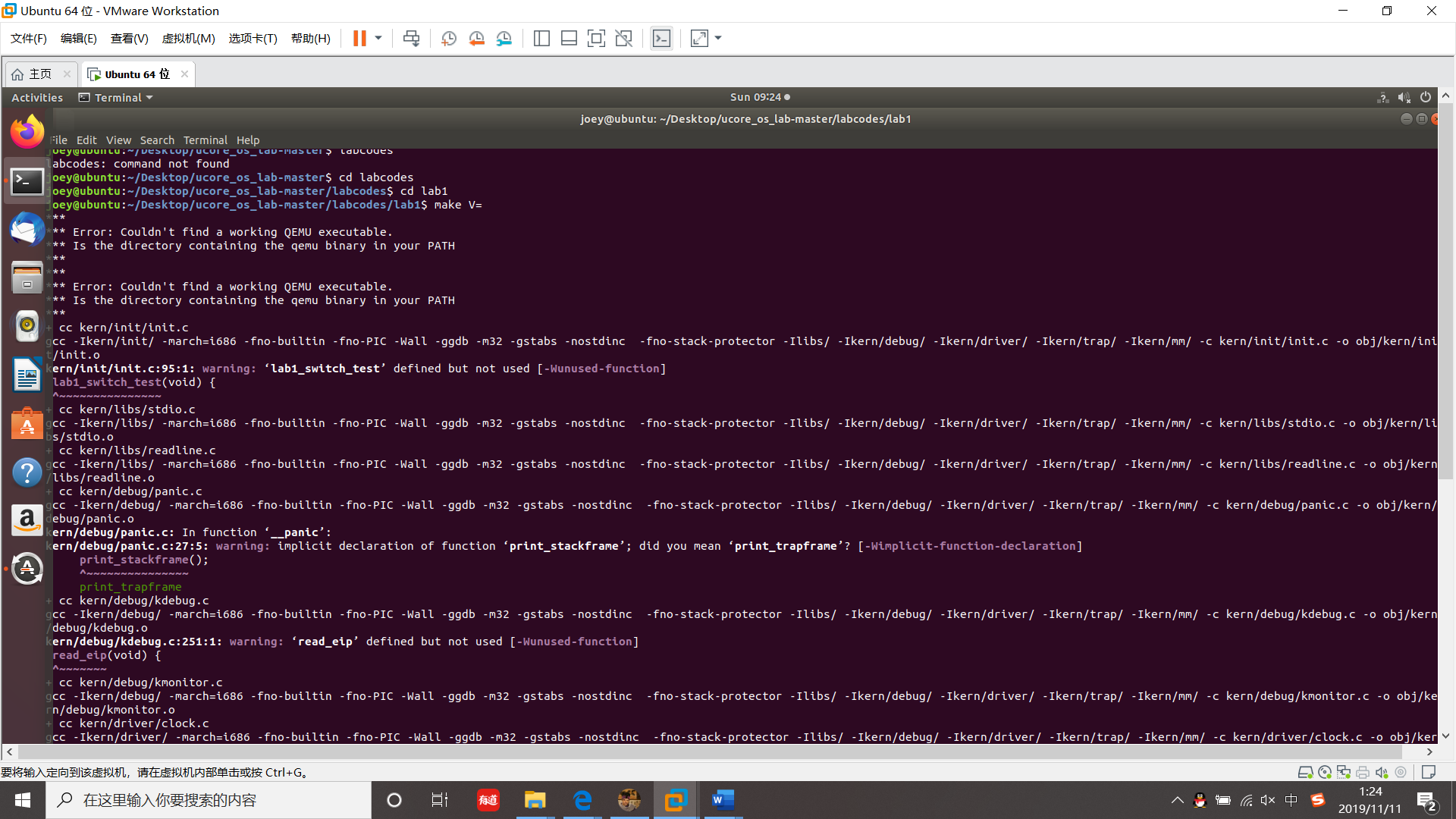
**练习1：理解通过make生成执行文件的过程。（要求在报告中写出对下述问题的回答）**

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)



* 行前面有+的，是编译指令，多数UNIX平台都通过CC调用它们的C编译程
* 下面都是具体执行的shell命令，gcc就是使用gcc编译器编译
* 参数进行分析：

-march 设置CPU类**型，可以选择 i8086, i186,** i286, i386, i486,等

-m32 编译生成32位程序

1. 一个被系统认为是符合规范的硬盘主引导扇区的特征是什么？

sign.c文件内容中：

int size = fread(buf, 1, st.st\_size, ifp);  
 if (size != st.st\_size) {  
 fprintf(stderr, "read '%s' error, size is %d.\n", argv[1], size);  
 return -1;  
 }  
  
  
buf[510] = 0x55;  
 buf[511] = 0xAA;  
 FILE \*ofp = fopen(argv[2], "wb+");  
 size = fwrite(buf, 1, 512, ofp);

所以一个磁盘主引导扇区只有512字节。

第510个字节是0x55，字节是0xAA。

**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）**

应该是：

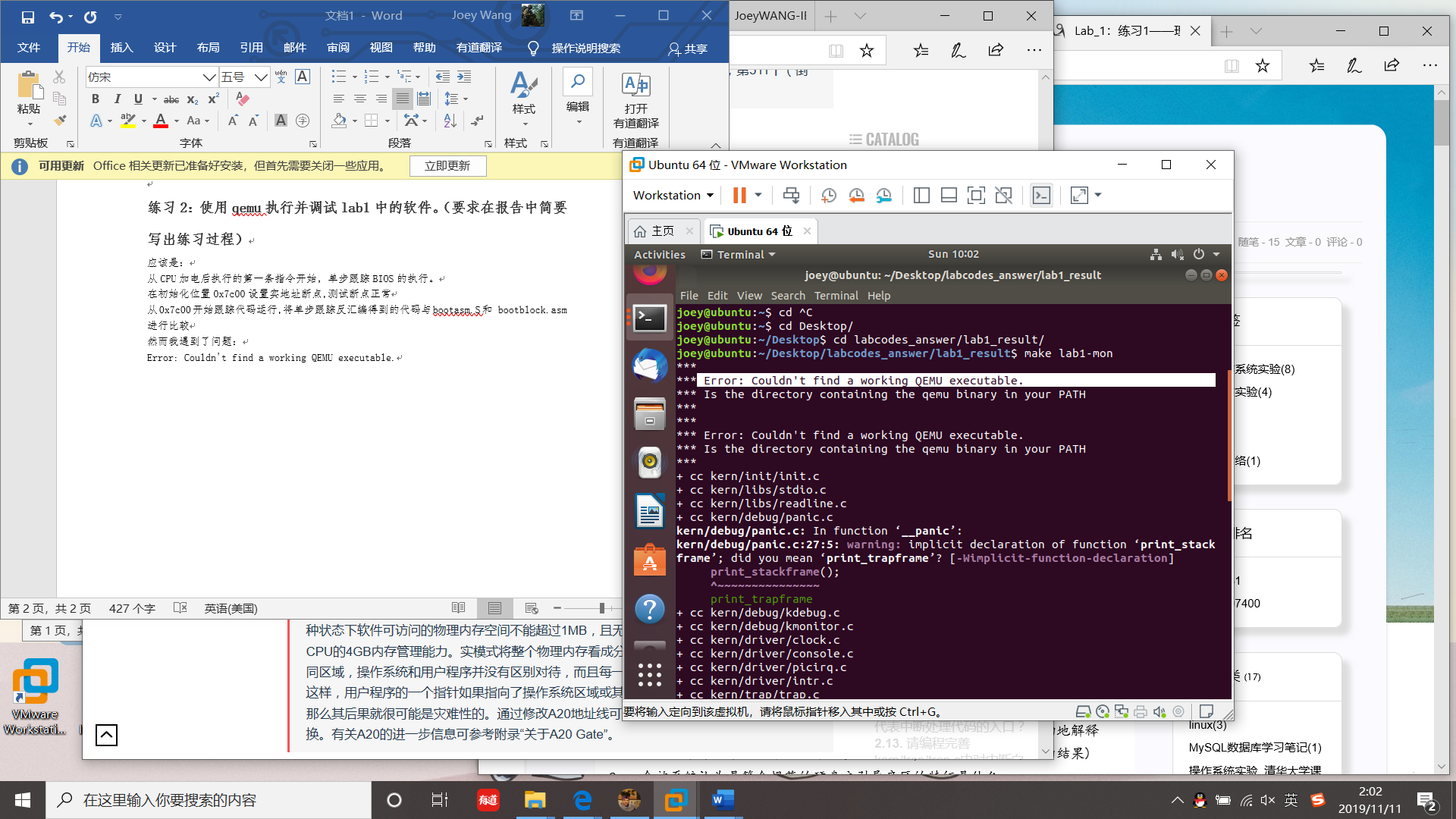
从CPU加电后执行的第一条指令开始，单步跟踪BIOS的执行。

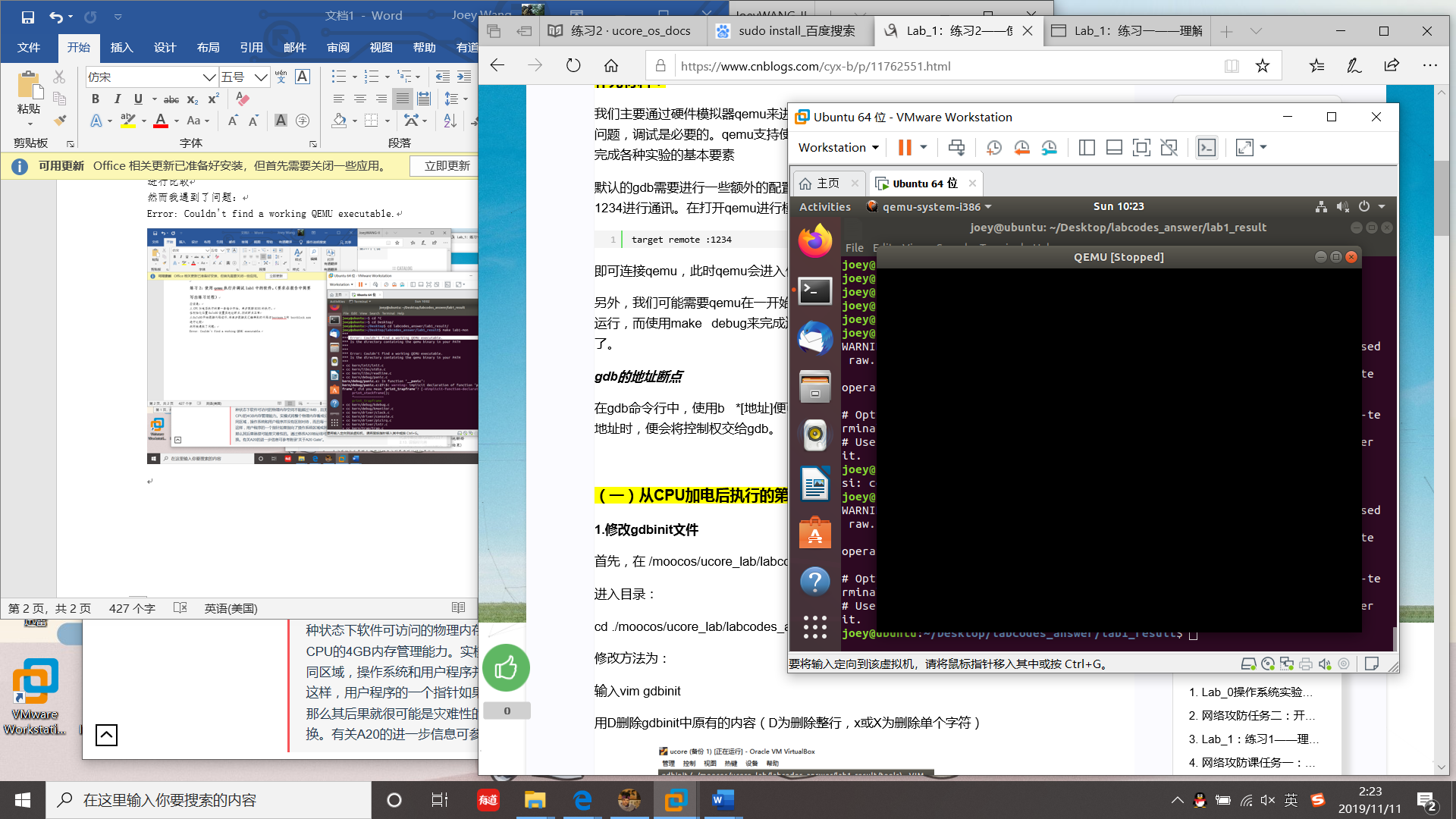
在初始化位置0x7c00设置实地址断点,测试断点正常

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

然而我遇到了问题：

Error: Couldn't find a working QEMU executable.





练习二失败，由于不明原因，qemu老出错。

**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）**

第一题：

1. 等待8042 Input buffer为空
2. 发送Write 8042 Output Port （P2） 命令到8042 Input buffer
3. 等待8042 Input buffer为空
4. 将8042 Output Port（P2） 对应字节的第2位置1，然后写入8042 Input buffer

第二题：

2.1什么是GDT表

它的GDT全称是Global Descriptor Table，中文名称叫“全局描述符表”，想要在“保护模式”下对内存进行寻址就先要有 GDT。GDT 表里的每一项叫做“段描述符”，用来记录每个内存分段的一些属性信息，每个“段描述符”占 8 字节。

在保护模式下，我们通过设置GDT将内存空间被分割为了一个又一个的段(这些段是可以重叠的)，这样我们就能实现不同的程序访问不同的内存空间。这和实模式下的寻址方式是不同的, 在实模式下我们只能使用address = segment << 4 | offset的方式进行寻址(虽然也是segment + offset的，但在实模式下我们并不会真正的进行分段)。在这种情况下，任何程序都能访问整个1MB的空间。而在保护模式下，通过分段的方式，程序并不能访问整个内存空间

2.2初始化GDT表

为了使分段存储管理机制正常运行，需要建立好段描述符和段描述符表，全局描述符表是一个保存多个段描述符的“数组”，其起始地址保存在全局描述符表寄存器GDTR中。GDTR长48位，其中高32位为基地址，低16位为段界限。这里只需要载入已经静态存储在引导区的GDT表和其描述符到GDTR寄存器。

第三题：

将cr0寄存器的PE位（cr0寄存器的最低位）设置为1，便使能和进入保护模式了。代码如下所示：

movl %cr0, %eax

orl $CR0\_PE\_ON, %eax

movl %eax, %cr0