实验五-RV64 时钟中断处理

姓名:王晶晶 学号:3200104880 学院:计算机科学与技术学院 课程名称：计算机系统Ⅱ

实验时间：2021.12.14 实验地点：紫金港机房中心 指导老师：申文博

一、实验目的和要求

- 学习 RISC-V 的异常处理相关寄存器与指令，完成对异常处理的初始化。
- 理解 CPU 上下文切换机制，并正确实现上下文切换功能。
- 编写异常处理函数，完成对特定异常的处理。
- 调用 OpenSBI 提供的接口，完成对时钟中断事件的设置。

二、实验内容和原理

2.1 中断(Interrupt)和异常(Exception)

中断是由硬件触发的非同步的对外部设备的请求。属于正常的请求，不会影响程序的正常运行。

异常是由软件触发的同步的内部请求。一般来说是由于设备出现异常状况引发的，如非法指令、非法地址和溢出等。属于异常事件，常常导致程序的退出。

2.2 控制状态寄存器(Control and Status Registers)

本次实验在Supervisor Mode(S Mode)下处理中断，与本次实验相关的控制状态寄存器主要有以下几个。

- sstatus(Supervisor Status Register)中存在一个SIE(Supervisor Interrupt Enable)比特位，当该比特位设置为1时，会对所有的S态异常响应，否则将会禁用所有S态异常。
- sie(Supervisor Interrupt Eable Register)。在RISC-V中，Interrupt被划分为三类Software Interrupt,Timer Interrupt，External Interrupt。在开启了sstatus[SIE]

之后，系统会根据sie 中的相关比特位来决定是否对该Interrupt进行处理。

- stvec(Supervisor Trap Vector Base Address Register)即所谓的“中断向量表基址”。stvec有两种模式：Direct模式，适用于系统中只有一个中断处理程序，其指向中断处理入口函数（本次实验中我们所用的模式）。Vectored模式，指向中断向量表，适用于系统中有多个中断处理程序。
- scause(Supervisor Cause Register)，会记录异常发生的原因，还会记录该异常是Interrupt还是Exception。
- sepc(Supervisor Exception Program Counter)，会记录触发异常的那条指令的地址。

2.3 上下文切换(Context Switch)

由于在处理异常时，有可能会改变系统的状态。所以在真正处理异常之前，有必要对系统的当前状态进行保存，在异常处理完成之后，再将系统恢复至原先的状态，就可以确保之前的程序继续正常运行。这里的系统状态通常是指寄存器，这些寄存器也叫做CPU的上下文(Context)。

2.4 异常处理程序

异常处理程序根据scause的值，进入不同的处理逻辑，在本次试验中仅有Superviosr Timer Interrupt。

2.5 时钟中断

时钟中断需要CPU硬件的支持。CPU以"时钟周期"为工作的基本时间单位，对逻辑门的时序电路进行同步。而时钟中断实际上就是“每隔若干个时钟周期执行一次的程序”。下面是与时钟中断相关的寄存器以及时钟中断的产生。

- mtime与mtimecmp(Machine Timer Register)。mtime是一个实时计时器，由硬件以恒定的频率自增。mtimecmp中保存着下一次时钟中断发生的时间点，当mtime的值大于或等于mtimecmp的值，系统就会触发一次时钟中断。因此我们只需要更新mtimecmp中的值，就可以设置下一次时钟中断的触发点。本实验中，OpenSBI提供了更新mtimecmp的接口sbi_set_timer。
- mcounteren(Counter-Enable Registers)。由于mtime是属于M态的寄存器，本次实验中在S态无法直接对其读写，借助OpenSBI在M态通过设置mcounteren寄存器的

TM比特位，可以在S态中可以通过time这个只读寄存器读取mtime的当前值，相关汇编指令是rdtime。

三、主要仪器设备

- Docker in Lab3

四、操作方法与实现步骤

4.1 实验步骤

1、准备工程
2、开启异常处理
3、实现上下文切换
4、实现异常处理函数
5、实现时钟中断相关处理函数
6、编译及测试

4.2 实验结果

4.2.1 head.S

```
.extern start_kernel
.extern _traps
.extern clock_set_next_event
    .section .text.init
    .globl _start
_start:
    #set stvec to _traps
    la t0,_traps
    #add t0,t0,t0
    #add t0,t0,t0
    csrw stvec,t0

    #set sie[STIE]=1
    csrr t1,sie
    addi t2,x0,32
    or t1,t1,t2
    csrw sie,t1

    #set first time interrupt

    #call clock_set_next_event
    rdtime t0
    li t1,10000000
    add a0,t1,t0
    add a1,x0,x0
    add a2,x0,x0
    add a3,x0,x0
    add a4,x0,x0
    add a5,x0,x0
    add a7,x0,x0
    add a6,x0,x0
    ecall

    #set sstatus[sie]=1
    csrr t1,sstatus
    addi t2,x0,2
    or t1,t1,t2
    csrw sstatus,t1

    la sp,boot_stack_top
    call start_kernel

    .section .bss.stack
    .globl boot_stack
boot_stack:
```

```
    .space 4096 # <-- change to your stack size

    .globl boot_stack_top
boot_stack_top:
```

4.2.2 entry.S

```
    .section .text.entry
    .align 2
    .globl _traps
    .extern trap_handler
_traps:
    # YOUR CODE HERE
    # -----------

        # 1. save 32 registers and sepc to stack
     sd sp, -8(sp)
     sd ra, -16(sp)
     sd gp, -24(sp)
     sd tp, -32(sp)
     sd t0, -40(sp)
     sd t1, -48(sp)
     sd t2, -56(sp)
     sd s0, -64(sp)
     sd s1, -72(sp)
     sd a0, -80(sp)
     sd a1, -88(sp)
     sd a2, -96(sp)
     sd a3, -104(sp)
     sd a4, -112(sp)
     sd a5, -120(sp)
     sd a6, -128(sp)
     sd a7, -136(sp)
     sd s2, -144(sp)
     sd s3, -152(sp)
     sd s4, -160(sp)
     sd s5, -168(sp)
     sd s6, -176(sp)
     sd s7, -184(sp)
     sd s8, -192(sp)
     sd s9, -200(sp)
     sd s10, -208(sp)
     sd s11, -216(sp)
     sd t3, -224(sp)
     sd t4, -232(sp)
     sd t5, -240(sp)
     sd t6, -248(sp)
     addi sp,sp,-248
    # -----------
        # 2. call trap_handler
     csrr a0,scause
     csrr a1,sepc
```

```
    call trap_handler
  # -----------

    # 3. restore sepc and 32 registers (x2(sp) should be restore last) from stack
    ld t6,(sp)
    ld t5,8(sp)
    ld t4,16(sp)
    ld t3,24(sp)
    ld s11,32(sp)
    ld s10,40(sp)
    ld s9,48(sp)
    ld s8,56(sp)
    ld s7,64(sp)
    ld s6,72(sp)
    ld s5,80(sp)
    ld s4,88(sp)
    ld s3,96(sp)
    ld s2,104(sp)
    ld a7,112(sp)
    ld a6,120(sp)
    ld a5,128(sp)
    ld a4,136(sp)
    ld a3,144(sp)
    ld a2,152(sp)
    ld a1,160(sp)
    ld a0,168(sp)
    ld s1,176(sp)
    ld s0,184(sp)
    ld t2,192(sp)
    ld t1,200(sp)
    ld t0,208(sp)
    ld tp,216(sp)
    ld gp,224(sp)
    ld ra,232(sp)
    ld sp,240(sp)
# -----------

    # 4. return from trap
    sret
  # -----------
```

4.2.3 trap.c

```c
// trap.c
#include "printk.h"
#include "sbi.h"
extern void clock_set_next_event();

void trap_handler(unsigned long scause, unsigned long sepc) {
    unsigned long x=scause,y=sepc,xi,yi;
    xi=0x8000000000000000;
    if(x>=xi){
        xi=0x8000000000000005;
        if(x==xi){
            //a supervisor time interrupt
            printk("kernel is running!\n");
            printk("[S] Supervisor Mode Timer Interrupt\n");
            clock_set_next_event();
        }
        else{
            //other interrupts
            ;
        }
    }
    else{
        ;//exception
    }
    return;
}
```

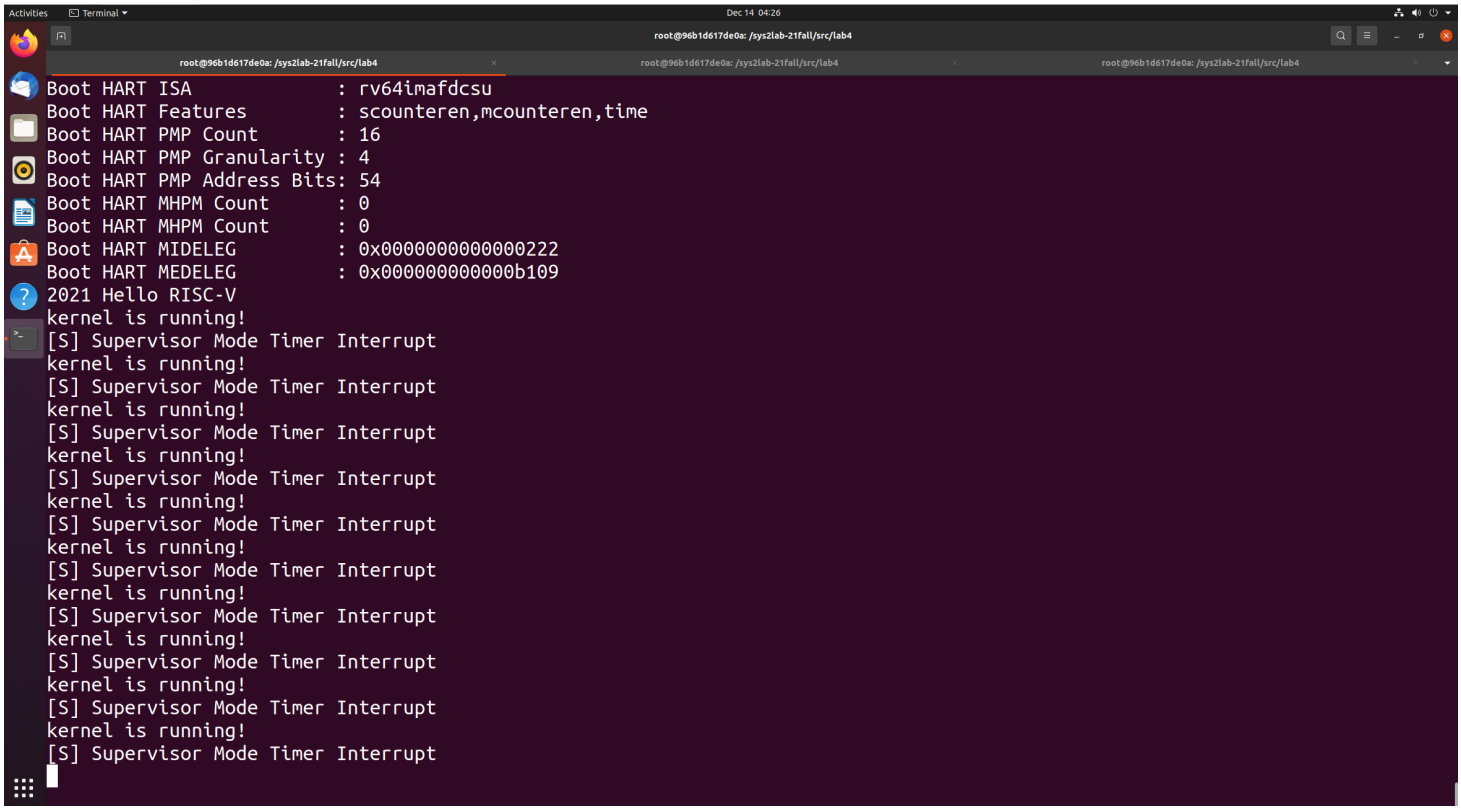
4.2.4 clock.c

```c
// clock.c
#include "sbi.h"
unsigned long TIMECLOCK = 10000000;

unsigned long get_cycles() {
   unsigned long ret_time;
   __asm__ volatile(
           "rdtime t0\n"
           "mv %[ret_time],t0"
           :[ret_time]"=r"(ret_time)
           :
           );
   return ret_time;
}

void clock_set_next_event() {
    unsigned long next = get_cycles() + TIMECLOCK;
    sbi_ecall(0x00,0,next,0,0,0,0,0);
    return;
}
```

4.2.5 make run



五、思考与心得

- 1、通过查看 RISC-V Privileged Spec 中的 medeleg 和 mideleg 解释上面 MIDELEG 值的含义。

  将SSIP(separate software interrupt-pending)，STIP(supervisor time interrupt-pending),SEIP(software external interrupts-pending)这四种中断或异常委托给S-Mode代理。