浙江大学

本科实验报告

课程名称：计算机体系结构

姓　　名：张天逸

学　　院：计算机科学与技术学院

系：计算机科学与技术系

专　　业：计算机科学与技术

学　　号：3220106424

指导教师：姜晓红

2024 年　9　月　9　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：Lab1：Pipelined CPU supporting RISC-V RV32I Instructions

学生姓名：张天逸 专业：计算机科学与技术 学号：3220106424

同组学生姓名： 指导老师：姜晓红

实验地点：曹西 301 实验日期：2024 年 9 月 9 日

一、 实验目的和要求

本次实验为完成支持 RV32I 指令集的流水线 CPU，目的为 1）学会使用 Vivado 进行硬件设计开发 2）理解 RISC-V RV32I 指令 3）掌握执行 RV32I 指令的流水线 CPU 设计方法 4）掌握流水线 Forwarding 和 Bypass 的方法 5）掌握 Predict-not-Taken 的 Stall 方法 6）掌握在 CPU 上执行 RISCV 程序的方法

要求 1）编写一段能验证 CPU 功能的 RISCV 测试代码（比如要实现乘法功能，就编写一段带有多个乘法指令的代码），并规定每行代码预期的输出 2）根据框架/从头开始 设计 CPU，使其能正确执行 RISCV 测试代码并产生预期的输出结果 3）将 RISCV 代码放在指令储存器中输入 CPU，进行波形仿真观察输出是否与预期一致 4）最后下载到实验板上运行 CPU

二、 实验内容和原理

Predict not taken 的实现思路：Predict not taken 默认跳转条件不成立，如果发现条件成立则把已经读入的指令 flush 掉，具体代码中，reg_FD_flush 表示在前一个时钟取到了错误的指令，flush 掉，reg_FD_flush 作为 flush 传入 REG_IF_ID.v，进入判断 if flush ，IR 等待 Control Hazards

相关代码：

HazardDetection.v

```
        if (reg_DE_flush) begin // flush from ID
            hazard_optype_EX <= 2'b00;
        end
        else begin
            hazard_optype_EX <= hazard_optype_ID; // pass from ID
        end

        assign reg_FD_stall = (hazard_optype_ID != HAZARD_ST) &
                              (hazard_optype_EX == HAZARD_MEM) &
                              (|rd_EXE) &
                              (((rd_EXE == rs1_ID) & rs1use_ID) | ((rd_EXE == rs2_ID) & rs2use_ID));

        assign reg_DE_flush = reg_FD_stall;
        assign PC_EN_IF = ~reg_FD_stall;
        assign reg_FD_flush = Branch_ID;
```

REG_IF_ID.v

```
        if(rst) begin
             IR_ID <= 32'h00000000;                          //复位清零
             PCurrent_ID <= 32'h00000000;                    //复位清零
            end
        else if(EN)begin
              if(Data_stall)begin
                  IR_ID <= IR_ID;                          //IR waiting for Data Hazards 并暂停取指
                  PCurrent_ID <= PCurrent_ID;  end         //保存对应PC指针
              else if(flush)begin
                     IR_ID <= 32'h00000013;              //IR waiting for Control Hazards i清s除指令并暂停
                     PCurrent_ID <= PCurrent_ID;  end    //清除指令的指针(测试)
              else begin
                     IR_ID <= IR;                        //正常取指,传送下一流水级译码
                     PCurrent_ID <= PCOUT;  end          //当前取指PC地址，Branch/Junp指令计算目标地址用(非PC+4)
            end
            else begin
                IR_ID <= IR_ID;
                PCurrent_ID <= PCurrent_ID;
            end
```

3 个 forward 的实现思路：

有三种类型 forward 需要判断：执行阶段到执行阶段的前递（EX to EX），A_EX_Forward 和 B_EX_Forward 来控制，条件为 rs1use_ID 或 rs2use_ID 为真，表示当前指令的源操作数正在使用，rs1_ID == rd_EXE 或 rs2_ID == rd_EXE，表示当前指令的源操作数寄存器与执行阶段指令的结果寄存器相同，|rs1_ID 或 |rs2_ID 为真，表示源操作数寄存器编号非零，hazard_optype_EX == HAZARD_EX，表示执行阶段的指令存在潜在的数据冒险；访存阶段到执行阶段的前递（MEM to EX），由 A_MEM_Forward 和 B_MEM_Forward 来控制，rs1use_ID 或 rs2use_ID 为真，表示当前指令的源操作数正在使用，rs1_ID == rd_MEM 或 rs2_ID == rd_MEM，表示当前指令的源操作数寄存器与访存阶段指令的结果寄存器相同，|rs1_ID 或 |rs2_ID 为真，表示源操作数寄存器编号非零，hazard_optype_MEM==HAZARD_MEM | hazard_optype_MEM == HAZARD_EX，表示访存阶段的指令存在潜在的数据冒险，且冒险类型为内存或执行，!A_EX_Forward 或 !B_EX_Forward，确保不会与执行阶段到执行阶段的前递冲突；写回阶段到访存阶段的前递（ST to MEM），由 ST_Forward 来控制，条件为：hazard_optype_EX == HAZARD_ST，表示执行阶段的指令是一条存储指令，hazard_optype_MEM == HAZARD_MEM，表示访存阶段的指令需要数据。rs2_EXE == rd_MEM，表示执行阶段的存储指令的源操作数寄存器与访存阶段指令的结果寄存器相同，|rs2_ID 为真，表示存储指令的源操作数寄存器编号非零。

相关代码：

HazardDetection.v

```
        wire A_EX_Forward, B_EX_Forward;
        wire A_MEM_Forward, B_MEM_Forward;

        assign A_EX_Forward = rs1use_ID & (rs1_ID == rd_EXE) & (|rs1_ID) & (hazard_optype_EX == HAZARD_EX);
        assign B_EX_Forward = rs2use_ID & (rs2_ID == rd_EXE) & (|rs2_ID) & (hazard_optype_EX == HAZARD_EX);

        assign A_MEM_Forward = rs1use_ID & (rs1_ID == rd_MEM) & (|rs1_ID) & (hazard_optype_MEM == HAZARD_MEM | hazard_optype_MEM == HAZARD_EX)&(!A_EX_Forward);
```

```
8.        assign B_MEM_Forward = rs2use_ID & (rs2_ID == rd_MEM) & (|rs2_ID) & (hazard_optype_MEM == HAZA
    RD_MEM | hazard_optype_MEM == HAZARD_EX)&(!B_EX_Forward);;
9.
10.       assign forward_ctrl_A = ({2{A_EX_Forward}} & 2'b01) |
11.                               ({2{A_MEM_Forward & (hazard_optype_MEM == HAZARD_EX)}} & 2'b10) |
12.                               ({2{A_MEM_Forward & (hazard_optype_MEM == HAZARD_MEM)}} & 2'b11);
13.
14.       assign forward_ctrl_B = ({2{B_EX_Forward}} & 2'b01) |
15.                               ({2{B_MEM_Forward & (hazard_optype_MEM == HAZARD_EX)}} & 2'b10) |
16.                               ({2{B_MEM_Forward & (hazard_optype_MEM == HAZARD_MEM)}} & 2'b11);
17.
18.       wire ST_Forward;
19.       assign ST_Forward = (hazard_optype_EX == HAZARD_ST) &
20.                           (hazard_optype_MEM == HAZARD_MEM) &
21.                           (rs2_EXE == rd_MEM) &
22.                           (|rs2_ID);
23.
24.       assign forward_ctrl_ls = ST_Forward;
```

顶层 RV32core 的连线

在顶层 RV32core 的连线中，我主要填写了不同阶段中数据前递多路选择器的输入和输出，以下是原理：

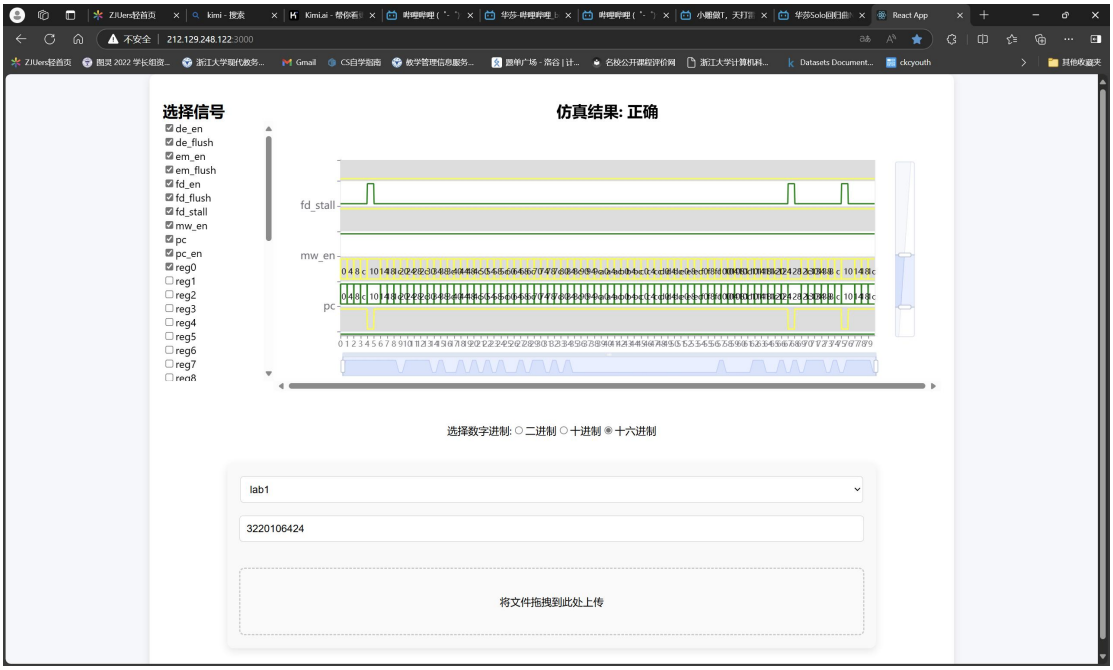
取指阶段：处理器顺序地从程序计数器（PC）中取出下一条指令。

译码阶段：处理器解码指令，包括分支指令。如果分支指令被预测为“not taken”，则更新 PC 以指向当前指令的下一条指令。

执行阶段：执行分支指令。如果实际的分支条件表明需要跳转，那么在执行阶段结束时，处理器需要更正 PC 以指向正确的跳转目标。

访存和写回阶段：如果分支预测错误，可能会导致后续指令错误地访问了内存或使用了错误的寄存器值。在这种情况下，处理器需要采取措施来解决数据冒险，比如暂停流水线，丢弃错误数据，或者重新加载正确的数据。

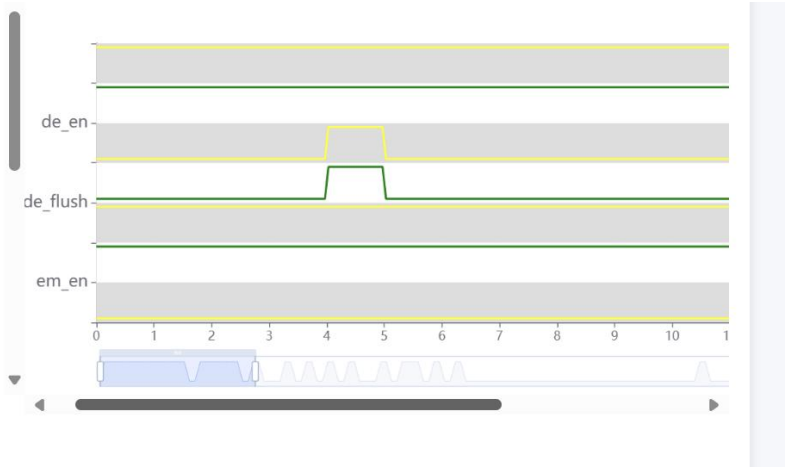
三、 实验过程和数据记录及结果分析

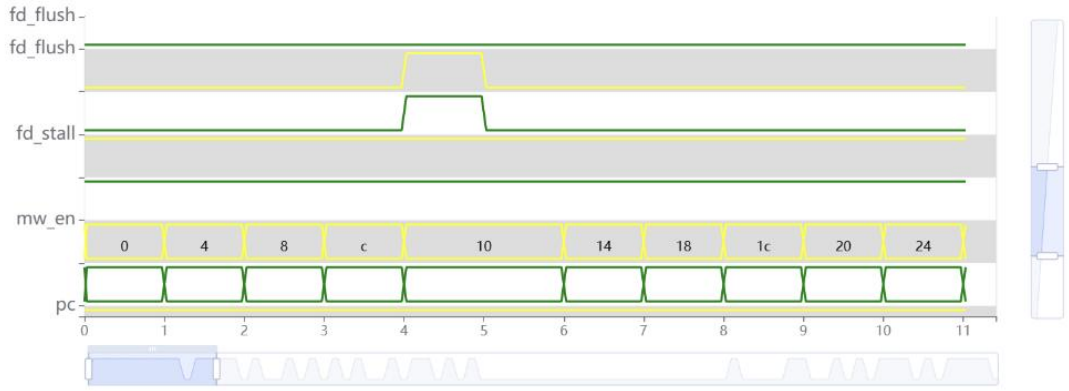


借助平台可以看到在使用助教给的 rom.hex 和 ram.hex 仿真结果正确，以下是代码分析：

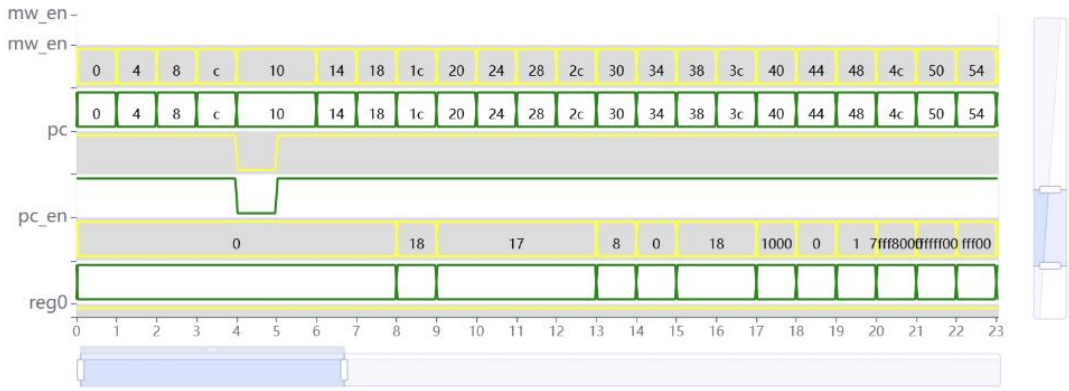
首先把 rom.hex 翻译成机器码

PC	Machine Code	Basic Code	LABEL	ASM	Comment
0x0	0x00000013	addi x0 x0 0	__start:	addi x0, x0, 0	
0x4	0x00402103	lw x2 4(x0)		lw x2, 4(x0)	
0x8	0x00802203	lw x4 8(x0)		lw x4, 8(x0)	
0xc	0x004100B3	add x1 x2 x4		add x1, x2, x4	LR
0x10	0xFFF08093	addi x1 x1 -1		addi x1, x1, -1	
0x14	0x00C02283	lw x5 12(x0)		lw x5, 12(x0)	
0x18	0x01002303	lw x6 16(x0)		lw x6, 16(x0)	
0x1c	0x01402383	lw x7 20(x0)		lw x7, 20(x0)	
0x20	0x402200B3	sub x1 x4 x2		sub x1,x4,x2	
0x24	0x002270B3	and x1 x4 x2		and x1,x4,x2	
0x28	0x002260B3	or x1 x4 x2		or x1.x4.x2	





选择数字进制: ○二进制 ○十进制 ◉十六进制



LR: 可以看到在第四周期，同时前一指令 EXE 的结果立即 forward 到下一条指令的 EXE 操作数，前一指令从 MEM 读出的结果 forward 到下一条指令的 MEM 写数据两种情况

由于 lw x4, 8(x0) 和 add x1, x2, x4 产生冲突，发生 load—use stall，因而 ID 和 IF 指令暂停，而 lw x2, 4(x0) 在 MEM 阶段，产生数据冒险，deflush，fd_stall 上升。

0x20	0x402200B3	sub x1 x4 x2		sub x1,x4,x2	
0x24	0x002270B3	and x1 x4 x2		and x1,x4,x2	
0x28	0x002260B3	or x1 x4 x2		or x1,x4,x2	
0x2c	0x002240B3	xor x1 x4 x2		xor x1,x4,x2	
0x30	0x002210B3	sll x1 x4 x2		sll x1,x4,x2	
0x34	0x002220B3	slt x1 x4 x2		slt x1,x4,x2	
0x38	0x004120B3	slt x1 x2 x4		slt x1,x2,x4	
0x3c	0x001350B3	srl x1 x6 x1		srl x1, x6, x1	RRR
0x40	0x402350B3	sra x1 x6 x2		sra x1, x6, x2	
0x44	0x4023D0B3	sra x1 x7 x2		sra x1, x7, x2	

仿真结果: 正确

RRR

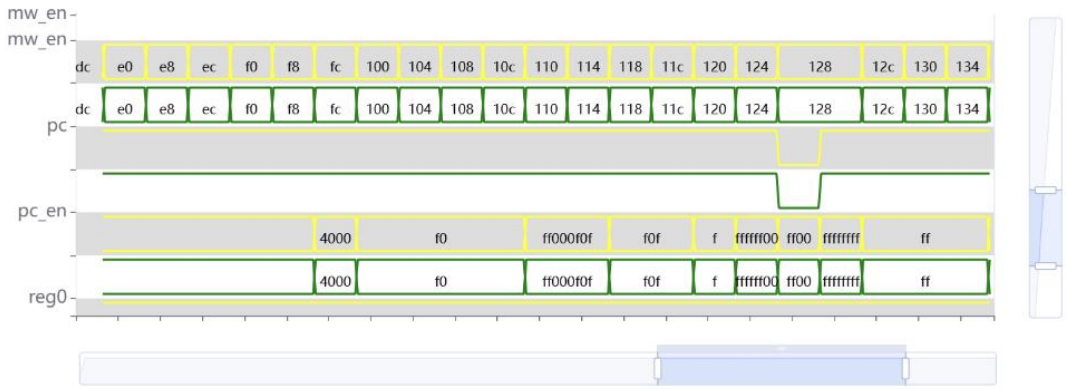
0x74	0xFFF33093	sltiu x1 x6 -1		sltiu x1, x6, -1	
0x78	0xFFF3B093	sltiu x1 x7 -1		sltiu x1, x7, -1	
0x7c	0x00508863	beq x1 x5 16		beq x1,x5,label0	RRB
0x80	0x00420663	beq x4 x4 12		beq x4,x4,label0	
0x84	0x00000013	addi x0 x0 0		addi x0,x0,0	
0x88	0x00000013	addi x0 x0 0		addi x0,x0,0	
0x8c	0x00421863	bne x4 x4 16	label0:	bne x4,x4,label1	
0x90	0x00521663	bne x4 x5 12		bne x4,x5,label1	
0x94	0x00000013	addi x0 x0 0		addi x0,x0,0	

RRB

由于在 ID 阶段已经判断 beq x4 , x4 , label0 需要跳转，fd_flush=1；IF 阶段的取指被抛弃，改用跳转目标取值，即 8c，后续相同

0xe4	0x00000013	addi x0 x0 0		addi x0,x0,0	
0xe8	0x000040B7	lui x1 4	label6:	lui x1,4	
0xec	0x00C000EF	jal x1 12		jal x1,12	
0xf0	0x00000013	addi x0 x0 0		addi x0,x0,0	
0xf4	0x00000013	addi x0 x0 0		addi x0,x0,0	
0xf8	0x01802403	lw x8 24(x0)		lw x8, 24(x0)	LS
0xfc	0x00802E23	sw x8 28(x0)		sw x8, 28(x0)	
0x100	0x01C02083	lw x1 28(x0)		lw x1, 28(x0)	
0x104	0x02801023	sh x8 32(x0)		sh x8, 32(x0)	
0x108	0x02002083	lw x1 32(x0)		lw x1, 32(x0)	

仿真结果: 正确

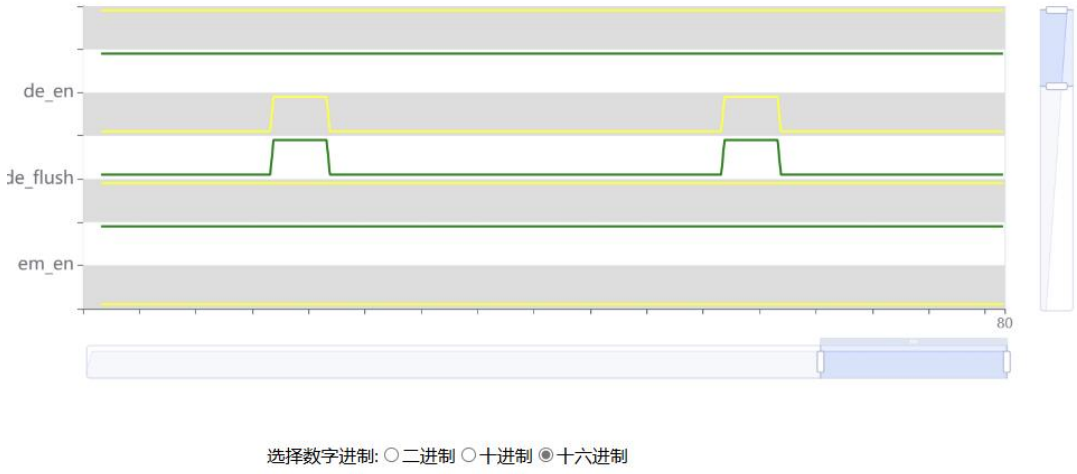


LS:

由于 sw x8，x28(x0)在 EX 阶段，lw x8，24(x0)在 MEM 阶段，前一指令从 MEM 读出的结果 forward 到下两条指令的 EXE 操
作数，将 Load 的结果立即 store。

0x110	0x02402083	lw x1 36(x0)		lw x1, 36(x0)	
0x114	0x01A01083	lh x1 26(x0)		lh x1, 26(x0)	
0x118	0x01A05083	lhu x1 26(x0)		lhu x1, 26(x0)	
0x11c	0x01B00083	lb x1 27(x0)		lb x1, 27(x0)	
0x120	0x01B04083	lbu x1 27(x0)		lbu x1, 27(x0)	
0x124	0xFC5082E3	beq x1 x5 -60		beq x1,x5,label6	LLB
0x128	0xFFFF0097	auipc x1, 0xffff0		auipc x1, 0xffff0	
0x12c	0x00400093	addi x1, x0, 4		addi x1, x0, 4	
0x130	0x00800093	addi x1, x0, 8		addi x1, x0, 8	
0x134	0x000080E7	jalr x1, 0(x1)		jalr x1, 0(x1)	RRB
0x138	0x00000013	addi x0 x0 0		addi x0,x0,0	

仿真结果: 正确

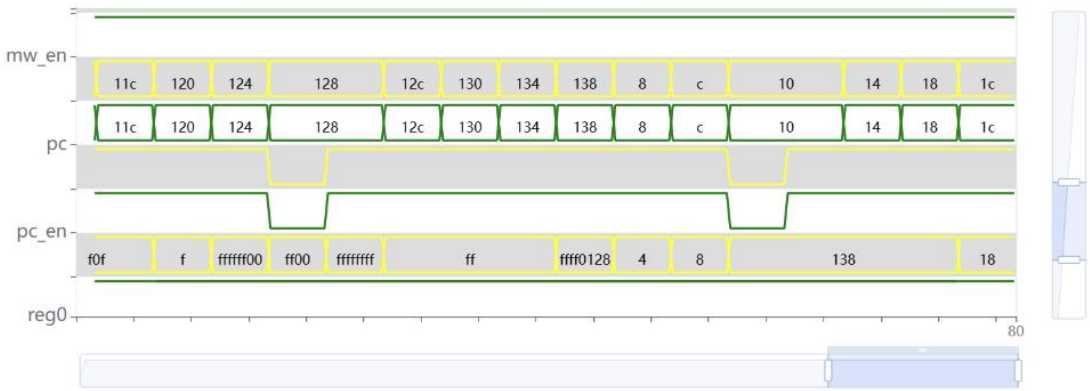


选择数字进制: ○二进制 ○十进制 ◉十六进制

仿真结果: 正确

选择数字进制: ○二进制 ○十进制 ◉十六进制

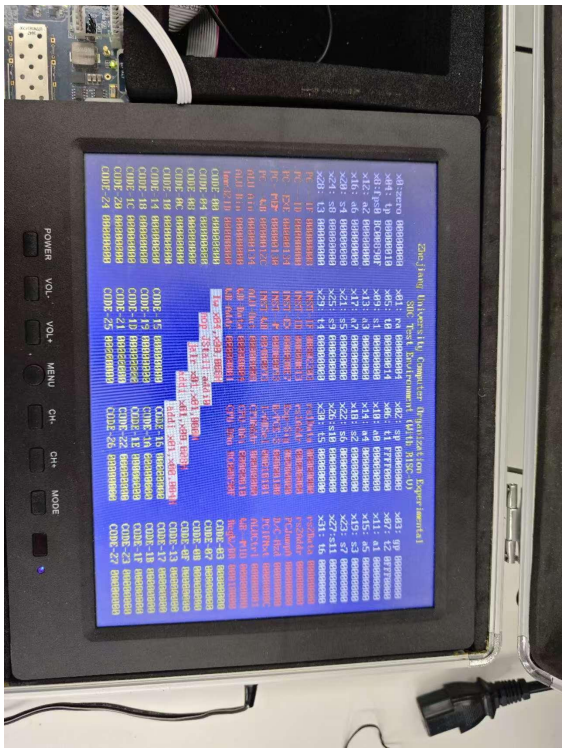
仿真结果: 正确



LLB

RRB:
JALR 在 EX 阶段被执行跳转到 0x08，前一指令 EXE 的结果在 MEM 级才 forward 到下两条指令的 EXE 操作数。

两张上板照片



四、　讨论与心得

本次实验主要是对上学期内容的复习，但是由于我之前计组学的不是很好，花费了约莫 4-5 天完成这次实验，花了两天 de 了 reg0 在第 20 周期 forward 的问题，最终在助教哥哥的提示下完成了本次实验，希望下次实验可以顺利完成。