

# 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 10 月 7 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： Lab01 Pipelined CPU supporting RISC-V RV132 Instructions

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 10 月 7 日

## 一、目标与原理

### 1-1 实验目标

- Understand RISC-V RV32I instructions
- Master the design methods of pipelined CPU executing RV32I instructions
- Master the method of Pipeline Forwarding Detection and bypass unit design
- Master the methods of 1-cycle stall of Predict-not-taken branch design
- Master methods of program verification of Pipelined CPU executing RV32I instructions

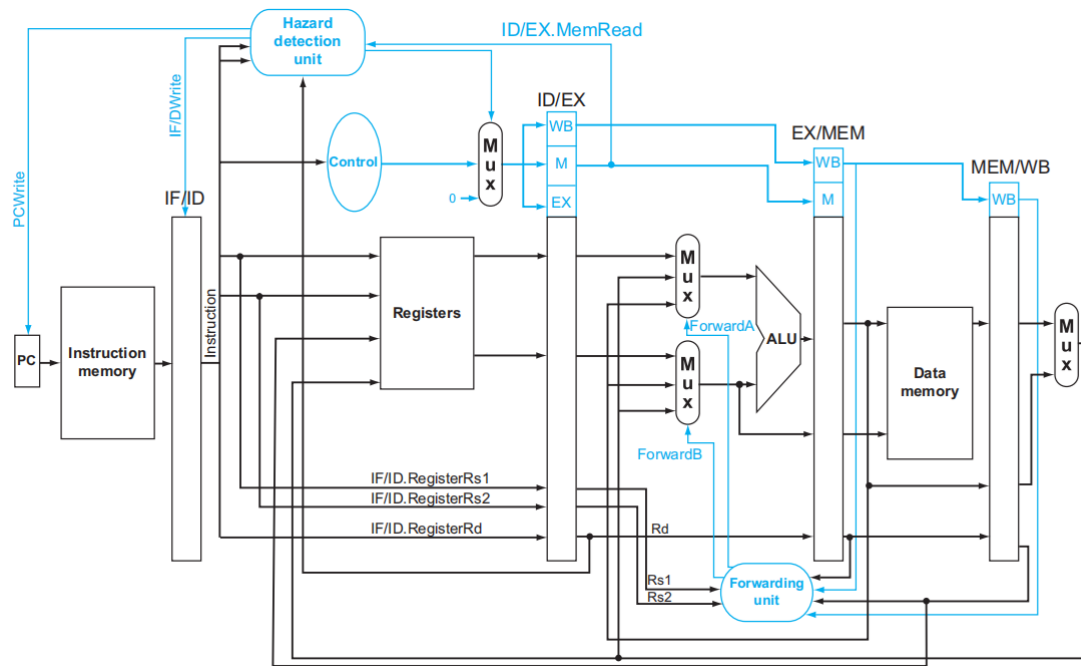
### 1-2 实验原理

在流水线中有三种 hazards: data hazard, structure hazard 和 control hazard。而实验中设计的五级流水线主要解决 data hazard 和 control hazard，解决方法为 forwarding、stall 和 predict-not-taken。

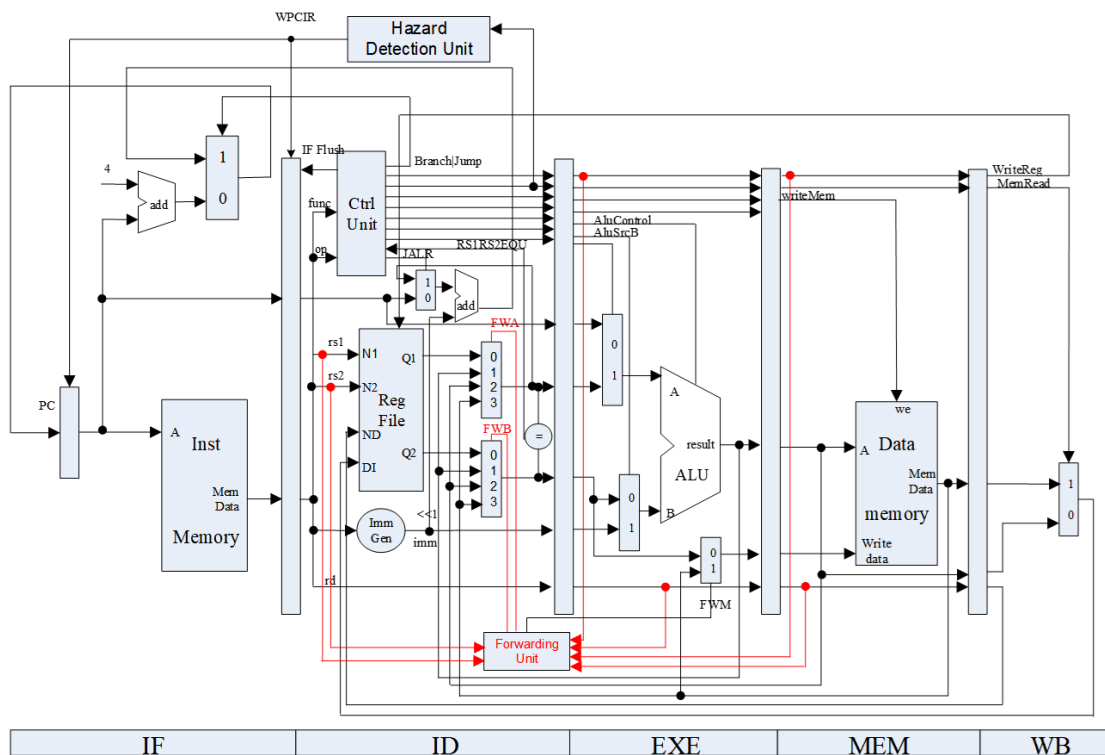
#### 1-2-1 forwarding

常见的 forwarding 实现为：将 MEM 阶段的 ALU 结果/WB 阶段的 ALU 结果/MemoryLoad 的结果 forward 给 EXE 阶段的 forwarding unit（图 1-1）。

与之不同的是，本实验提供的代码中，forwarding 的判断位于 ID 阶段，因此需要将 EX 和 MEM 阶段的数据 forward 给 ID 阶段（图 1-2）。此时，供选择的值（FWA/FWB）分别为 reg 中的值、EXE 阶段 ALU 结果、MEM 阶段 ALU 结果，以及 MEM 阶段 memory 中的值。



(图 1-1)



(图 1-2)

### 1-2-2 stall

Forwarding 无法解决的情况，需要 stall（插入空指令）。  
通过 IF 和 ID 阶段维持当前指令，冲刷后续阶段指令来实现。

### 1-2-3 predict-not-taken

流水线假设不发生跳转，继续按顺序执行指令。

若在 ID 阶段得知 Branch 结果时，发现预测错误，则 flush 下一条错误的指令（具体表现为 flush IF/ID pipeline register）。

## 二、操作方法与实验步骤

本实验框架代码已经给出，只需要补全以下四个文件中的代码：

CtrlUnit.v、HazardDetectionUnit.v、cmp\_32.v、RV32core.v

### 2-1 cmp\_32.v

本模块用于判断传入两个值的大小关系是否符合传入信号 ctrl（如 3'b001 代表两个值相等），符合时返回 1，否则返回 0。

```
assign c = (EQ & res_EQ |
            NE & res_NE |
            LT & res_LT |
            LTU & res_LTU |
            GE & res_GE |
            GEU & res_GEU
            ); //to fill sth. in ()
```

### 2-2 CtrlUnit.v

根据上下文补全指令解码部分（仅展示部分代码，下同）：

```
wire BEQ = Bop & funct3_0; //to fill sth. in
wire BNE = Bop & funct3_1; //to fill sth. in
wire BLT = Bop & funct3_4; //to fill sth. in
```

仿照 ImmSel 补全 cmp\_ctrl 信号（比较结果用来确定是否跳转）：

```
assign Branch = (cmp_res & B_valid) | JAL | JALR;
//to fill sth. in 2.再这里用比较后的结果确定是否跳转
parameter cmp_EQ = 3'b001; // copied from cmp_32.v
parameter cmp_NE = 3'b010;
parameter cmp_LT = 3'b011;
parameter cmp_LTU = 3'b100;
parameter cmp_GE = 3'b101;
parameter cmp_GEU = 3'b110;
assign cmp_ctrl = {3{BEQ}} & cmp_EQ |
                  {3{BNE}} & cmp_NE |
                  {3{BLT}} & cmp_LT |
                  {3{BLTU}} & cmp_LTU |
                  {3{BGE}} & cmp_GE |
```

```

        {3{BGEU}} & cmp_GEU;
//to fill sth. in 1.先这里判断应该怎么比较

```

根据 DataPath 补全信号:

```

assign ALUSrc_A = ~(JAL | JALR | AUIPC); //to fill sth. in
assign ALUSrc_B = I_valid | L_valid | S_valid | LUI | AUIPC; //to fill sth. in

```

区分 hazard 类型（一个原因在于提前到 ID 阶段判断时，需要用 hazard 类型来怕阶段 MEM 阶段 forward 的是 ALU 结果还是 memory 中的值）:

```

parameter hazard_optype_ALU = 2'd1;
// copied from PPT page-40, 和 HazardDetectionUnit.v 保持一致
parameter hazard_optype_LOAD = 2'd2;
parameter hazard_optype_STORE = 2'd3;
assign hazard_optype = hazard_optype_ALU & {2{R_valid | I_valid | LUI | AUIPC | JAL
    | JALR}} | hazard_optype_LOAD & {2{L_valid}} |
    hazard_optype_STORE & {2{S_valid}};
//to fill sth. in, ID 阶段解码指令是 ALU/LOAD/STORE

```

## 2-3 RV32core.v

根据 DataPath 补全模块调用代码（判断是否跳转 Branch）:

```

MUX2T1_32 mux_IF(.I0(PC_4_IF),.I1(jump_PC_ID),.s(Branch_ctrl),.o(next_PC_IF));
//to fill sth. in () +branch prediction, 0 不跳, 1 跳

```

根据图 1-2 补全模块调用代码（forward 部分）:

```

MUX4T1_32
mux_forward_A(.I0(rs1_data_reg),.I1(AlUout_EXE),.I2(AlUout_MEM),.I3(Datain_MEM),
//to fill sth. in ()
    .s(forward_ctrl_A),.o(rs1_data_ID));

MUX4T1_32
mux_forward_B(.I0(rs2_data_reg),.I1(AlUout_EXE),.I2(AlUout_MEM),.I3(Datain_MEM),
//to fill sth. in ()
    .s(forward_ctrl_B),.o(rs2_data_ID));

```

根据图 1-2 补全模块调用代码（EXE 部分）:

```

MUX2T1_32 mux_A_EXE(.I0(PC_EXE),.I1(rs1_data_EXE),.s(ALUSrc_A_EXE),.o(ALUA_EXE));
//to fill sth. in ()*
MUX2T1_32 mux_B_EXE(.I0(rs2_data_EXE),.I1(Imm_EXE),.s(ALUSrc_B_EXE),.o(ALUB_EXE));
//to fill sth. in ()

```

## 2-4 HazardDetectionUnit.v

处理 hazard（包括 forward, stall 和 predict-not-taken-branch）

判断是否符合 forward 或 stall 的情况：

```
wire forward_A_1 = rs1use_ID && rd_EXE && rs1_ID == rd_EXE && hazard_optype_EXE ==  
hazard_optype_ALU; // ALU_EXE  
wire forward_A_2 = rs1use_ID && rd_MEM && rs1_ID == rd_MEM && hazard_optype_MEM ==  
hazard_optype_ALU; // ALU_MEM  
wire forward_A_3 = rs1use_ID && rd_MEM && rs1_ID == rd_MEM && hazard_optype_MEM ==  
hazard_optype_LOAD; // LOAD_MEM  
wire forward_A_stall = rs1use_ID && rd_EXE && rs1_ID == rd_EXE && hazard_optype_EXE  
== hazard_optype_LOAD && hazard_optype_ID != hazard_optype_STORE;  
// 要用上一条指令 load 的结果
```

确定 forwardA 是三种情况中的哪一种：

```
assign forward_ctrl_A = {2{forward_A_1}} & 2'b01 |  
                        {2{forward_A_2}} & 2'b10 | {2{forward_A_3}} & 2'b11 ;
```

Stall: IF 和 ID 阶段维持当前指令，冲刷后续阶段的指令

```
// Case: stall  
wire stall = forward_A_stall | forward_B_stall;  
assign PC_EN_IF = ~stall; // PC 寄存器不会更新，IF 阶段维持当前指令  
assign reg_FD_stall = stall; // ID 阶段维持当前指令  
assign reg_DE_flush = stall; // EX 阶段不会执行 ID 阶段的指令  
  
assign forward_ctrl_ls = rs2_EXE == rd_MEM && hazard_optype_MEM == hazard_optype_LOAD  
&& hazard_optype_EXE == hazard_optype_STORE; // MEM 阶段 forward 的是 LOAD 指令
```

branch: ID 阶段知道结果错误，flush 后一条指令

```
// Case: branch  
assign reg_FD_flush = Branch_ID; // ID 阶段知道结果错误，flush 后一条指令
```

## 三、实验结果与分析

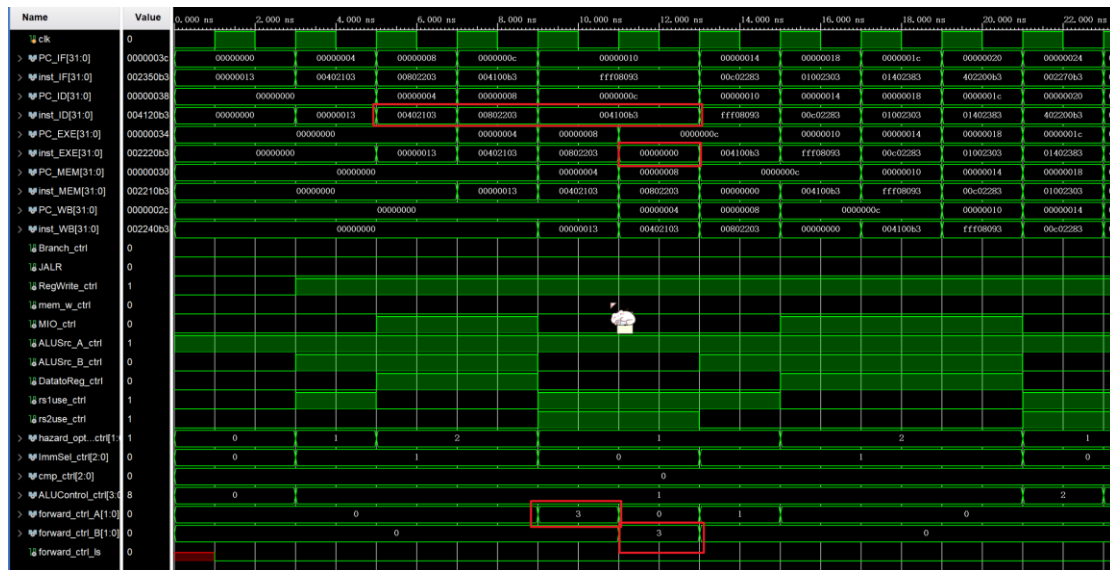
### 3-1 仿真结果

仿真结果完全符合预期，这里列举几个例子佐证：

#### Forward/stall:

截图中用红色方框框出对应指令和 forward/stall 信号。

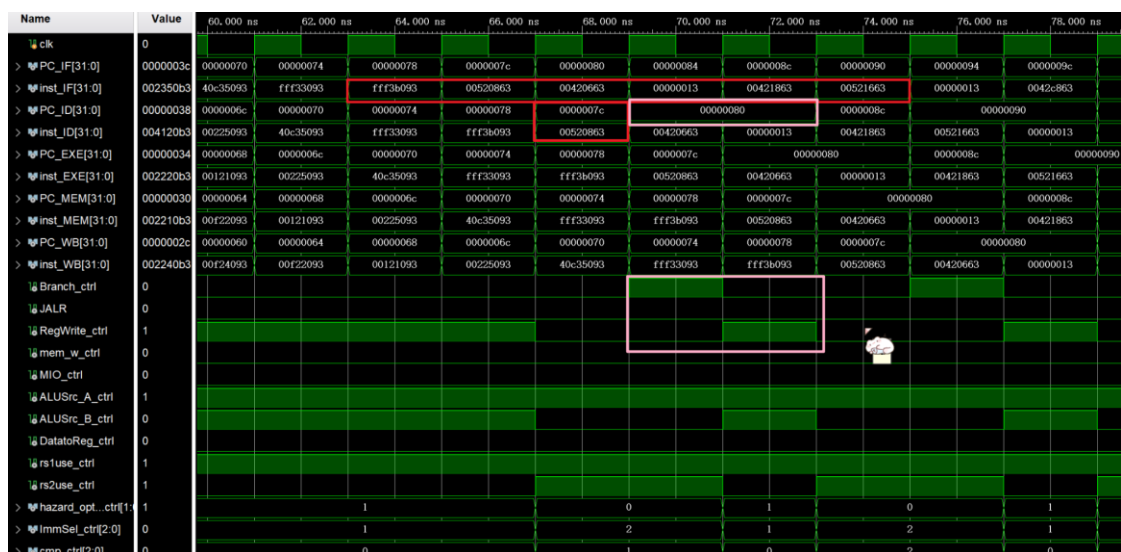
00402103	4		lw x2, 4(x0)
00802203	8		lw x4, 8(x0)
004100b3	C		add x1, x2, x4



## branch:

截图中用红色方框框出相关指令，粉色方框框出 branch 信号（包含了 stall）。

fff3b093	78		sltiu x1, x7, -1
00520863	7C		beq x4,x5,label0
00420663	80		beq x4,x4,label0
00000013	84		addi x0,x0,0
00000013	88		addi x0,x0,0
00421863	8C	label0:	bne x4,x4,label1
00521663	90		bne x4,x5,label1





## 3-2 上板验证

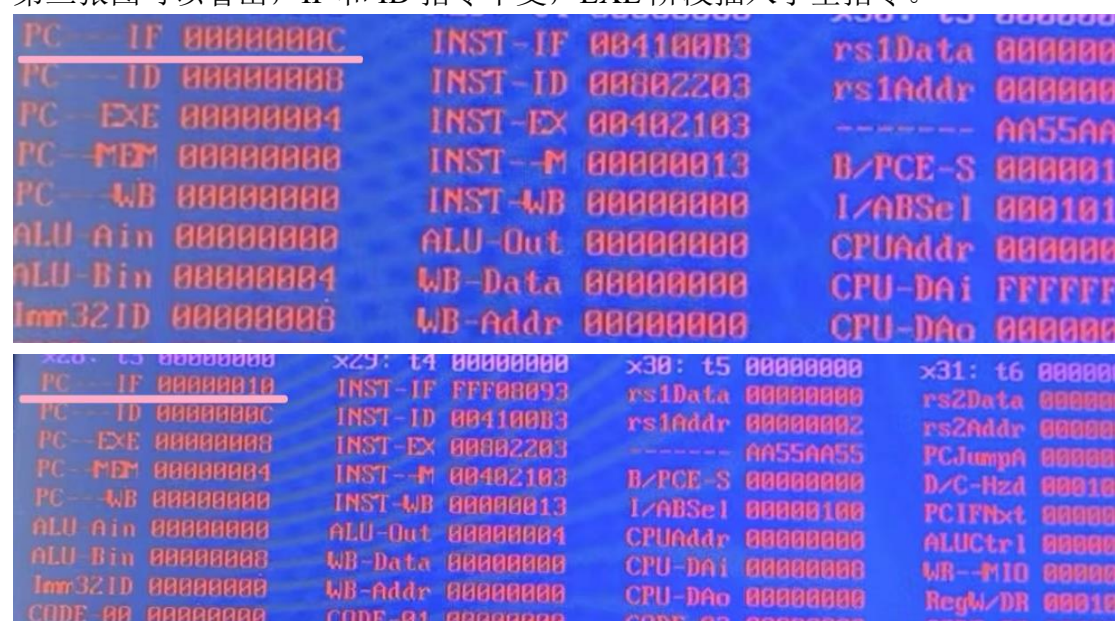
上板后界面如下：



下面截取连续几帧画面对报告 3-1 仿真结果中的两个例子进行说明：

**Forward/stall**（IF 阶段 PC 从 0C 到 14）：

第三张图可以看出，IF 和 ID 指令不变，EXE 阶段插入了空指令。





x24: s8 00000000	x25: s9 00000000	x26: s10 00000000	x27: s11 00000000
x28: t3 00000000	x29: t4 00000000	x30: t5 00000000	x31: t6 00000000
PC--IF 00000010	INST-IF FFF00093	rs1Data 00000000	rs2Data 00000000
PC--ID 0000000C	INST-ID 004100B3	rs1Addr 00000002	rs2Addr 00000000
PC--EXE 0000000C	INST-EX 00000000	----- AA55AA55	PCJumpA 00000000
PC--MEM 00000008	INST-M 00002203	B/PCE-S 00000100	D/C-Hzd 00000000
PC--WB 00000004	INST-WB 00402103	I/ABSel 00000100	PCIFNxt 00000000
ALU-Ain 00000000	ALU-Out 00000000	CPUAddr 00000000	ALUCtrl 00000000
ALU-Bin 00000000	WB-Data 00000000	CPU-Dai 00000010	WR--MIO 00000000
Imm32ID 00000000	WB-Addr 00000002	CPU-Dao 00000000	RegW/DR 00000000
CODE-00 00000000	CODE-01 00000000	CODE-02 00000000	CODE-03 00000000
CODE-04 00000000	CODE-05 00000000	CODE-06 00000000	CODE-07 00000000

x28: t3 00000000	x29: t4 00000000	x26: s10 00000000	x27: s11 00000000
PC--IF 00000014	INST-IF 00C02203	x30: t5 00000000	x31: t6 00000000
PC--ID 00000010	INST-ID FFF00093	rs1Data 00000000	rs2Data 00000000
PC--EXE 0000000C	INST-EX 004100B3	rs1Addr 00000001	rs2Addr 00000000
PC--MEM 0000000C	INST-M 00000000	----- AA55AA55	PCJumpA 00000000
PC--WB 00000008	INST-WB 00002203	B/PCE-S 00000100	D/C-Hzd 00000000
ALU-Ain 00000000	ALU-Out 00000000	I/ABSel 00010101	PCIFNxt 00000000
ALU-Bin 00000010	WB-Data 00000010	CPUAddr 00000000	ALUCtrl 00000000
Imm32ID FFFFFFFF	WB-Addr 00000004	CPU-Dai 00000010	WR--MIO 00000000
CODE-00 00000000	CODE-01 00000000	CPU-Dao 00000000	RegW/DR 00000000
CODE-04 00000000	CODE-05 00000000	CODE-02 00000000	CODE-03 00000000
CODE-08 00000000	CODE-09 00000000	CODE-06 00000000	CODE-07 00000000
CODE-0C 00000000	CODE-0D 00000000	CODE-0A 00000000	CODE-0B 00000000

**Branch** (IF 阶段 PC 从 78 到 84):

其中 PC\_IF 为 80 和 8C 的帧很快被跳过，对视频进行慢速处理后截图，因此部分数字不稳定，有闪动或是显示不清的情况。

x28: t3 00000000	x29: t4 00000000	x26: s10 00000000	x27: s11 00000000
PC--IF 00000070	INST-IF FFF30093	x30: t5 00000000	x31: t6 00000000
PC--ID 00000074	INST-ID FFF30093	rs1Data FFFF0000	rs2Data 00000000
PC--EXE 00000070	INST-EX 40C35093	rs1Addr 00000006	rs2Addr 00000000
PC--MEM 0000006C	INST-M 00225093	----- AA55AA55	PCJumpA 00000000
PC--WB 00000068	INST-WB 00121093	B/PCE-S 00000100	D/C-Hzd 00000000
ALU-Ain FFFF0000	ALU-Out 00000004	I/ABSel 00010101	PCIFNxt 00000000
ALU-Bin 0000040C	WB-Data 00000020	CPUAddr 00000000	ALUCtrl 00000000
Imm32ID FFFFFFFF	WB-Addr 00000001	CPU-Dai 00000000	WR--MIO 00000000
CODE-00 00000000	CODE-01 00000000	CPU-Dao 00000000	RegW/DR 00000000
CODE-04 00000000	CODE-05 00000000	CODE-02 00000000	CODE-03 00000000
CODE-08 00000000	CODE-09 00000000	CODE-06 00000000	CODE-07 00000000

x28: t3 00000000	x29: t4 00000000	x26: s10 00000000	x27: s11 00000000
PC--IF 0000007C	INST-IF 00520063	x30: t5 00000000	x31: t6 00000000
PC--ID 00000078	INST-ID FFF30093	rs1Data 0FFF0000	rs2Data 00000000
PC--EXE 00000074	INST-EX FFF30093	rs1Addr 00000007	rs2Addr 00000000
PC--MEM 00000070	INST-M 40C35093	----- AA55AA55	PCJumpA 00000000
PC--WB 0000006C	INST-WB 00225093	B/PCE-S 00000100	D/C-Hzd 00000000
ALU-Ain FFFF0000	ALU-Out FFFFFFFF	I/ABSel 00010101	PCIFNxt 00000000
ALU-Bin FFFFFFFF	WB-Data 00000004	CPUAddr 00000000	ALUCtrl 00000000
Imm32ID FFFFFFFF	WB-Addr 00000001	CPU-Dai 00000000	WR--MIO 00000000
CODE-00 00000000	CODE-01 00000000	CPU-Dao 00000000	RegW/DR 00000000
CODE-04 00000000	CODE-05 00000000	CODE-02 00000000	CODE-03 00000000
CODE-08 00000000	CODE-09 00000000	CODE-06 00000000	CODE-07 00000000



```

x24: s8 00000000 x25: s9 00000000 x22: s6 00000000
x28: t3 00000000 x29: t4 00000000 x26: s10 00000000
PC--IF 00000000 80 INST-IF 00420663 x30: t5 00000000
PC--ID 0000007C INST-ID 00520663 rs1Data 00000010
PC--EXE 00000070 INST-EX FFF30093 rs1Addr 00000001
PC--MEM 00000074 INST-M 00C35093 ----- AA55AA55
PC--WB 0000006C INST-WB 00225093 B/PCE-S 00000100
ALU-Ain FFFF0000 ALU-Out FFFFFFFF I/ABSel 00010101
ALU-Bin FFFFFFFF WB-Data 00000004 CPUAddr 00000000
Imm32ID FFFFFFFF WB-Addr 00000001 CPU-Dai 00000000
CODE-00 00000000 CODE-01 00000000 CPU-Dao 00000000
CODE-04 00000000 CODE-02 00000000

```

```

x28: t3 00000000 x29: t4 00000000 x28: s10 00000000
PC--IF 00000004 INST-IF 00000013 x30: t5 00000000
PC--ID 00000000 INST-ID 00420663 rs1Data 00000000
PC--EXE 0000007C INST-EX 00520663 rs1Addr 00000000
PC--MEM 00000078 INST-M FFF30093 ----- AA55AA55
PC--WB 00000074 INST-WB FFF30093 B/PCE-S 00010101
ALU-Ain 00000010 ALU-Out 00000001 I/ABSel 00020101
ALU-Bin 00000014 WB-Data 00000001 CPUAddr 00000000
Imm32ID 0000000C WB-Addr 00000001 CPU-Dai 00000000
CODE-00 00000000 CODE-01 00000000 CPU-Dao 00000000
CODE-04 00000000 CODE-02 00000000

```

```

x28: t3 00000000 x25: s9 00000000 x28: s10 00000000
PC--IF 0000000C x29: t4 00000000 x30: t5 00000000
PC--ID 00000000 INST-IF 00421063 rs1Data 00000000
PC--EXE 00000000 INST-ID 00000013 rs1Addr 00000000
PC--MEM 0000007C INST-EX 00420663 ----- AA55AA55
PC--WB 00000078 INST-M 00520663 B/PCE-S 00000100
ALU-Ain 00000010 ALU-Out 00000000 I/ABSel 00010101
ALU-Bin 00000010 WB-Data 00000001 CPUAddr 00000000
Imm32ID 00000000 WB-Addr 00000001 CPU-Dai FFFFFFFF
CODE-00 00000000 CODE-01 00000000 CPU-Dao 00000000
CODE-04 00000000 CODE-05 00000000 CODE-02 00000000

```

## 四、讨论、心得

本次实验基本上都是上学期计组所学的内容，因此难度并没有那么高。但是各种判断条件/mux 选择的线路都需要对照线路图仔细填写，一旦有细节出错就可能整个跑不出来（下图），因此实验过程还是略有些繁琐。

