

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 计算机体系结构 |
| 姓 名： | 汪辉 |
| 学 院： | 计算机科学与技术学院 |
| 系： | 计算机系 |
| 专 业： | 计算机科学与技术 |
| 学 号： | 3190105609 |
| 指导教师： | 陈文智 |

2021年12月21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： Pipelined CPU supporting multi-cycle operations

学生姓名： 王嘉豪 专业： 计算机科学与技术 学号： 3190105304

同组学生姓名： 汪辉 指导老师： 陈文智

实验地点： 曹西301 实验日期： 2021 年 12 月 21日

1. 实验目的和要求

- 了解支持多周期计算的流水线原理

- 掌握支持多周期计算的流水线的设计方法

- 掌握支持多周期计算的流水线的验证方法

1. 实验内容和原理
2. 实验内容：

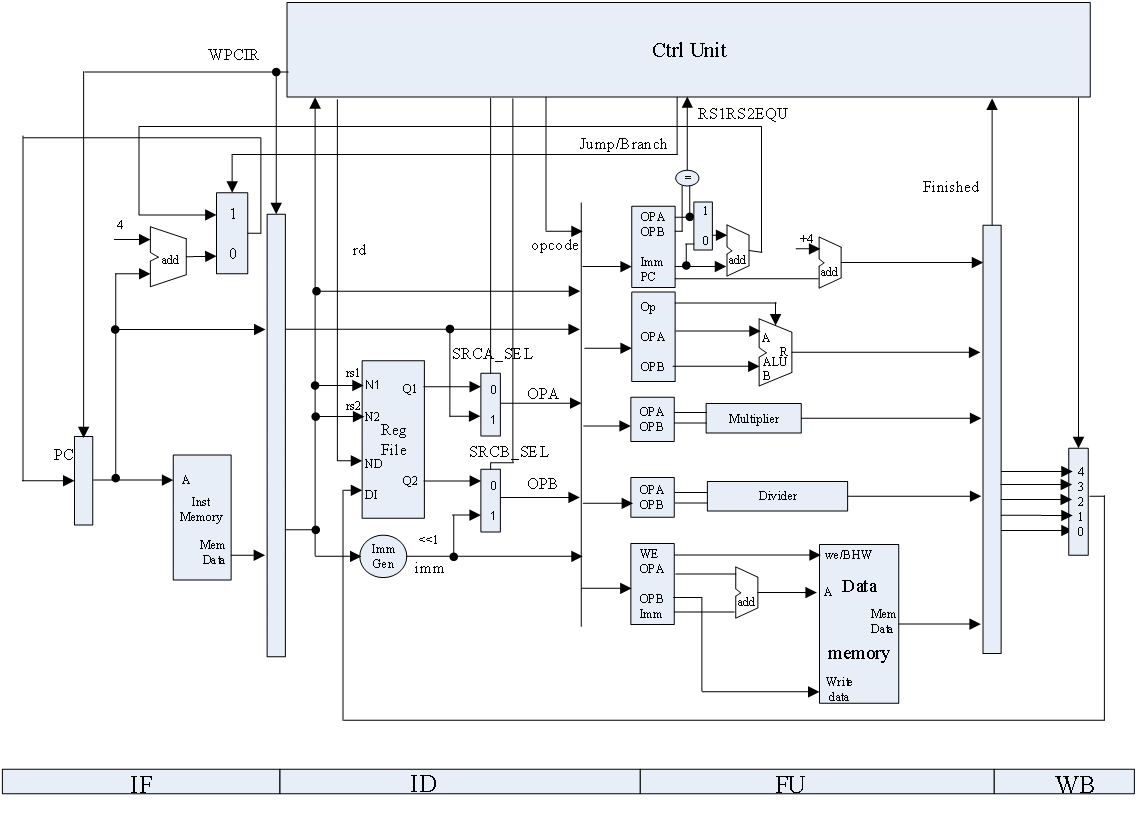
- 利用IF/ID/FU/WB模块和FU模块重新设计支持多周期计算的流水线

- 重新设计CPU的控制模块

- 通过程序验证流水线CPU并且观察程序的计算过程

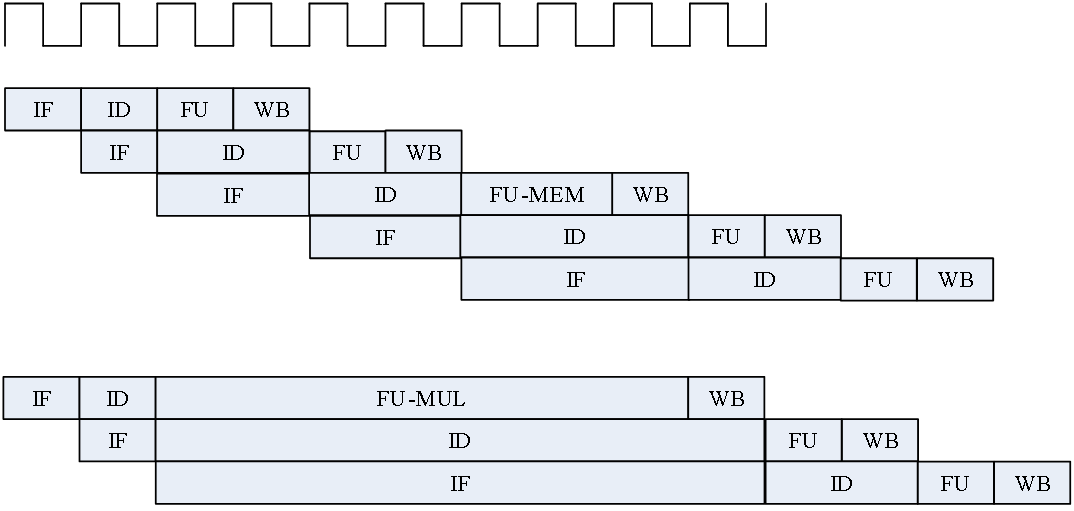
1. 实验原理：

2.1 Pipelined CPU



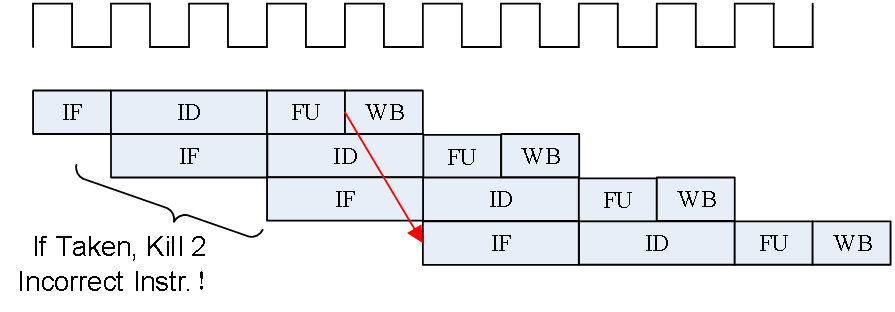
将不同的计算方式分开分别设计模块，当需要计算时，将需求发送给对应的模块。模块被占用期间，相同类型计算指令可以被阻塞。最后的结果通过多路选择器一个个输出。

2.2 Pipelines resolving Data Hazards



遇到数据竞争时，插入stall等到竞争不再存在，再继续运行。

4.3 Methods of resolving Control hazards



是否跳转和地址的计算在FU里面进行，采取预测不跳转的方式，假如最后需要跳转，那么就需要清理掉两条已经读入的错误指令。

1. 实验过程和数据记录

本实验主要是需要补充FU\_ALU.v，FU\_div.v，FU\_jump.v，FU\_mem.v，FU\_mul.v以及RV32core.v，根据相关文件功能，依次进行补充即可。

1. FU\_ALU

当state=0时，计算还未完成，分别将两个操作数ALUA、ALUB以及操作符ALUControl的值赋给A，B以及Control，并将state置1，标志下个周期完成计算。

|  |
| --- |
| **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** |

1. FU\_mul

乘法器的补充部分需要控制乘法执行七个周期，根据给出的state[6:0]寄存器，补充代码如下。每过一个周期，状态寄存器发生改变。这里直接使用移位操作符，避免较多的选择分支。

|  |
| --- |
| **wire[31:0] mulres;**  **always@(posedge clk) begin**  **if(EN & state == 0) begin // state == 0**  **A\_reg <= A;**  **B\_reg <= B;**  **state[6] <= 1;**  **end**  **else state <= (state >> 1);**  **end** |

1. FU\_div

当除法器被调用且处于未完成状态时，持续给除法模块输入除数被除数以及将它们的valid置为1，当计算完成时，res\_valid为1，此时将除数被除数的valid置为0。

|  |
| --- |
| **always@(posedge clk) begin**  **if(EN & ~state & ~res\_valid) begin // state == 0**  **A\_reg <= A;**  **A\_valid <= 1;**  **B\_reg <= B;**  **B\_valid <= 1;**  **state <= 1;**  **end**  **else if(res\_valid) begin**  **A\_valid <= 0;**  **B\_valid <= 0;**  **state <= 0;**  **end**  **end** |

1. FU\_mem

在第一个周期准备好内存操作需要的参数，经过两个周期进行内存访问以及数据的存取。

|  |
| --- |
| **reg[31:0] addr ;**  **always@(posedge clk) begin**  **if( EN && state==0 ) begin // state == 0**  **mem\_w\_reg <= mem\_w ;**  **bhw\_reg <= bhw ;**  **rs1\_data\_reg <= rs1\_data ;**  **rs2\_data\_reg <= rs2\_data ;**  **imm\_reg <= imm ;**  **addr <= rs1\_data+imm ;**  **state <= 2;**  **end**  **else if ( state == 2 ) begin**  **state <= 1;**  **end**  **else if ( state == 1 ) state <= 0 ;**  **end** |

1. FU\_jump

通过给cmp模块传入比较的值和比较方式，得出是否需要进行跳转。根据跳转指令类型，mux模块选择下一周期的PC值。

|  |
| --- |
| **reg[31:0] jalr\_pc, b\_jal\_pc ;**    **wire JAL = cmp\_ctrl\_reg == 3'b000 ;**  **wire BEQ = cmp\_ctrl\_reg == 3'b001 ;**  **wire BNE = cmp\_ctrl\_reg == 3'b010 ;**  **wire BLT = cmp\_ctrl\_reg == 3'b011 ;**  **wire BGE = cmp\_ctrl\_reg == 3'b100 ;**  **wire BLTU = cmp\_ctrl\_reg == 3'b101 ;**  **wire BGEU = cmp\_ctrl\_reg == 3'b110 ;**    **always@(posedge clk) begin**  **if(EN & ~state) begin // state == 0**  **state <= 1;**  **JALR\_reg <= JALR ;**  **cmp\_ctrl\_reg <= cmp\_ctrl ;**  **rs1\_data\_reg <= rs1\_data ;**  **rs2\_data\_reg <= rs2\_data ;**  **if ( JALR ) PC\_jump\_reg <= rs1\_data\_reg+imm ;**  **else PC\_jump\_reg <= PC+imm ;**  **PC\_wb\_reg <= PC+4 ;**  **end**  **else state <= 0;**  **end**    **wire EQUAL = rs1\_data\_reg == rs2\_data\_reg ;**  **wire NEQUAL = ~( rs1\_data\_reg == rs2\_data\_reg ) ;**  **wire LTU = rs1\_data\_reg < rs2\_data\_reg ;**  **wire GEU = rs1\_data\_reg >= rs2\_data\_reg ;**  **wire LT = rs1\_data\_reg[31]&&~rs2\_data\_reg[31] ||**  **rs1\_data\_reg[31]&&rs2\_data\_reg[31]&&rs1\_data\_reg>rs2\_data\_reg ||**  **~rs1\_data\_reg[31]&&~rs2\_data\_reg[31]&&rs1\_data\_reg<rs2\_data\_reg ;**  **wire GE = ~( rs1\_data\_reg[31]&&~rs2\_data\_reg[31] ||**  **rs1\_data\_reg[31]&&rs2\_data\_reg[31]&&rs1\_data\_reg>rs2\_data\_reg ||**  **~rs1\_data\_reg[31]&&~rs2\_data\_reg[31]&&rs1\_data\_reg<rs2\_data\_reg ) ;**    **assign cmp\_res = JAL | JALR\_reg |**  **BEQ & EQUAL |**  **BNE & NEQUAL |**  **BLT & LT |**  **BGE & GE |**  **BLTU & LTU |**  **BGEU & GEU ;**    **assign PC\_jump = PC\_jump\_reg ;**  **assign PC\_wb = PC\_wb\_reg ;** |

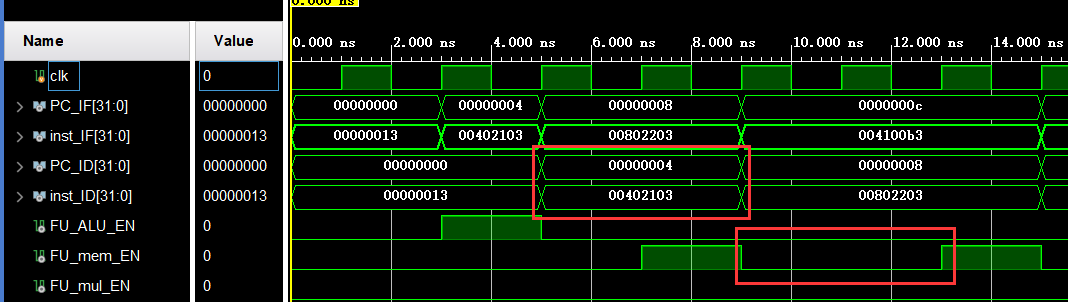
1. RV32core

如下补全即可。

|  |
| --- |
| **ImmGen imm\_gen(.inst\_field(inst\_ID),.ImmSel(ImmSel\_ctrl),.Imm\_out(Imm\_out\_ID));**  **MUX2T1\_32 mux\_imm\_ALU\_ID\_A(.I0(rs1\_data\_ID),.I1(Imm\_out\_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),.I1(ALUout\_WB),.I2(mem\_data\_WB),.I3(mulres\_WB),.I4(divres\_WB),.I5(PC\_wb\_WB),.o(wt\_data\_WB));** |

1. 实验结果分析
2. 仿真结果：

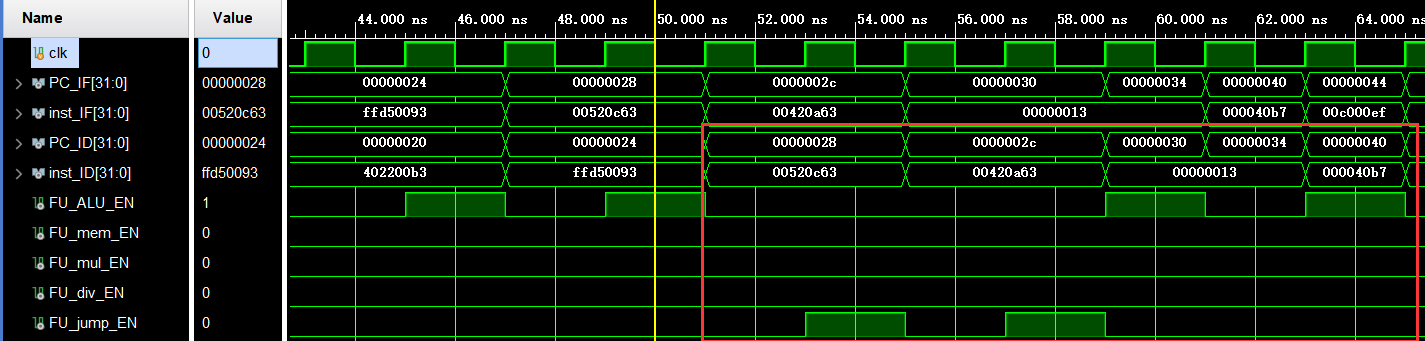
FU\_mem：当ID阶段译码为数据的存取时，FU\_mem\_EN被置为1，之后使用两个周期进行相关操作，再进行下一条指令的执行。



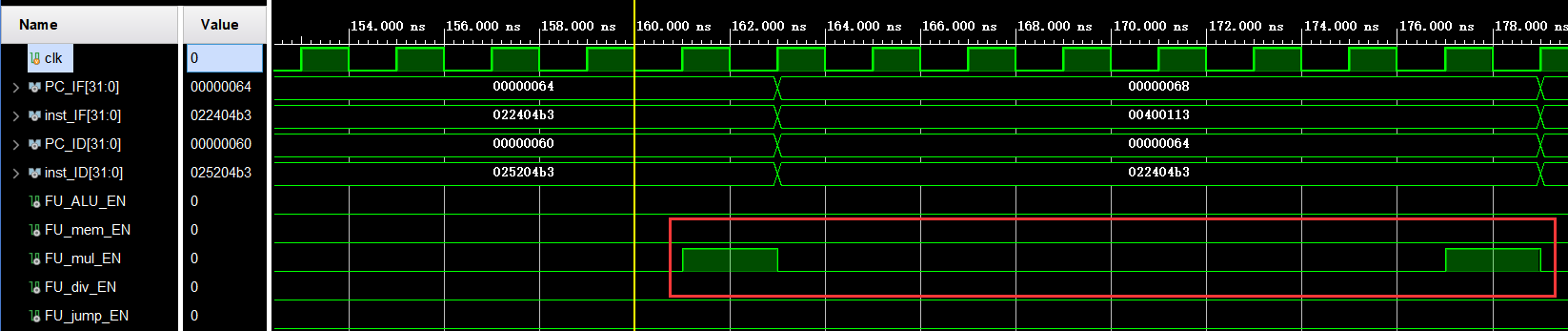
FU\_ALU：当ID阶段译码为ALU进行数据运算时，FU\_ALU\_EN被置为1，之后使用一个周期进行相关操作，再进行下一条指令的执行。



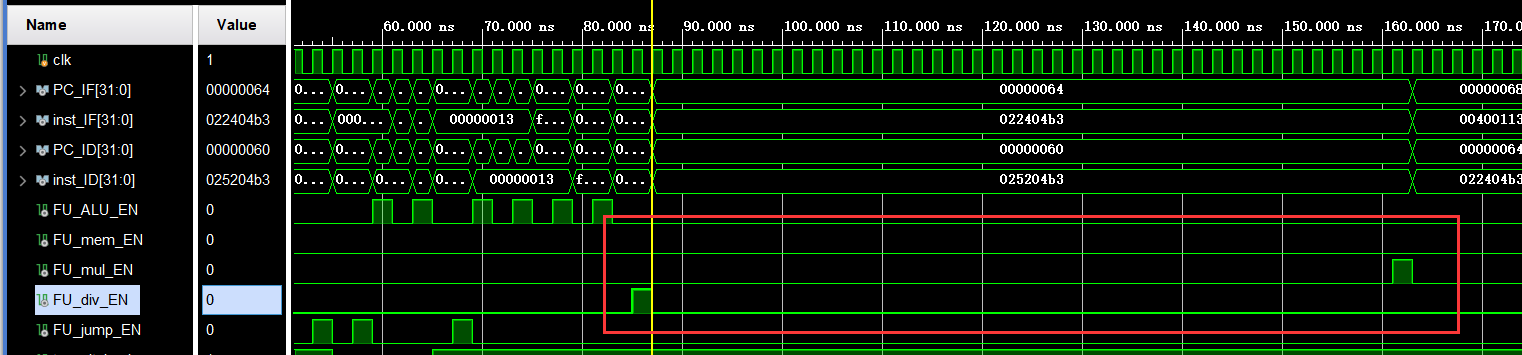
FU\_jump：当ID阶段译码为跳转相关时，FU\_jump\_EN被置为1，之后使用一个周期进行相关操作，再进行下一条指令的执行。



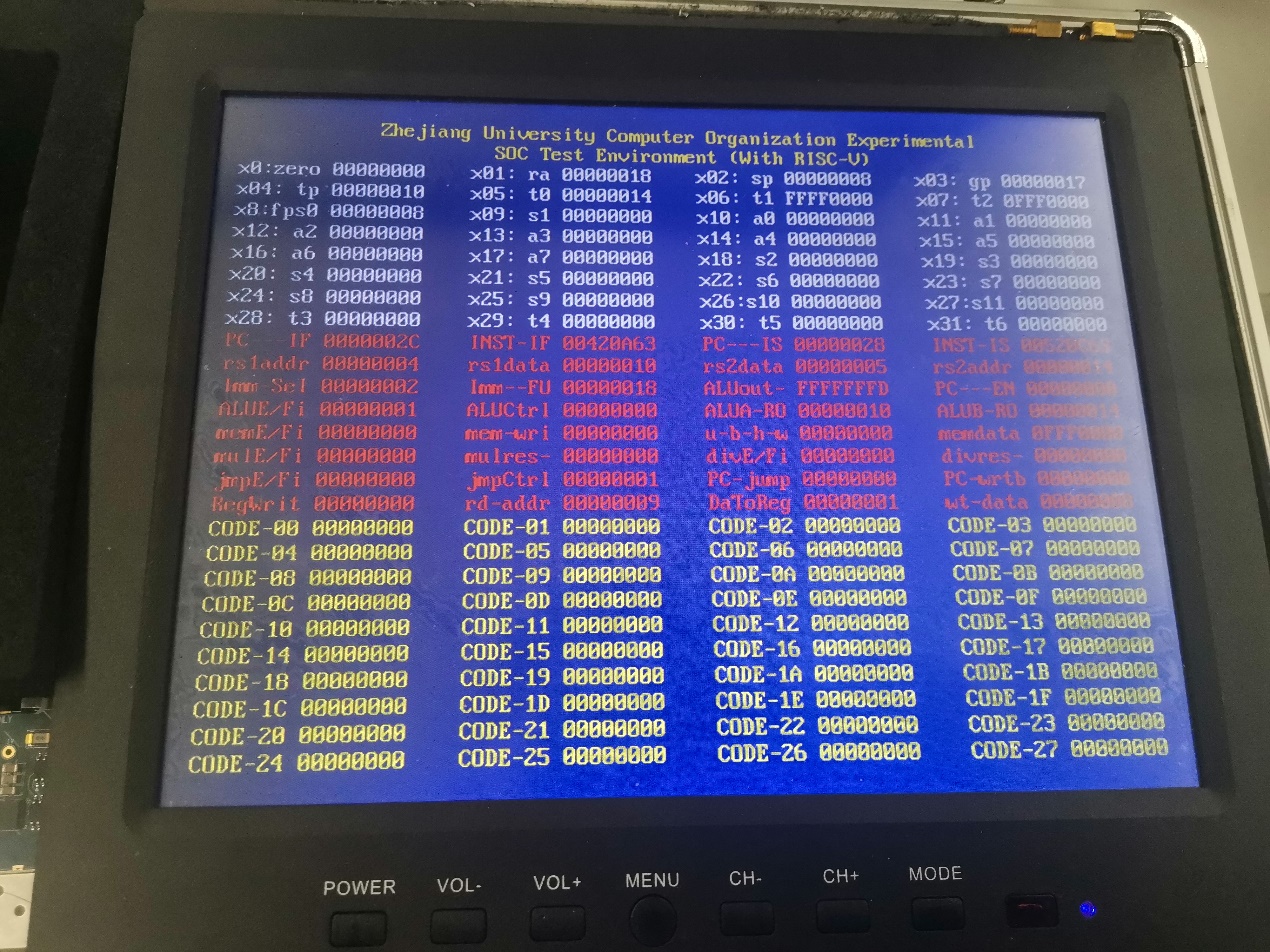
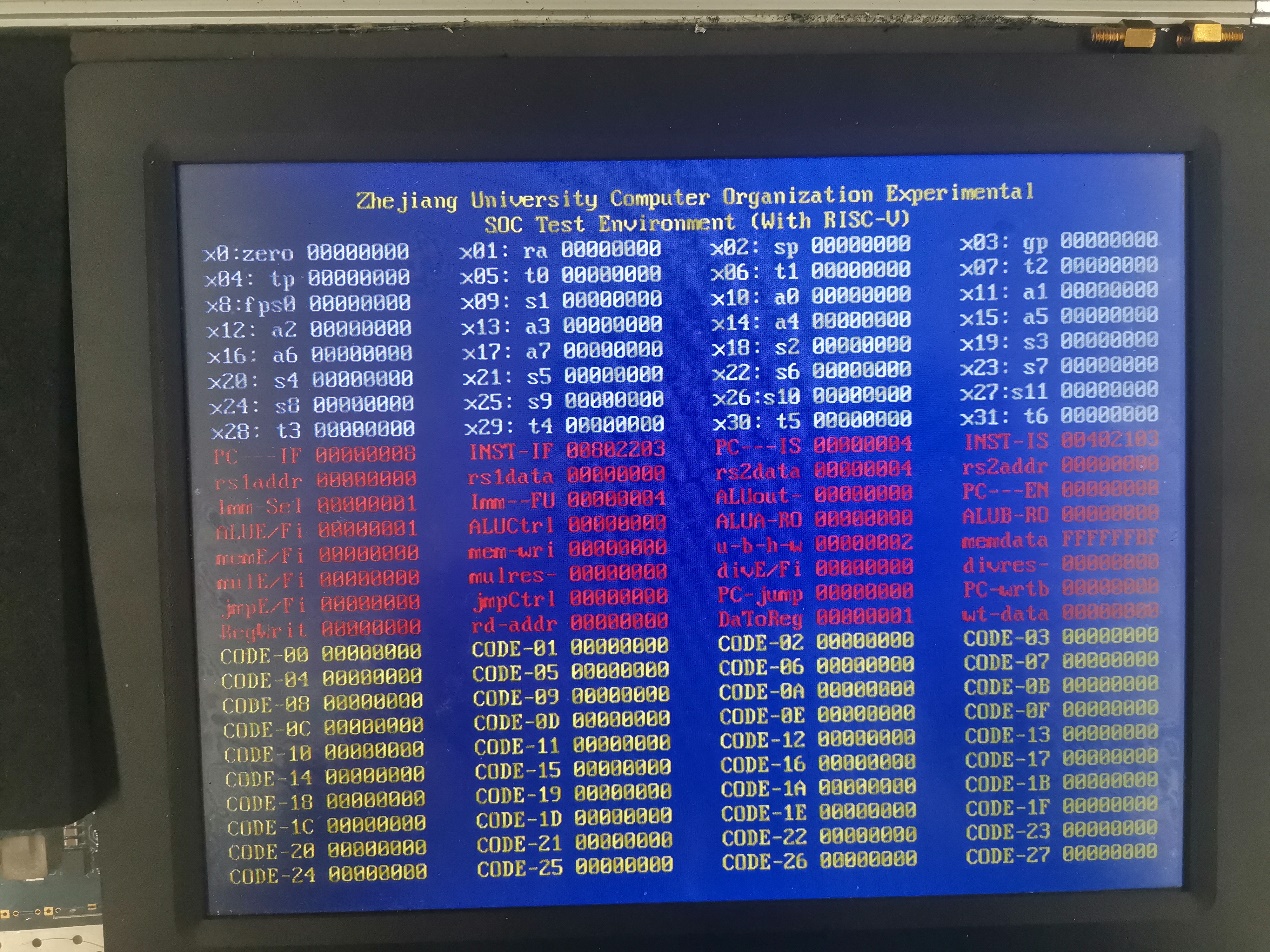
FU\_mul：当ID阶段译码为乘法运算时，FU\_mul\_EN被置为1，之后使用六个周期进行相关操作，再进行下一条指令的执行。



FU\_div：当ID阶段译码为除法运算时，FU\_div\_EN被置为1，之后使用若干个周期进行相关操作，再进行下一条指令的执行。除法器运算的周期数会随输入变化。



实验箱结果：



1. 讨论与心得

本次实验是在理解了多周期计算的流水线原理的基础上，设计多周期计算的流水线并对其进行验证。实验中已经提供了大部分的代码，只需要对FU相关的单元以及RV32Core进行填充，实验难度不高。实验主要是要求对于功能所需周期进行判断，然后使用state进行控制。在进行仿真验证时，需要比较小心注意每一个操作对应的波形，不然可能就会很难发现问题。RV32Core的补充也一定要注意，不然可能会存在上板之后出现问题的可能。