实验二：加载用户程序的监控程序

实验者：张海涛，张晗宇，张浩然，张镓伟
学号：15352405,15352406,15352407,15352408
院系：数据科学与计算机学院
专业：15 级软件工程（移动信息工程）
指导老师：凌应标

【实验题目】

加载用户程序的监控程序

【实验目的】

1.设计四个（或更多）有输出的用户可执行程序

2.设计加载程序的监控系统

【实验要求】

设计四个有输出的用户可执行程序，分别在屏幕 1/4 区域动态输出字符，如将用字符‘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，碰到屏幕相应 1/4 区域的边后产生反射，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。

修改参考原型代码，允许键盘输入，用于指定运行这四个有输出的用户可执行程序之一，要确保系统执行代码不超过 512 字节，以便放在引导扇区

自行组织映像盘的空间存放四个用户可执行程序

【实验方案】

1.虚拟机配置方法

无操作系统，10M硬盘，4MB内存，启动时连接软盘

2.软件工具和作用

Notepad++:用于生成.汇编语言文件

Nasm：用于编译.asm 类型的汇编语言文件，生成 bin 文件

Vmware Workstation 12Player: 用于创建虚拟机，模拟裸机环境

3.方案思想

1）用户可以在监控程序中通过不同的按键访问不同的可执行程序，分别在屏幕的四个角。

2）用户在可执行程序可以通过按键来访问其他可执行程序或者返回主程序。（按键正确跳转对应可执行程序，否则返回主程序）

4.实验原理

（1）在屏幕上显示文字

显示器

将那些内容以视觉可见的方式呈现在屏幕上。

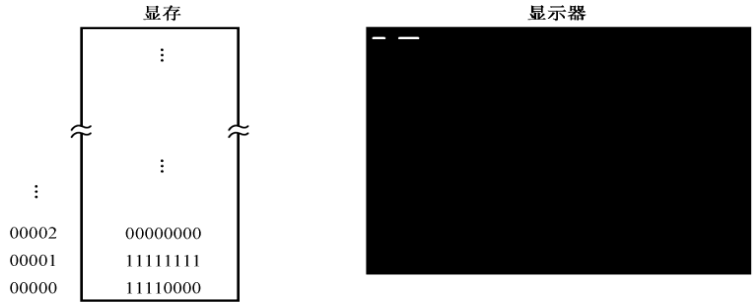
显示卡

为显示器提供内容，并控制显示器的显示模式和状态

图形方式：最小可控制单位为像素，VGA：640X400

文本方式：最小可控制单位为字符，VGA：25X80

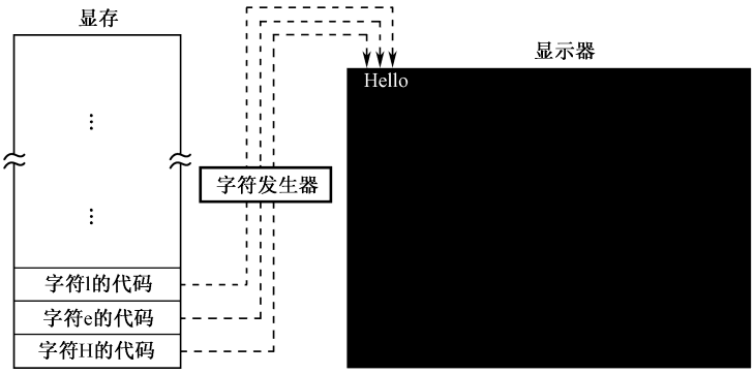
显示卡内存：存放像素或文字及相关属性



（2）字符显示原理

字符发生器和控制电路

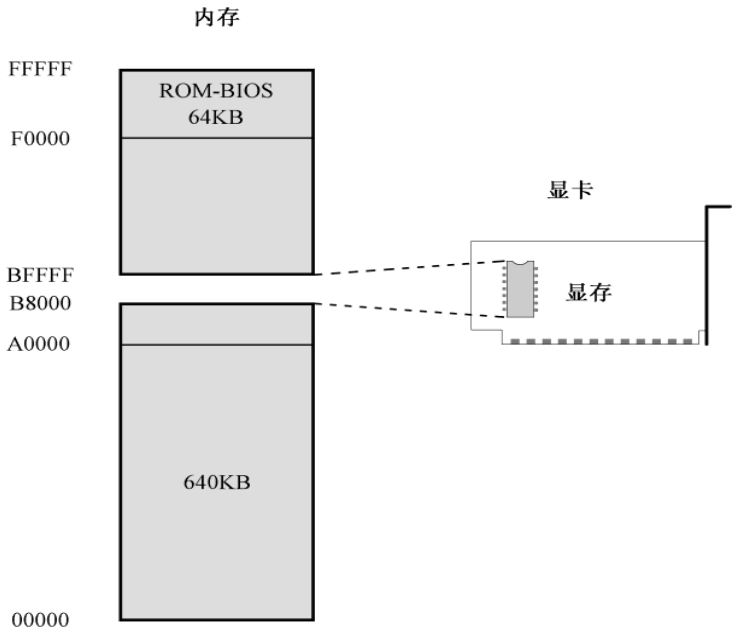
用代码来控制屏幕上的像素，使它们或明或暗以构成字符的轮廓



（3）字符方式显存地址空间

8086 可访问的内在 1MB

地址空间 B8000~BFFFF，共 32KB



（4）初始化段寄存器

访问显存使用逻辑地址

采用“段地址：偏移地址”的形式

显存段地址 B800

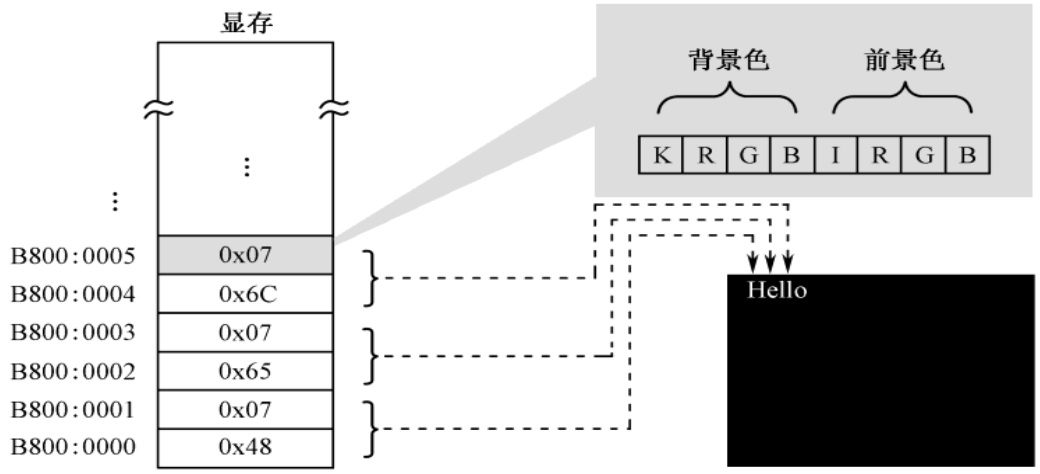
初始化段寄存器

规定：

不能将立即数直接传送到段寄存器

防止误操作

（5）显存与屏幕上字符的对应



（6）屏幕上字符的显示属性

R	G	B	背景色	前景色	
			K=0 时不闪烁，K=1 时闪烁	I=0	I=1
0	0	0	黑	黑	灰
0	0	1	蓝	蓝	浅蓝
0	1	0	绿	绿	浅绿
0	1	1	青	青	浅青
1	0	0	红	红	浅红
1	0	1	品（洋）红	品（洋）红	浅品（洋）红
1	1	0	棕	棕	黄
1	1	1	白	白	亮白

（7）显示字符

方法一：把字符的 ASC 码和属性编码送到对应的显存中

方法二：以电传方式显示单个字符（从当前光标位置开始，且更新当前光标位置）：

mov ah,0e ； 功能号

mov al,字符 ；ASCII 码

mov bl,0 ; BL 必须设为 0

int 10H ； 调用中断

读按键

返回值：AL=ASCII 码， AH=扩展码/扫描码

mov ah,0 ； 功能号

int 16H ； 调用中断

（8）X86 汇编语言语法

（9）显示字符串

mov ah,13H ； 功能号

mov al,1 ； 光标方式（光标放到串尾）

mov bl,属性 ； 背景与前景色

mov bh,0 ； 显示页号（第 0 页）

mov dh,行号

mov dl,列号

mov es,字符串段地址

mov bp,字符串偏移地址

mov cx,串长

int 10H ； 调用中断

（10）BIOS 调用

BIOS 是英文"Basic Input Output System"的缩略语，直译过来后中文名称就是"基本输入输出系统"。其实，它是一组固化到计算机内主板上一个 ROM 芯片上的程序，它保存着计算机最重要的基本输入输出的程序、系统设置信息、开机后自检程序和系统自启动程序。 其主要功能是为计算机提供最底层的、最直接的硬件设置和控制。

BIOS 芯片中主要存放：

自诊断程序：通过读取 CMOSRAM 中的内容识别硬件配置，并对其进行自检和初始化；

CMOS 设置程序：引导过程中，用特殊热键启动，进行设置后，存入 CMOS RAM 中；

系统自举装载程序：在自检成功后将磁盘相对 0 道 0 扇区上的引导程序装入内存，让其运行以装入 DOS 系统；

主要 I/O 设备的驱动程序和中断服务：由于 BIOS 直接和系统硬件资源打交道，因此总是针对某一类型的硬件系统，而各种硬件系统又各有不同，所以存在各种不同种类的 BIOS，随着硬件技术的发展，同一种 BIOS 也先后出现了不同的版本，新版本的 BIOS 比起老版本来说，功能更强。

BIOS 中中断例程即 BIOS 中断服务程序

是微机系统软、硬件之间的一个可编程接口，用于程序软件功能与微机硬件实现的衔接。DOS/Windows 操作系统对软、硬盘、光驱与键盘、显示器等外围设备的管理即建立在系统 BIOS 的基础上。程序员也可以通过 对 INT 5、INT 13 等终端的访问直接调用 BIOS 终端例程。

调用 BIOS 中断服务程序的方法

每个中断服务有特定的参数，一般使用指定的寄存器传递参数；

利用软中断指令调用

BIOS 中断调用的一般格式为：

mov ah,功能号

…… ；设置各种入口参数

int 中断号

常用 BIOS 调用：

功能	中断号	功能号
插入空行上滚显示页窗口	10H	06H
以电传方式显示单个字符	10H	0EH
显示字符串	10H	13H
复位磁盘系统	13H	00H
读扇区	13H	02H
读下一个按键	16H	00H

（11）BIOS 的 10H 调用

BIOS 的 10H 提供了显示字符串的调用

BIOS 的 10H 号调用功能与参数

显示字符串	10H	13H	AL：放置光标的方式 BL：字符属性字节 BH：显示页(0~3) DH：行位置(0~24) DL：列位置(0~79) CX：字符串的字节数 ES:BP：字符串的起始地址	AL=0/2 光标留在串头 AL=1/3 光标放到串尾 AL=0/1 串中只有字符 AL=2/3 串中字符和属性字节交替存储 BL：位 7 为 1 闪烁 位 6~4 为背景色 RGB 位 3 为 1 前景色高亮 位 2~0 为前景色 RGB

（12）BIOS 的 13H 调用

BIOS 的 13H 提供了磁盘读写的调用

BIOS 的 13H 号调用功能与参数

读扇区	13H	02H	AL：扇区数(1~255) DL：驱动器号(0 和 1 表示软盘，80H 和 81H 等表示硬盘或 U 盘) DH：磁头号(0~15) CH：柱面号的低 8 位 CL：0~5 位为起始扇区号(1~63)，6~7 位为硬盘柱面号的高 2 位(总共 10 位柱面号，取值 0~1023) ES：BX：读入数据在内存中的存储地址	返回值： ■ 操作完成后 ES：BX 指向数据区域的起始地址 ■ 出错时置进位标志 CF=1，错误代码存放在寄存器 AH 中 ■ 成功时 CF=0、AL=0

（13）返回 DOS

利用 4ch 功能调用返回 DOS（.COM/.EXE）

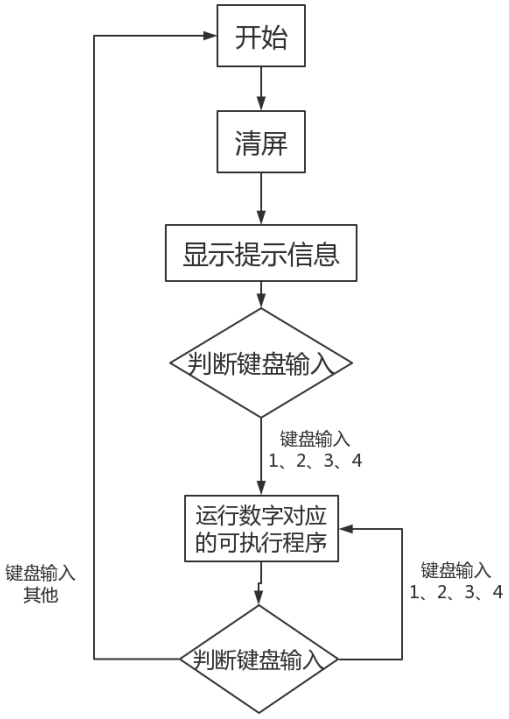
DOS 功能调用的 4ch 子功能（返回 DOS）：

入口参数：AH＝4ch，AL＝返回数码

产生终止程序执行返回操作系统的指令代码,它的可选参数是一个返回的数码，通常用 0 表示没有错误。例如程序结束点放置对应的代码是：

```
mov ax,4c00h
int 21h
```

5.程序流程



6. 程序关键代码及解释

【监控程序】

（1）初始化程序，显示字符串

首先是程序初始化部分，要想让监视程序和可执行子程序能在多个 bin 中实现相互跳转，需要让他们的偏移地址相同，由于跳转时要重新显示，所以要先调用清屏程序。字符串的显示，这次与第一次的不同，运用 BIOS 调用中 13H 功能号可以很方便的显示字符串。

```
org 7c00h ;引导地址
OffSetOfUserPrg1 equ 0A100h;偏移地址使其他执行程序以这个地址为基准
START:
    mov ax, cs
    mov ds, ax
    mov es, ax
    call CLEARScreen;调用清屏程序
    mov ah, 13h;功能号
    mov al, 01h
    mov bl,1bh
    mov bh,0 ; 第0页
    mov dh,0bh ; 第11行
    mov dl,1ah ; 第30列
    mov bp,Message1 ; BP=串地址
    mov cx,Message1_length ; 串长
    int 10h
    mov dh, 0eh
    mov dl, 2eh
    mov bp,Message2 ; BP=串地址
    mov cx,Message2_length ; 串长
    int 10h
    mov dh, 10h;
    mov dl, 2eh;
    mov bp,Message3 ; BP=串地址
    mov cx,Message3_length ; 串长
    int 10h
```

（2）判断跳转

首先使用 BIOS 调用判断有无按键输入，如果没有，就不断返回判断，如果有输入，跳转到下一步

```
BEGIN:
   mov ax,0100h
    int 16h
    jnz THEN
    int 10h
    jmp BEGIN
```

在判断完以后对输入的按键进行跳转，通过提前赋值给 program，使后面使用 BIOS 调用读盘时可以跳转到相应界面，扇区号为 2+program

```
THEN:;通过按键进行判断跳转界面，如果没按键就持续在当前界面停留
    mov ah,0;功能号
    int 16h
    cmp al,'1'
    jz P1
    cmp al,'2'
    jz P2
    cmp al,'3'
    jz P3
    cmp al,'4'
    jz P4
    jmp START
P1:
    mov byte[program],0
    jmp LOAD
P2:
    mov byte[program],1
    jmp LOAD
P3:
    mov byte[program],2
    jmp LOAD
P4:
    mov byte[program],3
    jmp LOAD
LOAD:
    call CLEARScreen
    mov ax, cs
    mov es, ax
    mov bx ,OffSetOfUserPrg1
    mov ax,0201h
    mov dx,0
    mov cx,02h
    add cl,byte[program]
    int 13h;读盘
    jmp OffSetOfUserPrg1
```

（3）清屏程序

使用之前的方法，调用二重循环，一个个往进写值，虽然比较占储存空间，但是比较容易想到。

（4）DEFINE 部分

单纯使用三个字符串，结尾用 55 AA 写值，用伪代码写入 4 个执行子程序，可以使编译后地址段在一起，便于调试

```
DFEINE:
    Message1 db 'PLEASE USE YOUR KEYBOARD TO INPUT COMMAND'
    Message1_length equ ($-Message1)
    Message2 db '15352405 15352406'
    Message2_length equ ($-Message2)
    program db 1
    Message3 db '15352407 15352408'
    Message3_length equ ($-Message3)
    x dw 0
    y dw 0
    times 510-($-$$) db 0
    db 0x55,0xaa
    incbin '1.bin'
    incbin '2.bin'
    incbin '3.bin'
    incbin '4.bin'
```

【可执行子程序】

第一个用户程序是在第一次的字符动态显示上加以修改完成的。
在程序完成初始化后延时之后调用 BIOS 判断键盘输入。

```
Start:
    mov cx,080h ;延时
    ;按键返回
    mov ax,0100h
    int 16h
    jnz Return
    jmp Delay

Return:
    jmp 0000:07c00h
```

主要需要修改运动轨迹的边界。

```
DnR:
    mov byte[state],bottomR ;将状态修改为朝右下方向运动
    mov byte[color1],03h    ;修改此时字符颜色
    cmp word[x1],11 ;判断是否到达下边界，到达则跳转到朝右上方向运动
    jz UpR
    cmp word[y1],39 ;判断是否到达右边界，到达则跳转到朝左下方向运动
    jz DnL

    inc word[x1] ;否则继续向右下方向运动
    inc word[y1]

    jmp Show    ;跳转至字符显示的部分
```

以及修改字符串的位置

```
ShowID:
    mov word[x2],5 ;选择中间的位置开始显示字符串
    mov word[y2],11
    mov si,0    ;用于记录字符串显示到第几位
    mov cx,18   ;循环次数
L1:
    call Calp   ;调用Calp计算偏移地址
    mov al,byte[id1+si] ;显示字符串的第si位
    mov byte[es:bx],al
    mov byte[es:bx+1],70h
    inc word[y2]
    inc si
    loop L1

    mov word[x2],6
    mov word[y2],11
    mov si,0
    mov cx,18
;显示第二行字符串
```

第二三个用户程序是可变色的心

代码段的起始地址

```
org 7c00h     ;代码段起始地址
START:        ;初始化ds（代码寄存器），es（附加段寄存器）
    mov ax,cs
    mov ds,ax
    mov es,ax
    mov byte[color],81h   ;初始化颜色变量
```

开始运行

```
BEGIN:
    ;delay
     mov cx,0F0h
    ;input from keyboard
    ;if input ,return
    mov ax,0100h
    int 16h      ;16h中断，键入内容
    jnz RETURN   ;zf寄存器为1，则跳转
    jmp DELAY
RETURN:
    jmp 0000:07c00h    ;返回调用它的主程序
```

主程序流程：

```
DELAY:
    push cx          ;循环的次数是保存在cx中的
    mov cx,0ffffh    ;无限循环DELAY2
DELAY2:
    loop DELAY2
    pop cx
    loop DELAY

    jmp heart        ;显示心形图案
    jmp START
```

Heart 函数的实现：将心形图案用多行字符串表示。

```
heart:     ;图形显示
    mov ax,1301h ;功能号
    mov bh,0
    mov bl,byte[color]   ;颜色
    mov dx,0035h   ;显示地址
    mov bp,line1   ;显示字符串
    mov cx,13      ;字符串长度
    int 10h
```

图像显示函数的颜色跳变

```
CHANGE:

    add byte[color],01h ;每次延迟之后，颜色变量加1
    cmp byte[color],86h ;跳变六次之后，初始化一次
    jz CHANGE2
    jmp BEGIN
CHANGE2:
    mov byte[color],81h ;初始化
    jmp START
end:
    jmp $
```

字符串的定义：

```
DEFINE:

    line1 db           '****       ****'
    line2 db         '*******     *******'
    line3 db     '************ ************'
    line4 db '******************************'
    color db 81h
    times 510-($-$$) db 0
    db 0x55,0xaa
```

第四个用户程序是显示一个日历。

首先将 ds 和 es 置于用户程序开始位置。

然后使用 BIOS 调用得到系统日历。接着依次读取世纪、年份、月份、日期，并把它们转化为 ASCII 码保存在字符串对应位置上中。然后使用 BIOS 调用显示字符串。

```
Start:
    mov ax,0a10h
    mov ds,ax
    mov es,ax
Begin:
    mov ax,0400h
    int 1ah
    mov di,date
    add di,7 ;跳过不需要修改的字符
    mov al,ch ;读取世纪
    call Trans ;将其转为ASCII码
    mov al,cl ;读取年份
    call Trans
    inc di
    mov al,dh ;读取月份
    call Trans
    inc di
    mov al,dl ;读取日期
    call Trans
    mov bp,line ;使用BIOS调用显示字符串
    mov cx,19
    mov ax,1301h
    mov bx,000fh
    mov dx,1232h
    int 10h
```

同样也需要判断键盘输入。

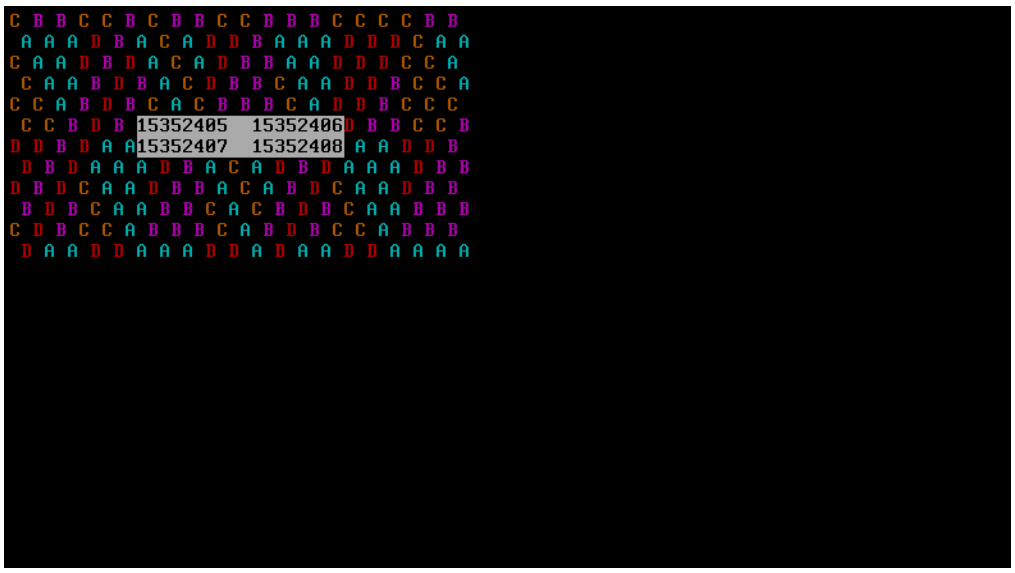
```
    mov ax,0100h
    int 16h
    jnz Return
    jmp Begin

Return:
    jmp 0000:07c00h
```

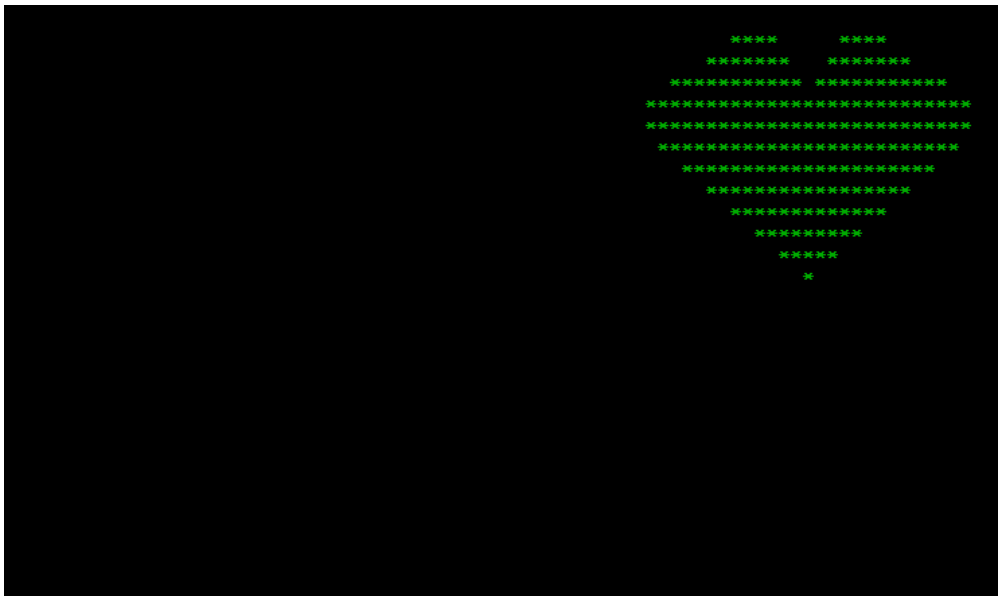
【实验过程】

首先进入监控程序主界面，按 1,2,3,4 会分别进入左上，右上，左下，右下的可执行子程序

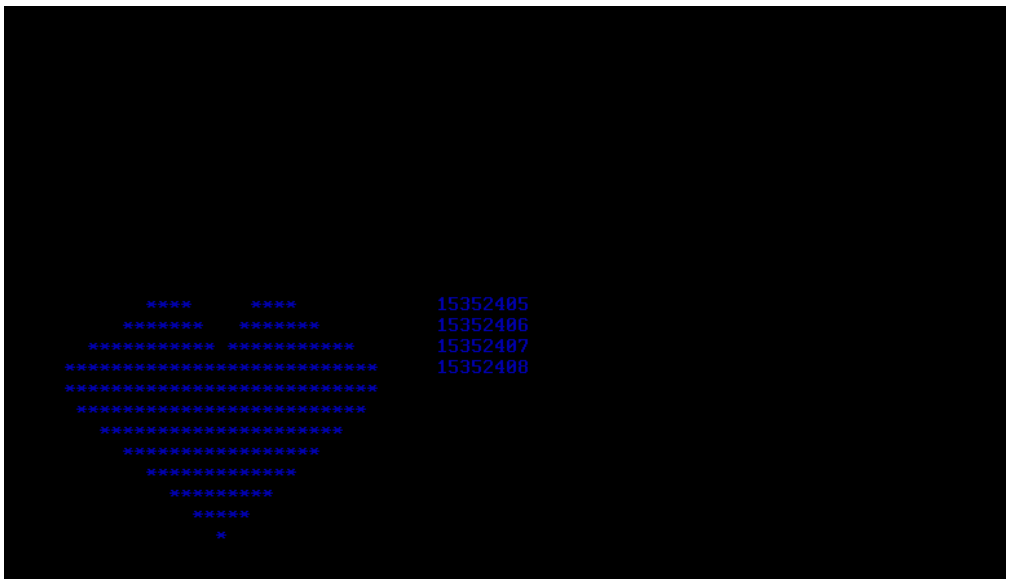
1 是对应左上的部分



2 是对应右上的部分



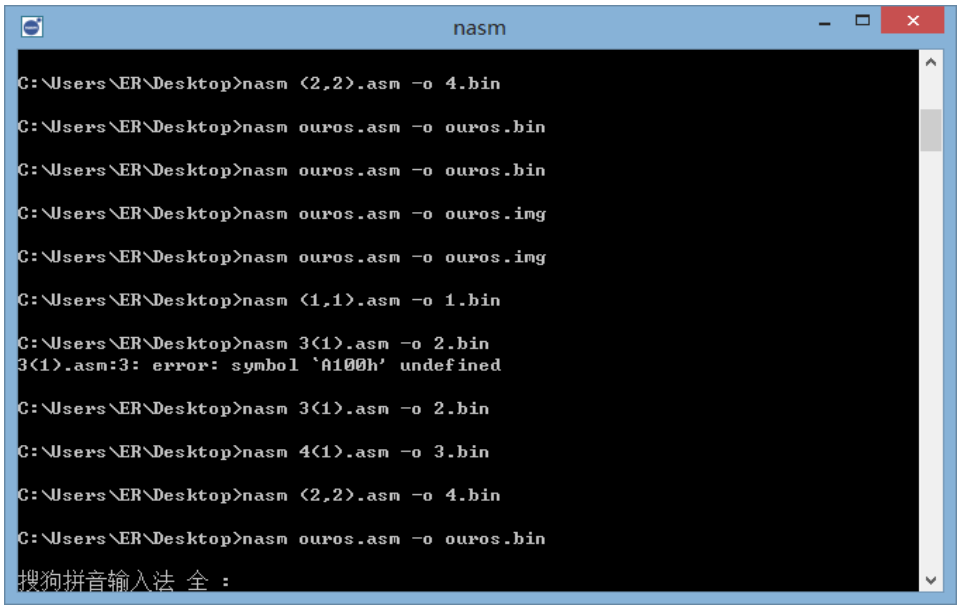
3 是对应左下的部分



4 是对应右下的部分

虚拟机的设置：

编译过程：



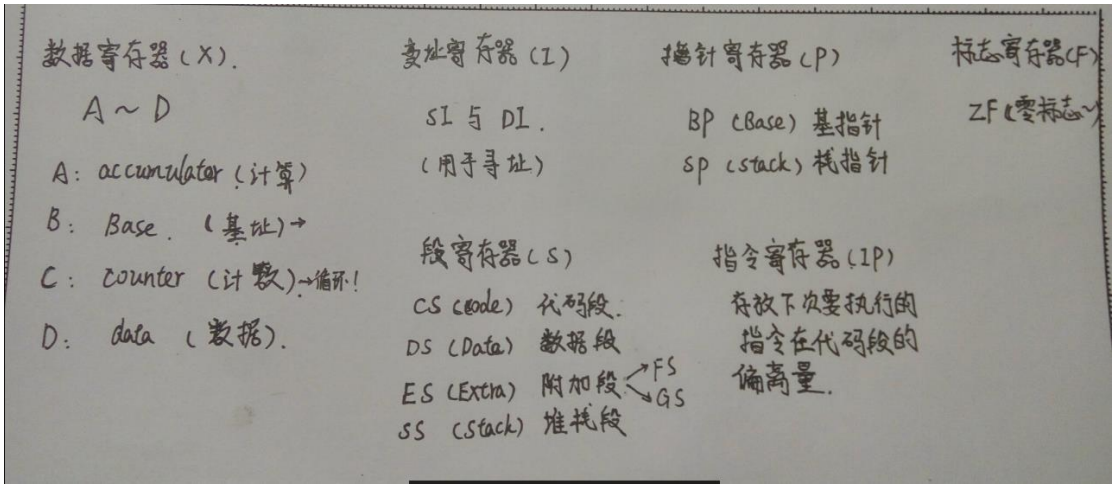
【实验总结】

张浩然的实验总结：

这次的实验我学习到最多的就是监控程序与用户程序，以及 BIOS 调用的相关知识。由于有了实验 1 的学习经验，以及实验 2 内容的要求本身也可以使用实验 1 中许多内容，所以我在写代码的时候比第一次实验时要熟练了不少，但是在调试代码时还是遇到了许多问题。遇到的第一个问题是写出来的 asm 文件无法使用 NASM 编译，一直显示 label or instruction expected at start of line 的错误，然而检查代码找不到错误。后来我发现可能是我使用过记事本修改这些代码，可能加进了不可识别的内容而没有显示出来。重新再一个新的 asm 文件里贴上代码便解决了问题。第二个问题便是我使用第一个用户程序的代码编译后直接在虚拟机上运行测试效果，发现字符串无法显示，考虑到这次的文件是不会出现大小超过的问题的，我不得不从其他方面寻找问题出在什么地方。参考范例的用户程序运行效果后，我发现不是我代码带来的问题，而是因为 ds 的位置设为了要求的 0A100h，在监控程序下便能正确地显示。第三个问题是显示日期的字符串乱码，查看老师的参考代码后，我发现使用 BIOS 调用中需要将 ES 与 DS 统一，修改后便能在监控程序下正确显示。这次实验的反思是代码的正确性和标准性。由于刚刚学习 X86 汇编，有时虽然编写出的代码能够按照预计效果运行，但会不会有代码写的不规范，不标准的地方。这是我目前比较担心，比较在意的地方。随着实验的不断进行，我觉得需要反过来审视现在写的代码对其优化修改。

张海涛的实验总结：

这次实验我的分工是实现一个可调用的子程序，一开始我以为有参照代码，实现起来会比较容易，但是事实是由于我很多基础的地方没有弄明白，所以整个设计过程无比艰难。首先是各种寄存器的作用，我查询之后，自己总结了一下。



总结之后，我再参照去理解源代码中的语句，既帮助了我理解代码含义，也加深了我对那些寄存器的用法规定。

踏出这一步之后，才发现有好多问题。例如字符显示：

这个解释起来就要涉及到显示缓冲区的问题了。

计算机机字符下共有80x25区域，共2000字符，每个字符占用2个字节，其中低字节为字符，高字节为字节的属性，共8位，每位都定义有显示属性，从高位到低位依次是 ：闪烁 背景红 背景绿 背景蓝 高亮 前景红 前景绿 前景蓝，这也是11111000b后3位置0的用意。最后3位就是改变字符颜色的关键。

经过一个晚上的调试，终于弄出了一个显示的图形，但是如何让这个达到动态显示的效果呢？我仔细研究了参考代码，发现动态的实现与 loop 指令有关。Loop 是循环指令，循环次数由计数寄存器 CX 指定。是否执行循环体的判断指令在循环体之后，所以，至少执行 1 次循环体，即至少循环 1 次。执行 LOOP 指令时，CPU 自动将 CX 的值减 1

直到 CX 为 0 ，循环结束。了解之后，我发现参考代码中还有 push 和 pop 的使用，又需要查询。push 和 pop 是堆栈操作指令,push [reg]/[num] 是将 reg 寄存器中的值或是数字 num 压入堆栈中,而 pop [reg]是将堆栈栈顶的值弹出到 reg 寄存器中,并将这个值从堆栈中删去.

堆栈可以看成是一个数组,但只能在栈顶(可以认为是数组的一端)对数据进行操作,起临时保存数据的作用,32 位汇编中,ss:esp 指向堆栈栈顶,16 位则是 ss:sp.。

了解 loop 实现之后，设想的动态效果还是没有实现，真的很让人焦虑。

我自己人工阅读运行了一下代码,发现问题发现 jmp 的跳转有逻辑问题，修改之后，动态效果实现。

在学习新东西时，心里都会有一种未知的恐惧，导致不敢大胆尝试，遇到问题，大部分时间都浪费在了焦虑上。

张晗宇的实验总结：

这次的实验主要是学习 BIOS 调用，在写清屏函数时，原本打算百度看看有没有更加简单的清屏函数，找了好几个，但是效果不是很好，最后还是模仿王烁程的清屏代码，虽然调用两个 loop 清屏的方法很占空间（毕竟只有 512 字节），但是在对汇编语言更深入理解之前也只能这样写。这次的字符串显示显然比上一次实验的简单了很多，不用再通过 loop 写，所占的字节空间也不大。在显示字符串的时候，一开始我以为 10H 在所有显示完再加就可以了，导致程序迟迟无法显示正确的字符串，检查的时候也没注意，导致花费了很多时间在字符串的显示上。一开始很好奇如何通过 BIOS 调用实现 5 个独立程序能合在一起运行，后来才知道他们的偏移地址相同，只要通过访问不同扇区就可以实现调用，其中监控程序需要引导地址，而其可执行子程序可以不需要在代码中添加引导地址，这也导致一开始写子程序时在开头未加引导地址，导致程序无法正常运行。由于一开始对 BIOS 调用不是很熟悉，在对心形的显示上出现了很多问题，最初什么都显示不了，我们通过一个个注释检查，最后发现 BIOS 调用有问题，还有就是没有加

```
times 510-($-$$) db 0
 db 0x55,0xaa
```

的子程序也会出问题，但是我们不知道是为什么。这次的实验相比较还是比较成功的，让我们了解到了 BIOS 调用的优势，以及知道了一种打破 512KB 的困境的方法。

张镓伟的实验总结：

这次我的工作是作为总设计师，由我决定 4 个子程序分别是什么内容并分派工作，然后我负责将 4 个子程序和连接起来以及帮助队友们 debug 3 个新增的程序的代码。这次实验帮助我们打破了上一次实验引导程序代码汇编后大小不能超过 512k 的限制。上一次的实验由于代码大小限制我们不得不放弃一些更多的显示内容。这一次我知道了可以通过将超过 512k 字节的程序分解为引导扇区部分和其他功能部分。相当于 C++中 main 函数里调用其它子函数，引导扇区就相当于 main，其他功能就相当于子函数。此外，我还学习了很多关于 BIOS 调用的知识。BIOS 调用可以让我们很方便地去实现一些功能比如我们这次的程序中的显示字符串，读取键盘输入，读取系统时间都用到了 BIOS 调用。在连接 4 个用户子程序的时候，我知道了可以使用类似 C++中的 include 的方式。这两次实验中我不断将汇编代码的编写方式与我熟悉的 C++相比较，发现了上述这些共同点让我对 x86 代码的理解有了一个提升。这次在帮队友 debug 的时候，一个队友在做显示会变色的心形图案的时候出现了不会变色的 bug，他检查了一整天没有查出错误，后来，我仔细地看了他代码发现他的循环每次都从初始化开始，也就是说颜色一直在初始化，这自然就不会变色。这告诉我们，写代码要细心，要理清逻辑。总的来说，这次实验还是学到了很多东西。