# Day 3

## Phase 1

### SubPhase 1

由于计算机只会把软驱中开始512字节作为启动区内容读入到内存当中，因此如果我们想要向内存中加载更多的程序内容，就需要利用一开始的这512字节内容，写一个IPL。他就像一个间谍一样（？），会把更多的自己人（代码）请到内存里来。Phase 1的任务是写IPL，将软盘中的更多内容读取到内存当中

osdev上面关于(AT)BIOS的介绍页面挂了，看不了文档，只能看书上的介绍。

```
AH=0x02；（读盘）
AH=0x03；（写盘）
AH=0x04；（校验）
AH=0x0c；（寻道）
AL=处理对象的扇区数；（只能同时处理连续的扇区）
CH=柱面号 &0xff;
CL=扇区号（0-5位）|（柱面号&0x300）>>2;
DH=磁头号；
DL=驱动器号；
ES:BX=缓冲地址；（校验及寻道时不使用）
返回值：
FLACS.CF==0：没有错误，AH==0
FLAGS.CF==1：有错误，错误号码存入AH内（与重置（reset）功能一样）
```

先不看书上的代码，自己编一波试试

```
mov ah, 0x02              ; 读盘
mov al, 1                 ; 1个扇区（512字节）
mov ch, 0                 ; 柱面
mov cl, 2                 ; 扇区
mov dh, 0                 ; 正面
mov dl, 0                 ; 这里一开始不知道写啥，最后参考了下书上代码
                          ; 是0的原因好像和软驱一般是A盘的原因差不多。
                          ; 可能我漏掉了什么东西
mov es, ???               ; 欲装载到的内存位置，
mov bx, ???               ; 暂时留空
int 0x13                  ; 读盘!
; ----------------------------------------
; 以下内容是看了书上代码之后添加的
jc  error                  ; 错误捕捉
```

**填补必要的代码后尝试用nask进行编译，提示有错误。nask只会提示有编译错误，但是不会告诉你在第几行，真是太垃圾了。**

与书上代码进行对比之后，发现要存到es中的立即数去ax里面转了一圈。也就是说，欲存入es中的立即数，先被存入了ax，然后从ax被mov到了es。 我在网上查到了如下资料 https://blog.csdn.net/bytxl/article/details/49487917

然后进行了修改

```
mov ax, ???
mov es, ax                  ; 欲装载到的内存位置，
mov bx, ???                 ; 暂时留空
mov ah, 0x02                ; 读盘
mov al, 1                   ; 1个扇区（512字节）
mov ch, 0                   ; 柱面
mov cl, 2                   ; 扇区
mov dh, 0                   ; 正面
mov dl, 0                   ; 这里一开始不知道写啥，最后参考了下书上代码
                            ; 是0的原因好像和软驱一般是A盘的原因差不多。
                            ; 可能我漏掉了什么东西
int 0x13                    ; 读盘!
; ----------------------------------------
; 以下内容是看了书上代码之后添加的
jc  error                    ; 错误捕捉
```

## SubPhase 2

书上介绍说软盘读取出错的概率比较高，有时候读取错误并不是因为软盘坏掉了，只要再尝试几次就好了。我也改动一下自己的代码

```
mov ax, 0x0820
mov es, ax                  ; 欲装载到的内存位置，
mov bx, 0
mov si, 0                   ; 初始化计数器，和书上用一样的吧
retry:
mov ah, 0x02                ; 读盘
mov al, 1                   ; 1个扇区（512字节）
mov ch, 0                   ; 柱面
mov cl, 2                   ; 扇区
mov dh, 0                   ; 正面
mov dl, 0                   ; A盘
int 0x13                    ; 读盘!
jnc complete                ;成功
add si, 1
cmp si, 5
jae error                   ; 试5次
mov ah, 0x00
mov dl, 0x00
int 0x13                    ; 软驱重置
jmp retry
```

## SubPhase 3

撸到18扇区

```
mov ax, 0x0820
mov es, ax                  ; 欲装载到的内存位置，
mov bx, 0                   ; 暂时留空
```

```
mov cl, 2                   ; 扇区

rloop:
mov si, 0                   ; 初始化计数器，和书上用一样的吧

retry:
mov ah, 0x02                ; 读盘
mov al, 1                   ; 1个扇区（512字节）
mov ch, 0                   ; 柱面
mov dh, 0                   ; 正面
mov dl, 0                   ; A盘
int 0x13                    ; 读盘!
jnc complete                ;成功
add si, 1
cmp si, 5
jae error                   ; 试5次
mov ah, 0x00
mov dl, 0x00
int 0x13                    ; 软驱重置
jmp retry
complete:
mov ax, es
add ax, 32                  ; 段寄存器不允许add指令
mov es, ax
add cl, 1
cmp cl, 18
jbe rloop
```

## SubPhase 4

撸背面

```
mov ax, 0x0820
mov es, ax                  ; 欲装载到的内存位置，
mov bx, 0                   ; 暂时留空
mov cl, 2                   ; 扇区
mov dh, 0                   ; 正面
rloop:
mov si, 0                   ; 初始化计数器，和书上用一样的吧

retry:
mov ah, 0x02                ; 读盘
mov al, 1                   ; 1个扇区（512字节）
mov ch, 0                   ; 柱面
mov dl, 0                   ; A盘
int 0x13                    ; 读盘!
jnc complete                ;成功
add si, 1
cmp si, 5
jae error                   ; 试5次
mov ah, 0x00
mov dl, 0x00
int 0x13                    ; 软驱重置
```

```
jmp retry
complete:
mov ax, es
add ax, 32                  ; 段寄存器不允许add指令
mov es, ax
add cl, 1
cmp cl, 18
jbe rloop
mov cl, 1
add dh, 1
cmp dh, 2
jb  rloop
```

撸更多的柱面

```
mov ax, 0x0820
mov es, ax                  ; 欲装载到的内存位置，
mov bx, 0                   ; 暂时留空
mov cl, 2                   ; 扇区
mov dh, 0                   ; 正面
mov ch, 0                   ; 柱面
rloop:
mov si, 0                   ; 初始化计数器，和书上用一样的吧

retry:
mov ah, 0x02                ; 读盘
mov al, 1                   ; 1个扇区（512字节）
mov dl, 0                   ; A盘
int 0x13                    ; 读盘!
jnc complete                ;成功
add si, 1
cmp si, 5
jae error                   ; 试5次
mov ah, 0x00
mov dl, 0x00
int 0x13                    ; 软驱重置
jmp retry
complete:
mov ax, es
add ax, 32                  ; 段寄存器不允许add指令
mov es, ax
add cl, 1
cmp cl, 18
jbe rloop
mov cl, 1
add dh, 1
cmp dh, 2
jb  rloop
mov dh, 0
add ch, 1
cmp ch, 10                  ; 读入10个柱面
jb  rloop
; ------------------------
```

```
fin:                    ; 补上error
hlt
jmp fin
```

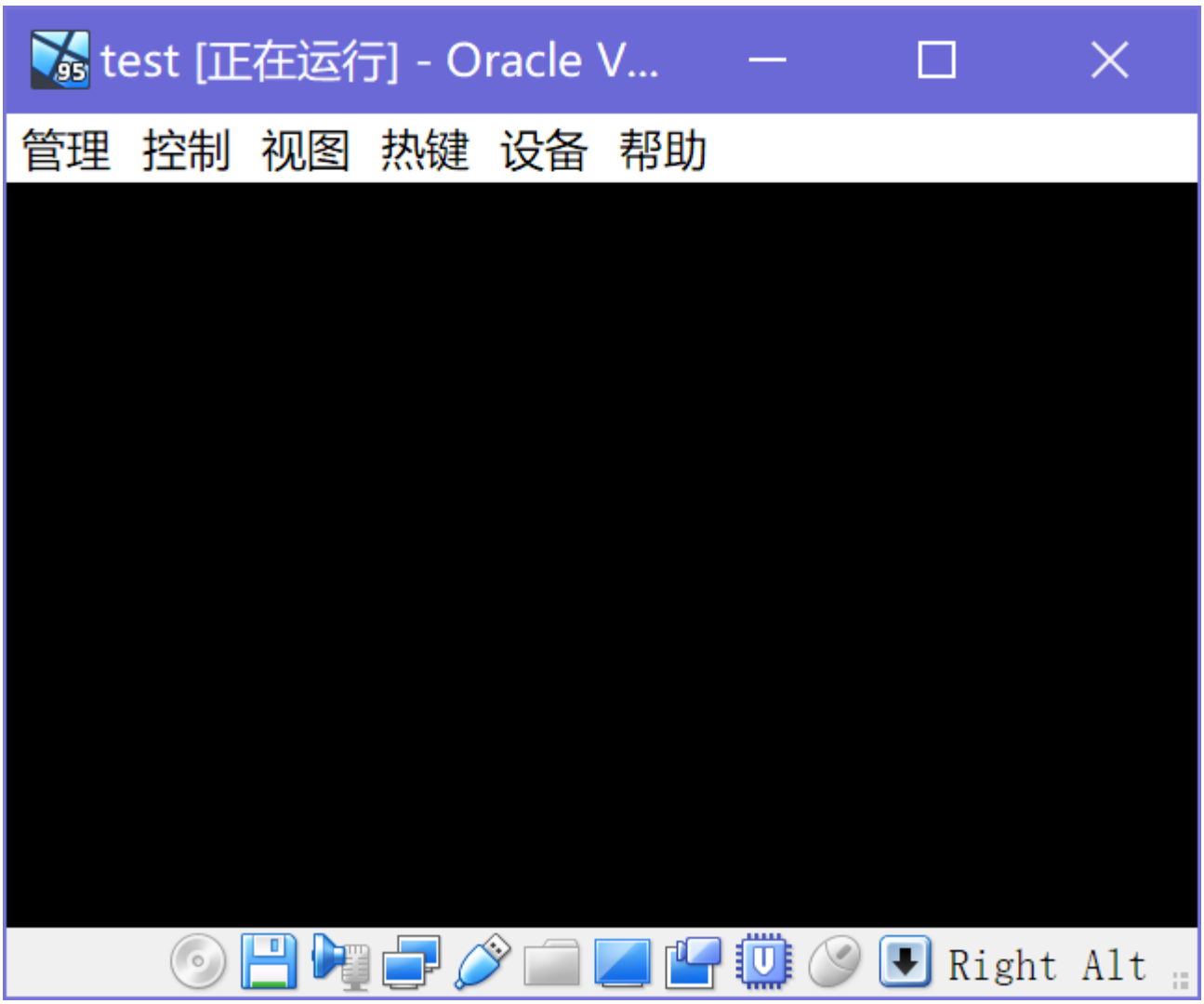
## Phase 2

发现软盘映像文件中文件的内容是从0x4200位置开始的。软盘中的内容被载入到0x8000-0x81ff（启动区），0x8200之后（其余内容）的地方。所以0x4200对应的内存地址是0xc200位置

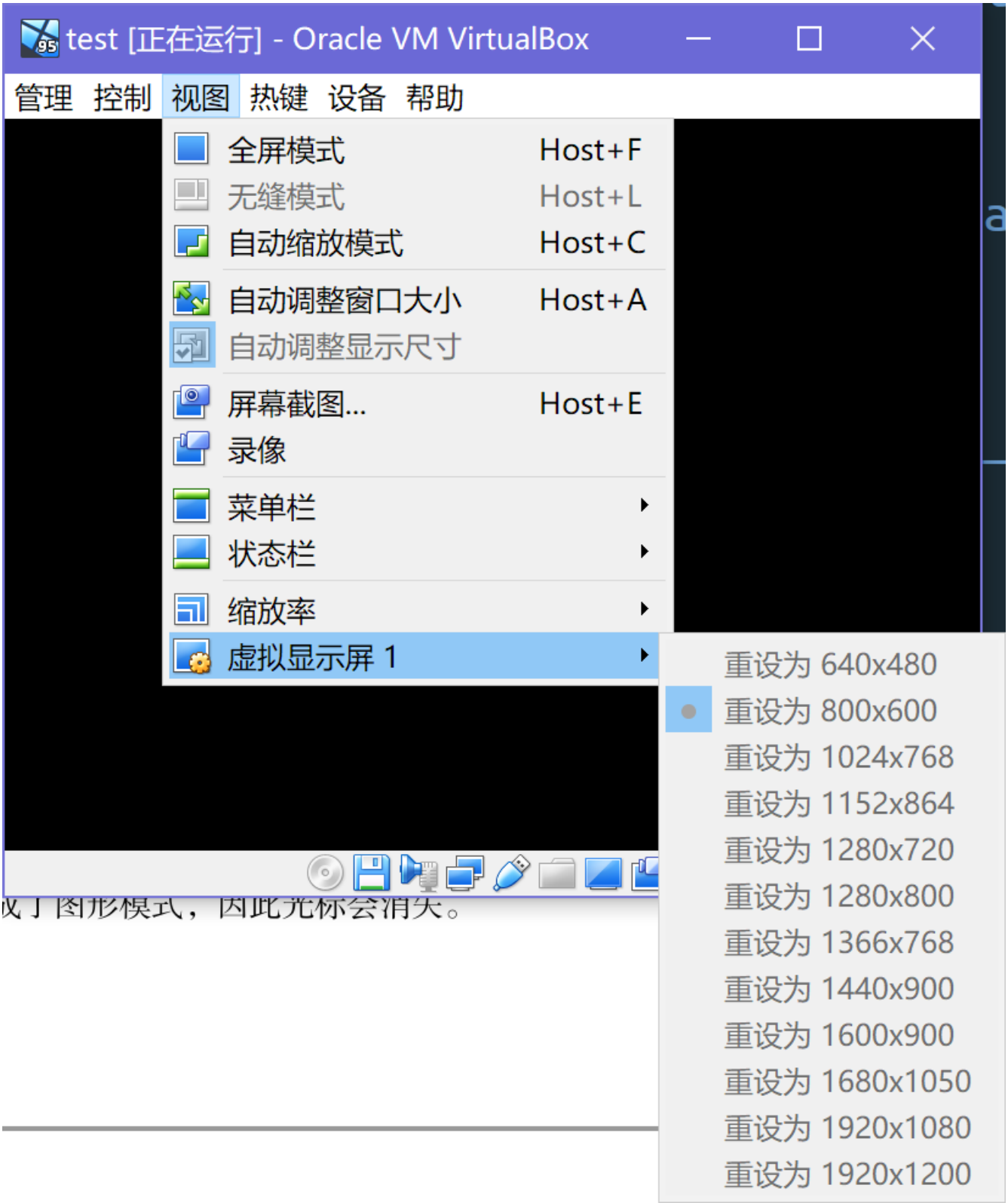
所以我们载入完软盘内容后可以jmp到0xc200的位置开始执行程序。

## Phase 3

测试图形模式，按照文档的内容修改源码，make img，扔到virtual box中跑一下




试一下0x6a




分辨率确实改变了

## Phase 4

32位模式前期准备。由于进入32位模式之后难以调用BIOS函数，所以我们要进行一些准备，保存一些必要信息。

显卡相关的信息是必要的。使用常量记录存储地址，并将相关信息保存到主存中并不是什么难事

除此之外，我们可以利用BIOS函数0x16来获取LED灯的状态。做法如下

```
mov ah, 0x02
int 0x16
mov [欲保存LED状态的主存地址], al
```

###

# Phase 5

导入C语言

为了导入C语言，作者写了一些代码，但作者并不打算现在就讲解这段程序，那我们就听他的吧。等之后再看看。

我们来更多的关注一下c语言是怎么变成最后写入软盘镜像的机器语言吧。

分析一波Makefile

```
TOOLPATH = ../z_tools/
MAKE     = $(TOOLPATH)make.exe -r
NASK     = $(TOOLPATH)nask.exe
EDIMG    = $(TOOLPATH)edimg.exe
IMGTOL   = $(TOOLPATH)imgtol.com
COPY     = copy
DEL      = del

# デフォルト動作

default :
        $(MAKE) img

# ファイル生成規則

ipl.bin : ipl.nas Makefile
        $(NASK) ipl.nas ipl.bin ipl.lst

haribote.sys : haribote.nas Makefile
        $(NASK) haribote.nas haribote.sys haribote.lst

haribote.img : ipl.bin haribote.sys Makefile
        $(EDIMG)   imgin:../z_tools/fdimg0at.tek \
                wbinimg src:ipl.bin len:512 from:0 to:0 \
                copy from:haribote.sys to:@: \
                imgout:haribote.img

# コマンド

img :
        $(MAKE) haribote.img

run :
        $(MAKE) img
        $(COPY) haribote.img ..\z_tools\qemu\fdimage0.bin
```

```
        $(MAKE) -C ../z_tools/qemu

install :
        $(MAKE) img
        $(IMGTOL) w a: haribote.img

clean :
        -$(DEL) ipl.bin
        -$(DEL) ipl.lst
        -$(DEL) haribote.sys
        -$(DEL) haribote.lst

src_only :
        $(MAKE) clean
        -$(DEL) haribote.img
```

haribote.img是通过ipl10.bin和haribote.sys用edimg工具合成的软盘镜像

ipl.bin是ipl.nas用nask编译得到的

haribote.sys是asmhead.bin和bootpack.hrb合并得来的

asmhead.bin是asmhead.nas用nask编译得到的

bootpack.hrb是bim2hrb工具转换bootpack.bim得到的。加上了文件头，进行了压缩等修改 （bim文件是链接后的映像）

bootpack.bim是obj2bim工具参照规则文件对bootpack.obj进行链接得到的

bootpack.obj是nask编译bootpack.nas得到的

bootpack.nas是使用gas2nas工具转换bootpack.gas得到的 （gas是GNU Assembler，语法与nask不同）

bootpack.gas是使用cc1处理bootpack.c得到的，是作者改造的gcc，可以直接输出gas

编译作者的代码，得到haribote.img运行，黑屏，运行正常！

# Phase 6

为c语言添加可用的函数

我们不能直接在c语言中使用汇编语言，所以我们通过链接的方式让c语言可以调用汇编函数。

按照以下格式

```
; naskfunc
; TAB=4
[FORMAT "WCOFF"] ; 制作目标文件的模式
[BITS 32] ; 制作32位模式用的机械语言
;制作目标文件的信息
[FILE "naskfunc.nas"] ; 源文件名信息
GLOBAL
① _io_hlt
; 程序中包含的函数名
;以下是实际的函数
```

```
[SECTION .text] ; 目标文件中写了这些之后再写程序
_io_hlt: ; void io_hlt(void);
HLT
RET
```

观察修改后的makefile

```
bootpack.bim : bootpack.obj naskfunc.obj Makefile
        $(OBJ2BIM) @$(RULEFILE) out:bootpack.bim stack:3136k map:bootpack.map \
                bootpack.obj naskfunc.obj
```

我们发现链接命令发生了改变。为了能够正确链接，添加了nask编译后的函数文件。

# 写在最后

今天在搞这个玩意的时候跟一位使用OSX的同学进行了交流，他尝试使用nasm来替代nask。他进行了一些语法上的对nasm的适应性改动，但是他的ipl仍然不能正常运行。我将我们俩编译后的ipl进行了对比，发现第一个不一样的汇编指令是`add ax, 0x20`。经过更多的资料搜索发现，nasm和nask并不兼容。

但是nask真的好难用啊，有语法错误只会提示错误的总数，不会提示你具体哪一行出错了