浙江大学

本科实验报告

课程名称：	计算机体系结构
姓 名：	张志心
学 院：	竺可桢学院
专 业：	混合班
学 号：	3210106357
指导教师：	常瑞
日 期：	2023 年 10 月 30 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：　实验2 – 流水线异常和中断设计

学生姓名：　张志心　　专业：　计算机科学与技术　　学号：　3210106357

同组学生姓名：　无　　指导教师：　常瑞　　助教：　邱明冉

实验地点：　曹光彪西301　　实验日期：　2023 年 10 月 30 日

1　实验目的

- 了解RISC-V简单的异常和中断
- 了解如何在流水线中添加异常和中断机制

2　实验环境

- **HDL**：Verilog、SystemVerilog
- **IDE**：Vivado
- **开发板**：NEXYS A7（XC7A100TCSG324）

3　实验要求

1. 实现csrrw, csrrs, csrrc, csrrwi, csrrsi, csrrci, ecall, mret指令
2. 实现mstatus, mtvec, mepc, mcause, mtval寄存器
3. 实现“异常和中断”节中列出的三种异常和**外部中断**，需要实现精确异常
4. 通过仿真测试和上板验证

4　实验步骤

1. 根据RISC-V非特权级手册和特权级手册在流水线内加入异常和中断机制；
   - `core/ExceptionUnit.v`

```
`timescale 1ns / 1ps

module ExceptionUnit(
    input clk, rst,
    input csr_rw_in,
    // write/set/clear (funct bits from instruction)
    input[1:0] csr_wsc_mode_in,
    input csr_w_imm_mux,
    input[11:0] csr_rw_addr_in,
```

```
    input[31:0] csr_w_data_reg,
    input[4:0] csr_w_data_imm,
    output[31:0] csr_r_data_out,
    input[4:0] mem_err_addr,
    input[31:0] illegal_inst_val,

    input interrupt,
    input illegal_inst,
    input l_access_fault,
    input s_access_fault,
    input ecall_m,

    input mret,

    input[31:0] epc_cur,
    input[31:0] epc_next,
    output[31:0] PC_redirect,
    output redirect_mux,

    output reg_FD_flush, reg_DE_flush, reg_EM_flush, reg_MW_flush,
    output RegWrite_cancel,
    output MemWrite_cancel
);
    // According to the diagram, design the Exception Unit
    // You can modify any code in this file if needed!
    wire[11:0] csr_waddr = trap ? 12'b1 : csr_rw_addr_in;
    reg[31:0] csr_wdata;
    reg csr_w;
    reg[1:0] csr_wsc;
    wire[11:0] csr_raddr = csr_rw_addr_in;

    wire[31:0] mstatus;
    wire[31:0] csr_rdata;
    wire[31:0] mtvec;
    wire[31:0] mepc;

    always @(*) begin
        if (trap) begin
            csr_w = 1'b1;
        end else begin
            csr_w = csr_rw_in;
        end
    end

    always @(*) begin
        if (trap) begin
            csr_wsc = 2'b01;
        end else begin
            csr_wsc = csr_wsc_mode_in;
        end
    end

    always @(*) begin
        if (trap) begin
            if (interrupt) begin
                csr_wdata = epc_cur;
```

```
            end else begin
                csr_wdata = epc_next;
            end
        end else begin
            csr_wdata = csr_w_imm_mux ?  {27'b0, csr_w_data_imm} :
csr_w_data_reg;
        end
    end

    wire trap = |{interrupt, illegal_inst, l_access_fault, s_access_fault,
ecall_m};
    reg[31:0] mtval_data;
    always @(*) begin
        if(illegal_inst) begin
            mtval_data = illegal_inst_val;
        end else if(l_access_fault | s_access_fault) begin
            mtval_data = {27'b0, mem_err_addr};
        end
        else begin
            mtval_data = 32'b0;
        end
    end

    CSRRegs
csr(.clk(clk),.rst(rst),.csr_w(csr_w),.raddr(csr_raddr),.waddr(csr_waddr),

.wdata(csr_wdata),.rdata(csr_rdata),.mstatus(mstatus),.csr_wsc_mode(csr_wsc),
        .mtvec(mtvec), .mepc(mepc), .mtval_data(mtval_data),
        .mtval_data_in(illegal_inst | l_access_fault | s_access_fault));

    assign csr_r_data_out = csr_rdata;

    assign PC_redirect = mret ? mepc : (trap ? mtvec : 32'h80000000);
    assign redirect_mux = mret | trap;
    assign MemWrite_cancel = trap;
    assign RegWrite_cancel = trap;
    assign reg_FD_flush = trap;
    assign reg_DE_flush = trap;
    assign reg_EM_flush = trap;
    assign reg_MW_flush = trap;

endmodule
```

- `core/CSRRegs.v`

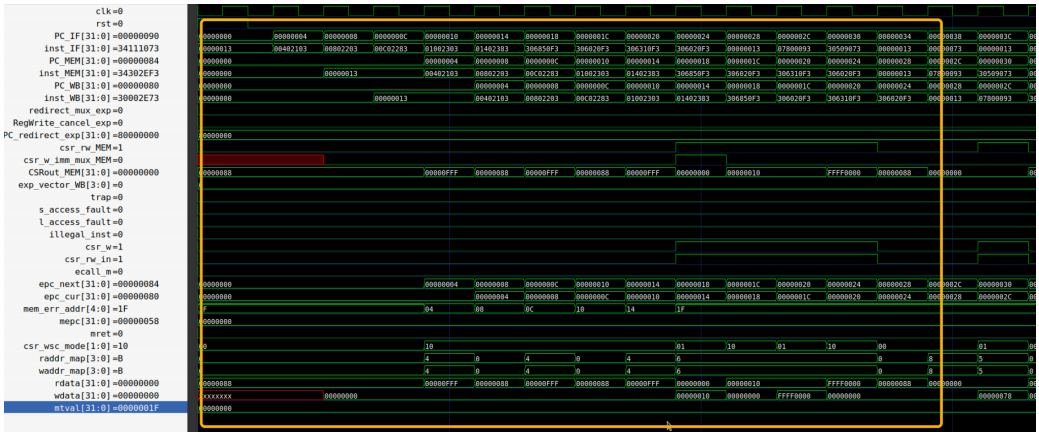
```
`timescale 1ns / 1ps

module CSRRegs(
    input clk, rst,
    input[11:0] raddr, waddr,
    input[31:0] wdata,
    input csr_w,
    input[1:0] csr_wsc_mode,
    input[31:0] epc_in,
```

```
    input[31:0] mtval_data,
    input mtval_data_in,

    output[31:0] rdata,
    output[31:0] mstatus,
    output[31:0] mepc,
    output[31:0] mtvec
);
    // You may need to modify this module for better efficiency

    reg[31:0] CSR [0:15];

    assign mepc = CSR[1];
    assign mtvec = CSR[5];

    // Address mapping. The address is 12 bits, but only 4 bits are used in
this module.
    wire raddr_valid = raddr[11:7] == 5'h6 && raddr[5:3] == 3'h0;
    wire[3:0] raddr_map = (raddr[6] << 3) + raddr[2:0];
    wire waddr_valid = waddr[11:7] == 5'h6 && waddr[5:3] == 3'h0;
    wire[3:0] waddr_map = (waddr[6] << 3) + waddr[2:0];

    assign mstatus = CSR[0];
    reg [31:0] mtval;
    reg [31:0] rdata_val;

    assign rdata = rdata_val;

    always@(*) begin
        if(raddr_map == 3) begin
            rdata_val = mtval;
        end else begin
            rdata_val = CSR[raddr_map];
        end
    end

    always@(posedge clk or posedge rst) begin
        if(rst) begin
            mtval <= 0;
        end
        else if(mtval_data_in) begin
            mtval <= mtval_data;
        end else if(csr_w & waddr_map == 3) begin
            case(csr_wsc_mode)
                2'b01: mtval <= wdata;
                2'b10: mtval <= mtval | wdata;
                2'b11: mtval <= mtval & ~wdata;
                default: mtval <= wdata;
            endcase
        end
    end

    always@(posedge clk or posedge rst) begin
        if(rst) begin
            CSR[0] <= 32'h88;
            CSR[1] <= 0;
```

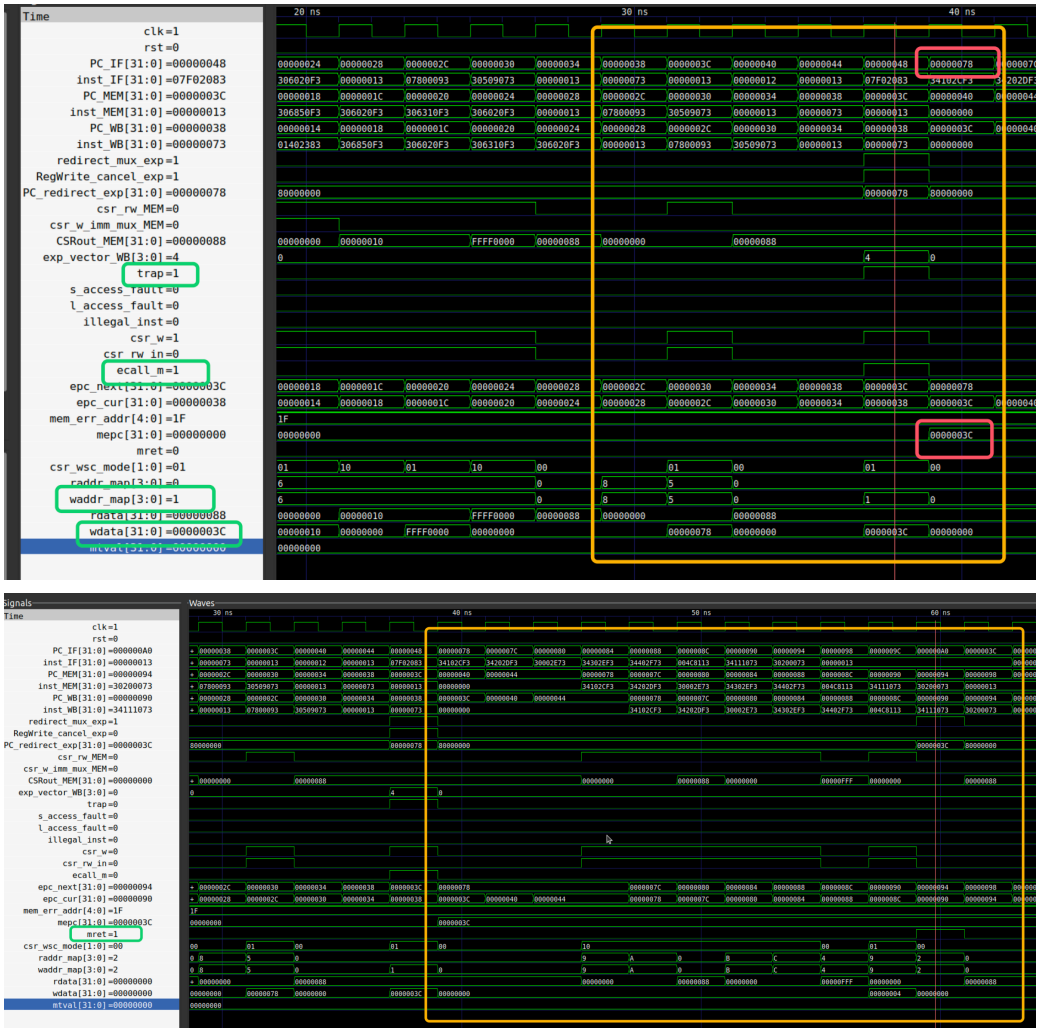
```
            CSR[2] <= 0;
            CSR[4] <= 32'hfff;
            CSR[5] <= 0;
            CSR[6] <= 0;
            CSR[7] <= 0;
            CSR[8] <= 0;
            CSR[9] <= 0;
            CSR[10] <= 0;
            CSR[11] <= 0;
            CSR[12] <= 0;
            CSR[13] <= 0;
            CSR[14] <= 0;
            CSR[15] <= 0;
        end
        else if(csr_w && waddr_map != 3) begin
            case(csr_wsc_mode)
                2'b01: CSR[waddr_map] <= wdata;
                2'b10: CSR[waddr_map] <= CSR[waddr_map] | wdata;
                2'b11: CSR[waddr_map] <= CSR[waddr_map] & ~wdata;
                default: CSR[waddr_map] <= wdata;
            endcase
        end
    end
endmodule
```

2. 在给定的SoC中，加入自己的CPU，通过仿真测试和上板验证。

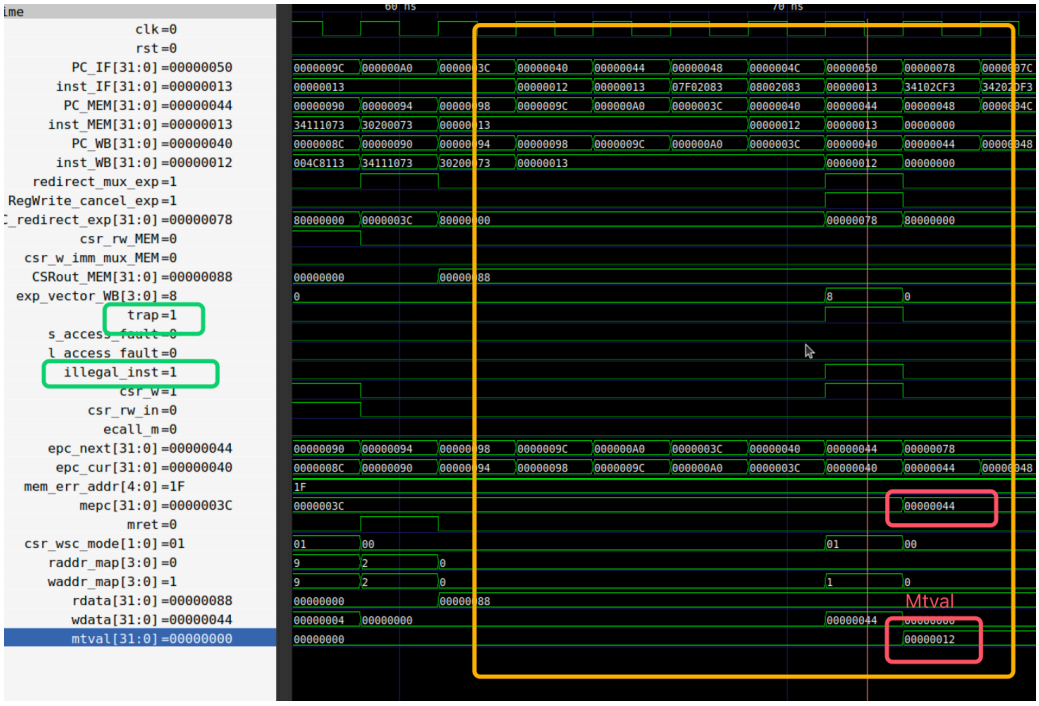
(a) PC from 00 to 34



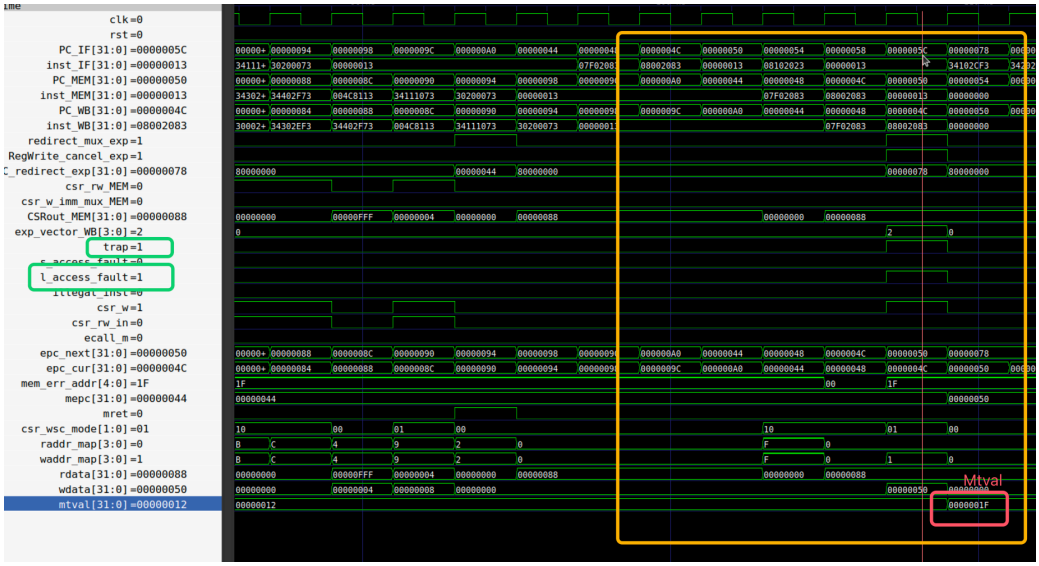
(b) Instruction = 00000073, PC = 38, ecall



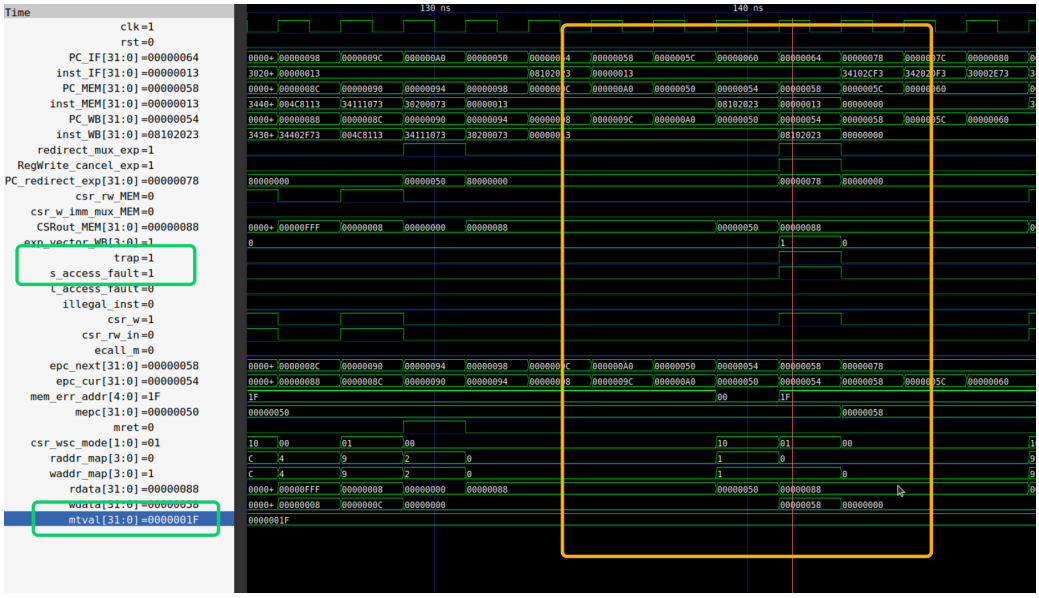
(c) **Instruction = 00000012, PC = 40, illegal inst**



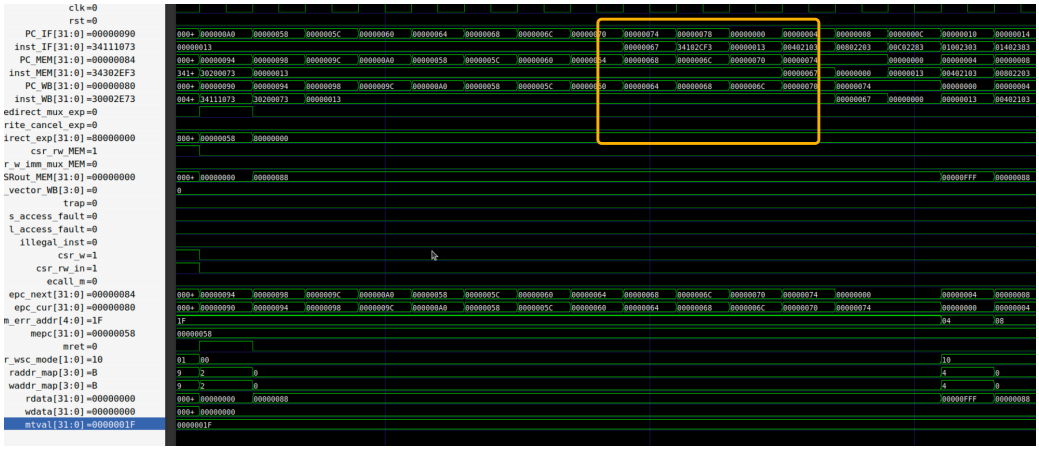
(d) **Instruction = 08002083, PC = 4C, L access fault**



(e) **Instruction = 08102023, PC = 54, S access fault**



(f) **Instruction = 00000067, PC = 74,** `jr x0`



5 思考题

1. 精确异常和非精确异常的区别是什么?

   异常场景上：精确异常是需要在异常处理完成后回到原处继续进行执行流的异常；非精确异常是在异常处理完成后结束程序运行，不再回到原处执行执行流的异常。

   流水线处理上：精确异常由于需要返回到原执行流，因此必须保证在异常触发前所有的指令都被执行，在异常触发后进入流水线的指令都被冲刷，特别是在多发射流水线中，需要多条流水线之间的额外同步；非精确异常不需要保证这一点。

2. 阅读测试代码，第一次导致trap的指令是哪条？trap之后的指令做了什么？如果实现了U mode，并以U mode从头开始执行测试指令，会出现什么新的异常？

   第一次导致 **trap** 的指令是 `0x38` 处的 **ecall**，如果回到 **U-mode** 执行指令，那么会在 `0x18` 处发生异常，因为用户模式无权执行 `cssrwi` 指令。

3. 为什么异常要传到最后一段即WB段后，才送入异常处理模块？可不可以一旦在某一段流水线发现了异常就送入异常处理模块，如果可以请说明异常处理模块应该如何处理异常；如果不可以，请说明理由。

   执行到 **WB** 段再处理异常是为了保证异常处理时，在异常指令之前的指令都被执行完成，并在此后再引入异常跳转目标需要执行的指令。从而使得 **CPU** 的状态对于异常前后的指令是正确的，否则，**CPU** 在异常处理切换状态时，对于异常发生前且还未执行完的指令，此时 **CPU** 的状态是错误的。