COD LAB5 实验报告

姓名：刘芷辰

学号：PB21111728

日期：2023.6.4

1. 实验题目

流水线CPU

2. 实验目标

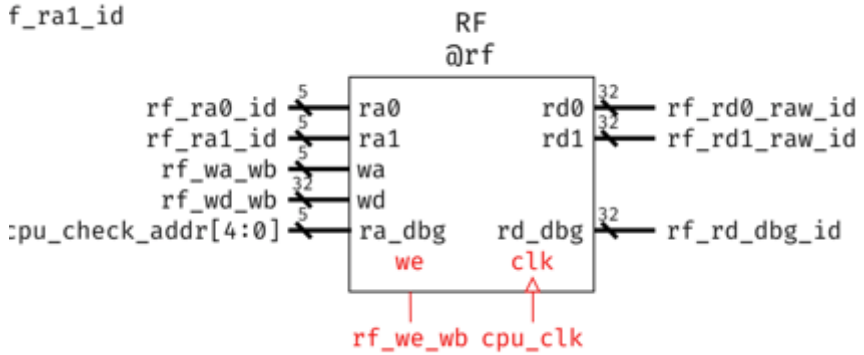
- 理解流水线CPU的结构和工作原理
- 熟练掌握流水线CPU数据通路和控制器的设计和描述方法
- 理解流水线CPU的调试方法

3. 实验内容

3.1 CPU设计与原理（主要描述和单周期区别较大的模块）

> RF模块

- 框图



- 代码

```
`timescale 1ns / 1ps

module RF(
    input clk,

    input [4:0] ra0,         //读端口0地址
    input [4:0] ra1,         //读端口1地址
    output reg [31:0] rd0,       //读端口0数据
    output reg [31:0] rd1,       //读端口1数据

    input we,                //写使能
    input [4:0] wa,          //写端口地址
    input [31:0] wd,         //写端口数据

    input [4:0] ra_dbg,
    output reg [31:0] rd_dbg
);
    reg [31:0] regfile[0:31];

    integer i;
    initial begin
        i = 0;
        while(i < 32) begin
            regfile[i] = 32'b0;
            i = i + 1;
        end
        regfile[2] = 32'h2ffc;
        regfile[3] = 32'h1800;
    end

    always @ (posedge clk) begin
        if (we) begin
            if (wa == 0) regfile[0] <= 0;
            else regfile[wa] <= wd;
        end
        else
            regfile[0] <= 0;
end

    always@(*)
    begin
        if(ra0 == 5'h0)
            rd0 <= 32'h0;
        else if(we & (ra0 == wa))
            rd0 <= wd;
```

```
        else
            rd0 <= regfile[ra0];
    end

    always@(*)
    begin
        if(ra1 == 5'h0)
            rd1 <= 32'h0;
        else if(we & (ra1 == wa))
            rd1 <= wd;
        else
            rd1 <= regfile[ra1];
    end

    always@(*)
    begin
        if(ra_dbg == 5'h0)
            rd_dbg <= 32'h0;
        else if(we & (ra_dbg == wa))
            rd_dbg <= wd;
        else
            rd_dbg <= regfile[ra_dbg];
    end


endmodule

```

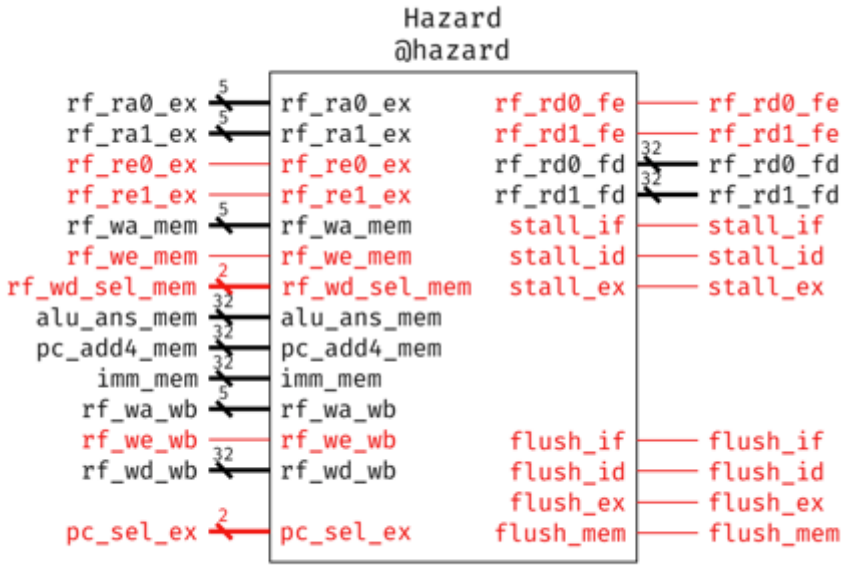
- 为了实现写优先，即在读写使能同时高电平时，能够直接将写入的数据读出，因此采用上述代码中

```
else if(we & (ra1 == wa))
    rd1 <= wd;
else
    rd1 <= regfile[ra1];
```

这样的结构来实现，保证写优先

> hazard模块

- 框图



- 代码

```
module Hazard(
    input [4:0]rf_ra0_ex,
    input [4:0]rf_ra1_ex,
    input rf_re0_ex,
    input rf_re1_ex,
    input [4:0]rf_wa_mem,
    input rf_we_mem,
    input [1:0]rf_wd_sel_mem,
    input [31:0]alu_ans_mem,
    input [31:0]pc_add4_mem,
    input [31:0]imm_mem,
    input [4:0]rf_wa_wb,
    input rf_we_wb,
    input [31:0]rf_wd_wb,
    input [1:0]pc_sel_ex,
    input jal_id,
    output reg rf_rd0_fe,
    output reg rf_rd1_fe,
    output reg [31:0]rf_rd0_fd,
    output reg [31:0]rf_rd1_fd,
    output reg stall_if,
    output reg stall_id,
    output reg stall_ex,
    output reg flush_if,
    output reg flush_id,
    output reg flush_ex,
    output reg flush_mem
```

```
);

//数据冒险-前递


        //在内存访问阶段（MEM）进行写操作，执行阶段（EX）进行读操作，
        //并且执行阶段需要读取的寄存器与内存阶段写入的寄存器相同,并非数据存储器读取结果
        //或者写回阶段（WB）进行写操作，执行阶段(EX)进行读操作，
        //并且执行阶段需要读取的寄存器与写回阶段写入的寄存器相同


//0
    always@(*)
    begin
        if((rf_we_mem & rf_re0_ex & (rf_ra0_ex == rf_wa_mem)) & (rf_wd_sel_mem != 2'b10))
        begin
            rf_rd0_fe = 1'b1;
            case(rf_wd_sel_mem)
                2'b00: begin
                    rf_rd0_fd = alu_ans_mem;
                end
                2'b01: begin
                    rf_rd0_fd = pc_add4_mem;
                end
                2'b11: begin
                    rf_rd0_fd = imm_mem;
                end
                default: rf_rd0_fd = 32'h0;
            endcase
        end
        else if(rf_we_wb & rf_re0_ex & (rf_ra0_ex == rf_wa_wb))
        begin
            rf_rd0_fe = 1'b1;
            rf_rd0_fd = rf_wd_wb;
        end
        else
            rf_rd0_fe = 1'b0;
    end

//1
    always@(*)
    begin
```

```
        if((rf_we_mem & rf_re1_ex & (rf_ra1_ex == rf_wa_mem)) & (rf_wd_sel_mem != 2'b10))
        begin
            rf_rd1_fe = 1'b1;
            case(rf_wd_sel_mem)
                2'b00: begin
                    rf_rd1_fd = alu_ans_mem;
                end
                2'b01: begin
                    rf_rd1_fd = pc_add4_mem;
                end
                2'b11: begin
                    rf_rd1_fd = imm_mem;
                end
                default: rf_rd1_fd = 32'h0;
            endcase
        end
        else if(rf_we_wb & rf_re1_ex & (rf_ra1_ex == rf_wa_wb))
        begin
            rf_rd1_fe = 1'b1;
            rf_rd1_fd = rf_wd_wb;
        end
        else
            rf_rd1_fe = 1'b0;
    end



//数据冒险-冒泡
// 这种冒险的判断方式前三步与上一小节相同，但最后一个条件为 MEM 段的写回数据
// 选择为数据存储器的结果

//stall_if、id、ex，flush_mem
    always@(*)
    begin

        if(rf_we_mem & ((rf_re0_ex & (rf_ra0_ex == rf_wa_mem)) | (rf_re1_ex & (rf_ra1_ex == rf_wa_mem))) & (rf_wd_sel_mem == 2'b10))
        begin
            stall_if = 1'b1;
            stall_id = 1'b1;
            stall_ex = 1'b1;
            flush_mem = 1'b1;
        end
        else begin
```

```
            stall_if = 1'b0;
            stall_id = 1'b0;
            stall_ex = 1'b0;
            flush_mem = 1'b0;
        end
    end


//控制冒险


//flush_id
    always@(*)
    begin
        if(((pc_sel_ex == 2'b01) | (pc_sel_ex == 2'b11)) | (jal_id == 1'b1))
            flush_id = 1'b1;
        else
            flush_id = 1'b0;
    end

//flush_ex
    always@(*)
    begin
        if((pc_sel_ex == 2'b01) | (pc_sel_ex == 2'b11))
            flush_ex = 1'b1;
        else
            flush_ex = 1'b0;
    end


endmodule

```

- 数据冒险-前递

  在内存访问阶段（MEM）进行写操作，执行阶段（EX）进行读操作，并且执行阶段需要读取的寄存器与内存阶段写入的寄存器相同,并非数据存储器读取结果，则将使能置为1，根据rf_wd_sel_mem选择前递数据

  在写回阶段（WB）进行写操作，执行阶段(EX)进行读操作，并且执行阶段需要读取的寄存器与写回阶段写入的寄存器相同，则将使能置为1，前递数据为rf_wd_wb

- 数据冒险-冒泡

  判断方式前三步与上一小节相同，但最后一个条件为 MEM 段的写回数据，则将三个stall信号置为1，并对EX/MEM段间寄存器清空

- 控制冒险

  在pc_sel_ex为跳转时，对 IF/ID 段间寄存器与 ID/EX 段间寄存器进行清空（jal选做在下面描述）

> 选做1：jal

- 主要变动：

```
//Immediate
    Immediate immediate(
        .inst(inst_id),
        .imm_type(imm_type_id),
        .imm(imm_id)
    );

    assign jal_pc = pc_cur_id + imm_id;
```

- ```
  module NPC_SEL(
      input [31:0]pc_add4_if,
      input [31:0]pc_jalr_ex,
      input [31:0]alu_ans_ex,
      input [31:0]jal_pc,
      input [1:0]pc_sel_ex,
      input jal_id,
      output reg [31:0]pc_next
  );

      always@(*)
      begin
          if(pc_sel_ex == 2'b01)  //jalr
              pc_next = pc_jalr_ex;
          else if(pc_sel_ex == 2'b11)     //br
              pc_next = alu_ans_ex;
          else if(jal_id)
              pc_next = jal_pc;
  ```
  ```

```
19          else
20              pc_next = pc_add4_if;
21      end
22
23  endmodule
```

- 实验原理

  在cpu模块中通过assgin计算jal_pc，因为pc_cur_id和imm_id均在id段得到，因此不必在ex中的alu计算，从而实现了jal提前到id段

  在NPC_SEL中，直接选择jal_pc,不需要通过alu在ex阶段的计算获得

  控制冒险时，jal信号为1时，只需清空IF/ID 段间寄存器

3.2 实验结果

> 估算流水线 CPU 的最小需要时钟周期与其相对单周期的指令平均所需时间的改进

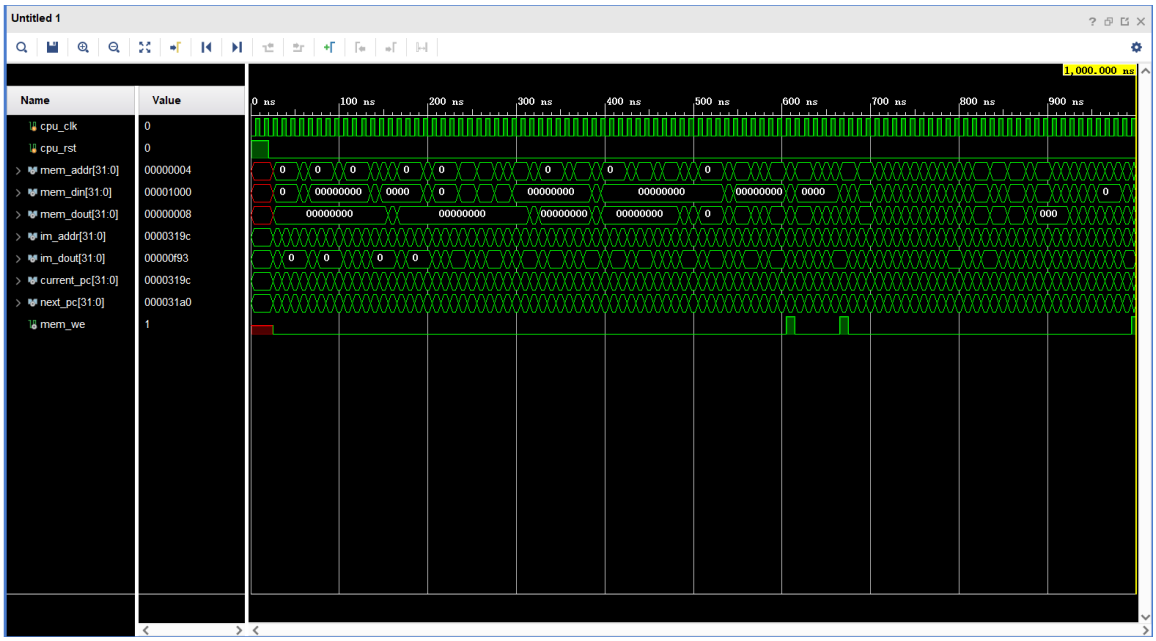
从作业6里面找到一组数据作为依据

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

- 流水线CPU最小时钟周期：350ps
- 单周期CPU周期：1250ps
- 改进：约3.57倍

> test

- 仿真



- 烧写

  已在线下检查

4. 总结

本次实验难度较大

通过流水线CPU的设计，更加了解了计算机体系结构，对riscv指令集的架构理解更加深入

对理解较为复杂的verilog代码之间的联系很有帮助