

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: Topic 2. pipelined CPU supporting exception & interrupt

指导教师: 何水兵 完成时间: 2023.11.7

姓名	詹含蓓	学号	3210106333
同组学生姓名	居圣桐	学号	3210105812
分工信息	詹含蓓: 实现 CPU 中断状态机 居圣桐: 实现其他类型中断, 嵌套中断		

一、 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.

二、 实验内容和原理

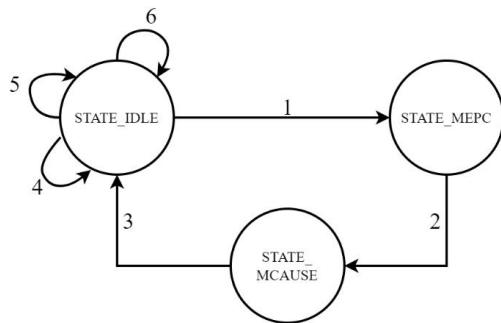
(一) 实验内容

- Design of Pipelined CPU supporting exception & interrupt.
 - Design datapath
 - Design Co-processor & Controller
- Verify the Pipelined CPU with program and observe the execution of program

(二) 实验原理

- 当 RISC-V 机器遇到中断或异常时, 其处理器会自动执行以下操作:
 1. 保存异常指令的程序计数器 (PC) 到名为 mepc 的寄存器中, 同时将 PC 设置为 mtvec。

2. mepc 指向导致异常的指令；对于中断，它指向中断处理后应继续执行的位置。
 3. 根据异常来源设置 mcause，并将 mtval 设置为引起错误的地址或其他适用于特定异常的信息。
 4. 将控制状态寄存器 mstatus 中的 MIE 位置清零以禁用中断，并将先前的 MIE 值保存到 MPIE 中。
- 保留在发生异常之前的权限模式到 mstatus 的 MPP 域中，然后将权限模式更改为 M。除了中断和异常处理外，中断命令还需与正常操作控制和状态寄存器的指令相兼容。
 - 中断过程分为状态机如下：后面的实验都基于这个状态机。



- 其他指令类型中断和软件中断
 - 实现嵌套中断的关键步骤如下：
 - 在中断处理程序中，将 MSTATUS 的 mie 置为 1。避免直接放弃置 0 的原因是确保中断发生到跳转到处理程序的过程中不处理其他中断，以避免潜在的未知错误。
 - 在跳转到中断处理程序之前，修改 MIE 寄存器的特定位，即 11、7、3 位。这些位分别对应外部中断、计时器中断和软件中断。当置 1 时表示可以触发，置 0 时表示不能触发。
 - 在触发中断时，根据具体中断种类修改 MIE 寄存器。确保按照中断类型适当地修改 MIE 寄存器。
 - 最后，利用空闲的 CSR 寄存器保存 MEPC 的值。这样可以有效地保存中断处理程序中的重要信息。

三、实验过程和数据记录及结果分析

(一) 实验过程

首先我们先按照流水线完成状态机的模板，代码截图为原始状态机代码，在实现嵌套中断的过程中进行了修改。

1. 设置寄存器地址和状态机编码

```
parameter MSTATUS = 12'h300;
parameter MTVEC   = 12'h305;
parameter MEPC    = 12'h341;
parameter MCAUSE   = 12'h342;
parameter MVAL     = 12'h343;
parameter MIE      = 12'h304;

parameter state_idle = 2'b00;
parameter state_mepc = 2'b01;
parameter state_mcause = 2'b10;
```

2. 根据状态机书写中断异常处理代码

(1) STATE_IDLE

① STATE_IDLE →(exception or interruption) STATE_MEPC

首先根据信号来判断是否为异常，并设置 trap 触发条件

```
assign exception = illegal_inst | l_access_fault | s_access_fault | ecall_m;
assign trap_in = mstatus[3] & (interrupt | exception);
```

设置此时 mstatus 寄存器，其中 mstatus.MPIE=mstatus.MIE，mstatus.MIE = 0，并且根据异常类型记录 cause，判断是异常还是中断来记录 epc。设置下一状态。

```

    case(cur_state)
    state_idle:begin
        if(trap_in) begin
            csr_w = 1'b1;
            csr_wdata = {mstatus[31:8], mstatus[3], mstatus[6:4], 1'b0, mstatus[2:0]};
            csr_waddr = MSTATUS;
            csr_wsc = 2'b01;
            next_state = state_mepc;
            if(exception) begin
                epc_record = epc_cur;
                if(illegal_inst) begin
                    cause_record = 32'h2;
                end
                else if(l_access_fault) begin
                    cause_record = 32'h5;
                end
                else if(s_access_fault) begin
                    cause_record = 32'h7;
                end
                else if(ecall_m) begin
                    cause_record = 32'hb;
                end
            end
            else begin
                epc_record = epc_next;
                cause_record = 32'h8000000b;
            end
        end
        else if(mret)begin

```

② STATE_IDLE → (mret) STATE_IDLE

设置此时 mstatus 寄存器, mstatus.MIE = mstatus.MPIE, mstatus.MIE = 1;

通过设置读地址为 mepc 的地址读取当前的 mepc。设置重定位的 mux 信号为 1。下一状态仍然为 IDLE。

```

    end
    else if(mret)begin
        csr_w = 1'b1;
        csr_wdata = {mstatus[31:8], 1'b1, mstatus[6:4], mstatus[7], mstatus[2:0]};
        csr_waddr = MSTATUS;
        csr_wsc = 2'b01;
        csr_raddr = MEPC;
        next_state = state_idle;
    end

```

③ STATE_IDLE → (csr insts) STATE_IDLE

将输入传入的信号传入 csr 操作接口来进行 csr 相关指令对 csr 的操作。

下一状态仍然为 IDLE。

```

    end
    else if(csr_rw_in)begin
        csr_w = 1'b1;
        csr_waddr = csr_rw_addr_in;
        csr_raddr = csr_rw_addr_in;
        csr_wdata = csr_w_data_reg;
        csr_wsc = csr_wsc_mode_in;
        next_state = state_idle;
    end

```

④ STATE_IDLE → (other) STATE_IDLE

保持当前状态，设置状态仍然为 STATE_IDLE

```

        end
    else begin
        next_state = state_idle;
    end
end

```

(2) STATE_MEPC → STATE_MCAUSE

把记录的 epc 存到 mepc 当中，读取 mtvec 获得 trap 跳转地址，同时释放掉 FD 的寄存器。设置重定位 PC 的 mux 信号为 1 跳转 trap 地址。

```

end
state_mepc:begin
    csr_w = 1'b1;
    csr_waddr = MEPC;
    csr_wdata = epc_record;
    csr_wsc = 2'b01;
    next_state = state_mcause;
    csr_raddr = MTVEC;
end

```

(3) STATE_MCAUSE → STATE_IDLE

把记录的 cause 存到 mcause 寄存器当中。

```

end
state_mcause:begin
    csr_w = 1'b1;
    csr_waddr = MCAUSE;
    csr_wdata = cause_record;
    csr_wsc = 2'b01;
    next_state = state_idle;
end
default:begin

```

(4) 根据状态图设置几个输出信号

```

assign RegWrite_cancel = (cur_state==state_idle) & exception;
assign PC_redirect = csr_r_data_out;
assign redirect_mux = ((cur_state==state_idle) & mret) | (cur_state==state_mepc);

```

(5) 流水线寄存器 flush

```

assign reg_FD_flush = ((cur_state==state_idle) & (trap_in | mret)) | (cur_state==state_mepc);
assign reg_DE_flush = (cur_state==state_idle) & (trap_in | mret);
assign reg_EM_flush = (cur_state==state_idle) & (trap_in | mret);
assign reg_MW_flush = (cur_state==state_idle) & trap_in;

```

(6) 其他情况保留在 IDLE 状态

```

end
default:begin
    next_state = state_idle;
end
endcase

```

(7) 根据时钟将下一个状态赋予当前状态

```

1      always @(posedge clk) begin
2          cur_state <= next_state;
3      end
4  endmodule

```

3. 实现其他类型中断

- (1) 添加其他中断信号
- (2) 在每两个阶段中的 REG 进行信号的传递，并在 exception_unit 中添加
- (3) 增加 state_idle 状态下识别出 trap_in 后的需要更新的 MCAUSE 值

```

if (cur_state == state_idle) begin
    if(exception) begin
        epc_record <= epc_cur;
        if(illegal_inst) begin
            cause_record <= 32'h2;
        end
        else if(l_access_fault) begin
            cause_record <= 32'h5;
        end
        else if(s_access_fault) begin
            cause_record <= 32'h7;
        end
        else if(ecall_m) begin
            cause_record <= 32'hb;
        end
    end
    else begin
        if(interrupt) begin
            epc_record = epc_next;
            cause_record = 32'h8000000b;
            modifymp = 2'b11;
        end
        else if(sinterrupt) begin
            epc_record = epc_next;
            cause_record = 32'h80000003;
            modifymp = 2'b10;
        end
    end
end

```

4. 实现嵌套中断，这个过程当中的部分是在软件层面实现。

1. 将 MCAUSE 的 mie 位置 0，以防止在中断跳转过程中触发其他中断。
2. 在中断程序中，执行 MEPC+4 的操作，以防止异常持续发生。
3. 将 mie 位置 1，以响应更高级别的中断。

```

trap:
csrr x25, 0x341 # mepc

addi x2, x25, 4

csrwr 0x341, x2

csrwi 0x300, 8

change = 1'b0;
case(cur_state)
state_idle:begin
    if(trap_in) begin
        csr_w = 1'b1;
        csr_wdata = {mstatus[31:8], mstatus[3], mstatus[6:4], 1'b0, mstatus[2:0]};
        csr_waddr = MSTATUS;
        csr_wsc = 2'b01;
        next_state = state_mepc;
        if(interrupt) begin
            modifymp = 2'b11;
        end
        else if(sinterrupt) begin
            modifymp = 2'b10;
        end
    end
end

```

- 引入了一个名为“modifymp”的状态变量指示当前中断状态
- 在 MIE 寄存器中，分别控制外部中断（11 位）、计时器中断（7 位）和软件中断（3 位）
- 当状态发生变化时保存 MEPC 的值，为此使用一个空闲的 CSR 寄存器进行保存
- 用一个名为“change”的变量来标识状态是否发生了变化。

```

if(change) begin
    case(modifymp)
        2'b00: begin
            CSR[4] <= 32'hfff;
        end
        2'b01: begin
            CSR[4] <= 32'hf7f;
            if(~(|CSR[12])) CSR[12] <= CSR[9];
            else begin
                CSR[9] <= CSR[12];
                CSR[12] <= 0;
            end
        end
        2'b10: begin
            CSR[4] <= 32'hf77;
            if(~(|CSR[13])) CSR[13] <= CSR[9];
            else begin
                CSR[9] <= CSR[13];
                CSR[13] <= 0;
            end
        end
        2'b11: begin
            CSR[4] <= 32'h777;
            if(~(|CSR[14])) CSR[14] <= CSR[9];
            else begin
                CSR[9] <= CSR[14];
                CSR[14] <= 0;
            end
        end
    endcase

```

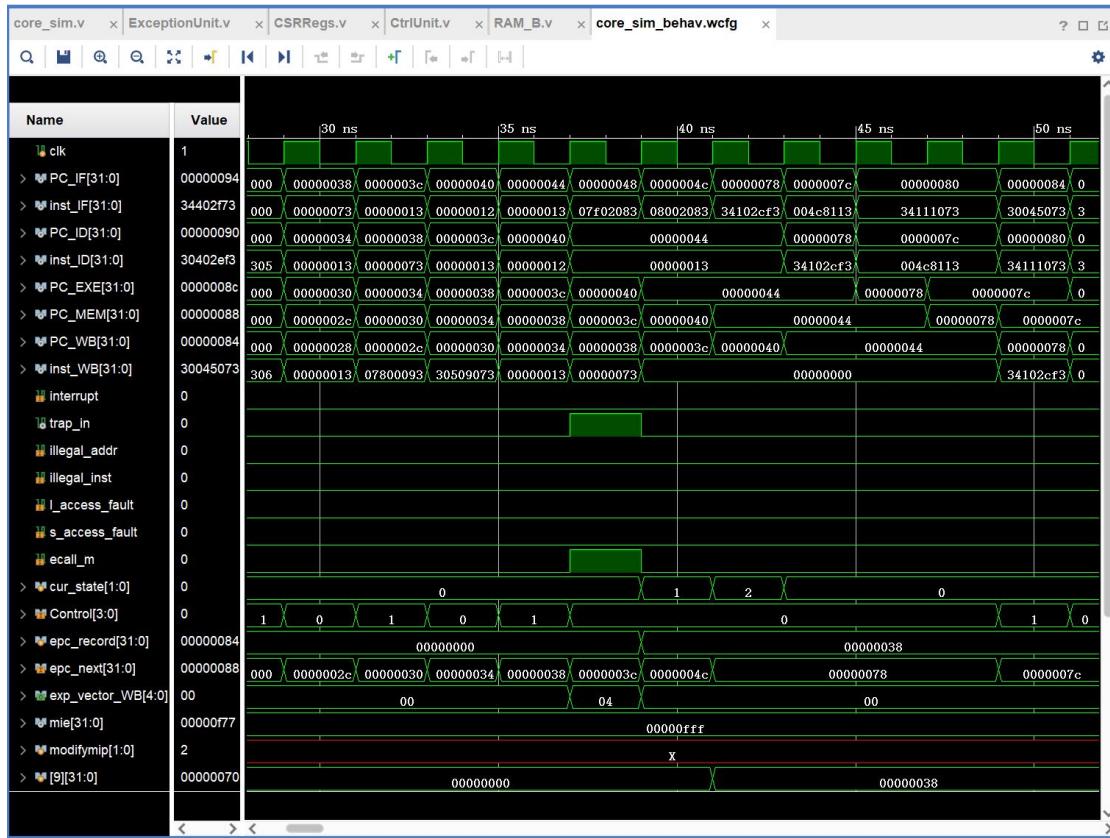
5. 进行仿真实验

- (1) 将仿真文件夹当中的 rom.hex 文件结尾增加四行 00000013
- (2) 将部分代码改为后添加的异常与中断
- (3) 增加了外部中断信号

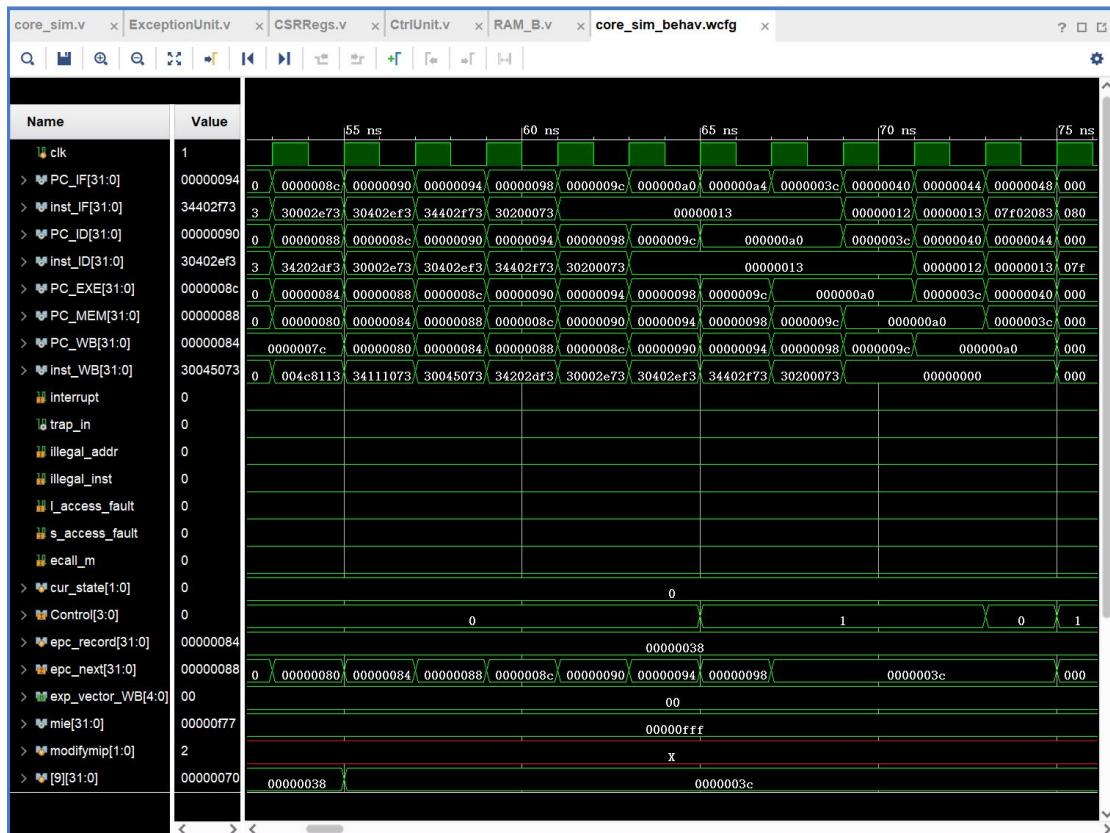
(三) 实验结果及分析

1. 仿真结果

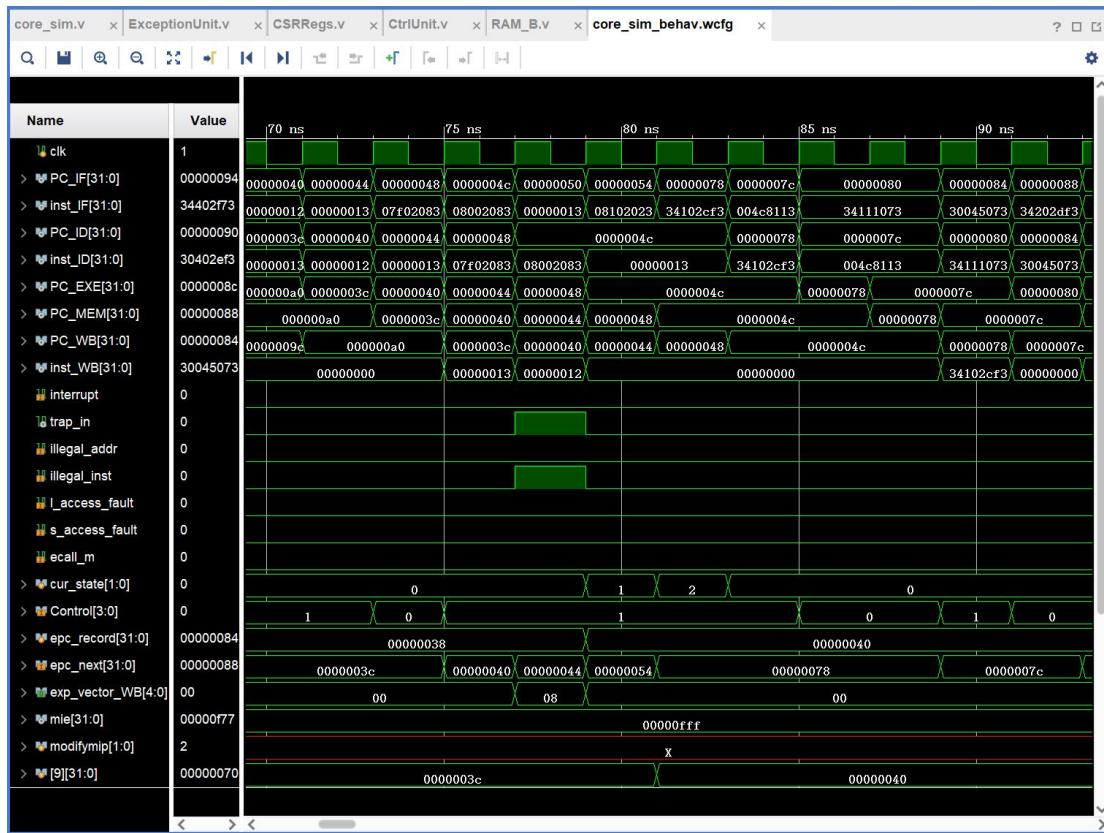
1. PC 为 38 时调用指令 ecall 进入中断程序，trap 入口 pc 为 38。在 WB 级响应，经过 MEPC 设置后跳转到 trap。



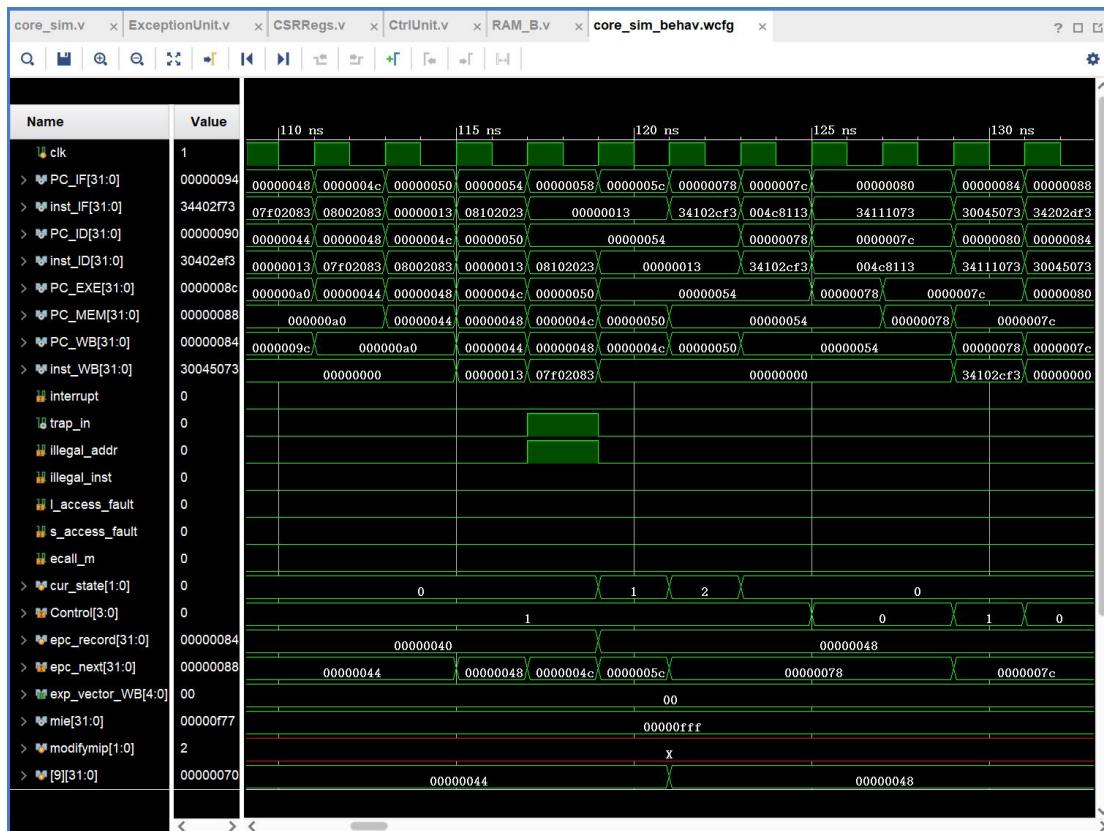
2. Trap 最后一条指令覆盖了储存的 mepc，修改为 mepc+4 也就是 3c。处理结束后 mret 回到 3c 继续执行程序。



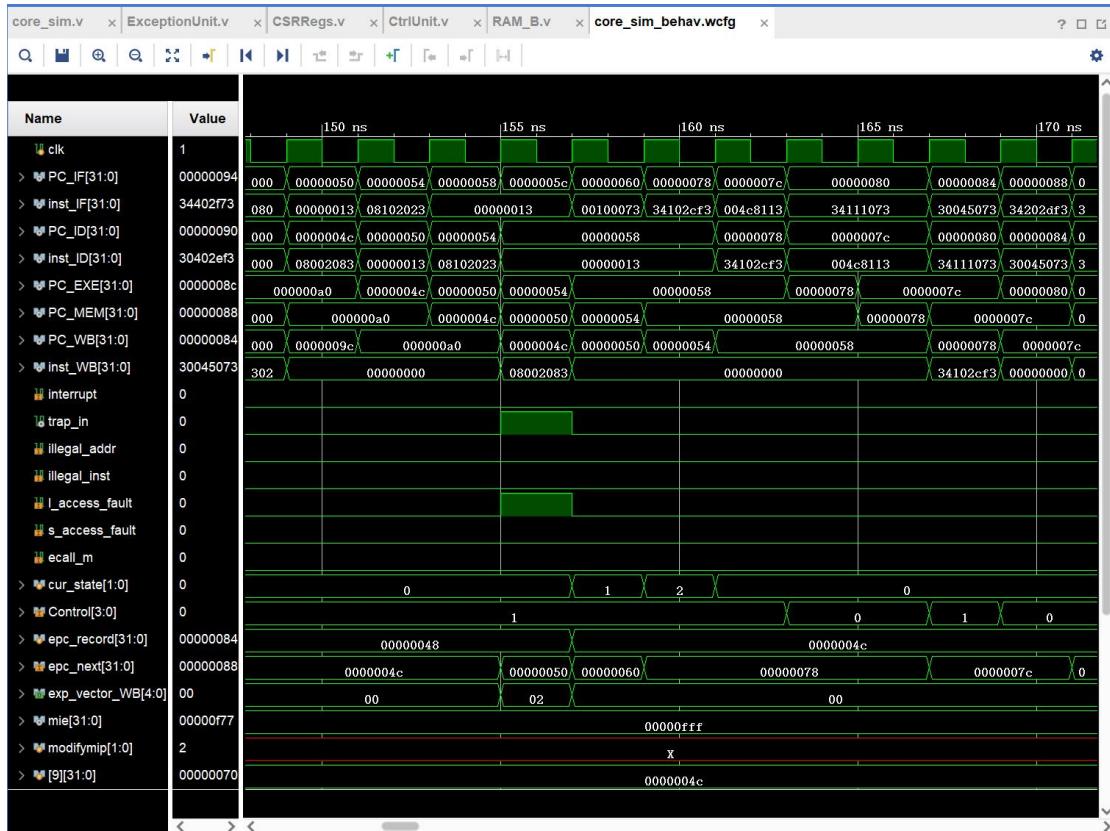
3. PC 为 40 处发生指令异常，跳转到 trap 函数地址处理异常。



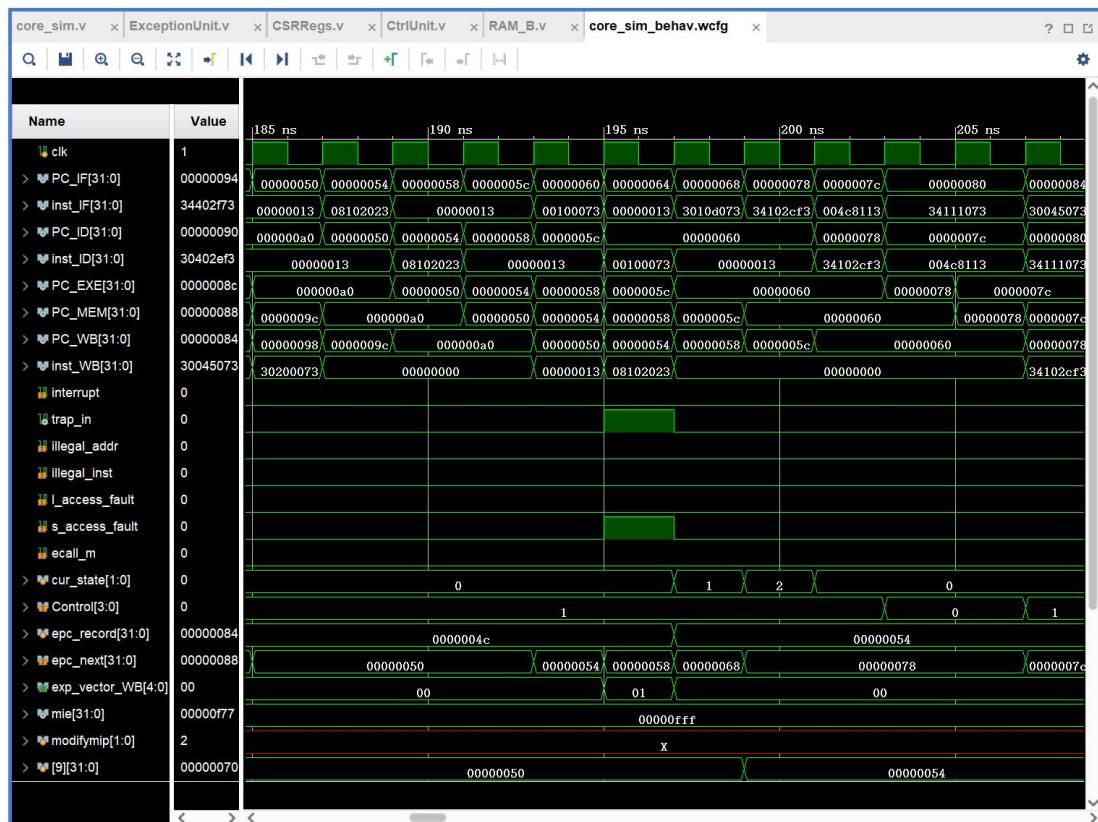
4. 在 PC=48 处发生的 illegal_address



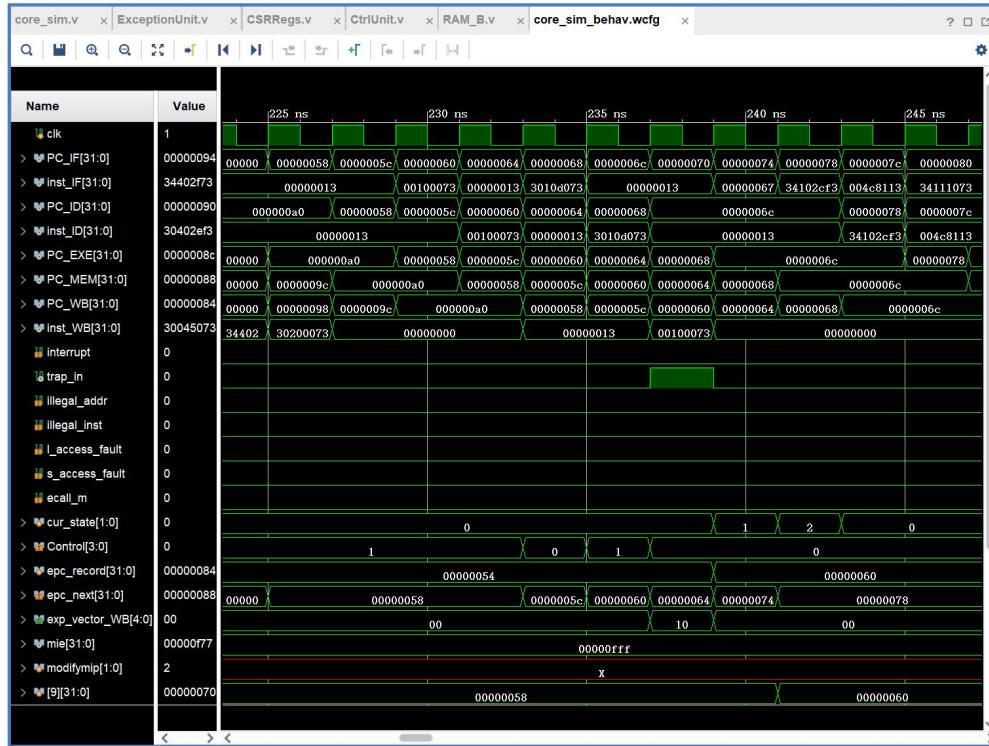
5. PC = 4C 处发生 1 access fault



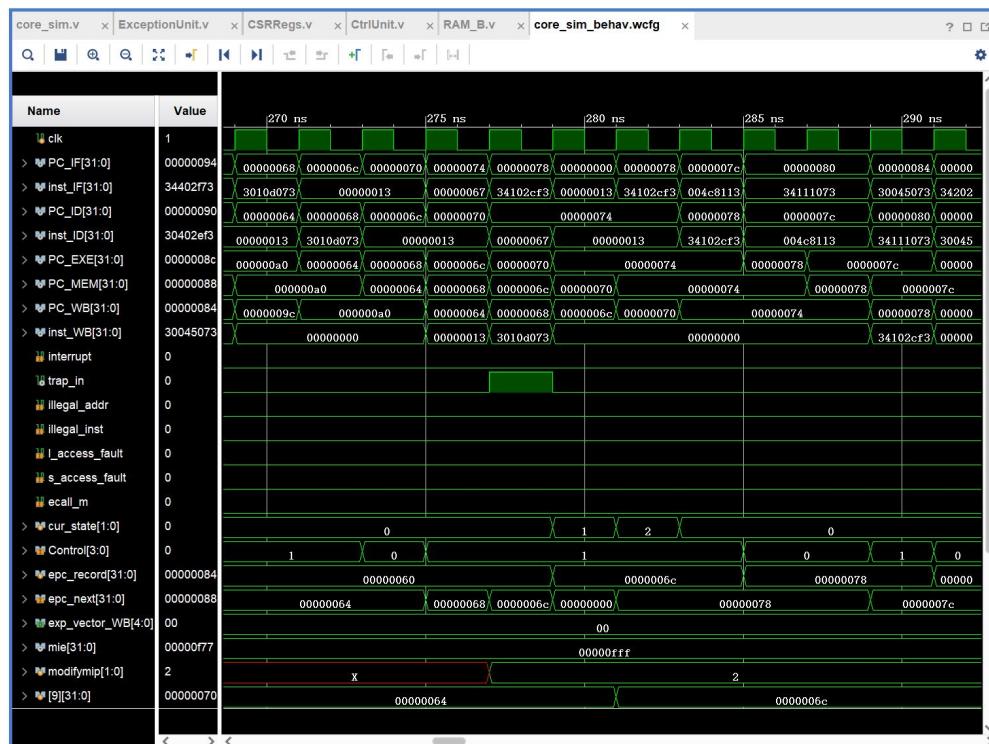
6. PC = 54 处发生 s access fault



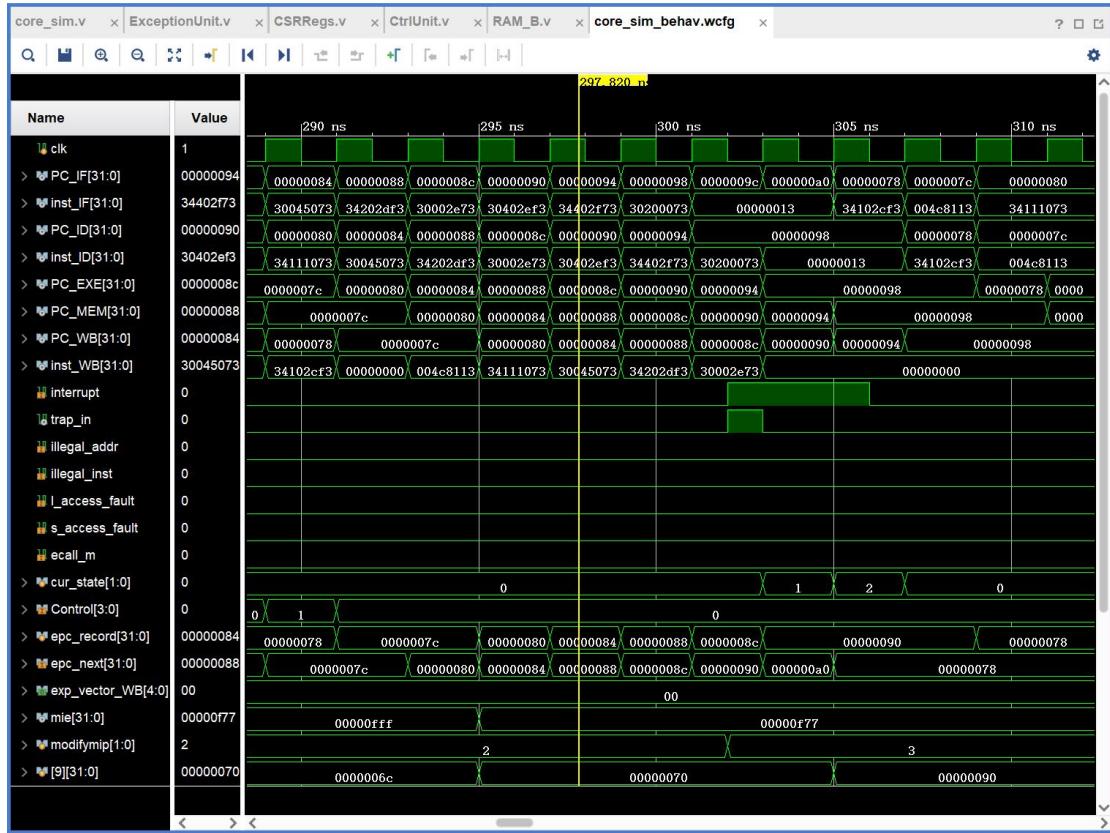
7. PC = 60 处发生 ebreak



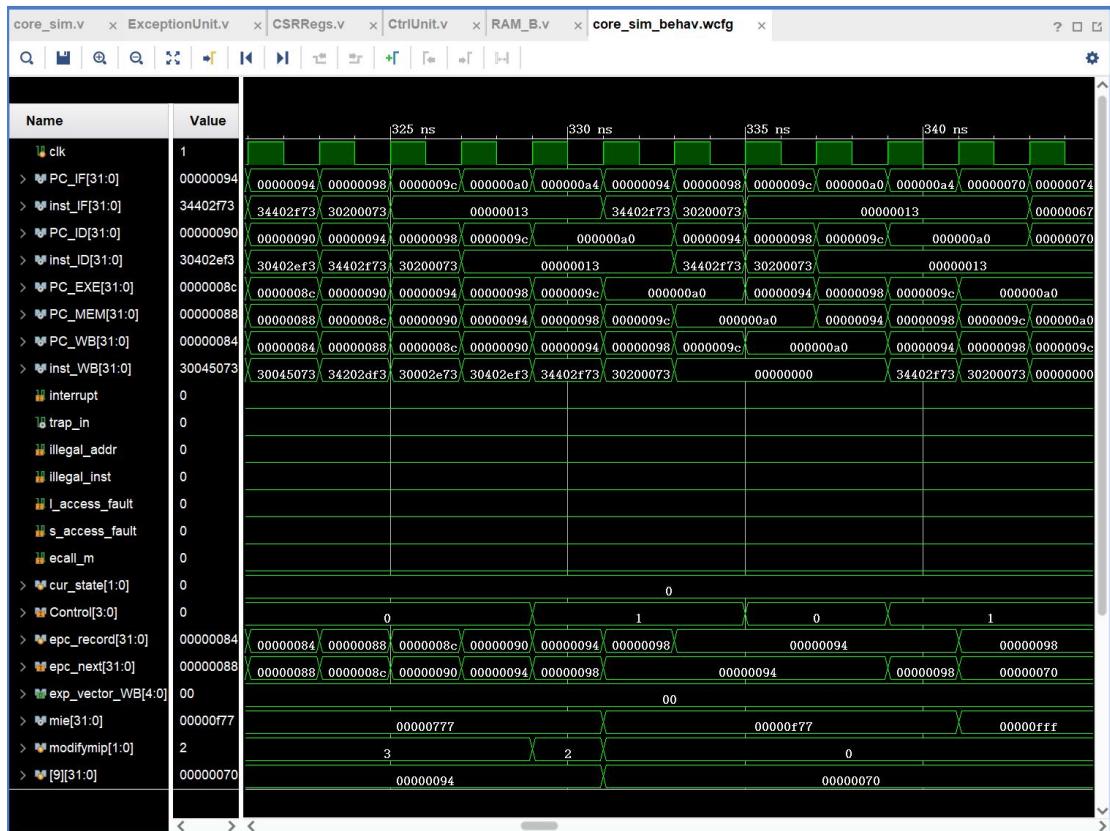
8. PC = 68 处发生软件中断、



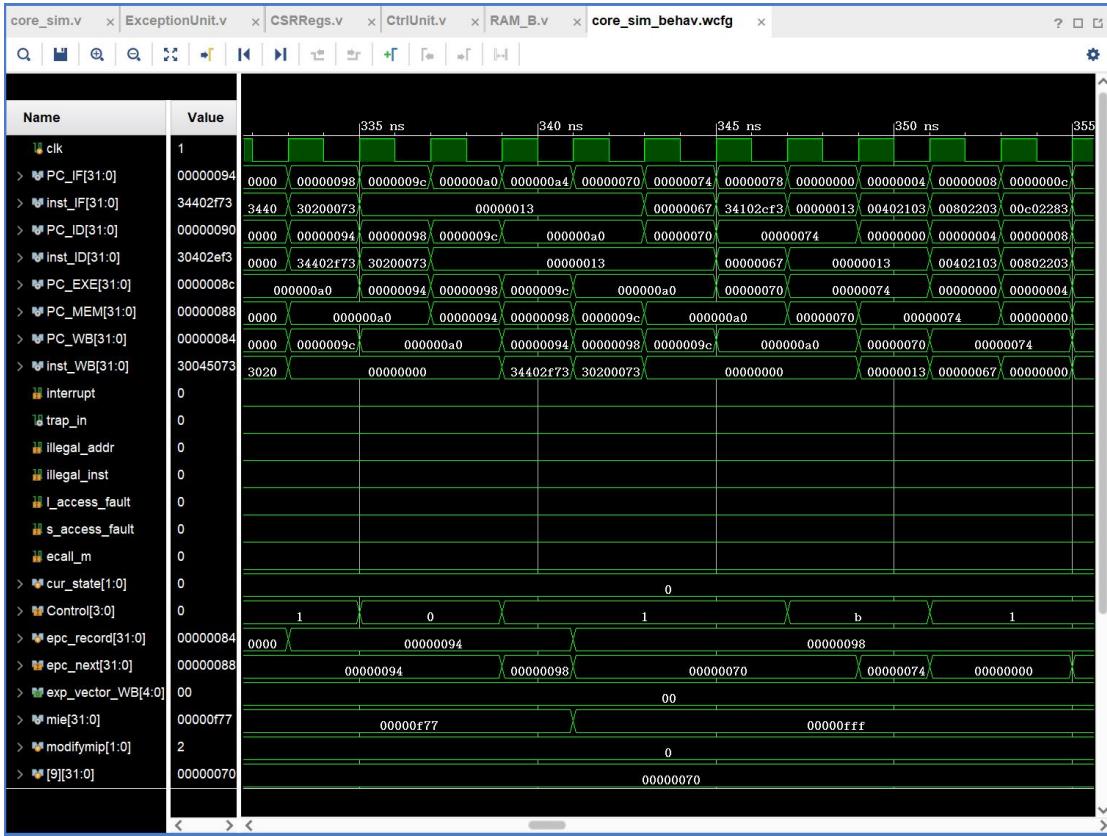
9. 在软件中断的处理程序中触发外部中断



10. 回退到软件中断:

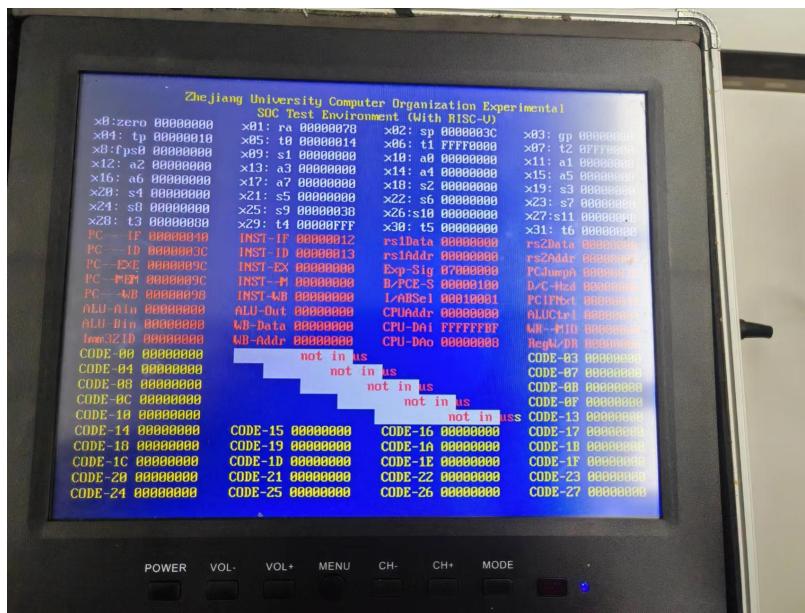
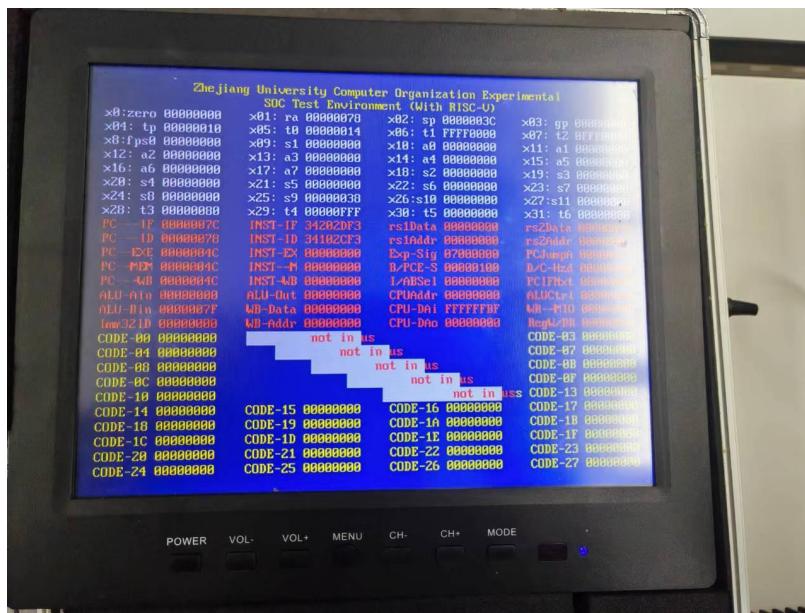


11. 回退到正常程序:



(1) 上板结果







四、讨论与心得

本次实验主要通过状态机实现了中断异常处理。实验让我对中断处理有了一个更深刻的学习和了解，尤其是系统的具体步骤和实现逻辑，这在未来会有很大的帮助。在实验过程当中由于由于测试代码的问题花费了很多的时间在找问题上，所幸最后得到了解决。

此外我们小组通过 bonus 的实现，对其他异常指令的处理和嵌套中断的实现有了更深刻的理解。