浙江大学

本科实验报告

课程名称：	计算机体系结构
姓　　名：	姜雨童
学　　院：	计算机科学与技术学院
专　　业：	计算机科学与技术
邮　　箱：	3220103450@zju.edu.cn
QQ　　号：	1369218489
电　　话：	18867766468
指导教师：	王小航
报告日期：	2024 年 11 月 28 日

浙江大学实验报告

课程名称：计算机体系结构 实验类型：综合

实验项目名称：Lab05 Dynamically Scheduled Pipelines using Scoreboarding

学生姓名：姜雨童 学号：33220103450 同组学生姓名：/

实验地点：玉泉曹西 301 实验日期：2024 年 11 月 28 日

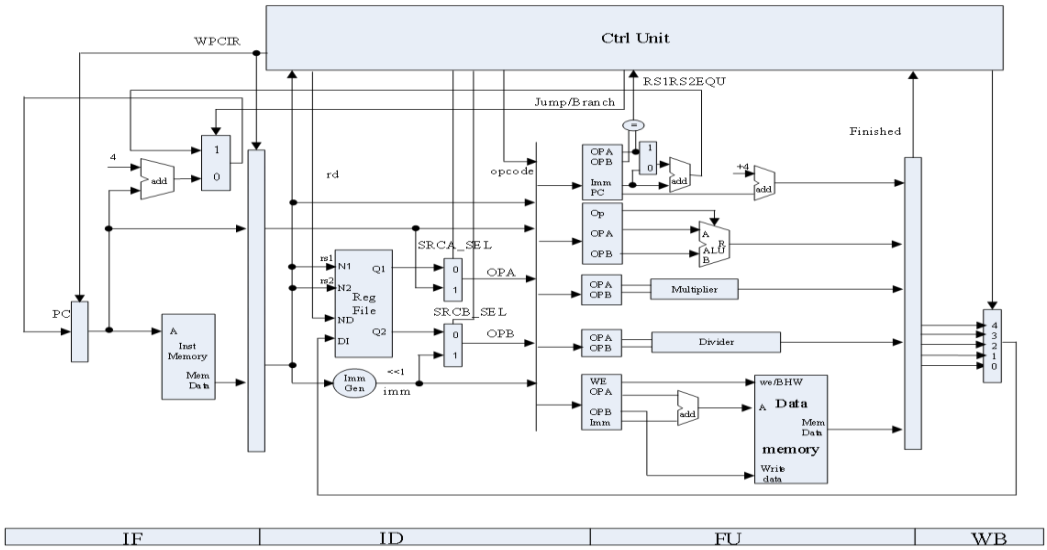
一、目标与原理

1-1 实验目的

- Redesign the pipelines with IF/ID/FU/WB stages and FU stage supporting **multicycle** operations.
- Redesign of CPU Controller.

1-2 实验内容

1. 更新数据通路，pipeline CPU 的 IF、ID 和 WB 阶段不改变，EXE 与 MEM 阶段合并为 FU。



2．补充完成 FU 所需的 ALU、乘法器、除法器、MEM 和 JUMP 单元。

3. 更新处理器控制模块，保证在 FU 阶段执行多周期操作时，其他部分产生 stall。

二、操作方法与实验步骤

2-1 ALU 模块

该模块定义了一个 state 寄存器，用来指示计算单元的占用状态/是否结束占用。使能信号 EN 为 1 且 ALU 模块未被占用时，开始执行 ALU 操作，并标记占用。

```
always@(posedge clk) begin
    if(EN & ~state) begin // state == 0
        Control <= ALUControl;
        A <= ALUA;
        B <= ALUB; //to fill sth.in
        state <= 1;
    end
    else state <= 0;
end
```

2-2 MUL 模块

浮点数乘法器计算需要 7 个时钟周期，因此该模块定义了一个 7 位的寄存器 state，开始计算时，最高位置一，每个周期将一向右移，最低为为一时结束计算。

```
always@(posedge clk) begin
    if(EN & ~|state) begin
        A_reg <= A;
        B_reg <= B;
        state <= 7'b1000000;
    end
    else state <= {1'b0, state[6:1]}
end
```

2-3 DIV 模块

同理，定义了用于判断的寄存器 state，此处 state 只有一位因为除法器计算结束会发出信号 res_valid。

```
always @(posedge clk) begin
    if(EN & ~state) begin
        A_valid <= 1;
        B_valid <= 1;
        A_reg <= A;
        B_reg <= B;
        state <= 1;
    end
    else if(res_valid) begin
        A_valid <= 0;
        B_valid <= 0;
```

```
            state <= 0;
        end
    end
```

2-4 JUMP 模块

根据数据通路：不发生跳转时的地址 PC_wb 为 PC+4；发生跳转时的地址 PC_jump 需要根据 JALR 信号判断来决定是 OP A 还是 PC 加上 imm。

```
    always@(posedge clk) begin
        if(EN & ~state) begin
            JALR_reg <= JALR;
            cmp_ctrl_reg <= cmp_ctrl;
            rs1_data_reg <= rs1_data;
            rs2_data_reg <= rs2_data;
            imm_reg <= imm;
            PC_reg <= PC;
            state <= 1;
        end
        else begin
            state <= 0;
        end
    end

    cmp_32 cmp(.a(rs1_data_reg),.b(rs2_data_reg),.ctrl(cmp_ctrl_reg),.c(cmp_res));
    assign PC_jump = (JALR)? (rs1_data_reg + imm_reg) : (PC_reg + imm_reg);
    assign PC_wb = PC_reg + 32'd4;
```

2-5 MEM 模块

同样地，由于内存访问也是多周期操作，state 为 2 位寄存器，需要位移控制：模块未被占用时值为零，开始访问内存时最高位置一，随后每周期右移一位，直到最低位为一结束。

```
    always@(posedge clk) begin
        if(EN & ~|state) begin
            mem_w_reg <= mem_w;
            bhw_reg <= bhw;
            rs1_data_reg <= rs1_data;
            rs2_data_reg <= rs2_data;
            imm_reg <= imm;
            state <= 2'b10;
        end
        else begin
            state <= (state >> 1);
        end
    end
    wire[31:0] addr;
    assign addr = rs1_data_reg + imm_reg;
```

2-6 RV32Core 模块

与前面实验的 pipelineCPU 类似，本实验也需要处理 data hazard 和 control hazard。
对于 data hazard，等 FU 和 WB 执行完后，其他被 stall 的模块才可恢复执行，进入 FU；对于 control hazard，实验中采用 predict-not-taken 策略
具体代码实现上，需要补全四行代码：ImmGen，ALU 输入的两个数据，以及 WB 阶段 mux_DtR。
另外，补全 Data2Reg 部分时接口和原理图并不完全一致，需要根据 CtrlUnit 部分的代码来：

```
wire[2:0] use_FU =  {3{use_ALU}}  & 3'd1 |
                    {3{use_MEM}}  & 3'd2 |
                    {3{use_MUL}}  & 3'd3 |
                    {3{use_DIV}}  & 3'd4 |
                    {3{use_JUMP}} & 3'd5 ;
```

```
ImmGen imm_gen(.ImmSel(ImmSel_ctrl),.inst_field(inst_ID),.Imm_out(Imm_out_ID));

MUX2T1_32 mux_imm_ALU_ID_A(.I0(rs1_data_ID),.I1(PC_ID),.s(ALUSrcA_ctrl),.o(ALUA_ID));

MUX2T1_32
mux_imm_ALU_ID_B(.I0(rs2_data_ID),.I1(Imm_out_ID),.s(ALUSrcB_ctrl),.o(ALUB_ID));

MUX8T1_32
mux_DtR(.s(DatatoReg_ctrl),.I0(32'd0),.I1(ALUout_WB),.I2(mem_data_WB),.I3(mulres_WB),.I4
(divres_WB),.I5(PC_wb_WB),.I6(32'd0),.I7(32'd0),.O(wt_data_WB));
```

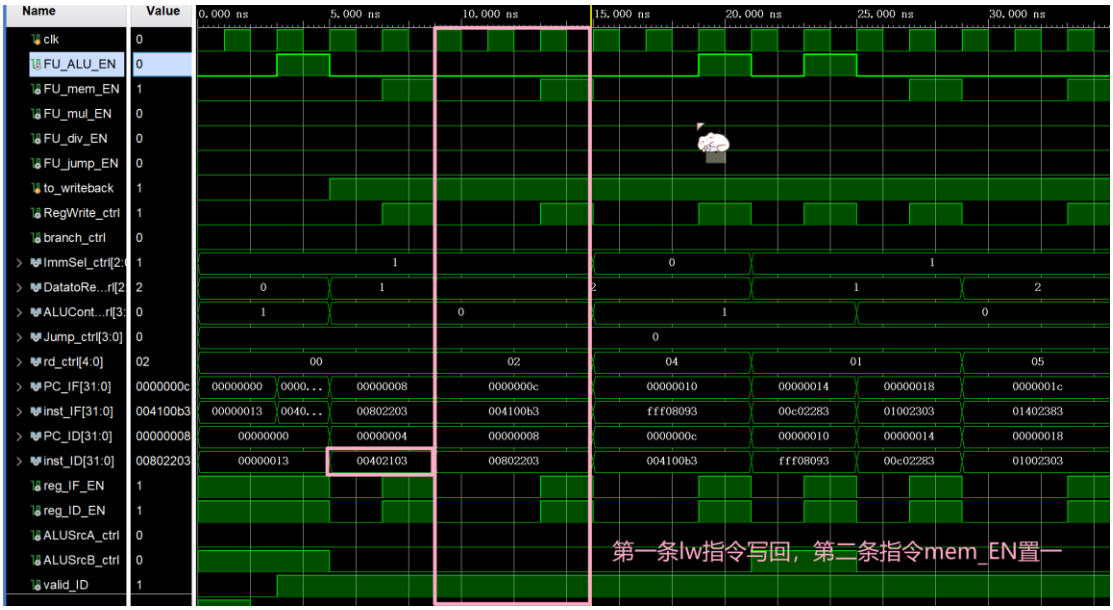
三、实验结果与分析

3-1 仿真

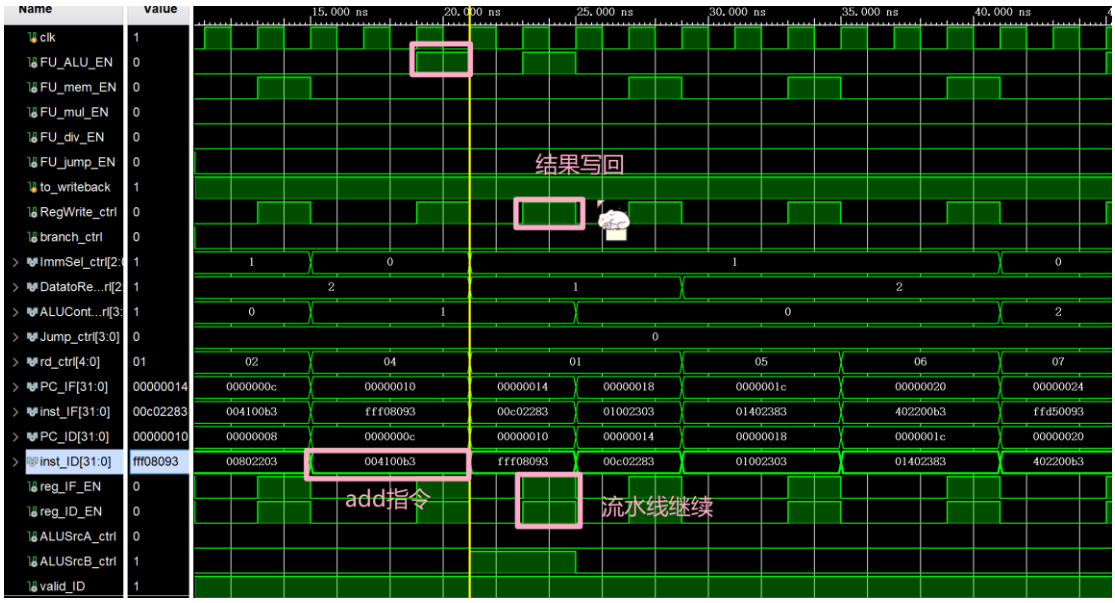
Rom.hex 内容如下：

NO.	Instruction	Addr.	Label	ASM
0	00000013	0	__start:	addi x0, x0, 0
1	00402103	4		lw x2, 4(x0)
2	00802203	8		lw x4, 8(x0)
3	004100b3	C		add x1, x2, x4
4	fff08093	10		addi x1, x1, -1
5	00c02283	14		lw x5, 12(x0)
6	01002303	18		lw x6, 16(x0)
7	01402383	1C		lw x7, 20(x0)
8	402200b3	20		sub x1,x4,x2
9	ffd50093	24		addi x1,x10,-3
10	00520c63	28		beq x4,x5,label0
11	00420a63	2C		beq x4,x4,label0
12	00000013	30		addi x0,x0,0
13	00000013	34		addi x0,x0,0
14	00000013	38		addi x0,x0,0
15	00000013	3C		addi x0,x0,0
16	000040b7	40	label0:	lui x1,4
17	00c000ef	44		jal x1,12
18	00000013	48		addi x0,x0,0
19	00000013	4C		addi x0,x0,0
20	00000013	50		addi x0,x0,0
21	00000013	54		addi x0,x0,0
22	ffff0097	58		auipc x1, 0xffff0
23	0223c433	5C		div x8, x7, x2
24	025204b3	60		mul x9, x4, x5
25	022404b3	64		mul x9, x8, x2
26	00400113	68		addi x2, x0, 4
27	000000e7	6C		jalr x1,0(x0)

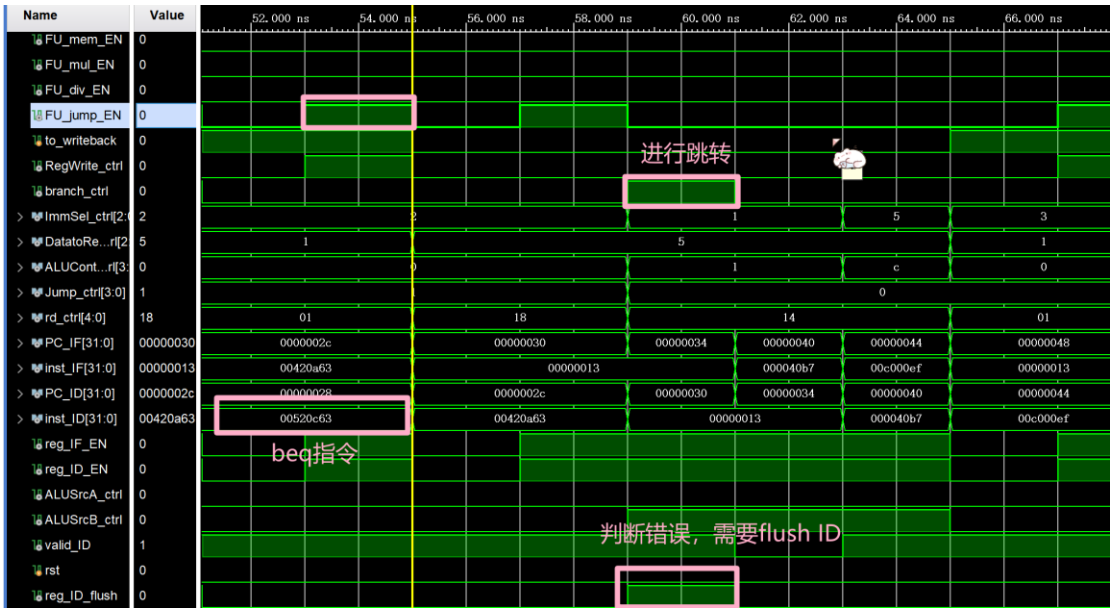
首先，lw 指令花费两个时钟周期，在第三个时钟周期将正确结果写回，继续流水线：



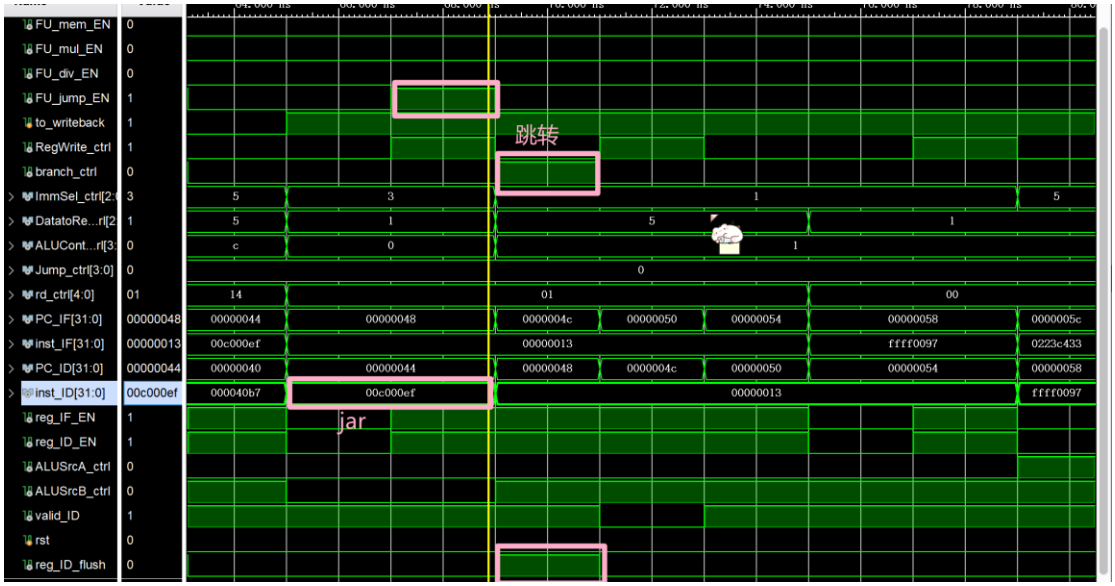
ALU 运算时，FU_ALU_EN 置一，执行一周期（此时流水线暂停）。结束后结果写回寄存器，流水线继续运行：



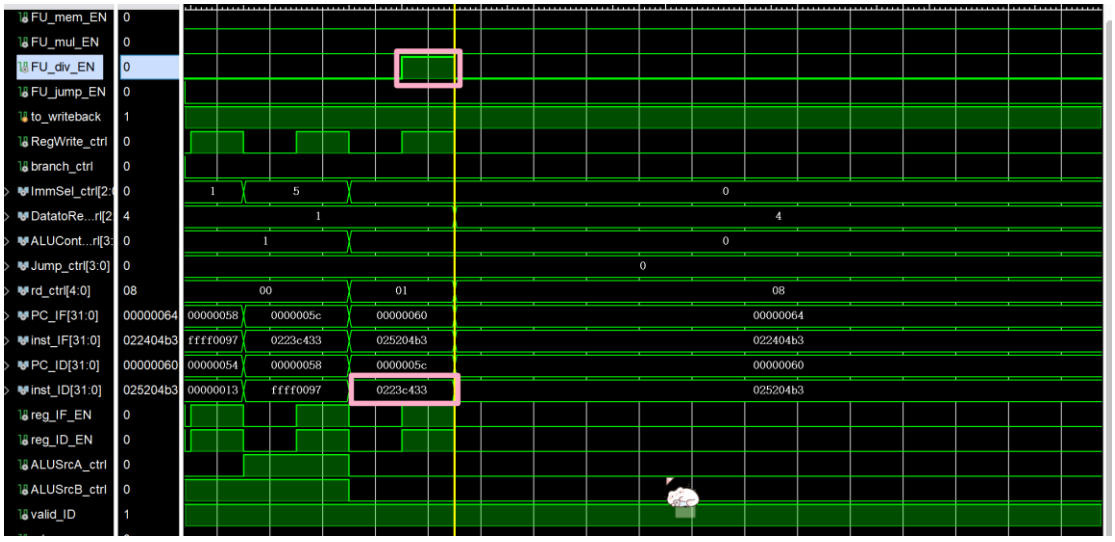
采用 predict-not-taken 策略，此处判断错误，需要跳转并 flush 部分指令：

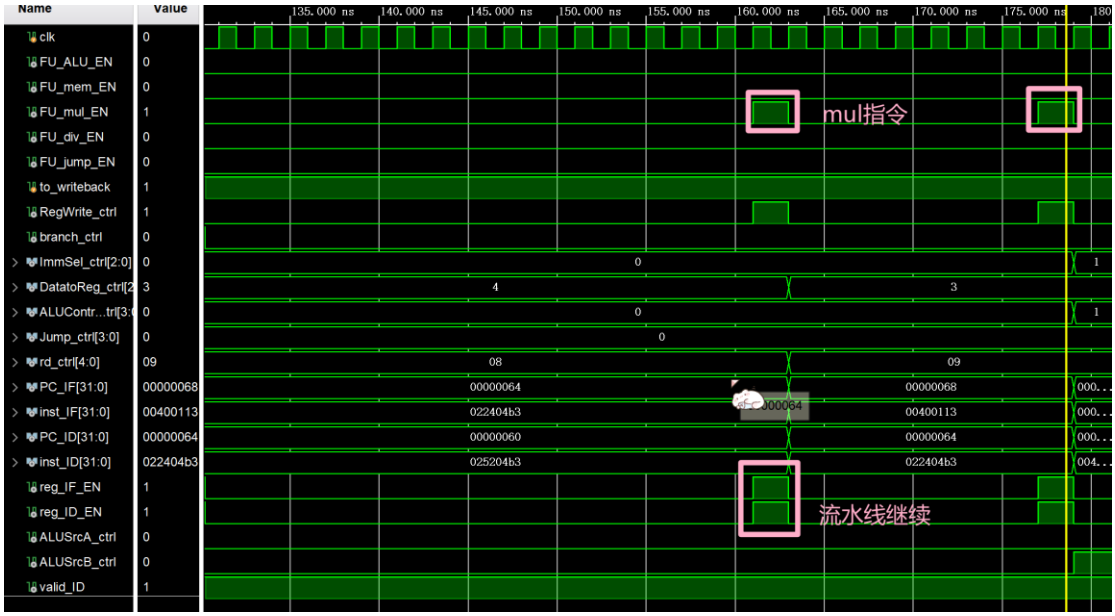


Jar 指令无条件跳转，并 flush 部分指令：

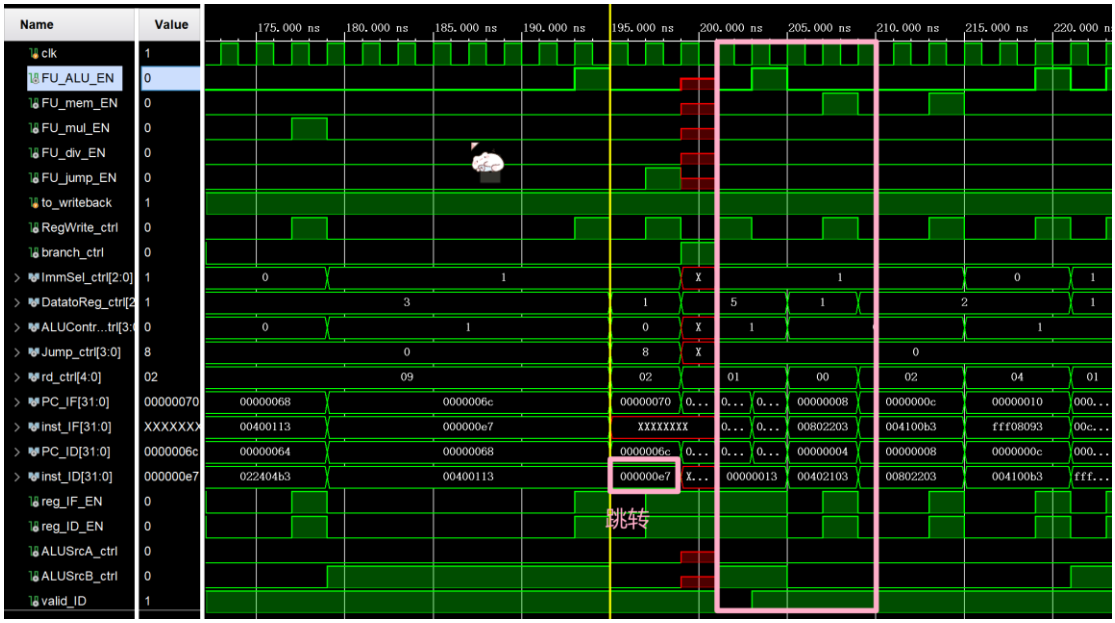


此处 Div 指令花费 38 个时钟周期，Mul 指令花费 8 个时钟周期：



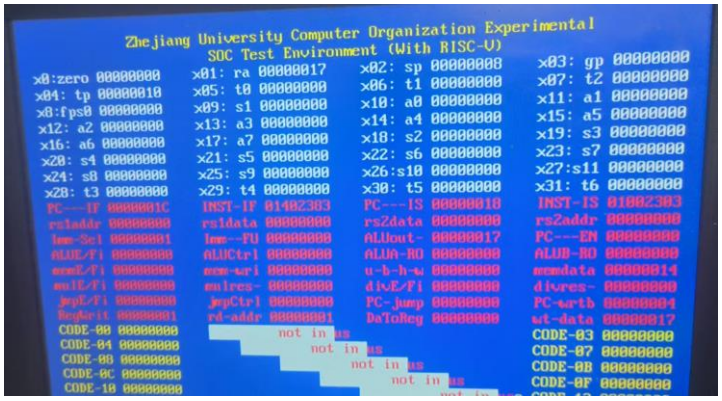


最后一条 jalr 指令，跳转后从头开始执行：



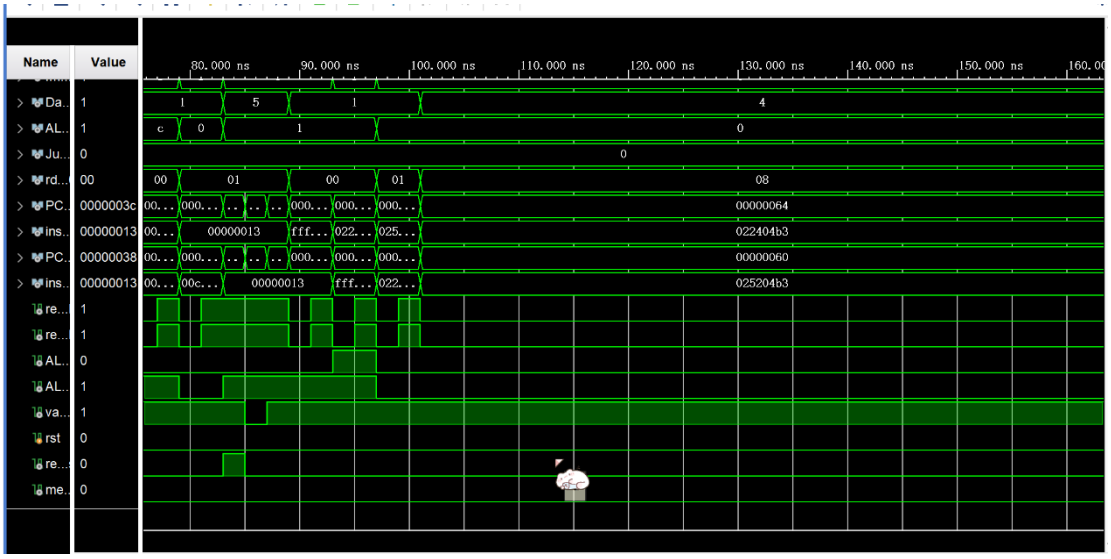
3-2 上板

由于上板是动态的不太好演示，且线下已经验收过上板结果了，这里只给出一张上板结果图作为辅证：

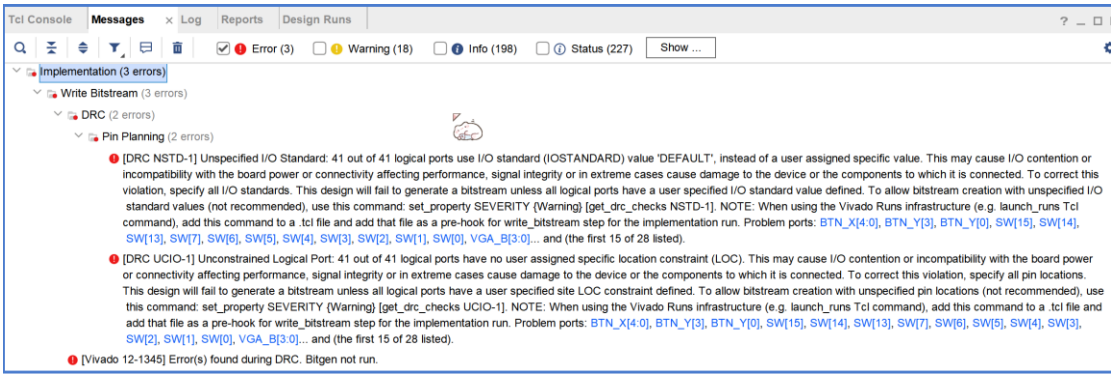


四、讨论、心得

本实验代码量不大，而且 ppt 内给出了详细的过程介绍和解释，总体而言难度不大。
实验中遇到过一个问题是仿真时发现在 div 结果写回的地方没有正确写回并继续流水线，而是一直卡在暂停的状态（下图）。
检查 div 模块代码发现是 `else if(res_valid) begin` 部分条件忘写了，加上后仿真结果正常。



再一个问题就是生成比特流的时候出现报错：



官方文档：56354 - Vivado write_bitstream - ERROR: [Drc 23-20] Rule violation (NSTD-1) Unspecified I/O Standard - X out of Y logical ports use I/O standard (IOSTANDARD) value 'DEFAULT', instead of a user assigned specific value

但是实验中不用这么麻烦，因为引脚约束文件已经在工程文件夹下，加入即可。