

**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： |  |
| 专 业： | 计算机科学与技术 |
| 学 号： | 3190105609 |
| 指导教师： | 陈文智 |

2021年 10月 26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 exception & interrupt**

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 10 月 26日

1. 实验目的和要求

* **Understand the principle of CPU exception & interrupt and its processing procedure.**
* **Master the design methods of pipelined CPU supporting exception & interrupt.**
* **master methods of program verification of Pipelined CPU supporting exception & interrupt.**

1. 实验内容和原理

* **Design of Pipelined CPU supporting exception & interrupt.**
  + Design exception unit
* **Verify the Pipelined CPU with program and observe the execution of program**

1. 实验过程和数据记录
2. 理解发生异常后的操作

CSR寄存器的地址为12位，但是实验只用到几个寄存器，分别是0x300的status，0x305的mtvec，0x341的mepc，0x342的mcause和0x343的mtval，实际上实验中需要在exception发生时修改的寄存器就是status、pc和cause三个。

实验涉及到的exception一共有3类：ecall、非定义指令和存取错误。当exception发生时，无论是什么类型的，都需要经过以下步骤：取出mtvec里的内容作为跳转的pc，将发生exception的pc存入mepc，记录status并写入mstatus以及记录cause写入mcause。

程序执行到跳转的pc后，进入一段trap指令段，遇到mret指令则跳回到发生mepc内的地址继续执行剩下的指令。

1. 理解CSR寄存器
2. Mtvec

Store the interruption handler entrance address

The base can be explained according to mode code

1. Mepc

Save the instruction address when exception raised or interruption happens.

* 1. the PC indicate the instruction that raise the exception
  2. the instruction need to be executed after back from interruption

1. Mcause

If mcause[31] == 1 then the trap was caused by an interruption.

Exception code field: a code identifying the last exception.

1. Mstatus

When an interruption/ exception raised: mstatus[7] 🡨 mstatus[3]

When MRET is executed: mstatus[3] 🡨 mstatus[7]

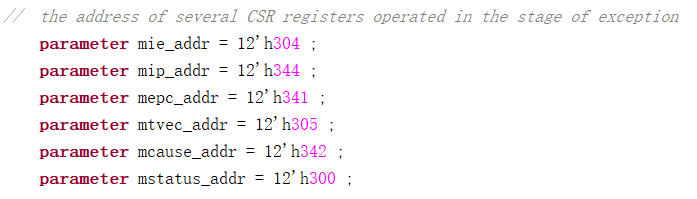
1. 理解3类CSR操作指令

* **csrrw rd, csr, rs1: t**🡨**CSRs[csr], CSRs[csr]**🡨**x[rs1], x[rd]**🡨**t**
* **csrrs rd, csr, rs1: t**🡨**CSRs[csr], CSRs[csr]** 🡨**t | x[rs1], x[rd]**🡨**t**
* **csrrc rd, csr, rs1: t**🡨**CSRs[csr], CSRs[csr]** 🡨**t & ~x[rs1], x[rd]**🡨**t**
* **csrrwi rd, csr, zimm[4..0]: x[rd]** 🡨**CSRs[csr], CSRs[csr]**🡨**zimm**
* **cssrrsi rd,csr,zimm[4..0]: t**🡨**CSRs[csr], CSRs[csr]**🡨**t | zimm; x[rd]**🡨**t**
* **csrrci rd, csr, zimm[4..0]: t**🡨**CSRs[csr], CSRs[csr]**🡨**t&~zimm; x[rd]**🡨**t**

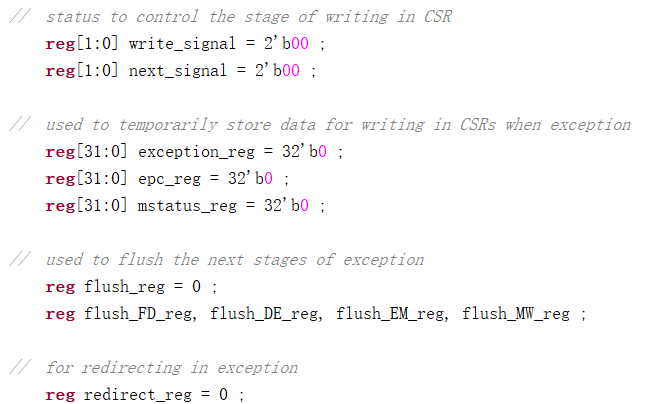
1. 设计完成Exception模块

实验中关于exception信号的处理和核心模块的信号传递都已经完成，只需要根据接口信息处理发生异常后的对应寄存器。并且Exception单元也设计好了CSR寄存器模块及其读写，只需要设计合理时序使得模块有序地完成各个寄存器的读和写即可。以下结合实际代码解读：

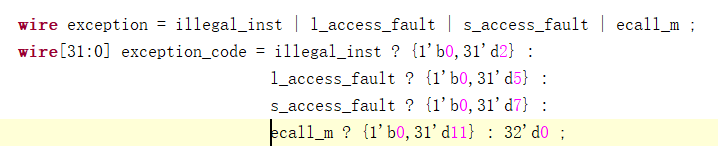
（1）实验操作的各个寄存器的地址



（2）模块用到的各个状态量



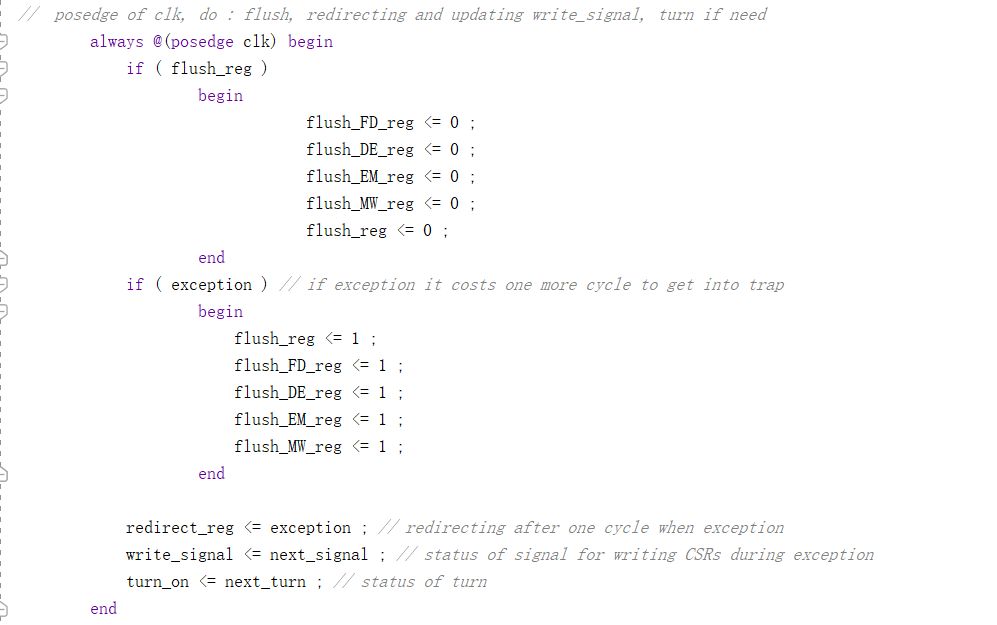
（3）exception发生时的变量



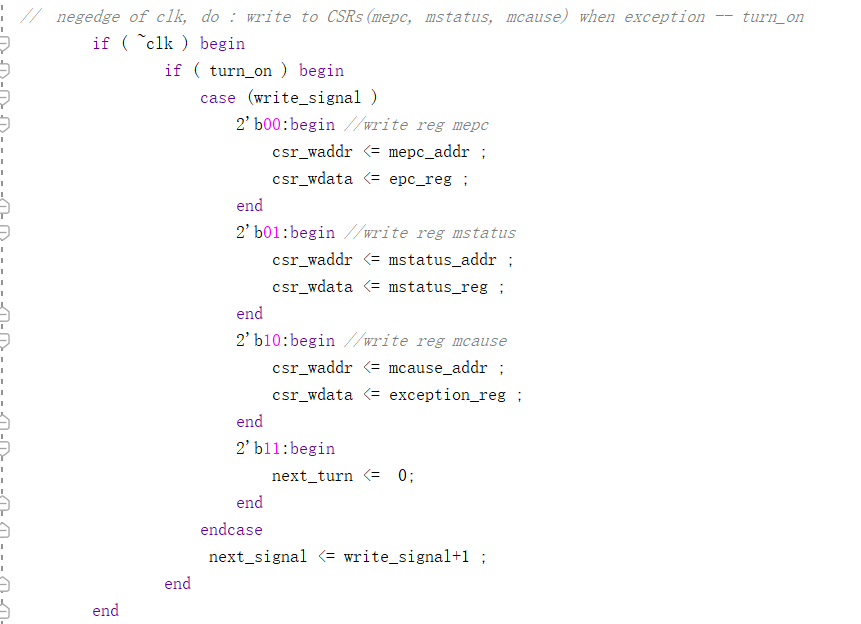
其中exception\_code为mcause的值。

实验设计了两个always模块组合而成的电路，一个用于处理各个状态寄存器，在时钟上升沿操作，另一个在\*操作，设定各个要写进CSR寄存器模块的值。其中，在时钟下降沿单独处理状态机的寄存器，为了能够在上升沿即让CSRregs模块完成读和写，这样节省了exception发生后的操作周期。

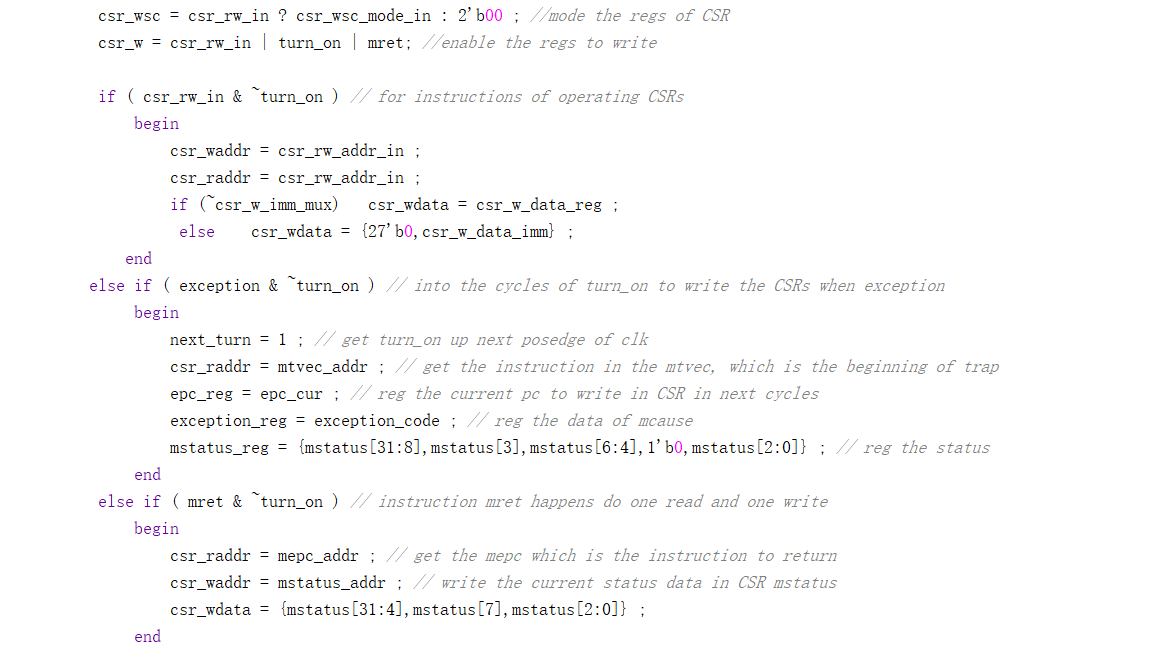
（4）时钟上升沿操作



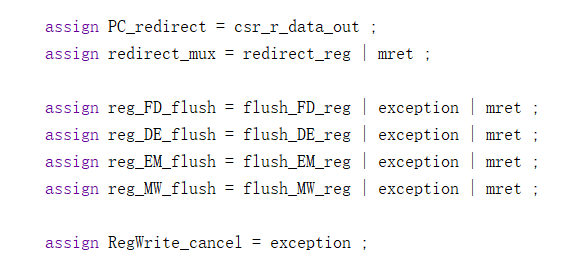
1. 时钟下降沿



1. Always@\*



1. 输出信号处理

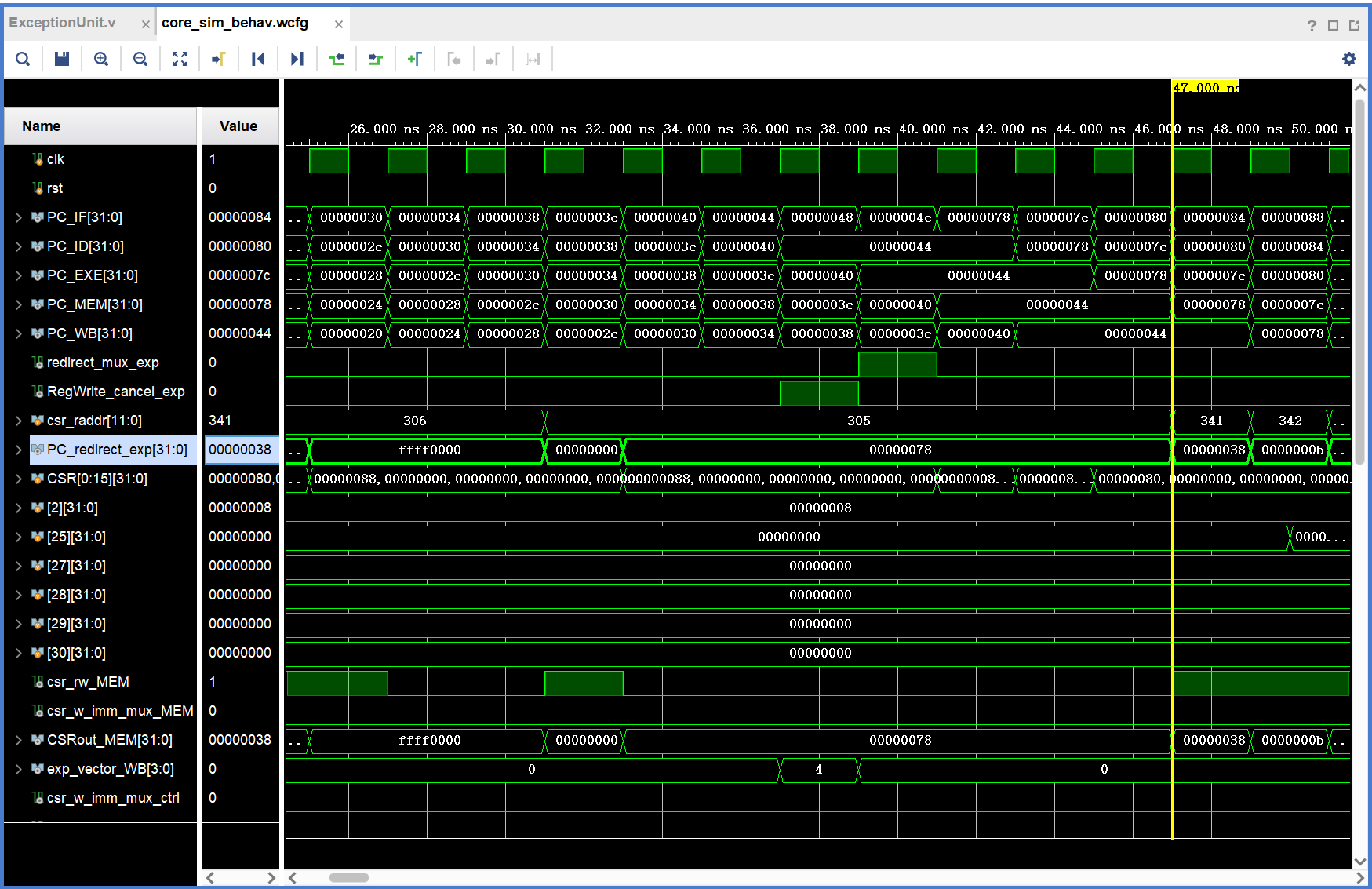


1. 一些时序设计的解释

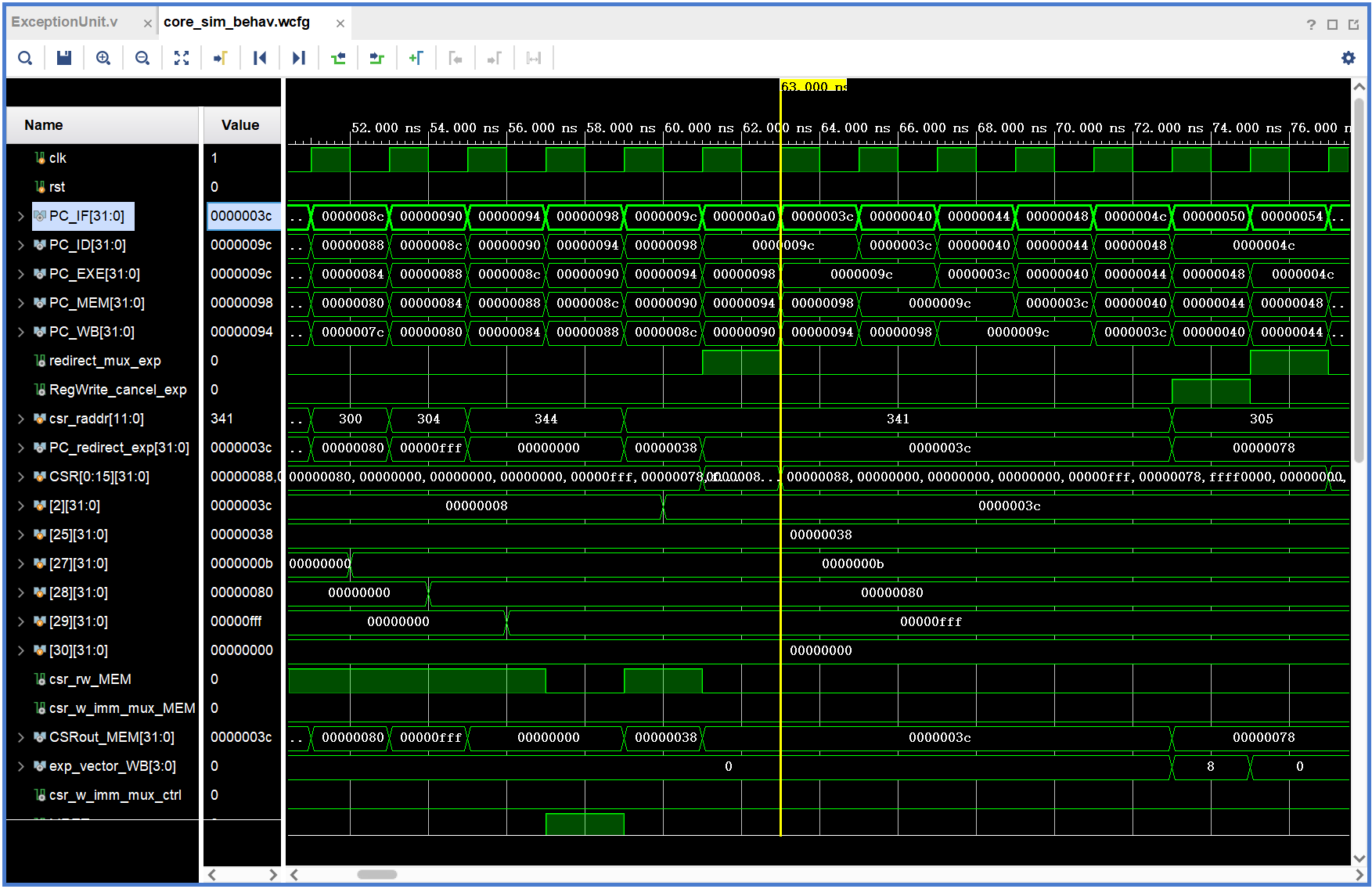
关于flush输出信号的处理，用到了寄存器暂存要清空的周期，这是在实验中通过仿真信号调整设计的，当exception发生时，该指令已经执行到了WB阶段，而前面阶段则相应地取到了exception后的3条指令且执行了1到3个阶段。这些阶段的中转寄存器里的内容在这个周期被flush掉，而后相应的执行结果仍会被存入，需要再flush一个周期来清空内容。Mret指令则不同，因为实验测试指令中mret后设计了对应的空指令来帮助跳转，即使不做更多的flush也不会影响流水线跳转后的运行。

1. 实验结果分析
2. 仿真波形

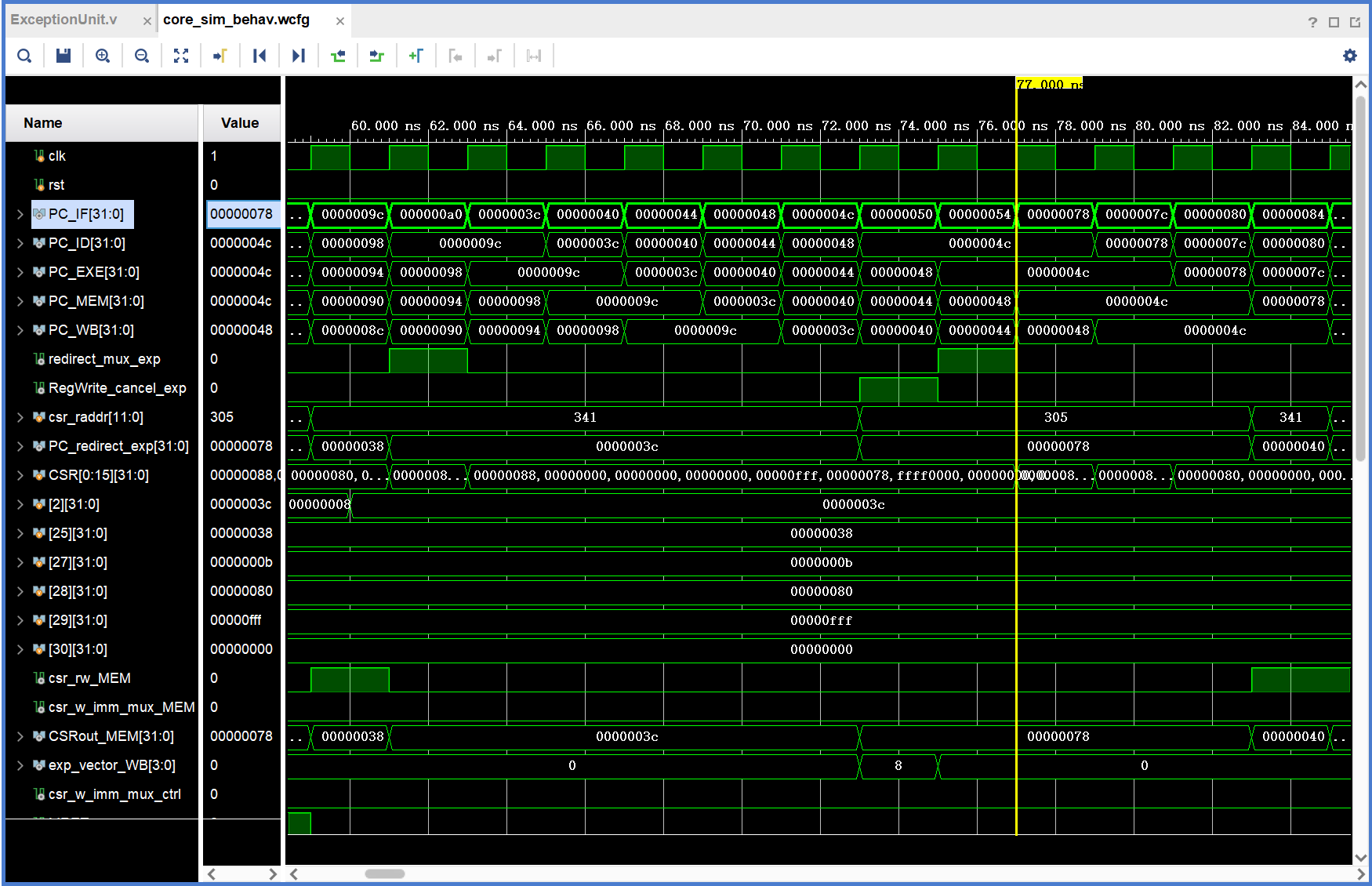
Ecall：38跳转到78



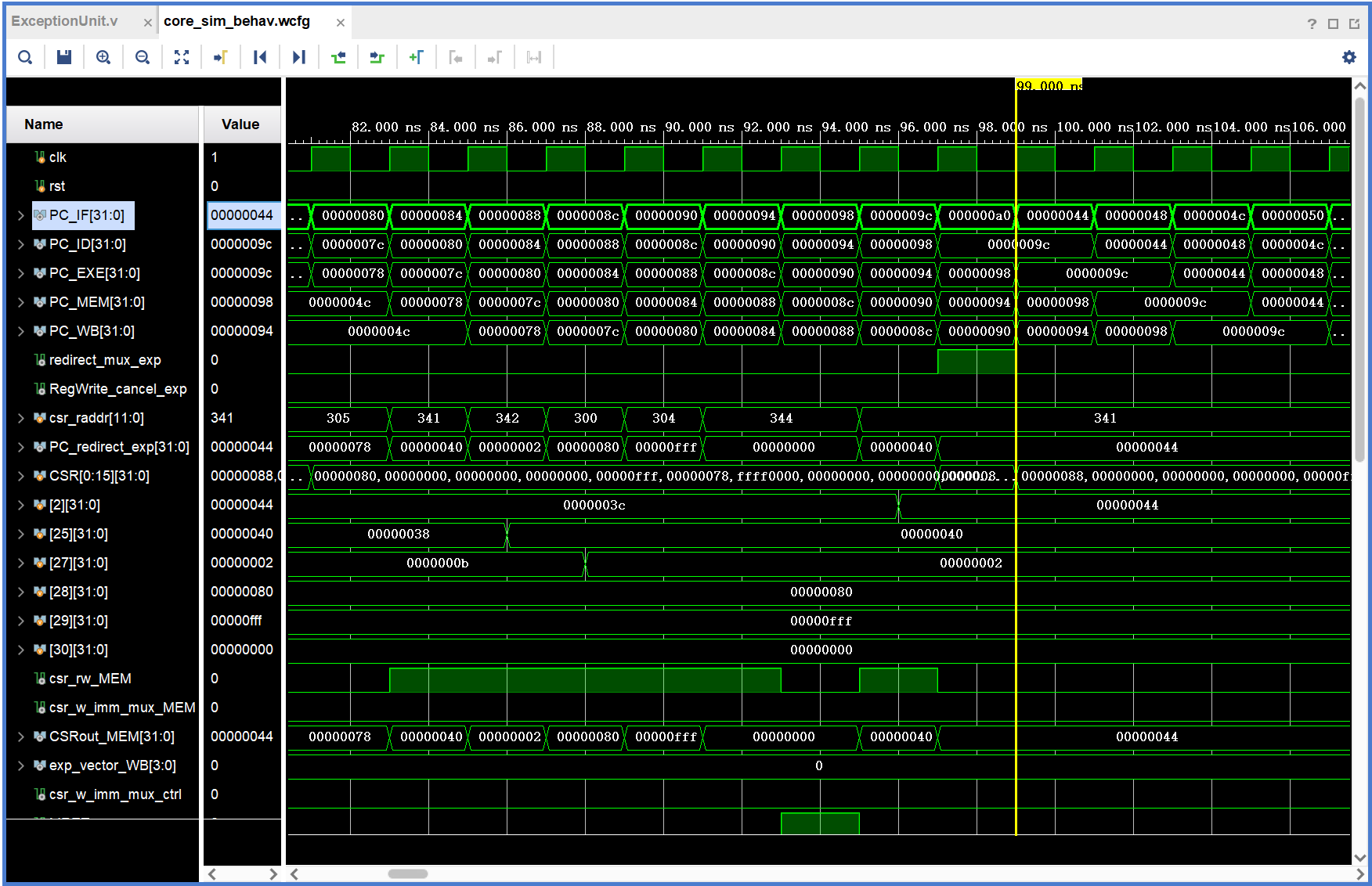
Mret：跳转回3c



Illegal：40跳转到78

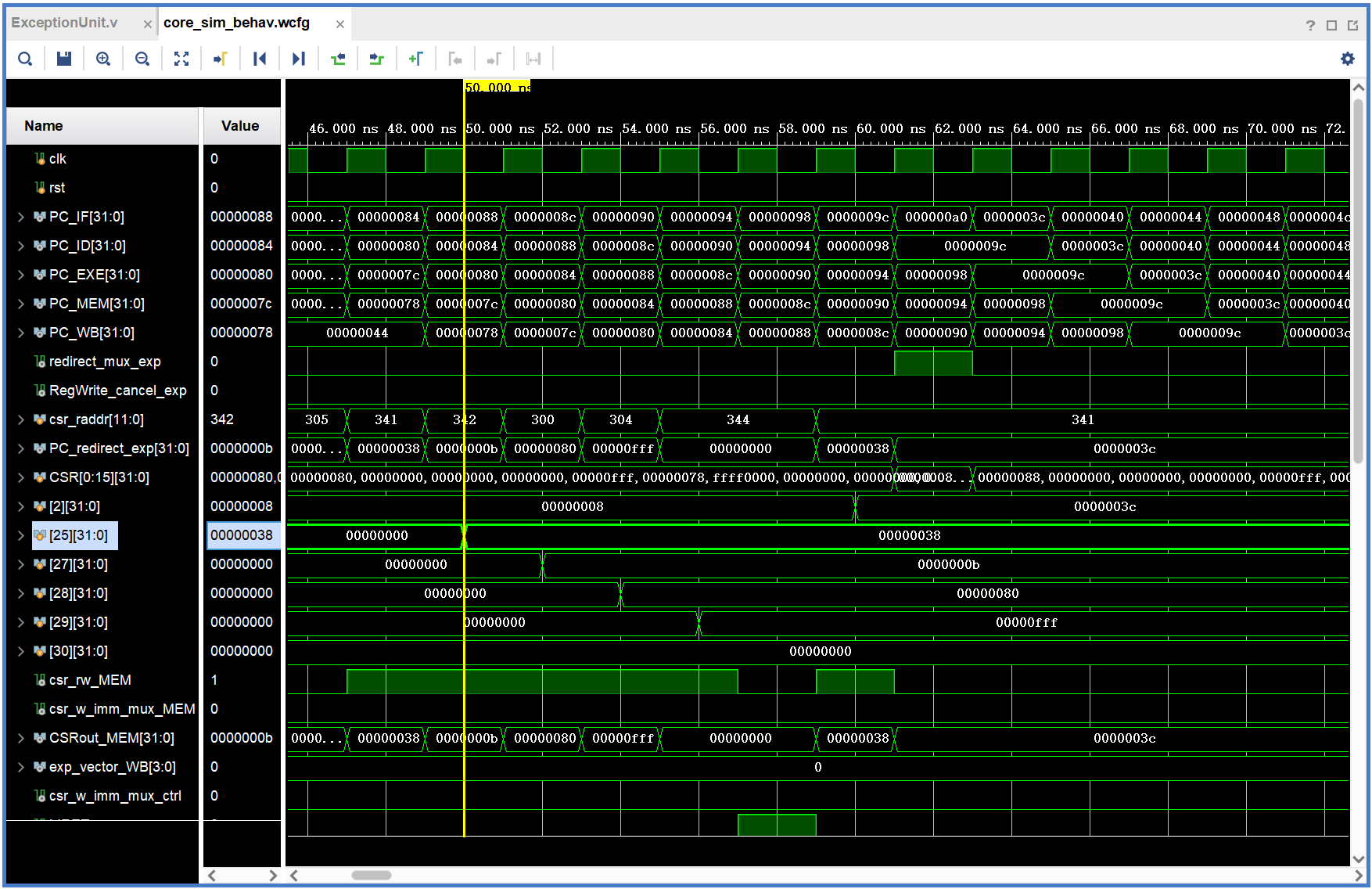


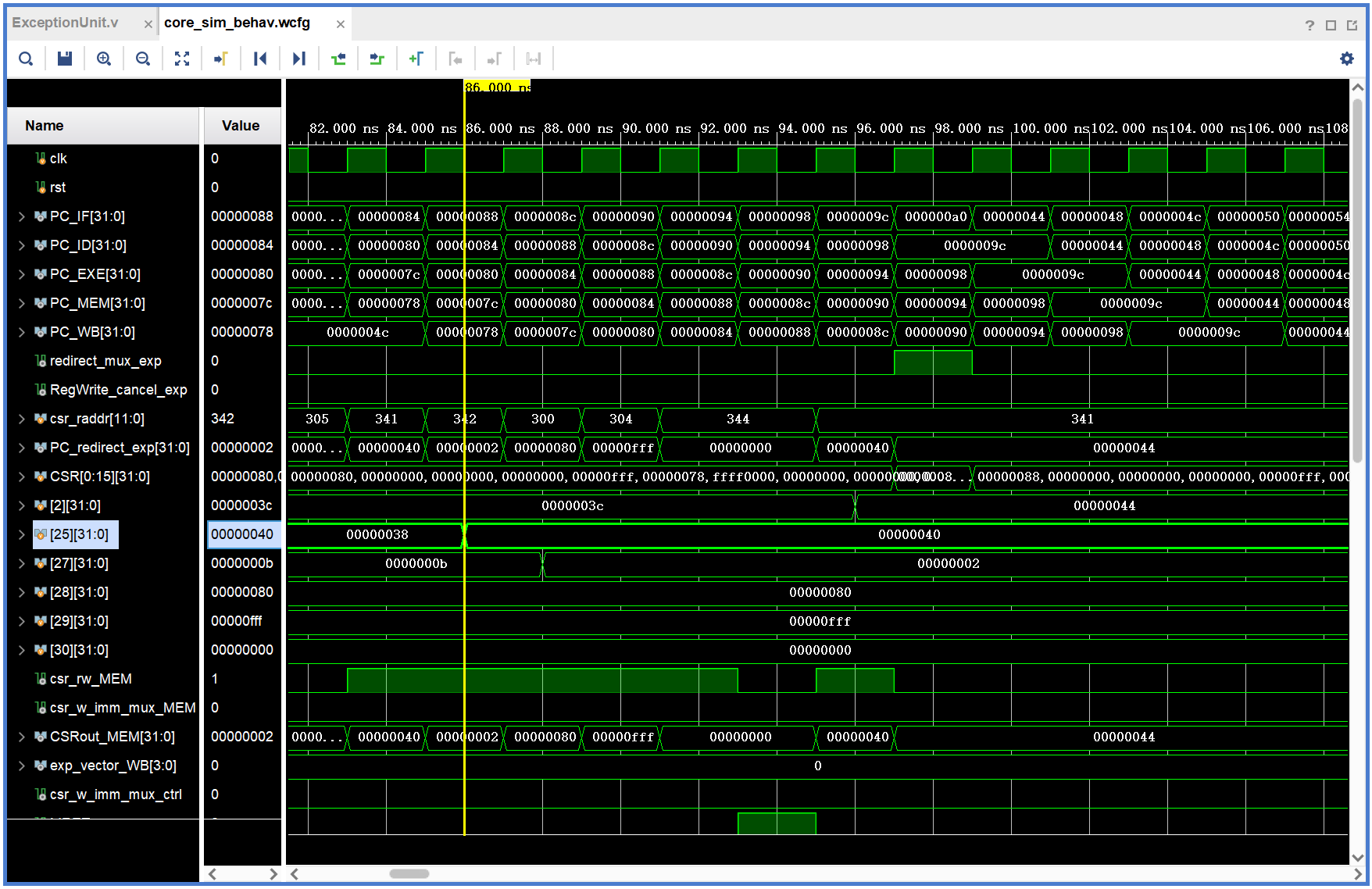
Mret：跳转回44



后续跳转与之前相同，均正常跳转。

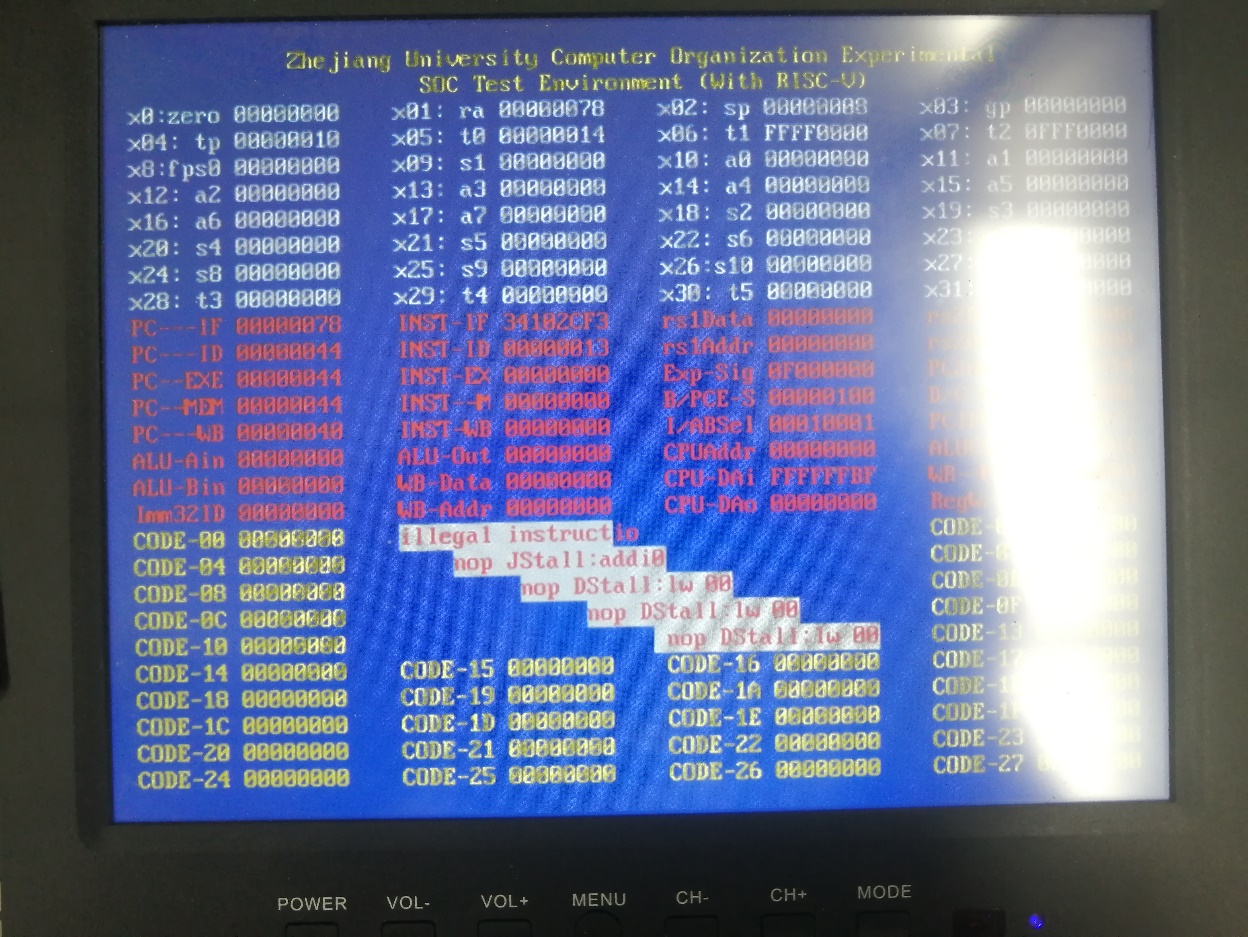
寄存器观察：





1. 实验箱记录

·第一次跳转ecall



·mret



·illegal指令跳转



·第二次mret



1. 讨论与心得

起初实验箱结果与仿真结果不一致，在exception发生时跳转失败，从助教了解到可能是由于改用了下降沿来处理信号，但是自己思考后又觉得下降沿处理的信号并没有很大的影响。仿真对实际电路的逻辑没有很严格的要求，只要信号处理合理就能得到正确的信号，后续改动了一些不严格的逻辑后才又成功在实验箱中观察到正常的跳转。一个可能的原因是我在不同的always模块里改动了同一个信号，这在实际电路中是会产生竞争冲突的。

此外，自己之前对于Verilog设计的理解不充分，也写出过*case X 2’b00:X<=X+1;* 这样违背电路逻辑的设计，这样的写法导致了仿真在遇到这个处理时直接产生了fatal error而直接全部中断。经过这一次实验后，自己对Verilog的设计又有了进一步的理解。