

# Lab4: RV64 时钟中断处理

朱熙哲

3220103361

2023 年 11 月 26 日

## 目录

<b>1</b>	<b>代码编写</b>	<b>2</b>
1.1	已提供的修改 . . . . .	2
1.2	head.S 开启异常处理 . . . . .	3
1.3	entry.S 实现上下文切换 . . . . .	4
1.4	trap.c 异常处理 . . . . .	5
1.5	clock.c 时钟中断相关函数 . . . . .	6
1.6	运行测试 . . . . .	7
<b>2</b>	<b>思考题</b>	<b>8</b>
2.1	解释 MIDELEG 值的含义 . . . . .	8
2.2	time 与 cycle 寄存器 . . . . .	8
2.3	一台不支持乘除法指令扩展的处理器上执行乘除法指令 . . .	8

# 1 代码编写

## 1.1 已提供的修改

将之前所有print.h puti puts的引用修改为printk.h printk. 也就是修改 test.c 与 main.c 的函数使用.

```
1 // main.c
2
3 #include "printk.h"
4
5 extern void test();
6
7 int start_kernel(){
8     printk("%d ZJU Computer System II\n", 2022);
9     test(); // DO NOT DELETE !!!
10    return 0;
11 }
```

其次, 依照要求修改 test.c、vmlinux 以及 head.S, 不再展示.

## 1.2 head.S 开启异常处理

```
1  _start:
2  # set stack pointer
3      la sp, stack_top
4  # set stvec = _traps
5      la a0, _traps
6      csrw stvec, a0
7  # set sie[STIE] = 1
8      li a0, 32
9      csrs sie, a0
10 # set first time interrupt
11     rdttime a0
12     li t0, 10000000
13     add a0, a0, t0
14     call sbi_set_timer
15 # set sstatus[SIE] = 1
16     csrs sstatus, 2
17
18     jal start_kernel
```

1. 设置栈指针, 用于之后调用 `set_timer` 函数.
2. 设置 `stvec` 为 `_traps` 地址, 且使用 `direct` 模式.  
地址对齐使得 `_traps` 的末两位为零, 即 `direct` 模式.
3. `sie` 的右数第 5 位为 `STIE`, 通过 `csrs` 将其设置为 1.
4. 设置第一次时钟中断:  
读取当前时钟, 加上一个中断周期, 然后设置时钟中断触发点.
5. `sstatus` 的右 1 位为 `SIE`, 通过 `csrs` 将其设置为 1, 以开启 S 态下的中断响应.

### 1.3 entry.S 实现上下文切换

```
1  _traps:
2      # 1. save 32 registers and sepc to stack
3      sd sp, -8(sp)
4      sd ra, -16(sp)
5      ...
6      sd t6, -248(sp)
7      addi sp, sp, -248
8      # -----
9      # 2. call trap_handler
10     csrr a0, scause
11     csrr a1, sepc
12     call trap_handler
13     # -----
14     # 3. restore sepc and 32 registers (x2(sp) should be
15     ↪ restore last) from stack
16     ld t6, 0(sp)
17     ld t5, 8(sp)
18     ...
19     ld ra, 232(sp)
20     ld sp, 240(sp)
21     # -----
22     # 4. return from trap
23     sret
24     # -----
```

1. sp 需要最后出栈, 因此最先入栈, 而后将 zero 外的寄存器依次入栈.
2. 将 scause 与 sepc 传递给 a0 与 a1, 随后调用 trap\_handler.
3. 按相反顺序依次出栈.

4. 从 S 态返回, 使用 sret.

## 1.4 trap.c 异常处理

```
1 // trap.c
2 #include "clock.h"
3 #include "printk.h"
4
5 void trap_handler(unsigned long scause, unsigned long sepc) {
6     if ((scause & 0x8000000000000000) &&
7         (scause & 0x7FFFFFFFFFFFFFFF) == 5) {
8         printk("[S] Supervisor Mode Timer Interrupt\n");
9         clock_set_next_event();
10        return;
11    }
12 }
```

根据手册, 时钟中断时 scause 的最高位为 1(判断为 interrupt), 剩余位的值为 5(Supervisor timer interrupt), 此时可以打印时钟中断信息, 并调用 clock\_set\_next\_event(), 设置下一个时钟中断.

## 1.5 clock.c 时钟中断相关函数

```
1 // clock.c
2 #include "sbi.h"
3 unsigned long TIMECLOCK = 10000000;
4
5 unsigned long get_cycles(){
6     unsigned long time;
7     asm volatile (
8         "rdtime %[time]"
9         : [time] "=r" (time)
10        : : "memory"
11    );
12    return time;
13 }
14 void clock_set_next_event(){
15     unsigned long next_time = get_cycles() + TIMECLOCK;
16     sbi_set_timer(next_time);
17 }
```

1. get\_cycles:  
通过内联汇编, 使用 rdtime 得到 time 寄存器的值, 并返回.
2. clock\_set\_next\_event:  
先得到当前 time, 加上一个中断周期并设置时钟中断.
3. 另外, 需要添加一个 clock.h 头文件.

```

1 // clock.h
2
3 #ifndef _CLOCK_H
4 #define _CLOCK_H
5 unsigned long get_cycles();
6 void clock_set_next_event();
7 #endif

```

## 1.6 运行测试

成功运行.

```

Boot HART ID : 0
Boot HART Domain : root
Boot HART Priv Version : v1.12
Boot HART Base ISA : rv64imafdc
Boot HART ISA Extensions : zicntr
Boot HART PMP Count : 16
Boot HART PMP Granularity : 4
Boot HART PMP Address Bits : 54
Boot HART MHPM Info : 0 (0x00000000)
Boot HART MIDELEG : 0x0000000000000222
Boot HART MEDELEG : 0x0000000000000b109
2022 ZJU Computer System II
kernel is running! Time: 1s
[S] Supervisor Mode Timer Interrupt
kernel is running! Time: 2s
[S] Supervisor Mode Timer Interrupt
kernel is running! Time: 3s
[S] Supervisor Mode Timer Interrupt
kernel is running! Time: 4s
[S] Supervisor Mode Timer Interrupt
kernel is running! Time: 5s
[S] Supervisor Mode Timer Interrupt
kernel is running! Time: 6s
[S] Supervisor Mode Timer Interrupt
kernel is running! Time: 7s
[S] Supervisor Mode Timer Interrupt
kernel is running! Time: 8s
[S] Supervisor Mode Timer Interrupt

```

## 2 思考题

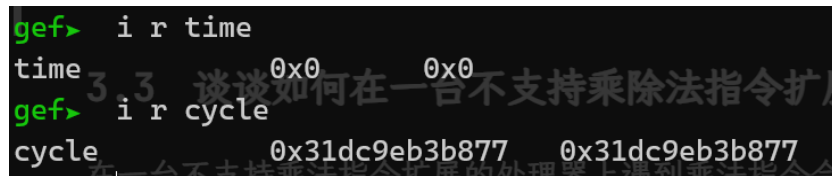
### 2.1 解释 MIDELEG 值的含义

MIDELEG(machine interrupt delegation register)、MEDELEG(machine exception delegation register) 并称 machine trap delegation registers (机器中断委托寄存器). 其对应的位和 scause 中定义相同, 表示是否将对应位上的中断委托给 S 模式来处理, 如果对应位为 1 则委托给 Supervisor, 否则由 Machine 模式处理。

MIDELEG 的值为 0x222(0b1000100010), 右数第 5 位被置为 1, 也就是 S 模式时钟中断, 若置 0, 则不会交由编写的时钟中断函数处理, 也就不会输出信息。

### 2.2 time 与 cycle 寄存器

使用 qemu 进行 debug 查看两个寄存器的初值. 发现 time 寄存器初值置 0, 而 cycle 寄存器没有清零。



```
gef> i r time
time 0x0
gef> i r cycle
cycle 0x31dc9eb3b877
```

time 寄存器需要记录运行时间, 因此初值置零. cycle 寄存器记录内核执行的时钟周期数, 使用意义不大, 而需要使用时记录相对值即可。

### 2.3 一台不支持乘除法指令扩展的处理器上执行乘除法指令

在一台未支持乘除法指令扩展的处理器上遇到乘除法指令会触发 Illegal Instruction 异常 (对应 scause 的值为 2), 同样会进入 trap handler 中, 此时非法指令的编码会被存储到 mtval (有委托情况下是 stval) 中, 可以在 handler 中读取非法指令进行判断, 然后处理得到正确结果。