**Lab1 171491121 硬件一班 王倩倩**

**实验一：系统软件启动过程**

**实验目的**：

操作系统是一个软件，也需要通过某种机制加载并运行它。在这里我们将通过另外一个更加 简单的软件-bootloader来完成这些工作。为此，我们需要完成一个能够切换到x86的保护模式 并显示字符的bootloader，为启动操作系统ucore做准备。lab1提供了一个非常小的bootloader 和ucore OS，整个bootloader执行代码小于512个字节，这样才能放到硬盘的主引导扇区中。 通过分析和实现这个bootloader和ucore OS，读者可以了解到：

**计算机原理**

1. CPU的编址与寻址: 基于分段机制的内存管理

2.CPU的中断机制

3.外设：串口/并口/CGA，时钟，硬盘

**Bootloader软件**

编译运行bootloader的过程

调试bootloader的方法

PC启动bootloader的过程

ELF执行文件的格式和加载

外设访问：读硬盘，在CGA上显示字符串

***实验内容：***

lab1中包含一个bootloader和一个OS。这个bootloader可以切换到X86保护模式，能够读磁盘 并加载ELF执行文件格式，并显示字符。而这lab1中的OS只是一个可以处理时钟中断和显示 字符的幼儿园级别OS。

**对实验报告的要求：**

**\***基于markdown格式来完成，以文本方式为主。

\*填写各个基本练习中要求完成的报告内容

**\***完成实验后，请分析ucore\_lab中提供的参考答案，并请在实验报告中说明你的实现与参考答案的区别

**\***列出你认为本实验中重要的知识点，以及与对应的OS原理中的知识点，并简要说明你对 二者的含义，关系，差异等方面的理解（也可能出现实验中的知识点没有对应的原理知 识点）

**\***列出你认为OS原理中很重要，但在实验中没有对应上的知识点

**练习1**：理解通过make生成执行文件的过程。（要求在报告中写 出对下述问题的回答）

列出本实验各练习中对应的OS原理的知识点，并说明本实验中的实现部分如何对应和体现了 原理中的基本概念和关键知识点。

在此练习中，大家需要通过静态分析代码来了解：

1. 操作系统镜像文件ucore.img是如何一步一步生成的？(需要比较详细地解释Makefile中每 一条相关命令和命令参数的含义，以及说明命令导致的结果) 2. 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？

补充材料：

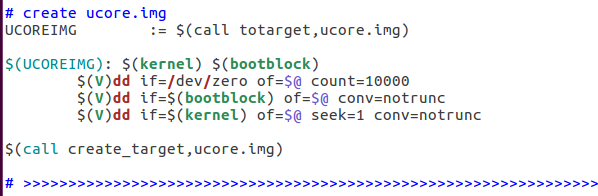
如何调试Makefile 当执行make时，一般只会显示输出，不会显示make到底执行了哪些命令。

如想了解make执行了哪些命令，可以执行：

$ make "V="

要获取更多有关make的信息，可上网查询，并请执行

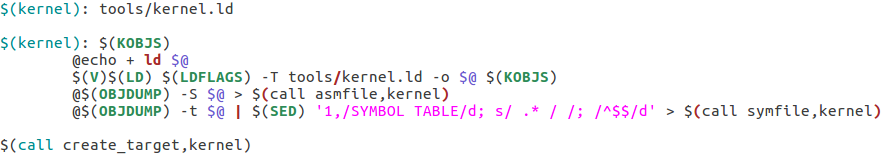
$ man make



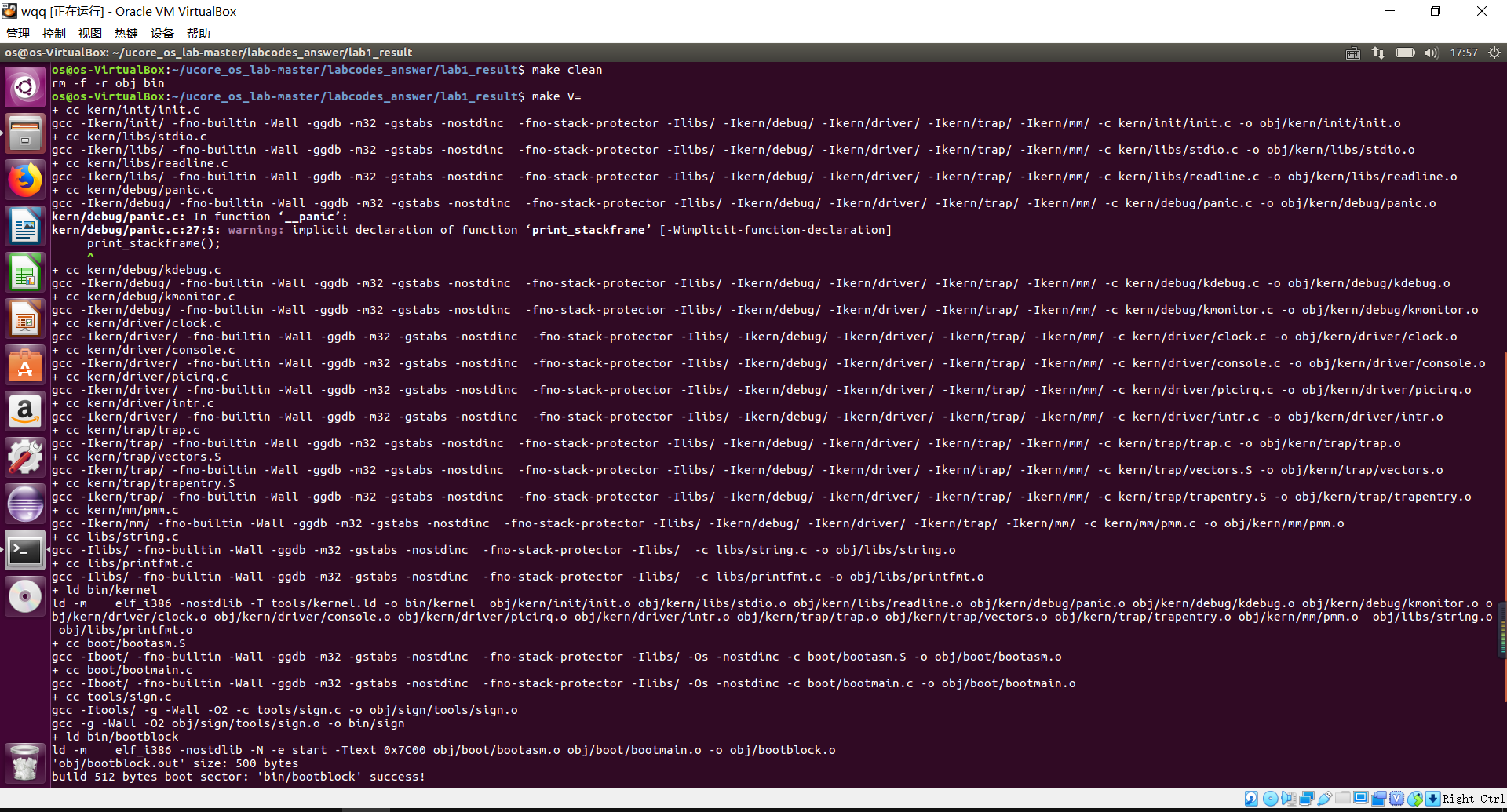
这个代码的意思是：首先创建一个大小为10000字节的块儿，然后将bootblock复制进去。

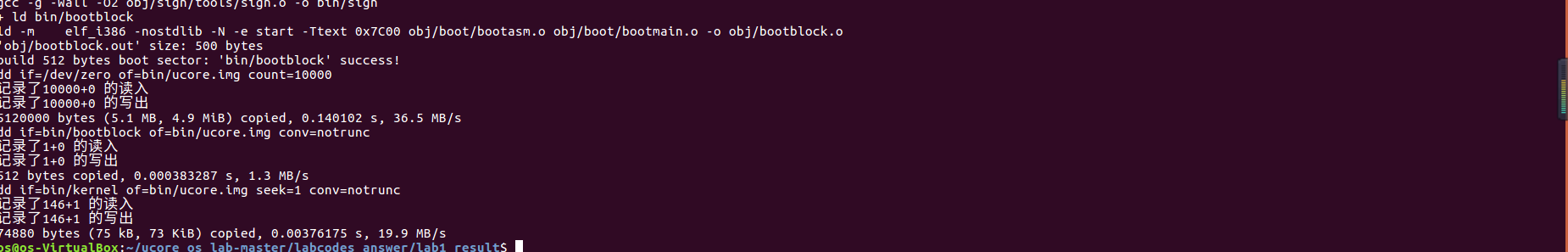
我们从makefile中最后的结果来反推一下这个过程，由此我们可以知道，要生成ucore.img需要kernel和bootblock

那如何生成kernel呢？

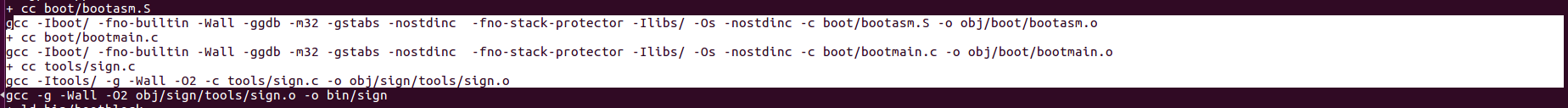


上面是生成kernel的代码：我们通过make V=来看一下执行了那些命令



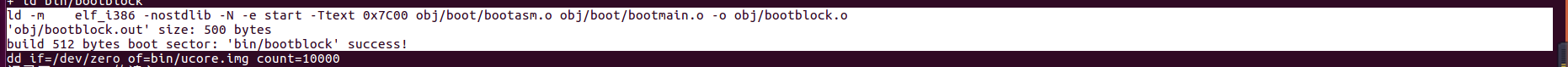


从这之中，我们可以看到



它调用了gcc，gcc来把一些c的源代码编译成了.o文件，也就是目标文件

然后通过ld，ld把这些目标文件会转换成一个执行程序，比如说其中的bootblock.out，这个就可以理解为是一个bootloader一个执行程序



【-m 模拟指定的连接器  
-nostdlib 不使用标准库  
-T 指定命令文件  
-o 指定输出文件的名称】

还有这个dd，是可以把这个bootloader放到一个虚拟的硬盘中去，生成一个硬盘叫ucore.img count,硬件模拟器就会基于这个虚拟硬盘中的数据来执行相应的这个代码



下图则是生成了两个代码，第一个是Bootloader,第二个叫Kernel Kernel事实上是Ucore的组成部分



**练习2**：使用qemu执行并调试lab1中的软件。（要求在报告中简 要写出练习过程）

为了熟悉使用qemu和gdb进行的调试工作，我们进行如下的小练习：

1. 从CPU加电后执行的第一条指令开始，单步跟踪BIOS的执行。

2. 在初始化位置0x7c00设置实地址断点,测试断点正常。

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。

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。

提示：参考附录“启动后第一条执行的指令”，可了解更详细的解释，以及如何单步调试和 查看BIOS代码。

提示：查看 labcodes\_answer/lab1\_result/tools/lab1init 文件，用如下命令试试如何调试 bootloader第一条指令：

$ cd labcodes\_answer/lab1\_result/

$ make lab1-mon

补充材料： 我们主要通过硬件模拟器qemu来进行各种实验。在实验的过程中我们可能会遇上 各种各样的问题，调试是必要的。qemu支持使用gdb进行的强大而方便的调试。所以用好 qemu和gdb是完成各种实验的基本要素。

默认的gdb需要进行一些额外的配置才进行qemu的调试任务。qemu和gdb之间使用网络端口 1234进行通讯。在打开qemu进行模拟之后，执行gdb并输入

target remote localhost:1234

即可连接qemu，此时qemu会进入停止状态，听从gdb的命令。

另外，我们可能需要qemu在一开始便进入等待模式，则我们不再使用make qemu开始系统的 运行，而使用make debug来完成这项工作。这样qemu便不会在gdb尚未连接的时候擅自运行 了。

gdb的地址断点

在gdb命令行中，使用b \*[地址]便可以在指定内存地址设置断点，当qemu中的cpu执行到指定 地址时，便会将控制权交给gdb。

关于代码的反汇编

有可能gdb无法正确获取当前qemu执行的汇编指令，通过如下配置可以在每次gdb命令行前强 制反汇编当前的指令，在gdb命令行或配置文件中添加：

define hook-stop

x/i $pc

End

gdb的单步命令

在gdb中，有next, nexti, step, stepi等指令来单步调试程序，他们功能各不相同，区别在于单 步的“跨度”上。

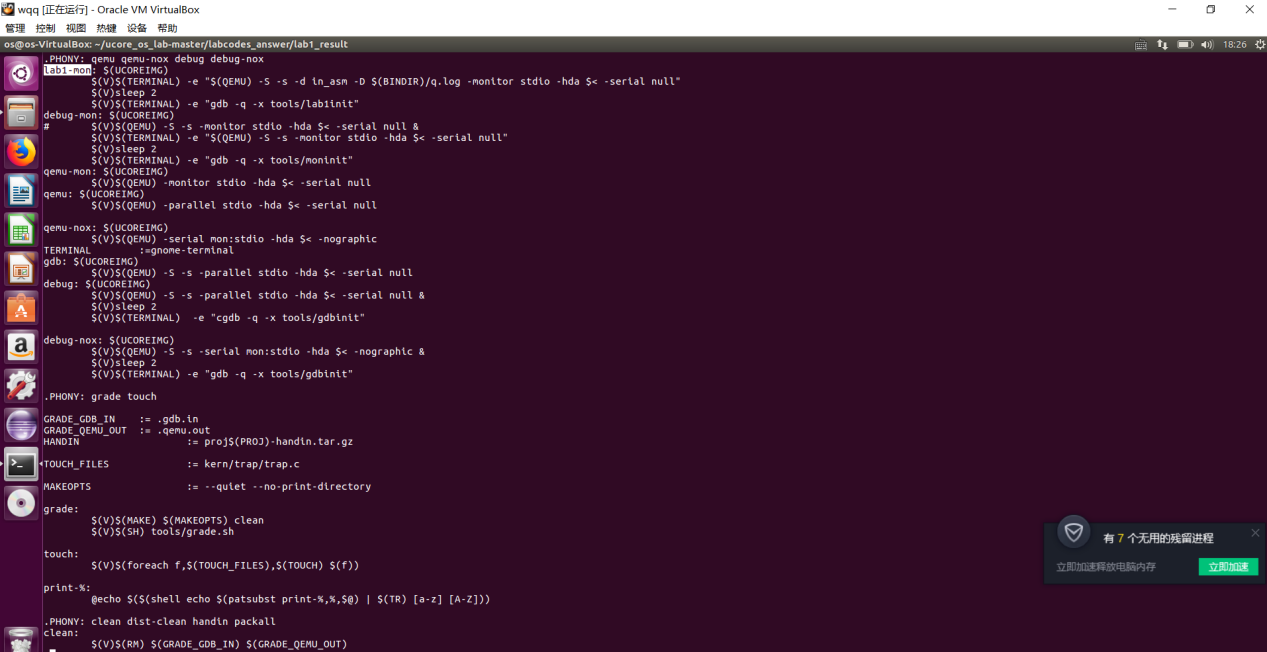
next 单步到程序源代码的下一行，不进入函数。

nexti 单步一条机器指令，不进入函数。

step 单步到下一个不同的源代码行（包括进入函数）。

stepi 单步一条机器指令。

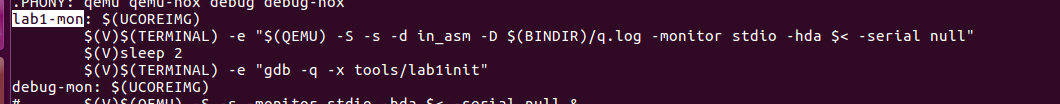
在终端 先输入make clean，在输入 less Makefile，接着输入/lab1-mon，就会出现



第一个是让qemu把它执行的指令给记录下来，把log信息给记录下来，放到那个q.log

再就是跟GDB结合来调试正在执行的Bootloader，现在我们还在Bootloader阶段

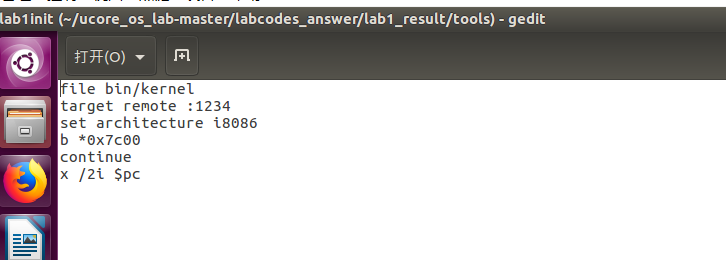
“gdb -q -x tools/lab1init”这个是一些初始化的一些执行指令



gdb -q -x tools/lab1init 下图是这个指令中的内容（我太菜了 我自己手动打开的）

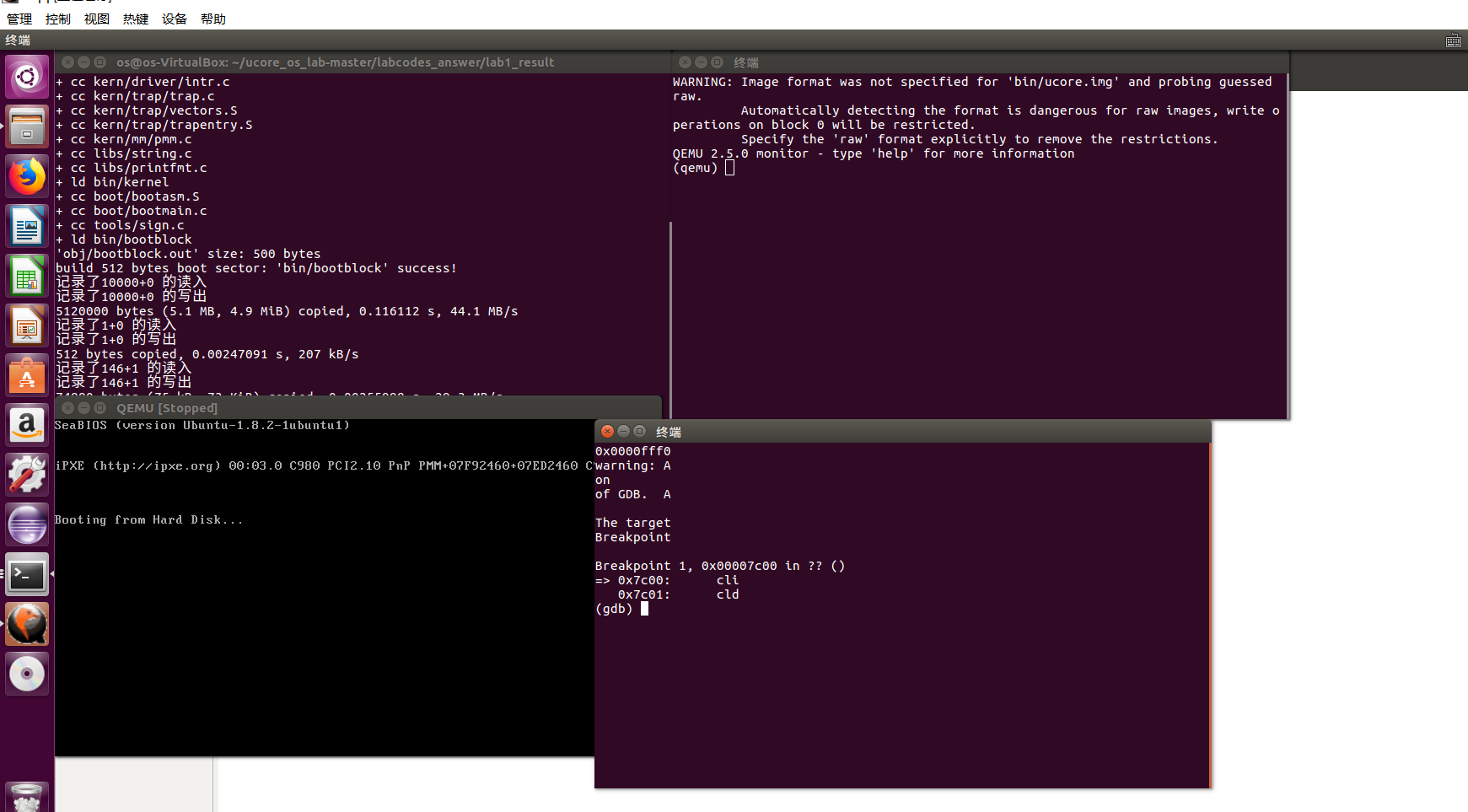
下图是GDB能够识别的一些命令，第一个 加载并kernel 加载符号信息 。第二条指令是与qemu进行连接，刚开始的时候BIOS是进入8086的16位实模式，一直到OX7c00在BIOS这个阶段启动，最后把Bootloader加载进去，把控制权交给bootloader，bootloader第一条指令就是在ox7c00处，break ox7c00，然后让这个程序继续运行

X/2i $pc把pc就是EIP就是指令的指针寄存器，存在当前正在执行的这个指令的地址，x是显示的意思，/2i是显示两条指令的意思

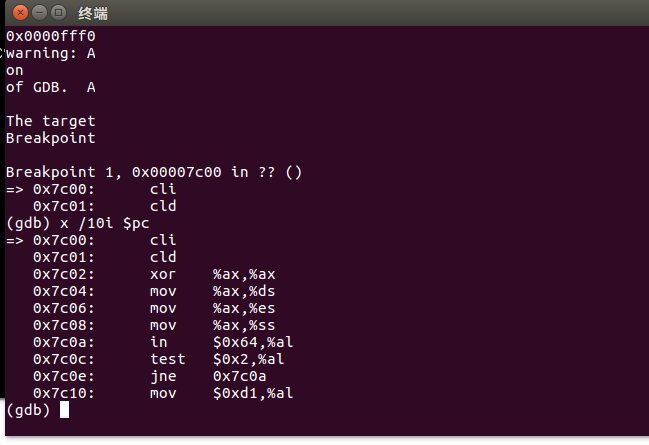


我们在lab\_result下在终端打开，然后执行make lab1-mon ，显示两个窗口，一个是qemu

但是它断下来了，断在ox7c00处



输入 x /10i $pc 可以把十条指令都执行出来



输入continue ，让它继续运行，

**练习3**：分析bootloader进入保护模式的过程。（要求在报告中 写出分析）

BIOS将通过读取硬盘主引导扇区到内存，并转跳到对应内存中的位置执行bootloader。请分 析bootloader是如何完成从实模式进入保护模式的。

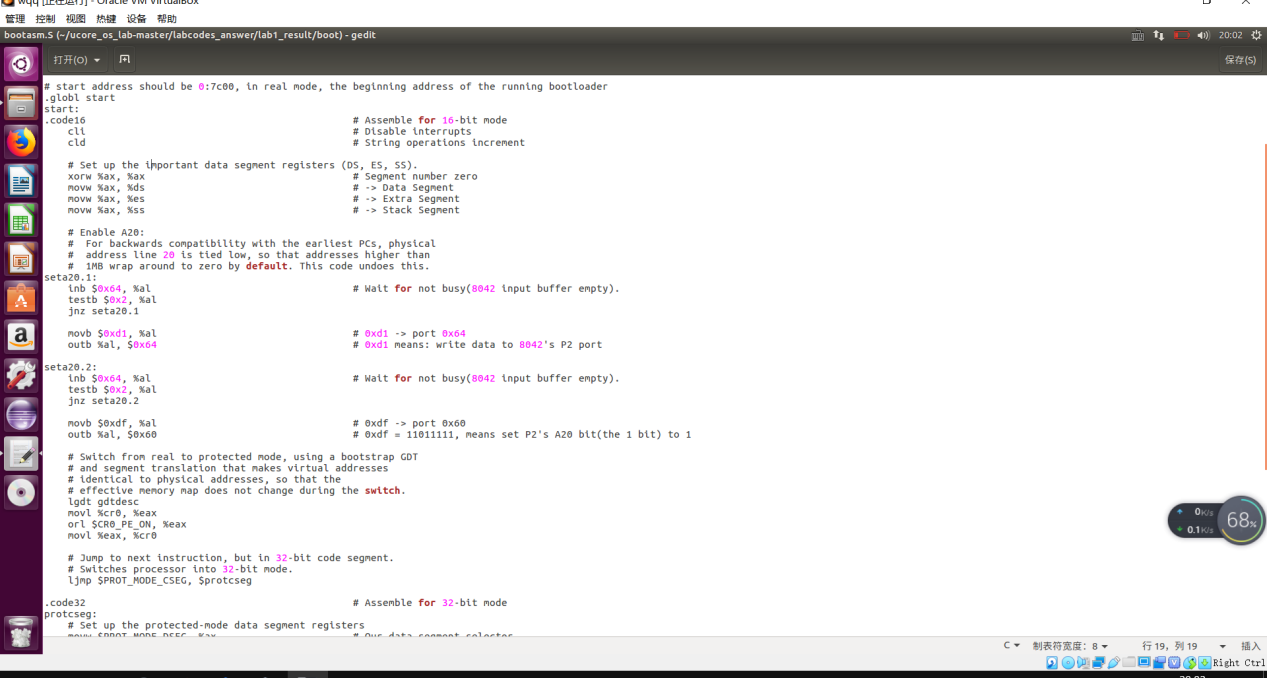
提示：需要阅读小节“保护模式和分段机制”和lab1/boot/bootasm.S源码，了解如何从实模式 切换到保护模式，需要了解：

**\***为何开启A20，以及如何开启A20

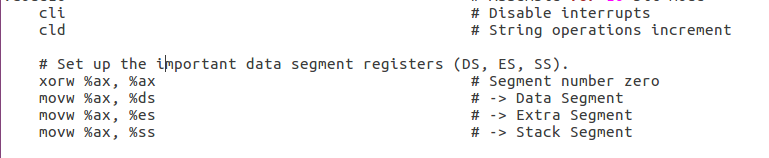
**\***如何初始化GDT表

**\***如何使能和进入保护模式

Bootloader完成了一些基本的功能，比如说把80386的保护模式开启使得现在的软件进入了一个32位寻址空间，寻址方式发生了改变

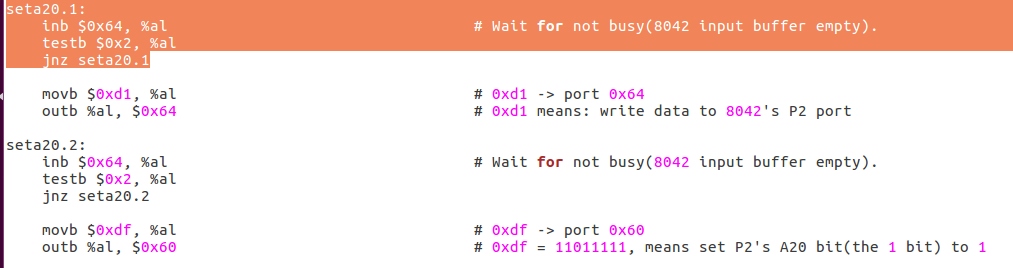


1. 关闭中断，将各个寄存器置为0



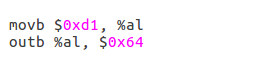
1. 开启A20

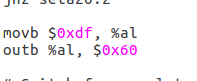
A20的地址线控制禁止时，则程序就像在8086中运行，1mb以上的地是不可访问的。而在保护模式下，A20地址线控制是要打开的，所以要通过键盘控制器上的A20线置于高电位，使得全部32位地址线可用。



#读取状态寄存器，等待8042键盘控制器闲置

#判断输入缓存是否为空

# 0xd1表示写输出端口命令，参数随后通过0x60端口写入

# 通过0x60写入数据11011111 即将A20置1

1. 加载GDT 表



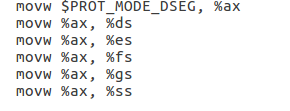
1. 将CR0的第0位置1



1. 长跳转到32位代码段，重装cs和EIP



1. 重装DS,ES 等段寄存器等



1. 转到保护模式完成，进入boot主方法

