浙江大学

《计算机体系结构》
实验报告

实验题目	：	Lab2
姓　　名	：	王晓宇
学　　号	：	3220104364
电子邮箱	：	3220104364@zju.edu.cn
联系电话	：	19550222634
授课教师	：	姜晓红
助　　教	：	简英钊&黄鸿宇

2024 年 10 月 11日

Lab2 Pipelined CPU supporting exception & interrupt

Lab2 Pipelined CPU supporting exception & interrupt

课程名称：　计算机组成与设计　　实验类型：　综合

实验项目名称：Lab2：Pipelined CPU supporting exception & interrupt

学生姓名：　王晓宇　学号：　3220104364　同组学生姓名：　无

实验地点：　玉泉曹西301室　　实验日期：　2024 年 10 月 13 日

1　实验目的和要求

- 理解 CPU 异常和中断的原理及其处理程序，包括何时发生中断和异常，以及如何在硬件层面处理它们。
- 掌握支持异常和中断的流水线 CPU 的设计方法，以及跳转到异常处理程序的过程。
- 掌握支持异常和中断的流水线 CPU 的程序验证方法，包括使用仿真测试中断引发的信号和 CSR 寄存器的值。

2　实验内容和原理

> Tips：请简要解释CSR寄存器模块、Exception Unit模块以及Control Unit模块的实现思路

2.1　CSRReg.v实现思路

CSR寄存器模块用于实现控制状态寄存器（CSR）的读写操作，并处理与中断和异常相关的状态信息。

2.1.1　CSR寄存器的物理地址转换

模块内部定义了一个32位宽、16个元素的寄存器数组 `CSR`，用于存储不同的CSR寄存器值。特定的CSR寄存器通过数组索引进行访问，例如：

- `CSR[0]`：`mstatus` 寄存器
- `CSR[4]`：`mie` 寄存器

- `CSR[5]`：`mtvec` 寄存器
- `CSR[9]`：`mepc` 寄存器
- `CSR[10]`：`mcause` 寄存器
- `CSR[11]`：`mtval` 寄存器

由于CSR地址是12位的，但该模块只使用了其中的4位，因此需要进行**地址映射**，在CPU进行指令读写时依旧按照CSR编码来执行，而在寄存器模块中映射为特定的几个寄存器，减少了内存使用和方便仿真观察。

2.1.2 CSR寄存器的初始化

在复位信号有效时，所有CSR寄存器被初始化为默认值。例如：

- `mstatus` 被初始化为 `32'h88`。
- `mie` 被初始化为 `32'hfff`。

2.1.3 中断和异常处理

在中断信号有效时，模块会更新特定的CSR寄存器：

- 修改`mstatus` 的`MIE`位置零以禁用中断，并把先前的`MIE`值保留到`MPIE`中。
- `mepc`、`mcause` 和 `mtval` 被写入相应的值。

在返回指令信号（`mret`）有效时

- 模块会恢复 `mstatus` 的先前的MIE。

2.1.4 CSR写操作

当 `csr_w` 信号有效时，根据 `csr_wsc_mode` 信号的值执行不同的写操作：

- `2'b01`：直接写入 `wdata`。
- `2'b10`：将 `wdata` 与当前值进行按位或操作。
- `2'b11`：将 `wdata` 的按位取反值与当前值进行按位与操作。
- 默认：直接写入 `wdata`。

2.2 ExceptionUnit.v实现思路

`ExceptionUnit`模块用于处理异常和中断，并控制流水线的状态。以下是该模块的实现思路：

2.2.1 输入和输出信号

- 输入信号：
  - `csr_rw_in`、`csr_wsc_mode_in`、`csr_w_imm_mux`、`csr_rw_addr_in`、`csr_w_data_reg` 和 `csr_w_data_imm`：与控制状态寄存器（CSR）相关的信号及值，用于控制CSR的读写操作。
  - `interrupt`、`illegal_inst`、`l_access_fault`、`s_access_fault` 和 `ecall_m`：与中断和异常相关的信号。
  - `mret`：返回指令信号。
  - `epc_cur` 和 `epc_next`：当前和下一个程序计数器（PC）的值。
- 输出信号：
  - `csr_r_data_out`：从CSR读取的数据。
  - `PC_redirect` 和 `redirect_mux`：用于PC重定向的信号。
  - `reg_FD_flush`、`reg_DE_flush`、`reg_EM_flush`、`reg_MW_flush` 和 `RegWrite_cancel`：用于控制流水线状态的信号。

2.2.2 异常和中断处理

- 异常编码：
  - 根据输入信号 `illegal_inst`、`l_access_fault`、`s_access_fault` 和 `ecall_m`，生成异常编码 `exception_index`。
  - `illegal_inst`——2
  - `l_access_fault`——5
  - `s_access_fault`——7
  - `ecall_m`——11
  - 默认0
- 异常使能：

    - 通过 `mstatus` 寄存器的第3位MIE确定是否使能异常处理。
- **中断或异常检测**：
    - 通过组合异常信号和中断信号，生成 `interrupt_or_exception` 信号。

2.2.3 流水线控制

- **流水线刷新**：
    - `((illegal_inst||l_access_fault||s_access_fault||ecall_m)&&enable_exception)||interrupt`
    - 当检测到中断或异常时，刷新流水线的各个阶段（FD、DE、EM、MW），通过设置 `reg_FD_flush`、`reg_DE_flush`、`reg_EM_flush` 和 `reg_MW_flush` 信号。
- **写回取消**：
    - 当发生加载访问故障`l_access_fault`时，取消写回操作，通过设置 `RegWrite_cancel` 信号。
- **PC重定向**：`interrupt_or_exception | mret`
    - 根据 `mret` 信号和 `interrupt_or_exception` 信号，只要有一个发生，那么通过设置 `PC_redirect` 和 `redirect_mux` 信号，跳入trap。

2.2.4 CSR寄存器操作

- **CSR修改数据**：
    - `mepc_w` 的值异常发生时mepc更新为PC，中断发生时mepc被更新为PC+4
    - csr_wdata的值根据上述trap的类型设计mcause的值，保存到mcause，这里使用{}来进行拼接操作。
- **CSR读写地址**：
    - 通过 `csr_rw_addr_in` 设置CSR的读地址 `csr_raddr`。
    - 通过 `csr_rw_addr_in` 和 `csr_w_imm_mux` 设置CSR的写地址 `csr_waddr` 和写数据 `csr_wdata`。
- **CSR写操作**：
    - 根据 `csr_rw_in` 信号，确定是否进行CSR写操作，并设置写使能信号 `csr_w` 和写操作模式 `csr_wsc`。

2.2.5 CSRRegs模块实例化

- 实例化 `CSRRegs` 模块，用于实现CSR的具体读写操作，并传递相关信号，如 `mstatus`、`mtvec`、`mepc`、`mie` 等。

2.3 CtrlUnit.v实现思路

这里的模块实现和Lab1相似，这里举例说明不同之处：

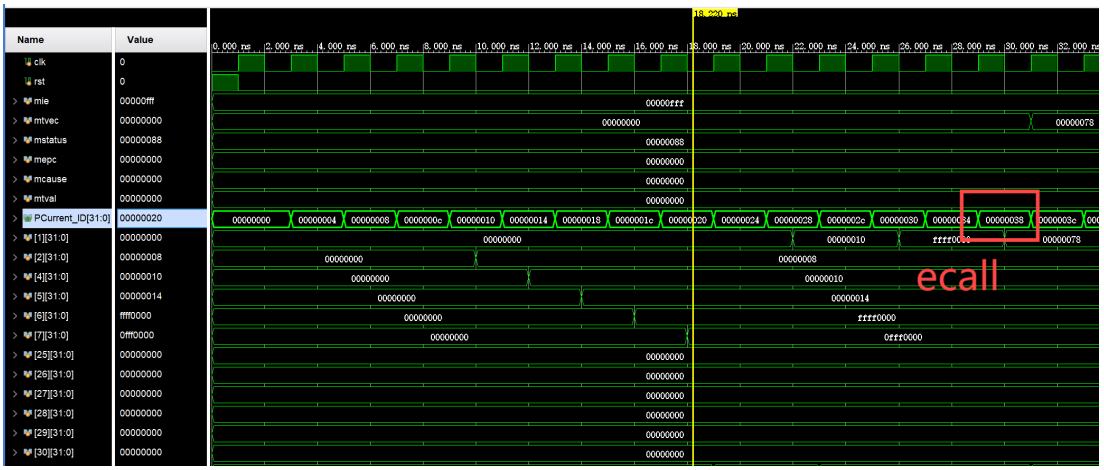
2.3.1 CSR信号引入

```
    wire CSRRW = CSRop & funct3_1;
    wire CSRRS = CSRop & funct3_2;
    wire CSRRC = CSRop & funct3_3;
    wire CSRRWI = CSRop & funct3_5;
    wire CSRRSI = CSRop & funct3_6;
    wire CSRRCI = CSRop & funct3_7;

    assign MRET = inst == 32'b0011000_00010_00000_000_00000_1110011;
    wire ECALL = inst == 32'b0111_0011;

    assign csr_rw = CSR_valid;
    assign csr_w_imm_mux = CSRRWI | CSRRSI | CSRRCI ;

    wire illegal_inst = ~(R_valid | I_valid | B_valid | JAL | JALR | L_valid | S_valid | LUI | AUIPC | CSR_valid | MRET | ECALL);
    assign exp_vector = {illegal_inst,ECALL};
```

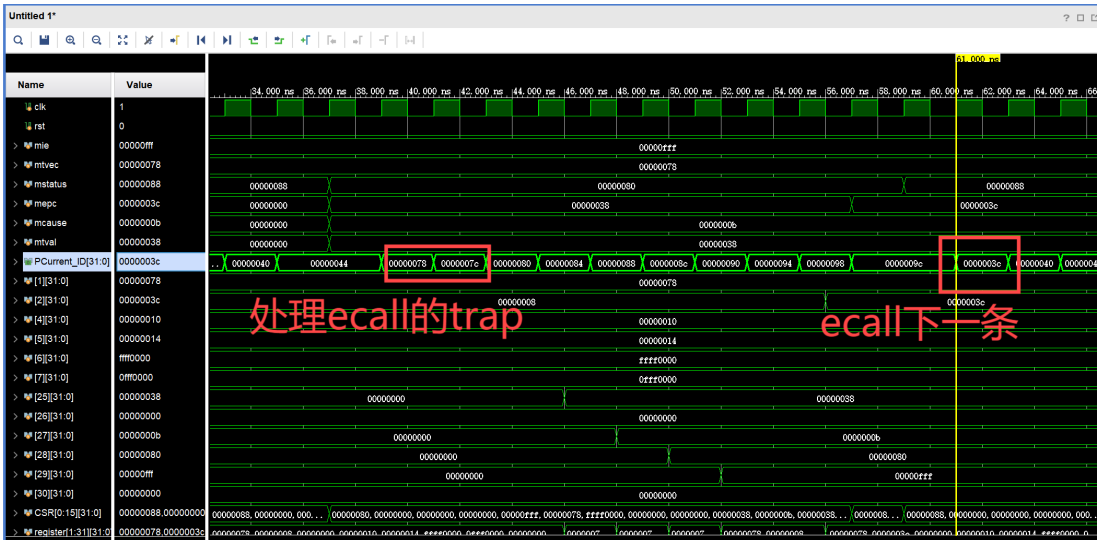
- 定义一些与控制状态寄存器（CSR）操作相关的信号，以及两个特定指令 `mret`、`ecall` 的检测信号
- 增加CSR指令mux选择信号，指明是 `CSRRWI` \ `CSRRSI` \ `CSRRCI` 信号之一
- 增加非法指令的检测，除了上述R|S等等指令之外的指令均为非法指令。
- 增加 `exp_vector` 信号，指明是 `illegal_inst`，`ECALL`，`l_access_fault`，`s_access_fault` 的哪一种。

3　实验过程和数据记录及结果分析

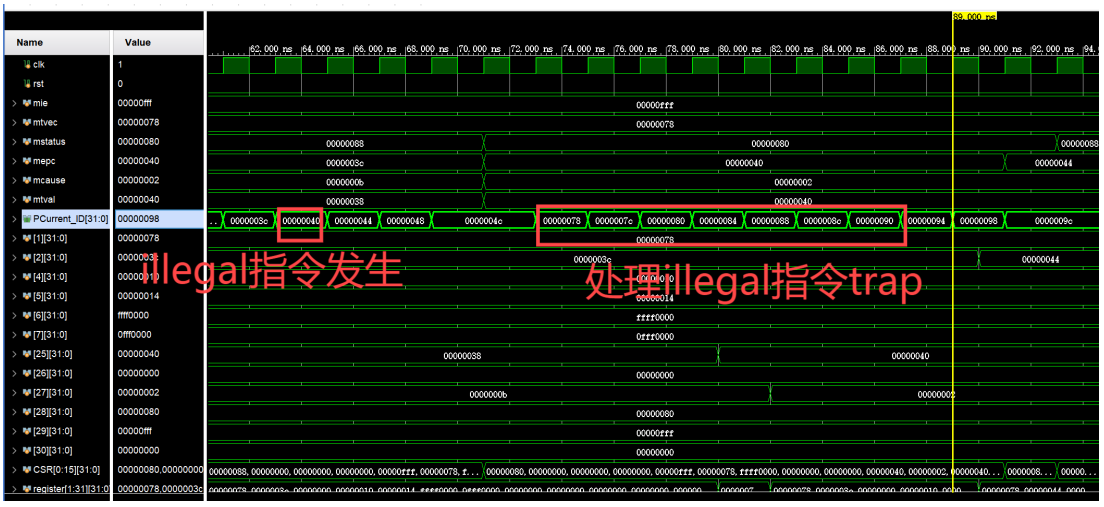
> Tips：请给出本次实验仿真的完整截图，并标注出ecall、csr指令、illegal指令、load/store访问错误、mret以及外部中断的位置（也就是实验得分点位置），并简要解释



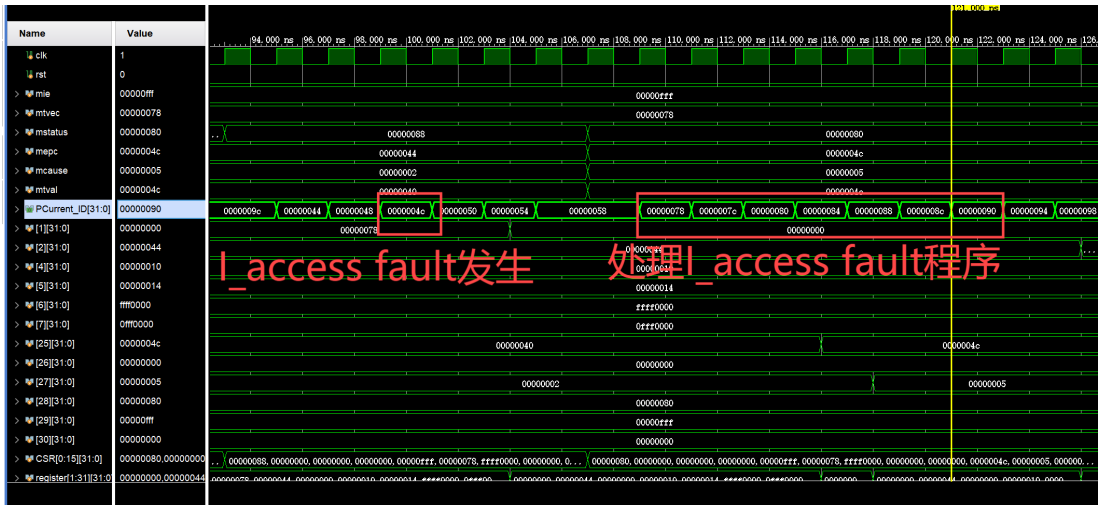
发生**ecall**在**WB**阶段，我们刷新其余流水线寄存器并进入**trap**处理



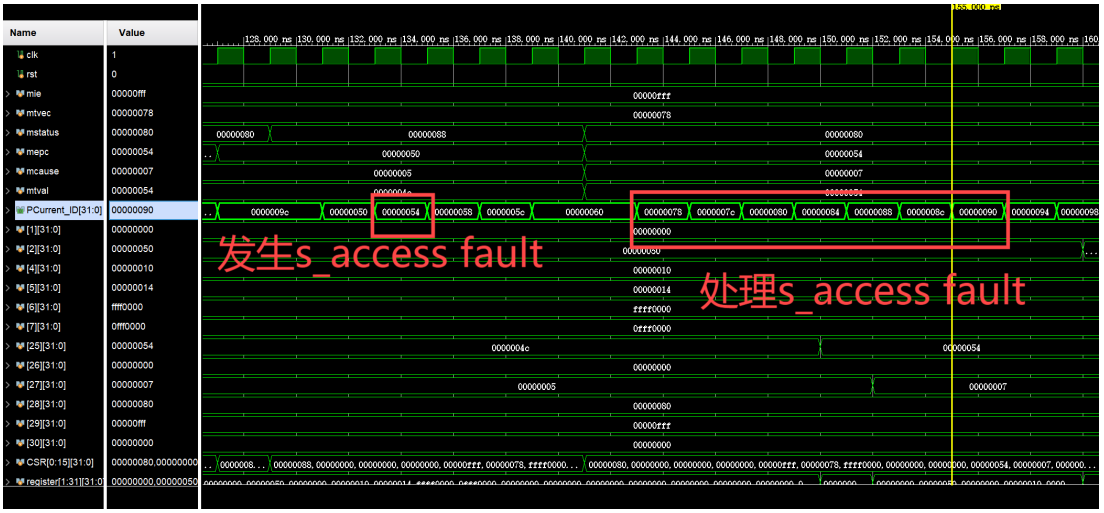
返回是**ecall**的下一条，表明处理完成，进行正常**pc**



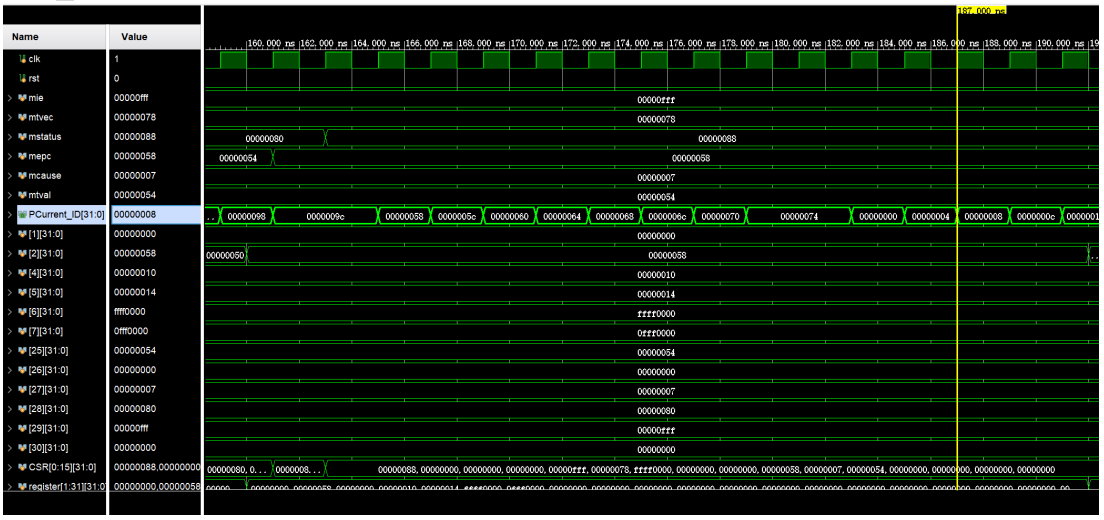
发生illegal在WB阶段，我们清空其余流水线寄存器并进入trap处理异常,返回是illegal的下一条，表明处理完成，进行正常pc



发生I_access_fault在WB阶段，我们清空其余流水线寄存器并进入trap处理异常，返回是I_access_fault的下一条，表明处理完成，进行正常pc



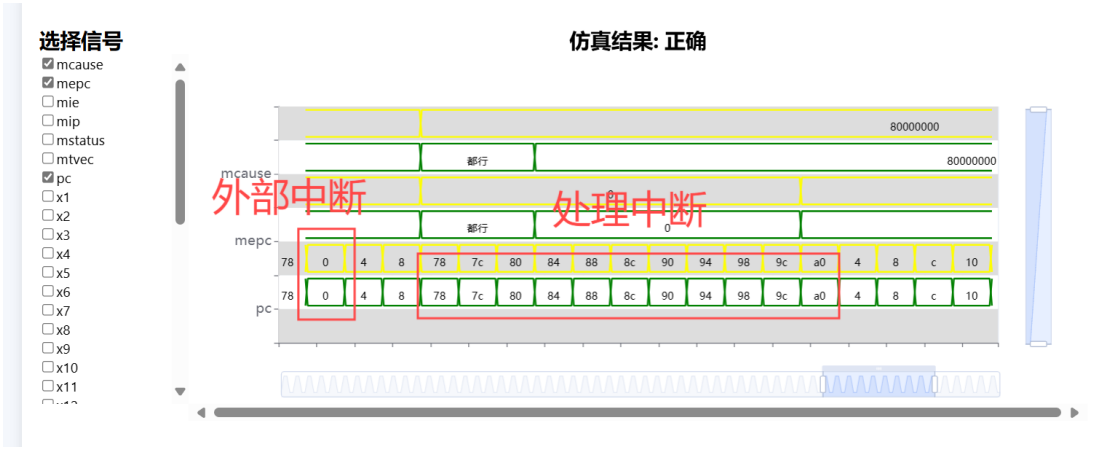
发生S_access_fault在WB阶段，我们清空其余流水线寄存器并进入trap处理异常，返回是S_access_fault的下一条，表明处理完成，进行正常pc



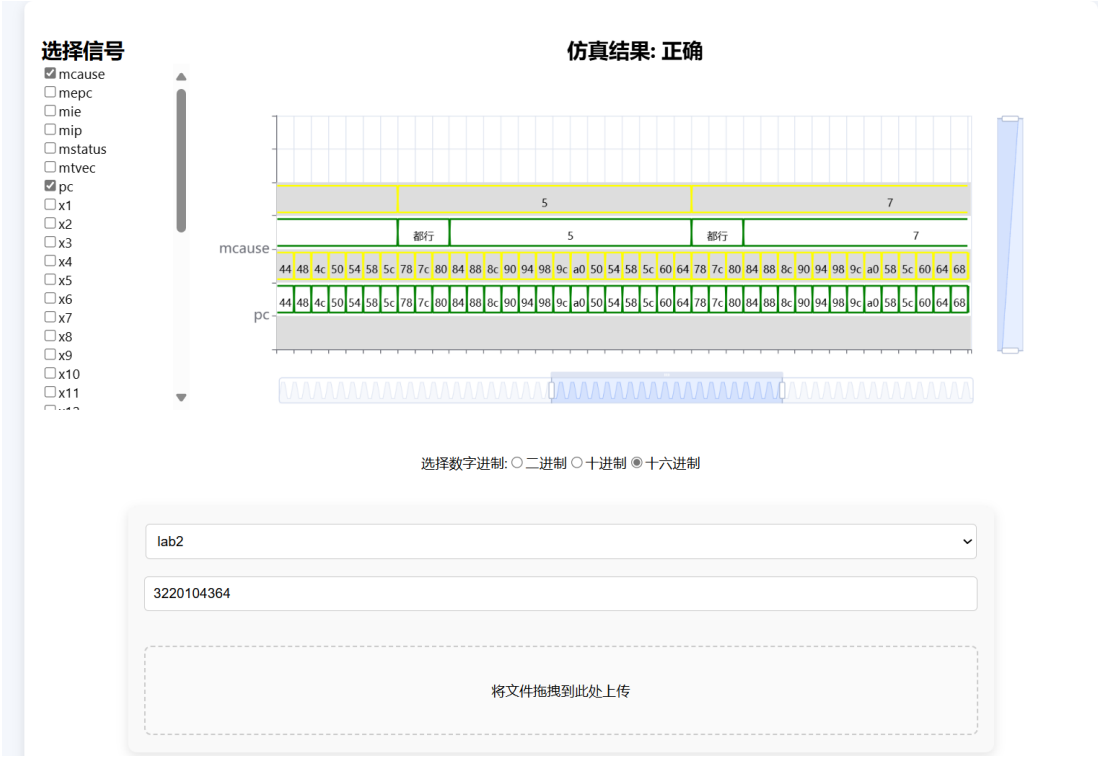
外部中断我们利用课程网站来进行仿真：

在第一轮结束后，`jr`指令跳回了第一条指令。

之后在0x0时进行了一次外部中断，`mcause == 0x80000000`表明的确是外部中断，如图所示：



以下为仿真通过示意图



4 讨论与心得

> Tips: 请写出对本次实验内容的深入讨论或者本次实验的心得体会。

设计思路在听了姜老师以及助教的补充说明之后清晰了不少，实验难度不大，用时不是很久，复习了特权级指令的内容以及处理中断异常的流程。