Lab1实验报告

班级：计算机科学与技术2班

姓名：许金曼

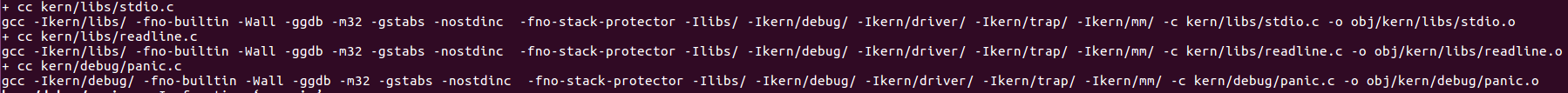
学号：171491220

1.操作系统镜像文件ucore.img是如何一步一步生成的？

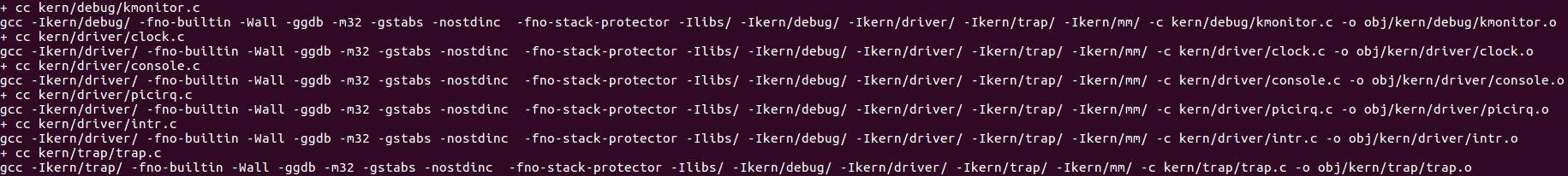
对lab1执行make “V=”操作，可以看到执行了哪些命令

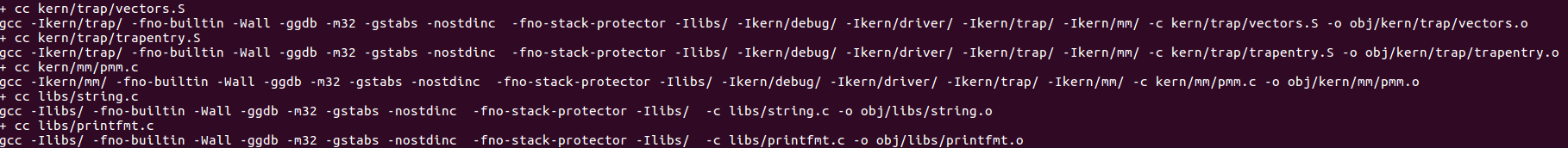
首先对kern文件下的.c和.S文件进行编译











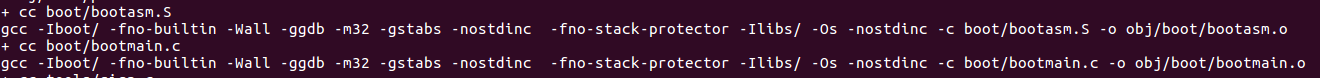
通过ld，将bin和kernel下所有程序连接起来



生成新的kernel程序



再对boot文件下的.c和.S文件进行编译



再对tools文件下的.c文件进行编译



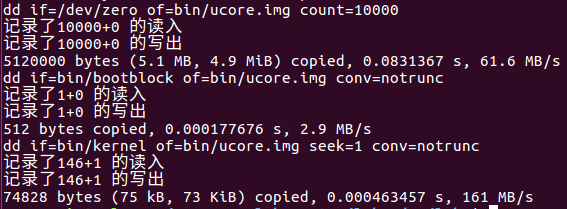
通过ld，将bin和bootblock连接起来



生成新的bootblock



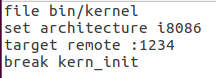
再通过dd命令，先将bootblock拷贝到了ucore.img，再将kernel拷贝到了ucore.img，所以可以得出bootblock是引导区，kernel是操作系统内核。



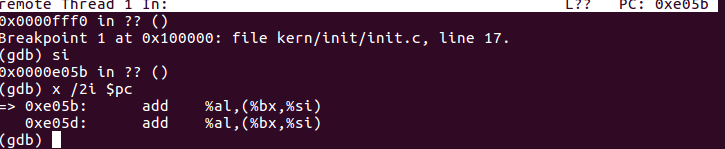
2.使用qemu执行并调试lab1中的软件

①从CPU加电后执行的第一条指令开始，单步跟踪BIOS的执行

首先在CPU加电后，CPU里的ROM存储器会将其里面保存的初始值传给各个寄存器，CS：IP=0×f000：fff0，此时系统处于实模式，地址总线只有20位，BIOS启动固件就在这个1M的空间里面，想要找到第一条指令的位置，然后break，就可以单步跟踪BIOS的执行。PC=16\*CS+IP，所以BIOS第一条指令的位置为0×ffff0。利用make debug来观察BIOS的单步执行

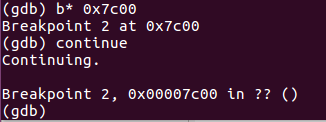


执行make debug后，在gdb调试界面用命令si单步跟踪BIOS，在用x /2i $pc看第一条指令



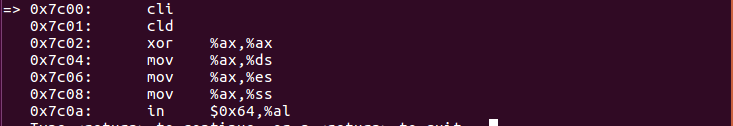
②在初始化位置0×7c00设置实地址断点，测试断点正常

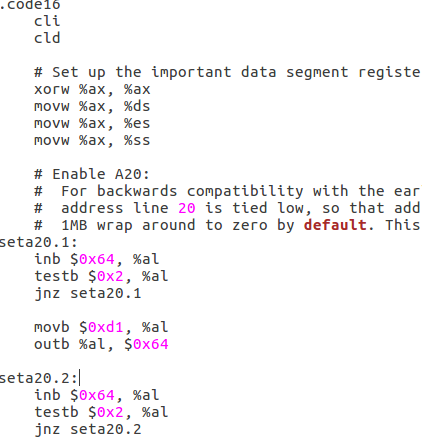
在0x7c00位置设置断点，continue



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

单步跟踪的得到的反汇编代码：





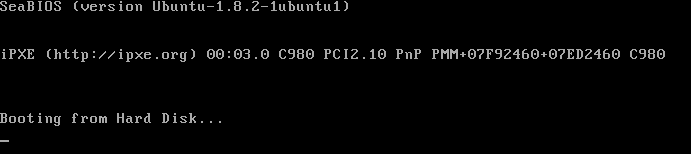
比较两者，可以看出代码几乎相同

④自己找一个bootloader或内核中的代码位置，设置断点并进行调试

在0x7c12位置设置断点



输入continue后，qemu继续正常工作



3.分析bootloader进入保护模式的过程

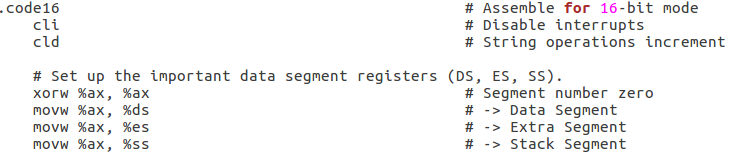
为何开启A20：

在8086/8088中，只有20根地址总线，所以可以访问的地址是2^20=1M，但由于8086/8088是16位地址模式，能够表示的地址范围是0-64K，所以为了在8086/8088下能够访问1M内存，Intel采取了分段的模式：16位段基地址:16位偏移。其绝对地址计算方法为：16位基地址左移4位+16位偏移=20位地址，但这种方式引起了新的问题，通过上述分段模式，能够表表示的最大内存为：FFFFh:FFFFh=FFFF0h+FFFFh=10FFEFh=1M+64K-16Bytes。但8086/8088只有20位地址线，如果访问100000h~10FFEFh之间的内存，则必须有第21根地址线，第21根地址线被称为A20Gate：如果A20 Gate被打开，则当程序员给出100000H-10FFEFH之间的地址的时候，系统将真正访问这块内存区域；如果A20Gate被禁止，则当程序员给出100000H-10FFEFH之间的地址的时候，系统仍然使用8086/8088的方式。

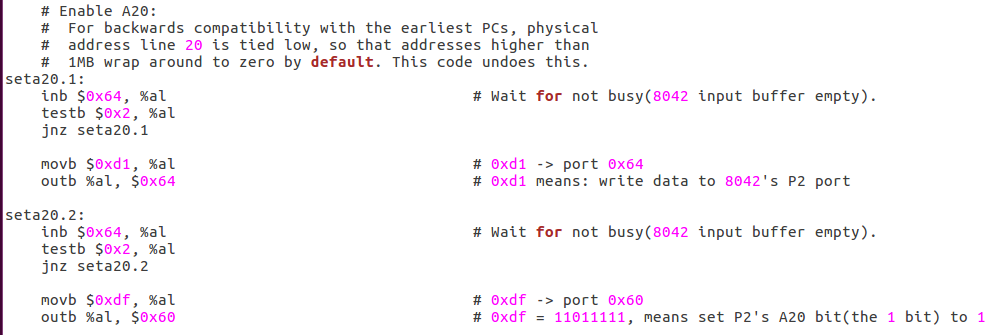
从该位置进入：



清理环境：



开启A20：



初始化GDT表：



进入保护模式：



首先将cr0寄存器里面的内容取出来，进行一个或操作，最后将得到的结果写入cr0中

通过一个长跳转指令进入保护模式：

