# 操作系统实验一

**学号：171491219**

**姓名：崔凯慧**

**一、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）**

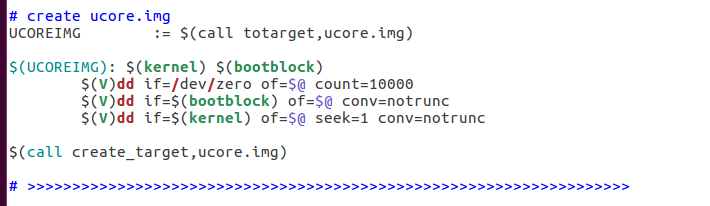
列出本实验各练习中对应的OS原理的知识点，并说明本实验中的实现部分如何对应和体现了原理中的基本概念和关键知识点。

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)

**实验结果：**

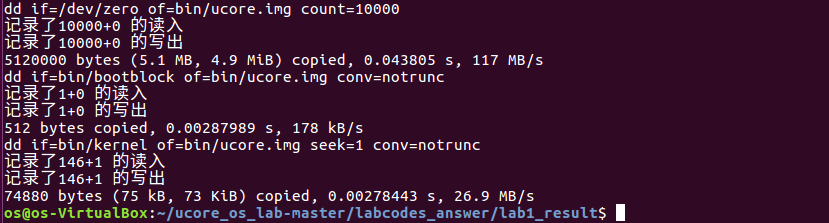
输入make “V=”然后分析：

第一组：创建一个ucore.img源码如下：

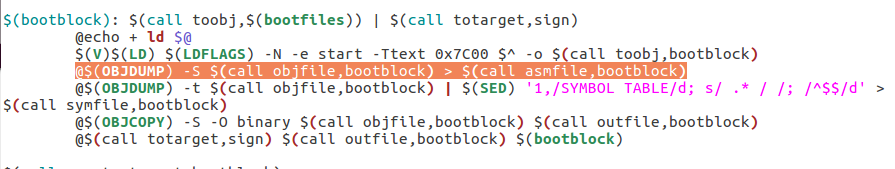


可看出它依赖的文件是kernel和bootblock

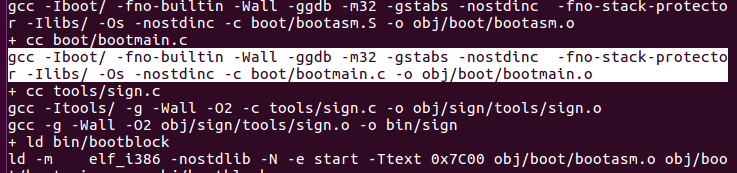
该源码编译后为：



第二组：bootblock的分析



bootblock中橘色代码编译的是.c文件，编译后的结果如下（白色框内）：



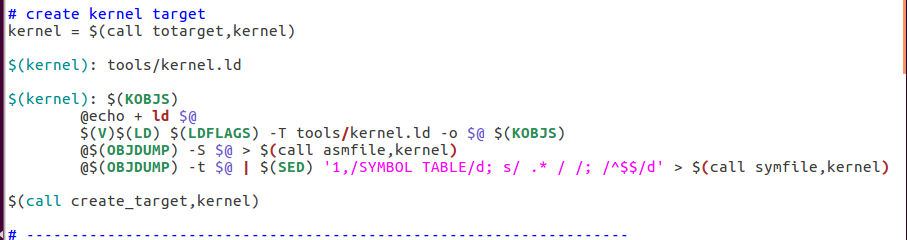
其中elf是文件格式。

-c 的含义：生成档案文件，创建打包文件

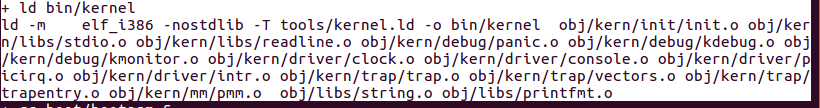
-o 的含义：用于指定输出(out)文件名。不用-o的话，一般会在当前文件夹下生成默认的a.out文件作为可执行程序。

第三组：kernel的分析

代码如下：



对应汇编语言为：



通过观察，发现文件是先由c语言文件通过gcc编译，之后两次链接生成两个o文件，最后生成ucore.img

**二、使用qemu执行并调试lab1中的软件**

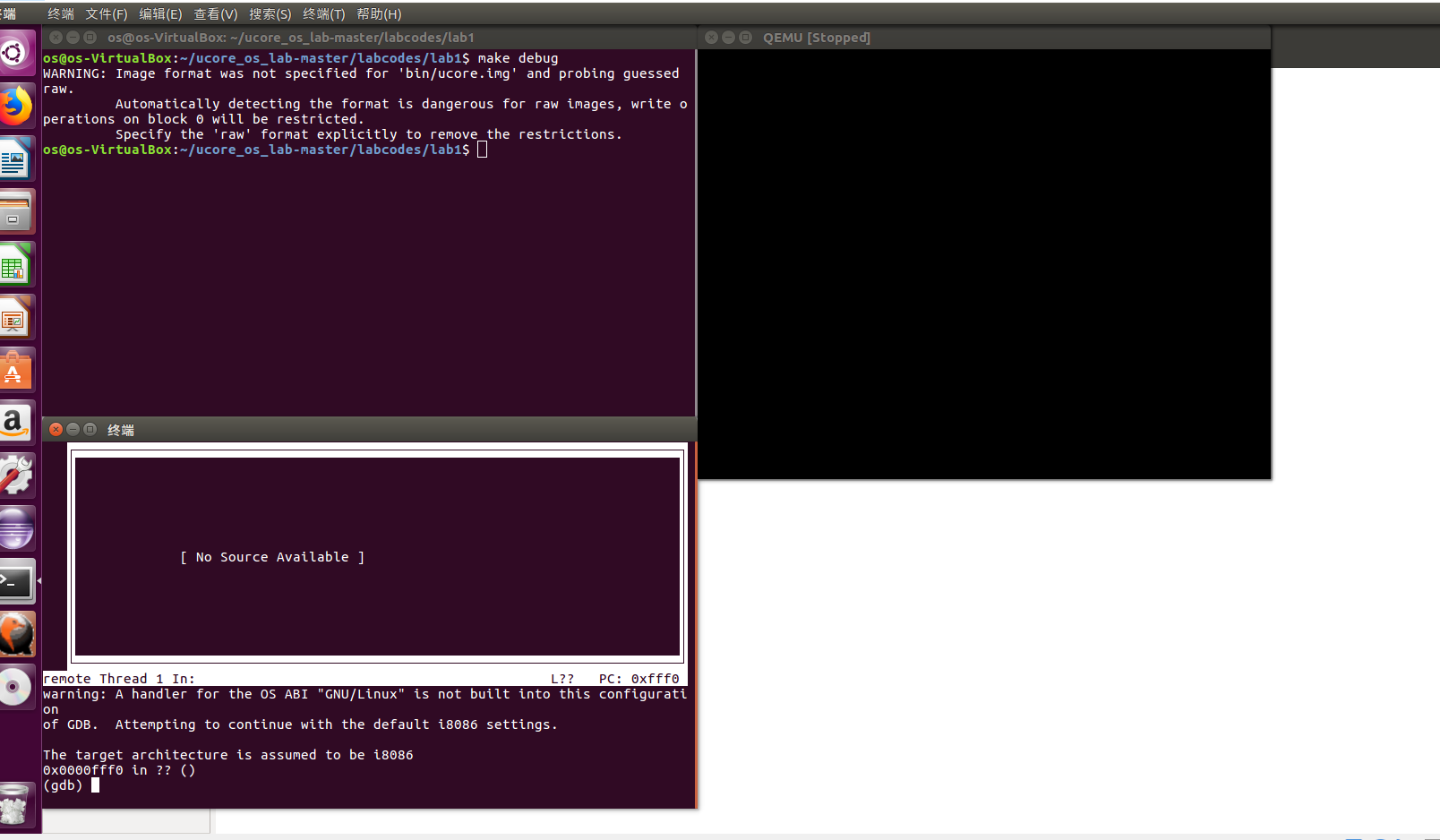
1. 从CPU加电后执行的第一条指令开始，单步跟踪BIOS的执行。

①修改lab1/tools/gdbinit，内容为：

Set architecture i8086

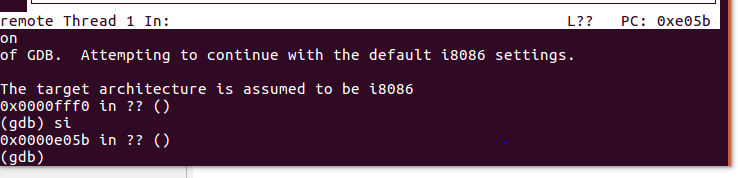
Target remote :1234

②在lab1目录下执行make debug

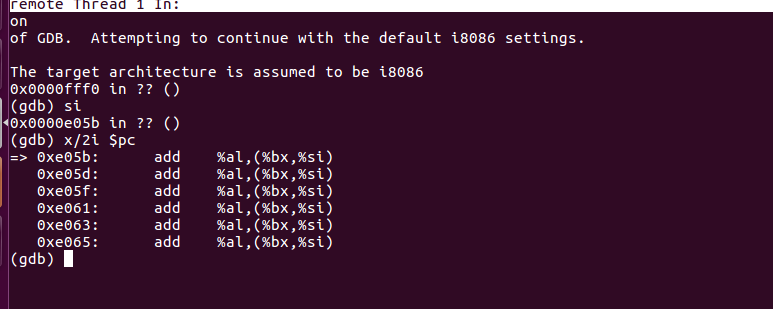


③在看到gdb的调试界面后，执行命令si

即可单步跟踪BIOS了

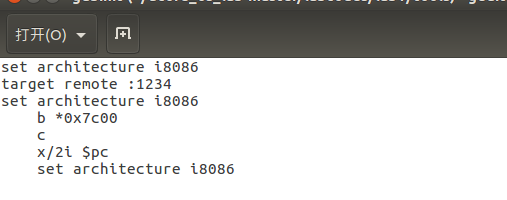


④在gdb界面下，执行x/2i $pc来查看BIOS的代码（此处我按了三次回车）

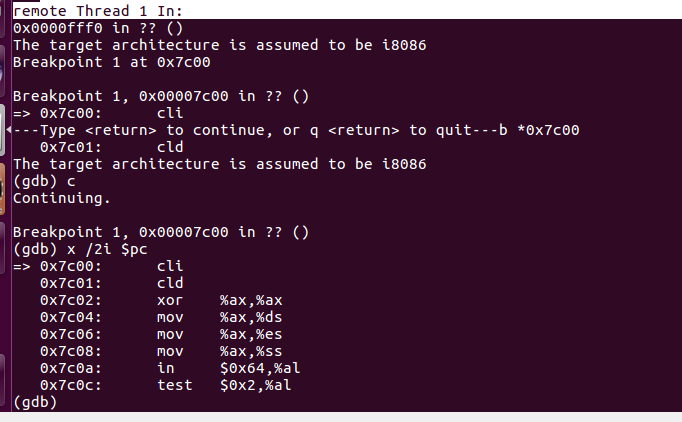


2.在初始化位置0x7c00设置实地址断点,测试断点正常。

①将gdbinit文件内容改成如下形式：

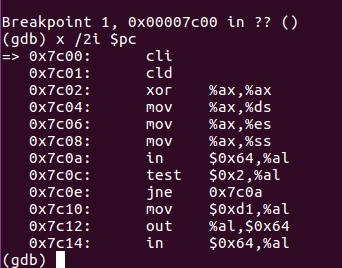


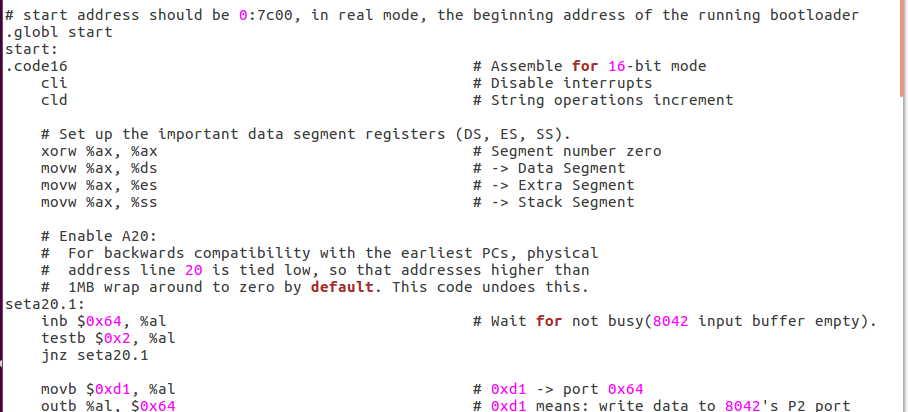
②运行make debug可得：



断点正常

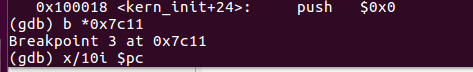
3.从0x7c00开始跟踪代码运行,将单步跟踪反汇编得到的代码与bootasm.S和 bootblock.asm进行比较。

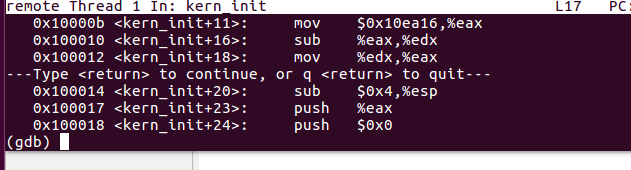




比较结果：相同

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。

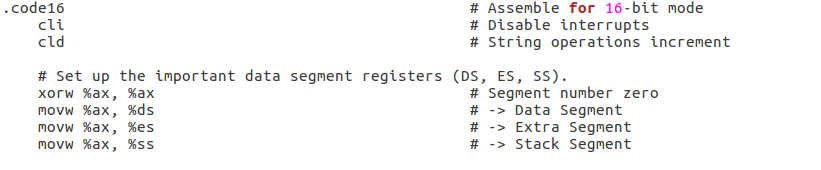




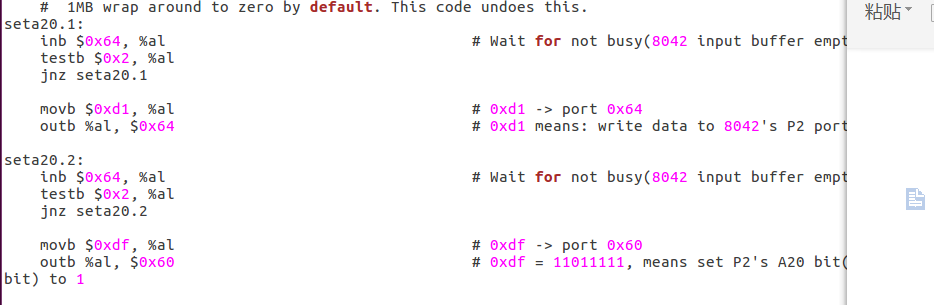
**三、分析bootloader进入保护模式的过程**



1. 清理环境：包括将flag置零和将段寄存器置零



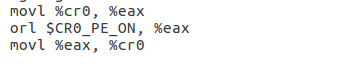
1. 开启A20：通过将键盘控制器上的A20线置于高电位，全部32条地址线可用，可用访问4G的内存空间。



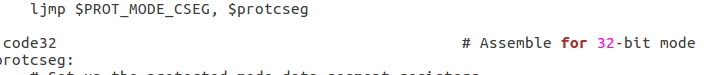
1. 初始化GDT表



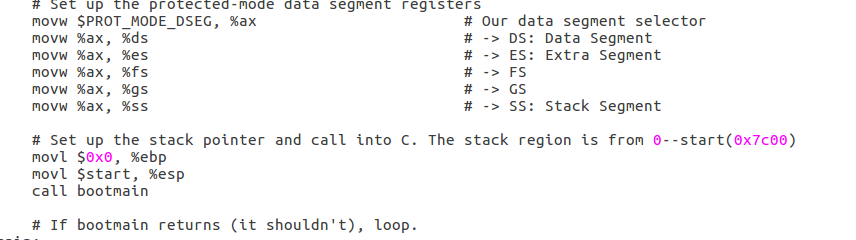
1. 进入保护模式：通过将cr0寄存器PE位置1便开启了保护模式



1. 通过长跳转更新cs的基地址



1. 设置段寄存器，并建立堆栈



1. 转到保护模式完成，进入boot主方法

