**实验一：系统软件启动过程**

1. 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：

1. 计算机原理

CPU的编址与寻址: 基于分段机制的内存管理

CPU的中断机制

外设：串口/并口/CGA，时钟，硬盘

1. Bootloader软件

编译运行bootloader的过程

调试bootloader的方法

调试ucore OS的方法

函数调用关系：在汇编级了解函数调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

1. 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。

1. 练习1：

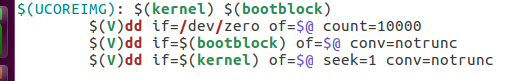
理解通过make生成执行文件的过程。（要求在报告中写 出对下述问题的回答）

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：

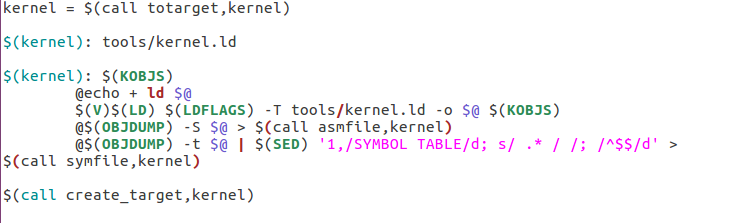
1. 操作系统镜像文件ucore.img是如何一步一步生成的？(需要比较详细地解释Makefile中每 一条相关命令和命令参数的含义，以及说明命令导致的结果)

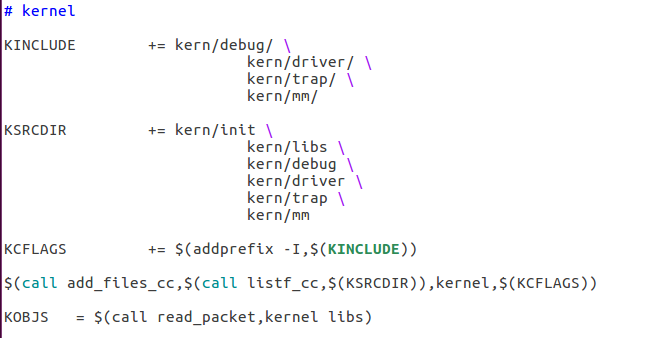
答：生成ucore.img的代码是：



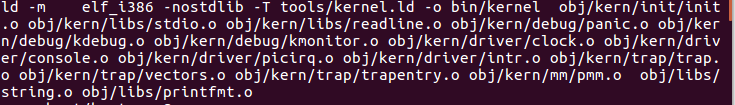
**要生成ucore.img首先要拥有kernel和bootblock两个可执行文件**

**Kernel相关代码:**



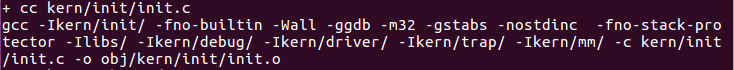


生成kernel需要以下文件：



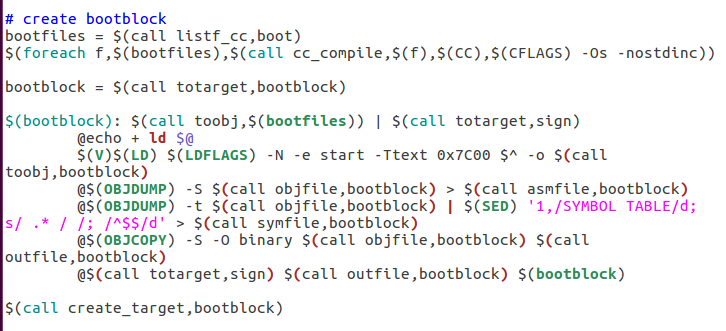
.o文件需要.c和.s通过gcc编译而成

生成Init.o需要的命令：



其他.o文件与init.o类似

**生成Bootblock相关代码：**



生成bootblock需要以下文件：



生成bootasm.o命令：



生成boomain.o命令：



生成sign命令：



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. 修改 lab1/tools/gdbinit



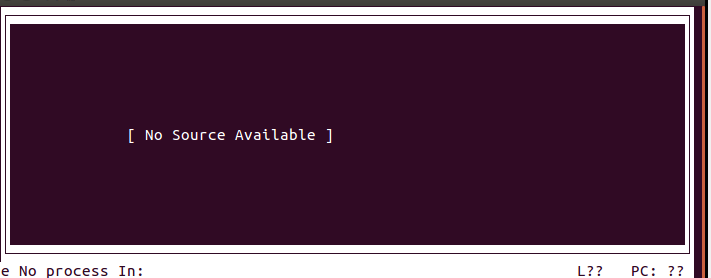
1. 在 lab1目录下，执行

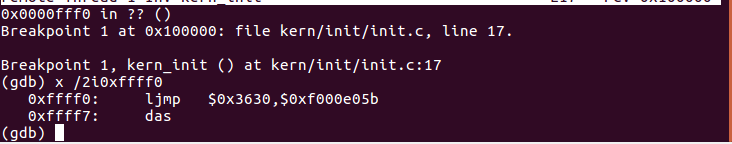


1. 使用

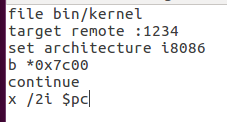


来查看BOIS的代码，结果如下:



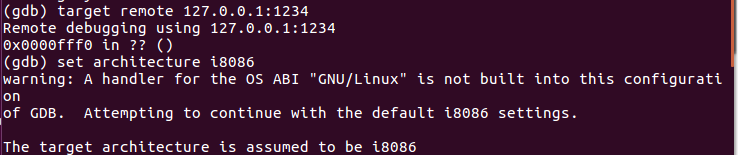


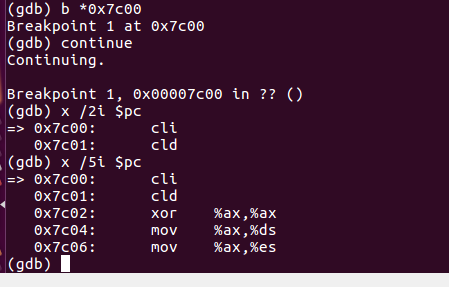
1. 在初始化位置0x7c00设置实地址断点,测试断点正常。





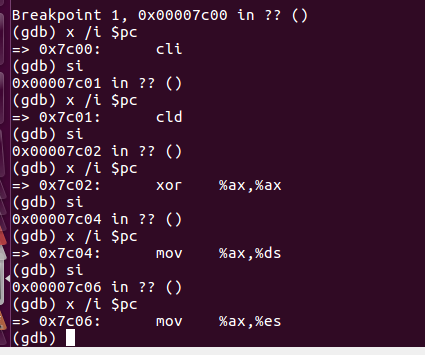




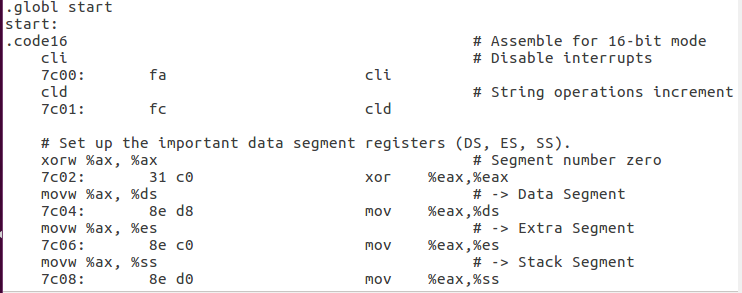


1. 从0x7c00开始跟踪代码运行,将单步跟踪反汇编得到的代码与bootasm.S和 bootblock.asm进行比较。

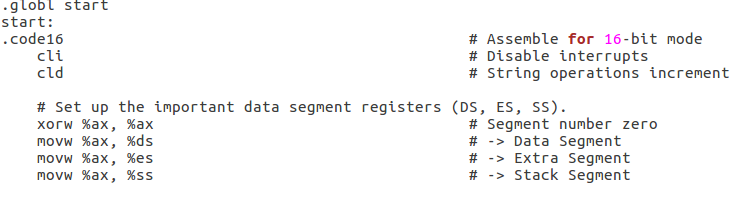
在0x7c00处break，然后使用si和x/ I $pc指令一行一行的跟踪，得到反汇编代码为:



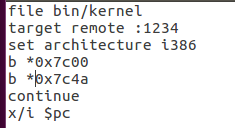
Bootblock.asm代码：



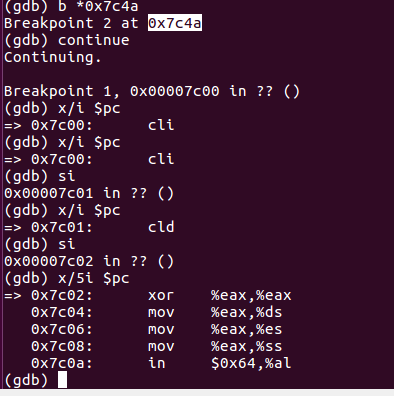
Bootasm.s代码:



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



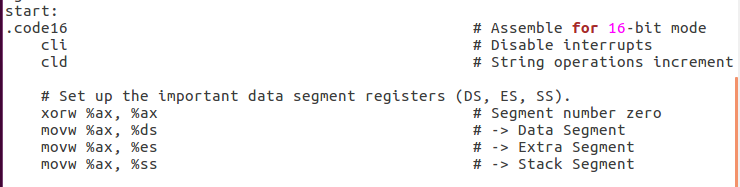
断点设置正常：



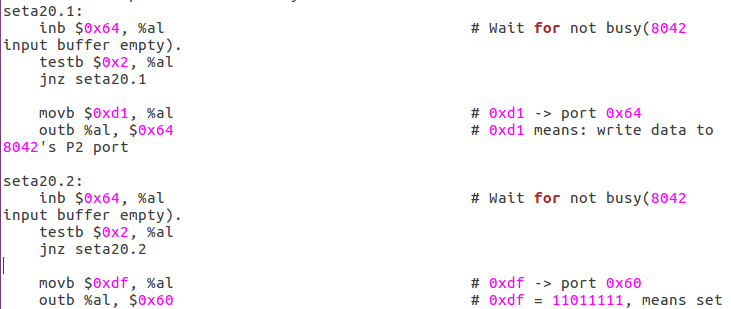
练习3：分析bootloader进入保护模式的过程。（要求在报告中 写出分析）

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。

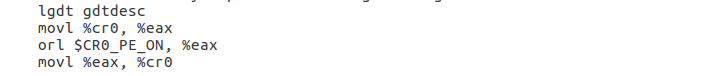
1. 清理环境，对实模式下的段寄存器进行初始化



1. 开启A20，将A20置于高位，全部32条地址线可用，可以访问4G的内存



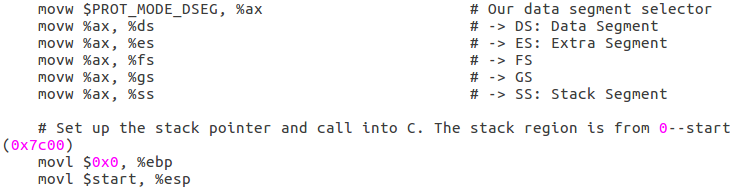
1. 初始化GDT,将实模式切换保护模式



1. 将处理器转至32位，跳转



1. 设置保护模式下的段寄存器，并建立堆栈



1. 进入 bootmain

