

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 操作系统 |
| 实验名称： | RV64 内核引导与时钟中断处理 |
| 姓 名： | 蔡雨谦 |
| 学 院： | 计算机学院 |
| 系： | 计算机系 |
| 专 业： | 计算机科学与技术 |
| 学 号： | 3210102466 |
| 指导教师： | 寿黎但 |

2023年 9月 28日

**浙江大学实验报告**

# 实验目的

* 学习RISC-V 汇编， 编写 head.S 实现跳转到内核运行的第一个 C 函数。
* 学习OpenSBI，理解 OpenSBI 在实验中所起到的作用，并调用 OpenSBI 提供的接口完成字符的输出。
* 学习Makefile 相关知识， 补充项目中的 Makefile 文件， 来完成对整个工程的管理。
* 学习RISC-V 的 trap 处理相关寄存器与指令，完成对 trap 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trap 处理函数，完成对特定 trap 的处理。
* 调用OpenSBI 提供的接口，完成对时钟中断事件的设置。

# 实验环境

* Ubuntu 22.04.2 LTS Windows Subsystem for Linux 2

# 操作方法与实验步骤

1. **准备工程**

从课程提供的github仓库获取实验代码框架，准备进行后续实验。

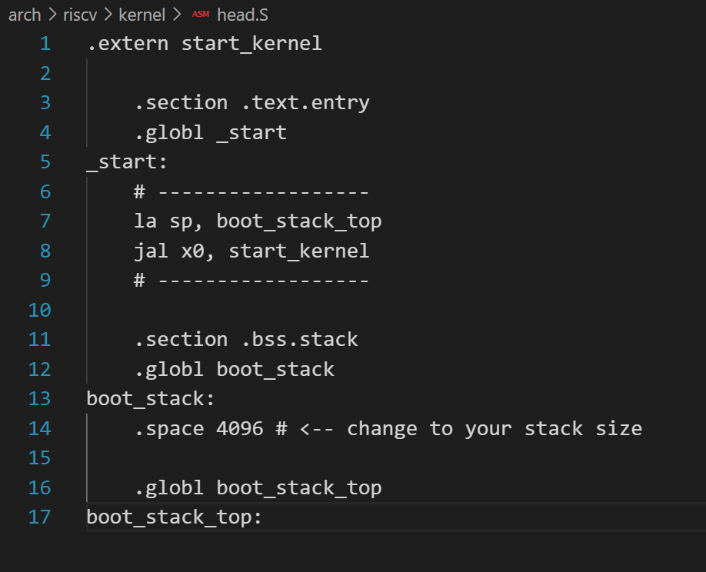
1. **RV64内核引导**

实现RV64内核引导需要完善的文件：

* arch/riscv/kernel/head.S
* lib/Makefile
* arch/riscv/kernel/sbi.c
* arch/riscv/include/defs.h

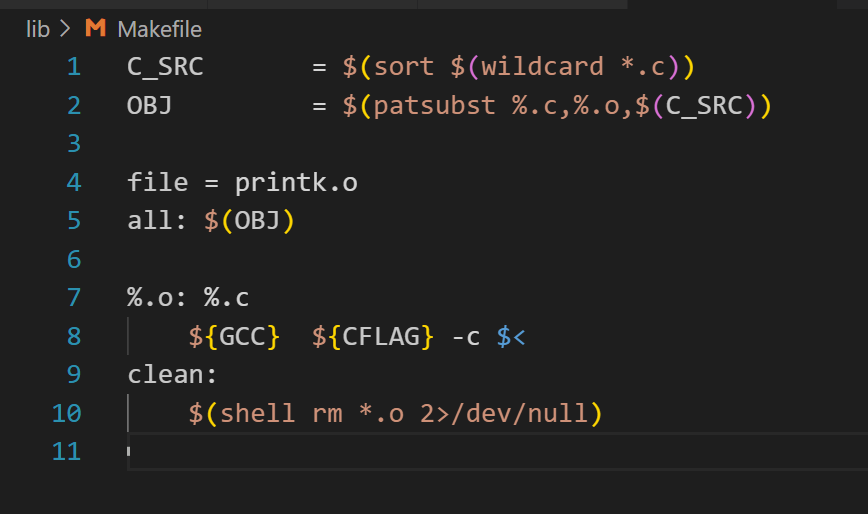
2.1 编写head.S

head.S的作用是引导内核的启动。设置一个大小为4kb(4096)的栈，并通过跳转指令跳转到start\_kernel函数。

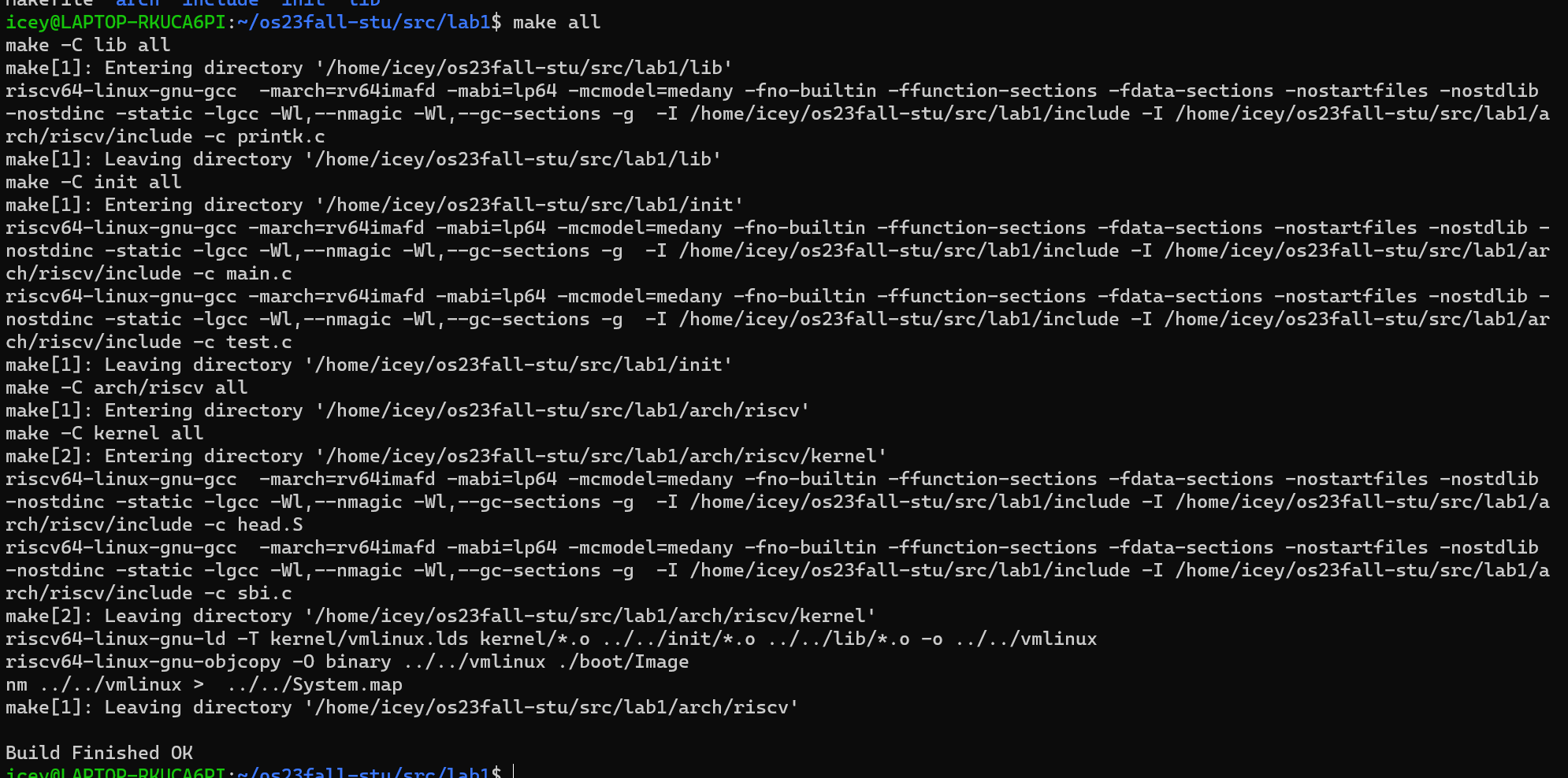


2.2 完善makefile脚本

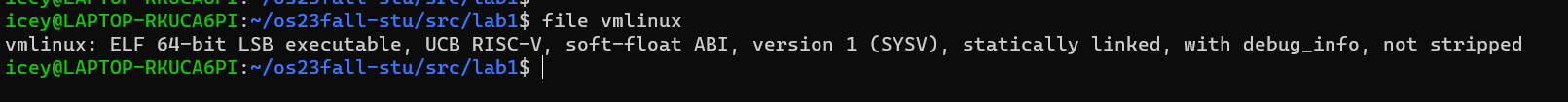
参照已经给出的其他makefile 文件，将目录下的全部.c文件编译成.o文件。



运行make all指令可以进行编译。

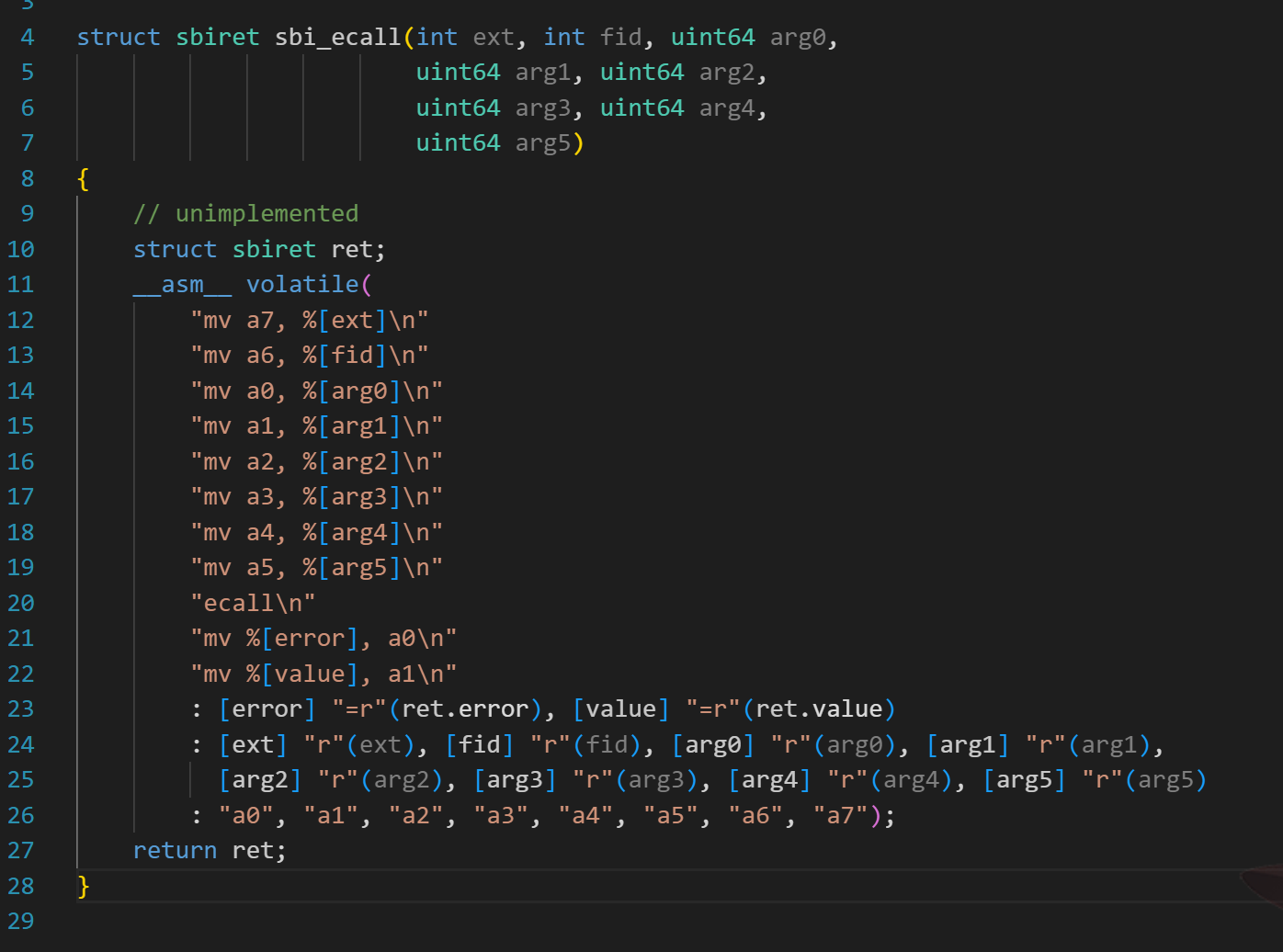


编译后会生成vmlinux内核文件，使用file vmlinux指令查看：



2.3 补充sbi.c

参照示例一，使用asm volitile向C代码中插入内联汇编代码。将 ext放入寄存器 a7 中，fid放入寄存器 a6 中，将 arg0 - arg5 放入寄存器 a0 - a5 中。使用 ecall 指令。ecall 之后系统会进入 M 模式。返回结果存放在寄存器 a0、a1 中，其中a0为 error，a1为返回值。将a0、a1的值放进一个sbiret变量中作为sbi\_ecall的返回值。

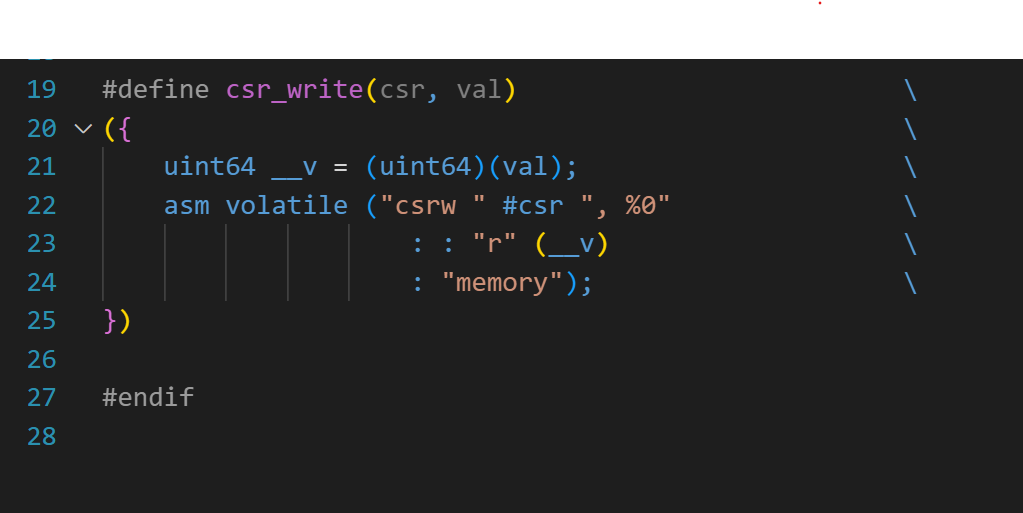


2.4 修改defs.h

参照示例二，使用asm volitile插入内联汇编代码，完善csr\_read宏，将csr寄存器的值读到变量\_V中。



代码框架中给出的csr\_write宏则是将val的值写进csr寄存器中。



2.5 运行RV64 内核。

使用make all指令编译项目文件夹得到的内核可以通过make run指令运行。



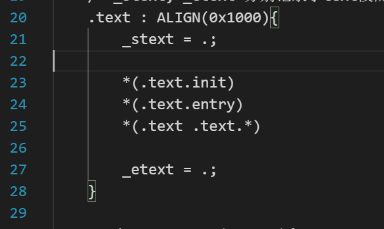
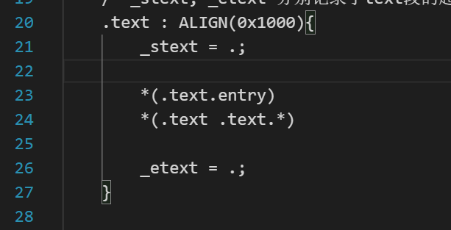
1. **RV64时钟中断处理**

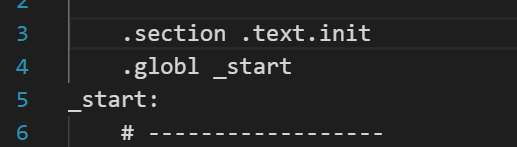
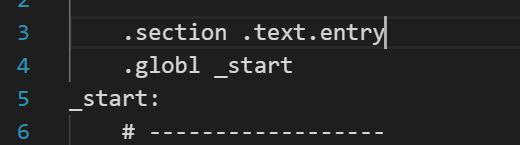
实现RV64时钟中断处理需要完善或添加的文件：

* arch/riscv/kernel/head.S
* arch/riscv/kernel/entry.S
* arch/riscv/kernel/trap.c
* arch/riscv/kernel/clock.c

3.1 准备工作

vmlinux.lds与head.S文件中有需要修改的部分如下(左为修改前，右为修改后)：

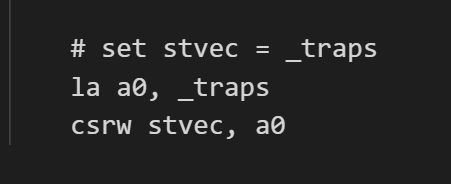




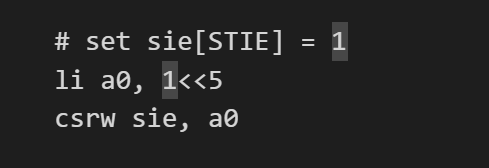
3.2 开启trap处理

在运行start\_kernel前需要在head.S中对一些寄存器进行初始化：

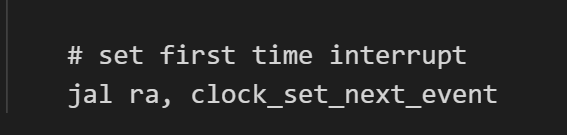
设置stvec,将\_traps所表示的地址写入stvec



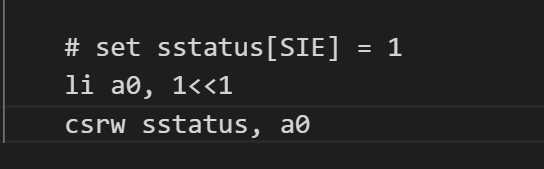
开启时钟中断，将sie[STIE]置1，STIE是第五位：



设置第一次时钟中断



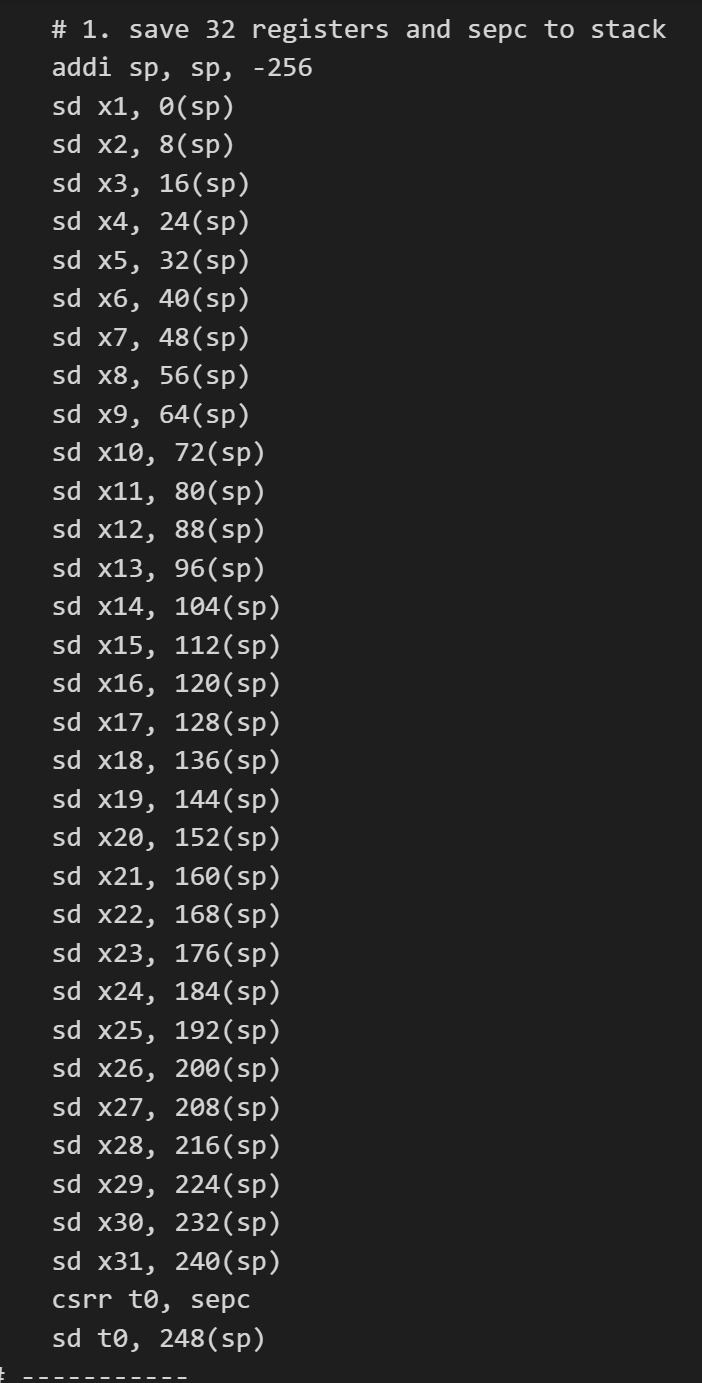
开启S态下的中断响应，将sstatus[SIE]置1，SIE是第一位：



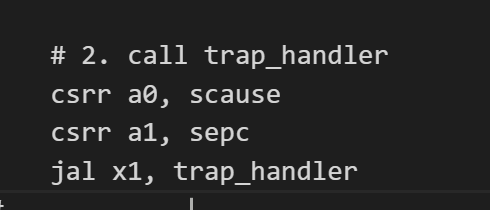
3.3 实现上下文切换

在arch/riscv/kernel/ 目录下添加 entry.S 文件，文件包含以下内容：

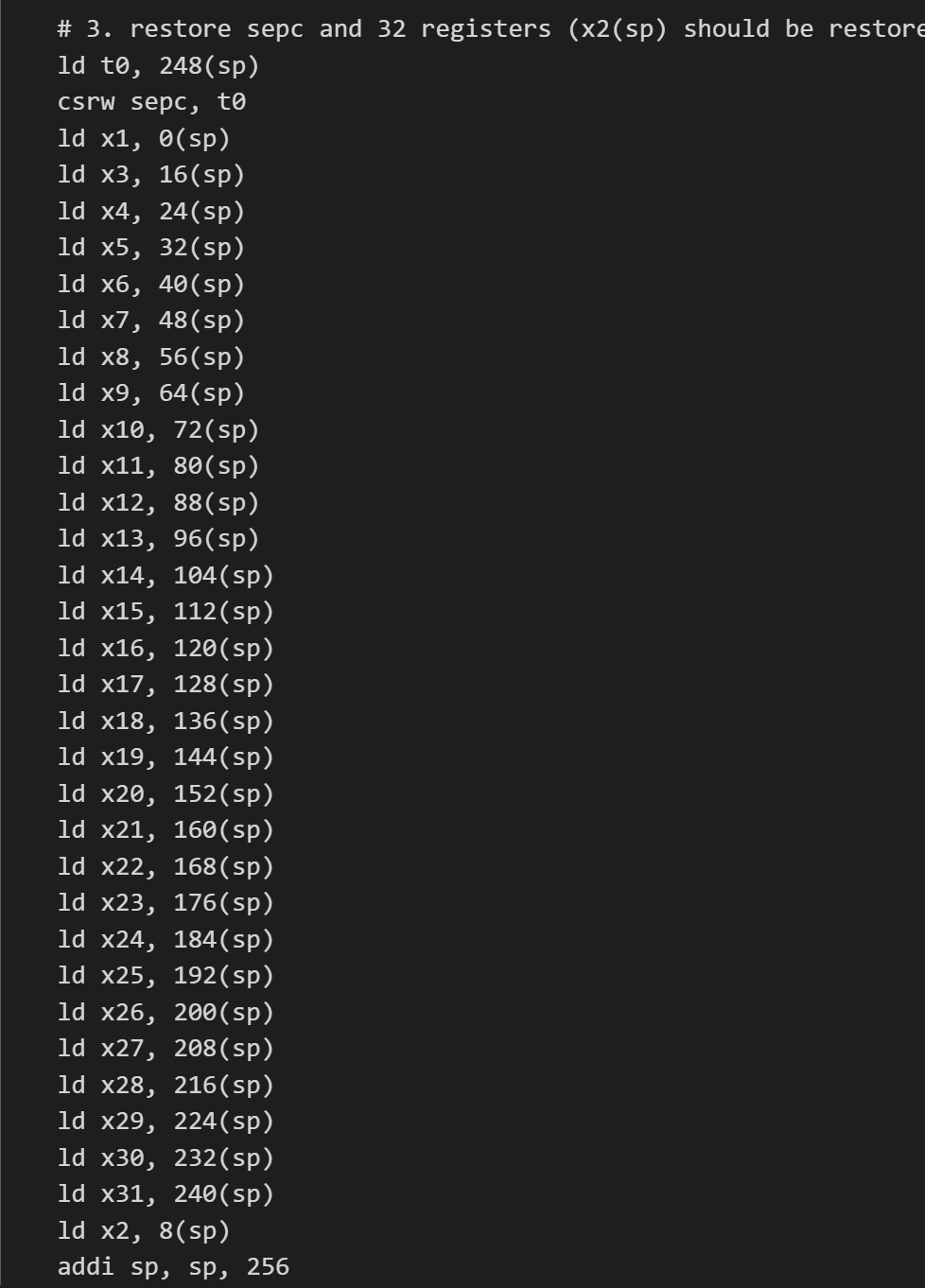
保存CPU 的寄存器到内存中



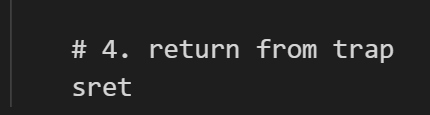
将scause 和 sepc 中的值传入 trap 处理函数trap\_handler



从内存中恢复 CPU 的寄存器

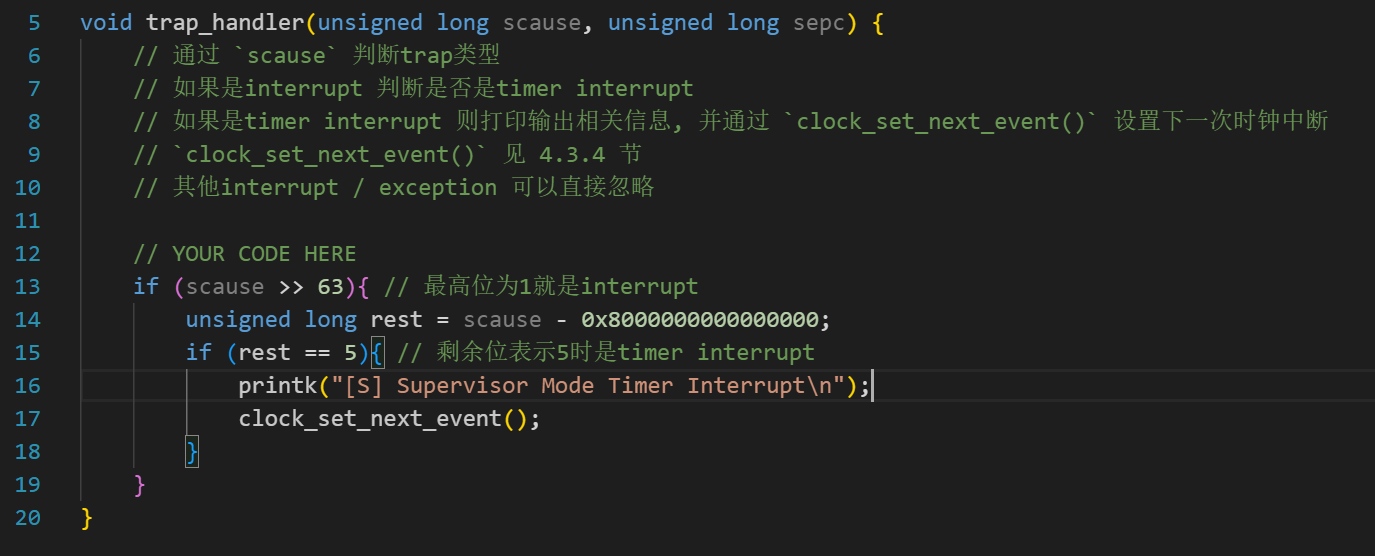


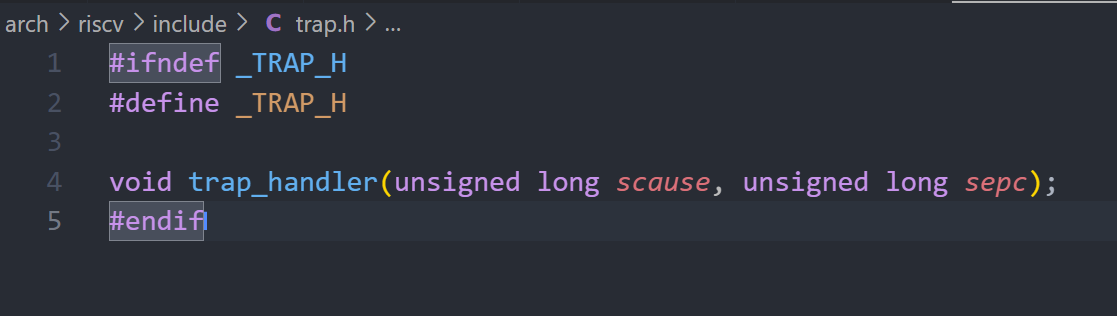
从trap中返回



3.4 实现trap处理函数

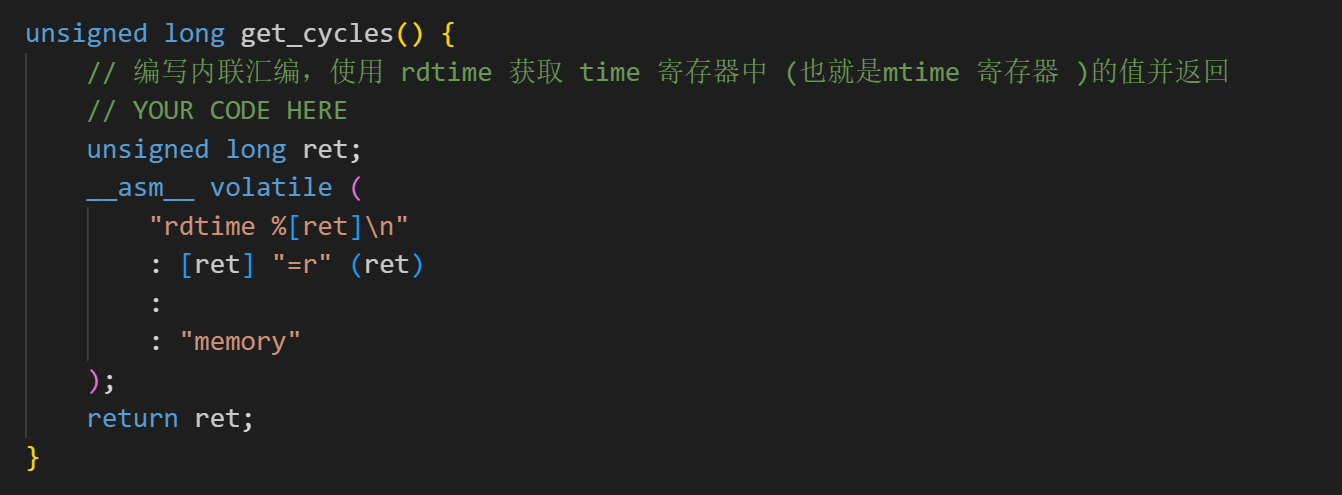
在 arch/riscv/kernel/ 目录下添加 trap.c 文件，文件包含trap处理函数trap\_handler(), scause和sepc寄存器的值作为参数。scause寄存器最高位为1时为interrupt， 剩余位表示5时为timer interrupt。在 arch/riscv/include/ 目录下添加 对应trap.h 文件

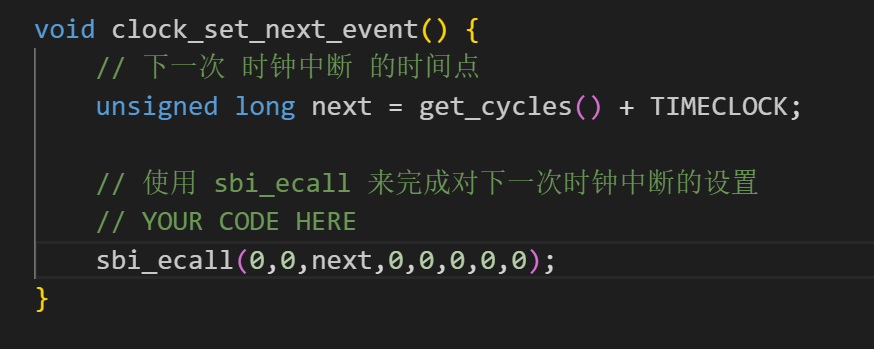




3.5 实现时钟中断相关函数

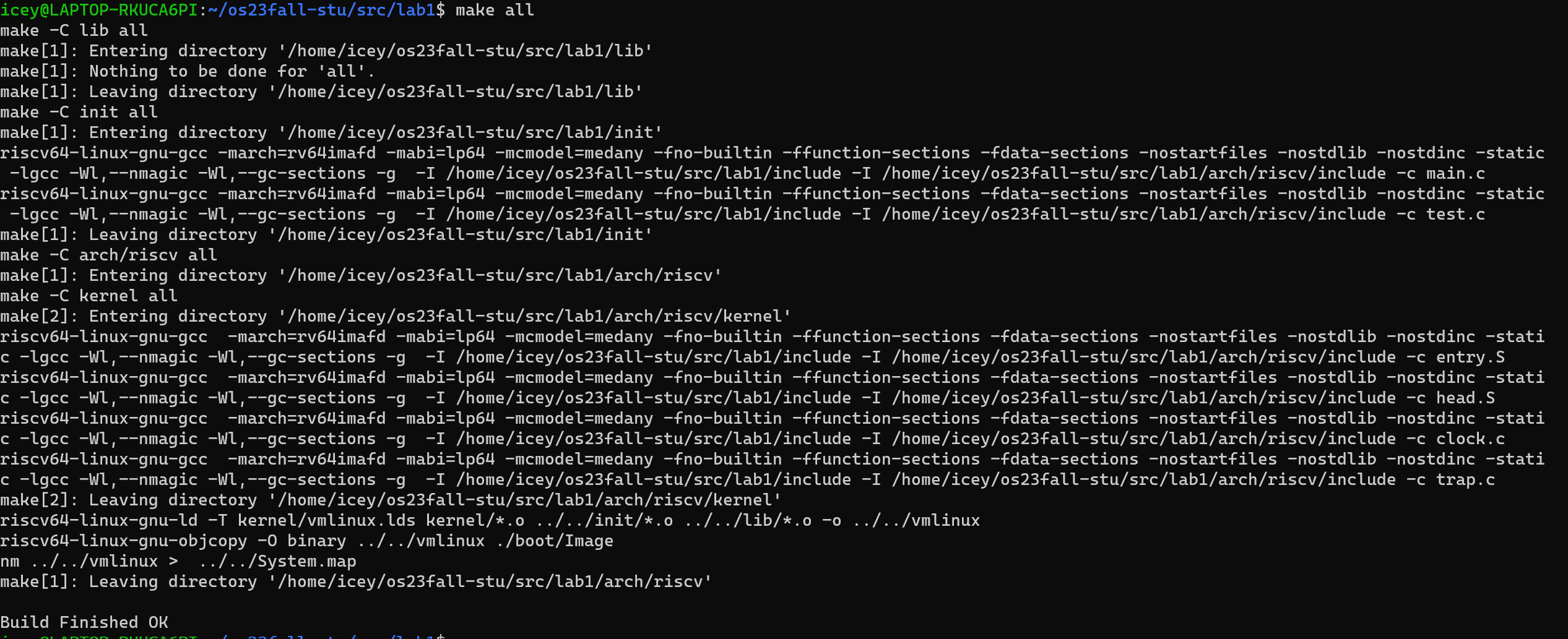
在 arch/riscv/kernel/ 目录下添加 clock.c 文件。文件包含两个函数：get\_cycles ( ) : 使用 rdtime 汇编指令获得当前 time 寄存器中的值; clock\_set\_next\_event ( ) : 调用 sbi\_ecall，设置下一个时钟中断事件。



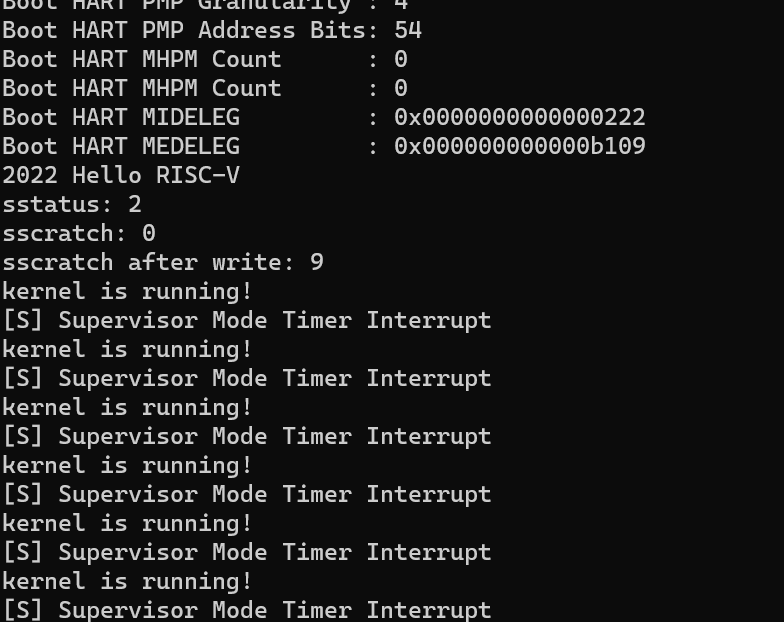


3.6 编译测试

虽然新添了.c文件，但是在RV64内核引导部分写的makefile文件将全部.c文件一起处理了，所以不用修改。



运行make run指令，有如下输出：



# 四、思考题

1. 请总结一下 RISC-V 的 calling convention，并解释 Caller / Callee Saved Register 有什么区别？

calling convention(函数调用规范)：

1）将参数存储到函数能够访问到的位置；

2）跳转到函数开始位置（使用jal指令）

3）获取函数需要的局部存储资源，按需保留寄存器

4）执行函数中的指令

5）将返回值存储到调用者能够访问到的位置，回复寄存器，释放局部存储资源

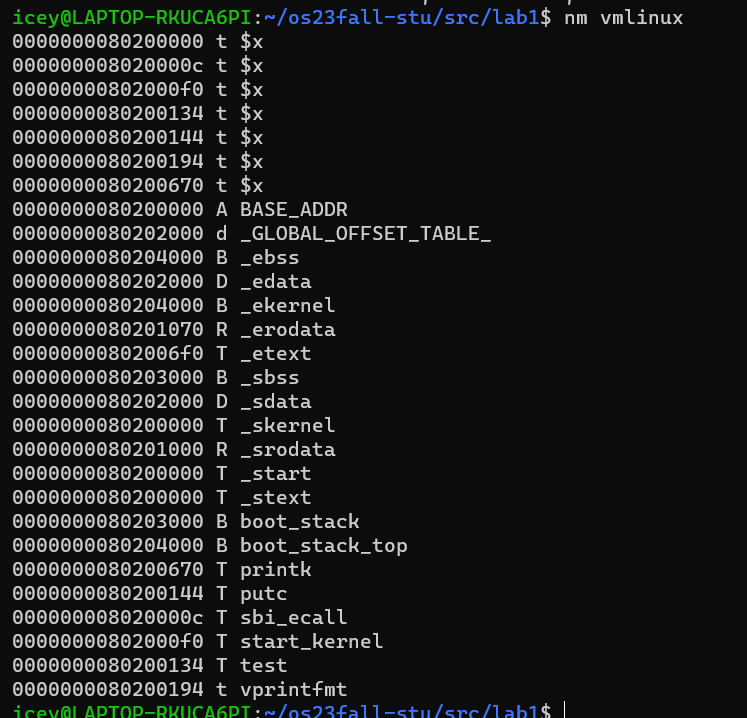
6）返回调用函数的位置（使用ret指令）

Caller / Callee Saved Register的区别：

caller saved registers（调用者保存寄存器）也叫易失性寄存器，在程序调用的过程中，这些寄存器中的值不需要被保存，如果某一个程序需要保存这个寄存器的值，需要调用者自己压入栈；

callee saved registers（被调用者保存寄存器）也叫非易失性寄存器，在程序调用过程中，这些寄存器中的值需要被保存，不能被覆盖；当某个程序调用这些寄存器，被调用寄存器会先保存这些值然后再进行调用，且在调用结束后恢复被调用之前的值；

2. 编译之后，通过 System.map 查看 vmlinux.lds 中自定义符号的值（截图）。

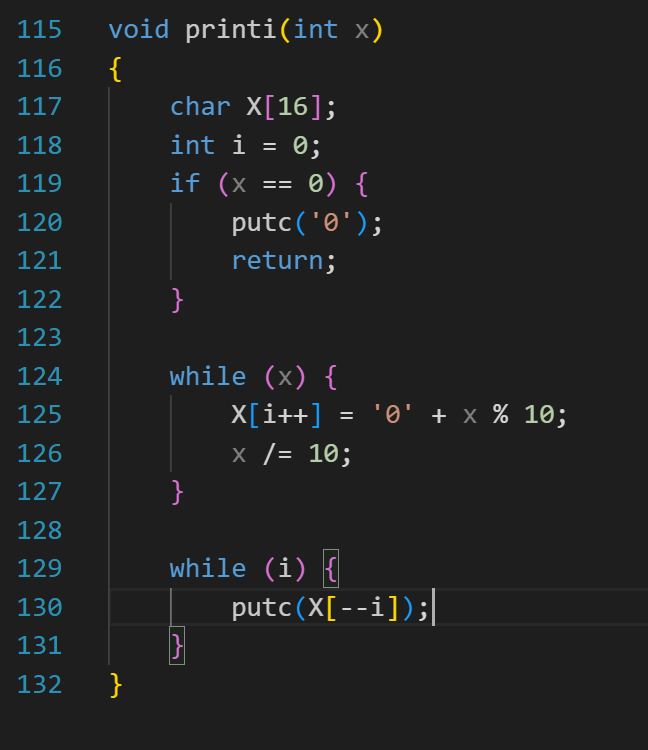


3. 用 csr\_read 宏读取 sstatus 寄存器的值，对照 RISC-V 手册解释其含义（截图）。

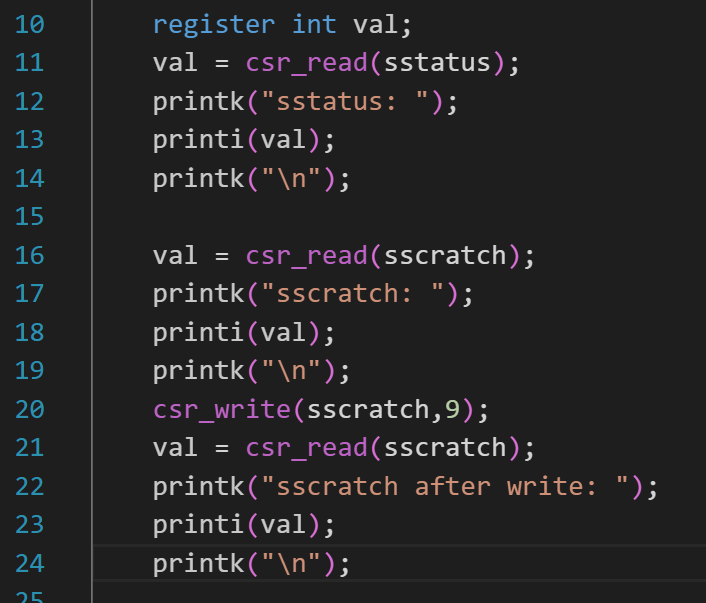
4. 用 csr\_write 宏向 sscratch 寄存器写入数据，并验证是否写入成功（截图）。

3 and 4：

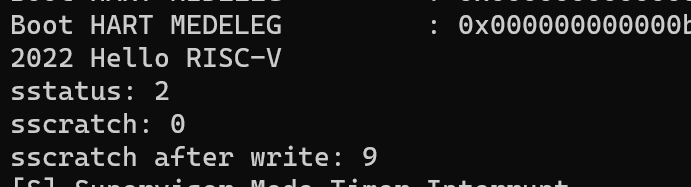
要输出寄存器的值，需要一个输出函数，负责输出int类型的值：



在 main.c中添加代码用来读写寄存器：



输出如下：



sstatus寄存器的第1位(SIE)为1说明允许S模式的中断。

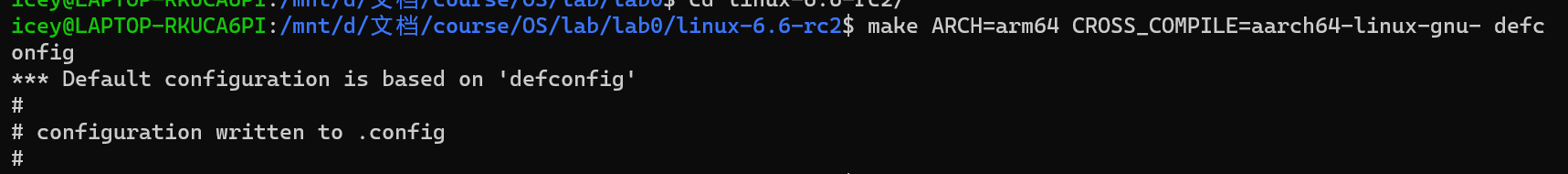
sscratch寄存器前后值的变化说明写入成功。

5. Detail your steps about how to get arch/arm64/kernel/sys.i

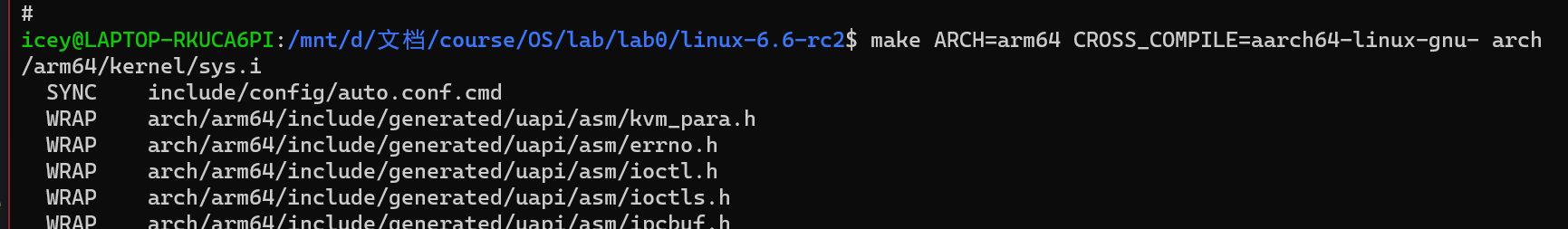
进入linux文件夹：

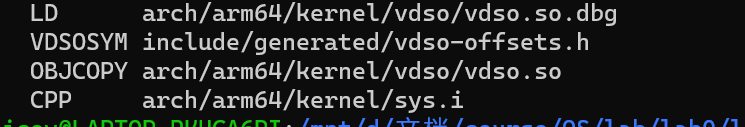


配置defconfig:



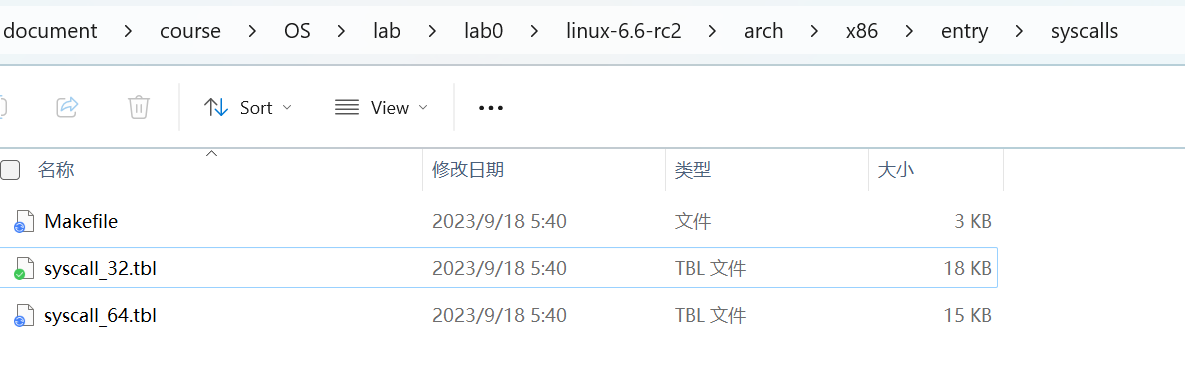
生成sys.i文件：



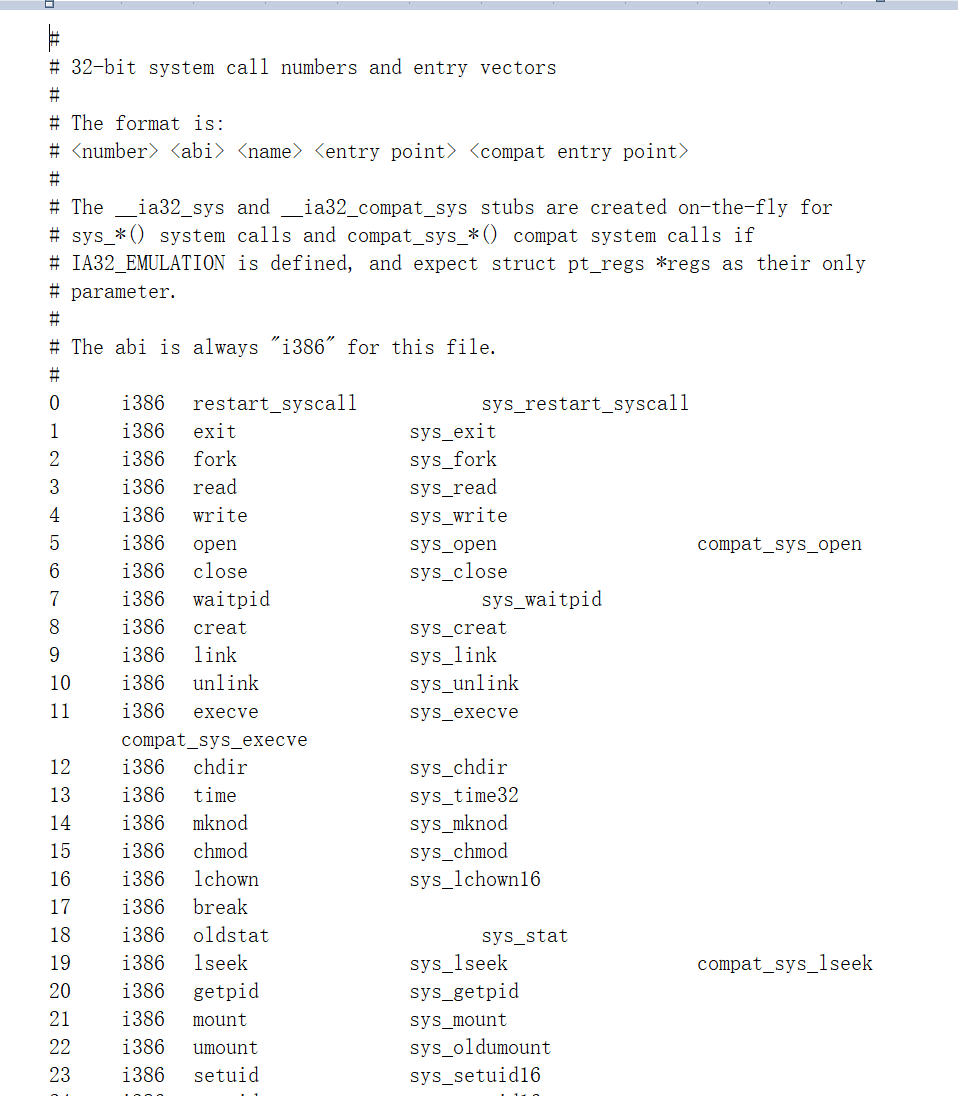


6. Find system call table of Linux v6.0 for ARM32, RISC-V(32 bit), RISC-V(64 bit), x86(32 bit), x86\_64 List source code file, the whole system call table with macro expanded, screenshot every step.

System call table for x86:

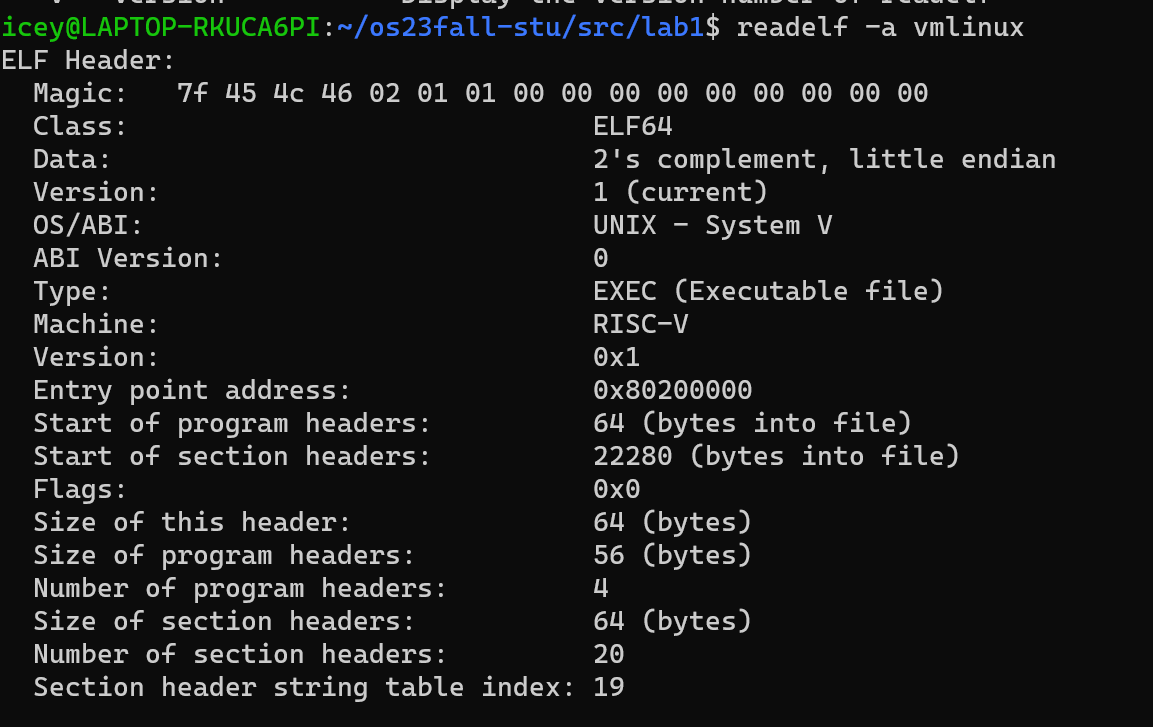


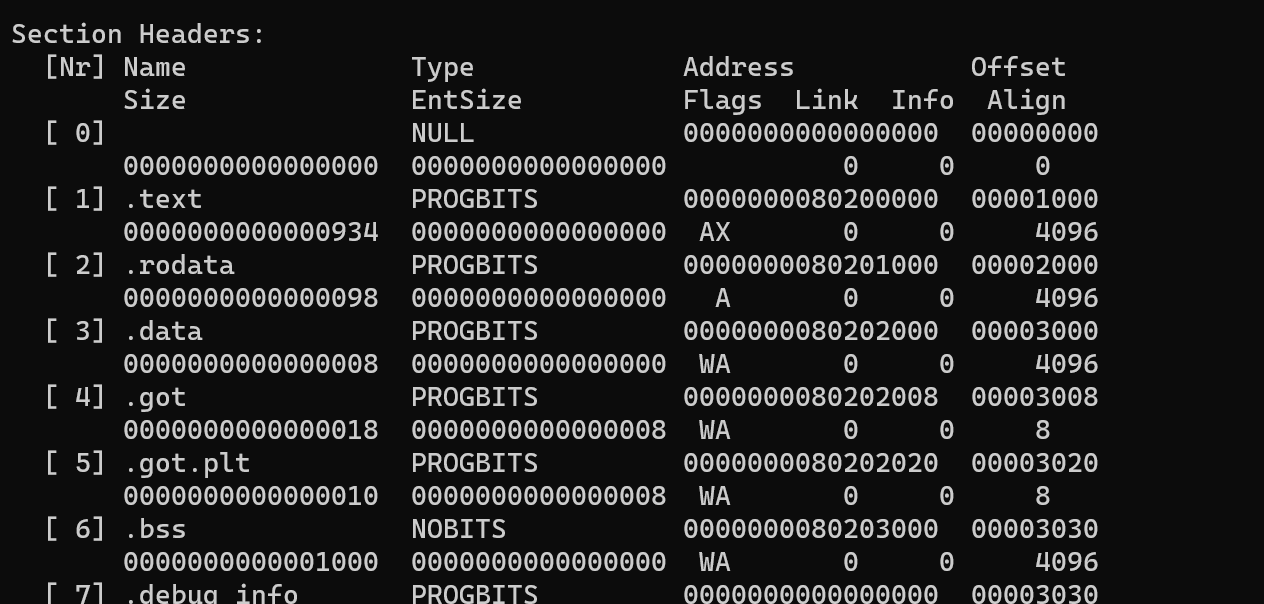
如图所示：

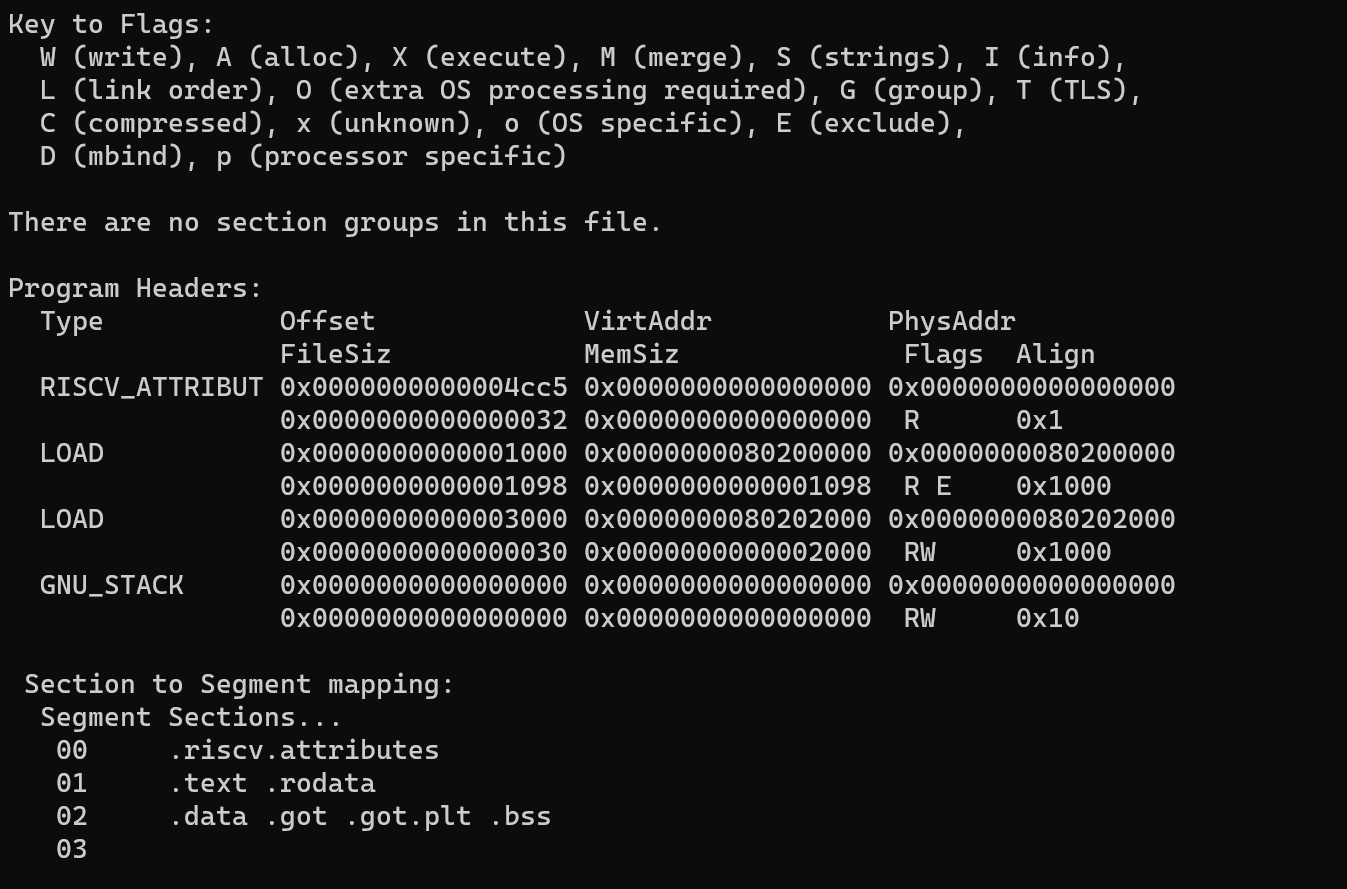


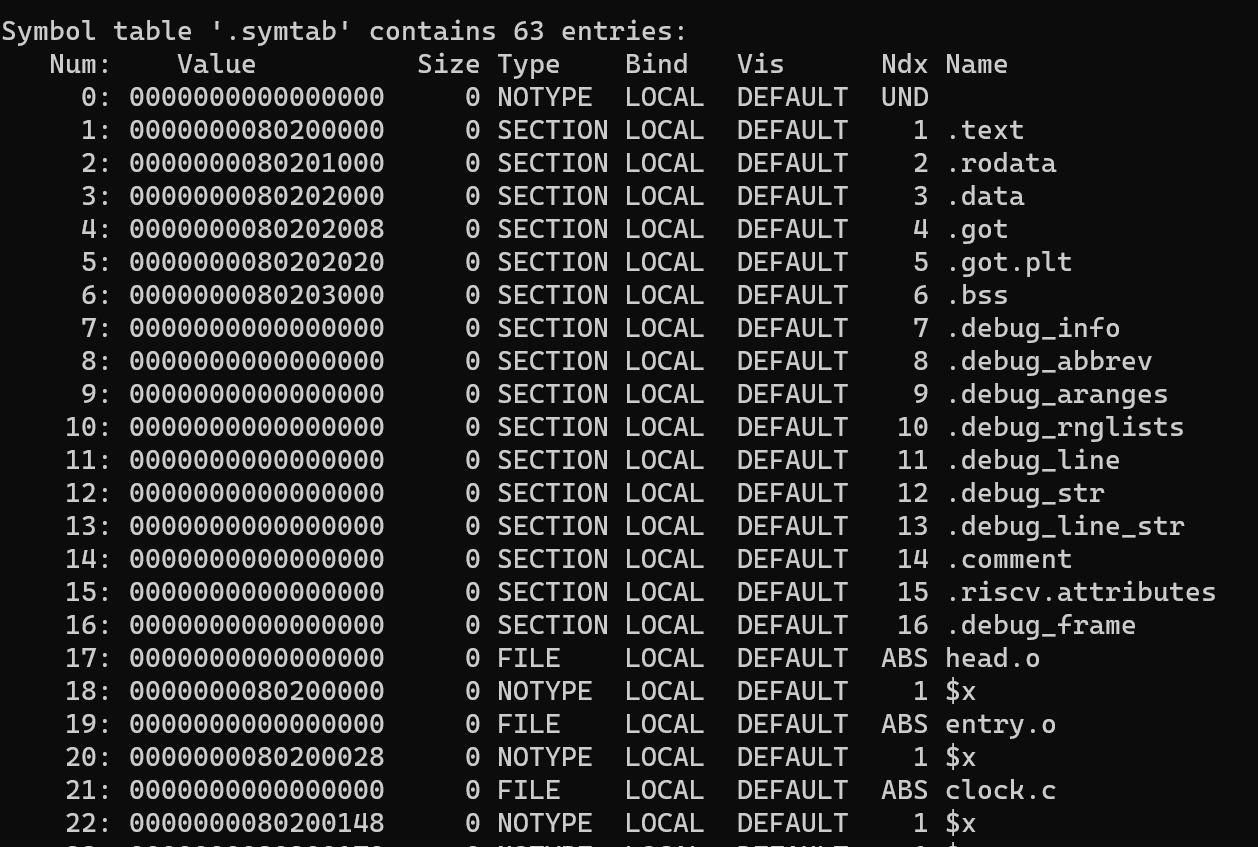
7. Explain what is ELF file? Try readelf and objdump command on an ELF file, give screenshot of the output. Run an ELF file and cat /proc/PID/maps to give its memory layout.

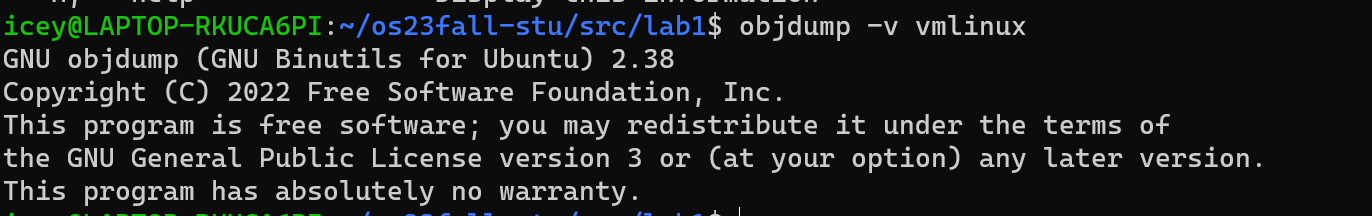
ELF(executable and linkable format)是一种可执行文件格式，主要用于linux平台，windows平台下相对应的文件为coff格式。

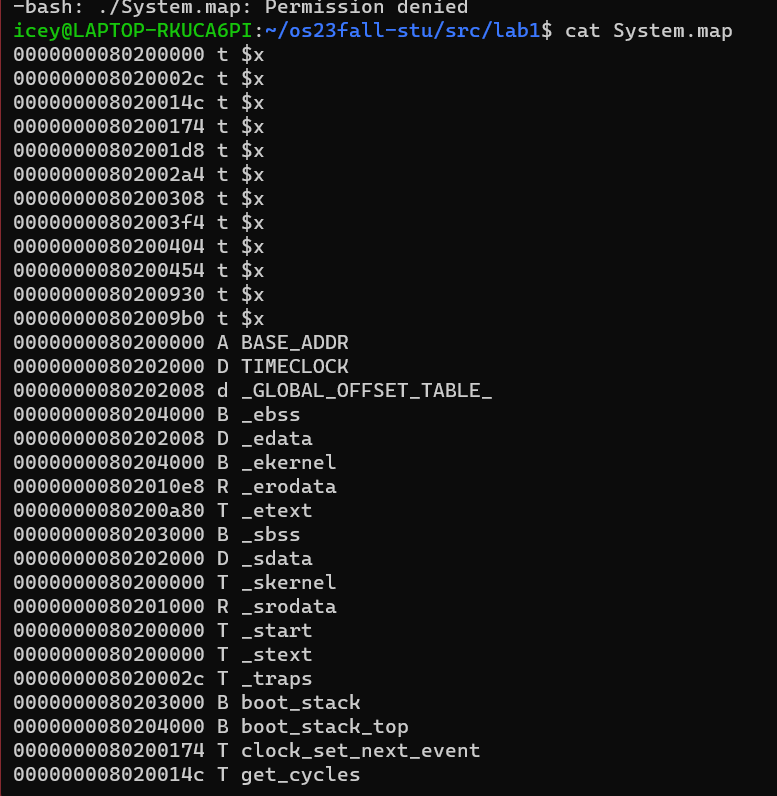












8. 在我们使用 make run 时， OpenSBI 会产生如下输出 :



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

MIDELEG值为0x222，即第1、5、9位为1，也就是说这些位上的中断由S模式处理。同样，MEDELEG寄存器的对应位的异常由S 模式处理。

# 五、讨论心得

1、编写makefile文件时，可以将文件夹下的所有.c文件用all来代替，这样添加新的.c文件时不用修改原makefile文件。

2、本实验最重要的部分还是对扩展知识的理解上，无论是makefile的编写还是汇编语言的使用都是以前没有深入了解过的内容。虽然本次实验只使用到较小部分的扩展知识，按照实验指导进行实验就很顺利，但为了后续实验也能顺利进行，必须更全面地学习这些知识。

3、内核运行后出现的界面包含了许多信息，做lab0时没有仔细看，做lab1的思考题时才注意到这点。包括OpenSBI的版本、平台的名称、一些寄存器的值等等。



4、实现时钟中断的实验中，在往head.S文件增添代码时，要添在加载栈地址的代码之后。

