Lab 6

PB22030892 刘铠瑜

任务1：前递模块

FORWARDING.v

```
module FORWARDING (
    input                   [ 4 : 0]            rf_ra0_ex,
    input                   [ 4 : 0]            rf_ra1_ex,

    input                   [ 4 : 0]            rf_wa_mem,
    input                   [ 0 : 0]            rf_we_mem,
    input                   [ 4 : 0]            rf_wa_wb,
    input                   [ 0 : 0]            rf_we_wb,

    output      reg         [ 1 : 0]            forward0,
    output      reg         [ 1 : 0]            forward1
);

    always @(*) begin
        forward0 = 2'b00;
        forward1 = 2'b00;

        if (rf_wa_mem != 0 && rf_wa_mem == rf_ra0_ex && rf_we_mem) begin
            forward0 = 2'b10;
        end else if (rf_wa_wb != 0 && rf_wa_wb == rf_ra0_ex && rf_we_wb) begin
            forward0 = 2'b01;
        end

        if (rf_wa_mem != 0 && rf_wa_mem == rf_ra1_ex && rf_we_mem) begin
            forward1 = 2'b10;
        end else if (rf_wa_wb != 0 && rf_wa_wb == rf_ra1_ex && rf_we_wb) begin
            forward1 = 2'b01;
        end
    end
endmodule
```

CPU.v

```
    MUX_3 rf_rd0_mux (
        .src0          (rf_rd0_ex_old),
        .src1          (rf_wd_wb),
        .src2          (alu_res_mem),
        .sel           (forward0),
        .res           (rf_rd0_ex)
    );

    MUX_3 rf_rd1_mux (
        .src0          (rf_rd1_ex_old),
        .src1          (rf_wd_wb),
        .src2          (alu_res_mem),
        .sel           (forward1),
        .res           (rf_rd1_ex)
    );
```

前递模块的主要功能是解决流水线中的数据冒险问题。当指令需要的数据尚未写入寄存器文件时，前递模块会根据当前流水线的状态，将数据从后续阶段（MEM 或 WB）直接传递到当前阶段，从而避免停顿。
需要注意的一点是，当 MEM 和 WB 都与 EX 阶段需要的寄存器地址匹配时，应选择 MEM 阶段以获得最新数据

- **输入信号**:
  - `rf_ra0_ex` 和 `rf_ra1_ex`：当前 EX 阶段的寄存器读地址。
  - `rf_wa_mem` 和 `rf_wa_wb`：MEM 和 WB 阶段的寄存器写地址。
  - `rf_we_mem` 和 `rf_we_wb`：MEM 和 WB 阶段的寄存器写使能信号。
- **输出信号**:
  - `forward0` 和 `forward1`：前递选择信号，用于控制 MUX 的选择。

任务2：冒险检测模块

HAZARD_DETECT.v

```
module HAZARD_DETECT (
    input                   [ 1 : 0]            rf_rd_sel_ex,
    input                   [ 0 : 0]            rf_we_ex,
    input                   [ 4 : 0]            rf_wa_ex,
    input                   [ 4 : 0]            rf_ra0_id,
    input                   [ 4 : 0]            rf_ra1_id,

    input                   [ 1 : 0]            npc_sel_ex,

    output      reg         [ 0 : 0]            stall_pc,
    output      reg         [ 0 : 0]            stall_if_id,
    output      reg         [ 0 : 0]            flush_if_id,
    output      reg         [ 0 : 0]            flush_id_ex
);

    always @(*) begin
        stall_pc = 1'b0;
        stall_if_id = 1'b0;
        flush_if_id = 1'b0;
        flush_id_ex = 1'b0;

        // LOAD-USE HAZARD
        if (rf_rd_sel_ex == 2'b10 && rf_we_ex && rf_wa_ex != 0) begin
            if (rf_wa_ex == rf_ra0_id || rf_wa_ex == rf_ra1_id) begin
                stall_pc = 1'b1;
                stall_if_id = 1'b1;
                flush_id_ex = 1'b1;
            end
        end

        if (npc_sel_ex != 0) begin
            flush_if_id = 1'b1;
            flush_id_ex = 1'b1;
        end
    end
endmodule
```

该模块主要处理以下两种情况：

1. **Load-Use Hazard**：当 ID 阶段的指令需要使用 MEM 阶段加载的数据时，需要停顿流水线，直到数据可用。
   具体实现: 停顿 `PC` 和 `IF/ID` 这两个流水线寄存器，并清空 `ID/EX` 来插入一个 `nop` 指令。
2. **跳转或分支指令**：当遇到跳转或分支指令时，需要刷新流水线，以避免指令乱序执行。
   具体实现：冲刷掉分支指令后面的两条指令，即清空 `IF/ID` 和 `ID/EX` 这两个流水线寄存器。