1.计算机的启动过程

bochs模拟的是x86硬件系统，载入内存的分类：
1.程序被加载器加载到内存的某个区域
2.CPU的cs:ip寄存器指向这个程序的起始地址

2.BIOS

BIOS:Base Input & Output System

2.1 底层地址空间设置

从Intel8086开始了解信息，实模式下的内存布局：
20位地址总线对应于1MB大小的内存空间。
0-0x9FFFF 640KB对应了DRAM，也就是我们的内存条，这里面有可用区、MBR程序、BIOS区、中断向量表等内容。

0xF0000-0xFFFFF 64KB，对应于计算机的ROM，这里面存储的就是BIOS的代码。
BIOS代码的主要功能是检测、初始化硬件，BIOS直接调用硬件的初始化接口。
与此同时，BIOS还建立了中断向量表，这样就可以通过INT num 中断函数，来直接对硬件进行基本的IO操作。
由于是在实模式下，所以不需要过于复杂，这就是BIOS带一个基本的原因。

这两大段中间还有一些空间，对应于显示器等其他外存的空间。

在计算机中，并不是只有主板上的内存条才能通过地址总线访问，还有一些别的外设需要地址总线访问。比如说显存、硬盘控制器、ROM等，都需要一部分地址空间。
所以在CPU眼中，DRAM的地址空间只占了一部分。

2.2 BIOS苏醒过程

BIOS被映射到内存空间的底部：0xF0000-0xFFFFF，只要访问这个地址，就访问了BIOS。这个访问过程是写死在硬件里面的。

BIOS的入口地址为0xFFFF0
在开机时，CPU的cs：is寄存器被强制初始化为0xF000:0xFFF0 16位?

在实模式下，段基址*16，左移四位，之后，等效地址为0xFFFF0，即为BIOS函数的入口地址。
实模式下，这里的寄存器只有16位，所以有以上操作，来访问20位的内存空间。

0xFFFF0距离0xFFFFF只有16个字节，空间太小，无法执行诸多处理。
于是，这里存储的实际上是一个跳转指令：

```
jmp far f000:e05b
```

之后，BIOS运行起来，检测内存、显卡等信息，初始化硬件，之后再0x000-0x3FF处初始化数据结构，建立了中断向量表IVT，并且填写中断例程。

2.3 0x7c00

BIOS最后一项工作时校验启动盘中位于0盘0道1扇区的内容。
CHS采用1开始编号第一个扇区，LBA采用0开始编号第一个扇区。

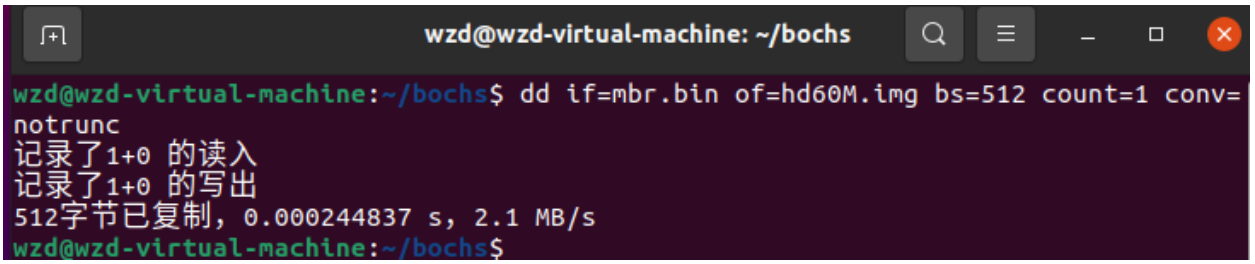
如果BIOS发现此扇区的末尾两个字节为0x55和0xaa，也就是 0101 0101 1010 1010 序列，就认为在这个扇区中存在可执行的程序。

跳转到0x7c00的方法：
jmp 0:0x7c00
于是，段基址寄存器被替换为0

0x7c00是一个魔数，来源于上古版本的遗留。
一个扇区大小为512kb，在BIOS启动时，会触发0x19h中断，寻找计算机中所有可用的磁盘，把第一个扇区加载到0x7c00位置。
MBR的内存规范为32KB，选择32KB内存空间的最后1KB来加载MBR（自己的程序+栈空间），这个是DRAM内存空间的底部空间，包含IVT中断向量表、MBR程序等。

2.4 让MBR跑起来

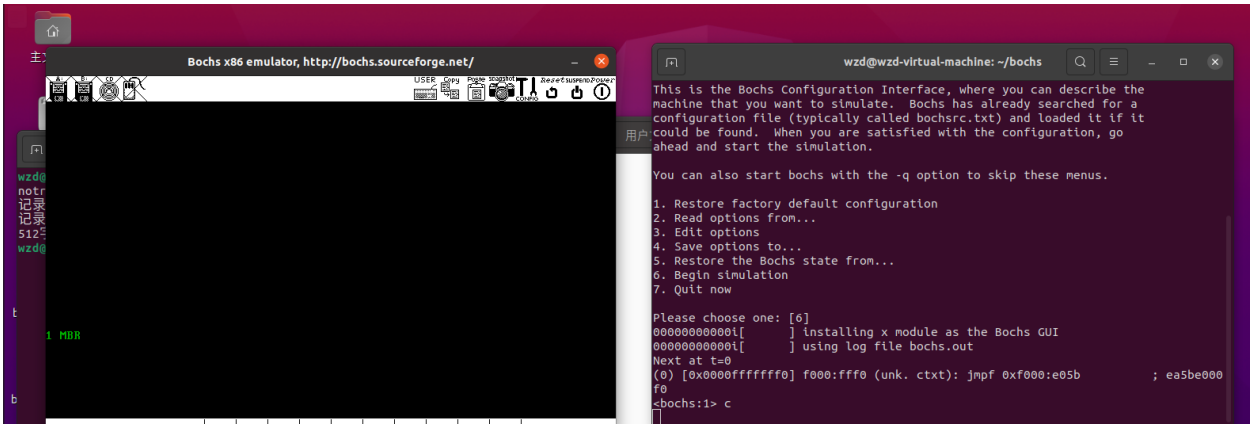
编写MBR程序的汇编代码，最后两个字节是0xaa、0x55。
使用汇编器编译成二进制文件，之后把这个二进制文件写入磁盘的第一个扇区。
sudo nasm -o mbr.bin mbr.S
dd if=mbr.bin of=hd60M.img bs=512 count=1 conv=notrunc



bin/bochs -f bochsrc.disk

这条命令是虚拟机开始模拟的指令：

最终成功运行起来了MBR汇编代码的内容，这是完全独立于操作系统的。



不得不说，这样一点点看下来，还是非常高兴滴！