

# 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

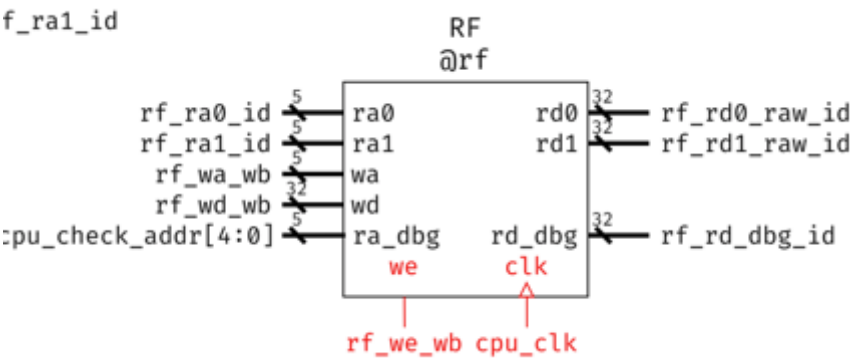
- 理解流水线CPU的结构和工作原理
- 熟练掌握流水线CPU数据通路和控制器的设计和描述方法
- 理解流水线CPU的调试方法

## 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

```

1  `timescale 1ns / 1ps
2
3  module RF(
4      input clk,
5
6      input [4:0] ra0,          //读端口0地址
7      input [4:0] ra1,          //读端口1地址
8      output reg [31:0] rd0,     //读端口0数据
9      output reg [31:0] rd1,     //读端口1数据
10
11     input we,                  //写使能
12     input [4:0] wa,            //写端口地址
13     input [31:0] wd,           //写端口数据
14
15     input [4:0] ra_dbg,
16     output reg [31:0] rd_dbg
17 );
18     reg [31:0] regfile[0:31];
19
20     integer i;
21     initial begin
22         i = 0;
23         while(i < 32) begin
24             regfile[i] = 32'b0;
25             i = i + 1;
26         end
27         regfile[2] = 32'h2ffc;
28         regfile[3] = 32'h1800;
29     end
30
31     always @ (posedge clk) begin
32         if (we) begin
33             if (wa == 0) regfile[0] <= 0;
34             else regfile[wa] <= wd;
35         end
36         else
37             regfile[0] <= 0;
38     end
39
40     always@(*)
41     begin
42         if(ra0 == 5'h0)
43             rd0 <= 32'h0;
44         else if(we & (ra0 == wa))
45             rd0 <= wd;

```

```

46         else
47             rd0 <= regfile[ra0];
48         end
49
50         always@(*)
51         begin
52             if(ra1 == 5'h0)
53                 rd1 <= 32'h0;
54             else if(we & (ra1 == wa))
55                 rd1 <= wd;
56             else
57                 rd1 <= regfile[ra1];
58         end
59
60         always@(*)
61         begin
62             if(ra_dbg == 5'h0)
63                 rd_dbg <= 32'h0;
64             else if(we & (ra_dbg == wa))
65                 rd_dbg <= wd;
66             else
67                 rd_dbg <= regfile[ra_dbg];
68         end
69
70
71     endmodule
72

```

- 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

```

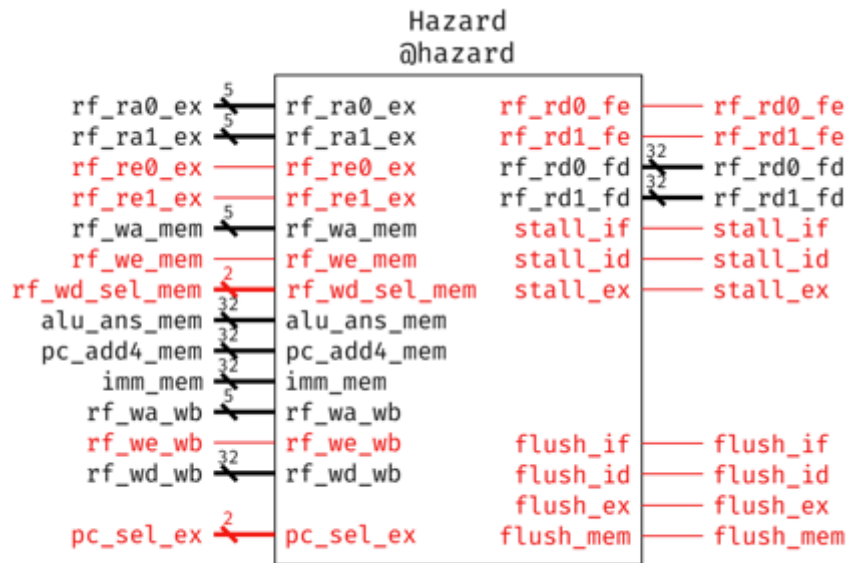
        else if(we & (ra1 == wa))
            rd1 <= wd;
        else
            rd1 <= regfile[ra1];

```

这样的结构来实现，保证写优先

## hazard模块

- 框图



- 代码

```

1  module Hazard(
2      input  [4:0]rf_ra0_ex,
3      input  [4:0]rf_ra1_ex,
4      input  rf_re0_ex,
5      input  rf_re1_ex,
6      input  [4:0]rf_wa_mem,
7      input  rf_we_mem,
8      input  [1:0]rf_wd_sel_mem,
9      input  [31:0]alu_ans_mem,
10     input  [31:0]pc_add4_mem,
11     input  [31:0]imm_mem,
12     input  [4:0]rf_wa_wb,
13     input  rf_we_wb,
14     input  [31:0]rf_wd_wb,
15     input  [1:0]pc_sel_ex,
16     input  jal_id,
17     output reg rf_rd0_fe,
18     output reg rf_rd1_fe,
19     output reg [31:0]rf_rd0_fd,
20     output reg [31:0]rf_rd1_fd,
21     output reg stall_if,
22     output reg stall_id,
23     output reg stall_ex,
24     output reg flush_if,
25     output reg flush_id,
26     output reg flush_ex,
27     output reg flush_mem

```

```

28 );
29
30 //数据冒险-前递
31
32
33 //在内存访问阶段（MEM）进行写操作，执行阶段（EX）进行读操作，
34 //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
35 //或者写回阶段（WB）进行写操作，执行阶段（EX）进行读操作，
36 //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
37
38
39 //0
40     always@(*)
41     begin
42         if((rf_we_mem & rf_re0_ex & (rf_ra0_ex == rf_wa_mem)) &
   (rf_wd_sel_mem != 2'b10))
43         begin
44             rf_rd0_fe = 1'b1;
45             case(rf_wd_sel_mem)
46                 2'b00: begin
47                     rf_rd0_fd = alu_ans_mem;
48                 end
49                 2'b01: begin
50                     rf_rd0_fd = pc_add4_mem;
51                 end
52                 2'b11: begin
53                     rf_rd0_fd = imm_mem;
54                 end
55                 default: rf_rd0_fd = 32'h0;
56             endcase
57         end
58         else if(rf_we_wb & rf_re0_ex & (rf_ra0_ex == rf_wa_wb))
59         begin
60             rf_rd0_fe = 1'b1;
61             rf_rd0_fd = rf_wd_wb;
62         end
63         else
64             rf_rd0_fe = 1'b0;
65     end
66
67 //1
68     always@(*)
69     begin

```

```

70         if((rf_we_mem & rf_rel_ex & (rf_ral_ex == rf_wa_mem)) &
(rf_wd_sel_mem != 2'b10))
71             begin
72                 rf_rdl_fe = 1'b1;
73                 case(rf_wd_sel_mem)
74                     2'b00: begin
75                         rf_rdl_fd = alu_ans_mem;
76                     end
77                     2'b01: begin
78                         rf_rdl_fd = pc_add4_mem;
79                     end
80                     2'b11: begin
81                         rf_rdl_fd = imm_mem;
82                     end
83                     default: rf_rdl_fd = 32'h0;
84                 endcase
85             end
86         else if(rf_we_wb & rf_rel_ex & (rf_ral_ex == rf_wa_wb))
87             begin
88                 rf_rdl_fe = 1'b1;
89                 rf_rdl_fd = rf_wd_wb;
90             end
91         else
92             rf_rdl_fe = 1'b0;
93     end
94
95
96
97     //数据冒险-冒泡
98     // 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
99     // 选择为数据存储器器的结果
100
101     //stall_if、id、ex、flush_mem
102     always@(*)
103     begin
104
105         if(rf_we_mem & ((rf_re0_ex & (rf_ra0_ex == rf_wa_mem)) |
(rf_rel_ex & (rf_ral_ex == rf_wa_mem))) & (rf_wd_sel_mem == 2'b10))
106             begin
107                 stall_if = 1'b1;
108                 stall_id = 1'b1;
109                 stall_ex = 1'b1;
110                 flush_mem = 1'b1;
111             end
112             else begin

```

```

113         stall_if = 1'b0;
114         stall_id = 1'b0;
115         stall_ex = 1'b0;
116         flush_mem = 1'b0;
117     end
118 end
119
120
121 //控制冒险
122
123
124 //flush_id
125     always@(*)
126     begin
127         if((pc_sel_ex == 2'b01) | (pc_sel_ex == 2'b11)) | (jal_id ==
1'b1))
128             flush_id = 1'b1;
129         else
130             flush_id = 1'b0;
131     end
132
133 //flush_ex
134     always@(*)
135     begin
136         if((pc_sel_ex == 2'b01) | (pc_sel_ex == 2'b11))
137             flush_ex = 1'b1;
138         else
139             flush_ex = 1'b0;
140     end
141
142
143 endmodule
144

```

- 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;
```

- ```
1  module NPC_SEL(
2      input  [31:0]pc_add4_if,
3      input  [31:0]pc_jalr_ex,
4      input  [31:0]alu_ans_ex,
5      input  [31:0]jal_pc,
6      input  [1:0]pc_sel_ex,
7      input  jal_id,
8      output reg [31:0]pc_next
9  );
10
11  always@(*)
12  begin
13      if(pc_sel_ex == 2'b01) //jalr
14          pc_next = pc_jalr_ex;
15      else if(pc_sel_ex == 2'b11) //br
16          pc_next = alu_ans_ex;
17      else if(jal_id)
18          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assign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

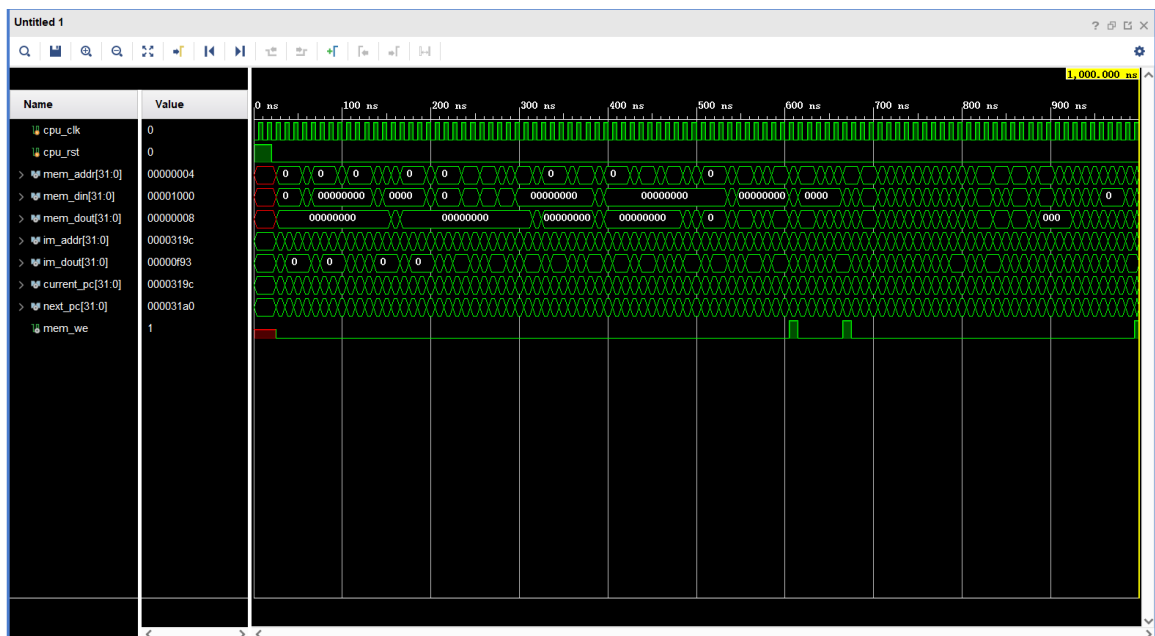
从作业6里面找到一组数据作为依据

| 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