哈工大操作系统-L2笔记

0.预备知识

0.1关于汇编的一些网络笔记:

https://blog.csdn.net/PGZXB/article/details/118443675?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_baidulandingword~default-0.control&spm=1001.2101.3001.4242

https://blog.csdn.net/qq_39654127/article/details/88698911?utm_medium=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control

0.2关于实验:

https://www.lanqiao.cn/courses/115

0.3关于内存和寻址:

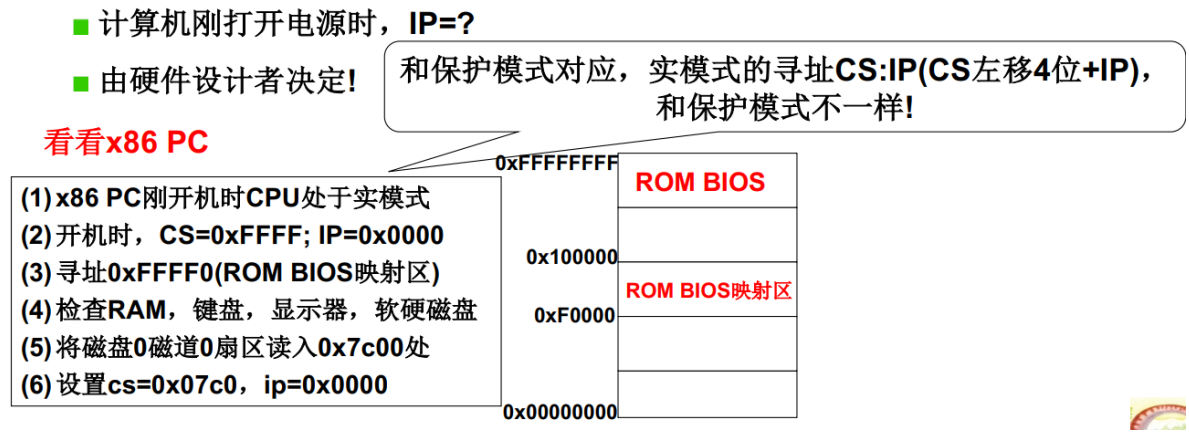
- 内存是以字节为单位的。
- 实模式寻址参考 至于实模式寻址，8086一共有20条地址线，但寄存器只有16bit。所以采取CS+IP的寻址方式。CS左移4位+IP。

0.4关于8086的各个寄存器:

https://blog.csdn.net/weixin_40913261/article/details/90762210

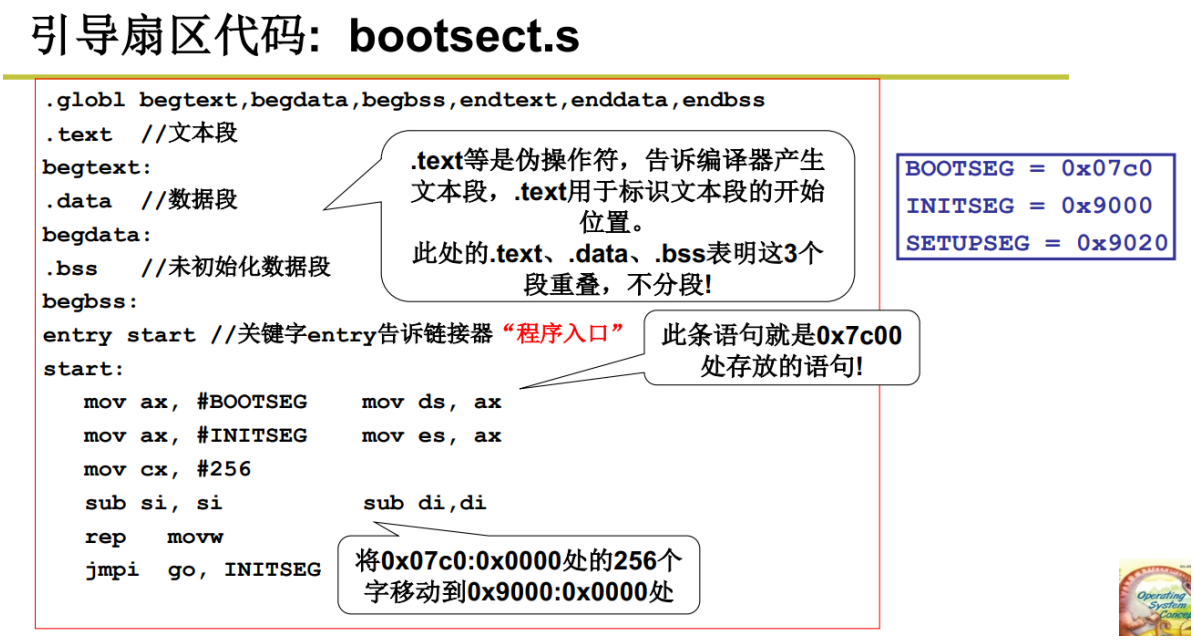
1.刚打开电源时，计算机做了什么(bootsect.s)

1.1 读入BIOS后，再读入引导扇区



- 内存中有一部分是固化的，为ROM BIOS，满足系统的基本输入输出功能。
- 因此，一开机，计算机会自动寻址ROM BIOS在内存中的映射区（即0XFFFF0)
- 检查完系统硬件之后，会读入**0磁道0扇区(引导扇区)**的512个字节到内存的0x7c00处。
  - 引导扇区就是启动设备的第一个扇区，开机界面也在这个区。
  - 引导扇区的代码叫bootsect.s，是一个汇编代码。

1.2 bootsect.s开始引导1--从0x7c00移动到0x9000



- (move 到 rep的5行)bootsect.s将内存中引导扇区的512个字节从0x7c00处直接移动到了0x9000处。
  - rep movw配合使用，rep是重复cx寄存器中的次数, movw将ds:si指向的内存字单元中的字送入es:di中，然后根据标志寄存器df位的值，将si和di递增2或递减2。
  - **为何要移动到0x9000处：为操作系统移动到0地址腾出空间。**是因为后面setup.s要将操作系统从0地址开始存放，如果操作系统很长将会把7c00的东西也覆盖掉，也会将后面正在执行的setup.s覆盖。
- (jmpi的一行)然后bootsect.s跳到INITSEG处继续执行。
  - 首先，代码开头已经把bootsect.s移动到了INITSEG。
  - 那我们如果要继续执行bootsect.s，应该是到INITSEG那边找jmpi的下一行代码。
  - 而jmpi的下一部分的段落名字就叫go。
  - 所以jmpi go, INITSEG，就是跳转到CS=INITSEG, IP=go的地方。继续执行bootsect.s。

1.3 bootsect.s开始引导2--读入setup的4个扇区

jmpi go, INITSEG

- jmpi (jump intersegment段间跳转): cs=INITSEG, ip=go

```
go: mov ax,cs //cs=0x9000
    mov ds,ax  mov es,ax  mov ss,ax  mov sp,#0xff00
load_setup:  //载入setup模块
    mov dx,#0x0000   mov cx,#0x0002  mov bx,#0x0200
    mov ax,#0x0200+SETUPLEN  int 0x13 //BIOS中断
    jnc ok_load_setup
    mov dx,#0x0000
    mov ax,#0x0000 //复位
    int 0x13
    j   load_setup //重读
```

为call做准备!

0x13是BIOS读磁盘扇区的中断: ah=0x02-读磁盘，al=扇区数量(SETUPLEN=4)，ch=柱面号，cl=开始扇区，dh=磁头号，dl=驱动器号，es:bx=内存地址

INT	AH	功 能 说 明	入 口 参 数	返 回 参 数
13	2	读磁盘	AL=扇区数 CH,CL=磁盘号，扇区号 DH,DL=磁头号，驱动器号 ES:BX=数据缓冲区地址	读成功：AH=0 AL=读取的扇区数 读失败：AH=出错代码

- go 这一段，sp和ss没看懂。但是其余的都是在设置int0x13中断的参数。
  - 由下表可以看出，int0x13，如果要读磁盘，则ax的高八位ah为0x0200，低八位是扇区数。然后es和bx是内存的地址。可以看到es是0x9000而bx是0x0200，因此最终地址为0x9000_0020_0，因为是16进制的，所以从000到200处为512个B，也就保存着我们的Bootsect.s。而磁盘的接着的4个扇区则从200之后开始。

1.3 显示在开机

读入setup模块后: ok_load_setup

```
Ok_load_setup:  //载入setup模块
    mov dl,#0x00  mov ax,#0x0800 //ah=8获得磁盘参数
    int 0x13      mov ch,#0x00   mov sectors,cx
    mov ah,#0x03  xor bh,bh      int 0x10 //读光标
    mov cx,#24    mov bx,#0x0007
    mov bp,#msg1  mov ax,#1301   int 0x10 //显示字符
    mov ax,#SYSSEG //SYSSEG=0x1000
    mov es,ax
    call read_it //读入system模块
    jmpi 0,SETUPSEG
```

显示这24个字符将是大家的第一个“创举”！

7是显示属性!

转入0x9020:0x0000
执行setup.s

```
bootsect.s中的数据 //在文件末尾
sectors: .word 0 //磁道扇区数
msg1:.byte 13,10
     .ascii "Loading system..."
     .byte 13,10,13,10
```

- boot工作:读setup，读system...

- 读取光标位置，然后将字符显示在光标的位置上。显示则使用int0x10中断。

1.4把剩余的system也读入内存

read_it //读入system模块

system模块可能很大，要跨越磁道!

- 为什么读入system模块还需要定义一个函数?

```
read_it:  mov ax,es     cmp ax,#ENDSEG    jb ok1_read
          ret
ok1_read:
   mov ax,sectors
   sub ax,sread //sread是当前磁道已读扇区数,ax未读扇区数
   call read_track //读磁道...
```

ENDSEG=SYSSEG+SYSSIZE
SYSSIZE=0x8000 //该变量可根据Image大小设定(编译操作系统时)

- 引导扇区的末尾 //BIOS用以识别引导扇区

否则会打出非引导设备

```
.org 510
   .word 0xAA55 //扇区的最后两个字节
```

- 可以转入setup执行了，jmpi 0, SETUPSEG

- 使用read_it读入剩余的系统
- 读入完毕后，我们的bootsect.s就执行完毕了
- 接下来交给setup.s