# 操作系统原理实验报告

# 实验项目一：接管裸机的控制权之自定义引导程序的实现


院 (系) 名 称：　数据科学与计算机学院

专 业 名 称：　计算机科学与技术

学 生 姓 名：　王永锋

学 生 学 号：　16337237

指 导 教 师：　凌应标


二〇一八年三月十日

# 目　录

# 1　实验目的及要求

## 1.1　实验目的

1. 学会使用虚拟机启动 DOS 引导盘，并使用 DOS 引导盘运行 16 位的汇编程序。
2. 学会使用 winhex 修改磁盘镜像文件。
3. 学会使用 nasm 编译 x86 汇编程序。
4. 能够自己独立编写引导程序，并能引导虚拟机正常启动。

## 1.2　实验要求

本次实验，要求需要完成以下目标：

- **搭建和应用实验环境**
  虚拟机安装，生成一个基本配置的虚拟机 PC 和多个 1.44MB 容量的虚拟软盘，将其中一个虚拟软盘用 DOS 格式化为 DOS 引导盘，用 WinHex 工具将其中一个虚拟软盘的首扇区填满你的个人信息。
- **接管裸机的控制权**
  设计 IBM_PC 的一个引导扇区程序，程序功能是：用字符‘A’从屏幕左边某行位置 45 度角下斜射出，保持一个可观察的适当速度直线运动，碰到屏幕的边后产生反射，改变方向运动，如此类推，不断运动；在此基础上，增加你的个性扩展，如同时控制两个运动的轨迹，或炫酷动态变色，个性画面，如此等等，自由不限。还要在屏幕某个区域特别的方式显示你的学号姓名等个人信息。将这个程序的机器码放进放进第三张虚拟软盘的首扇区，并用此软盘引导你的 PC，直到成功。

# 2　实验 1 方案：搭建和应用实验环境

## 2.1　实验工具和环境

本次实验平台 (部分参考 [4] 搭建在 win10 系统上，通过编写 shell 脚本，连接 nasm 编译工具，dd 二进制文件覆写工具，与 bochs 虚拟机加载配置文件与镜像（具体工具链详见下表 2.1，实现了**编写代码后一键启动虚拟机查看结果**。同时，使用 bochs 虚拟机中提供的 bochsdbg.exe 工具，还能够对产生的镜像文件进行**单步调试**（相关调试样例可见 TODO: 在写实验 2 方案的时候补上），不仅大大提高了编写汇编文件后查看实现效果的速度，在命令行中通过发送指令对虚拟机的运行进行单步调试，查看各寄存器和内存，能够迅速定位操作系统中存在的问题并且解决它，从而提高我们的开发效率。

**表 2.1　本实验所使用的工具链**

| 软件名称 | 用途 |
|---|---|
| bash | 一个命令行终端，可提供 linux 的一些命令与执行 shell 脚本 |
| nasm | 将 x86 汇编文件编译成.bin 二进制文件 |
| dd | 将二进制文件的内容写进软盘镜像中 |
| winhex | 以二进制形式查看及修改软盘镜像文件 |
| bochs | 虚拟机，用于加载装有自定义引导程序的软盘 |
| bochsdbg | 调试工具，用于给装有自定义引导程序的软盘文件进行调试 |

## 2.2　制作 DOS 引导盘并启动

### 2.2.1　操作步骤

**步骤 1** 从 bochs 官网上，找到了 FreeDOS 的镜像文件。
**步骤 2** 修改 bochs 虚拟机的配置文件，使其指向刚下载的镜像文件。
**步骤 3** 使用 bochs 打开 FreeDOS，并进行简单的使用与实验。

### 2.2.2　操作结果

操作结果，详见图 2.1。

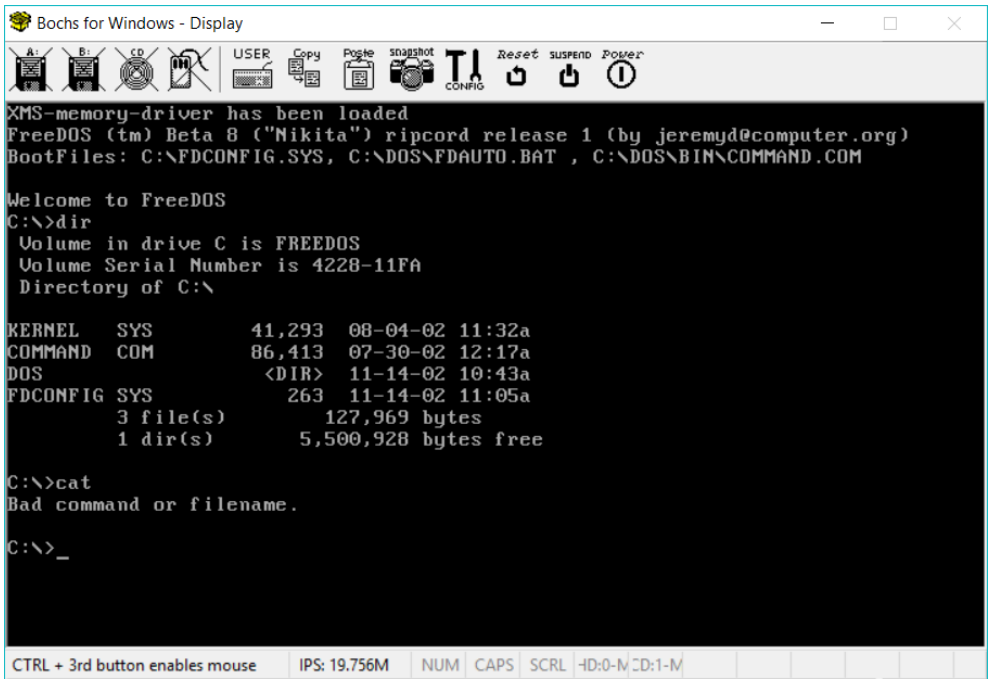


图 2.1　使用 bochs 加载 freeDOS 镜像的截图

## 2.3　使用个人信息填满引导扇区

### 2.3.1　操作步骤

**步骤 1：编写代码，生成填满个人信息的二进制文件。** 考虑到个人信息需要填满软盘第一个扇区的前 512 个字节，手工复制粘贴费时费力，在这里我采用 times 语句，将我的个人信息复制 100 次之后生成二进制文件。代码如下所示。

```
times 100 dw '16337237 wang yong feng'
```

**步骤 2：使用 dd 工具，将生成的二进制文件写到软盘镜像中。** 在命令行下，可以使用这样的命令进行二进制文件的覆写。

```
dd if=a.bin of=a.img bs=512 count=1 conv=notrunc
```

**步骤 3** 使用 WinHex 工具，查看修改后的二进制文件是否已经被成功覆写，并确认覆写结果是否正确。

### 2.3.2　操作结果

个人信息“16337237 wang yong feng”成功填充到软盘镜像文件中，如图 2.2所示

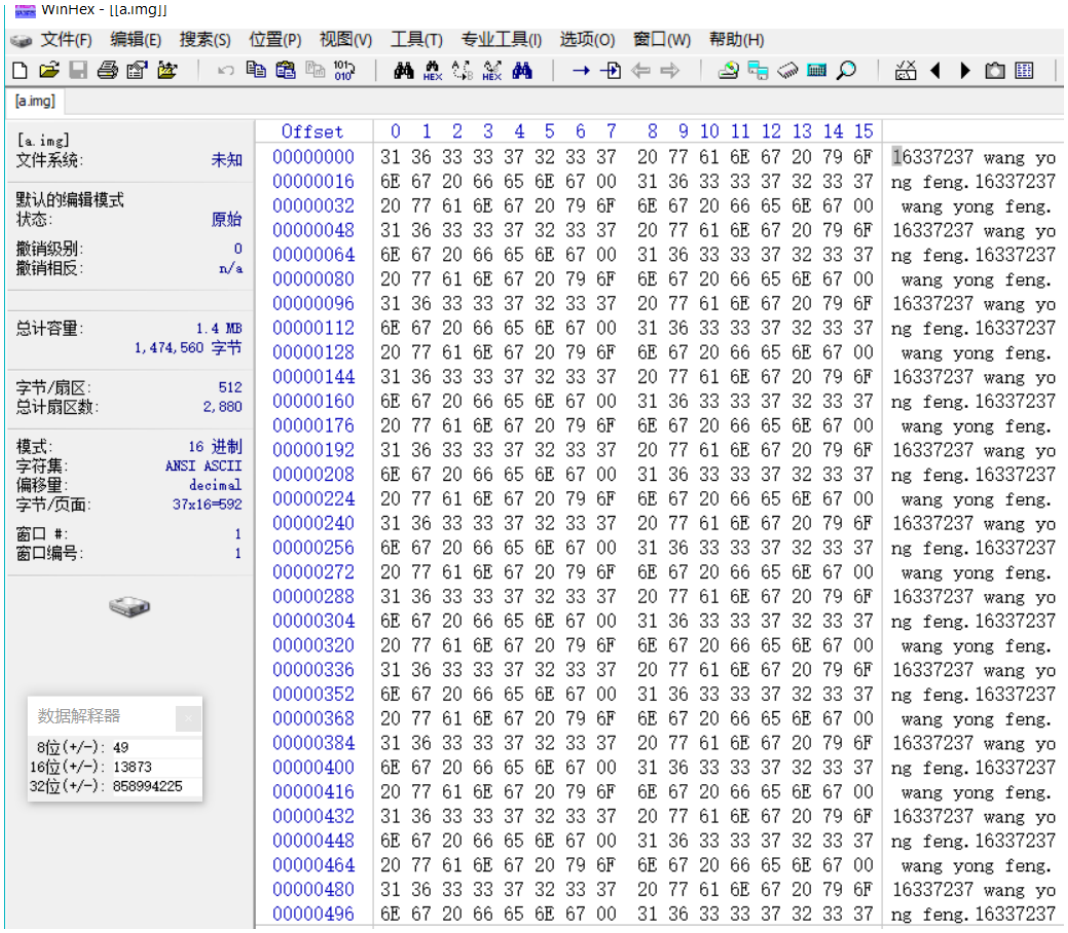


图 2.2　使用 winhex 查看填充个人信息后的软盘镜像第一扇区

# 3 实验 2 方案: 自定义引导程序的实现

## 3.1 相关基础原理

本次实验能够成功进行，离不开这些基础原理的支撑。

### 3.1.1 电脑开机流程与主引导扇区

计算机开机后，访问硬盘时所必须要读取的首个扇区称为主引导扇区 [3]。在 BIOS 的调控下，计算机会寻找到在第 511 和 512 个字节处为 55aa 的磁盘，并认定该盘为引导盘，从而将该盘的第一个扇区加载进内存的 0000:7c00 处，并将 CPU 的控制权交给位于内存地址 0000:7c00 处的指令，这就是我们这一次实验要实现的位于引导盘主引导扇区中的引导程序。

因此，加载进内存的位置由于历史遗留问题，固定在了 0000:7c00 处，因此，我们的汇编代码中，必须加上如下的一条：

```
1 org 7c00h
```

来确保汇编代码中的绝对地址在运行时仍然保持正确。

### 3.1.2 显存与显示模式

显卡都有自己的显示存储器，我们要控制显示输出，只需要往显存写入响应的数据即可。对于显存的读写交互，系统将显示存储器映射到了内存中的 0000:B800:0000:FFFF 中，因此，我们只需要对内存中的这一段地址进行操作，即可控制显示的输出 [5] 。如图 3.1所示。

对显存进行操作的时候，我们还需要知道，在文本模式下，能够显示的字符有 80*25 个，并且，每一个字符的显示，都需要操作两个字节。第一个字节为该字符的 ascii 码，而第二个字符为该字符的显示格式。显示格式相关资料可在网上或书中找到，此处不多加赘述。

### 3.1.3 几个在汇编程序中用到的中断

在本实验中，有两个软中断的作用显得尤为关键：

- **int 16h** : 用于控制键盘操作。
- **int 13h** : 用于将磁盘内容写进内存。

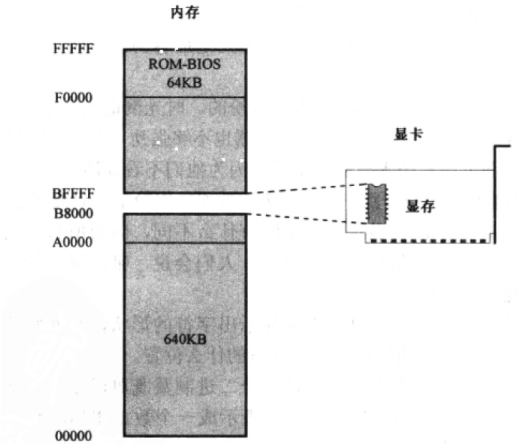


图 3.1　文本模式下 x 显存到内存的映射

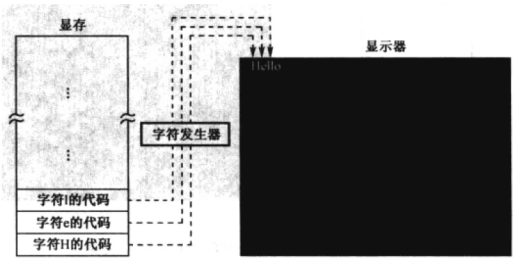


图 3.2　字符在屏幕上的显示原理

以下是相关中断的详细说明

1. **int 16h 0 号中断**可从键盘缓冲区中读取一个按键信号，其中扫描法放在 ah，按键对应 ascII 码放在 al。在这一次实验中，主要就是用 int 16h 0 号中断，从 al 中读取按键对应的 ascII 码，从而实现根据按键选择功能的效果。
2. **int 16h 1 号中断**不断的检测键盘缓冲区。一旦检测到键盘缓冲区有输入，会将标志寄存器中的 zf 置为 1。在本次实验中，就是通过一个循环不停地检测键盘缓冲区，来实现对键盘操作的实时回应。
3. **int 13h 2 号中断**该中断可以将磁盘的某一个或几个扇区加载进内存的一个特定的位置中。该终端所需参数比较多，可见表 3.1。

## 3.2　实验工具和环境

此处不赘述，可见前文表 2.1。

**表 3.1　INT 13h 2 号中断所需参数列表**

| 参数对应寄存器 | 相应应该放入的参数类型 |
|---|---|
| AH | 中断号 02h |
| AL | 一次中断读取的扇区数目 |
| CH | 起始磁道 |
| CL | 起始扇区 |
| DH | 磁头号 |
| DL | 驱动器号 |
| ES:BX | 写进内存的地址 |

## 3.3　程序功能说明

本程序（完整程序代码见附件）在主引导扇区及第二第三扇区中编写的程序实现了以下功能（主界面可见图 4.1）。

1. 有一个跳动的颜色会变的笑脸会在屏幕上跳来跳去，一旦遇到边缘则反弹。
2. 右下角有我自己的学号和姓名的拼音。
3. 键盘按下 C，可实现清屏（不清学号和姓名）
4. 键盘按下 D，可通过字符画的形式在 80*25 有限的空间内展示我名字的汉字形式。可见图 4.2

# 4　实验结果

本程序经 nasm 编译后在 bochs 虚拟机下运行的效果可见图 4.1与图 4.2

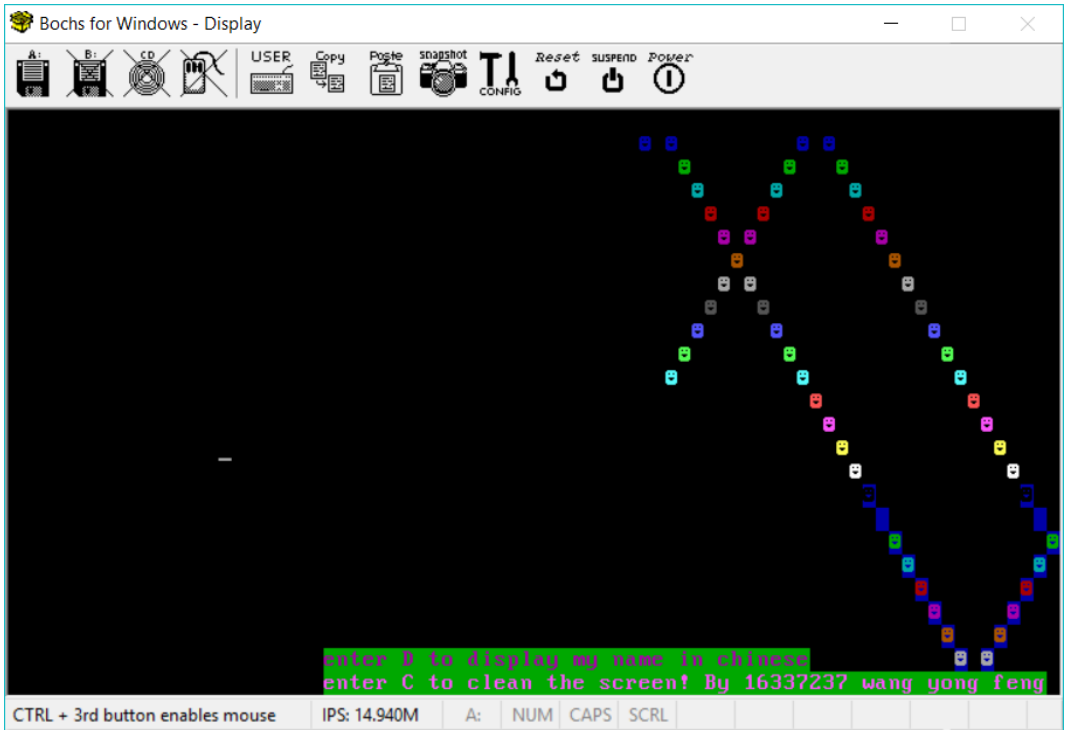


图 4.1　自定义引导程序的主界面

通过完成本实验，我们完成了以下工作，验证了以下结论：

1. 搭建了一套完整的，支持 32 位汇编代码编写的，且高效率的操作系统开发环境，并在实验的过程中进一步熟悉了该开发环境与调试工具的使用。
2. 验证了书本上所说的电脑开机时所做的一部分工作，如选择引导盘是通过选择第一个扇区末尾为 0x55aa 来判断的，如会将主引导扇区加载到内存地址为 0000:7c00 的位置中，通过自己的探索，也找到了通过主引导扇区中的引导程序，加载在磁盘其他位置的系统内核的方法。

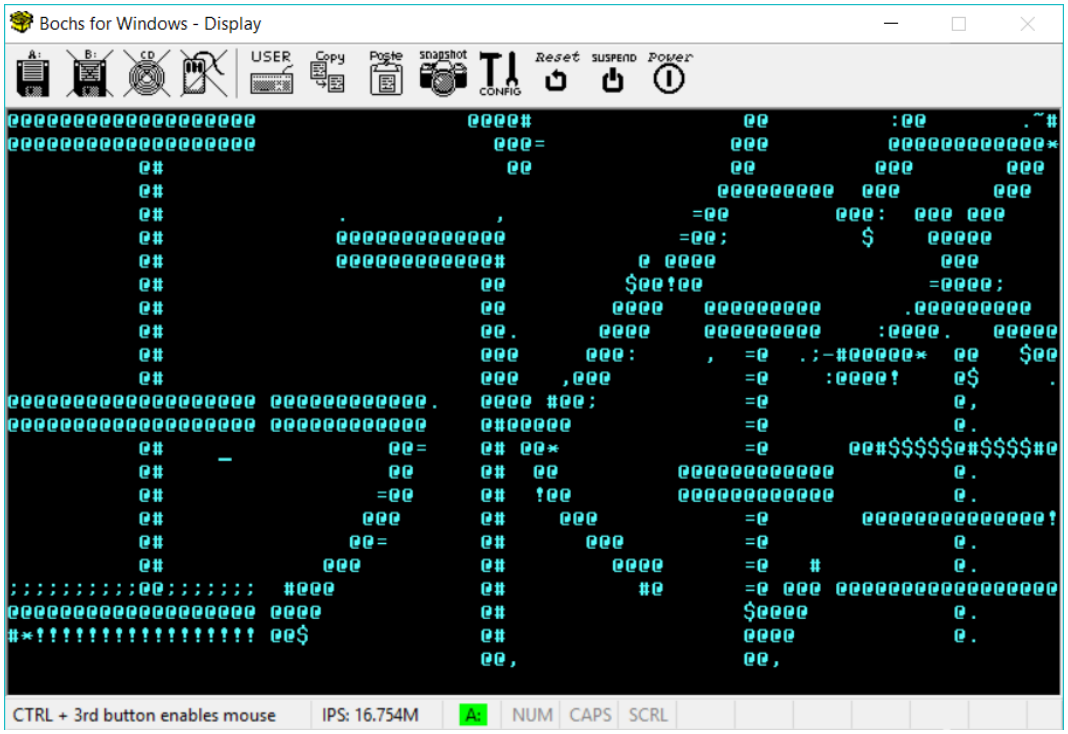


图 4.2　自定义引导程序按下 D 后的界面

# 5　实验总结

这是第一次操作系统实验，也是我第一次使用汇编语言直接对硬件进行操作（虽然是虚拟的硬件）。其实，我们现在对操作系统的学习的门槛已经降低了很多，这首先要归功于在主板上已经内置写好的 BIOS 引导程序，有了这个引导程序，才能够进一步将 CPU 的控制权交给到我们能够直接编程的引导盘的主引导扇区，我们才能够做到“接管裸机的控制权”。

由于有过一些 x86 汇编语言的基础，在实验刚开始进行的时候其实问题还不太大，上学期的计算机组成原理课程中，我曾经使用 X86 汇编语言与 masn 汇编器，实现了一个很简单的小游戏，在这个过程中，对中断的理解也加深了很多。也正是有过之前的经历，这一次才能够大胆地将中断用在引导程序中。

但是说这个实验很简单，那暂时还是不可能的。由于平时用汇编语言还比较少，加上这一次使用的工具是 nasm，一个更为先进也更为优良的汇编器，与我原本用的 masm 在语法上有一点点区别，我还是花了一些时间取看 nasm 的官方文档，来去尽快的适应新的汇编器。值得庆幸的是，nasm 的官方文档里面就有一节就是写给用过 masm 的人，为了他们更快的适应 nasm。由此可见，在不熟悉的工具面前，官方的文档比起网上的各种博客而言，往往有效而又全面得多。

但是这个实验的成功，还得归功于我相对比较好的资料检索能力与独立解决问题的能力。512 个字节的限制，应该说是阻挡了大多数有意思的想法，一开始的我甚至想着就简简单单的完成实验基本要求就好。还好后来，自己的好奇推动了自己进一步往前探索：“肯定有办法加载其他扇区的内容啊，不然操作系统的内核怎么加载进来的?”，在网上找了很多理论上的资料都没有讲到这方面的解决方法，后来终于发现，是自己搜索问题的关键词不太恰当。换了一下关键词，很快就找到了自己的问题的解决方法。

还有一点想说的是，在开发操作系统的过程上，开发效率是非常重要的一点。影响开发效率有两个重要的因素，一个是修改代码后显示效果的时间，另一个是遇到问题后定位问题的途径多少。

为了能够达到修改代码后能够一键打开虚拟机查看结果，我在自己原有的一点 shell 编程经验的基础上，完成了一个比较简单的脚本用于一键打开虚拟机查看结果。有时候我看到有同学先输入命令行编译，然后打开 winhex 复制粘贴二进制代码，然后另存为镜像文件，然后打开 vmware 虚拟机跑，这一系列的过程需要比较多繁琐的键盘和鼠标的操作。虽然说这些操作也并不是那么繁杂，但是当调试达到一定次数，这种操作的烦躁确乎会影响自己学习的积极性。因此，想了一下，

这一套方案还是有必要向同学们推广一下。

遇到问题，是一件很常见的事情，但是我们有多少种方式可以去定位问题的所在呢？光靠编译输出查看结果的确是一个途径，但是这个途径能够提供给我们的信息实在是太少了，我们只能够看到显示在显示屏的显存的情况，但是对 CPU 的各种内部的参数，却完全没有任何获知的渠道。此时，一个类似于 gdb 的调试工具就显得尤为必要。因此，我选择 bochs 虚拟机的另外一个理由也是因为它为我们提供了一个堪比 gdb 般好用的 bochsdbg.exe。这一次实验的完成，中间很多问题的定位，都是多亏了这个工具给我的信息（包括查看各种内部寄存器，内存内容）我才能够准确地定位问题。同时也用这一套工具，帮了个别同学检查他们代码中的问题。

这一次实验，还是给了我不少收获的，在做实验的中间，也查找了不少资料，大体想来，对操作系统原理的理解又深了一些。下一次继续加油吧。

# 6 实验难点及亮点

## 6.1 突破主引导扇区 512 字节的限制

在设计功能并且尝试实现的过程中，我发现一旦我写的代码多了一点，就很容易导致代码长度超出 512 个字节，进而导致我的代码不能够被正确的执行。这是一个很严重的问题，这导致我不能够自由的添加我想要实现的功能。经过上网搜索，在网友的 CSDN 博客 [1] 中，找到了相关的解决方法, 我们可以使用 x86 中自带的 int 13h 中断 [2] 中的 2 号中断，将第二个扇区及之后的数据加载进原来代码段之后的那一段内存中。相关参数说明也可在小小节 3.1.3中找到。以下为我在引导扇区中编写的加载第 2 8 个扇区的代码。

```
load_os:
  mov ah,02h                    ;读磁盘扇区
  mov al,07h                    ;读取7个扇区
  mov ch,00h                    ;起始磁道
  mov cl,02h                    ;起始扇区
  mov dh,00h                    ;磁头号
  mov dl,00h                    ;驱动器号
  mov bx,os                     ;os为标签，地址为第二扇区的
    第一个字节
  int 13h
  ret
```

在调用该过程之后，在第一个扇区之后的代码就可以加载进内存，从而顺利执行，突破了 512 字节的限制，而这与引导程序加载系统内核的原理相同。

## 6.2 读取键盘的操作

我设计的功能中，涉及到与键盘之间的交互，比如通过按 C 来进行清屏。读取键盘的功能，是通过 int16h 中断实现的。但是这里有个问题，我希望我写的引导程序能够时时刻刻都能够响应我对键盘的操作，但是引导程序又不能够停止运行，也就是说，**不能够让读取键盘的操作阻塞主界面中笑脸的移动。**

为了解决这个问题，我在编写代码的时候，设计了一个循环用于检测是否有操作键盘，在检测的时候还要去执行响应的笑脸移动过程 (loop1)，以及显示学号姓名过程 (display_infomation)。实现的代码如下：

```
check_keyboard:
  call loop1
  call display_infomation
  call display_message
  mov ah, 01
  int 16h
  ; 如果没有按， zf为0P
  ; 如果有按，往下执行
  jz check_keyboard

  ; 当检测到按键，判断按键是否需要响应，以及决定响应的功能
  ; 从键盘读入字符,扫描码读进ah,asciI码读进al
  mov ah, 00h
  int 16h
  ; 判断字符
  cmp al, 'c'
  mov word [status], 0720h
  jnz check_keyboard_d
  call clean_screen
  jmp check_keyboard
  check_keyboard_d:
  cmp al, 'd'
  jnz check_keyboard_h
  call clean_screen
  call display_chinese_name
  jmp check_keyboard
  check_keyboard_h:
  jmp check_keyboard
```

## 6.3 源程序中的一处不完善的地方

老师给的源代码实现了一个简单的 asiII 字符的移动与遇墙反弹，但是在程序的开发调试过程中，我发现了这个程序有一个比较严重的漏洞。当该字符移动到墙角的时候，字符所在的位置可能会越界，这会导致字符的 x,y 坐标与字符实际处在的位置不相符合，原先遇墙反弹的效果也会消失，变成了隔一段时间字符出现一次。

# 参考文献


[1] 从 boot 读取软盘扇区中的汇编 - CSDN 博客.

[2] W. contributors. Int 13h — wikipedia, the free encyclopedia, 2017. [Online; accessed 11-March-2018].

[3] W. contributors. 主引导记录 — 维基百科, 自由的百科全书, 2018. [Online; accessed 11-March-2018].

[4] 于渊. *Orange'S:* 一个操作系统的实现. 电子工业出版社, 2009.

[5] 李忠. *x86* 汇编语言从实模式到保护模式. 电子工业出版社, 2013.

# 附录 A　运行汇编文件的脚本

这是我为了方便，在 win10 平台下的 bash 终端编写的 shell 脚本。

```
name=$1
nasm -f bin -l $1.list $1.asm -o $1.bin
# add the feature : if the image not exist, create the image
if [ ! -f "a.img" ]
then
    echo "a.img not exist!"
    echo "use bximage to create a image";
    exit
else
    echo "a.img exists !"
fi
dd if=$1.bin of=a.img bs=512 count=1 conv=notrunc
if [ ! -f "bochsrc.bxrc" ]
then
    echo "bochsrc.bxrc not exist!"
    echo "copy a bochsrc.bxrc to this path quickly";
    exit
else
    echo "bochsrc.bxrc exists !"
fi
bochs -q
```