OS实验一 系统软件启动过程

171491328 罗晓璐

一、实验目的：

操作系统是一个软件，也需要通过某种机制去加载运行它。在这里，我们通过软件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。通过该实验，读者可以了解到：

计算机原理：

CPU的编址与寻址：基于分段机智的内存管理

CPU的中断机制

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

ucore OS 软件：

编译运行ucore OS的过程

ucore OS的启动过程

调试ucore OS的方法

函数调用关系：在汇编级了解用栈的结构和处理过程

中断管理：与软件相关的中断处理

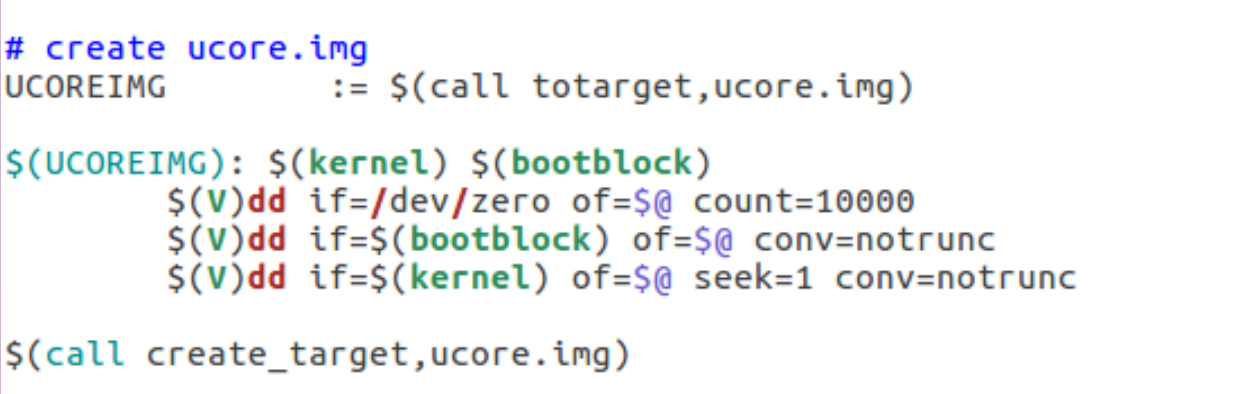
外设管理：时钟

二、实验内容：

1、理解通过make生成执行文件的过程。

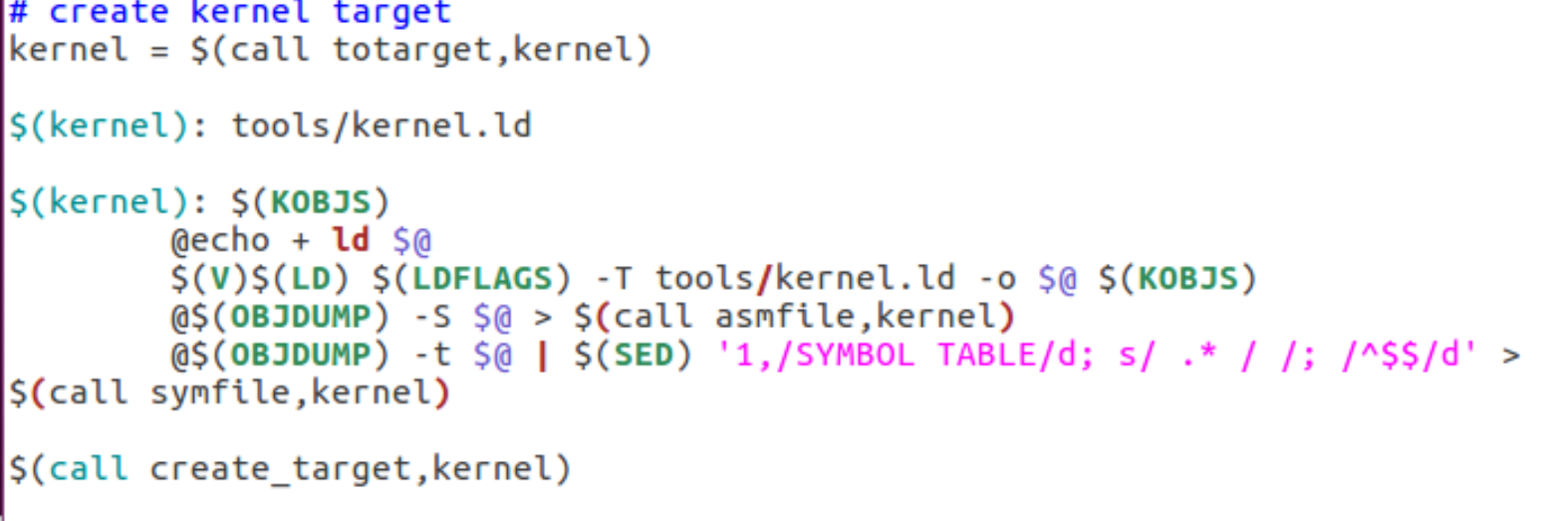
列出本实验各练习中对应的OS原理知识点，并说明本实验的实现部分如何应对和体现了原理中的基本概念和关键知识点

1. 操作系统镜像文件ucore.img是如何一步一步生成的？

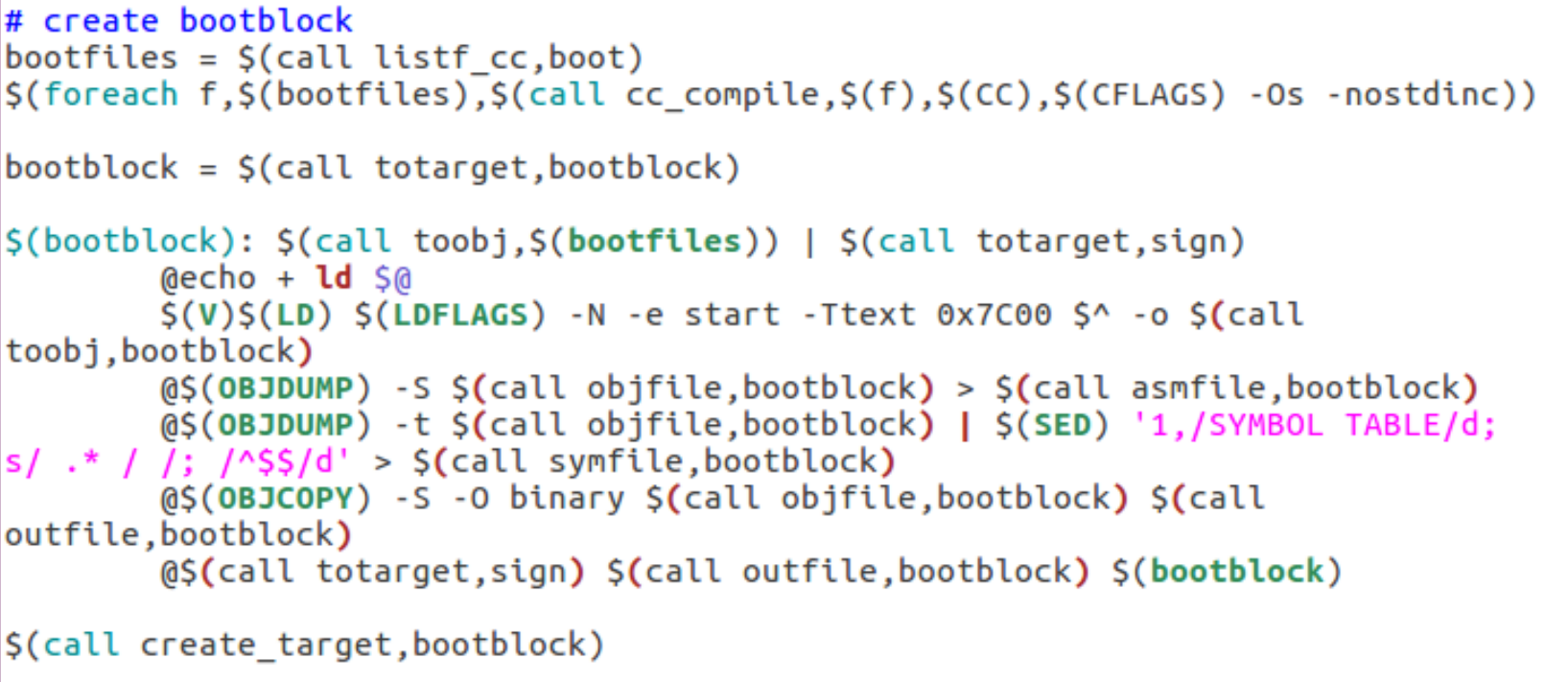


这是最后一步，申请一个大小为10000的空间来存放ucore.img；把bootblock和kernel放到ucore.img里，生成了ucore.img。

那如何生成kernel和bootblock呢？

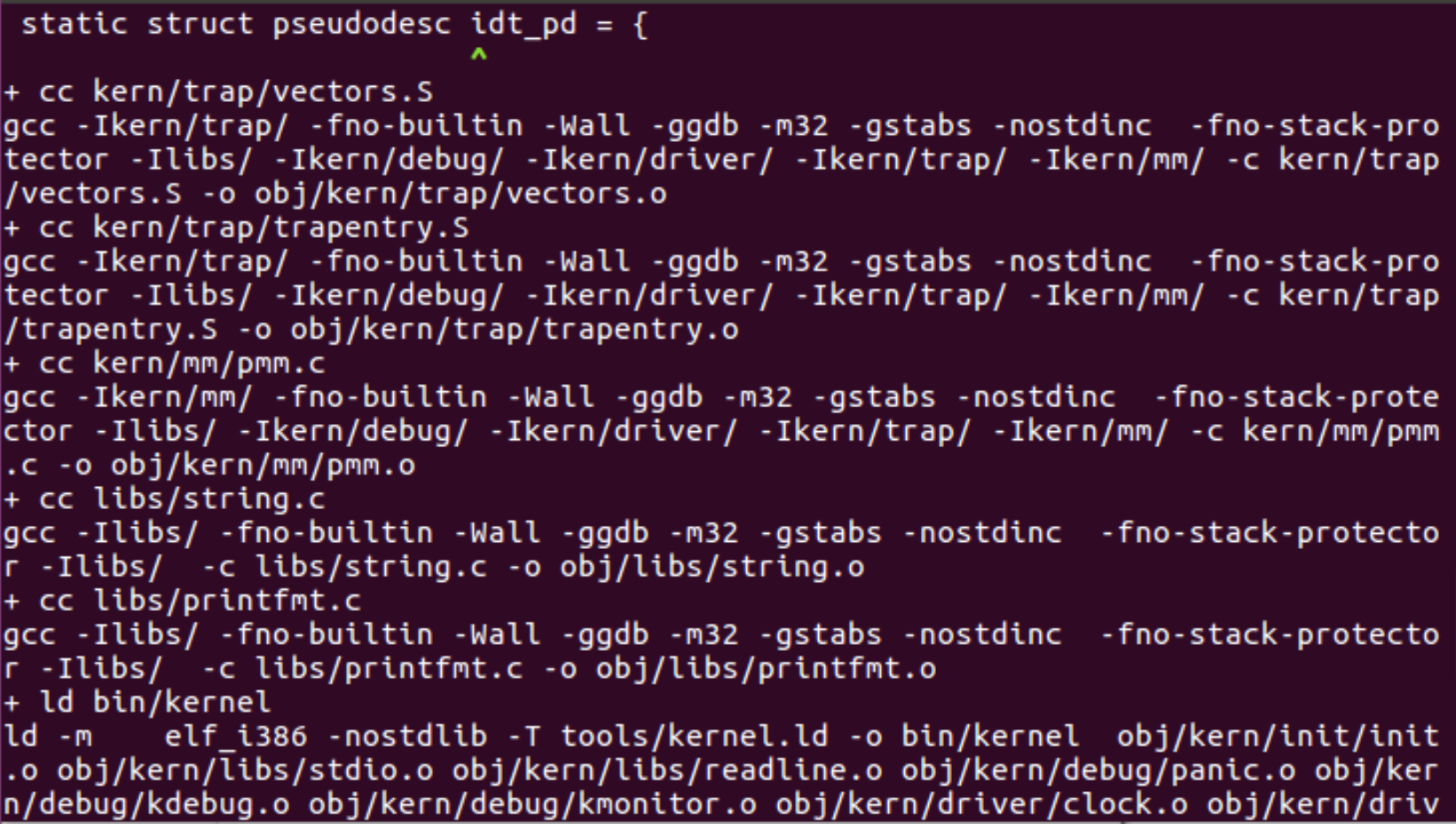


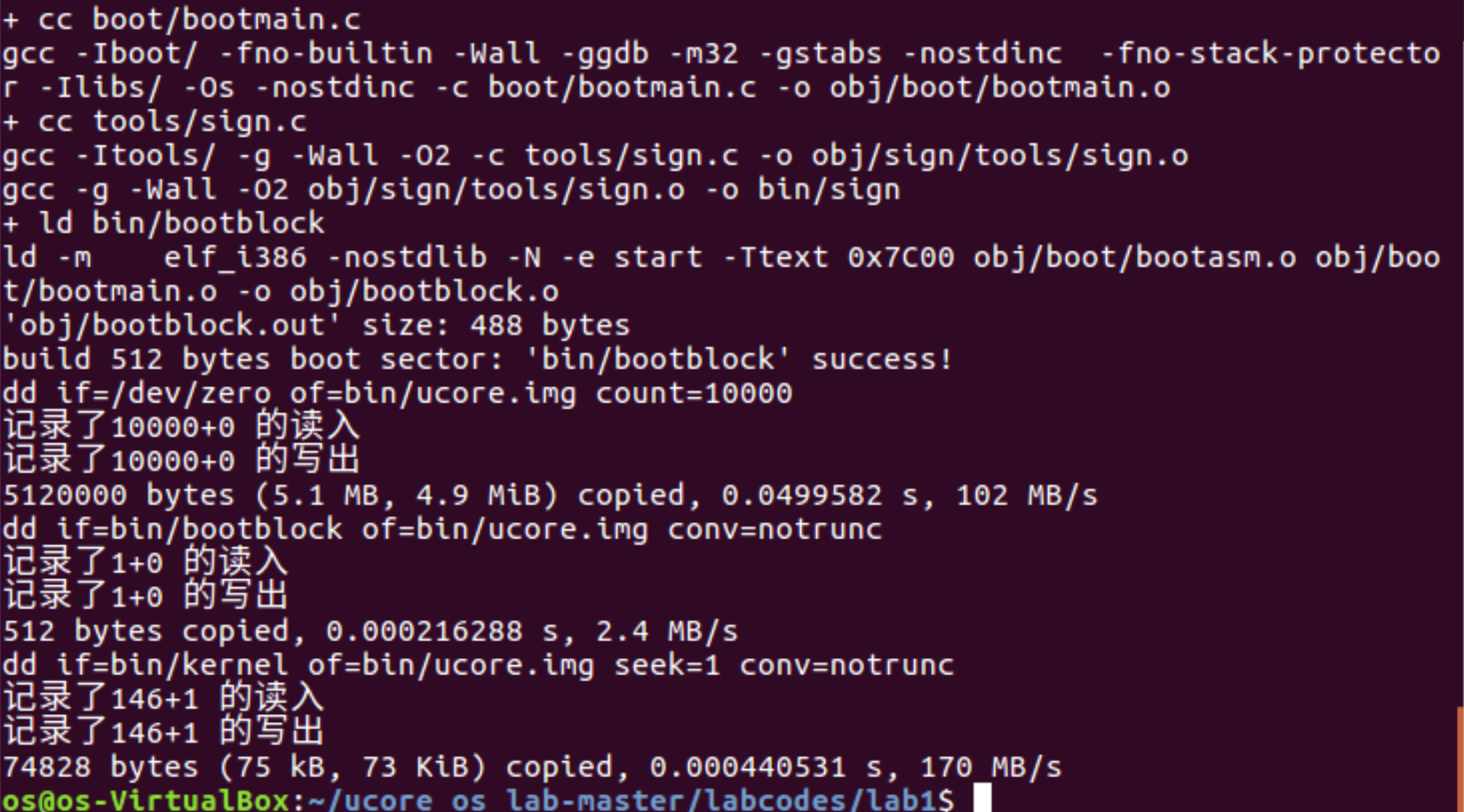
#生成kernel



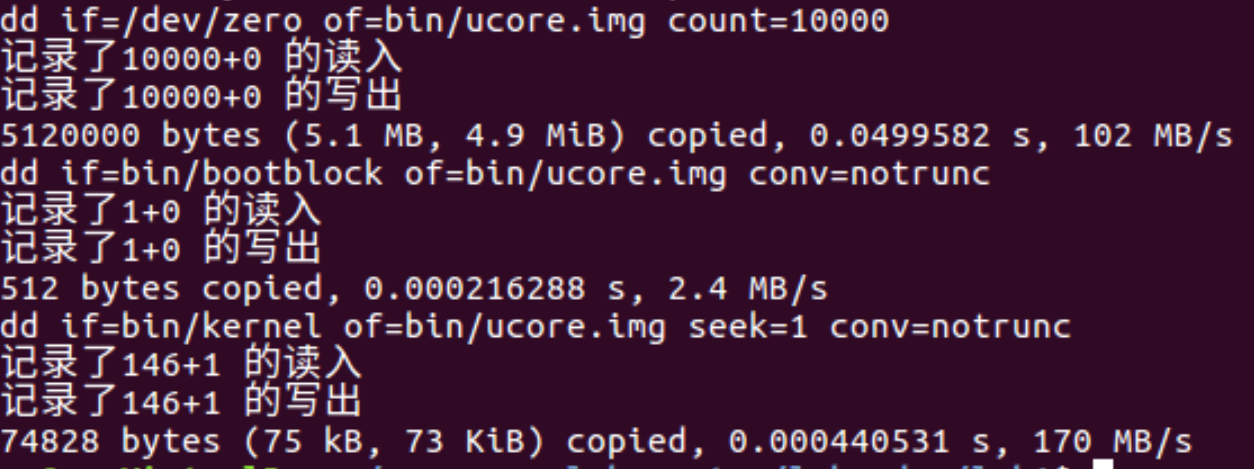
#生成bootblock

现在通过结果反推回去，通过“make v=”命令来看一下都执行了哪些命令



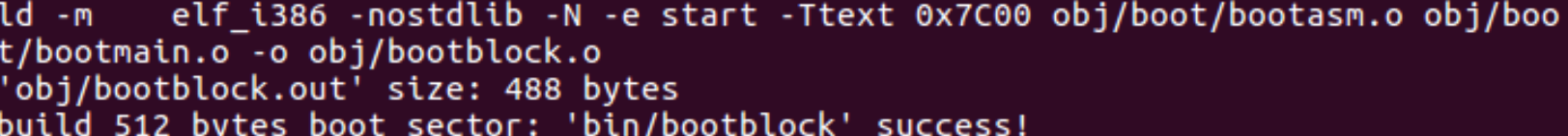


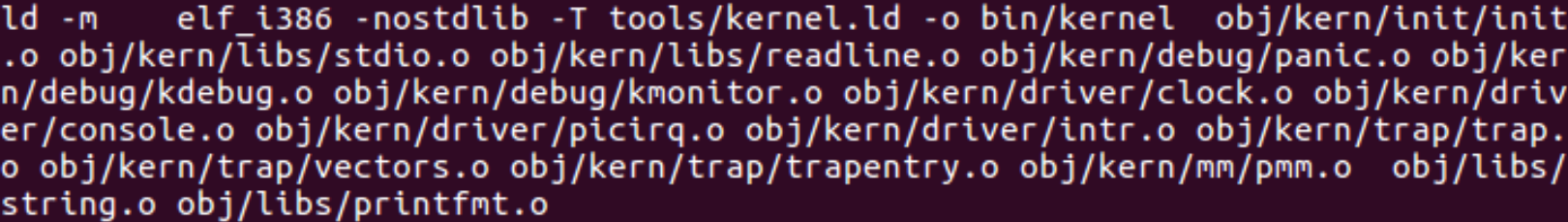
其中

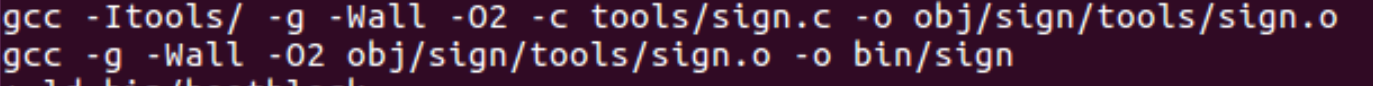


中的三条dd后面的语句就是之前提到的最后的三条语句，

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。



并且生成了两个代码，第一个是Bootloader,第二个叫Kernel Kernel，事实上是Ucore的组成部分。

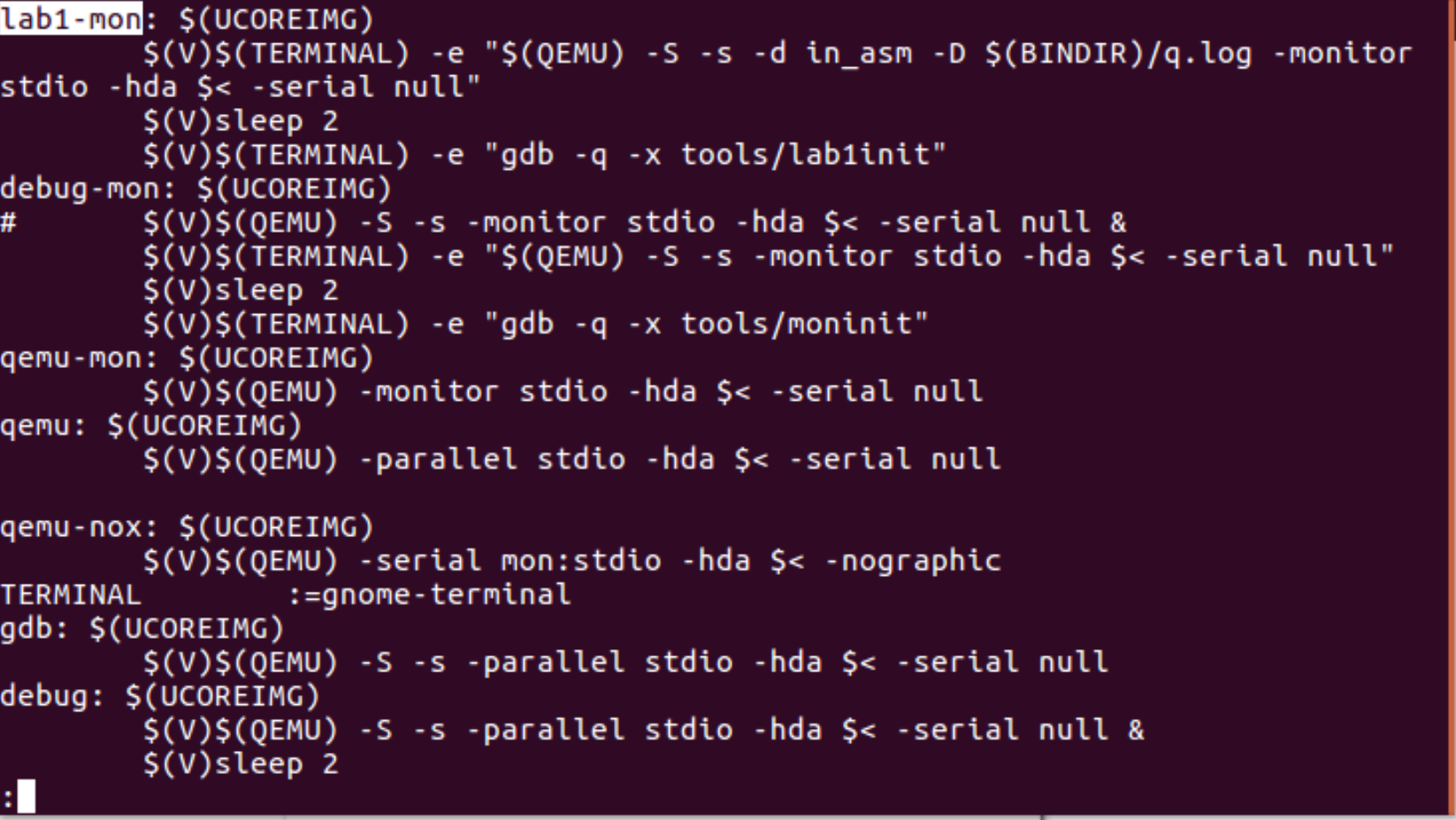


然后一些gcc命令把.c源文件生成.o的目标文件

2、使用qemu执行并调试lab1中的软件，写出练习过程

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. 在初始位置Ox7c00设置实地址断点，测试断点正常
3. 从Ox7c00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

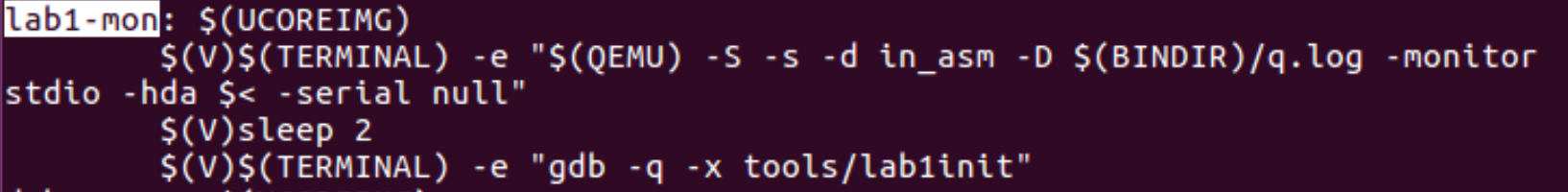
在终端 先输入make clean，再输入 less Makefile，接着输入/lab1-mon，就会出现



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

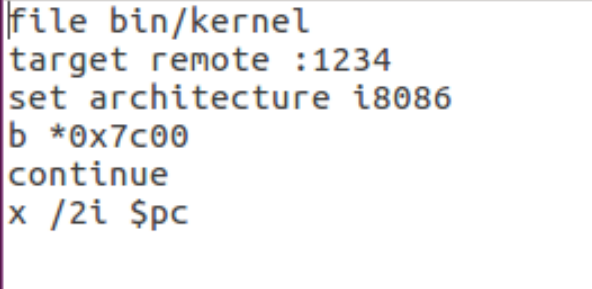
再就是跟GDB结合来调试正在执行的Bootloader，现在我们还在Bootloader阶段

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



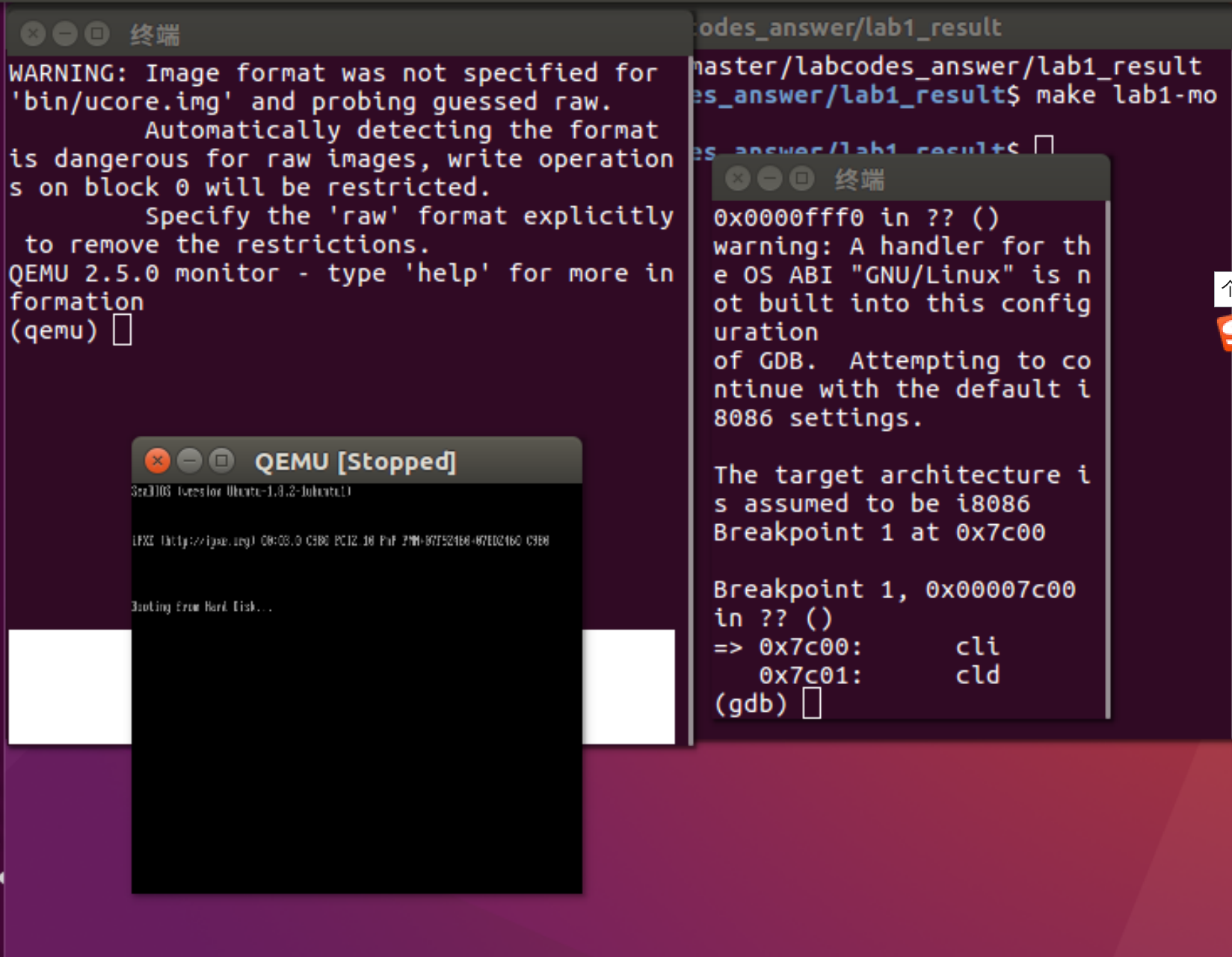
下图是GDB能够识别的一些命令，第一个 加载并kernel 加载符号信息 。第二条指令是与qemu进行连接，刚开始的时候BIOS是进入8086的16位实模式，一直到OX7c00在BIOS这个阶段启动，最后把Bootloader加载进去，把控制权交给bootloader，bootloader第一条指令就是在ox7c00处，break ox7c00，然后让这个程序继续运行

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

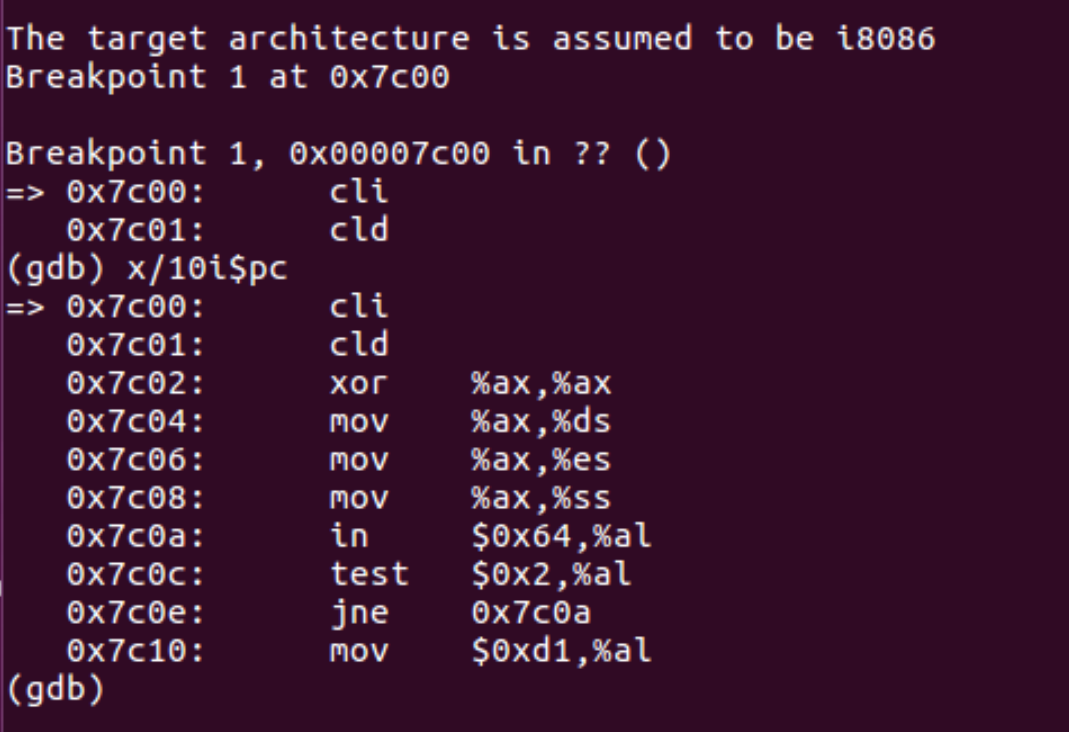


在lab1\_result下在终端打开，然后执行make lab1-mon ，显示两个窗口，一个是qemu

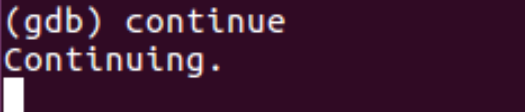
但是它断下来了，断在ox7c00处



输入 x /10i $pc 可以把十条指令都执行出来



输入continue继续执行



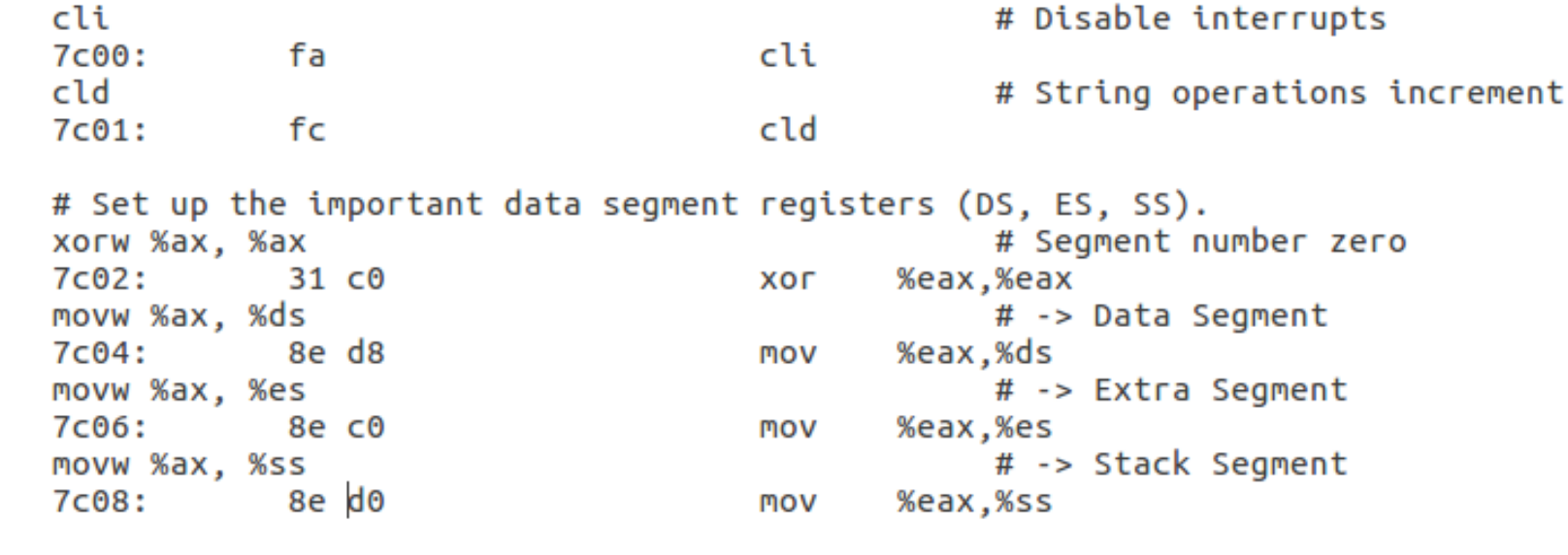
3、分析BootLoader进入保护模式的过程

* + 1. 为何开启A20，以及如何开启A20
    2. 如何初始化GDT表
    3. 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

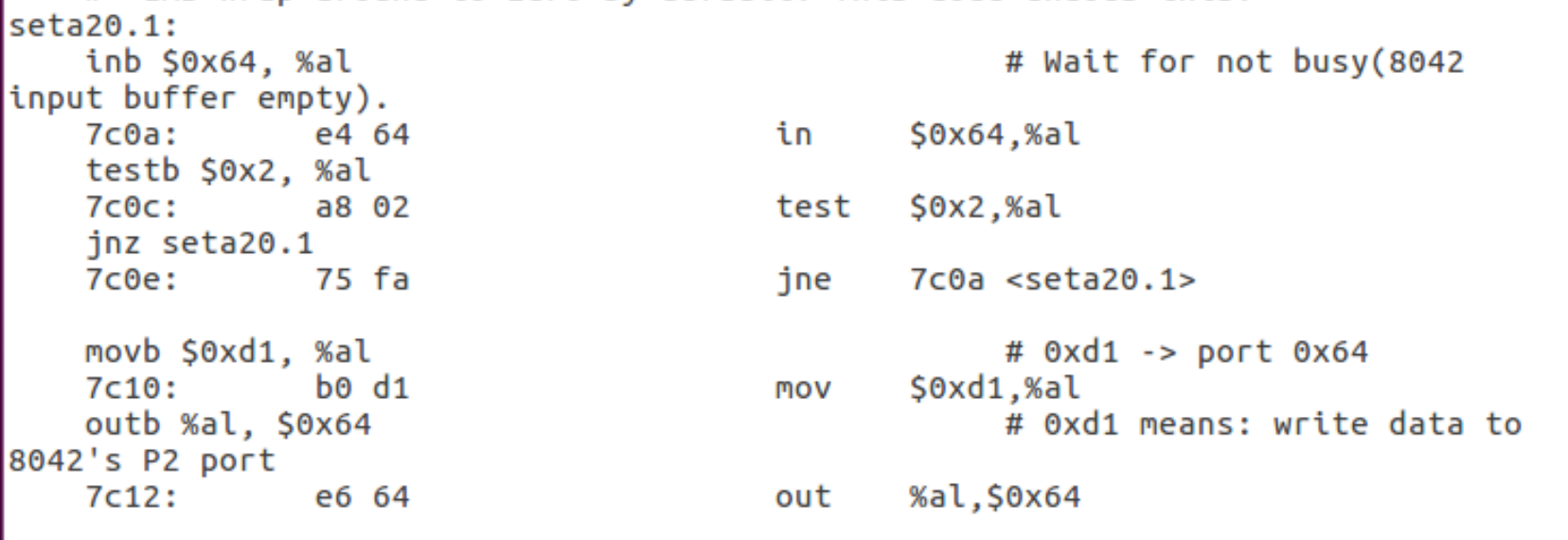
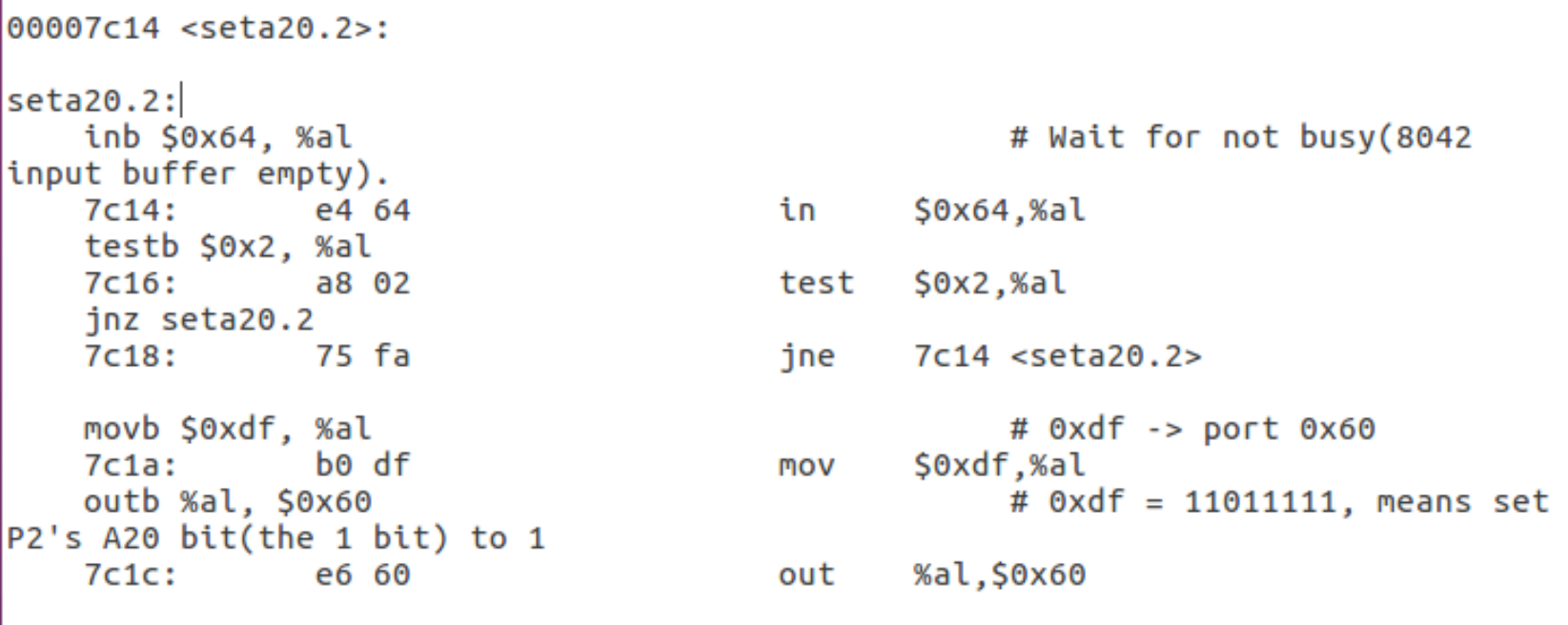


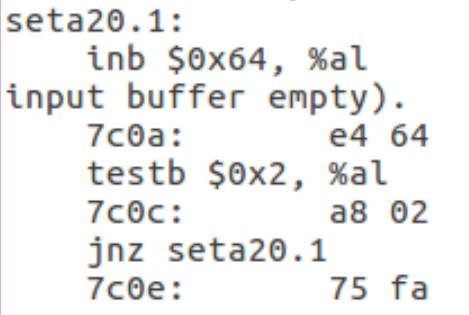
1. 关闭中断，将各个寄存器置为0



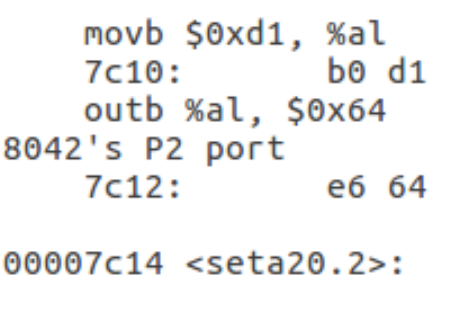
2.开启A20

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。

1. 
2. 



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



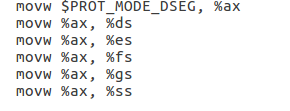
1. 将CR0的第0位置1



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

