实验四 异常与中断

首先，更新代码仓库：

```
cd Arch-2022Spring

# 首先，commit 之前的代码。如果你已经 commit，就跳过下面两行
git add .
git commit -m "lab3"

git remote -v
# 如果地址是git://github.com/FDUCSLG/Arch-2022Spring-FDU.git，则执行这一步：
git remote set-url origin https://github.com/FDUCSLG/Arch-2022Spring-FDU.git

# 这一步如果卡很久，就 Ctrl-C 后重试
git fetch --all

git merge origin/master
# Merge 之后可以需要处理 Merge Conflict，在VS Code里可以很方便地处理
git submodule update
```

> 内存请求的地址没对齐，怎么办?
>
> 运行一段死循环，电脑会卡死吗？如果拿 lab3 的 CPU 去跑一段死循环代码，可能真的会卡死。

本次实验需要实现异常与中断。

0 基础知识

每台电脑上都运行了操作系统 （OS, Operating System）。操作系统提供了：

- 资源抽象、提供服务。`putchar` 函数对应什么操作？在我们的实验中，是往某个 uncached 地址写一个 8bit 的数据。对于不同的设备，这个地址可能不同（或许，重启电脑后，这个地址又变了）。操作系统给程序员提供了一个 `putchar` 的接口，程序员就不需要知道 `putchar` 是怎么实现的这种细节（我们下学期就会知道 `putchar` 做了什么）。
- 资源隔离、保障安全。我写的程序，放到电脑上跑，别的程序会不会篡改我的数据？别的程序会不会占着 CPU 导致我的程序跑不了？这时就需要 OS 来管理。

为了实现 OS 资源隔离、保障安全的功能，硬件需要有特权等级，分为用户态与内核态。类比用户程序能使用的通用寄存器（regfile）和内存，内核态也有一部分寄存器和内存，用户程序无法访问这些资源。本实验主要关注特权资源中的寄存器部分。

操作系统在两种情况下获得执行权，分为同步与异步。

同步的情况（称为异常）：执行这条指令时，立刻将执行权交给操作系统，从硬件的角度来看就是把 pc 设置为 OS 的处理函数。比如，遇到了 `ld a1, 1(zero)`，地址不对齐，是一个非法操作，这个指令不能执行，这时就要交给管理员来处理。异常未必是用户程序的恶意或过失行为，用户程序可以主动地通过异常切换到内核，来申请内核服务（比如 `putchar`），这种一般称为系统调用。

异步地情况（称为中断）：相比于异常的确定下陷，中断的下陷时机是不确定的，但中断也是必要的。为了避免死循环让电脑死机（以及其他种种原因），操作系统每隔一段时间就需要获得控制权。这就是时钟中断的作用，每隔一段时间就触发一次，让操作系统获得控制权。

1 需要维护的特权资源

需要实现 CSR 的 mstatus, mtvec, mip, mie, mscratch, mcause, mtval；选做：pmpcfg0, pmpaddr0

需要实现 mode 寄存器

1.1 CSR

详见中文手册第十章。

csr 可通过 csrrw 等指令进行写入。指令的具体行为详见中文手册的附录。

csr 作为寄存器，也会有数据冲突。但是，csr 不要转发！csr 的每次改变，都应刷新流水线（普通的写入，刷新流水线后，从 pc + 4 开始继续执行）！

发生异常或中断时，也会修改 csr。进行以下操作：

- `mepc <- pc`，`pc_nxt <- mtvec`
- 设置 mcause。`mcause[63] <- 1 if interrupt else 0`，`mcause[62:0] <- code`
- `mpie <- mstatus.mie`，`mstatus.mie <- 0`
- `mstatus.mpp <- mode`
- 清除流水线。取消 dreq.valid 和 ireq.valid

执行 mret 指令时，进行以下操作：

- `pc <- mepc`
- `mstatus.mie <- mstatus.mpie`
- `mstatus.mpie = 1'b1`
- `mstatus.mpp <- 2'b0`
- 清除流水线。取消 dreq.valid 和 ireq.valid

1.2 mode

初始状态下，mode 为 3。

触发异常或中断时，mode 变为 3。

执行 mret 指令时，mode 变为 `mstatus.mpp`。

1.3 实现方法

可参考我写的一些代码。包括 csr 编码、mstatus 域。

异常在流水线各个阶段都可以进行检测（比如，pc 对齐的检测可以放在 fetch 阶段），但处理都放在 Writeback 阶段。中断、mret 也在 writeback 执行。

pcselect 多了两个来源：mepc （mret 时），mtvec（遇到中断或异常时）。

需要清除整条流水线。

接入 `DifftestCSRSTATE`：

```
// core.sv 中已有 DifftestCSRState，需要接线
// 0 表示不用接
DifftestCSRState DifftestCSRState(
        .clock              (clk),
        .coreid             (0),
        .priviledgeMode     (csr.mode_nxt),
        .mstatus            (csr.regs_nxt.mstatus),
        .sstatus            (csr.regs_nxt.mstatus & 64'h800000030001e000),
        .mepc               (csr.regs_nxt.mepc),
        .sepc               (0),
        .mtval              (csr.regs_nxt.mtval),
        .stval              (0),
        .mtvec              (csr.regs_nxt.mtvec),
        .stvec              (0),
        .mcause             (csr.regs_nxt.mcause),
        .scause             (0),
        .satp               (0),
        .mip                (csr.regs_nxt.mip),
        .mie                (csr.regs_nxt.mie),
        .mscratch           (csr.regs_nxt.mscratch),
        .sscratch           (0),
        .mideleg            (0),
        .medeleg            (0)
);
```

2 需要实现的指令与异常中断类型

CSRRW CSRRS CSRRC CSRRWI CSRRSI CSRRCI MRET ECALL

需要支持的异常类型：

- 指令地址不对齐
- 数据地址不对齐（读、写）
- 非法指令
- ecall

需要支持的中断类型：

- 时钟中断
- 软件中断

3 测试方法

3.1 Verilator 仿真

检测是否通过：

- **使用 `make test-lab4 TEST=all`，查看是否有 `HIT GOOD TRAP`。**
- 选做：`make test-lab4full TEST=all`
  - 选做不加分
  - 选做内容和操作系统课程有一定关联，操作系统的实验有可能使用 riscv 架构。大家可以在假期里自学一下 riscv 的 privilege 文档。

4 作业提交

DDL：**6月12日23:59**

提交内容：以学号命名的`tar`压缩包，如`18307130024.tar`，包括 `vsrc` 和 `report.pdf`。

生成方法：

```
mkdir 18307130024
cp -r vsrc 18307130024
cp report.pdf 18307130024
tar cf 18307130024.tar 18307130024
```

你需要通过 Verilator 仿真。

实验报告需要包括：

- 流水线的改动
- 测试通过的截图

实验报告里不应有大段代码的复制。

通过测试+实验报告有上述内容，本次实验就可以满分。

实验报告里可以有：

- 对本门课程实验（文档、代码风格、视频录制等）的建议
- 对后续内容的期待