wire [31 : 0]   inst_ex;
    wire [ 4 : 0]   alu_op_ex;
    wire [31 : 0]   imm_ex;
    wire [ 4 : 0]   rf_wa_ex;
    wire [ 0 : 0]   rf_we_ex;
    wire [ 0 : 0]   alu_src0_sel_ex;
    wire [ 0 : 0]   alu_src1_sel_ex;
    wire [ 3 : 0]   dmem_access_ex;
    wire [ 0 : 0]   dmem_we_ex;
    wire [ 1 : 0]   rf_wd_sel_ex;
    wire [ 3 : 0]   br_type_ex;
    wire [31 : 0]   rf_rd0_ex;
    wire [31 : 0]   rf_rd1_ex;

    PIPELINE_REG id_ex (
        .clk                (clk),
        .rst                (rst),
        .en                 (global_en),
        .flush              (commit_ex),
        .stall              (1'b0),

        // input
        .commit_in          (commit_id),
        .pc_in              (pc_id),
        .pc_add4_in         (pc_add4_id),
        .inst_in            (inst_id),
        .alu_op_in          (alu_op_id),
        .imm_in             (imm_id),
        .rf_wa_in           (rf_wa_id),
        .rf_we_in           (rf_we_id),
        .alu_src0_sel_in    (alu_src0_sel_id),
        .alu_src1_sel_in    (alu_src1_sel_id),
        .dmem_access_in     (dmem_access_id),
        .dmem_we_in         (dmem_we_id),
        .rf_wd_sel_in       (rf_wd_sel_id),
        .br_type_in         (br_type_id),
        .rf_rd0_in          (rf_rd0_id),
        .rf_rd1_in          (rf_rd1_id),
        .alu_res_in         (32'b0),
        .dmem_rd_out_in     (32'b0),
        .dmem_wd_out_in     (32'b0),
        // output
        .commit_out         (commit_ex),
        .pc_out             (pc_ex),
        .pc_add4_out        (pc_add4_ex),
        .inst_out           (inst_ex),
        .alu_op_out         (alu_op_ex),
        .imm_out            (imm_ex),
        .rf_wa_out          (rf_wa_ex),
        .rf_we_out          (rf_we_ex),
        .alu_src0_sel_out   (alu_src0_sel_ex),
        .alu_src1_sel_out   (alu_src1_sel_ex),
        .dmem_access_out    (dmem_access_ex),
        .dmem_we_out        (dmem_we_ex),
        .rf_wd_sel_out      (rf_wd_sel_ex),
        .br_type_out        (br_type_ex),
        .rf_rd0_out         (rf_rd0_ex),
        .rf_rd1_out         (rf_rd1_ex)
    );

/* ------------------------------ EX Stage ------------------------------ */
    wire [31 : 0]   alu_res_ex;
    wire [31 : 0]   alu_src0_ex;
    wire [31 : 0]   alu_src1_ex;
    wire [ 1 : 0]   npc_sel_ex;
    wire [31 : 0]   npc_ex;
    wire [31 : 0]   pc_offset_ex;
    wire [31 : 0]   pc_jalr_ex;

    MUX_2 mux0 (
        .src0           (rf_rd0_ex),
        .src1           (pc_ex),
        .sel            (alu_src0_sel_ex),
        .res            (alu_src0_ex)
    );

    MUX_2 mux1 (
        .src0           (rf_rd1_ex),
        .src1           (imm_ex),
        .sel            (alu_src1_sel_ex),
        .res            (alu_src1_ex)
    );

    ALU alu (
        .alu_op         (alu_op_ex),
        .alu_src0       (alu_src0_ex),
        .alu_src1       (alu_src1_ex),
        .alu_res        (alu_res_ex)
    );

    BRANCH branch (
        .br_type        (br_type_ex),
        .br_src0        (rf_rd0_ex),
        .br_src1        (rf_rd1_ex),
        .npc_sel        (npc_sel_ex)
    );

    assign pc_offset_ex = alu_res_ex;

    CLR_LSB clr_lsb (
        .in             (pc_offset_ex),
        .out            (pc_jalr_ex)
    );

    MUX_3 npc_mux (
        .src0           (pc_add4_ex),
        .src1           (pc_offset_ex),
        .src2           (pc_jalr_ex),
        .sel            (npc_sel_ex),
        .res            (npc_ex)
    );

/* ------------------------------ EX/MEM ------------------------------ */

    wire [ 0 : 0]   commit_mem;
    wire [31 : 0]   pc_mem;
    wire [31 : 0]   pc_add4_mem;
    wire [31 : 0]   inst_mem;
    wire [ 4 : 0]   rf_wa_mem;
    wire [ 0 : 0]   rf_we_mem;
    wire [ 3 : 0]   dmem_access_mem;
    wire [ 0 : 0]   dmem_we_mem;
    wire [ 1 : 0]   rf_wd_sel_mem;
    wire [31 : 0]   rf_rd1_mem;
    wire [31 : 0]   alu_res_mem;

    PIPELINE_REG ex_mem (
        .clk                (clk),
        .rst                (rst),
        .en                 (global_en),
        .flush              (1'b0),
        .stall              (1'b0),

        // input
        .commit_in          (commit_ex),
        .pc_in              (pc_ex),
        .pc_add4_in         (pc_add4_ex),
        .inst_in            (inst_ex),
        .alu_op_in          (5'b0),
        .imm_in             (32'b0),
        .rf_wa_in           (rf_wa_ex),
        .rf_we_in           (rf_we_ex),
        .alu_src0_sel_in    (1'b0),
        .alu_src1_sel_in    (1'b0),
        .dmem_access_in     (dmem_access_ex),
        .dmem_we_in         (dmem_we_ex),
        .rf_wd_sel_in       (rf_wd_sel_ex),
        .br_type_in         (4'b0),
        .rf_rd0_in          (32'b0),
        .rf_rd1_in          (rf_rd1_ex),
        .alu_res_in         (alu_res_ex),
        .dmem_rd_out_in     (32'b0),
        .dmem_wd_out_in     (32'b0),

        // output
        .commit_out         (commit_mem),
        .pc_out             (pc_mem),
        .pc_add4_out        (pc_add4_mem),
        .inst_out           (inst_mem),
        .rf_wa_out          (rf_wa_mem),
        .rf_we_out          (rf_we_mem),
        .dmem_access_out    (dmem_access_mem),
        .dmem_we_out        (dmem_we_mem),
        .rf_wd_sel_out      (rf_wd_sel_mem),
        .rf_rd1_out         (rf_rd1_mem),
        .alu_res_out        (alu_res_mem)
    );

/* ------------------------------ MEM Stage ------------------------------ */

    wire [31 : 0]   dmem_wd_in_mem;
    wire [31 : 0]   dmem_rd_in_mem;
    wire [31 : 0]   dmem_wd_out_mem;
    wire [31 : 0]   dmem_rd_out_mem;

    SLU slu (
        .addr           (dmem_addr),
        .dmem_access    (dmem_access_mem),
        .rd_in          (dmem_rd_in_mem),
        .wd_in          (dmem_wd_in_mem),
        .rd_out         (dmem_rd_out_mem),
        .wd_out         (dmem_wd_out_mem)
    );

    assign dmem_wd_in_mem = rf_rd1_mem;
    assign dmem_we = dmem_we_mem;
    assign dmem_wdata = dmem_wd_out_mem;
    assign dmem_rd_in_mem = dmem_rdata;
    assign dmem_addr = alu_res_mem;

/* ------------------------------ MEM/WB ------------------------------ */

    wire [ 0 : 0]   commit_wb;
    wire [31 : 0]   pc_wb;
    wire [31 : 0]   pc_add4_wb;
    wire [31 : 0]   inst_wb;
    wire [ 4 : 0]   rf_wa_wb;
    wire [ 0 : 0]   rf_we_wb;
    wire [ 0 : 0]   dmem_we_wb;
    wire [ 1 : 0]   rf_wd_sel_wb;
    wire [31 : 0]   alu_res_wb;
    wire [31 : 0]   dmem_rd_out_wb;
    wire [31 : 0]   dmem_wd_out_wb;
    PIPELINE_REG mem_wb (
        .clk                    (clk),
        .rst                    (rst),
        .en                     (global_en),
        .flush                  (1'b0),
        .stall                  (1'b0),

        // input
        .commit_in              (commit_mem),
        .pc_in                  (pc_mem),
        .pc_add4_in             (pc_add4_mem),
        .inst_in                (inst_mem),
        .alu_op_in              (5'b0),
        .imm_in                 (32'b0),
        .rf_wa_in               (rf_wa_mem),
        .rf_we_in               (rf_we_mem),
        .alu_src0_sel_in        (1'b0),
        .alu_src1_sel_in        (1'b0),
        .dmem_access_in         (4'b0),
        .dmem_we_in             (dmem_we_mem),
        .rf_wd_sel_in           (rf_wd_sel_mem),
        .br_type_in             (4'b0),
        .rf_rd0_in              (32'b0),
        .rf_rd1_in              (32'b0),
        .alu_res_in             (alu_res_mem),
        .dmem_rd_out_in         (dmem_rd_out_mem),
        .dmem_wd_out_in         (dmem_wd_out_mem),

        // output
        .commit_out             (commit_wb),
        .pc_out                 (pc_wb),
        .pc_add4_out            (pc_add4_wb),
        .inst_out               (inst_wb),
        .rf_wa_out              (rf_wa_wb),
        .rf_we_out              (rf_we_wb),
        .dmem_we_out            (dmem_we_wb),
        .rf_wd_sel_out          (rf_wd_sel_wb),
        .alu_res_out            (alu_res_wb),
        .dmem_rd_out_out        (dmem_rd_out_wb),
        .dmem_wd_out_out        (dmem_wd_out_wb)
    );

/* ------------------------------ WB Stage ------------------------------ */

    wire [31 : 0]   rf_wd_wb;
    wire [31 : 0]   dmem_addr_wb;
    wire [31 : 0]   dmem_wdata_wb;

    MUX_3 rf_wd_mux (
        .src0           (pc_add4_wb),
        .src1           (alu_res_wb),
        .src2           (dmem_rd_out_wb),
        .sel            (rf_wd_sel_wb),
        .res            (rf_wd_wb)
    );

    assign dmem_addr_wb = alu_res_wb;
    assign dmem_wdata_wb = dmem_wd_out_wb;

endmodule
```