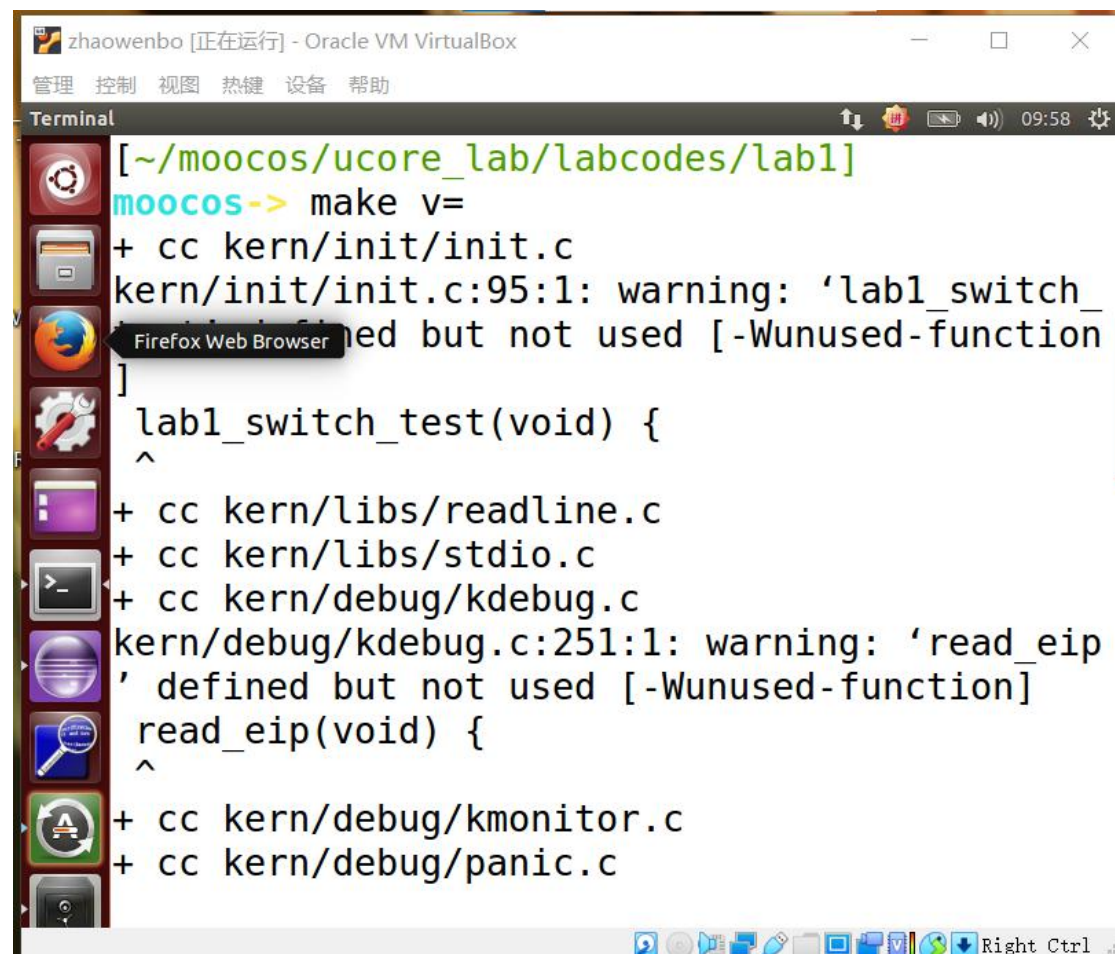


问题 1:

1: 将 c 文件编译为 o 文件



The screenshot shows a terminal window titled "zhaowenbo [正在运行] - Oracle VM VirtualBox". The terminal is running in the directory `~/moocos/ucore_lab/labcodes/lab1`. The user has entered the command `moocos-> make v=`. The terminal output shows the compilation of several C files into object files using the `cc` compiler. The files being compiled are `kern/init/init.c`, `kern/libs/readline.c`, `kern/libs/stdio.c`, `kern/debug/kdebug.c`, `kern/debug/kmonitor.c`, and `kern/debug/panic.c`. There are two warnings displayed: one for `lab1_switch_` in `kern/init/init.c:95:1` and another for `read_eip` in `kern/debug/kdebug.c:251:1`, both indicating that the functions are defined but not used. The terminal also shows the definition of `lab1_switch_test(void)` and `read_eip(void)`. The terminal window has a sidebar with icons for various applications, including a terminal, a file manager, a web browser, and a settings window. The bottom status bar shows the time as 09:58 and the keyboard layout as Right Ctrl.

```
[~/moocos/ucore_lab/labcodes/lab1]
moocos-> make v=
+ cc kern/init/init.c
kern/init/init.c:95:1: warning: 'lab1_switch_
' defined but not used [-Wunused-function]
]
lab1_switch_test(void) {
^
+ cc kern/libs/readline.c
+ cc kern/libs/stdio.c
+ cc kern/debug/kdebug.c
kern/debug/kdebug.c:251:1: warning: 'read_eip
' defined but not used [-Wunused-function]
read_eip(void) {
^
+ cc kern/debug/kmonitor.c
+ cc kern/debug/panic.c
```

zhaowenbo [正在运行] - Oracle VM VirtualBox

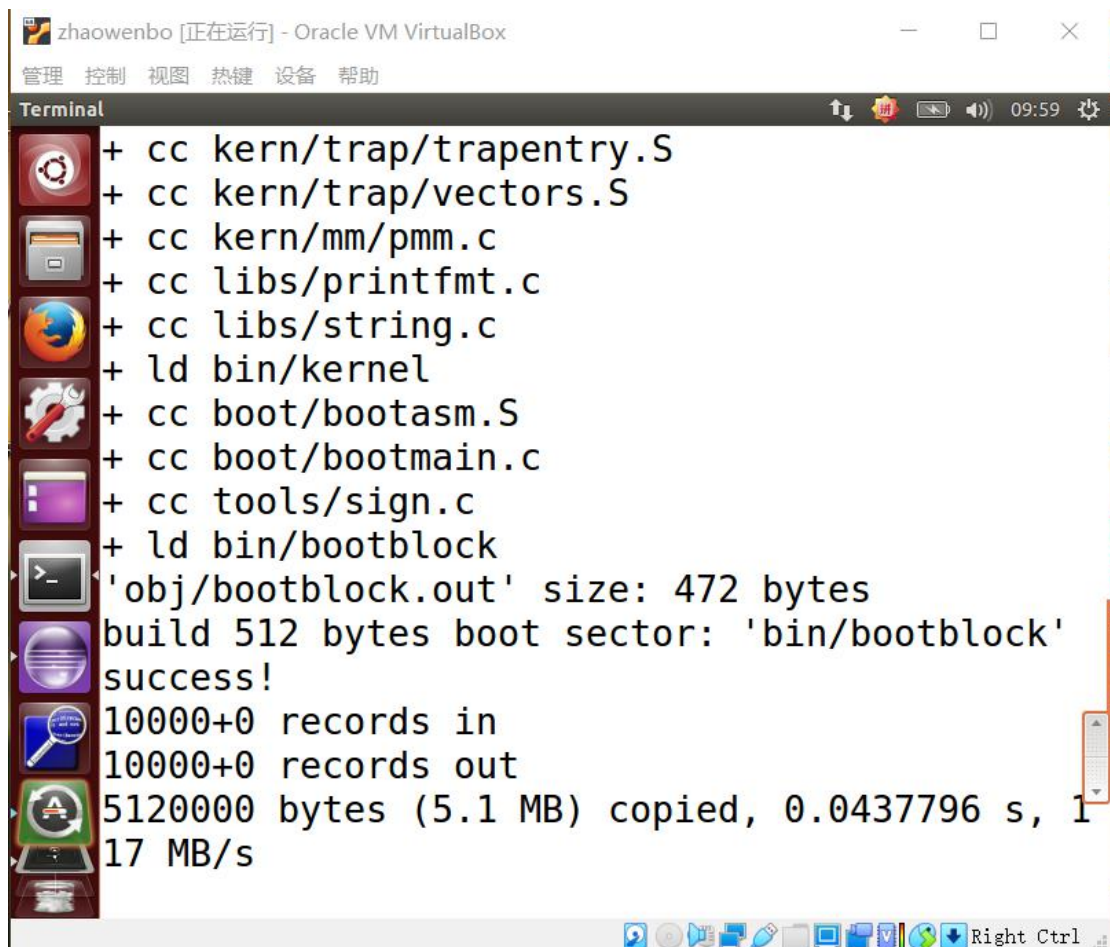
管理 控制 视图 热键 设备 帮助

Terminal 09:59

```
^
+ cc kern/debug/kmonitor.c
+ cc kern/debug/panic.c
+ cc kern/driver/clock.c
+ cc kern/driver/console.c
+ cc kern/driver/intr.c
+ cc kern/driver/picirq.c
+ cc kern/trap/trap.c
kern/trap/trap.c:14:13: warning: 'print_ticks'
defined but not used [-Wunused-function]
static void print_ticks() {
^
kern/trap/trap.c:30:26: warning: 'idt_pd' def
ined but not used [-Wunused-variable]
static struct pseudodesc idt_pd = {
^
+ cc kern/trap/trapentry.S
```

Right Ctrl

2. ld 命令根据链接脚本文件 kernel.ld 将生成的*.o 文件，链接成 BIN 目录下的 kernel 文件。
- 3.通过 GCC 编译器将 boot 目录下的.c,.S 文件以及 tools 目录下的 sign.c 文件编译成 OBJ 目录下的*.o 文件。
- 4.ld 命令将生成的*.o 文件，链接成 BIN 目录下的 bootblock 文件。



The screenshot shows a terminal window titled "zhaowenbo [正在运行] - Oracle VM VirtualBox". The terminal displays a series of commands and their outputs, including compilation of kernel components and the creation of a boot block. The commands are prefixed with a "+" sign, likely indicating they were added to a script. The outputs show the size of the boot block, successful build of the boot sector, and disk write statistics.

```
zhaowenbo [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Terminal
+ cc kern/trap/trapentry.S
+ cc kern/trap/vectors.S
+ cc kern/mm/pmm.c
+ cc libs/printfmt.c
+ cc libs/string.c
+ ld bin/kernel
+ cc boot/bootasm.S
+ cc boot/bootmain.c
+ cc tools/sign.c
+ ld bin/bootblock
'obj/bootblock.out' size: 472 bytes
build 512 bytes boot sector: 'bin/bootblock'
success!
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB) copied, 0.0437796 s, 1
17 MB/s
```

问题 2:

大小为 512 字节

多余的空间填 0

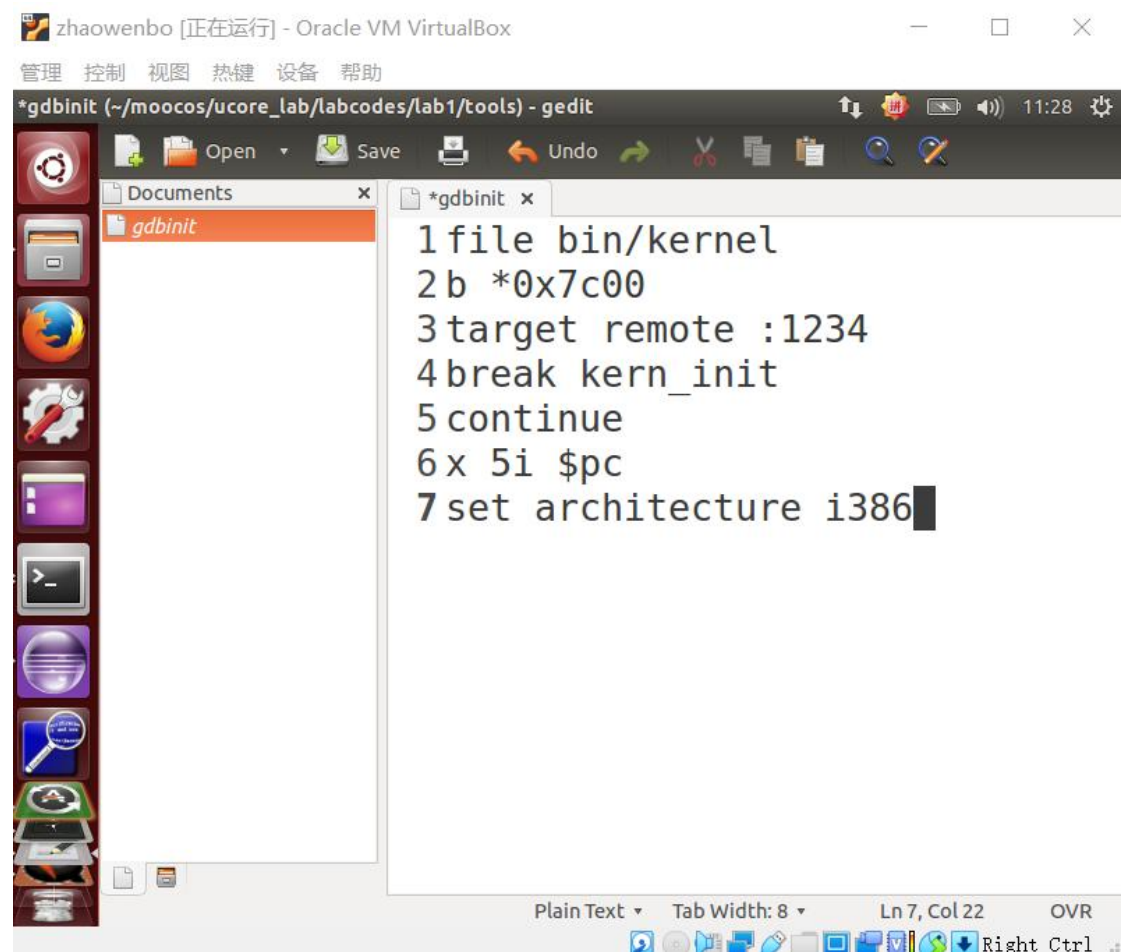
第 510 个（倒数第二个）字节是 0x55,

第 511 个（倒数第一个）字节是 0xAA。

练习 2

问题 1:

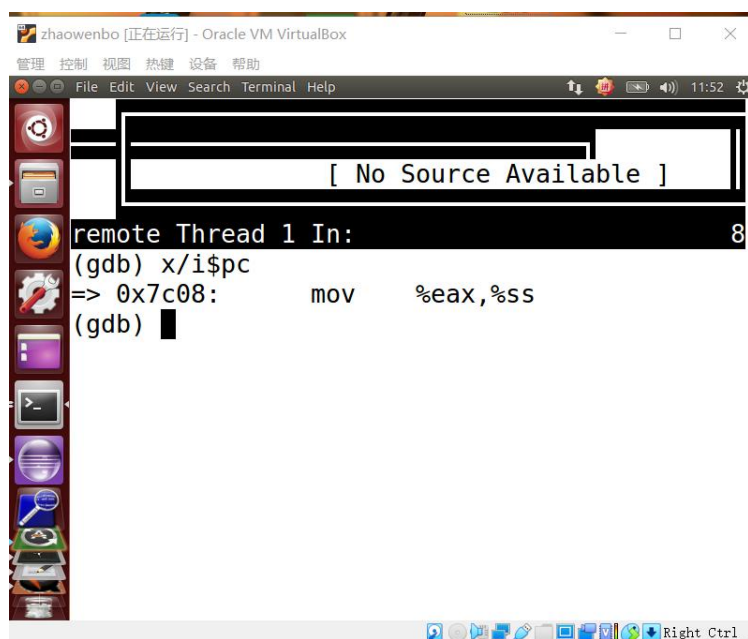
修改 tools 文件



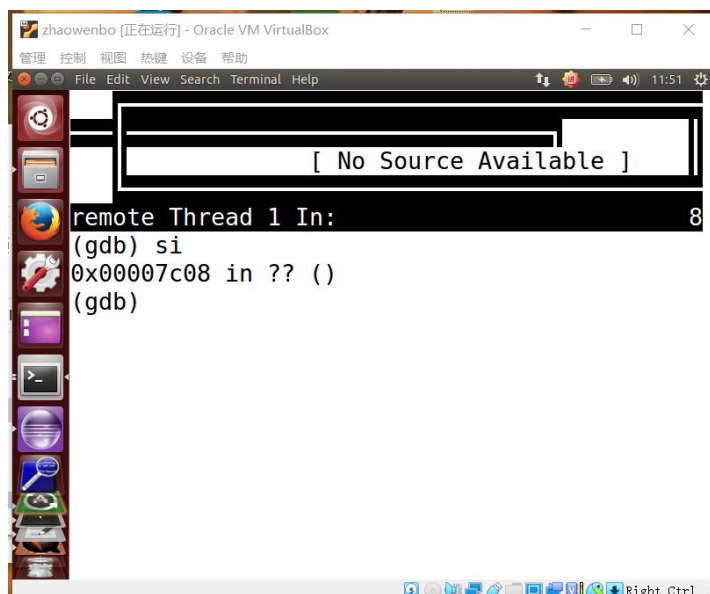
断点停留于 0x00007c00，是最初设置的断点

问题 2:

使用 si 和 x/i\$pc 得到如下跟踪情况



```
zhaowenbo [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
File Edit View Search Terminal Help
[ No Source Available ]
remote Thread 1 In: 8
(gdb) x/i$pc
=> 0x7c08: mov %eax,%ss
(gdb)
```

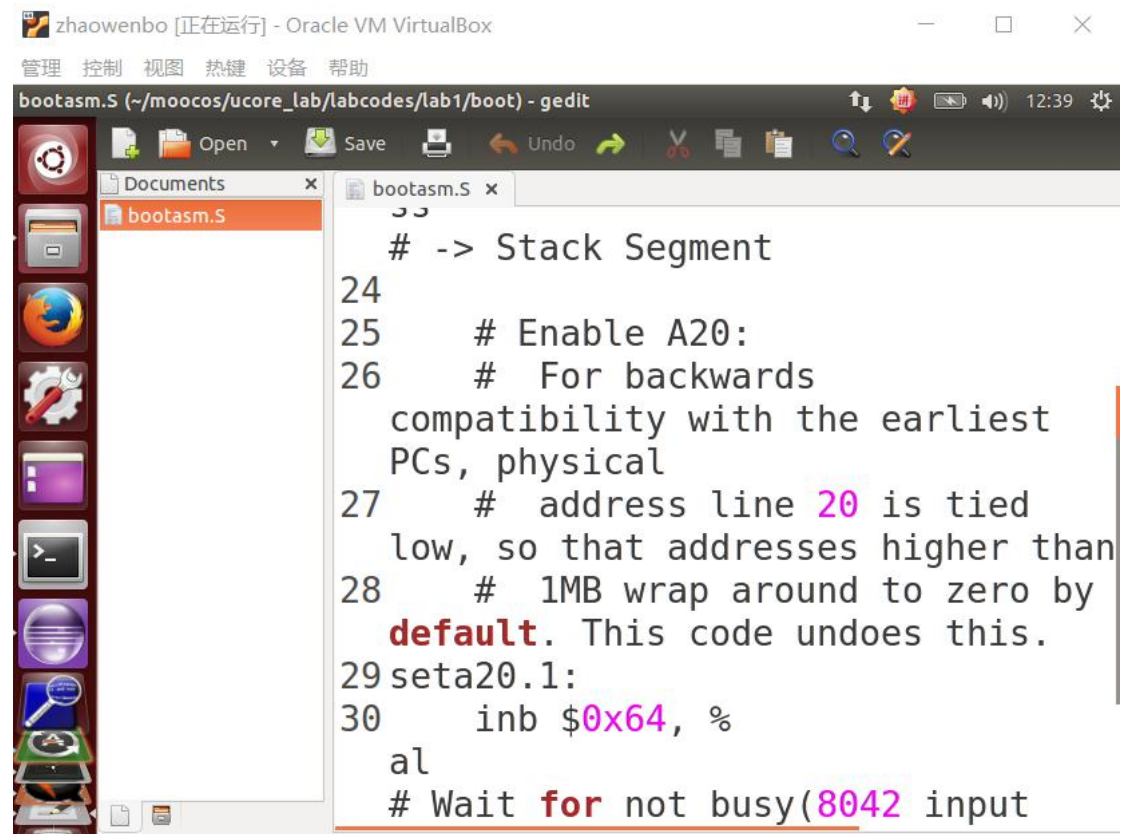


```
zhaowenbo [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
File Edit View Search Terminal Help
[ No Source Available ]
remote Thread 1 In: 8
(gdb) si
0x00007c08 in ?? ()
(gdb)
```

练习 3:

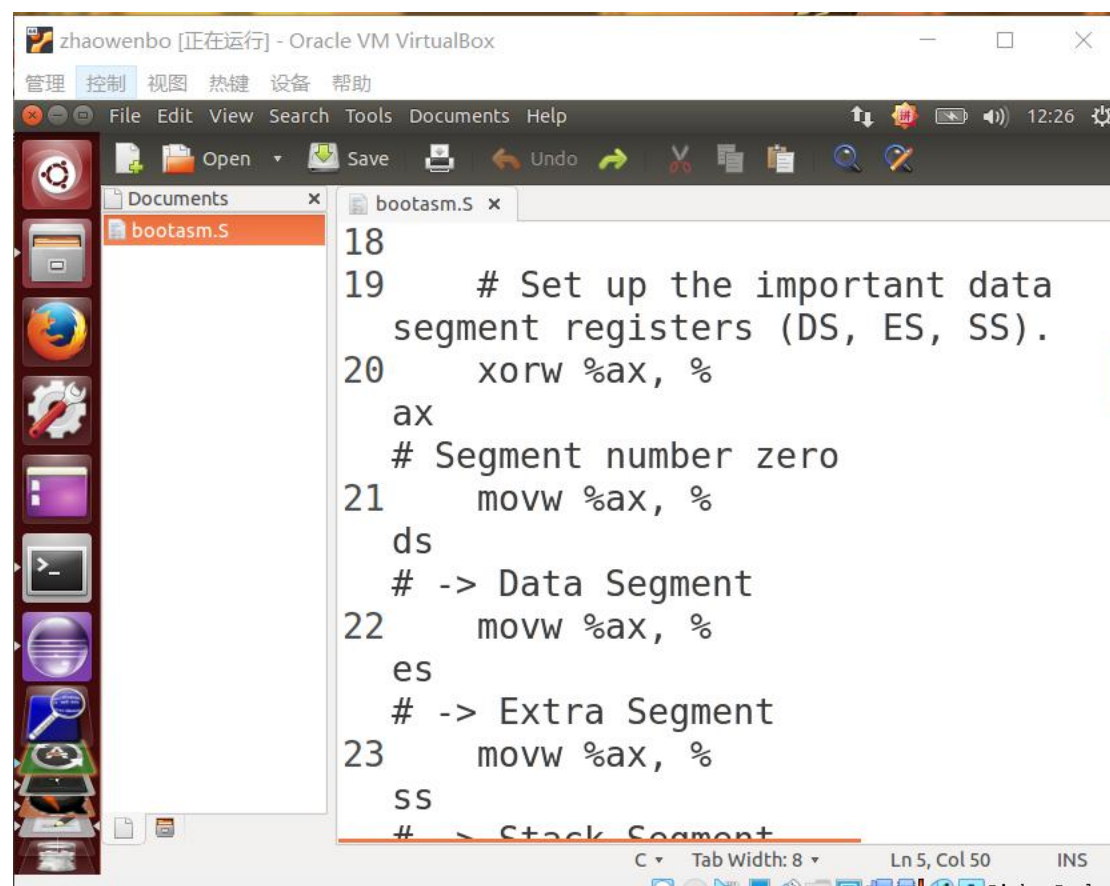
问题 1:

1. 启用 A20: 为了与最早的 PC 向后兼容, 物理地址线 20 在低电平, 因此地址高于 1MB 默认回零。此代码撤消了此操作。使得全部的 32 位地址线可用。



```
bootasm.S (~/.moocos/ucore_lab/labcodes/lab1/boot) - gedit
# -> Stack Segment
24
25     # Enable A20:
26     #   For backwards
        compatibility with the earliest
        PCs, physical
27     #   address line 20 is tied
        low, so that addresses higher than
28     #   1MB wrap around to zero by
        default. This code undoes this.
29 seta20.1:
30     inb $0x64, %
        al
        # Wait for not busy(8042 input
```


1.将各个寄存器归零

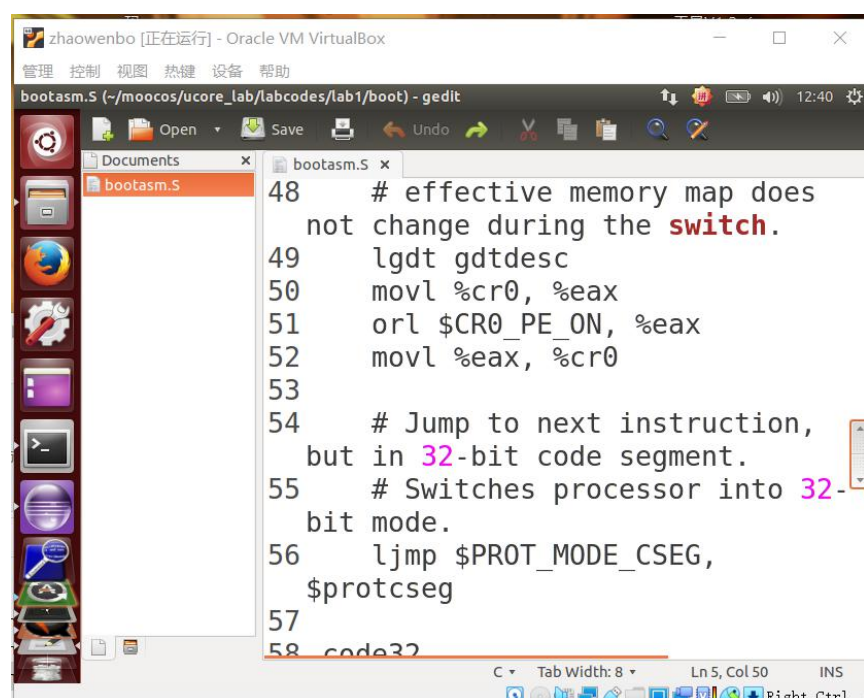


The screenshot shows a VirtualBox window titled 'zhaowenbo [正在运行] - Oracle VM VirtualBox'. Inside, a file editor window titled 'bootasm.S' is open. The code in the editor is as follows:

```
18
19     # Set up the important data
    segment registers (DS, ES, SS).
20     xorw %ax, %
    ax
    # Segment number zero
21     movw %ax, %
    ds
    # -> Data Segment
22     movw %ax, %
    es
    # -> Extra Segment
23     movw %ax, %
    ss
    # -> Stack Segment
```

The status bar at the bottom indicates 'Ln 5, Col 50' and 'INS'.

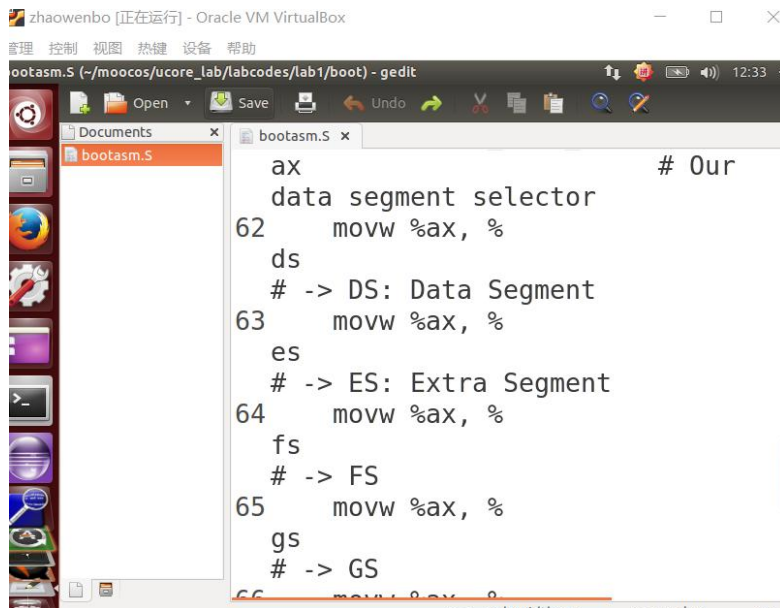
2.加载 gdt 表，并重装寄存器



The screenshot shows a VirtualBox window titled 'zhaowenbo [正在运行] - Oracle VM VirtualBox'. Inside, a file editor window titled 'bootasm.S' is open. The code in the editor is as follows:

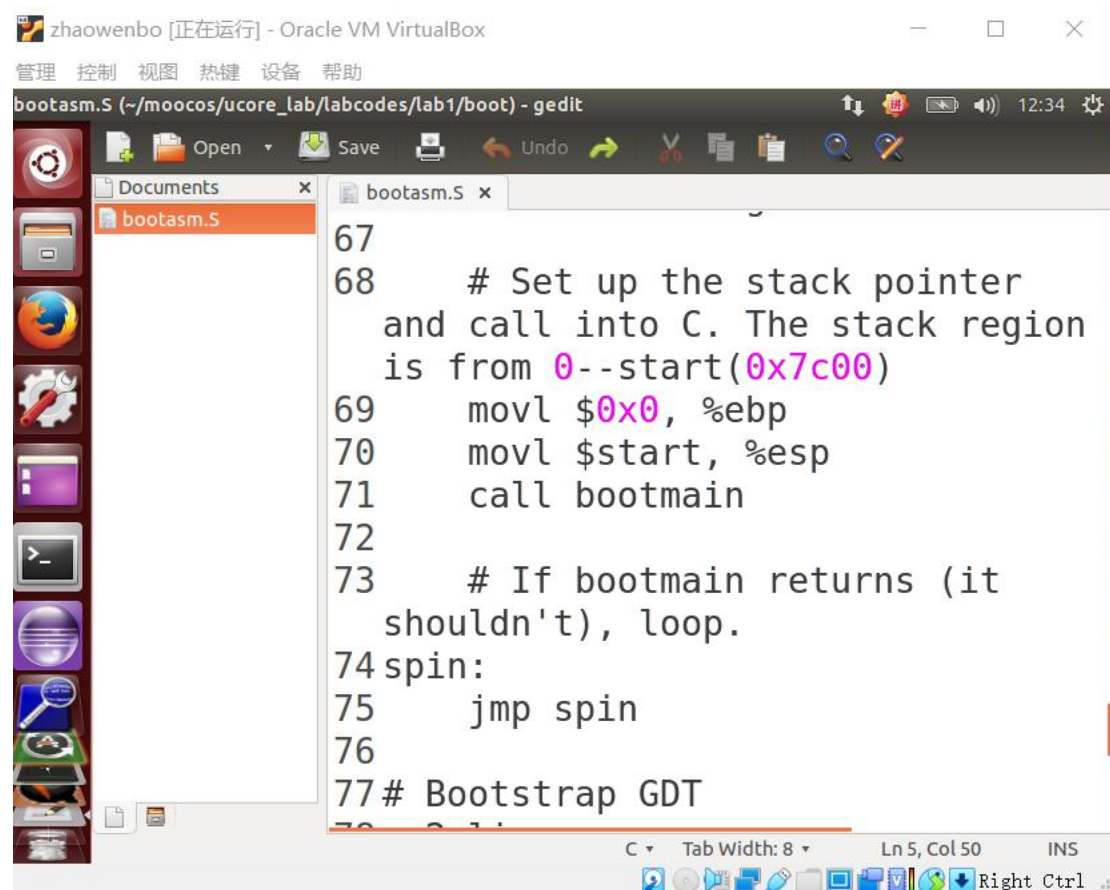
```
48     # effective memory map does
    not change during the switch.
49     lgdt gdt desc
50     movl %cr0, %eax
51     orl $CR0_PE_ON, %eax
52     movl %eax, %cr0
53
54     # Jump to next instruction,
    but in 32-bit code segment.
55     # Switches processor into 32-
    bit mode.
56     ljmp $PROT_MODE_CSEG,
    $protcseg
57
58     code32
```

The status bar at the bottom indicates 'Ln 5, Col 50' and 'INS'.



```
ax                                # Our
data segment selector
62    movw %ax, %
ds
# -> DS: Data Segment
63    movw %ax, %
es
# -> ES: Extra Segment
64    movw %ax, %
fs
# -> FS
65    movw %ax, %
gs
# -> GS
66    movw %ax, %
```

3.进入 bootmain，开始下一步运行



```
67
68    # Set up the stack pointer
    and call into C. The stack region
    is from 0--start(0x7c00)
69    movl $0x0, %ebp
70    movl $start, %esp
71    call bootmain
72
73    # If bootmain returns (it
    shouldn't), loop.
74spin:
75    jmp spin
76
77# Bootstrap GDT
78
```


练习 4:

bootloder 如何读取硬盘扇区:

等待磁盘准备好;

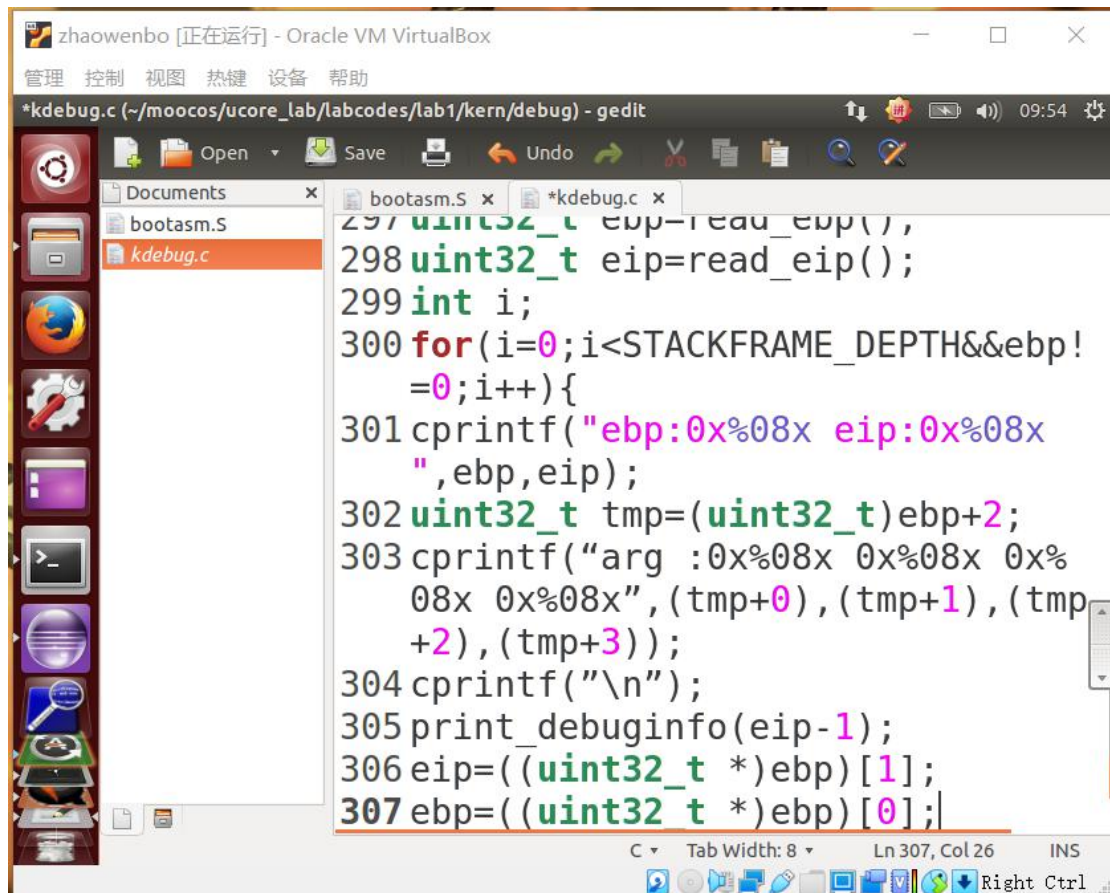
发出读取扇区的命令;

等待磁盘准备好;

把磁盘扇区数据读到指定内存。

练习 5:

代码如下:



The screenshot shows a VirtualBox window titled 'zhaowenbo [正在运行] - Oracle VM VirtualBox'. Inside the VM, a gedit editor is open with the file path '*kdebug.c (~/.moocos/ucore_lab/labcodes/lab1/kern/debug) - gedit'. The editor displays C code for a kernel debug function. The code includes comments in Chinese and uses various data types like 'uint32_t' and 'int'. It reads the stack frame depth and prints debug information. The status bar at the bottom indicates 'Ln 307, Col 26' and 'INS'.

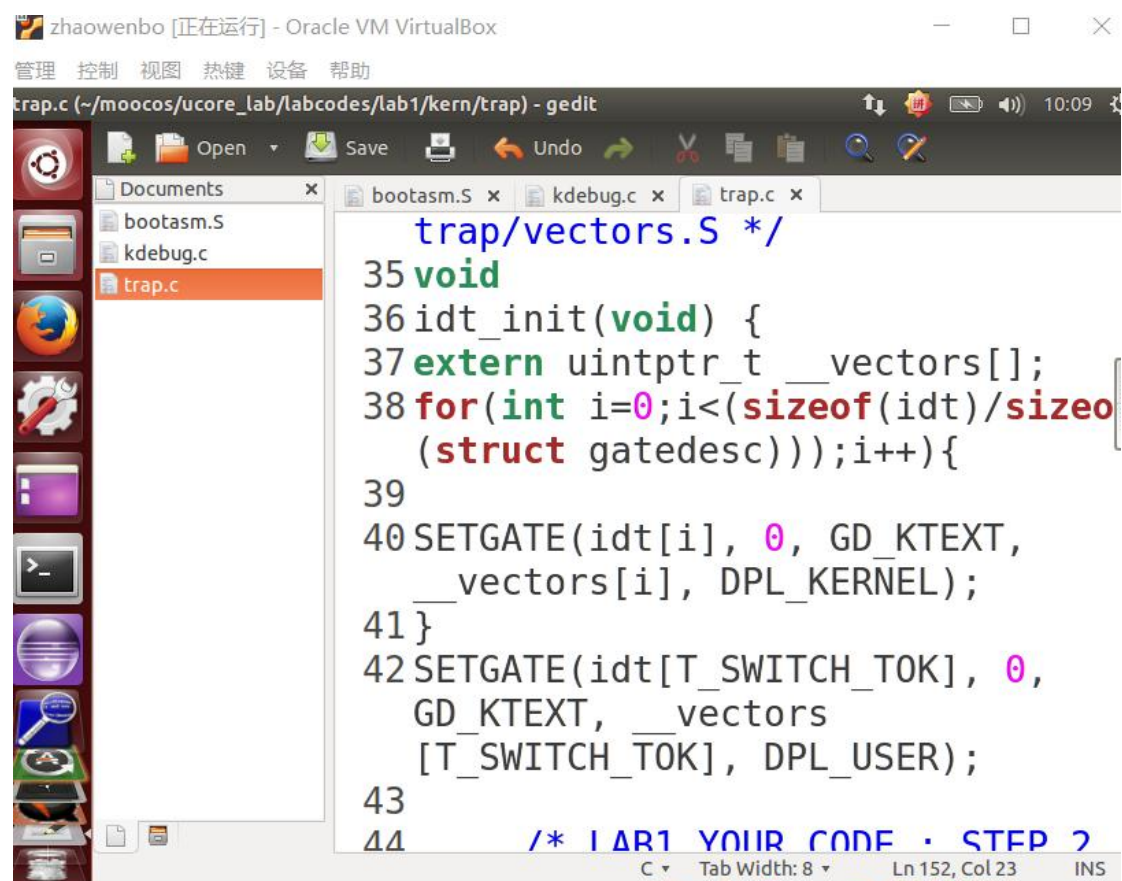
```
297 uint32_t ebp=read_ebp(),
298 uint32_t eip=read_eip();
299 int i;
300 for(i=0;i<STACKFRAME_DEPTH&&ebp!=0;i++){
301 cprintf("ebp:0x%08x eip:0x%08x\n",ebp,eip);
302 uint32_t tmp=(uint32_t)ebp+2;
303 cprintf("arg :0x%08x 0x%08x 0x%08x 0x%08x", (tmp+0), (tmp+1), (tmp+2), (tmp+3));
304 cprintf("\n");
305 print_debuginfo(eip-1);
306 eip=((uint32_t *)ebp)[1];
307 ebp=((uint32_t *)ebp)[0];
```

练习 6:

问题 1:

中断描述符表一个表项占 8 字节。其中 0-15 位和 48-63 位分别为 offset 的低 16 位和高 16 位。16~31 位为段选择子。通过段选择子获得段基址，加上段内偏移量即可得到中断处理代码的入口。

问题 2:

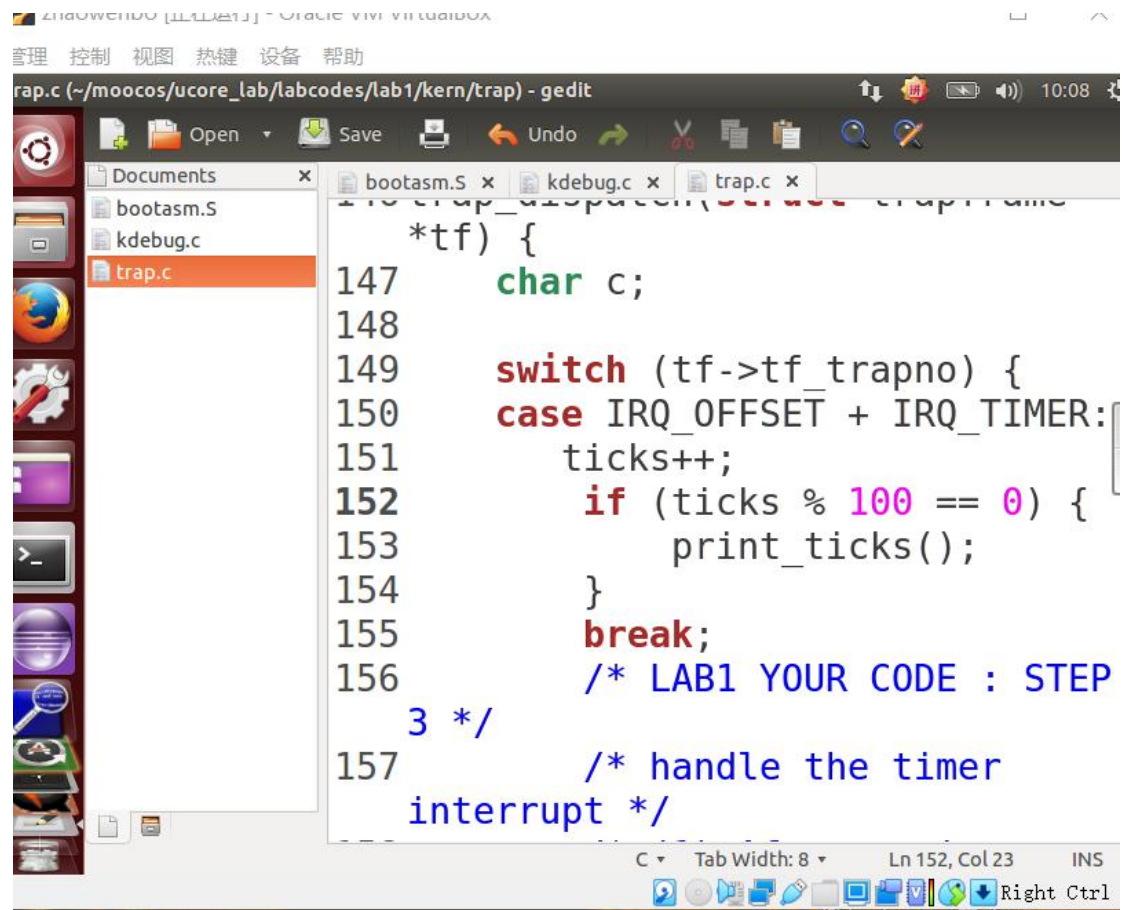


The screenshot shows a virtual machine window titled "zhaowenbo [正在运行] - Oracle VM VirtualBox". Inside the VM, a terminal window titled "trap.c (~/.moocos/ucore_lab/labcodes/lab1/kern/trap) - gedit" is open. The terminal displays the following C code:

```
trap/vectors.S */
35 void
36 idt_init(void) {
37     extern uintptr_t __vectors[];
38     for(int i=0; i<(sizeof(idt)/sizeof(
        (struct gatedesc))); i++){
39
40     SETGATE(idt[i], 0, GD_KTEXT,
        __vectors[i], DPL_KERNEL);
41 }
42 SETGATE(idt[T_SWITCH_TOK], 0,
        GD_KTEXT, __vectors
        [T_SWITCH_TOK], DPL_USER);
43
44     /* LAB1 YOUR CODE - STEP 2
```

The code is written in a C editor with syntax highlighting. The left sidebar shows a file explorer with the following files: Documents, bootasm.S, kdebug.c, and trap.c. The status bar at the bottom indicates "C", "Tab Width: 8", "Ln 152, Col 23", and "INS".

问题 3:



```
trap.c (~/.moocos/ucoore_lab/labcodes/lab1/kern/trap) - gedit
147     char c;
148
149     switch (tf->tf_trapno) {
150     case IRQ_OFFSET + IRQ_TIMER:
151         ticks++;
152         if (ticks % 100 == 0) {
153             print_ticks();
154         }
155         break;
156         /* LAB1 YOUR CODE : STEP
3 */
157         /* handle the timer
interrupt */
```

