**PartB**

**Exercise 3.**

**Set a breakpoint at address 0x7c00, which is where the boot sector will be loaded. Continue execution until that break point. Trace through the code in boot/boot.S, using the source code and the disassembly file obj/boot/boot.asm to keep track of where you are. Also use the x/i command in GDB to disassemble sequences of instructions in the boot loader, and compare the original boot loader source code with both the GNU disassembly in obj/boot/boot.asm and the GDB**

**Trace into bootmain() in boot/main.c, and then into readsect(). Identify the exact assembly instructions that correspond to each of the statements in readsect(). Trace through the rest of readsect() and back out into bootmain(), and identify the begin and end of the for loop that reads the remaining sectors of the kernel from the disk. Find out what code will run when the loop is finished, set a breakpoint there, and continue to that breakpoint. Then step through the remainder of the boot loader.**

* boot.S

进入实模式后运行在16位模式下，关闭中断、复位串操作方向标志位；设置段寄存器值为0。

接下来激活A20地址线，具体步骤位将0x64端口的值设为0xd1，将0x60端口的值设为0xdf。

打开A20后，将全局描述符表载入，将cr0的第一位置1进入保护模式。

保护模式打开后，跳转至32位的代码段处(保护模式内核代码段)，切换处理器到32位模式。

跳转到内核代码段后，初始化各个数据段寄存器的值，包括数据段寄存器、额外段寄存器和栈段寄存器。之后设置栈指针并调用C函数bootmain。bootmain不应该返回，如果返回了，则进入无限循环中。bootmain中的C代码如下：

* bootmain

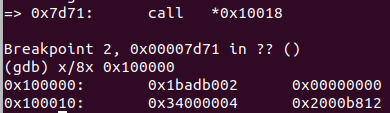
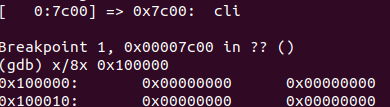
首先从磁盘中粗去第一页的信息，并判断是否正确读取了一个elf文件，如果是则继续读取程序的各个段，最后从elf的入口处开始执行，进入内核，内核不应该返回。

根据对上述代码的分析，可知练习三对应的答案分别是：

1. 程序执行ljmp后跳转到保护模式的代码段中，正式开始执行32位模式的代码，即初始化段寄存器的代码那一块。
2. bootloader执行的最后一条指令为C语言中进入elf文件第一段的调用语句，gdb调试可知为0x7d71: call \*10018。kernel执行的第一条语句为0x10000c : movw $0x1234, 0x472。
3. bootloader从elf文件的头部表中得到段的数量及每个段的具体信息，从而从磁盘中读取完整的kernel。

**Exercise 5.**

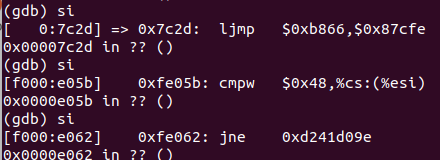
**Reset the machine (exit QEMU/GDB and start them again). Examine the 8 words of memory at 0x00100000 at the point the BIOS enters the boot loader, and then again at the point the boot loader enters the kernel. Why are they different? What is there at the second breakpoint? (You do not really need to use QEMU to answer this question. Just think.)**



如图中所示，该地址处字节并不相同，显然是应为载入了kernel造成的，使用objdump –h查看kernel可知0x100000为.text的载入位置。

**Exercise 6.**

**Trace through the first few instructions of the boot loader again and identify the first instruction that would "break" or otherwise do the wrong thing if you were to get the boot loader's link address wrong. Then change the link address in boot/Makefrag to something wrong, run make clean, recompile the lab with make, and trace into the boot loader again to see what happens. Don't forget to change the link address back and make clean afterwards!**



如上图所示，修改了链接地址后，在跳转到32位代码地址的地方出现了错误，代码回到了BIOS的开始，继续执行会进入一个无限循环。