

**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： **Dynamically Scheduled Pipelines using Scoreboarding**

学生姓名： 汪辉 专业： 计算机科学与技术 学号： 3190105609

同组学生姓名： 王嘉豪 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

- 理解支持多周期指令的流水线设计方法

- 理解scoreboard动态调度算法的设计原则

- 掌握支持多周期计算的流水线的设计方法

- 掌握支持scoreboard动态调度算法的流水线设计

- 理解掌握scoreboard算法的验证原则

1. 实验内容和原理
2. 实验内容：

* **Redesign the pipelines with IF/IS/RO/FU/WB stages and supporting multicycle operations.**
* **Design of a scoreboard and integrate it to CPU.**
* **Verify the Pipelined CPU with program and observe the execution of program.**

1. 实验原理：

2.1 Pipelined CPU



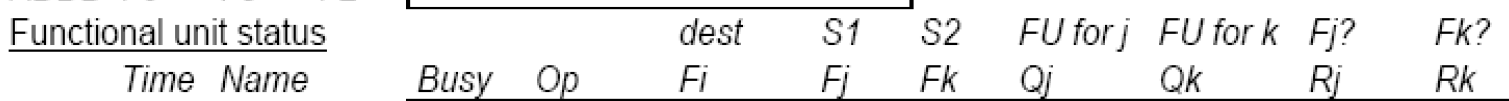
多周期的运算模块在实验五中设计完成，实验六中任务主要为在ctrlunit中实现scoreboard的动态调度。其中，为了满足调度的选择，需要实现和管理下列结构：Functional Unit Status、Register Result Status。

2.2 Functional Unit Status

FUS是5个32位数据的数组，



存储了当前JUMP、ALU、MEM、MUL、DIV五个模块的状态。其中32位各个位的含义在宏中定义，参考下图：



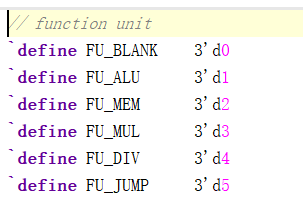
实验工程中的32位末位是一个代表unit的当前指令是否完成的位FU\_DONE。

2.3 Register Result Status

RRS存储着当前32个寄存器的数据来源，若是可立即获得的则RRS为0，否则为对应的功能单元，



数组的每一项有3位，根据宏的内容，只用到了6个数据。



1. 实验过程和数据记录

实验六补充完成CtrlUnit.v中的各部分信号处理和组合逻辑，根据scoreboard的算法步骤和功能，依次进行补充即可。

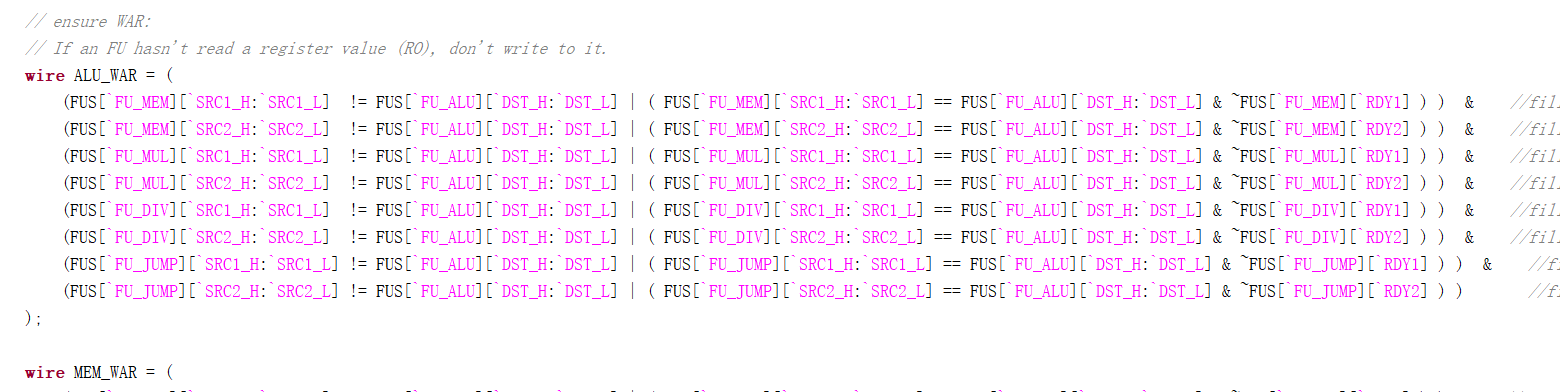
3.1 normal stall

normal\_stall是全流水线stall，目的是等待特定单元完成计算以排除结构竞争和WAW冒险。

|  |
| --- |
| **// normal stall: structural hazard or WAW**  **assign normal\_stall = {**  **// structural hazard**  **FUS[`FU\_ALU][`BUSY]&use\_ALU |**  **FUS[`FU\_MEM][`BUSY]&use\_MEM |**  **FUS[`FU\_MUL][`BUSY]&use\_MUL |**  **FUS[`FU\_DIV][`BUSY]&use\_DIV |**  **FUS[`FU\_JUMP][`BUSY]&use\_JUMP |**  **// WAW**  **( RRS[dst] != `FU\_BLANK )**  **}; //fill sth. here** |

3.2 ensure WAR

WAR冒险的处理有一些复杂，实验代码中，各个单元的WAR信号用于判断是否存在WAR冒险，在使用各个WAR时需要注意当WAR为1时表示不存在WAR冒险可以正常写，由于这和第一反应不同，在实现这个信号处理时更需特别注意，比如信号中的与和或的关系。



3.3 Issue

满足发射条件时，进行FUS的初始化，并填写相应的RRS：

|  |
| --- |
| **if (RO\_en) begin**  **// not busy, no WAW, write info to FUS and RRS**  **if (|dst) RRS[dst] <= use\_FU;**  **FUS[use\_FU][`BUSY] <= 1'b1;**  **FUS[use\_FU][`OP\_H:`OP\_L] <= op ;**  **FUS[use\_FU][`DST\_H:`DST\_L] <= dst ;**  **FUS[use\_FU][`SRC1\_H:`SRC1\_L] <= src1 ;**  **FUS[use\_FU][`SRC2\_H:`SRC2\_L] <= src2 ;**  **FUS[use\_FU][`FU1\_H:`FU1\_L] <= fu1 ;**  **FUS[use\_FU][`FU2\_H:`FU2\_L] <= fu2 ;**  **FUS[use\_FU][`RDY1] <= rdy1 ;**  **FUS[use\_FU][`RDY2] <= rdy2 ;**  **FUS[use\_FU][`FU\_DONE] <= 1'b0 ;**  **//fill sth. here.**  **IMM[use\_FU] <= imm;**  **PCR[use\_FU] <= PC;**  **end** |

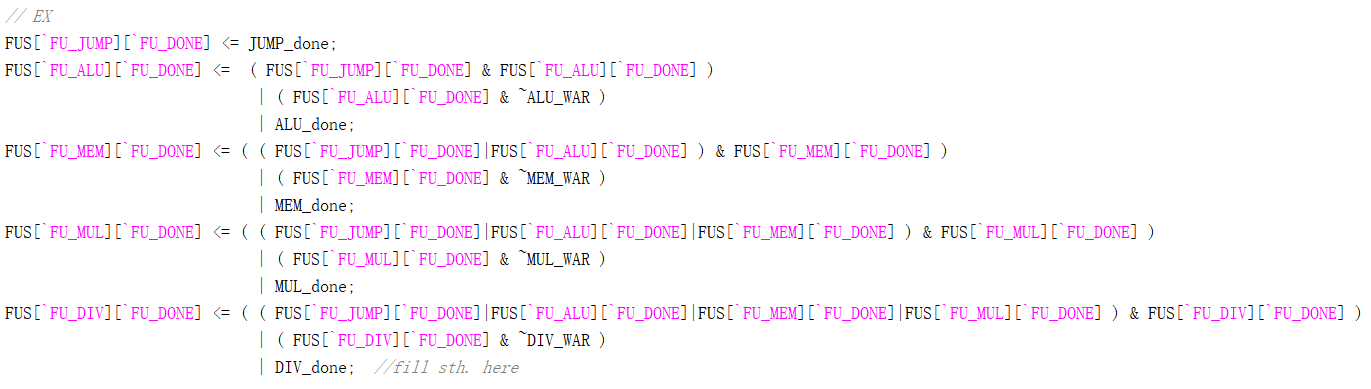
3.3 Read Operand

Issue之后等待操作需要的参数，当其他功能单元完成src的写后，rdy位会被置1，只有当两个源操作数的rdy都为1时，寄存器数据才被正常读取，并且为了确保不产生对后续指令的持续影响，RO阶段只有1个周期，当读取出数据后，立即将rdy立即置0。

|  |
| --- |
| **// RO**  **if (FUS[`FU\_JUMP][`RDY1] & FUS[`FU\_JUMP][`RDY2]) begin**  **// JUMP**  **FUS[`FU\_JUMP][`RDY1] <= 1'b0;**  **FUS[`FU\_JUMP][`RDY2] <= 1'b0;**  **end** |
|  |

3.4 Execute

按照一定先后顺序依次完成5个功能单元的运算，需要注意的是，在每个周期内，功能单元最多有一个完成，实验中设计的顺序如下：JUMP、ALU、MEM、MUL、DIV。



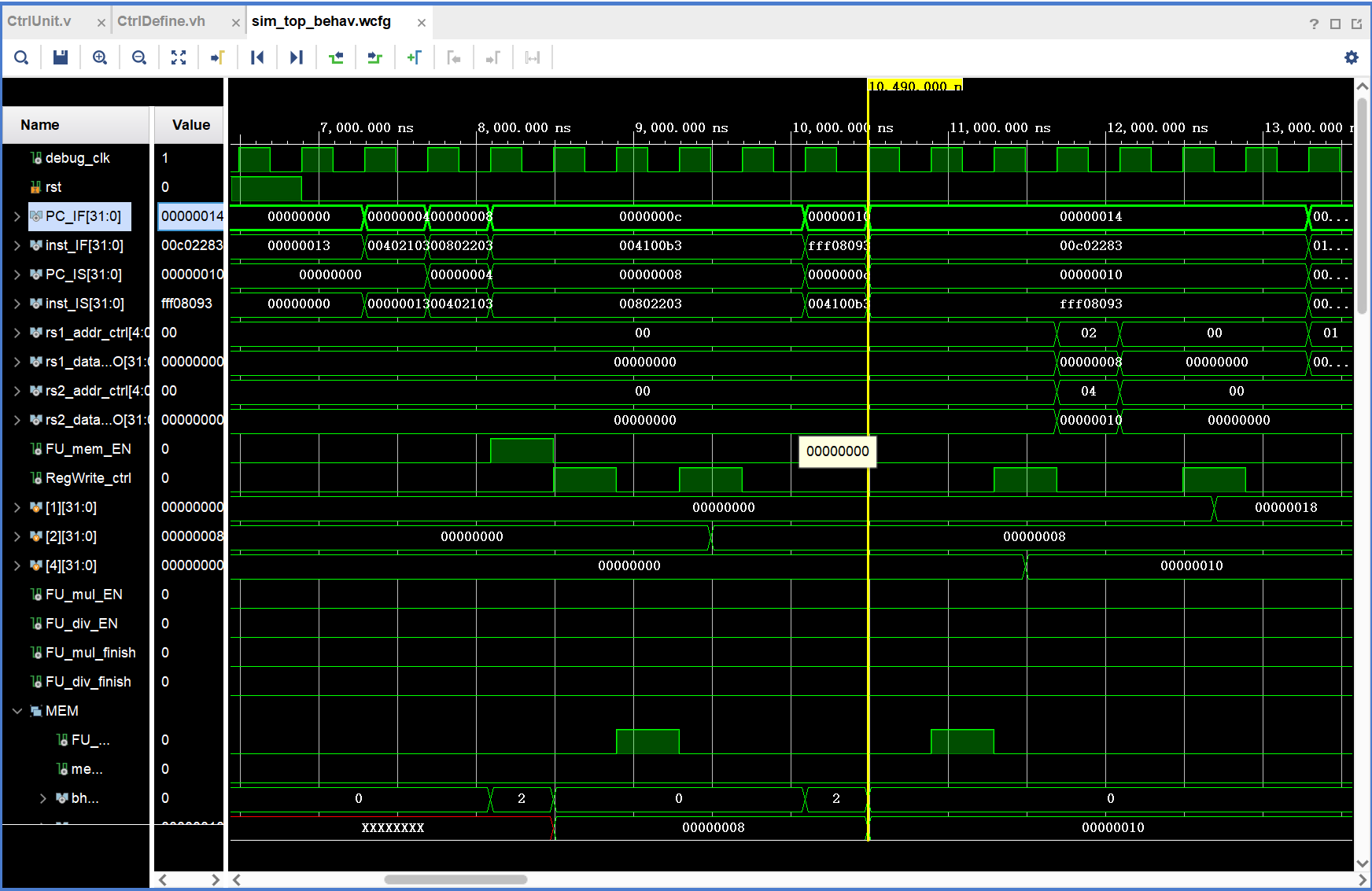
3.5 Write Back

写回数据后，之前被阻塞RO的指令需要正常执行，所以需要把rdy置1，这样下一周期数据即可被正常读取。以JUMP单元为例：

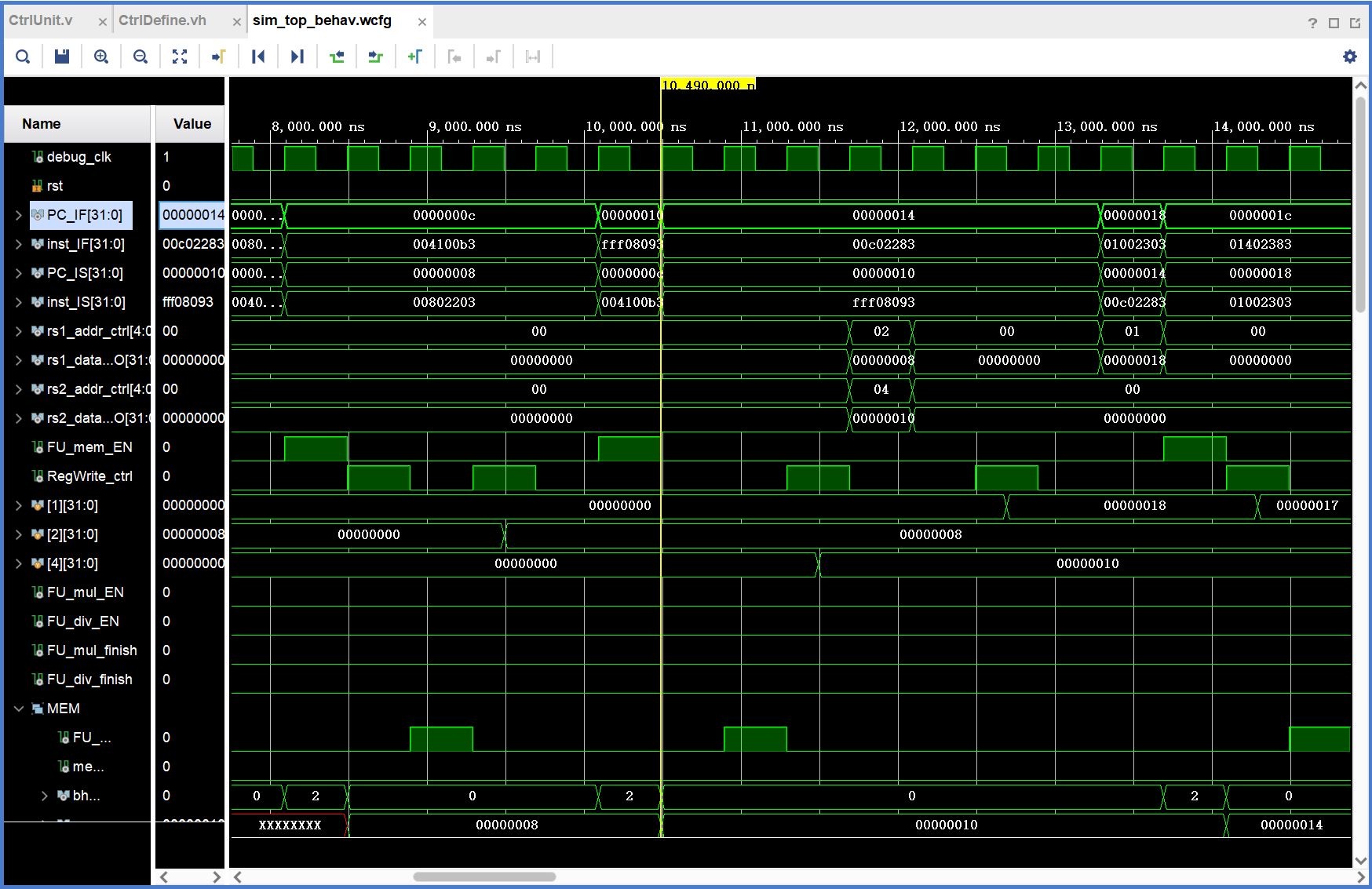
|  |
| --- |
| **/\* WB \*/**  **// JUMP**  **if (FUS[`FU\_JUMP][`FU\_DONE] & JUMP\_WAR) begin**  **RRS[FUS[`FU\_JUMP][`DST\_H:`DST\_L]] <=`FU\_BLANK ;**  **FUS[`FU\_JUMP] <= 32'b0 ;**  **// ensure RAW**  **if (FUS[`FU\_ALU][`FU1\_H:`FU1\_L] == `FU\_JUMP) FUS[`FU\_ALU][`RDY1] <= 1'b1 ; //fill sth. here**  **if (FUS[`FU\_MEM][`FU1\_H:`FU1\_L] == `FU\_JUMP) FUS[`FU\_MEM][`RDY1] <= 1'b1 ; //fill sth. here**  **if (FUS[`FU\_MUL][`FU1\_H:`FU1\_L] == `FU\_JUMP) FUS[`FU\_MUL][`RDY1] <= 1'b1 ; //fill sth. here**  **if (FUS[`FU\_DIV][`FU1\_H:`FU1\_L] == `FU\_JUMP) FUS[`FU\_DIV][`RDY1] <= 1'b1 ; //fill sth. here**  **if (FUS[`FU\_ALU][`FU2\_H:`FU2\_L] == `FU\_JUMP) FUS[`FU\_ALU][`RDY2] <= 1'b1 ; //fill sth. here**  **if (FUS[`FU\_MEM][`FU2\_H:`FU2\_L] == `FU\_JUMP) FUS[`FU\_MEM][`RDY2] <= 1'b1 ; //fill sth. here**  **if (FUS[`FU\_MUL][`FU2\_H:`FU2\_L] == `FU\_JUMP) FUS[`FU\_MUL][`RDY2] <= 1'b1 ; //fill sth. here**  **if (FUS[`FU\_DIV][`FU2\_H:`FU2\_L] == `FU\_JUMP) FUS[`FU\_DIV][`RDY2] <= 1'b1 ; //fill sth. here**  **end** |

1. 实验结果分析
2. 仿真结果：

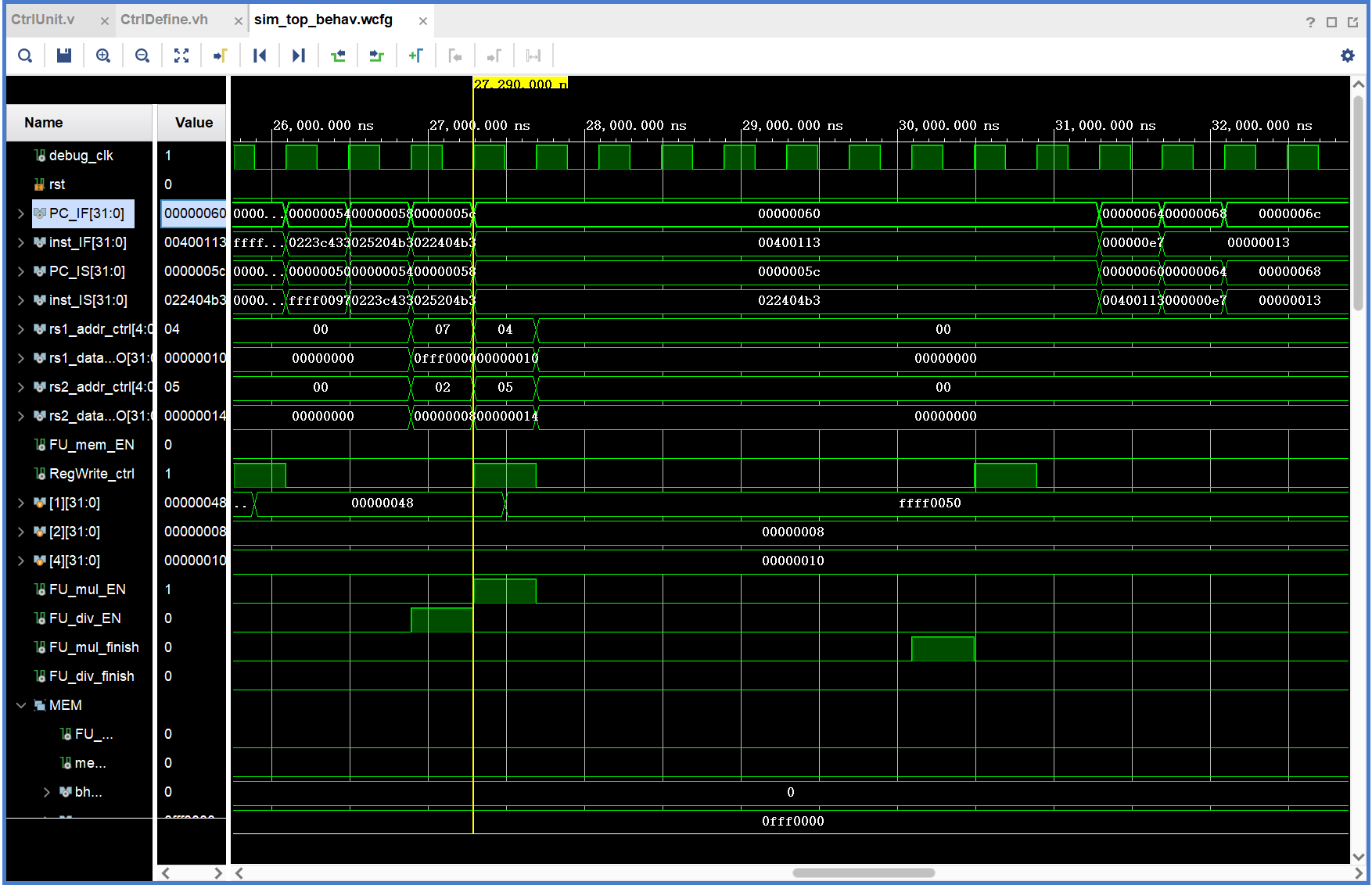
Structural Hazard：Functional Unit被占用时指令不能发射，比如汇编代码中的连续两条lw指令，可以从仿真中看到第二条load指令被阻塞了5个周期。



WAW：当前面的指令还没有完成目的寄存器的写回时，下一条相同目的寄存器的指令也不能发射，下图中指令10处连续的第二条写x1的指令被阻塞。

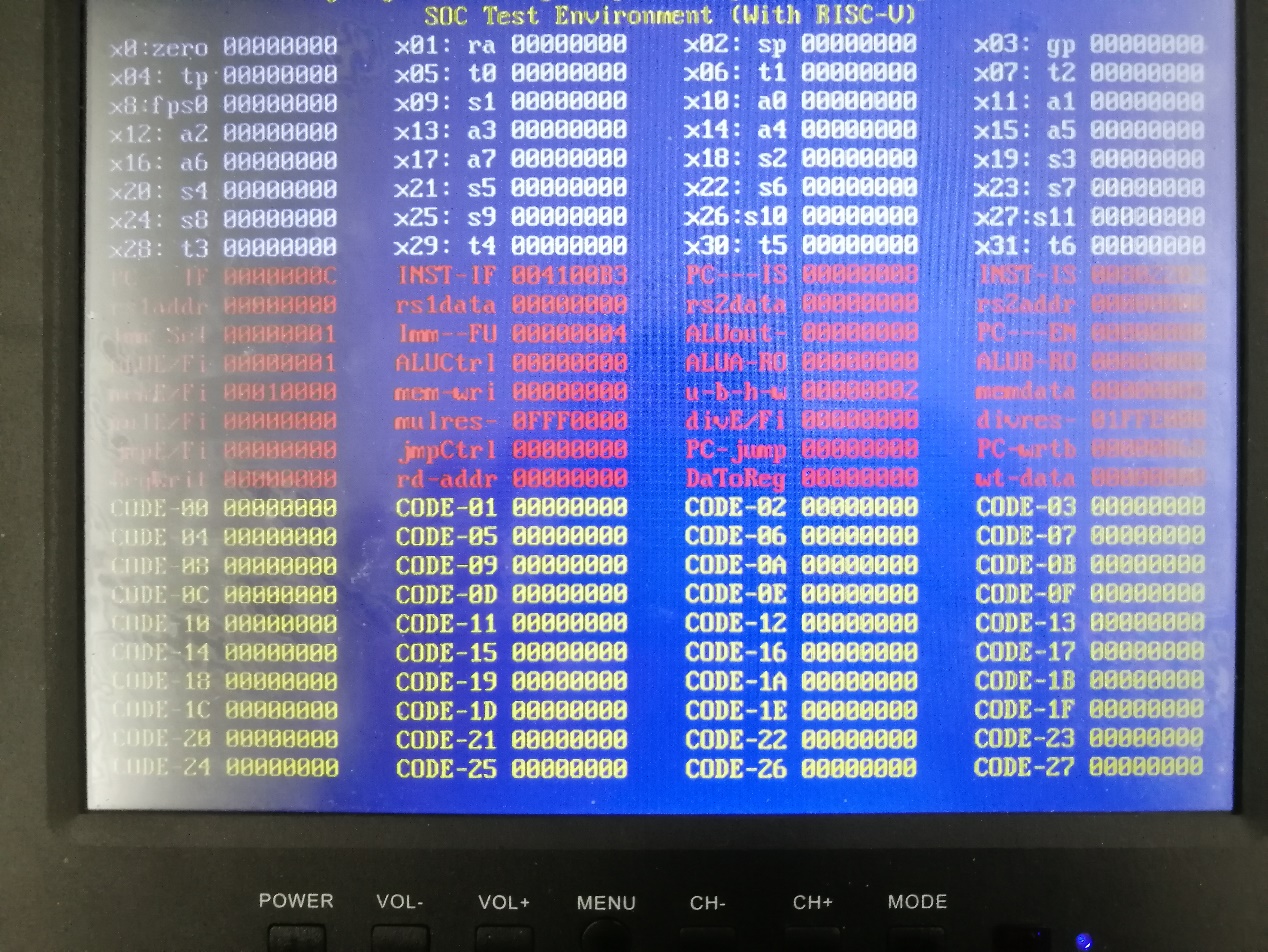


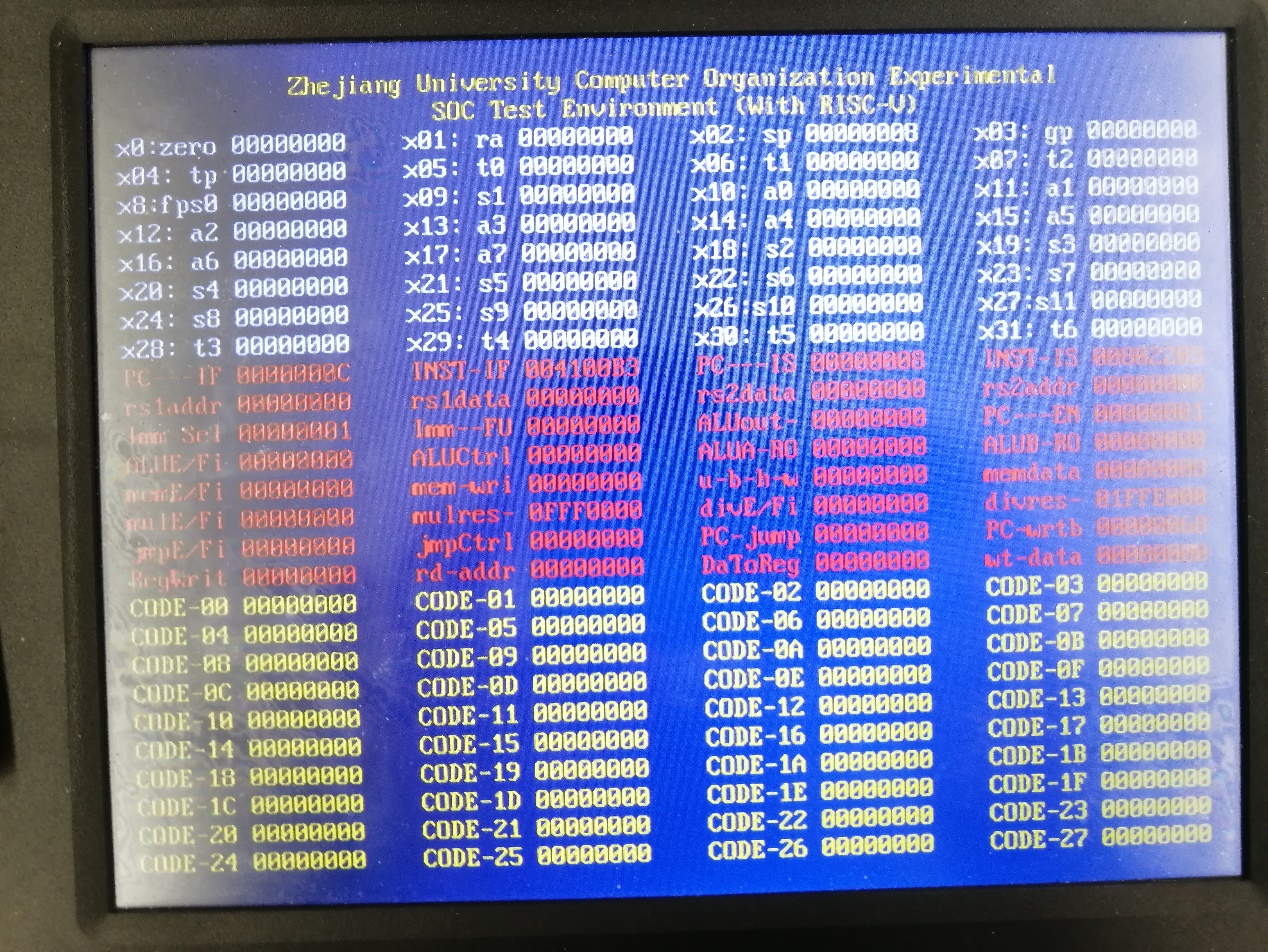
RAW：指令5C处发生RAW冒险，指令被阻塞。同时由于阻塞时长，源操作数已经顺利读取，下一条指令即使存在WAR冒险也可以依次正常运行。



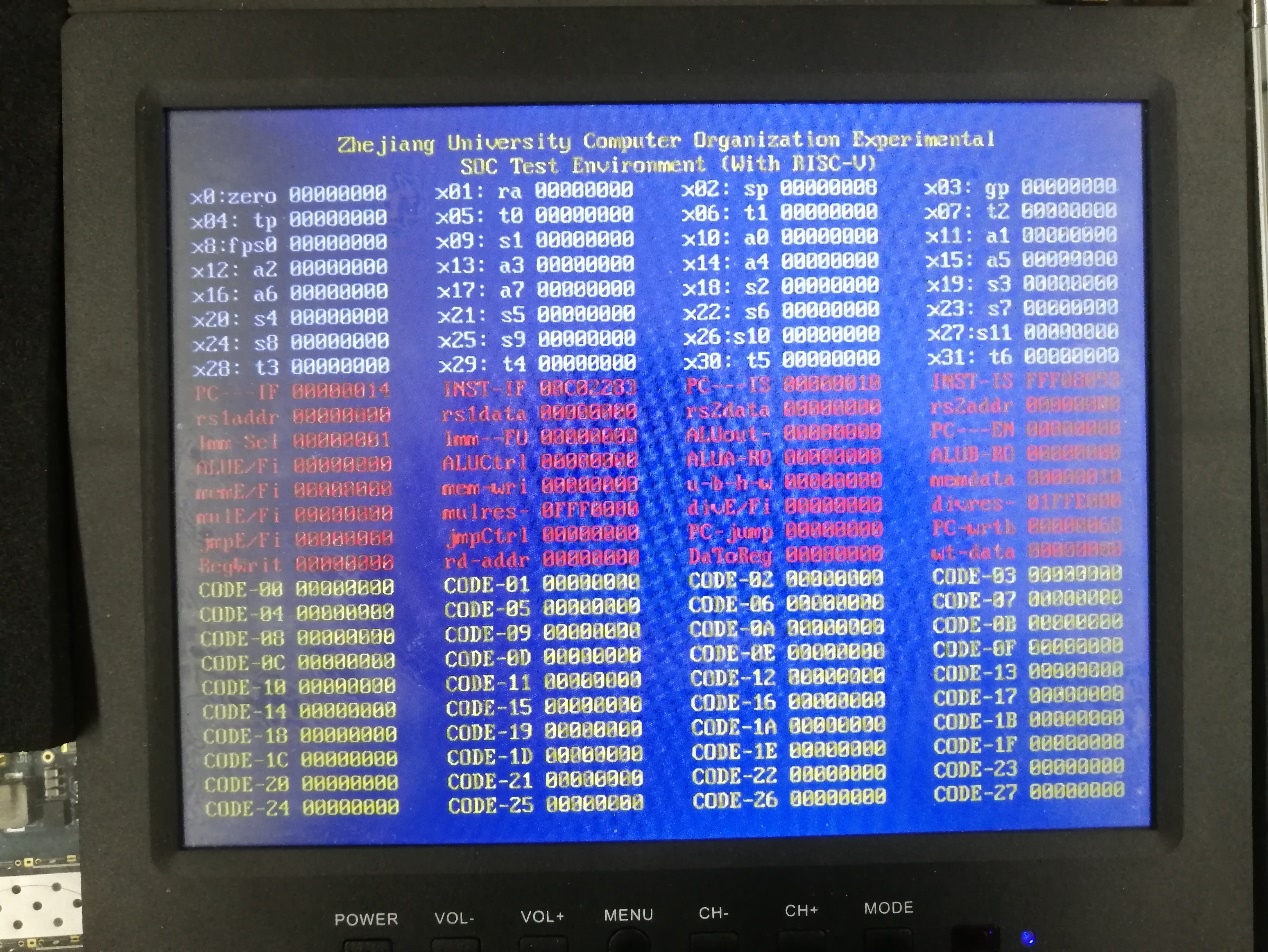
2．实验箱结果：

第二条lw指令前后：

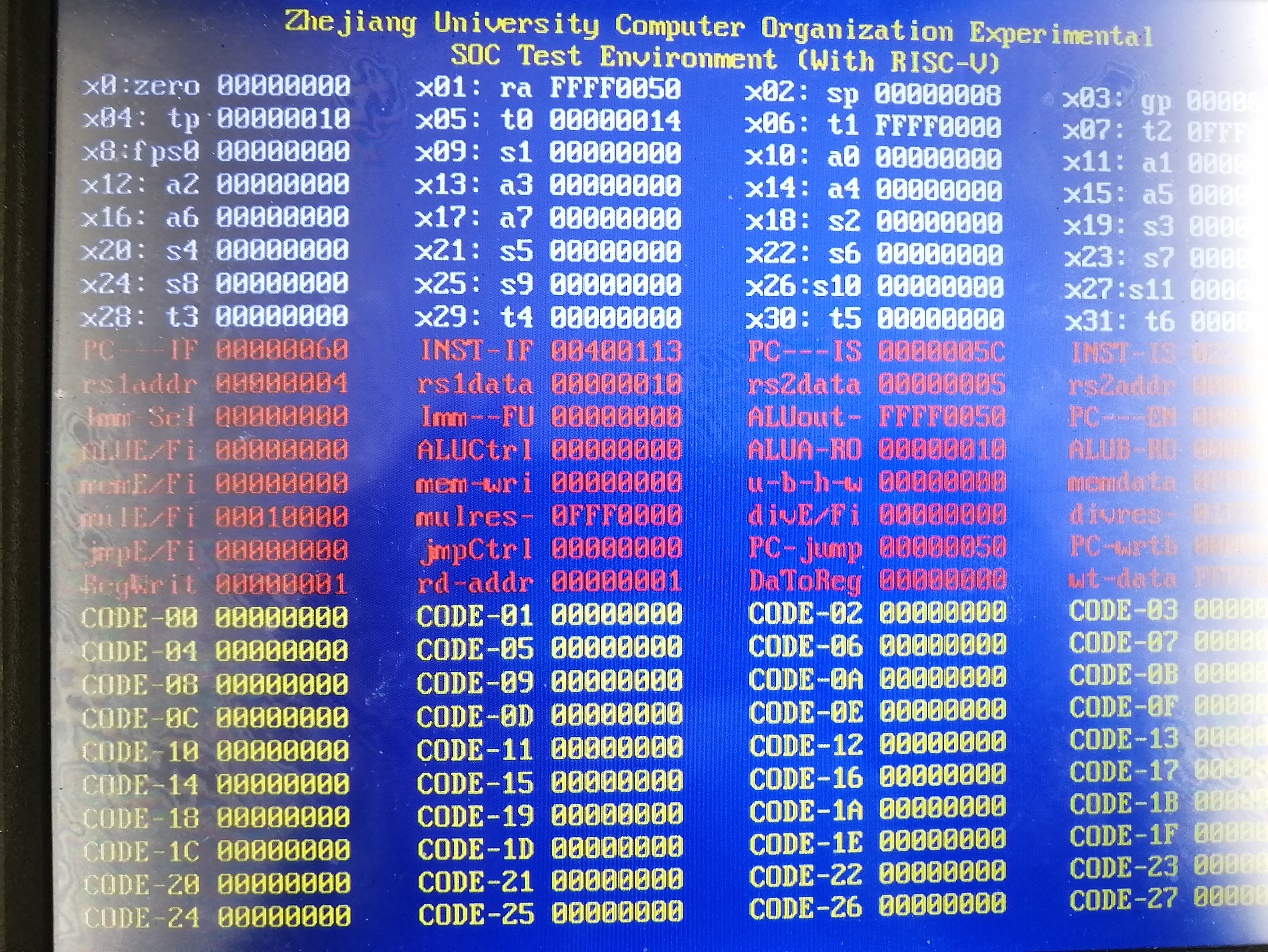


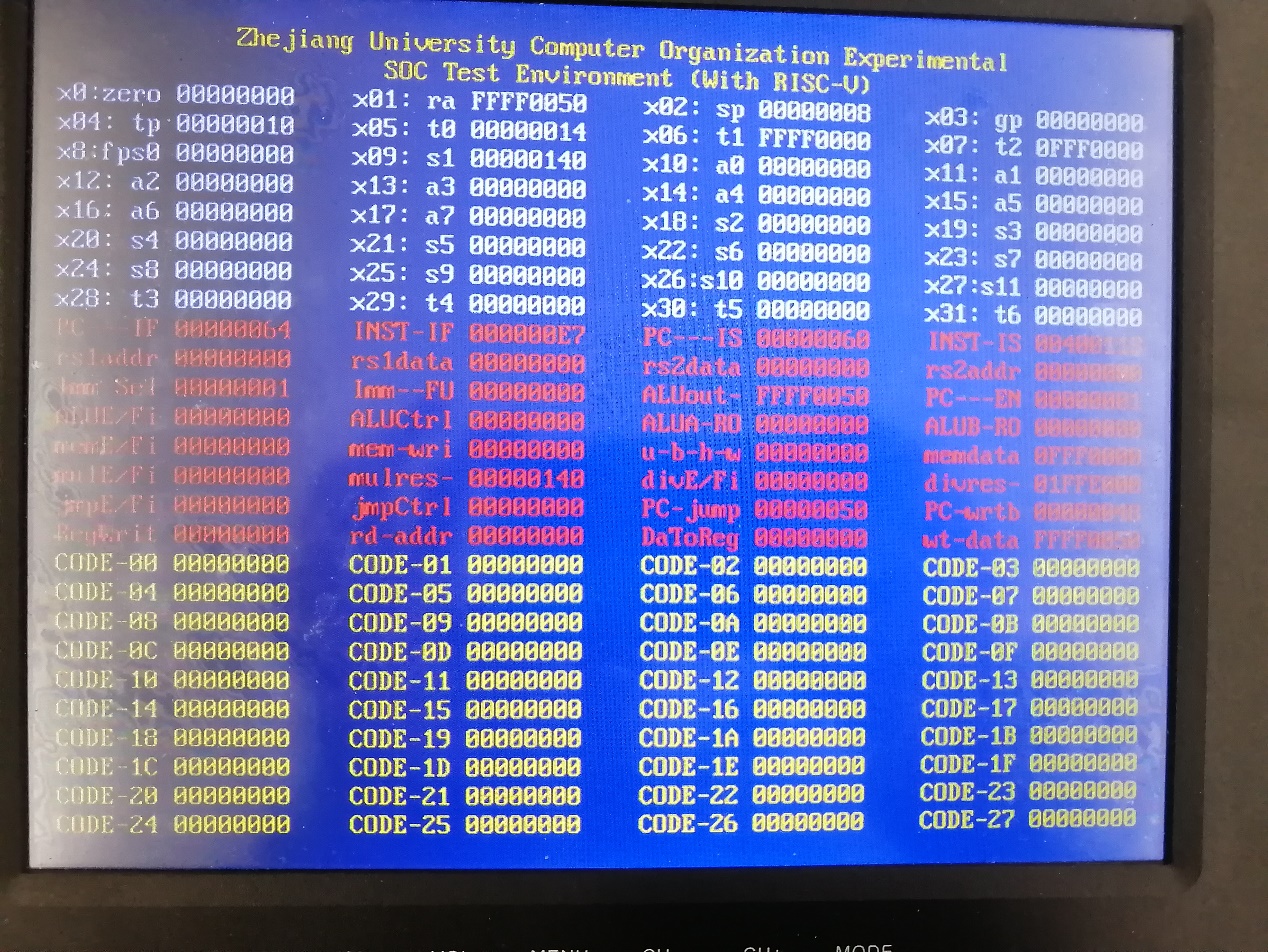


WAW指令运行时；



WAR处：





1. 讨论与心得

本次实验是在实验五完成了支持多周期计算的流水线设计的基础上进行，不同于往几次实验修改一系列单元，scoreboard只需要设计ctrlunit，但这也导致了控制单元内容变得非常繁杂，对实验代码的编辑产生了一定的阻碍。

在掌握了scoreboard调度算法的基础上，完成此次实验还是相对不难，实验中花费时间调试的地方在于很多的细节。比如，执行阶段，常常需要考虑当前周期执行的到底是哪条指令，而不能通通将所有功能单元视为执行完成。WAR信号的反向逻辑也给实验带来不小的理解难度。指令的两个源操作数都ready后，下一周期必须将其写0，因此WAR判断时通过rdy为0来判断是否已经读取出寄存器数据。