Lab 2: RISC-V64 时钟中断处理

1 实验简介

学习在RISC-V上的异常处理相关机制，以时钟中断为例，编写时钟中断处理函数。

2 实验环境

- Docker Image

3 背景知识

3.1 什么是异常(trap)

异常(trap)是指是不寻常的运行时事件，由硬件或软件产生，当异常产生时控制权将会转移至异常处理程序。异常是操作系统最基础的概念，一个没有异常的操作系统无法进行正常交互。

RISC-V将异常分为两类。一类是硬件中断(interrupt)，它是与指令流异步的外部事件，比如鼠标的单击。另外一类是同步异常(exception)，这类异常在指令执行期间产生，如访问了无效的存储器地址或执行了具有无效操作码的指令时。

这里我们用异常(trap)作为硬件中断(interrupt)和同步异常(exception)的集合，另外trap指的是发生硬件中断或者同步异常时控制权转移到handler的过程。

> 后文统一用异常指代trap，中断/硬件中断指代interrupt，同步异常指代exception

3.2 Machine Mode下的异常

3.2.1 异常需要的寄存器

Machine mode异常需要使用的寄存器首先有lab1提到的mstatus，mip，mie，mtvec寄存器，这些寄存器需要我们操作；剩下还有mepc，mcause寄存器，这些寄存器在异常发生时硬件会自动置位，它们的功能如下：

- mepc（Machine Exception Program Counter）：通常指向异常处理后应该恢复执行的位置
- mcause（Machine Cause Register）：保存异常的种类，具体可以查看请参考The RISC-V Instruction Set Manual 3.1.16节（P.40）

事实上，异常还与mideleg和medeleg两个寄存器密切相关，它们的功能将在**Supervisor mode下的异常**部分讲解。

与时钟中断相关的还有mtime和mtimecmp寄存器，它们的功能如下：

- mtime（Machine Time Register）：保存时钟计数，这个值会由硬件自增
- mtimecmp（Machine Time Compare Register）：保存需要比较的时钟计数，当mtime的值大于或等于mtimecmp的值时，时钟中断触发

需要注意的是，mtime和mtimecmp寄存器需要用MMIO的方式即使用内存访问指令（sd，ld等）的方式交互，可以将它们理解为Machine mode下的一个外设。

3.2.2 硬件中断的处理（以时钟中断为例）

简单地来说，中断处理经过了三个流程，中断触发，判断可以处理还是忽略，可以处理时调用处理函数。

1. 中断触发：

   时钟中断的触发条件是这个hart（硬件线程）的时间比较器mtimecmp小于实数计数器mtime。

2. 判断是否可处理：

当时钟中断触发时，并不一定会响应中断信号。Machine mode只有在全局中断使能位mstatus[mie]置位时才会产生中断，如果在Supervisor mode下触发了Machine mode的中断，此时无视mstatus[mie]直接响应，即运行在低权限模式下，高权限模式的全局中断使能位一直是enable状态。此外，每个中断在控制状态寄存器mie中都有自己的使能位，对于特定中断来说，需要考虑自己对应的使能位，而控制状态寄存器mip中又指示目前待处理的中断。以时钟中断为例，只有当mstatus[mie]=1，mie[mtie]=1，且mip[mtip]=1时，才可以处理机器的时钟中断。其中mstatus[mie]以及mie[mtie]需要我们自己设置，而mip[mtip]在中断触发时会被硬件自动置位。

3. 调用处理函数：

当满足对应中断的处理条件时，硬件首先会发生一些状态转换（参考中文手册10.3节），并跳转到对应的异常处理函数中，在异常处理函数中我们可以通过分析异常产生的原因判断具体为哪一种，然后执行对应的处理。为了处理异常结束后不影响hart正常的运行状态，我们首先需要保存当前的状态即上下文切换。我们可以先用栈上的一段空间来把**全部**寄存器保存，保存完之后执行到我们编写的异常处理函数主体，结束后退出。函数调用规范可以参考中文开源手册3.2节。

3.2.3 同步异常的处理

同步异常的触发条件是当前指令执行了未经定义的行为，比如Illegal instruction，也没有判断可以处理还是忽略的步骤，硬件会直接经历一些状态转换，然后跳到对应的异常处理函数。又比如环境调用同步异常ecall主要是用于低权限的mode需要高权限的mode的相关操作时使用的，比如U-mode call S-mode 通常用于使用系统调用，而S-mode call M-mode通常用于在S-mode操作某些硬件，这在本实验会用到。

需要注意的是，不管是中断还是同步异常，都会经历相似的硬件状态转换，并跳到同一个异常处理地址（由mtvec/stvec指定），再由处理函数分析异常出现原因并进行不同的处理。

3.3 Supervisor Mode下的异常

由于hart位于Supervisor mode，我们需要在Supervisor mode下处理异常。这时首先要提到委托（delegation）机制。

3.3.1 委托（delegation）

RISC-V架构所有mode的异常在默认情况下都跳转到Machine mode处理。为了提高性能，RISC-V支持将低权限mode产生的异常委托给对应mode处理。这里使用了mideleg和medeleg两个寄存器，它们的功能如下：（更具体的介绍参考The RISC-V Instruction Set Manual 3.1.8节）

- mideleg（Machine Interrupt Delegation）：控制将哪些异常委托给对应mode处理，它的结构可以参考mip寄存器，比如将mip中stip对应的位置位会将Supervisor mode的时钟中断委托给Supervisor mode处理
- medeleg（Machine Exception Delegation）：控制将哪些异常委托给对应mode处理，它的各个位对应mcause寄存器的返回值

3.3.2 Supervisor Mode下时钟中断处理流程

事实上，虽然在mideleg中设置了将Supervisor mode产生的时钟中断委托给Supervisor mode，委托并没有完成。因为硬件产生的时钟中断仍会发到Machine mode（mtime寄存器是Machine mode的设备），所以我们需要手动触发Supervisor mode下的时钟中断。

此前，假设设置好[m|s]status以及[m|s]ie，即我们已经满足了时钟中断在两种mode下触发的使能条件。接下来一个时钟中断的委托流程如下：

1. 当mtimecmp小于mtime时，触发时钟中断并且硬件自动置位mip[mtip]。
2. 此时mstatus[mie]=1，mie[mtie]=1，且mip[mtip]=1 表示可以处理machine mode的时钟中断。
3. 此时hart发生了异常，硬件会自动经历状态转换，其中pc被设置被mtvec，即跳转到我们设置好的machine mode处理函数入口。
4. machine mode处理函数分析异常原因，判断为时钟中断，为了将时钟中断委托给supervisor mode，于是将mip[stip]置位，并且为了防止在supervisor mode处理时钟中断时继续触发machine mode时钟中断，于是同时将mie[mtie]清零。
5. machine mode处理函数处理完成并退出，此时sstatus[sie]=1，sie[stie]=1，且sip[stip]=1(由于sip是mip的子集，所以第4步中令mip[stip]置位等同于将sip[stip]置位)，于是触发supervisor mode的时钟中断。
6. 此时hart发生了异常，硬件自动经历状态转换，其中pc被设置为stvec，即跳转到我们设置好的supervisor mode处理函数入口。

7. supervisor mode处理函数分析异常原因，判断为时钟中断，于是进行相应的操作，完成后将sip[stip]清零，表示处理完毕，然后利用ecall触发异常，跳转到machine mode的异常处理函数进行最后的收尾。
8. machine mode异常处理函数分析异常原因，发现为ecall from S-mode，于是设置mtimecmp+=100000，并且设置mie[mtie]恢复machine mode的中断使能，保证下一次时钟中断可以触发。
9. 函数逐级返回，整个委托的时钟中断处理完毕。

4 实验步骤

4.1 环境搭建

4.1.1 建立映射

同lab1的文件夹映射方法，目录名为lab2。

4.1.2 组织文件结构

文件结构如下：

```
lab2
├── arch
│   └── riscv
│       ├── include
│       │   └── put.h
│       ├── kernel
│       │   ├── entry.S
│       │   ├── head.S
│       │   ├── Makefile
│       │   ├── strap.c
│       │   └── vmlinux.lds
│       └── Makefile
├── include
│   ├── put.h
│   └── test.h
├── init
│   ├── main.c
│   ├── Makefile
│   └── test.c
├── lib
│   ├── Makefile
│   └── put.c
└── Makefile
```

其中put.h,put.c,test.h,test.c,main.c会提供给同学们。将lab1中实现的Makefile放置到对应目录下，并在lib目录新建Makefile，将lib目录下的文件纳入到整个编译的工程管理中。

4.1.3 文件必要修改

由于裸机程序需要在.text段起始位置执行，所以需要利用vmlinux.lds中.text段的定义来确保head.S中的.text段被放置在其他.text段之前。具体的做法如下：

1. 首先修改head.S中的.text命名为.text.init

```
<<<<< before
.section .text
============
.section .text.init
>>>>> after
```

2. 修改entry.S中的.text命名为.text.entry

```
<<<<< before
.section .text
============
.section .text.entry
>>>>> after
```

3. 修改lds文件中的.text展开方式

```
<<<<< before
.text : { *(.text) }
============
.text : {
        *(.text.init)
        *(.text.entry)
        *(.text)
}
>>>>> after
```

4.2 head.S模式切换前添加功能

4.2.1 初始化bss段

确认vmlinux.lds中有对bss段的定义，在head.S模式切换之前初始化bss段的内存。

> 思考题：观察vmlinux和image，解释为什么要初始化bss段。

4.2.2 初始化mtimecmp寄存器

将mtimecmp寄存器的值初始化为mtime+1000000。其中mtime对应的映射地址是0x200bff8，mtimecmp对应的映射地址是0x2004000。参考这里（内网访问）

4.2.3 设置时钟中断委托

首先需要设置mideleg对应位来打开时钟中断的委托，除此以外需要设置[m|s]status以及[m|s]ie来打开时钟中断的使能（具体为设置mstatus[mie],sstatus[sie],mie[mtie],sie[stie]）。

4.3 编写machine mode的异常处理代码

4.3.1 上下文切换

lab1中我们设置mtvec指向一个空的trap_m函数。在此trap_m函数中，首先我们需要利用sp开辟一段空间，并保存所有寄存器以及必要的CSRs的值，并且在mret之前恢复所有寄存器的值，并设置sp回收空间。

4.3.2 编写处理代码

保存寄存器之后，我们需要编写对应的异常处理代码，这里我们只需处理两种异常。一种是时钟中断，另外一种是ecall from S-mode。

在处理时钟中断时，需要完成以下功能：

- disable mie[mtie]，即禁用时钟中断，避免之后Supervisor mode处理时钟中断同时继续触发时钟中断。

- enable sip[stip]，设置Supervisor mode timer interrupt的pending位，为之后触发Supervisor mode下的时钟中断做准备。

在处理ecall from S-mode时，需要完成以下功能：

- set mtimecmp += 100000 此时设置mtimecmp同时硬件会clear mip[mtip]
- enable mie[mtie] 恢复machine mode的时钟中断使能

思考题：当处理同步异常时应该在退出前给mepc+4，当处理中断时则不需要，请解释为什么要这样做。

4.4 编写Supervisor mode的异常处理代码entry.S

4.4.1 上下文切换

将head.S中的trap_s函数（即Supervisor mode的异常处理函数）移动到entry.S中，并首先保存所有寄存器以及必要的CSRs的值。

4.4.2 异常处理

对异常进行处理，仅需实现对时钟中断的处理，功能是输出已经产生的时钟中断的个数。（为了方便测试，请每产生100000个时钟中断输出一行显示信息）

4.5 编译及测试

仿照lab1进行调试，预期的实验结果如下：(请对test.c做修改，确保输出自己的组号，例第4组修改XX为04)

```
ZJU OS LAB 2             GROUP-XX
[S] Supervisor Mode Timer Interrupt 0
[S] Supervisor Mode Timer Interrupt 1
[S] Supervisor Mode Timer Interrupt 2
[S] Supervisor Mode Timer Interrupt 3
[S] Supervisor Mode Timer Interrupt 4
[S] Supervisor Mode Timer Interrupt 5
[S] Supervisor Mode Timer Interrupt 6
[S] Supervisor Mode Timer Interrupt 7
[S] Supervisor Mode Timer Interrupt 8
[S] Supervisor Mode Timer Interrupt 9
[S] Supervisor Mode Timer Interrupt 10
[S] Supervisor Mode Timer Interrupt 11
[S] Supervisor Mode Timer Interrupt 12
[S] Supervisor Mode Timer Interrupt 13
[S] Supervisor Mode Timer Interrupt 14
[S] Supervisor Mode Timer Interrupt 15
[S] Supervisor Mode Timer Interrupt 16
[S] Supervisor Mode Timer Interrupt 17
[S] Supervisor Mode Timer Interrupt 18
[S] Supervisor Mode Timer Interrupt 19
[S] Supervisor Mode Timer Interrupt 20
[S] Supervisor Mode Timer Interrupt 21
[S] Supervisor Mode Timer Interrupt 22
[S] Supervisor Mode Timer Interrupt 23
[S] Supervisor Mode Timer Interrupt 24
```

Hint：如果觉得直接完成实验困难，可以先完成machine mode下的时钟中断处理，再完成supervisor mode下的时钟中断处理。

5 实验任务与要求

请仔细阅读背景知识，确保理解RISC-V异常委托与异常处理机制，并按照实验步骤完成实验，撰写实验报告，需提交实验报告以及整个工程的压缩包。

- 实验报告：
  - 各实验步骤截图以及结果分析
  - 回答思考题
  - 实验结束后心得体会
  - 对实验指导的建议（可选）
- 工程文件
  - 所有source code（确保make clean）
- 最终目录
  - 将Lab2_319010XXXX目录压缩并打包（若分组，则一组只需要一位同学提交）

```
Lab2_319010XXXX
├── report
│   └── 319010XXXX.pdf（若分组，所有组员学号在报告内部声明）
└── source
    ├── arch
    │   └── riscv
    │       ├── include
    │       │   └── put.h
    │       ├── kernel
    │       │   ├── entry.S
    │       │   ├── head.S
    │       │   ├── Makefile
    │       │   ├── strap.c
    │       │   └── vmlinux.lds
    │       └── Makefile
    ├── include
    │   ├── put.h
    │   └── test.h
    ├── init
    │   ├── main.c
    │   ├── Makefile
    │   └── test.c
    ├── lib
    │   ├── Makefile
    │   └── put.c
    └── Makefile
```