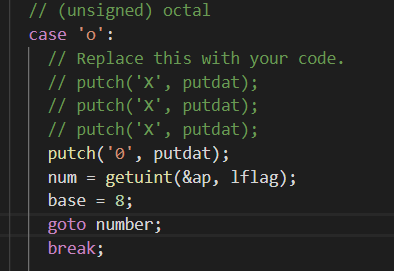
**Lab1 Document**

**5150809010012**

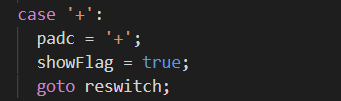
**程浩**

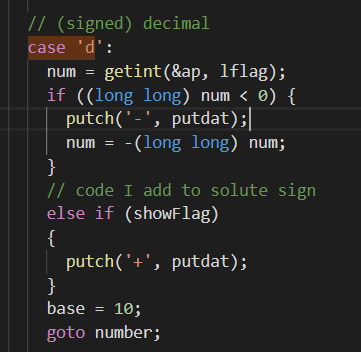
**代码解释：**

1. 实现%o打印八进制数

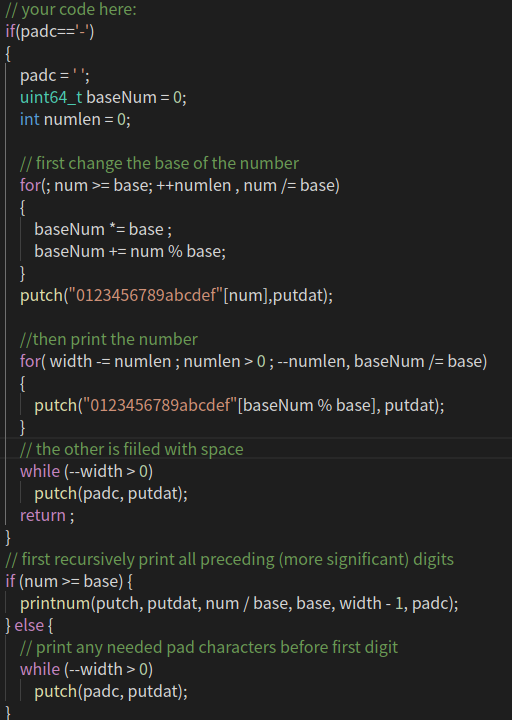


1. 实现%d中，‘d’前有’+‘显示打印的符号（showFlag是检测前面是否有+号以打印出符号）

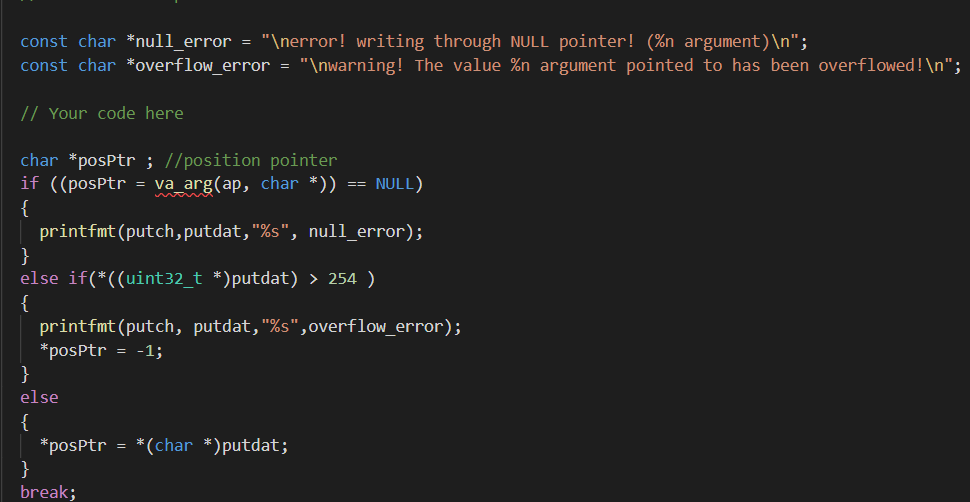




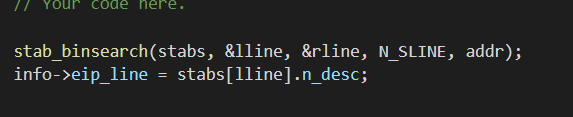
1. 实现%d中，‘d’前有‘-’将所需打印字符居左并以‘-’后的数字个空格填充

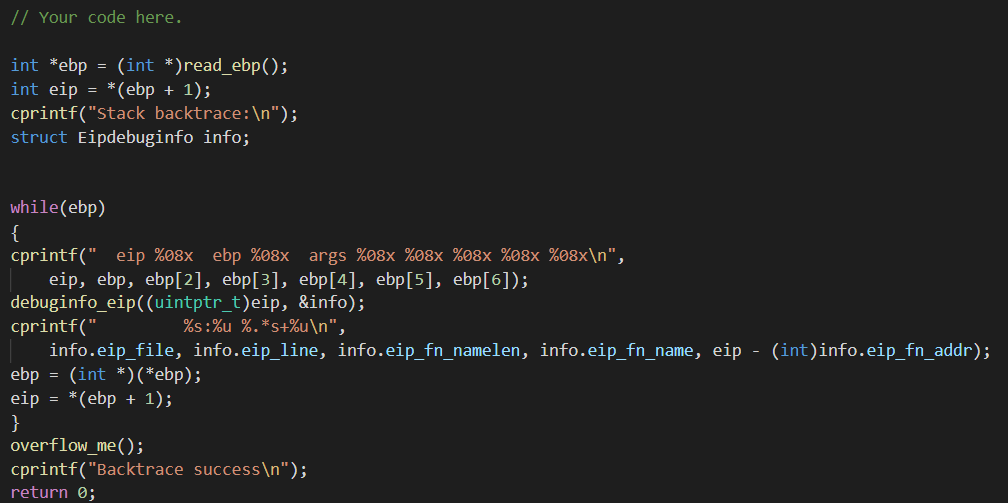


1. 实现%n，打印char\*类型的变量，并添加对空指针和溢出的处理

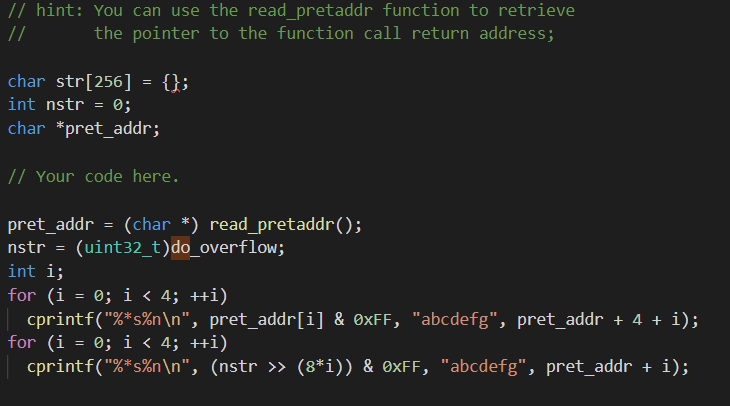


1. 利用kedebug.c中提供的debuginfo\_eip，并对它进行部分修改，实现完整的debuginfo函数，并按照lab1文档中所给的输出格式输出





1. 在do\_overflow中替换return address（hint：



**部分exercise解答：**

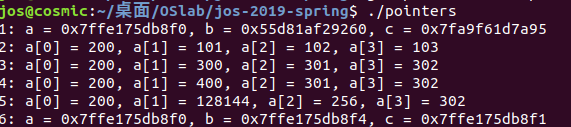
Exercise 2

代码执行后可以发现，BIOS的主要功能就是在控制，初始化，检测各种底层设备，比如时钟，GDTR寄存器，以及设置中断向量表。其中，他作为PC其中后第一段程序，他的重要功能就是负责把操作系统导入内存，然后把控制权交给操作系统。当在磁盘中找打了操作系统，BIOS就会把该磁盘的第一个扇区（启动区）先加载到内存中，其中，一个很重要的boot loader程序会负责将操作系统导入到内存中。

Exercise 3

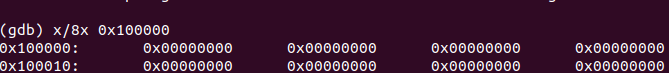
* 首先，在boot.S文件中，计算机工作在实模式下，此时地址为16bit模式；当执行了long jump后（也就是这一句ljmp $PROT\_MODE\_CSEG, $protcseg），正式进入32bit地址模式。
* 它之行的最后一条指令是boot/main.c中bootmain中的最后一条”((void (\*)(void)) (ELFHDR->e\_entry))();”，即跳转到操作系统内核的起始指令处；该程序第一条指令位于kern/entry.S中，“movw $0x1234,0x472 “。
* 操作系统中的扇区信息都是存放在操作系统文件中的program header table中的，在这个表中的每个表项（对应操作系统某一段）的内容包括了这个段的大小，段起始地址的offset等信息，所以找到这个表就能确认要读取多少扇区；这个表一般存放在操作系统内核文件的ELF头部中。

Exercise 4

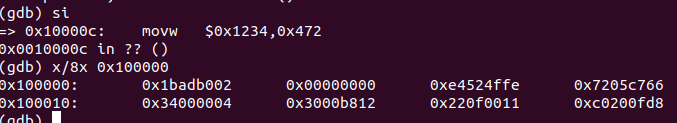


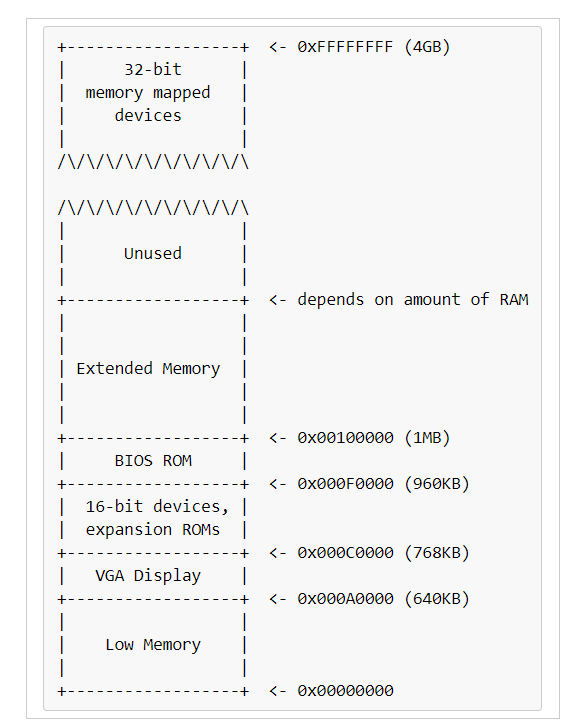
Exercise 5

BIOS entered the boot loader.

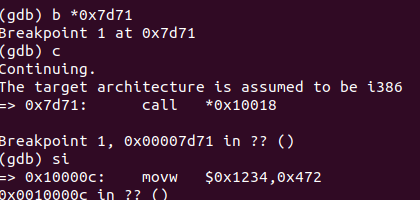


Boot loader entered the kernel.



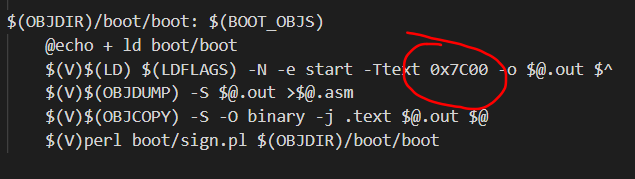


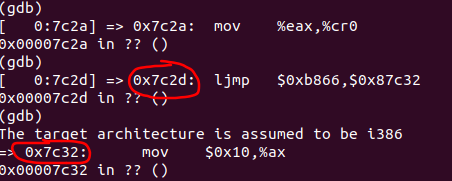
因为在bootmain执行的最后，它会将各个程序段送到地址0x100000处，由上图以及程序入口地址0x10000c可推测，这里面存放的是指令段 .text的内容。



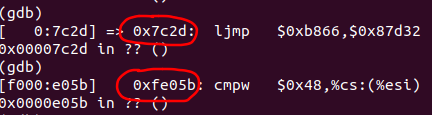
Exercise 6

将Makefrag中的boot loader链接地址从原先的0x7C00修改为0x7D00





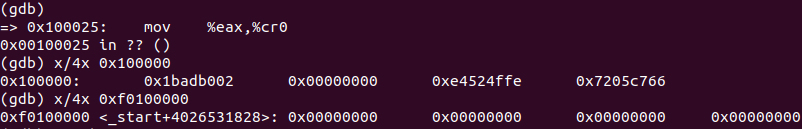
修改前



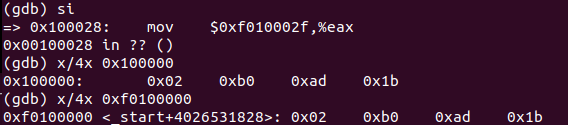
修改后

可以发现在long jump时发生了错误，跳到不知道哪儿去了。

Exercise 7

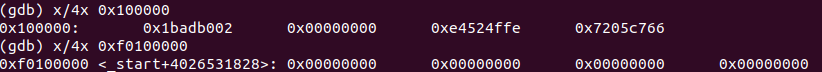


movl %eax, %cr0执行前

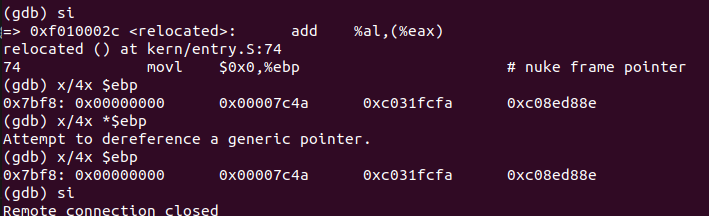


movl %eax, %cr0执行后

发现，在执行过后存放在内核虚拟地址0xf0100000中的内容已经映射到了物理地址0x100000中；接着，注释掉movl %eax, %cr0，编译后调试



由于此时没有将内核的虚拟地址值映射到低地址，所以执行前后无变化



但是在执行到movl $0x0,%ebp这个语句时发生了错误，原因是超出了访存范围。

Exercise 8&9

1. Console.c export了除static修饰的函数，其中printf中的putch()调用了console.c中的cputchar。