

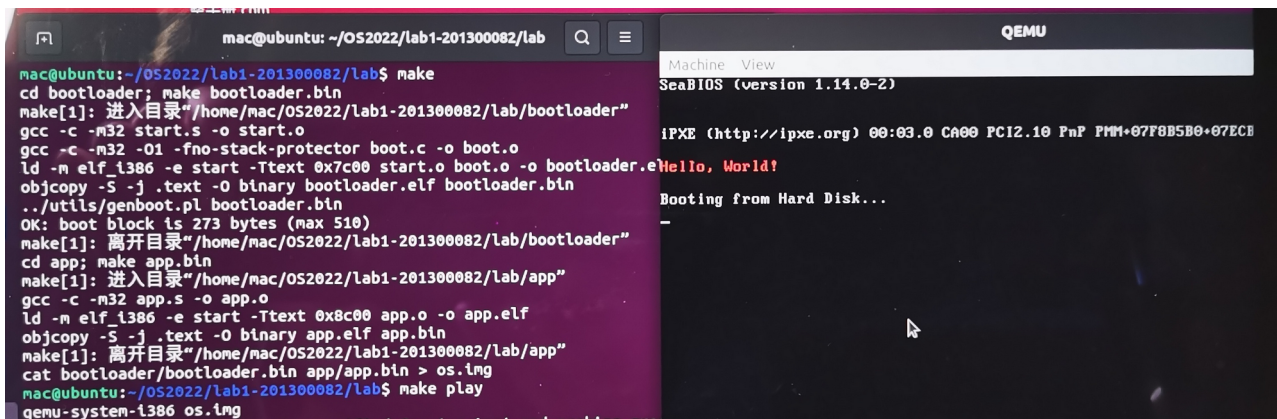
# lab1 实验报告

吴松栗 201300082 [1103755779@qq.com](mailto:1103755779@qq.com)

## 一. 实验进度

我已完成所有实验部分，即三个实验。

## 二. 实验结果



```
mac@ubuntu: ~/OS2022/lab1-201300082/lab$ make
cd bootloader; make bootloader.bin
make[1]: 进入目录"/home/mac/OS2022/lab1-201300082/lab/bootloader"
gcc -c -m32 start.s -o start.o
gcc -c -m32 -O1 -fno-stack-protector boot.c -o boot.o
ld -m elf_i386 -e start -Ttext 0x7c00 start.o boot.o -o bootloader.elf
objcopy -S -j .text -O binary bootloader.elf bootloader.bin
../utils/genboot.pl bootloader.bin
OK: boot block is 273 bytes (max 510)
make[1]: 离开目录"/home/mac/OS2022/lab1-201300082/lab/bootloader"
cd app; make app.bin
make[1]: 进入目录"/home/mac/OS2022/lab1-201300082/lab/app"
gcc -c -m32 app.s -o app.o
ld -m elf_i386 -e start -Ttext 0x8c00 app.o -o app.elf
objcopy -S -j .text -O binary app.elf app.bin
make[1]: 离开目录"/home/mac/OS2022/lab1-201300082/lab/app"
cat bootloader/bootloader.bin app/app.bin > os.img
mac@ubuntu: ~/OS2022/lab1-201300082/lab$ make play
qemu-system-i386 os.img
```

The screenshot shows a terminal window with the command 'make' being executed. The output shows the compilation of 'start.s' and 'boot.c' into 'start.o' and 'boot.o', followed by linking them into 'bootloader.elf' and then 'bootloader.bin'. The 'bootloader.bin' is then used to create 'os.img'. The terminal also shows the execution of 'make play', which runs the 'os.img' file in a QEMU environment. The QEMU window shows the 'SeabIOS (version 1.14.0-2)' boot screen, displaying the IPXE (http://ipxe.org) boot loader and the 'Hello, World!' message. The boot loader is currently booting from the Hard Disk.

## 三. 实验修改的代码位置

### 1. start.s:

(1). 将实模式启动改为实模式启动后切换到保护模式

(2). 查询手册和资料，对 DS ES FS GS SS 进行初始化

因为 ds 在 gdt 第二项,所以:  $(2 \ll 3)$

因为 gs 在第三项,所以:  $(3 \ll 3)$

(3). 查询手册和资料，对栈顶指针 ESP 进行了初始化

## 2. boot.c:

在 bootmain 中改为通过 readSect 完成一号扇区的加载,并通过 elf 的调用进入到位于 app.s 的 Hello World 程序。

## 四. 自由汇报

1. 我刚开始看代码相当疑惑, code16 和 code32 是啥意思, 什么区别。

我查了一下相关资料, 发现区别在于编译的不同。

code16: 因为处于实模式下, 所以需要编译成实模式的代码。希望编译器将下面的代码要编译成 16 位代码。

code32 : 因为处于保护模式下, 所以需要编译成保护模式的代码。希望编译器将下面的代码要编译成 32 位代码。

## 2.为什么 gdt 第一个表项是空的?

我看了看知乎相关解答, 具体来说, gdt 第一个表项一个空 (全局描述符) 选择子, 选择子的描述符索引值为 0, TI 指示位为 0, RPL 可以为任意值, 用这种方式表明当前任务没有 LDT。这里的空选择子因为 TI 为 0, 所以它实际上指向了 GDT 的第 0 项描述符, 第 0 项的作用类似于 C 语言中 NULL 的用法, 它虽然是一个描述符, 但却只起到了标志的作用。规定 GDT 的第 0 项描述符为空描述符, 其 8 个字节全为 0, 就是这个原因。

## 五. 思考题

### 1. EX1:

#### 1.1 名词解释

CPU：一般指中央处理器；

内存：用于暂时存放 CPU 中的运算数据，以及与硬盘等外部存储器交换的数据

BIOS：Basic Input Output System，基本输入输出系统。

磁盘：利用磁记录技术存储数据的存储器；

主引导扇区：它是硬盘上的第一个扇区,由硬盘厂商就预留了。一个扇区的大小通常是 512 个字节。主引导扇区，不依赖于具体的操作系统,也就是说主引导扇区可以启动任何一种操作系统。里面存储的是硬盘的基本信息，包括硬盘分成几个区，每个区的大小，每个区从什么位置开始，从什么位置结束。

操作系统：是管理计算机硬件与软件资源的计算机程序。

#### 1.2 关系

- 1.21 内存是计算机与 CPU 进行沟通的桥梁。计算机中所有程序的运行都是在内存中进行的；CPU 总是优先从内存获取指令，只要计算机在运行中，CPU 就会把需要运算的数据调到内存中进行运算，当运算完成后 CPU 再将结果传送出来，内存的运行也决定了计算机的稳定运行。
- 1.22 处理器的第一条指令会将地址定位到 BIOS 存储器中，让初始化程序开始执行，CPU 会执行 BIOS 的一小部分代码。
- 1.23 内存里的数据来自于 BIOS 和磁盘。硬盘是用来长时间存贮数据的，是存储设备的一种，负责存储数据。而计算机硬件系统是由 CPU 和内存组成的，内存虽然也是存储设备，但是他的主要任务是整个硬件系统的缓存，原因是个硬件的传输速率不一样，如果没有内存这个缓存，各硬件间的速率不一样，就会导致因某个硬件过快或过慢导致整个硬件系统的崩溃。
- 1.24 指令通过控制 cpu 的输出和输入位置，可以让 cpu 寄存器向 I/O 端口寄存器发送数据，外围设备通过与 I/O 寄存器交换数据，从而达到控制外围设备的效果。而操作系统的作用是封装一套指令序列，cpu 通过执行这套指令序列，可以让硬件完成一系列动作。而应用程序如果需要对硬件进行操作，只需要将操作系统的这套指令序列复制到自己的程序中即可。
- 1.25. 主引导扇区包含了加载程序

## 1.26. 操作系统在装载前都存在于磁盘内,在加载程序执行完以后被装载到内存里

### 2. EX2:

中断向量表：顾名思义，就是中断向量的集合。而中断向量就是中断源的识别标志，可用来形成相应的中断服务程序的入口地址或存放中断服务程序的首地址称为中断向量。

### 3.EX3:

段的长度是偏移地址可以取的数值规定的，在 8086cpu 中，偏移地址使用一个 16 位的二进制数表示，其表示范围是（0000H:0FFFFH），总共有  $2^{16}$  (2 的 16 次方)=64K 个不同的取值，一个内存单元使用 1 个偏移地址，故一个段的大小是 64K

### 4 EX4

print 部分自然不用多说，这部分主要是将 buff 指向 0x55 和 0xaa，然后进行判断，是否 0x55 和 0xaa 部分可以 open,否则触发 die。

原因在于，正常结果的末尾是魔数 0x55 和 0xaa，通过检查末尾两个字节是否为 0x55 和 0xaa，即是否合法。如果成功找到了魔数，BIOS 将会跳到 0x7c00 的内存位置，执行启动代码。否则，如果没有检查到魔数，将会逐个尝试设备是否可行。

### 5 EX5

系统上电后，CPU 开始执行第一条程序，因为 CPU 也不知道哪个是操作系统内核，哪个是用户程序，它唯一会做的就是执行一条指令，并

输出结果，然后再从内存的紧接着的单元取下一条指令。而程序的跳转是由指令本身修改指令指针（PC）指向新的内存地址（或者由外部中断或异常引起 PC 的修改）实现的。CPU 上电后，PC 从地址 0 开始执行，而地址 0 处必然是一条可执行指令，在个人计算机中，CPU 上电是从主板的 BIOS 中开始执行的，BIOS 是挂接在 CPU 的地址总线上的。BIOS 程序会完成主板的监测和基本初始化，并构建起中断向量表，然后再从硬盘加载第一个扇区内容到内存。当 BIOS 从磁盘加载第一个引导扇区（512 字节），之后，会检查其中是否可执行代码，如果不是，那么就报错“不是启动盘”。之后，磁盘的第一个扇区中会有信息指出哪个分区是活动分区，加载活动分区的引导扇区，正式启动操作系统的加载过程。此时是运行于实模式的，因为此时的 CPU 处于初始化前的状态，寄存器中的值是初始化值，只有更改寄存器的值为某些特定值才能启动保护模式。操作系统的引导程序会初始化基本的硬件系统，如内存控制器、堆栈及其他一些 CPU 寄存器。然后把操作系统内核拷贝到内存中，内核在内存中的物理位置有些是固定的，有些是不需要固定的，对于哪些是必须固定在特定地址，这由操作系统设计决定。在后面就是启动保护模式了。

## 六 总结

本次实验主要了解了一些电脑启动的细节，也查询了很多资料和手册，收获很多，特别感谢一下助教哥哥们，我多次提一些简单的问题，助教们都耐心解答和回答。