第1课 操作系统是如何启动的

《跟着瓦利哥学写OS》

自我介绍

- 瓦利哥，浙江大学计算机系本科/硕士毕业
- 百度工作6年，从事数据仓库方向
- 与OS的缘分：
  - 大三选OS课老师做导师，结果老师不研究OS，选了网格计算方向（云计算前身）；
  - 曾打算做一个将集群内其他机器的内存作为本机的swap缓存区的研究；
  - 研究大半年后（2006年），认为网格计算真正走向现实要到20年后，遂专心做工程；
  - 工作后先在百度知道，后负责日志统计平台，逐步演变为数据仓库，所做的云计算发现就是网格计算的发展，人生就是一个轮回。

联系方式

- 自己动手写操作系统QQ群：82616767，目前群里有300人，非常活跃，加群需答题考核。

- 新浪微博：sangwf

- 邮箱：sangwf@gmail.com

- Github：https://github.com/sangwf/，正在写一个小OS，Walle OS。

准备课程

- 汇编语言
  - 汇编语言（第三版），王爽 / x86汇编语言，李忠
- C语言
  - The C Programming Language, K&R
- 操作系统
  - 操作系统概念 / 自己动手写操作系统 / Linux内核完全剖析
- 计算机组成
  - 计算机组成原理
- 编译原理
  - Compilers: Principles, Techniques, and Tools（龙书）

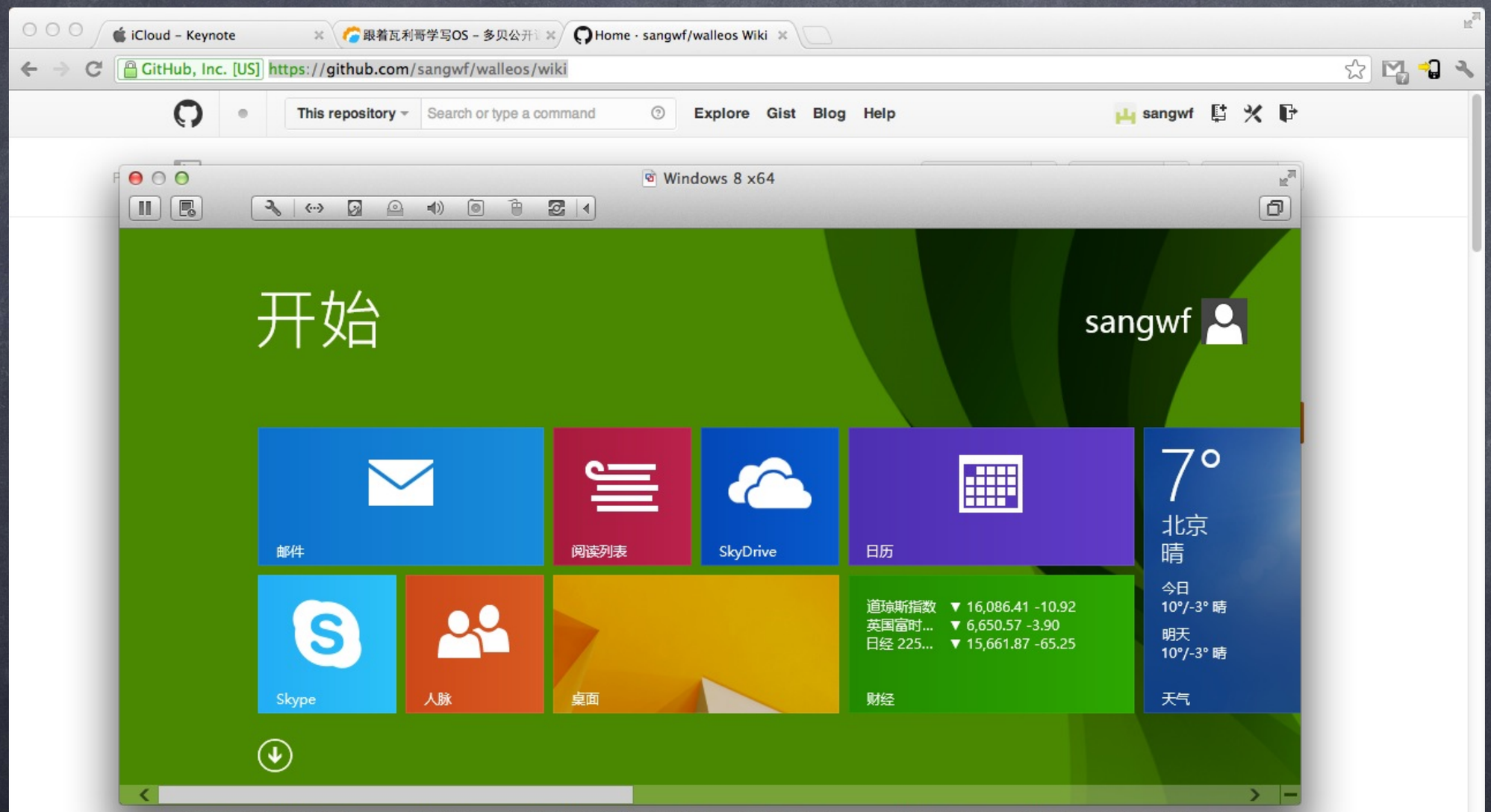
开发环境

- 操作系统：Linux/Mac OS
- 编辑器：Vi
- 汇编器：Nasm（http://www.nasm.us/）
- 编译器：Gcc
- 虚拟机：VMWare/Bochs ...

什么是虚拟机

- 虚拟机是一个软件，能够模拟计算机硬件环境
- 操作系统里运行操作系统，想想电影《盗梦空间》
- 好处：
  - 开发调试更高效
  - 避免系统被搞坏

Mac OS虚拟一个Win 8



[2013-9-27] 我如果想要mac os下编译的macho格式的程序能够顺利执行，必须要有和Mac os一样的虚拟空间，这样的话，我后面还是要实现虚拟内存机制，不然编译出的程序无法直接使用，总是有各种内存越界的问题，暂时先不急着实现。现在的主要任务是研究清楚unix设计里fork相关联的内容，以及内核剖析里的Linux 0.12的fork的具体实现。

操作系统的启动过程

- 1，按下电源
- 2，运行BIOS（Basic Input Output System）
- 3，加电自检POST（Power On Self Test）
  - 检查硬件是否正常
- 4，加载Boot Loader引导程序
- 5，引导操作系统

详解BIOS

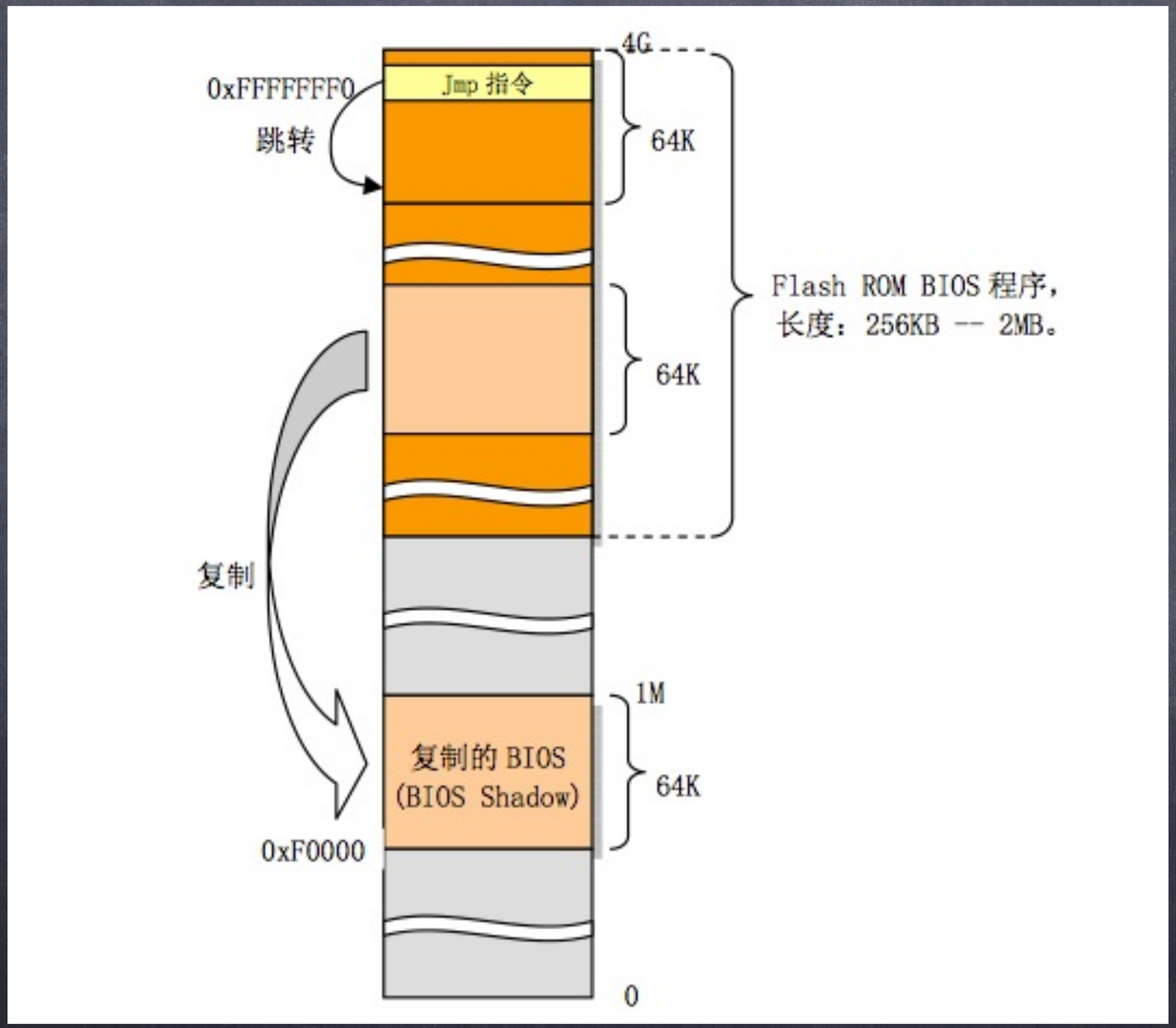
- 从名字可知，它也是一个软件系统，甚至可以说是一个简单的操作系统。
- 负责检测内存等硬件的状态，并加载Boot扇区。
- 提供一整套的硬件操作接口（通过int中断指令，先简单理解为函数），方便Boot程序调用。

加载boot扇区

- 开机后，Bios所在的ROM可以被CPU直接访问，执行一段硬件检测指令后，会复制自身到0xF0000-0xFFFFF内存区域去，即1M内存的最后64K，并跳转过去继续执行。

- 依次检测硬件存储器的第0个扇区的第510和511字节的值，如果分别是0x55，0xAA，则说明是个有效的Boot扇区，则将其复制到内存地址0x7C00处，并将CS设置为0，且IP设置为0x7C00，然后跳转到此地址开始执行，即运行Boot扇区的代码。

BIOS内存映射图


4G
0xFFFFFFF0
Jmp 指令
跳转
64K
Flash ROM BIOS 程序，
长度：256KB -- 2MB。
64K
复制
1M
复制的 BIOS
(BIOS Shadow)
64K
0xF0000
0

CPU的基本执行原理

- CPU就像推石头的西西弗斯，不断的从CS：IP指定的内存地址获取到指令，并执行。

- CS和IP都是16位的寄存器，在实模式与保护模式课上，会更深入的讲解。

Boot的功能

- Boot程序会通过BIOS提供的存储器读取中断指令，读取更多的内容到内存中，并从实模式到保护模式（寻址范围从1M到4G），然后就可以加载进来真正的内核了。
- 内核如何初始化，我们后面的章节会讲。

BIOS提供的中断调用

- int 0x10；用于屏幕显示
  - 寄存器AH = 0x0E时，表示显示AL对应的字符。
  - 寄存器AH = 0x13时，表示显示一个字符串。ES：BP对应字符串的地址，CX存放字符串的长度。（DH、DL）= 坐标（行、列）

第一个实验

- 打印寄存器CS的值

编写汇编代码

```
 1 mov ax, cs ;将cs寄存器的值保存到al
 2 mov al, ah ;先获取cs的高8位
 3 add al, 48 ;48是字符0的asc码，这样便于显示
 4 mov ah, 0x0E ;ah用于bios字符显示中断的类型选择
 5              ;0x0e表示显示一个字符
 6 int 0x10
 7
 8 mov ax, cs ;获取cs的值，主要是低8位
 9 add al, 48
10 mov ah, 0x0E
11 int 0x10
12
13 hlt ; 让CPU停止运行
14
15 times 510 - ($-$$) db 0 ;填充一堆的0
16                          ;$表示当前位置，$$表示文件头部
17 db 0x55
18 db 0xAA ;上面两行用于设置引导扇区的标识
```

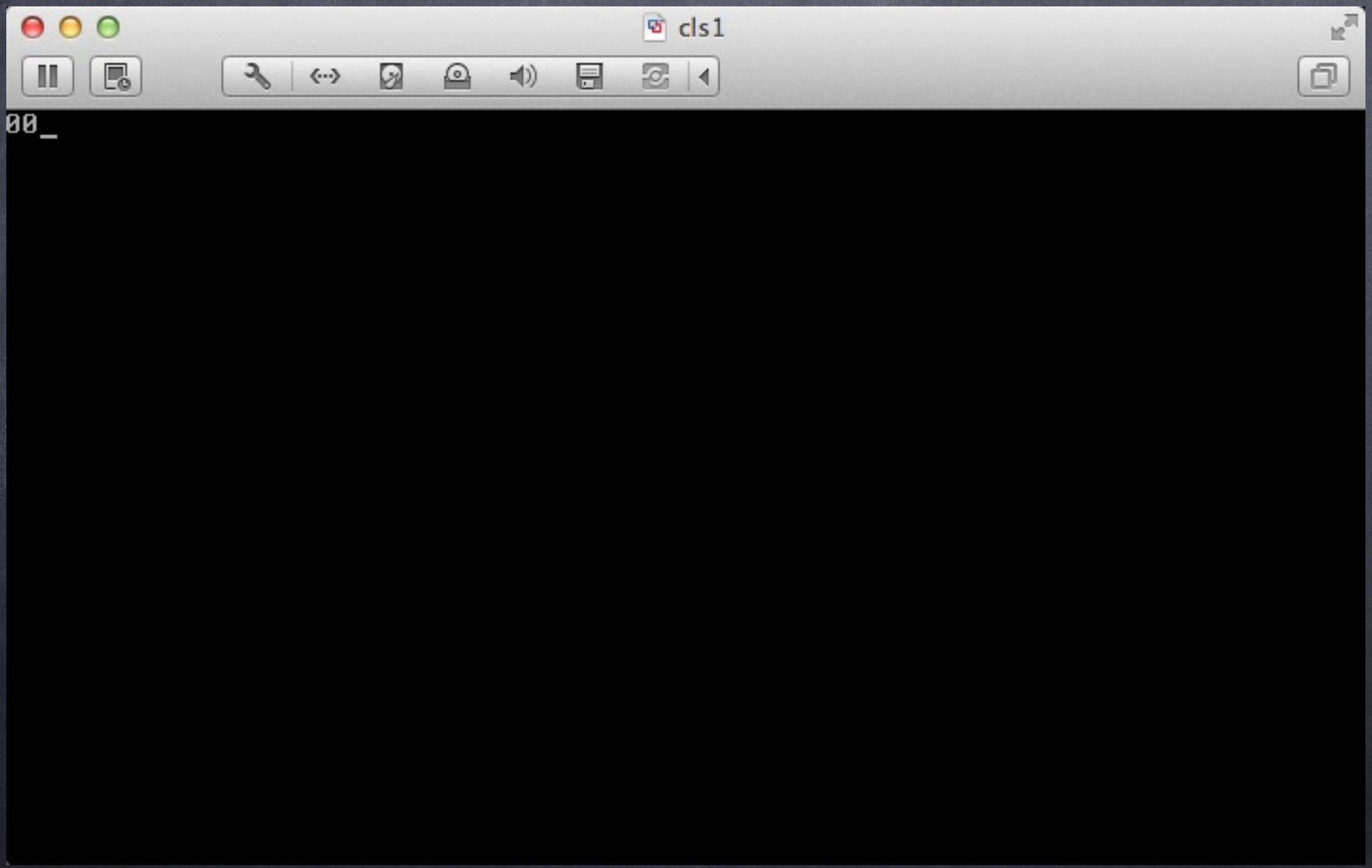
制作启动软盘

- 保存为文件checkcs.s
- 使用nasm汇编生成机器指令:
  - nasm -f bin checkcs.s -o checkcs.bin
- 生成一个启动软盘：
  - dd conv=sync if=checkcs.bin of=boot.img bs=1440k count=1

使用虚拟机VMWare运行

- 打开虚拟机资源库，添加一个新系统，类型选（其它、其它），并添加设备软盘。
- 添加软盘时，会让你选择软盘文件，就选择刚只制作的boot.img文件，然后启动运行。

打印寄存器CS的运行结果



第二个实验

- 在屏幕打印一个字符串Hello, world!

代码

```
1  BOOTSEG equ 0x07C0 ;宏定义，用于初始化ES
2
3  mov bp, HelloMsg ;将HelloMsg的地址赋值给bp
4  mov ax, BOOTSEG
5  mov es, ax          ;es:bp = 字符串地址
6                      ;HelloMsg只是段内的偏移地址
7                      ;而代码被加载到0x7C00，故设置段地址
8                      ;即es * 16 + bp为实际内存地址
9  mov cx, 13 ;串的长度
10 mov ax, 0x1301 ;AH = 0x13，表示串打印
11                ;AL = 0x01，表示打印后光标移动
12 mov bx, 0x000C ;BH = 0x00，表示页码
13                ;BL = 0x0C，表示红色
14 mov dx, 0x0000 ;第0行0列开始
15 int 0x10
```

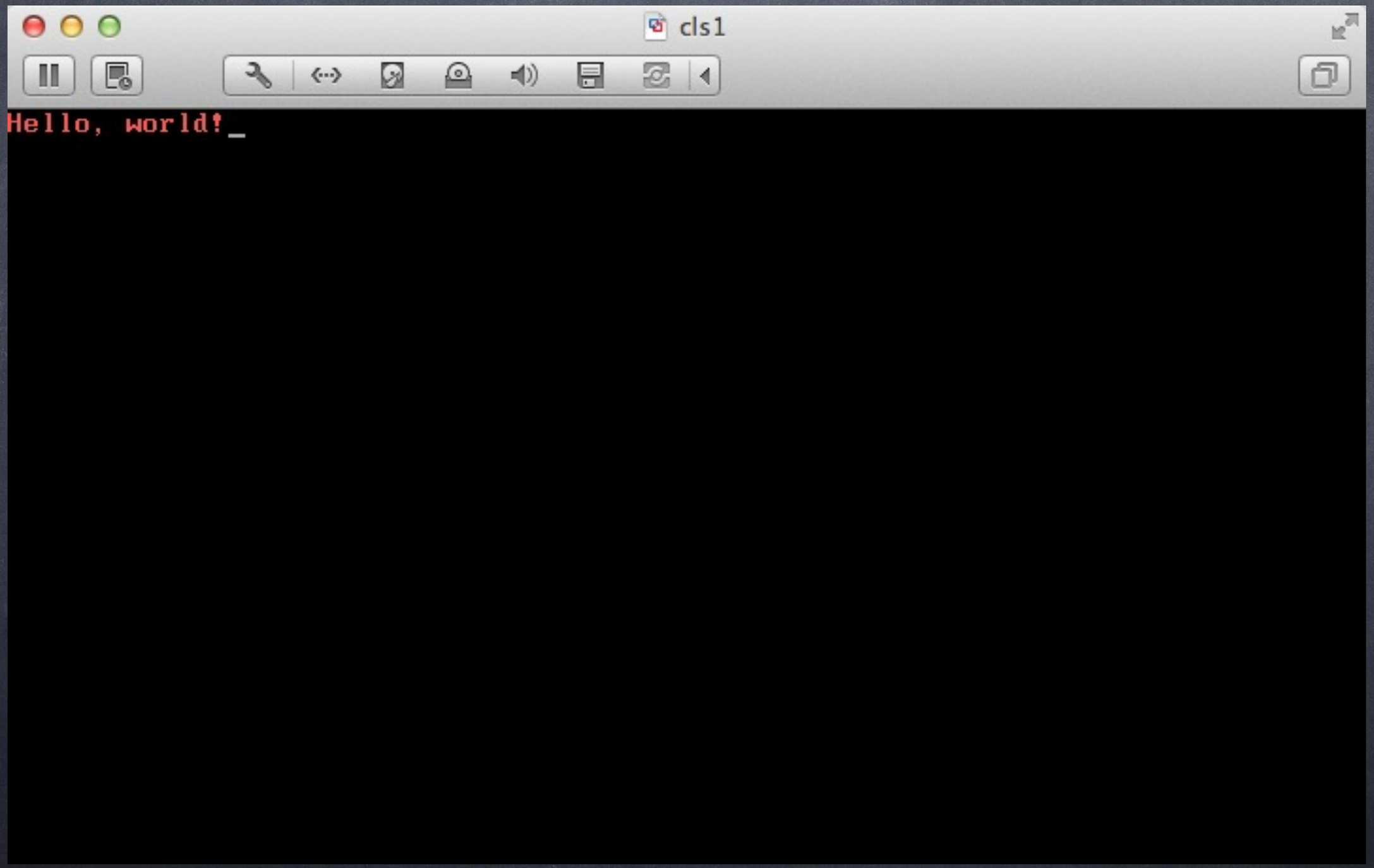
代码

```
17 loop1:
18     jmp loop1 ;jmp段内跳转，是相对当前地址
19               ;翻译成机器指令后，是0xeb 0xfe
20               ;0xeb表示jmp段内跳转的指令号
21               ;0xfe即-2，表示往前跳2个字节
22               ;这是因为执行本指令时，IP已经移到下条
23               ;指令开头了。
24               ;所以不会是jmp 0x00这样的写法。
25
26 HelloMsg:
27     db "Hello, world!" ;共13个字符
28
29 times 510 - ($ - $$) db 0 ;填充一堆的0
30                           ;$表示当前位置，$$表示文件头部
31 db 0x55
32 db 0xAA ;上面两行用于设置引导扇区的标识
```

生成软盘镜像

- 汇编：
  - nasm -f bin printhello.s -o printhello.bin
- 生成镜像：
  - dd conv=sync if=printhello.bin of=helloboot.img bs=1440k count=1

运行结果



实验总结

- 本质上，我们的实验是BIOS系统下写的一个应用。
- BIOS就是一个操作系统，我们调用了int 0x10与BIOS打交道。这和真正的操作系统下，应用程序与操作系统内核打交道的方式是一致的。

思考

- 我们在实验2中，我们使用jmp loop1，可以用jmp BOOTSEG: loop1吗，这有什么差异?
- 为什么现代操作系统中，不使用BIOS提供的硬件访问接口，而是重新实现一套？

谢谢！