# 实验一实验报告

## 【个人信息】

院系：数据科学与计算机学院

专业：计算机科学与技术（超算方向）

班级：教务2班

姓名：劳马东

学号：16337113

邮箱：laomd@mail2.sysu.edu.cn

## 【实验题目】

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XXXPC，直到成功。

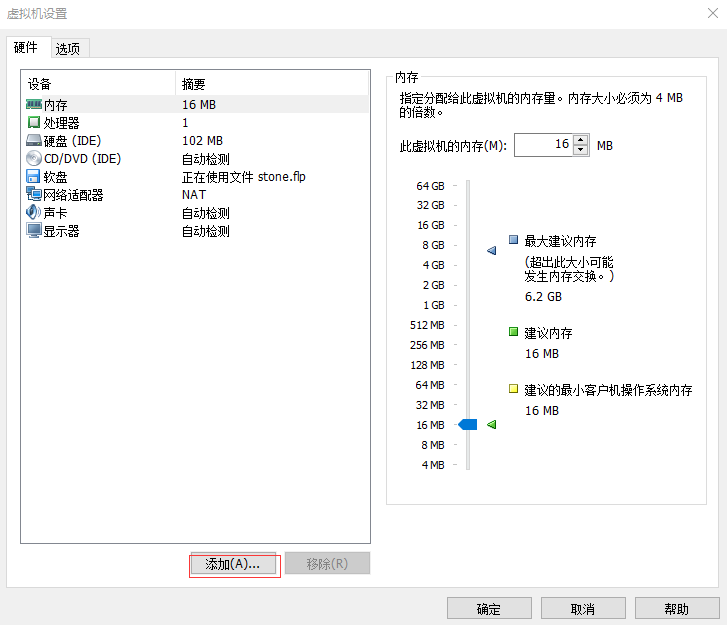
## 【实验目的】

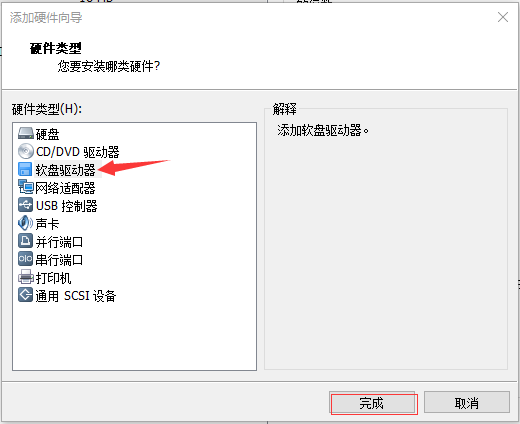
1. 熟悉GCC+NASM工具组合的使用；
2. 熟悉x86汇编语言语法；
3. 熟悉在裸机上的编程；

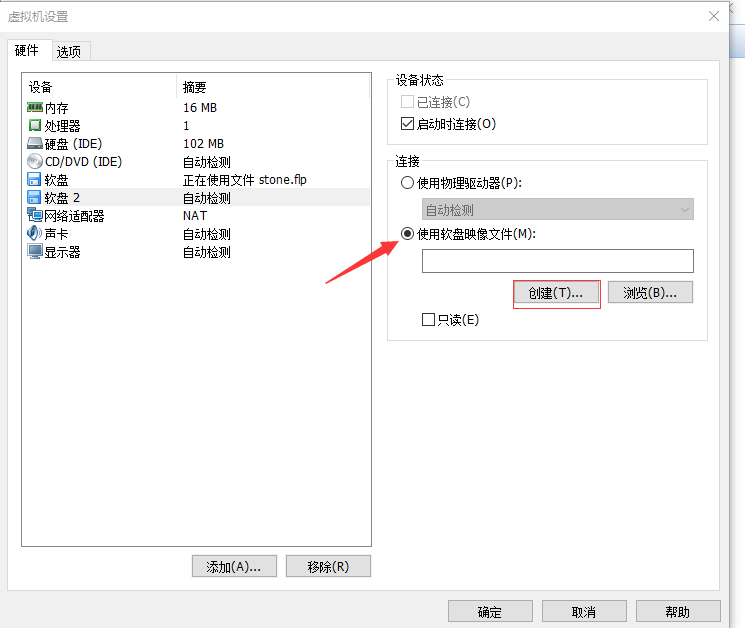
## 【实验方案】

1. **虚拟机配置方法**

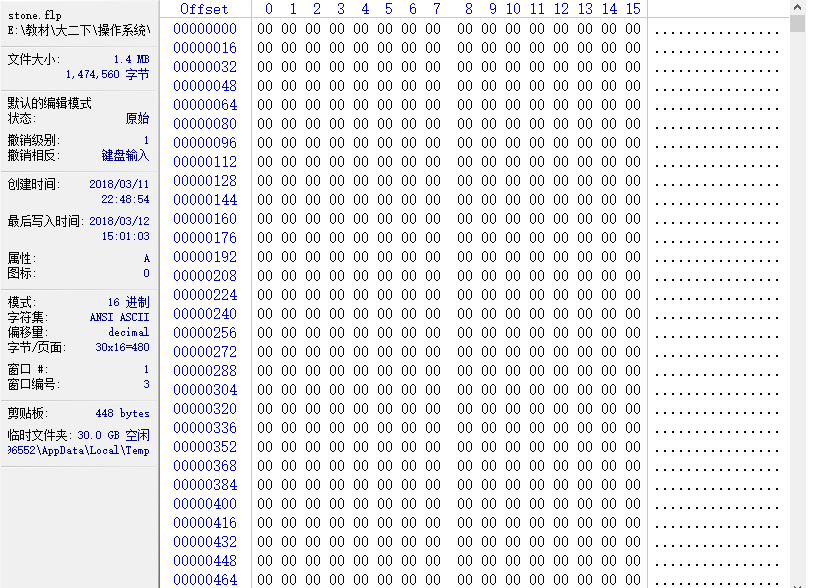
虚拟机的创建，在选择操作系统时候选择“其他-DOS”，最大内存大小改为0.1GB，弄成单个文件，以下是创建软盘的方法：







创建好的软盘用winHex打开查看是全0的初始状态，大小在1.4MB左右。



1. **软件工具与作用**
2. **实验支撑环境**

硬件：个人计算机

主机操作系统：Windows

虚拟机软件：VMware，创建虚拟机和引导软盘，运行虚拟机。

1. DOS虚拟机
2. **实验开发工具**

汇编语言工具：x86汇编语言

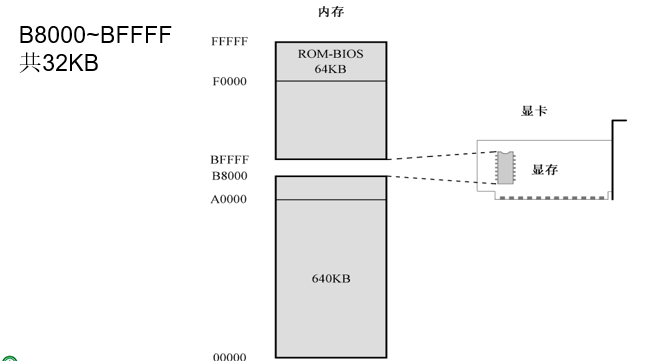
高级语言工具：标准c语言

磁盘映像文件浏览编辑工具WinHex：读写com可执行程序或软盘文件

调试工具：Bochs

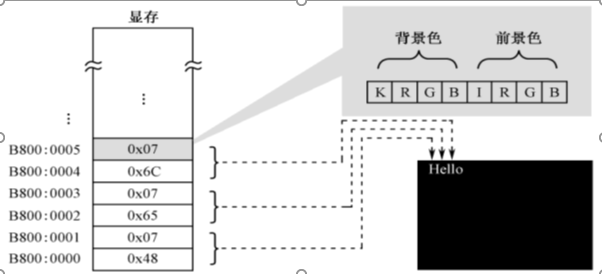
1. **实验原理**
2. **引导程序原理**

当BIOS启动时，会将启动设备的第一个扇区（通常是512B大小）加载到内存的0x07c0处，并开始执行。另外，对于8086机器，内存的0xb800到0xbffff的地址被映射到显卡的显存中，因而，如果想要通过显卡来在屏幕上显示字符的话，必须将字符写到这块内存中。需要注意的是，BIOS在启动首扇区时，会检查其代码格式，只有当最后两个字节为0x55、0xAA时，才是符合要求的引导程序。



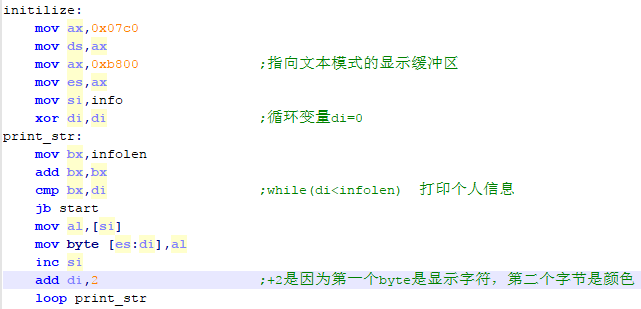
1. **字符显示、前后景颜色和闪烁原理**

对于两个相邻的字节，第一个字节是字符本身，把这个字符的AscaII码存到显存的相应位置，就能显示字符；第二个字节是字符属性，控制字符的前后景颜色、是否闪烁。如果这个字节的八位假设为KRGBIRGB，通过设置这八位的01值，就可以控制这些属性，具体01与属性的关系如下图：





1. **程序关键模块**
   1. 打印个人信息



C:\Users\96552\AppData\Roaming\Tencent\Users\965524991\TIM\WinTemp\RichOle\(VI_)BB0@AVHNJS@]_FUID4.png

首先初始化段寄存器ds和cs（由于不能将立即数直接赋值给段寄存器，也出于安全考虑，需要先把立即数赋给ax，然后通过ax传给段寄存器），把字符串首地址给si，循环变量初始化为0，然后进入循环。当循环变量小于字符串长度的时候，不断打印字符串中的字符。如上所说，一个字符的显示和属性需要两个字节，因此每次循环时，di实际上应该加2，这也是为了以后的可扩展性考虑，比如以上代码还没有设置字符的属性，假如以后要配置属性，只需要加上一行如“mov byte [es:di+1],7”。

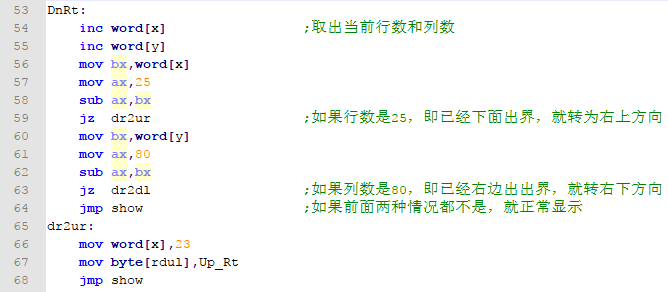
* 1. 主循环





将rdul逐个与四个方向（实际上定义为四个数字1、2、3、4），如果相等（即结果为0），就执行相应函数，往相应方向运动。

* 1. 运动和转向

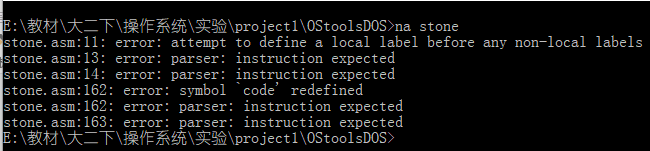


相关解释在注释中已有说明，其余方向的转向和运动类似。

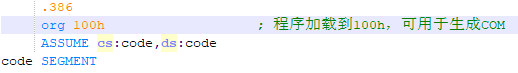
## 【实验过程】

1. **改正编译错误**

第一次用nasm编译器编译stone.asm时，发现有比较多的编译错误。如下图：



查看源程序文件的11、13、14行，如下图：



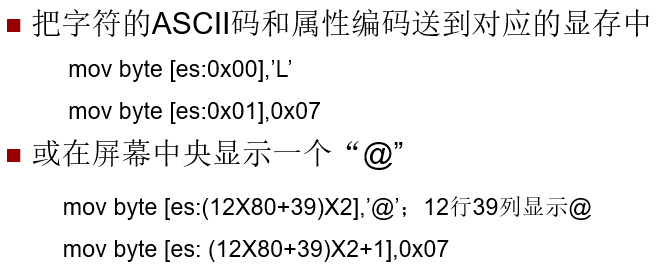
由于初学x86汇编（上学期的计算机组成原理学习的是MIPS汇编语言），对这些语法不熟悉，于是请教了15级的一个师姐，她说这些代码都是用于生成com文件的，在引导程序中不需要，可以注释掉，自己也上百度查了查每句话的作用，发现确实这么回事，于是就去掉了这几句以后后面的code ends、end start。去掉之后再编译，发现又出现了新的错误：

C:\Users\96552\AppData\Roaming\Tencent\Users\965524991\TIM\WinTemp\RichOle\1D999(}MGUJ%@)_`@$Q0)$A.png

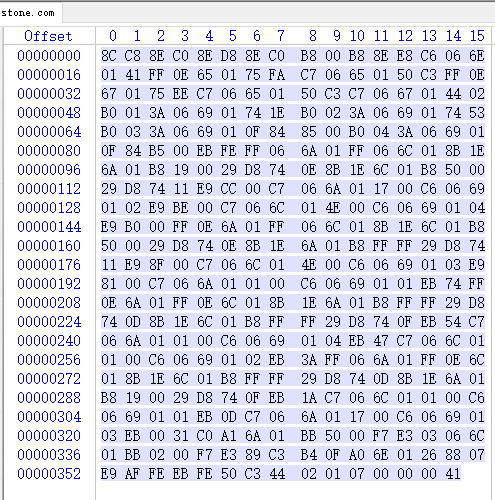
翻到149行，代码是这样的：

C:\Users\96552\AppData\Roaming\Tencent\Users\965524991\TIM\WinTemp\RichOle\S0V2FKP_B~$%RH~Y2Q7KGWH.png

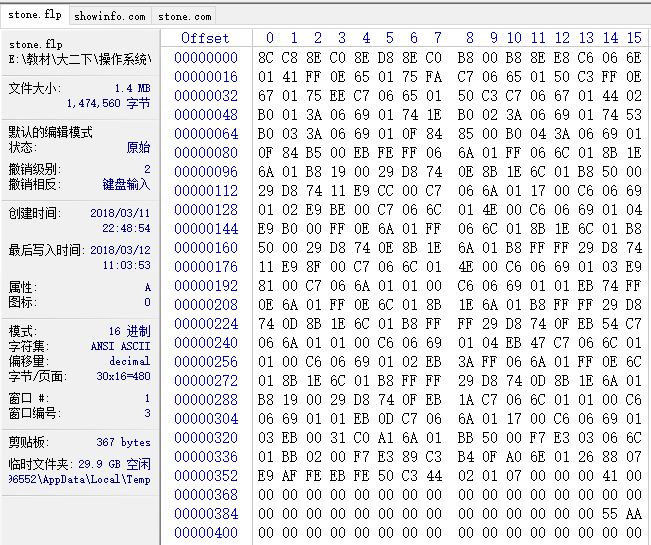
似曾相识！这不是老师在实验课上给我们展示的打印@的代码吗？可是又有点不一样，老师的代码是这样的：



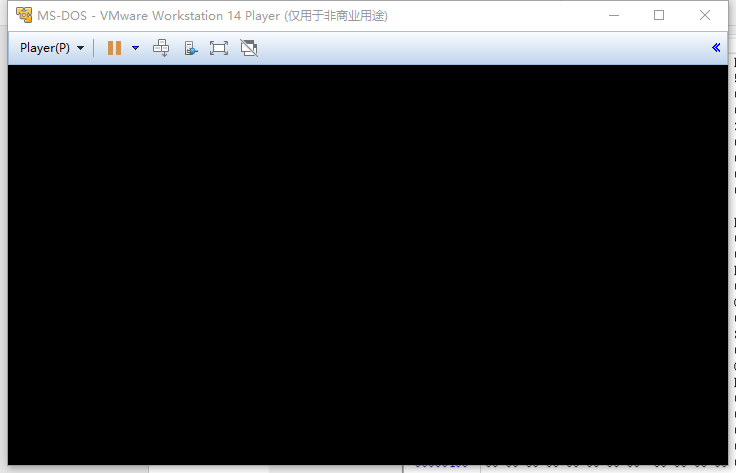
由上面一行的“mov ah,0FH”的注释可以知道，这一行的作用是调颜色和闪烁，相当于老师代码的第二个mov，因此问题必然出现在把字符送到显存的那一行！对比一看，原来有语法错误，访问显存应该使用“段地址：偏移地址”的形式，而且这部分用中括号括起来，而编译错误那一行把段地址放在中括号外了，而且送的字符不应该是ax，而是al。改正之后，编译通过！编译之后生成的com文件用winHex查看，如下:



复制到软盘文件中，在末尾加上55、AA：



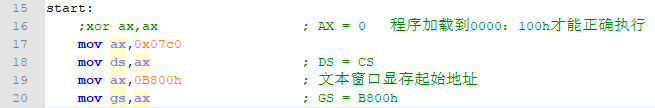
运行虚拟机：



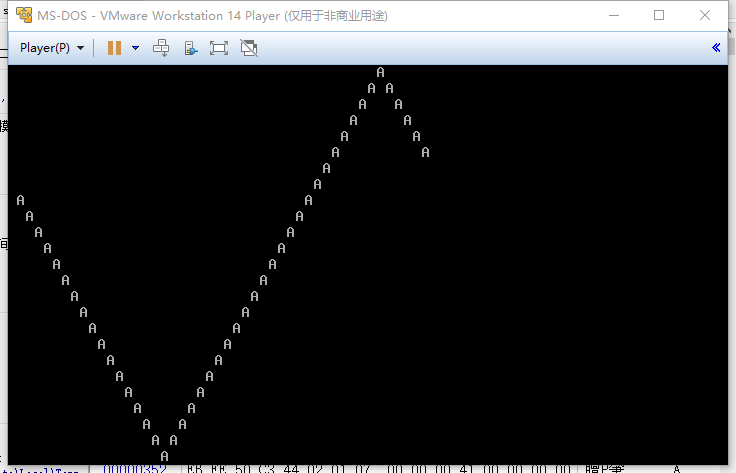
什么都没有！问题出在哪里？

C:\Users\96552\AppData\Roaming\Tencent\Users\965524991\TIM\WinTemp\RichOle\NPR3)7TOK6{XKG~)$I1%`BT.png

由于这里是把0b800h赋值给gs段寄存器，而下面用的是es寄存器，显存映射错误，而且没有把程序加载到内存的0x07c0处，当然无法显示，改成这样：

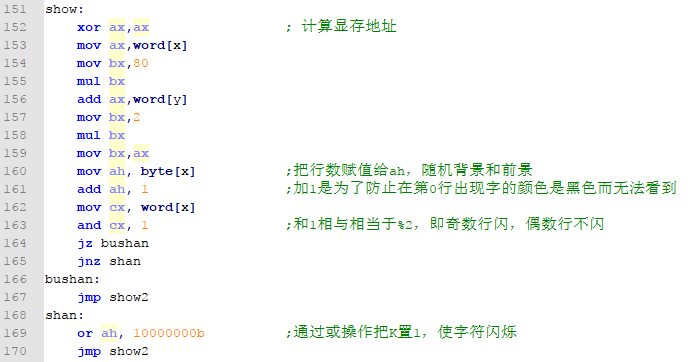


就可以正常运行了：



1. **字符A按层闪烁和变色**

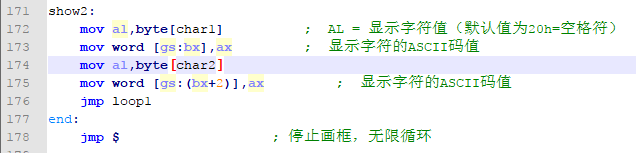
查看老师提供的源程序可以发现，显示字符的代码逻辑大部分在show函数中，而且通过注释掉“mov ah”这一行发现，运行之后颜色改变，说明ah寄存器是控制颜色和闪烁的。于是就写了一个类似于if-else的逻辑：



相关语句的作用在注释中已经说明得比较清楚。

1. **同时控制两个字符**

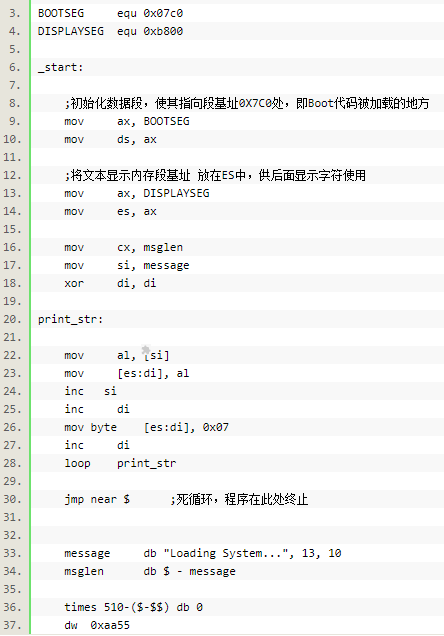
想法是通过在A字符的后一列再控制一个字符，因此在把字符A送到缓存之后，再次重复操作，就可以把字符B“同时”送到显存，这样就在屏幕上显示了两个字符。



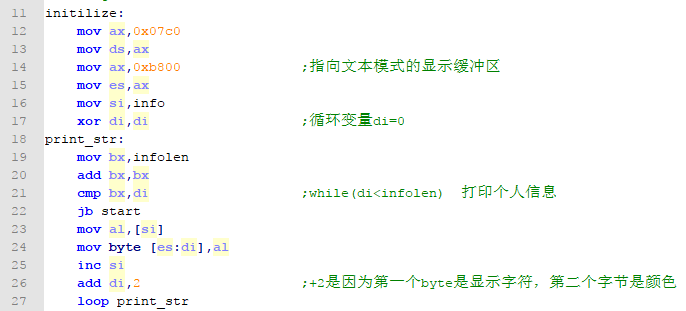
C:\Users\96552\AppData\Roaming\Tencent\Users\965524991\TIM\WinTemp\RichOle\333U_LJB1]Q9(02VIH(%NVG.png

1. **打印个人信息**

一开始是采用硬编码的形式，即用很多条类似“mov byte [es:0],’1’”、“mov byte [es:1],7”这样的语句来打印个人信息，但这样带来的问题：一是可扩展性极差，如果以后要显示其他字符串，又要重复一遍这个繁琐的操作；二是容易把源程序文件弄得很大（不能超过512B），实际上在做了以上的功能之后，再采用这种方式，文件已经比512B大得多了，这样一来就没法添加其他代码。于是在百度上搜索NASM的x86汇编语言怎么输出字符串，CSDN找到的示例程序如下：

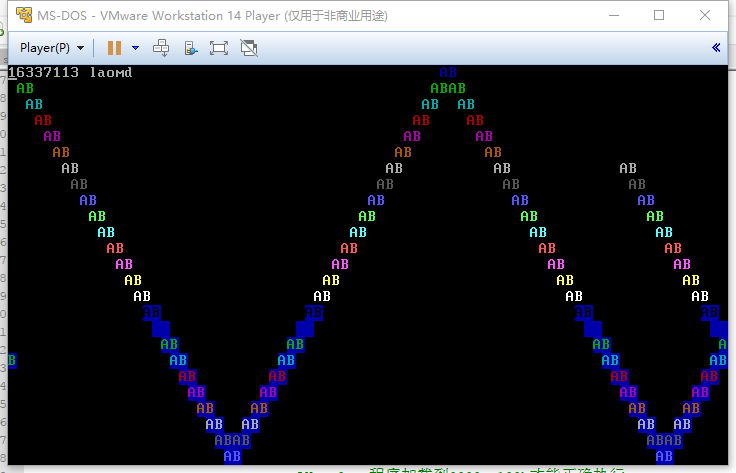


编译运行一遍发现没什么问题，于是模仿着写了一段。由于这个示例程序没有循环停止的逻辑，而我们的程序需要在打印个人信息之后转而执行字符A弹射的代码，因此就对这段代码自己做了一些修改，如下图：



以上代码相当于一个for或者while循环，进入循环之前先判断循环变量是否超出范围，如果每有，就打印对应位置的字符，否则退出循环，转而执行start。具体逻辑在注释以及上面的关键模块中已有说明。

程序最终的结构如下图：



可以看到，屏幕左上角显示了个人的学号姓名拼音，其余部分字符AB同时并排弹射，而且在不同的行上，字符的颜色、背景等不一样。

## 【实验总结】

1. 一开始编译源代码文件的时候看到这么多的编译错误摸不着头脑，因为没怎么接触过x86汇编语言，对一些语法的报错信息没法很快明白，迅速找到解决方法。一开始是向高年级的同学请教，但是因为她也不是特别清楚程序的整体，就没法做到帮我解决遇到的所有问题。于是就把老师给的PPT看了一遍，写了一些简单的程序，通过所谓的“控制变量法”一行行地注释，看会出现什么结果，来猜出每行代码的作用，并通过百度搜索相关知识验证。个人建议老师以后在给代码的时候，如果的确是让我们来改正一些编译错误可以明说，因为不说容易让人误以为是工具不行或者平台不支持，这就容易浪费时间在排错上。就像这次实验，一开始大家在群里讨论为什么会出现编译错误，很多人都说是nasm编译不行，要该用masm，但是后来又有人反应masm编译也同样有语法错误。
2. 关于字体变色，一开始为了找到哪一行代码是调整颜色的，也是通过一行行注释，当然不是每一行都选，只是挑那些看起来像的，范围也大部分局限在show函数中。当注释掉“mov ah”那一行的时候，运行结果发生了惊人的变化！字符A在弹的过程中居然伴随着分层变色，可是自己还没写字体变色的逻辑呀！而且有一个很奇怪的现象，左半边的字符色层比右半边的低一层，左半边也没有运动到最顶端。于是认真分析了其余代码，发现并没有关于颜色变化的代码，只是一些撞到边缘后弹射的逻辑。后来也请教了其他同学，发现他们也有同样的现象，但是他们也没能回答上来为什么会这样，希望老师能在评语中给出解释，不胜感激！

## 【参考文献】

1. Irvine K R. Assembly Language for x86 Processors[M]. 清华大学出版社, 2011.
2. 实验一PPT
3. http://blog.csdn.net/sivolin/article/details/40859987

开机输出字符串 - sivolin的专栏 - CSDN博客