实验一：系统软件启动过程

练习1：理解通过make生成执行文件的过程。

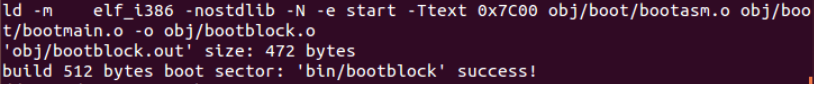
1. 操作系统镜像文件ucore.img是如何一步一步生成的

在编译时使用make V=，相当于设置一个标记，把make编译执行的过程全部展示出来。

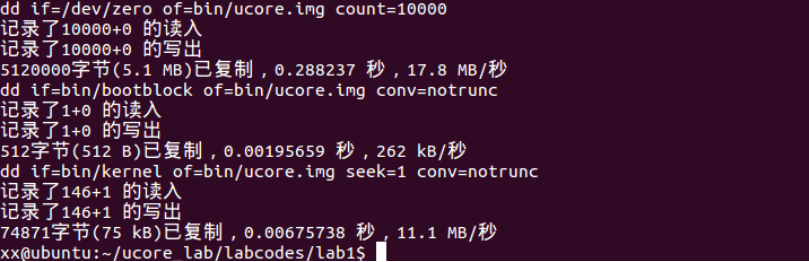
首先，调用了GCC，把C的源代码编译成.o的目标文件。



然后，调用了ld，把这些目标文件换成一个可执行程序，例如，转换成为bootloader的一个执行程序bootblock.out。

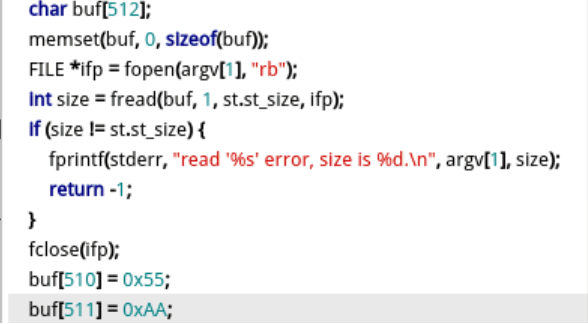


最后，调用了dd，把bootloader、bootblock和kernel放到虚拟的硬盘uCore.img里面。



1. 一个被系统认为是符合规范的硬盘主引导扇区的特征是什么

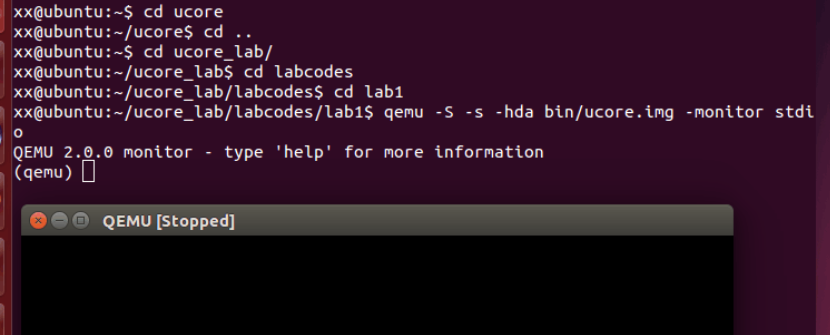
在sign.c中，先申请了一个512字节的空间buf，然后将buf初始化为全0，再将主引导程序写入这个空间，最后，在buf的最后两个字节写入55AA。所以硬盘的主引导扇区的特征是主引导扇区的512个字节的最后两个字节是55AA。



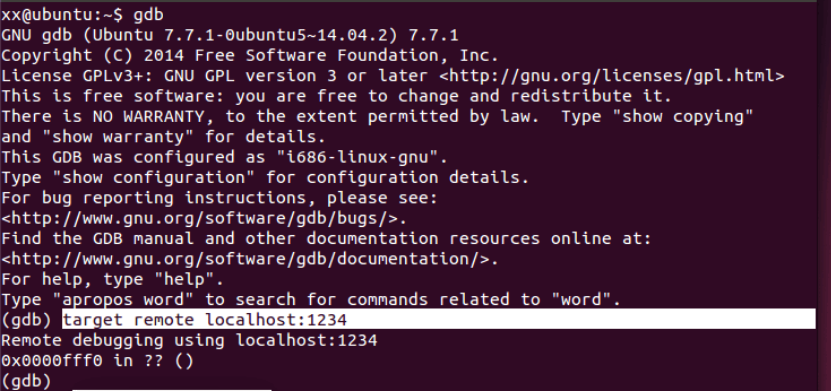
练习2：使用qemu执行并调试lab1中的软件。

1. 从CPU加电后执行的第一条指令开始，单步跟踪BIOS的执行。

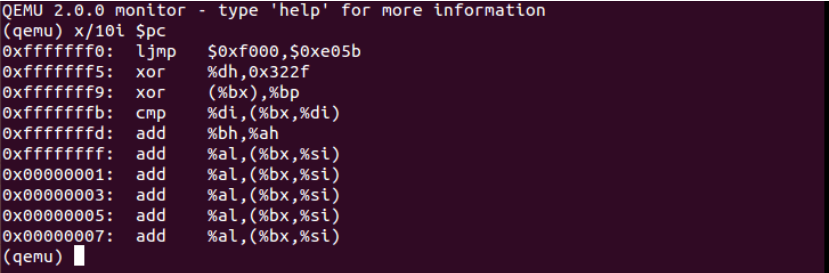
启动qemu，并让它进入-S状态。

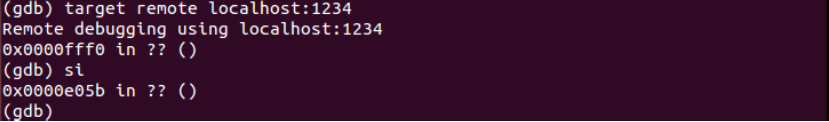


开启另外一个终端，执行gdb命令，绑定端口1234。



在QEMU窗口中使用 x/10i $pc 查看最近10条指令的反汇编内容。在gdb中使用si命令执行单步，显示位置。可以看到一启动，就处于0xfffffff0的位置，第一条指令是个长跳转指令。



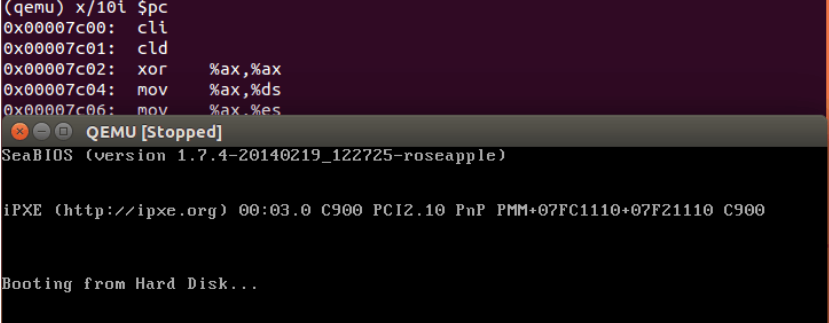


1. 在初始化位置0x7c00设置实地址断点,测试断点正常。

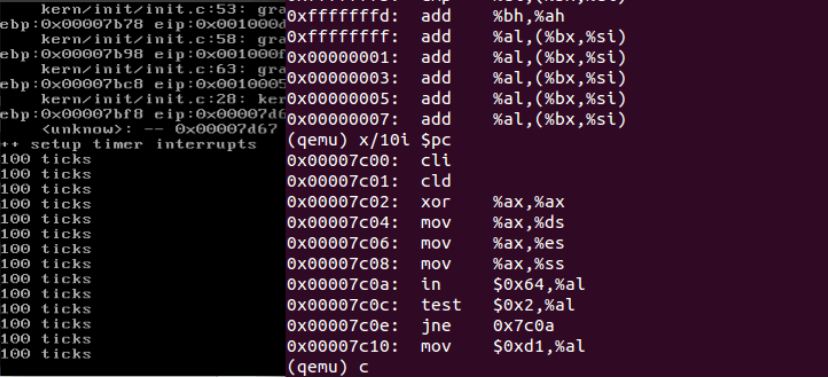
输入b\*0x7c00设置断点，输入c执行到断点。



输入x/10i$pc 查看最近10条指令的反汇编内容，可以看到qemu执行到断点0x7c00.断点工作正常。

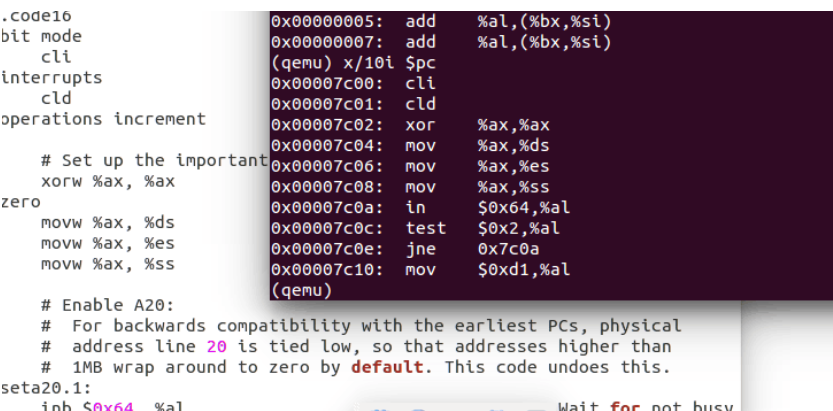


再次输入c执行，qemu继续工作。得到结论，断点工作正常。



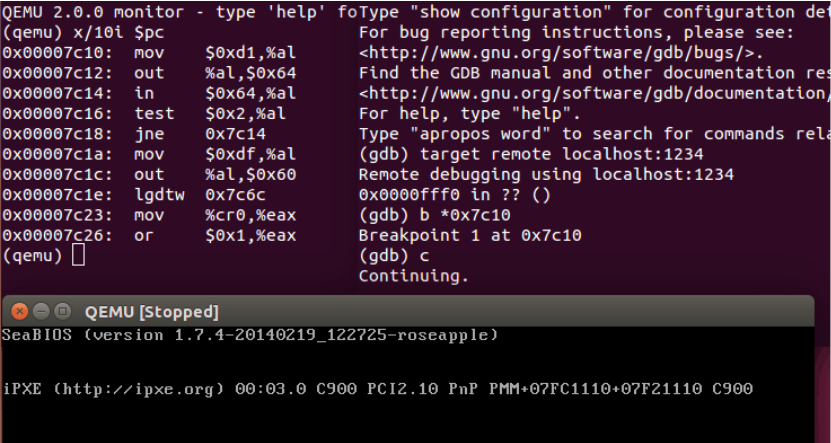
3. 从0x7c00开始跟踪代码运行,将单步跟踪反汇编得到的代码与bootasm.S和 bootblock.asm进行比较。

Gdb得到的反汇编代码与boottasm.S和bootblock.asm中的代码基本一致。

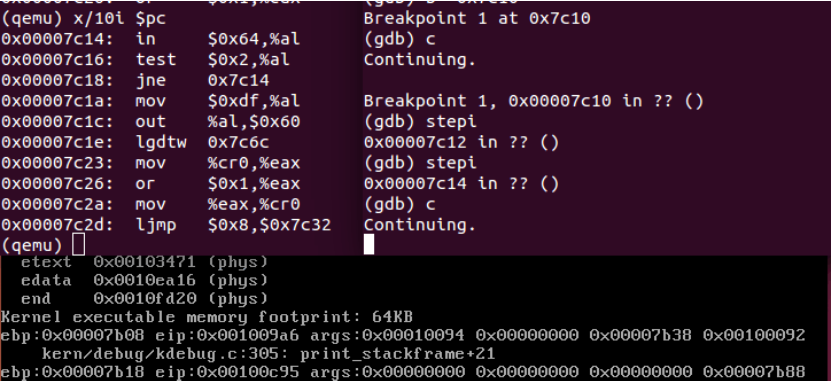


4. 自己找一个bootloader或内核中的代码位置，设置断点并进行测试。

输入b\*0x7c10设置断点，输入c执行到断点。输入x/10i$pc 查看最近10条指令的反汇编内容。



输入stepi，单步执行一条机器指令。再一次输入x/10i $pc 查看最近10条指令的反汇编内容。最后，输入c，qemu继续工作。

练习3：分析bootloader进入保护模式的过程。

从%cs=0$，pc=0x7c00进入bootloader。

.globl start

Start:

准备：将中断标志位清0，不允许中断，设置增址，将段寄存器置0。

.code16:

cli

cld

xorw %ax,%ax

movw %ax,%ds

movw %ax,%es

movw %ax,%ss

开启A20：通过将键盘控制器上的A20线置于高电位，使全部32条地址线可用，进而可以访问4G的内存空间。

seta20.1：#等待8042键盘控制器不忙

inb $0x64, %al

testb $0x2,%al

jnz seta20.1

movb $0xd1,%al #发送写8042输出端口的指令

outb %al,$0x64

seta20.2: #等待8042键盘控制器不忙

inb $0x64, %al

testb $0x2,%al

jnz seta20.2

movb $0xdf,%al #打开A20

outb %al,$0x60

加载GDTR、初始化GDT表：GDT表和其描述符已经静态储存在引导区中，载入即可。

lgdt gdtdesc

进入保护模式：通过将cr0寄存器PE位置1便开启了保护模式。

movl %cr0,%eax

orl $CR0\_PE\_ON,%eax

movl %eax,%cr0

通过长跳转更新cs的基地址。

ljmp $PROT\_MODE\_CSEG,%a

.code32

protcseg:

设置段寄存器，并建立堆栈。

movw $PROT\_MODE\_DSEG,%ax

movw %ax,%ds

movw %ax,%es

movw %ax,%fs

movw %ax,%gs

movw %ax,%ss

movl $0x0,%ebp

movl $start,%esp

完成进入保护模式，进入bootmain。

call bootmain