



《操作系统实验》 实验报告

(实验三)

学 院 名 称 : 数据科学与计算机学院

专业 (班级) : 17 级计算机类教务 3 班

学 生 姓 名 : 姚森舰

学 号 : 17341189

时 间 : 2019 年 3 月 28 日

一. 实验题目

将操作系统分离为引导程序和系统内核，由引导程序加载内核，用 C 和汇编实现操作系统内核，并且在内核的 C 模块中增加批处理能力。

二. 实验目的

- (1) 学习 C 语言与汇编语言交叉调用、混合编程；
- (2) 熟悉 TCC 与 TASM 的来编译C和汇编；
- (3) 加深对操作系统内核的理解。

三. 实验要求

- (1) 将实验二的操作系统分离为引导程序和系统内核，由引导程序加载内核，用 C和汇编实现操作系统内核；
- (2) 扩展内核汇编代码，增加一些有用的输入输出函数，供 C 模块中调用；
- (3) 提供用户程序返回内核的一种解决方案；
- (4) 在内核的 C 模块中实现增加批处理能力；
- (5) 在磁盘上建立一个表，记录用户程序的存储安排；
- (6) 可以在控制台命令查到用户程序的信息，如程序名、在磁盘映像文件中的位置等；
- (7) 设计一种命令，命令中可加载多个用户程序，依次执行，并能在控制台发出命令；
- (8) 在引导系统前，将一组命令存放在磁盘映像中，系统可以解释执行；

四. 实验方案

实验环境：Windows10 +VMware

实验工具：NASM + Winhex + Tasm + Tcc + Dosbox + Notepad++

大概流程：用Notepad++编写C语言程序和汇编程序，用nasm把实验二中的用户程序和引导程序编译成二进制代码，在Dosbox下用TCC编译C代码，用TASM编译内核的部分汇编代码，在用Tlink将编译好的C和汇编代码link成.com文件。通过VMware创建带软盘的虚拟机，利用winhex将编译好的引导程序二进制代码写到虚拟机的软盘的第1个扇区，将文件数据写到第2个扇区，将内核代码放到3~12扇区，然后将用户程序放到第13~16扇区，然后在Vmware上运行测试虚拟机和程序。

大概思路：由引导程序引入到内核，内核里面最基础的输入输出、显示、读数据部分等等由汇编实现，再用 C 语言利用这些函数做一些复杂一些的工作，比如输出字符串等

等。再用 C 语言实现通过用户的输入，调用用户程序或者一些其他指令，实现类似命令行的功能。

关键代码及部分程序解释：

```
extern void cls();           /* 清屏 */
extern void printchar(char ch, int pos); /* 在指定位置输出字符 */
extern void showchar(char phar);      /* 输出字符 */
extern void getChar();              /* 输入字符 */
extern void progInfo();             /* 读程序信息 */
extern void loadp(int index);        /* 加载程序 */
void prints(char const *Messeage);   /* 输出字符串 */
int strlen(char const *Messeage);    /* 计算字符串长度 */
int strcmp(char const *m1, char const *m2); /* 比较字符串 */
void helpList();                   /* 输出提示信息 */
void getinfo();                   /* 将程序信息输入到结构体存储 */
void listshow();                  /* l指令，显示程序信息 */
void runprog(int index);          /* 运行程序n */
void readcmd();                  /* 读指令 */
void MyCmd();                    /* 运行指令 */
```

一些基本函数，解释见注释，实现见源文件

```
while(1){
    getChar();
    if(save == 8){
        char k = ' ';
        index_of_str--;
        recv[index_of_str] = '\0';
        showchar(save);
        showchar(k);
        showchar(save);
        continue;
    }
    showchar(save);

    if(save == 32){ /*空格不考虑*/
        continue;
    }

    if(save == 13){ /*回车命令输入完毕*/
        recv[index_of_str] = '\0';
        break;
    }
}
```

退格的实现

```
struct programs{
    char name[8];
    char pos[3];
    char size[2];
};

struct programs p[4];
```

装程序信息的结构体

```
fun4:
    name4 db "4.com"
    times 56-($-$$) db 0
    pos4 db "16"
    times 60-($-$$) db 0
    size4 db "1"
    times 64-($-$$) db 0

script:
    cmd1 db "r 1 2 3 4 1"
    times 96-($-$$) db 0
```

程序信息和批处理文件内容，后者可以更改测试

```

public _progInfo
_progInfo proc

    push cx
    push ax
    push dx
    push bx

    mov ah,2           ; 功能号
    mov bx,9c00h
    mov al,1           ; 扇区数
    mov dl,0           ; 驱动器号 ; 软盘为0, 硬盘和U盘为80H
    mov dh,0           ; 磁头号 ; 起始编号为0
    mov ch,0           ; 柱面号 ; 起始编号为0
    mov cl,2           ; 起始扇区号 ; 起始编号为1
    mov [_readdata],bx
    int 13H ;          调用读磁盘BIOS的13h功能

    pop bx
    pop dx
    pop ax
    pop cx
    ret

_progInfo endp

```

读取用户程序信息，将数据在内存的位置给readdata指针读取信息

```

public _loadp
_loadp proc
    push cx
    push ax
    push dx
    push bp
    push bx
    mov bp,sp

    mov cl,byte ptr[bp+2+2+2+2+2] ; int\IP\cx\ax\dx\bp\bx\ex
    add cl,12                     ; 13扇区开始, i>=1
    mov bx,offset0fUserProg
    mov ah,2                     ; 功能号
    mov al,1                     ; 扇区数
    mov dl,0                     ; 驱动器号 ; 软盘为0, 硬盘和U盘为80H
    mov dh,0                     ; 磁头号 ; 起始编号为0
    mov ch,0                     ; 柱面号 ; 起始编号为0
    int 13H ;                     调用读磁盘BIOS的13h功能

    call bx

    pop bx
    pop bp
    pop dx
    pop ax
    pop cx
    ret
_loadp endp

```

加载用户程序，并跳转执行，在用户程序的最后用ret返回内核

```

}
readcmd();
if(strcmp(recv,help) == 1 || strcmp(recv,quit) == 1 || strcmp(recv,list) == 1 || strcmp(recv,run) == 1 ||
   prints("\r\n");
   if(strcmp(recv,help) == 1){

       else if( strcmp(recv,quit) == 1){

       else if(strcmp(recv,list) == 1){

       else if(recv[0] == 'r'){

       else if(strcmp(recv,clear) == 1){

       else if(strcmp(recv,"init.cmd") == 1){

       else{
           prints("Invalid input.\n\r");
           continue;
       }
   }
}

```

读指令及运行指令

指令输入以回车结束，可以退格，可以空格，输入不正确是会有提示，输入过程中最好只输入数字和字符串，而ctrl, esc, shift, 方向键等这些键可能会导致此次输入无法正常执行，需要重新输入。

l: 展示文件信息；

h: 显示提示信息；

c: 清屏；

r: 运行对应编号的程序，比如“r 1”即运行用户程序程序1，“r 4 1 2 3”即依次运行用户程序4，用户程序1，用户程序2，用户程序3. 指令有无空格都行，运行完后自动返回；

q: 退出；

init.cmd: 运行磁盘上的批处理文件。文件内容在第二扇区，可以修改测试；

扇区安排:

1: 引导程序

2: 程序信息及批处理文件内容

3~12: 内核

13~16: 用户程序

五. 实验过程与思想

1. 安装 VMware 并按要求创建一个无操作系统的裸机。

▼ 设备	
内存	4 MB
处理器	1
硬盘 (IDE)	102 MB
CD/DVD (IDE)	自动检测
软盘	正在使用文件 C:...
网络适配器	NAT
声卡	自动检测
显示器	自动检测

安装 Dosbox 并将 TCC, TASM, Tlink 等 exe 文件粘贴到对应文件

gc.bat	2014/10/8 13:18	Windows 批处理...	1 KB
gcc.exe	2013/10/6 1:17	应用程序	1,777 KB
ld.bat	2014/10/8 12:51	Windows 批处理...	1 KB
na.bat	2013/12/9 14:55	Windows 批处理...	1 KB
nasm.exe	2013/1/2 17:06	应用程序	726 KB
setting.bat	2014/10/8 12:31	Windows 批处理...	1 KB
startCmd.bat	2013/5/9 12:58	Windows 批处理...	1 KB
ta.bat	2013/5/18 15:04	Windows 批处理...	1 KB
TASM.EXE	1996/2/21 5:00	应用程序	133 KB
tc.bat	2013/12/10 10:51	Windows 批处理...	1 KB
TCC.EXE	2012/5/8 12:05	应用程序	177 KB
TLINK.EXE	2012/5/8 12:05	应用程序	22 KB

2. 编写汇编程序:

关键代码见实验方案，具体代码见代码文件

3. 编译:

将引导程序，用户程序和文件信息编译成二进制:

```
nasm
Microsoft Windows [版本 10.0.17134.648]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\user\AppData\Local\bin\NASM>nasm BootLoader.asm -o boot.bin

C:\Users\user\AppData\Local\bin\NASM>nasm os3_1.asm -o 1.bin

C:\Users\user\AppData\Local\bin\NASM>nasm os3_2.asm -o 2.bin

C:\Users\user\AppData\Local\bin\NASM>nasm os3_3.asm -o 3.bin

C:\Users\user\AppData\Local\bin\NASM>nasm os3_4.asm -o 4.bin

C:\Users\user\AppData\Local\bin\NASM>nasm data.asm -o data.bin

C:\Users\user\AppData\Local\bin\NASM>
```

将二进制文件利用 Winhex 写到对应扇区:

000001A0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000001B0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000001C0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000001D0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000001E0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000001F0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 55 AA	U ^a
00000200	31 2E 63 6F 6D 00 00 00	31 33 00 00 31 00 00 00	1.ccm 13 1
00000210	32 2E 63 6F 6D 00 00 00	31 34 00 00 31 00 00 00	2.ccm 14 1
00000220	33 2E 63 6F 6D 00 00 00	31 35 00 00 31 00 00 00	3.ccm 15 1
00000230	34 2E 63 6F 6D 00 00 00	31 36 00 00 31 00 00 00	4.ccm 16 1
00000240	72 20 31 20 32 20 33 20	34 20 31 00 00 00 00 00	r 1 2 3 4 1
00000250	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000260	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	

将内核的 C 代码和汇编代码在 Dosbox 下利用 tcc, tasm, tlink 编译链接:

```
DOS
BOX
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
Unable to change to: os.

C:\ASM\OS>tcc -mt -c -ok.obj kernel.c
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
kernel.c:

    Available memory 438500

C:\ASM\OS>tcc -mt -c -ok.obj kernel.c
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
kernel.c:

    Available memory 438500

C:\ASM\OS>tasm hunbian.asm h.obj
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file:  hunbian.asm to h.obj
Error messages:   None
Warning messages: None
Passes:          1
Remaining memory: 466k

C:\ASM\OS>
```

```
DOS
BOX
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
kernel.c:

    Available memory 438500

C:\ASM\OS>tcc -mt -c -ok.obj kernel.c
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
kernel.c:






    Available memory 438500

C:\ASM\OS>tasm hunbian.asm h.obj
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file:  hunbian.asm to h.obj
Error messages:   None
Warning messages: None
Passes:          1
Remaining memory: 466k

C:\ASM\OS>tlink /3 /t h.obj k.obj,test.com,,
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

C:\ASM\OS>
```

-  H.MAP
-  H.OBJ
-  hunbian.asm
-  K.OBJ
-  kernel.c

运行测试:

在 Vmware 上运行虚拟机:

进入界面:

```
-----
:                               Welcome to YaoOS                               :
-----

Command help:
-l: to get some help imformation.
-r n: (n is the index of user program.) to run the user program with index 'n'.
    For example: input "r 1 3" to run user program1 and then program3.
-h: to get some help imformation.
-c: clear screen.
-q: quit.
-init.cmd: to run the batch command in the disk.
user $: _
```

l 指令: 显示程序信息和 init.cmd 批处理文件的内容

```
-----
:                               Welcome to YaoOS                               :
-----

Command help:
-l: to get some help imformation.
-r n: (n is the index of user program.) to run the user program with index 'n'.
    For example: input "r 1 3" to run user program1 and then program3.
-h: to get some help imformation.
-c: clear screen.
-q: quit.
-init.cmd: to run the batch command in the disk.
user $: l

filename      shanqu      size
1.com         13             1
2.com         14             1
3.com         15             1
4.com         16             1
batch command:r 1 2 3 4 1
user $: _
```

h 指令: 显示帮助信息, 各条指令的意义

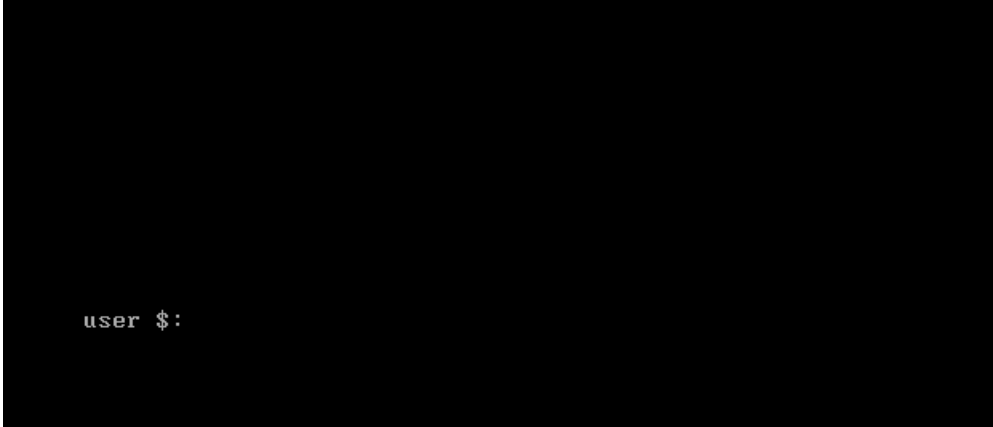
```
-----
:                               Welcome to YaoOS                               :
-----

-r n: (n is the index of user program.) to run the user program with index 'n'.
    For example: input "r 1 3" to run user program1 and then program3.
-h: to get some help imformation.
-c: clear screen.
-q: quit.
-init.cmd: to run the batch command in the disk.
user $: l

filename      shanqu      size
1.com         13             1
2.com         14             1
3.com         15             1
4.com         16             1
batch command:r 1 2 3 4 1
user $: h
Command help:
-l: to get some help imformation.
-r n: (n is the index of user program.) to run the user program with index 'n'.
    For example: input "r 1 3" to run user program1 and then program3.
-h: to get some help imformation.
-c: clear screen.
-q: quit.
-init.cmd: to run the batch command in the disk.

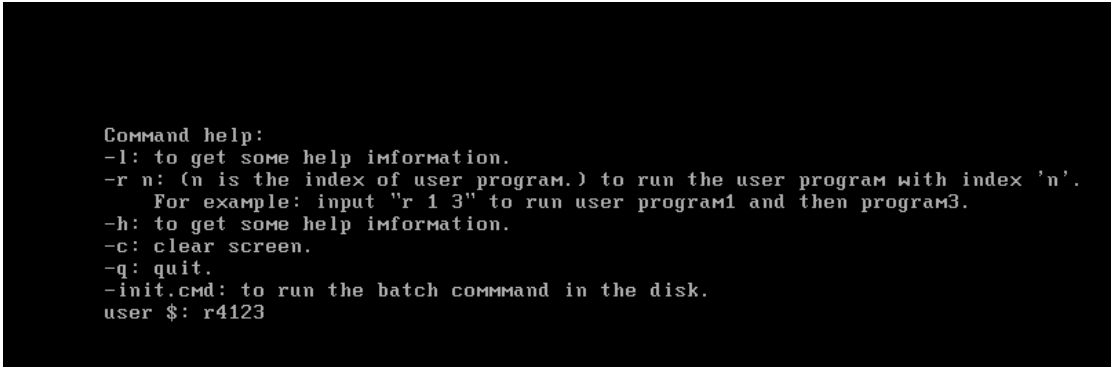
user $: _
```

c 指令：清屏

A terminal window with a black background. The text "user \$:" is visible at the bottom left.

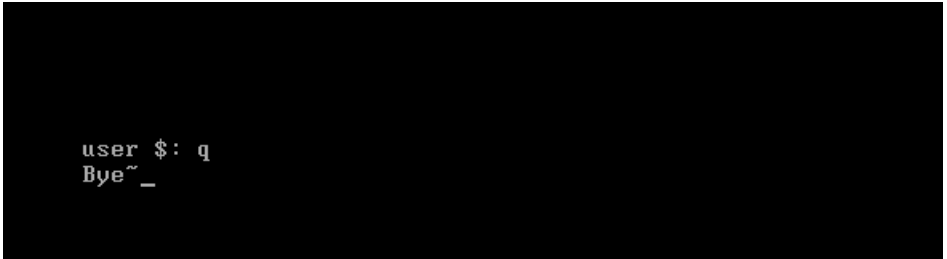
```
user $:
```

r 指令：运行对应编号的程序，比如“r 1”即运行用户程序程序 1，“r 4 1 2 3”即依次运行用户程序 4，用户程序 1，用户程序 2，用户程序 3. 指令有无空格都行。

A terminal window with a black background. The text "Command help:" is followed by a list of commands and their descriptions. The last line is "user \$: r4123".

```
Command help:
-l: to get some help imformation.
-r n: (n is the index of user program.) to run the user program with index 'n'.
    For example: input "r 1 3" to run user program1 and then program3.
-h: to get some help imformation.
-c: clear screen.
-q: quit.
-init.cmd: to run the batch command in the disk.
user $: r4123
```

q 指令：退出

A terminal window with a black background. The text "user \$: q" is followed by "Bye~_" on the next line.

```
user $: q
Bye~_
```

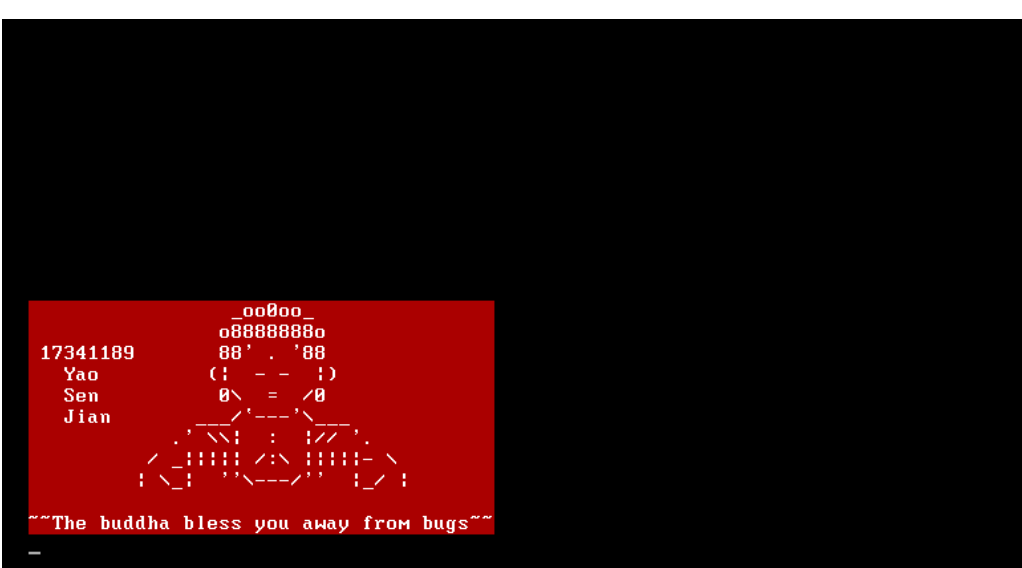
用户程序 1



用户程序 2



用户程序 3



```
_oo0oo_
o88888888o
88' . '88
(: - - :)
0\ = /0
   ^-----^
  / \      | \     :|       \| 
 / _|||_|_|_|_|_|_\ / _|||_|_|_|_|_\
/_-|||||-'''\---/'-\_-|||||-'''\--\_
|:\__|\ '''\----/\'''\___| :\__|\
```

The buddha bless you away from bugs~~~

```

Command help:
-l: to get some help information.
-r n: (n is the index of user program.) to run the user program with index 'n'.
    For example: input "r 1 3" to run user program1 and then program3.
-h: to get some help information.
-c: clear screen.
-q: quit.
-init.cmd: to run the batch command in the disk.
user $: init.com

```

```

-----
:                               Welcome to YaoOS                               :
-----

Command help:
-l: to get some help information.
-r n: (n is the index of user program.) to run the user program with index 'n'.
    For example: input "r 1 3" to run user program1 and then program3.
-h: to get some help information.
-c: clear screen.
-q: quit.
-init.cmd: to run the batch command in the disk.
user $: pppppppppppp
The command is too long, please input again.
user $: kk
Invalid input.
user $:
```

六. 实验心得和体会

这个实验花了很多时间，一路踩坑走过来，有时候一天只能发现一个 bug，解决了马上又会遇到新的问题，很是头痛。但是实际上还是有部分功能是没有做的很好，比如在磁盘里写信息只能写字符串，所以程序信息只能自己手写进去，读出来还得转换才能使用，但时间问题，我并没有去实现这个字符串转数字的函数，这就导致我的内核在读信息的时候对信息格式有很严格的要求，比如扇区位置必须两位数，程序的大小必须不超过 10 个扇区。想要实现这个函数得考虑很多情况，而且实验的主要目的也不是这个，所以没有实现。因此，我的内核还有很多地方值得改进。

通过这个实验，我对 C 语言和汇编的互相调用、混合编译有了一定的了解，对操作系统结构的认识也更深了一层。另外写程序还是先要自顶向下想好结构，想清楚了，理论基础了解了，再去自底向上地去实现，这样既能使得结构清晰，也能减少 bug 的出现。我开始做的时候就是没有想清楚直接上手，结果 bug 又多又难找，花费了很多时间，最后很大一部分还是重新写过了。

另外，即使我压栈了，我原来的第三个程序也会有问题，就是无法返回，找了很久也没找到问题，所以我将原来的第三用户程序改成了现在的程序。

下面就是碰到的一些问题：

1. Tcc+Tasm+Tlink 使用问题。这套工具比较古老，使用的时候遇到了很多问题，如：不支持行注释，只能用 `/**/`；其中 `include` 的 `asm` 文件文件名不能太长，超过 7 个字符就会说找不到文件；另外，c 程序中的变量必须在函数内部最前面声明，否则会说语法错误；在将 `obj` 文件链接起来时，汇编部分的 `obj` 文件应放在前面，这个可能是程序执行的先后问题；还有比较字符时，字符要用单引号，我在程序里面有一个语句是 `recv[1] == 'r'`，如果这里用双引号，结果就会出问题，程序就不对，但我也没找到原因所在，所以猜是编译器的问题。
2. 寄存器数据保护的问题。子程序开始一定要把用到的寄存器压栈保护，否则在某些地方会出问题。我用的是 `call` 子程序在内存的地址实现调用用户程序，开始没压栈，程序在同学的虚拟机上能正常返回，在我的上就不能，同学的程序在我的虚拟机上也能返回，最后才发现应该是自己的程序中的某些值被改变了。而有的程序没有压栈也能正常执行，我觉得这是个运气问题，只是正好没有修改关键的值而使程序的异常没有表现出来。
3. `MOV AX,[ES:BX]` 会提示 warning，改成 `MOV AX,ES:[BX]` 即可。
4. 加载内核的时候，需要 `jmp` 到对应的内存位置，比如我的是 `0A100H`，跳转的之后必须写成 `jmp A00H:100H`，如果写成 `jmp 0A100H` 就会跳错位置。我的理解是前者，改变了 CS，使其变为 A00，跳到了段值为 A000，偏移量为 100H 的位置，是正确的，后者则是只改变了 IP，跳到了当前段值，偏移量为 A100H 的位置，就不是我们想要的位置，就会出错。我的理解是这样，但不知道正确与否。

总的来说，踩过了那些坑，花费了很多时间，但完成后受益匪浅，不过多和同学讨论一下，分享一下遇到的问题，应该能提高一下效率。过去好几天都因为这个实验心烦，终于做完了，高兴。

七. 参考资料

-
- [1] [int 13h 参数详解](#)
 - [2] [TASM 汇编与 TurboC 混合编程](#)
 - [3] [CALL 和 RET 指令](#)