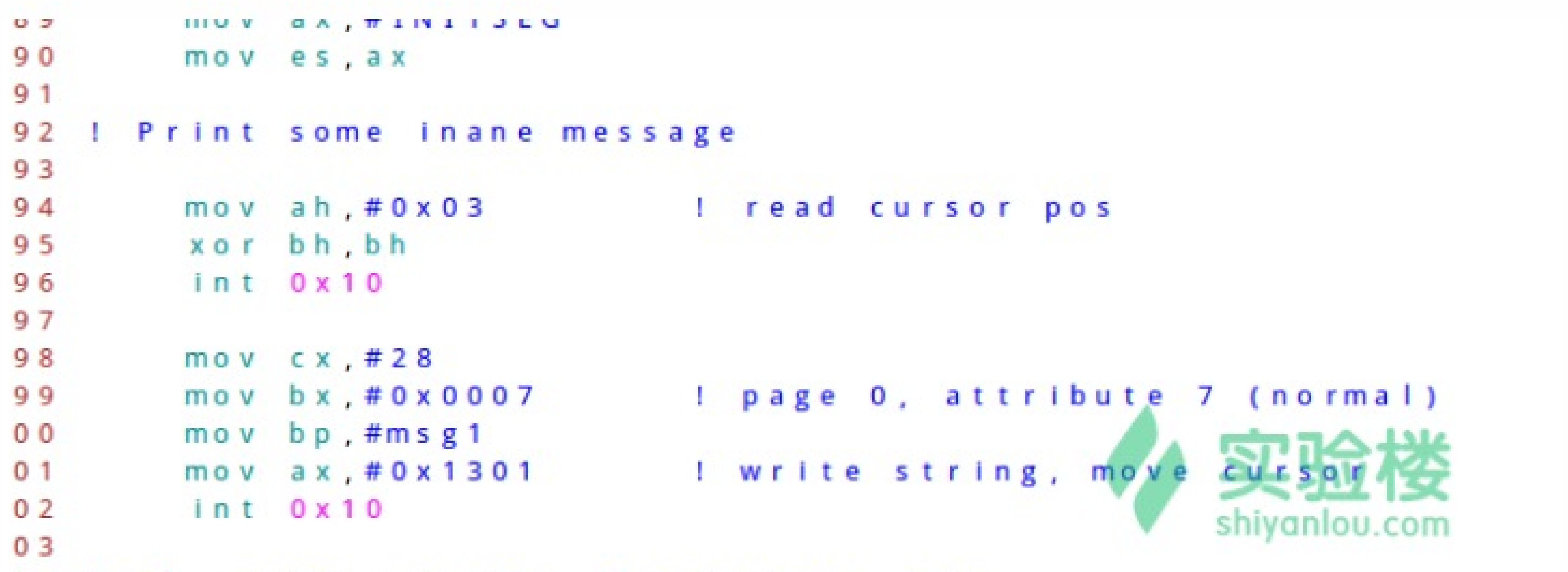
操作系统的引导

一.改写bootsect.s在屏幕上打出的Logo信息

1.注意msg1段中还含有回车,换行,计算总字节数时不要漏了

```
        .word 0

msg1:
        .byte 13,10
        .ascii "BrickOS is booting..."
        .byte 13,10,13,10

```

所以CX=22+6=28 bytes

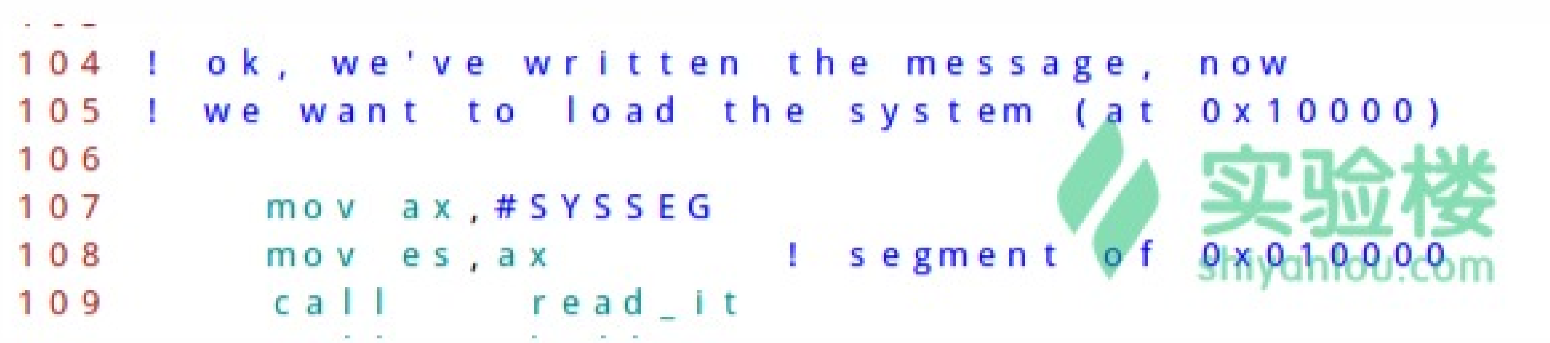


```
        mov es,ax

! Print some inane message

        mov ah,#0x03            ! read cursor pos
        xor bh,bh
        int 0x10

        mov cx,#28
        mov bx,#0x0007          ! page 0, attribute 7 (normal)
        mov bp,#msg1
        mov ax,#0x1301          ! write string, move cursor
        int 0x10

```

2.删掉bootsect.s中多余的代码.因为实验要求中说:

> setup.s不再加载Linux内核.

即无需加载system模块,所以bootsect.s中加载system的部分都可删掉. 比如,这个把system读到0x10000的read_it函数删掉



```
! ok, we've written the message, now
! we want to load the system (at 0x10000)

        mov ax,#SYSSEG
        mov es,ax               ! segment of 0x010000
        call    read_it
```

```
! no 64kB boundaries are crossed. We try to load it as fast as
! possible, loading whole tracks whenever we can.
!
! in:   es - starting address segment (normally 0x1000)
!
sread:  .word 1+SETUPLEN        ! sectors read of current track
head:   .word 0                 ! current head
track:  .word 0                 ! current track

read_it:
        mov ax,es
        test ax,#0x0fff
die:    jne die                 ! es must be at 64kB boundary
        xor bx,bx               ! bx is starting address within segment
rp_read:
        mov ax,es
        cmp ax,#ENDSEG          ! have we loaded all yet?
        jb ok1_read
        ret
ok1_read:
        seg cs
        mov ax,sectors
        sub ax,sread
        mov cx,ax
        shl cx,#9
        add cx,bx
        jnc ok2_read
        je ok2_read
        xor ax,ax
        sub ax,bx
        shr ax,#9
ok2_read:
        call read_track
        mov cx,ax
        add ax,sread
        seg cs
        cmp ax,sectors
        jne ok3_read
```

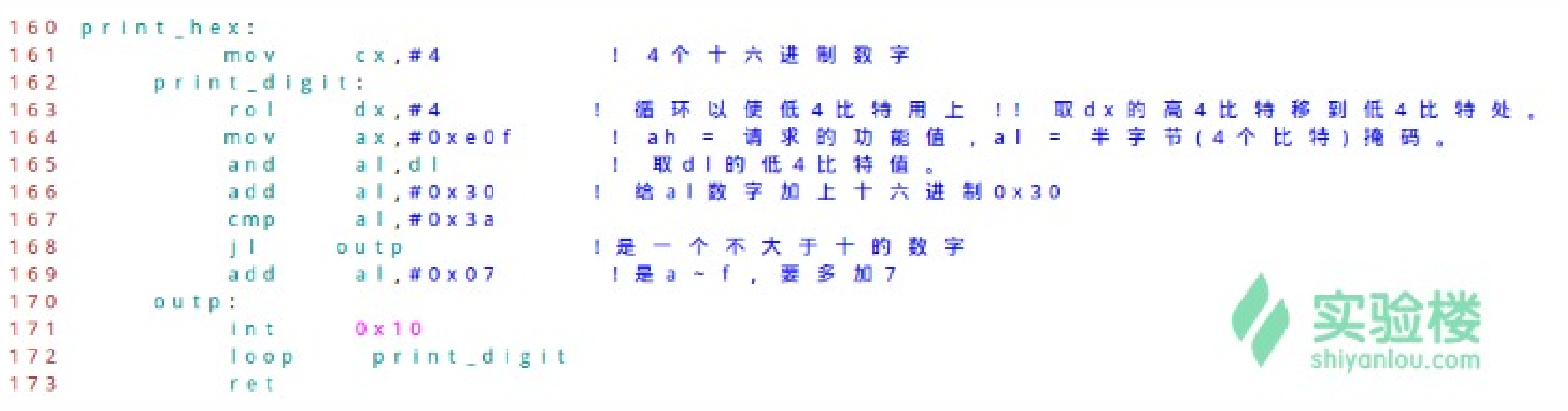
二.改写setup.s

1.在准备好标号msg2后,在setup.s的起始处打出"Now we are in SETUP"

```
entry start
start:

! ok, the read went well so we get current cursor position and save it for
! posterity.
        mov ax,cs               !因为跳到cs==0x9020处执行setup,所以es的值也要随之改变
        mov es,ax

        mov ah,#0x03            ! read cursor pos
        xor bh,bh
        int 0x10

        mov cx,#23
        mov bx,#0x0007          ! 打出弱智信息
        mov bp,#msg2
        mov ax,#0x1301          ! write string, move cursor
        int 0x10

        mov ax,#INITSEG ! this is done in bootsect already, but...
        mov ds,ax
        mov ah,#0x03    ! read cursor pos
        xor bh,bh
        int 0x10        ! save it in known place, con_init fetches
        mov [0],dx      ! it from 0x90000.
```

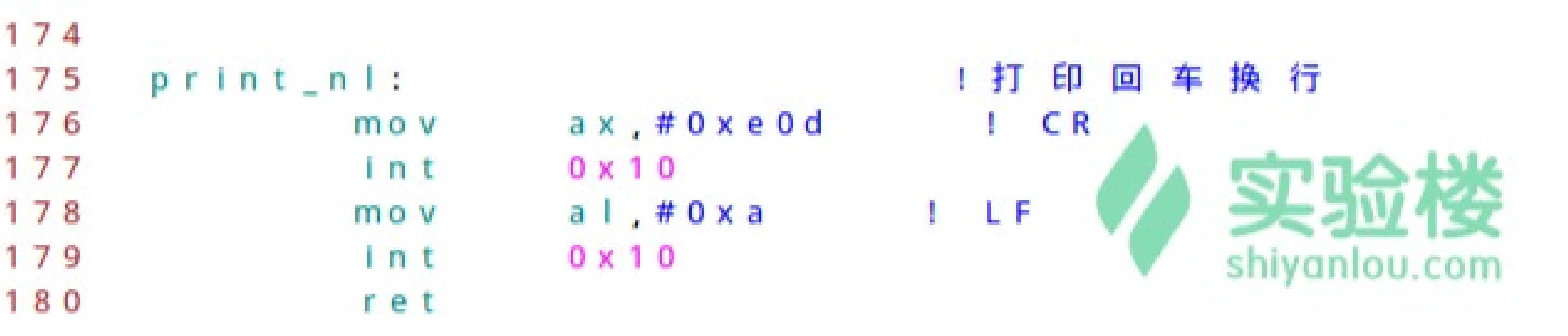
2.打出硬件信息

-把各硬件的参数以16进制输出,所以准备好函数print_hex,函数参数存放在DX中



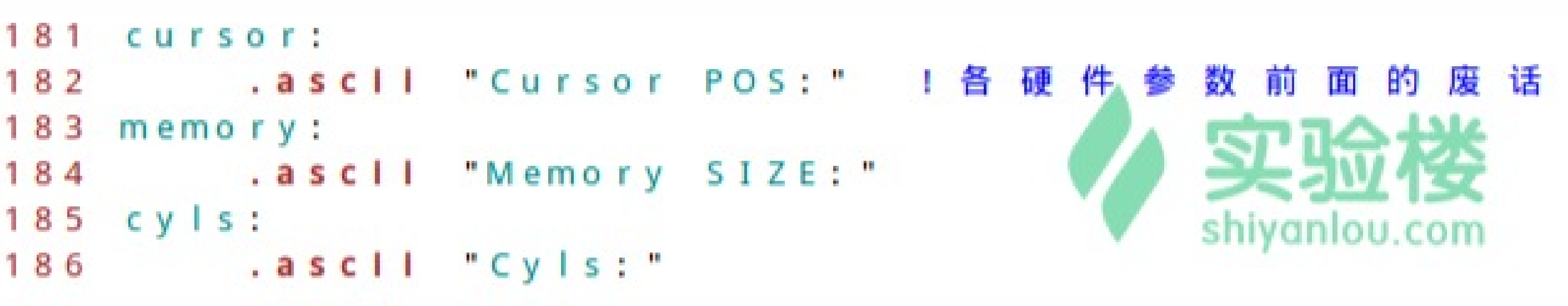
```
print_hex:
            mov     cx,#4           ! 4个十六进制数字
        print_digit:
            rol     dx,#4          ! 循环以使低4比特用上 !! 取dx的高4比特移到低4比特处。
            mov     ax,#0xe0f       ! ah = 请求的功能值，al = 半字节(4个比特)掩码。
            and     al,dl           ! 取dl的低4比特值。
            add     al,#0x30       ! 给al数字加上十六进制0x30
            cmp     al,#0x3a
            jl    outp             !是一个不大于十的数字
            add     al,#0x07        !是a~f，要多加7
        outp:
            int     0x10
            loop     print_digit
            ret
```

-还要专门用一个函数打印换行回车



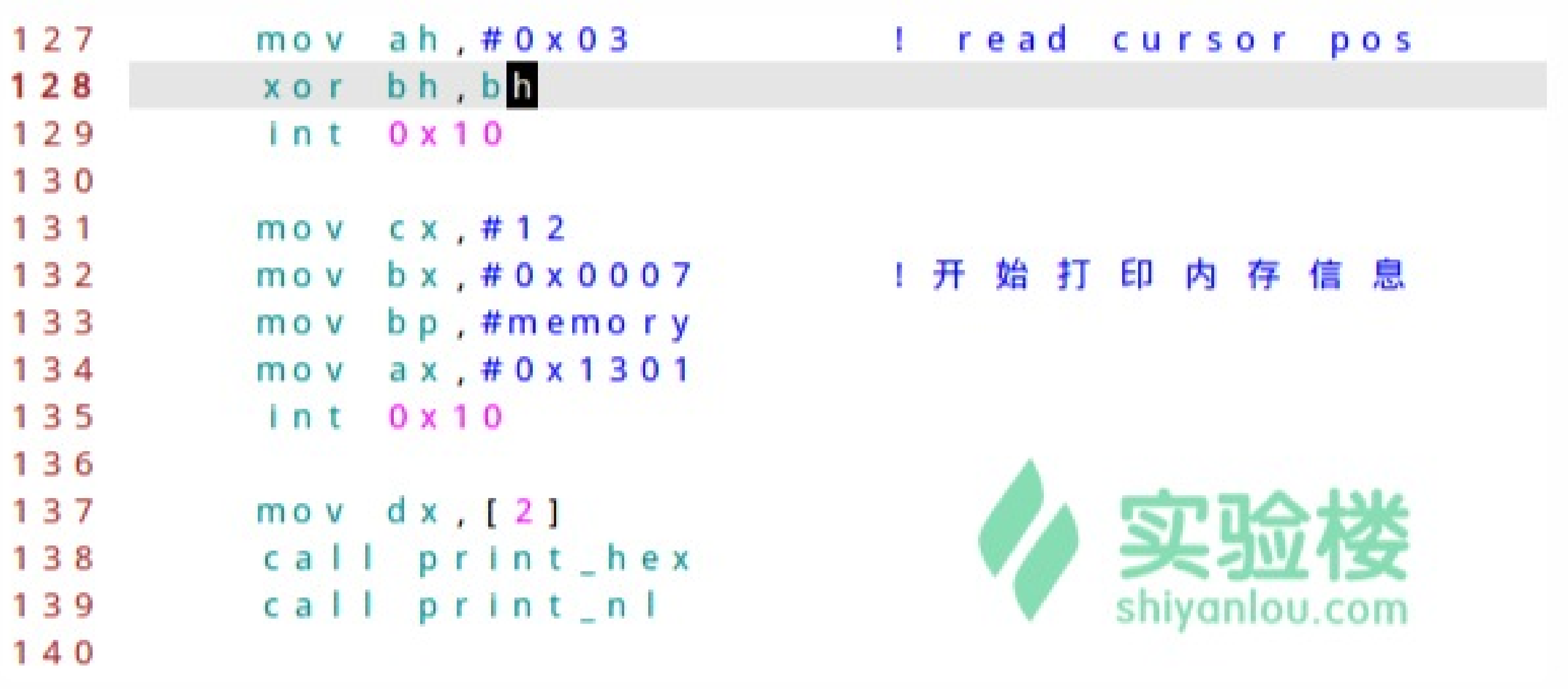
```

 print_nl:                          !打印回车换行
            mov     ax,#0xe0d       ! CR
            int     0x10
            mov     al,#0xa       ! LF
            int     0x10
            ret
```

-准备好硬件参数前的提示信息



```
cursor:
        .ascii "Cursor POS:"   !各硬件参数前面的废话
memory:
        .ascii "Memory SIZE:"
cyls:
        .ascii "Cyls:"
```

-然后就可以正式的输出硬件信息了.以内存信息为例,先用0x10中断定位光标位置,再用该中断的 0x13号功能打印提示信息.最后调用print_hex打印16进制的硬件参数



```
        mov ah,#0x03            ! read cursor pos
        xor bh,bh
        int 0x10

        mov cx,#12
        mov bx,#0x0007          !开始打印内存信息
        mov bp,#memory
        mov ax,#0x1301
        int 0x10

        mov dx,[2]
        call print_hex
        call print_nl

```

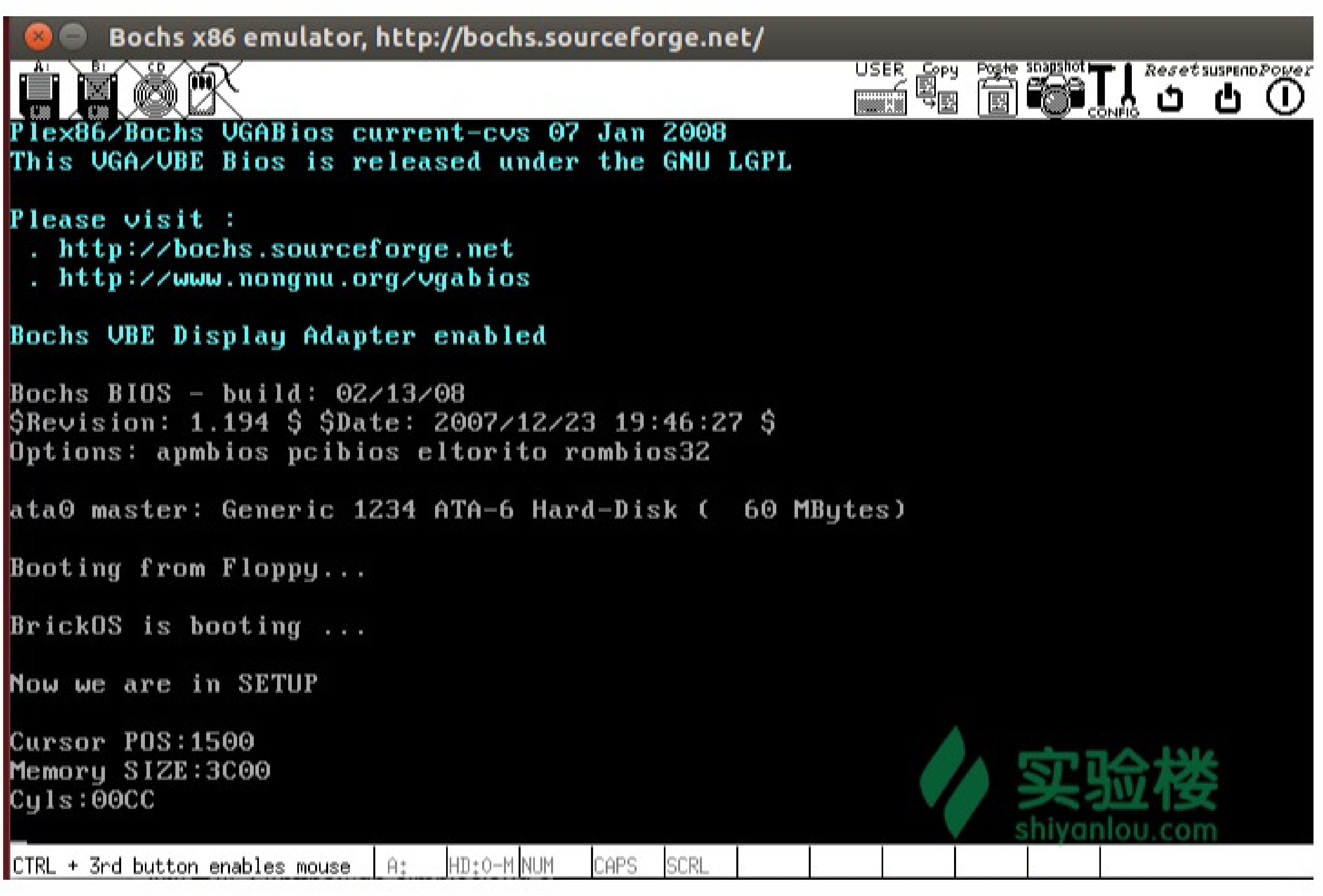
3.删除多余的代码 同理,由于无需继续载入system模块,所以把setup.s中诸如 -"把system模块移动到0x0000处" -"重设中断向量表" -"从实模式进入保护模式" -"跳到system模块执行" 等等都可删除

4.也因为现在这个操作系统只剩bootsect.s和setup.s了,不可能继续往下执行,所以在打印玩硬件参数后, 用一个无限循环保持上述信息在屏幕上显示

```

        pop dx

unlimit:
        jmp unlimit

```

三.运行结果

如图



```
Plex86/Bochs VGABios current-cvs 07 Jan 2008
This VGA/VBE Bios is released under the GNU LGPL

Please visit :
 . http://bochs.sourceforge.net
 . http://www.nongnu.org/vgabios

Bochs VBE Display Adapter enabled

Bochs BIOS - build: 02/13/08
$Revision: 1.194 $ $Date: 2007/12/23 19:46:27 $
Options: apmbios pcibios eltorito rombios32

ata0 master: Generic 1234 ATA-6 Hard-Disk (  60 MBytes)

Booting from Floppy...

BrickOS is booting ...

Now we are in SETUP

Cursor POS:1500
Memory SIZE:3C00
Cyls:00CC
```

与Bochs的配置文件bochs/bochsrc.bxrc中的信息进行对比:

```
megs: 16
vgaromimage: file=$OSLAB_PATH/bochs/vgabios.bin
floppya: 1_44="$OSLAB_PATH/linux-0.11/Image", status=inserted
ata0-master: type=disk, path="$OSLAB_PATH/hdc-0.11.img", mode=flat, cylinders=204, heads=16, spt=38
boot: a
log: $OSLAB_PATH/bochsout.txt
```

发现参数都是吻合的,说明实验成功

四.找出x86为了向下兼容而多此一举的步骤

通过这个实验,个人认为有:

1.必须以老掉牙的实模式启动操作系统,则可寻址的范围只有那1M多.尽管当下计算机的内存大小以及寻址能力已经大大超过这个范围了

2.由于1中的原因,编写的bootsect.s和setup.s所占空间都不能太大,以免超过实模式的寻址范围.