

# 中山大学数据科学与计算机学院本科生实验报告

## (2016 学年秋季学期)

课程名称：操作系统实验

任课教师：凌应标

教学助理 (TA):

年级	15 级	专业 (方向)	软件工程 (移动信息工程)
学号	15352461	姓名	宗嘉希 (组长)
学号	15352443	姓名	钟凌山
学号	15352448	姓名	周禅城
电话	18022724490	Email	zongjx@mail2.sysu.edu.cn
开始日期	2017.04.21	完成日期	2017.04.27

### 【实验题目】

设置中断与编写中断服务程序

### 【实验目的】

熟悉时钟中断、键盘中断，学习如何编写中断

### 【实验要求】

- 1、操作系统工作期间，利用时钟中断，在屏幕 24 行 79 列位置轮流显示 ' | '、' / ' 和 ' \ '，适当控制显示速度，以方便观察效果。
- 2、编写键盘中断响应程序，原有的你设计的用户程序运行时，键盘事件会做出有事反应：当键盘有按键时，屏幕适当位置显示 " OUCH! OUCH! "。
- 3、在内核中，对 33 号、34 号、35 号和 36 号中断编写中断服务程序，分别在屏幕 1/4 区域内显示一些个性化信息。再编写一个汇编语言的程序，作为用户程序，利用 int 33、int 34、int 35 和 int 36 产生中断调用你这 4 个服务程序。

## 【实验方案】

### 一、 硬件及虚拟机配置

硬件：操作系统为 win10 的笔记本电脑

虚拟机配置：无操作系统，10MB 硬盘，4MB 内存，启动时连接软盘

### 二、 软件工具及作用

Nasm:用于编译汇编程序，生成.bin 文件

WinHex:用于向软盘写入程序

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

Notepad++: 用于编辑汇编语言文件

TCC：用于编译 C 文件，生成 OBJ 文件

TASM：用于编译 ASM 文件，生成 OBJ 文件

TLINK：用于把两个 OBJ 文件连接，生成 COM 文件

Dosbox：用于提供 16 位运行环境，为 TCC+TASM+TLINK 编译链接提供一个环境

## 【程序功能】

开机进入内核，内核提供 5 种指令输入，如下图：

```
Z
ZWelcome to our team's system!
ZYou can input some legal instructions to active the functions
Z  If you want to run the program by Batching,input 'batch'
Z    For example: batch 1 2 3 4
Z  If you want to run the program by Time-Sharing,input 'time'
Z    For example: time 1 2 3 4
Z  If you want to know the information of the program,input 'show'
Z    For example: show
Z  If you want to run the default instructions,input 'default'
Z    For example: default
Z  If you want to run the interrupt test program,input 'interrupt'
Z    For example: interrupt
Z
ZOrz >_
Z
Z
Z
Z
OS Running!
```

1. 时钟中断：在内核运行期间，会在屏幕边框逆时针显示字符，每次经过左上角变更字符，每次显示变更颜色。

在进入用户程序之前，恢复默认中断向量表以停止边框显示；在从用户程序返回内核后，重新设置中断向量表继续显示边框。

2. 键盘中断：在内核进入用户程序前，修改当前键盘中断向量表，使用户程序中每次按键盘都会显示“Ouch!Ouch!”，且显示位置不断变化。从用户程序返回内核后，恢复默认中断向量表，以保证内核正常的输入功能。

3. 批处理功能：指令 batch 之后，首先输入要批处理程序的序号，范围为 0~4。之后依次运行对应序号的程序。

4. 分时功能：指令 batch 之后，首先输入要分时运行的程序的序号，范围为 0~4。之后同时运行对应序号的程序。

5. 显示用户程序信息：指令 `show`,可以显示用户程序的信息。

7. 命令行输入：左下的 `Orz>`处输入命令，此处模仿 `cmd`，在输入过程中，若输入错误，可以进行删除一定字符以修正输入。同时，命令对空格不要求，可没有空格或有多个空格。命令以回车结束。

## 【实验过程】

首先，配置好虚拟机，创建好一个 1.44MB 的虚拟软盘并连接到虚拟机。

然后我们首先设计了四个不一样的有输出的用户可执行程序代码，分别编译出四份可执行程序。四个程序的内容是：

1、程序显示在屏幕的左上角的四分之一的位置，用字符 ‘Z’ 从屏幕左上角位置 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。在最左上角打印出组员 1 的姓名和学号。

2、程序显示在屏幕的右上角的四分之一的位置，用字符 ‘Z’ 从屏幕右上角位置 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。在最右上角打印出组员 2 的姓名和学号。

3、程序显示在屏幕的左下角的四分之一的位置，用字符 ‘Z’ 从屏幕左上角位置 45 度角上斜射出，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。在最左下角打印出组员 3 的姓名和学号。

4、程序显示在屏幕的右下角的四分之一的位置，用字符 ‘Z’ 从屏幕左上角位置 45 度角上斜射出，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。在最右下角打印出所有组员的姓名和学号。

这四个程序其实和实验 3 的是一样的，没有进行大的改动，直接使用就可以了。

因为是以实验 3 的内容为基础，在里面加上几个中断，因此我们把实验 3 的内核拿出来使用，保留实验 3 的所有功能并进行改动。

首先，我们可以来看一下 c 语言文件：

以下是外部函数的声明：

```
1  /*内核c程序*/
2  extern void clear(); /*连接外部的clear()函数,用于清屏*/
3  extern void inputchar(); /*连接外部的inputchar()函数,用于读入字符*/
4  extern void printchar(); /*连接外部的printchar()函数,用于输出字符*/
5  extern void printstring(); /*连接外部的printstring()函数,用于输出字符串*/
6  extern void setcursor(); /*连接外部的setcursor()函数,用于设置光标*/
7  extern void load_prog(); /*连接外部的load_prog()函数,用于加载外部程序*/
8  extern void run_prog(); /*连接外部的run_prog()函数,用于跳转到外部程序*/
9  extern int put_color_char(); /*连接外部的put_color_char()函数,用于打印颜色字符*/
```

把以上函数声明好后就要声明出要显示的字符串：

```
11 /*以下将创建若干长度为80的字符串变量(屏幕最大长度为80),用于存放提示语句*/
12 char message1[80]="\n Welcome to our team's system!\n";
13 char message3[80]=" You can input some legal instructions to active the functions\n";
14 char message4[80]=" If you want to run the program by Batching,input \'batch\'\n";
15 char message5[80]=" For example: batch 1 2 3 4\n";
16 char message6[80]=" If you want to run the program by Time-Sharing,input \'time\'\n";
17 char message7[80]=" For example: time 1 2 3 4\n";
18 char message8[80]=" If you want to know the information of the program,input \'show\'\n";
19 char message9[80]=" For example: show\n";
20 char message10[80]=" If you want to run the default instructions,input \'default\'\n";
21 char message11[80]=" For example: default\n";
22 char message12[80]="\n If you want to continue,input \'yes\',otherwise input \'no\'\n";
23 char message13[80]="\n OS will run instructions \'batch 1 2 3 4\' and \'time 1 2 3 4\'\n";
24 char message14[80]=" If you want to run the interrupt test program,input \'interrupt\'\n";
25 char message15[80]=" For example: interrupt\n\n";
26 char message16[80]=" Show as below:\n";
27 char message_cmd[80]=" Orz >";
28 char message_ins1[80]=" User Program1:\n Name:program_LU Size:1024byte Position:part LU\n";
29 char message_ins2[80]=" User Program2:\n Name:program_RU Size:1024byte Position:part RU\n";
30 char message_ins3[80]=" User Program3:\n Name:program_LD Size:1024byte Position:part LD\n";
31 char message_ins4[80]=" User Program4:\n Name:program_RD Size:1024byte Position:part RD\n";
32 char message_ins5[80]=" Interrupt Program:\n Name:program_Interrupt Size:1024byte \n";
33 char string[80];
34 char message_error[20]=" Input Error!!!\n";
35 const char batch_order[20]="batch 1 2 3 4";
36 const char time_order[80]="time 1 2 3 4";
```

还有部分需要用到的变量：

```
37 int len=0; /*代表长度的变量*/
38 int pos=0; /*代表位置的变量*/
39 char ch; /*代表输入的字符*/
40 int x=0; /*纵向位置的变量*/
41 int y=0; /*横向位置的变量*/
42 int i=0; /*用于计数的变量*/
43 int j=0; /*用于计数的变量*/
44 int k=0; /*用于计数的变量*/
45 int data1=0; /*数据存储的变量*/
46 int offset_prog=41216; /*用户程序的偏移量,0a10h*10h+0h=37120(初始位置)*/
47 const int offset_prog1=41216; /*第一个用户程序的偏移地址 0a10h*10h+0h=37120*/
48 const int offset_prog2=42240; /*第二个用户程序的偏移地址 0a50h*10h+0h=38144*/
49 const int offset_prog3=43264; /*第三个用户程序的偏移地址 0a90h*10h+0h=39168*/
50 const int offset_prog4=44288; /*第四个用户程序的偏移地址 0ad0h*10h+0h=40192*/
51 const int offset_prog_int=45312; /*测试中断的用户程序的偏移地址 0b10h*10h+0h=45312*/
52 const int hex600h=1536; /*600h的十进制,作为内存地址用于标记用户程序的调用情况*/
53 int offset_begin=41216; /*初始位置内存偏移量*/
54 int num_sector=10; /*扇区的总数为10个*/
55 int pos_sector=2; /*起始扇区的编号为2,第一个扇区放置的是引导程序*/
```

接下来就要定义一些在 c 语言里面比较容易实现的函数了：

输入数据的函数：

```
150 int input() { /*控制输入字符的函数*/
151     inputchar(); /*读入一个字符的函数*/
152     if(ch=='\b'){ /*如果读入的字符是删除键Backspace*/
153         if(y>6&&y<79){ /*如果光标的横向位置是在6到79之间*/
154             y--; /*光标向左移一位*/
155             cal_pos(); /*重新计算光标的坐标*/
156             putchar(' '); /*输出一个空白格*/
157             y--; /*y变量自减1*/
158             cal_pos(); /*重新计算光标的位置*/
159         }
160         return 0; /*如果输入的是删除键，则返回0*/
161     }
162     else if(ch==13); /*回车键*/
163     else putchar(ch); /*如果不是以上情况，就显示字符*/
164     return 1; /*如果不是以上情况，就返回1*/
165 }
```

调用输入命令的函数：

```
167 cin_cmd() { /*输出命令提示符的函数*/
168     printstring(message_cmd); /*打印字符串的函数*/
169     i=0; /*初始字符串下标*/
170     while(1){ /*while循环*/
171         if(input()){ /*如果输入的不是删除键*/
172             if(ch==13) break; /*如果输入的字符是回车键，则跳出while循环*/
173             string[i++]=ch; /*否则把该字符添加到string的末尾*/
174         }
175         else if(i>0) i--; /*如果输入的是删除键，就删除一个字符*/
176     }
177     for(j=0;j<i;j++) /*进入for循环，遍历string字符串*/
178         if(string[j]==' '){ /*如果字符串首位是空格*/
179             for(k=1;k<i;k++) string[k-1]=string[k]; /*把首位空格去掉*/
180             i--; /*字符串长度减1*/
181         }
182     }
183     else break; /*如果字符串首位不是空格，就跳出循环*/
184     string[i]='\0'; /*字符串末尾补上一个空格，也可以把i-1*/
185     len=i; /*记录下字符串的长度i*/
186     printstring("\n"); /*换行*/
187 }
```

计算光标的坐标的函数：

```
189 cal_pos() { /*计算字符或光标的坐标的函数*/
190     if(y>79){ /*假如y方向大于79(到达了右边界)*/
191         y=0; /*y变为0，到达下一行的起始位置*/
192         x++; /*行数x加1*/
193     }
194     if(x>24) clear(); /*假如x大于24(到达了下边界)，清空屏幕到下一页*/
195     pos=(x*80+y)*2; /*重新计算位置*/
196     setcursor(); /*放置光标的函数，在pos的位置放置光标*/
197 }
```



## 实现分时功能命令的函数：

```

235 time() {
236     clear();
237     offset_begin=offset_prog1;
238     num_sector=10;
239     pos_sector=2;
240     load_prog(offset_begin,num_sector,pos_sector);
241     filldata(hex600h,0);
242     filldata(hex600h+2,0);
243     filldata(hex600h+4,0);
244     filldata(hex600h+6,0);
245     for(i=0;i<len;i++){
246         if(string[i]=='1')
247             filldata(hex600h,1);
248         else if(string[i]=='2')
249             filldata(hex600h+2,1);
250         else if(string[i]=='3')
251             filldata(hex600h+4,1);
252         else if(string[i]=='4')
253             filldata(hex600h+6,1);
254     }
255     while(1){
256         j=0;
257         readdata(hex600h);
258         j+=data1;
259         if(data1){
260             offset_prog=offset_prog1;
261             run_prog();
262         }
263         readdata(hex600h+2);
264         j+=data1;
265         if(data1){
266             offset_prog=offset_prog2;
267             run_prog();
268         }
269         readdata(hex600h+4);
270         j+=data1;
271         if(data1){
272             offset_prog=offset_prog3;
273             run_prog();
274         }
275         readdata(hex600h+6);
276         j+=data1;
277         if(data1){
278             offset_prog=offset_prog4;
279             run_prog();
280         }
281         if(j==0) break;
282     }
283 }

```

/\*实现伪分时处理的函数\*/  
/\*清屏，清除引导程序所显示的内容\*/  
/\*设置初始位置内存偏移量等于第一个用户程序的偏移量\*/  
/\*扇区的总数为10个\*/  
/\*起始扇区的编号为2\*/  
/\*装载用户程序到内存，利用中断读取扇区\*/  
/\*向hex600h(600h)这个内存位置填进0，表示第一个用户程序暂不运行\*/  
/\*向hex600h+2(602h)这个内存位置填进0，表示第二个用户程序暂不运行\*/  
/\*向hex600h+4(604h)这个内存位置填进0，表示第三个用户程序暂不运行\*/  
/\*向hex600h+6(606h)这个内存位置填进0，表示第四个用户程序暂不运行\*/  
/\*进入for循环，遍历字符串(输入的指令)\*/  
/\*如果字符串中出现'1'字符\*/  
/\*向hex600h(600h)这个内存位置填进1，表示第一个用户程序将会运行\*/  
/\*如果字符串中出现'2'字符\*/  
/\*向hex600h(602h)这个内存位置填进1，表示第二个用户程序将会运行\*/  
/\*如果字符串中出现'3'字符\*/  
/\*向hex600h(604h)这个内存位置填进1，表示第三个用户程序将会运行\*/  
/\*如果字符串中出现'4'字符\*/  
/\*向hex600h(606h)这个内存位置填进1，表示第四个用户程序将会运行\*/  
/\*进入while循环，程序将会分时运行\*/  
/\*记录器j初始为0，此处用于记录是否有程序运行\*/  
/\*读取hex600h(600h)地址中的值，取出的数据将会存到data变量中\*/  
/\*将data的值加到记录器j中\*/  
/\*如果data中的值非0\*/  
/\*把第一个用户程序的偏移地址写入到初始地址变量中\*/  
/\*调用运行程序的函数(此处将使用offset\_prog变量)\*/  
/\*读取hex600h+2(602h)地址中的值，取出的数据将会存到data变量中\*/  
/\*将data的值加到记录器j中\*/  
/\*如果data中的值非0\*/  
/\*把第二个用户程序的偏移地址写入到初始地址变量中\*/  
/\*读取hex600h+4(604h)地址中的值，取出的数据将会存到data变量中\*/  
/\*将data的值加到记录器j中\*/  
/\*如果data中的值非0\*/  
/\*把第三个用户程序的偏移地址写入到初始地址变量中\*/  
/\*调用运行程序的函数(此处将使用offset\_prog变量)\*/  
/\*读取hex600h+6(606h)地址中的值，取出的数据将会存到data变量中\*/  
/\*将data的值加到记录器j中\*/  
/\*如果data中的值非0\*/  
/\*把第四个用户程序的偏移地址写入到初始地址变量中\*/  
/\*调用运行程序的函数(此处将使用offset\_prog变量)\*/  
/\*如果4个数据都为0，则退出循环\*/

## 实现批处理功能命令的函数：

```

199 batch() {
200     clear();
201     offset_begin=offset_prog1;
202     num_sector=10;
203     pos_sector=2;
204     load_prog(offset_begin,num_sector,pos_sector);
205     for(i=0;i<len;i++){
206         if(string[i]=='1'){
207             offset_prog=offset_prog1;
208             run_prog();
209         }
210         else if(string[i]=='2'){
211             offset_prog=offset_prog2;
212             run_prog();
213         }
214         else if(string[i]=='3'){
215             offset_prog=offset_prog3;
216             run_prog();
217         }
218         else if(string[i]=='4'){
219             offset_prog=offset_prog4;
220             run_prog();
221         }
222     }

```

/\*实现批处理功能的函数\*/  
/\*清屏，清除引导程序所显示的内容\*/  
/\*设置初始位置内存偏移量等于第一个用户程序的偏移量\*/  
/\*扇区的总数为10个\*/  
/\*起始扇区的编号为2\*/  
/\*装载用户程序到内存，利用中断读取扇区\*/  
/\*进入for循环，遍历字符串(输入的指令)\*/  
/\*如果字符串中出现'1'字符\*/  
/\*把第一个用户程序的偏移地址写入到初始地址变量中\*/  
/\*调用运行程序的函数(此处将使用offset\_prog变量)\*/  
/\*如果字符串中出现'2'字符\*/  
/\*把第二个用户程序的偏移地址写入到初始地址变量中\*/  
/\*调用运行程序的函数(此处将使用offset\_prog变量)\*/  
/\*如果字符串中出现'3'字符\*/  
/\*把第三个用户程序的偏移地址写入到初始地址变量中\*/  
/\*调用运行程序的函数(此处将使用offset\_prog变量)\*/  
/\*如果字符串中出现'4'字符\*/  
/\*把第四个用户程序的偏移地址写入到初始地址变量中\*/  
/\*调用运行程序的函数(此处将使用offset\_prog变量)\*/

接下来就是与中断有关的函数了：

```
57 char rch;
58 int is_rotate, rpx, rpy, rcolor;
59 int rotate_state;
60 c_rotate() {
61     rpx=23;
62     rpy=78;
63     rcolor=15;
64     put_color_char('O', rpx, 68, rcolor);
65     put_color_char('S', rpx, 69, rcolor);
66     put_color_char('R', rpx, 71, rcolor);
67     put_color_char('u', rpx, 72, rcolor);
68     put_color_char('n', rpx, 73, rcolor);
69     put_color_char('n', rpx, 74, rcolor);
70     put_color_char('i', rpx, 75, rcolor);
71     put_color_char('n', rpx, 76, rcolor);
72     put_color_char('g', rpx, 77, rcolor);
73     if(is_rotate!=1){
74         rotate_state=1;
75         is_rotate=1;
76     }
77     if(rotate_state==1){
78         rch='-';
79         put_color_char(rch, rpx, rpy, rcolor);
80         rotate_state++;
81     }
82     else if(rotate_state==4){
83         rch='/';
84         put_color_char(rch, rpx, rpy, rcolor);
85         rotate_state++;
86     }
87     else if(rotate_state==7){
88         rch='|';
89         put_color_char(rch, rpx, rpy, rcolor);
90         rotate_state++;
91     }
92     else if(rotate_state==10){
93         rch='\\';
94         put_color_char(rch, rpx, rpy, rcolor);
95         rotate_state=1;
96     }
97     else{
98         rotate_state++;
99     }
```

/\*表示要输出的一个字符变量，显示旋转的杠\*/  
/\*整型变量，用于记录字符打印的坐标、颜色、初始化状态\*/  
/\*整型变量表示杠旋转的状态\*/  
/\*控制旋转的杠的函数\*/  
/\*固定纵坐标为23\*/  
/\*固定横坐标为79\*/  
/\*颜色为黑底白字\*/  
/\*以下几行代码是在旋转的横杠前输出Running字样\*/  
  
/\*如果横杠没有在转\*/  
/\*初始化横杠的状态\*/  
/\*把横杠设置为旋转状态\*/  
  
/\*如果状态为1，即横杠为'-'\*/  
/\*输出'-'\*/  
/\*状态加1， 也做延时\*/  
  
/\*如果状态为4，即横杠为'/'\*/  
/\*输出'/'\*/  
/\*状态加1， 也做延时\*/  
  
/\*如果状态为7，即横杠为'|'\*/  
/\*输出'|'\*/  
/\*状态加1， 也做延时\*/  
  
/\*如果状态为10，即横杠为'\'\*/  
/\*输出'\'\*/  
/\*状态加1， 也做延时\*/  
  
/\*如果都不在状态\*/  
/\*状态加1， 也做延时\*/

```
102 int is_ouch;
103 c_ouch() {
104     if(is_ouch==1){
105         x=y=pos=0;
106         is_ouch=0;
107     }
108     printstring("Ouch!");
109 }
```

/\*整型变量，用于判断是否已经初始化\*/  
/\*键盘中断显示Ouch程序\*/  
/\*初始化\*/  
/\*初始位置为左上角\*/  
/\*标记更改\*/  
  
/\*打印"Ouch!"\*/



```

112 int px,py,pcolor,pdir,pbegin; /*整型变量，用于记录字符打印的坐标、颜色、方向、初始化状态*/
113 char pch; /*表示要打印的字符的符号变量*/
114 int pdx[4]; /*表示字符转弯的数组变量*/
115 int pdy[4]; /*表示字符转弯的数组变量*/
116 void c_paint() { /*时钟中断画边框程序*/
117     if(pbegin!=1) /*初始化*/
118     {
119         px=0; /*初始纵坐标为0*/
120         py=0; /*初始横坐标*/
121         pcolor=1; /*初始颜色为1*/
122         pdir=0; /*初始方向*/
123         pch='Z'; /*初始字符为'Z'*/
124         pbegin=1; /*改变初始化标志*/
125         pdx[0]=1,pdx[1]=0,pdx[2]=-1,pdx[3]=0; /*定义数组变量*/
126         pdy[0]=0,pdy[1]=1,pdy[2]=0,pdy[3]=-1; /*定义数组变量*/
127     }
128     put_color_char(pch,px,py,pcolor); /*显示字符，更新颜色和位置*/
129     pcolor=(pcolor+1)%15+1; /*更新颜色*/
130     px+=pdx[pdir]; /*更新纵坐标方向*/
131     py+=pdy[pdir]; /*更新横坐标方向*/
132     if(pdir==1||pdir==3) /*如果在上方或下方，就再显示一个字符，并更新位置*/
133     {
134         if(py>0&&py<79)
135         {
136             put_color_char(pch,px,py,pcolor);
137             px+=pdx[pdir];
138             py+=pdy[pdir];
139         }
140     }
141     if(px==py&&px==0) pch--; /*在左上角顶点变更字符*/
142     if(pch=='A') pch='Z'; /*当字符到达'A'的时候，变回'Z'*/
143
144     if(px==24&&py==0|| /*在四个角的顶点出变更方向*/
145        px==24&&py==79||
146        px==0&&py==79||
147        px==0&&py==0) pdir=(pdir+1)%4;
148 }

```

主函数如下：

```

285 void main() { /*主函数*/
286     while(1) { /*进入while循环，开始运行程序*/
287         clear(); /*清屏操作*/
288         printstring(message1); /*打印message1字符串*/
289         printstring(message3); /*打印message3字符串*/
290         printstring(message4); /*打印message4字符串*/
291         printstring(message5); /*打印message5字符串*/
292         printstring(message6); /*打印message6字符串*/
293         printstring(message7); /*打印message7字符串*/
294         printstring(message8); /*打印message8字符串*/
295         printstring(message9); /*打印message9字符串*/
296         printstring(message10); /*打印message10字符串*/
297         printstring(message11); /*打印message11字符串*/
298         printstring(message14); /*打印message14字符串*/
299         printstring(message15); /*打印message15字符串*/
300         cin_cmd(); /*显示命令提示符，并接收输入的指令*/
301         if(string[0]=='b') { /*如果接收到的指令的首位字符是'b'，则意味着调用了批处理功能的函数*/
302             batch(); /*调用batch函数，实现批处理功能*/
303             printstring(message12); /*打印message12字符串*/
304             cin_cmd(); /*显示命令提示符，并接收输入的指令*/
305             if(string[0]=='y') continue; /*如果输入的字符串的首位字符是'y'，则重新进入while循环*/
306         }
307         else if(string[0]=='t') { /*如果接收到的指令的首位字符是't'，则意味着调用了伪分时功能的函数*/
308             time(); /*调用batch函数，实现批处理功能*/
309             printstring(message12); /*打印message12字符串*/
310             cin_cmd(); /*显示命令提示符，并接收输入的指令*/
311             if(string[0]=='y') continue; /*如果输入的字符串的首位字符是'y'，则重新进入while循环*/
312         }
313         else if(string[0]=='i') { /*如果接收到的指令的首位字符是'i'，则意味着调用了中断功能的函数*/
314             interrupt1(); /*调用interrupt函数，实现中断*/
315             printstring(message12); /*打印message12字符串*/
316             cin_cmd(); /*显示命令提示符，并接收输入的指令*/
317             if(string[0]=='y') continue; /*如果输入的字符串的首位字符是'y'，则重新进入while循环*/
318         }
319         else if(string[0]=='s') { /*如果接收到的指令的首位字符是's'，则显示每个程序的信息*/
320             printstring(message_ins1); /*由于在打印这些字符串的时候，刚好打印完会跳到下一页导致看不到任何东西*/
321             printstring(message_ins1); /*因此多输出几行字符串把页面压倒下一页的开头*/

```

```

322     printstring(message1);
323     printstring(message_ins1);
324     printstring(message_ins1);
325     printstring(message16);
326     printstring(message_ins1);          /*打印message_ins1字符串*/
327     printstring(message_ins2);          /*打印message_ins2字符串*/
328     printstring(message_ins3);          /*打印message_ins3字符串*/
329     printstring(message_ins4);          /*打印message_ins4字符串*/
330     printstring(message_ins5);          /*打印message_ins5字符串*/
331     printstring(message12);             /*打印message12字符串*/
332     cin_cmd();                          /*显示命令提示符，并接收输入的指令*/
333     if(string[0]=='y') continue;         /*如果输入的字符串的首位字符是'y'，则重新进入while循环*/
334 }
335 else if(string[0]=='d'){                /*如果接收到的指令的首位字符是'd'，则显示每个程序的信息*/
336     printstring(message13);             /*打印message13字符串*/
337     printstring(message12);             /*打印message12字符串*/
338     cin_cmd();                          /*显示命令提示符，并接收输入的指令*/
339     if(string[0]=='y'){                  /*如果输入的字符串的首位字符是'y'*/
340         for(i=0;i<80;i++){              /*进入for循环，将批处理功能的指令存到string中*/
341             string[i]=batch_order[i];    /*字符串(批处理指令)转移*/
342             if(batch_order[i]=='\0'){      /*如果到达字符串尾部*/
343                 len=i;                   /*记录下字符串的长度*/
344                 break;                   /*跳出for循环*/
345             }
346         }
347         batch();                          /*调用batch函数，实现批处理功能*/
348         printstring(message12);           /*打印message12字符串*/
349         cin_cmd();                       /*显示命令提示符，并接收输入的指令*/
350         if(string[0]!='y') continue;      /*如果输入的字符串的首位字符是'y'，继续运行程序*/
351         for(i=0;i<80;i++){              /*进入for循环，将批处理功能的指令存到string中*/
352             string[i]=time_order[i];      /*字符串(伪分时指令)转移*/
353             if(time_order[i]=='\0'){      /*如果到达字符串尾部*/
354                 len=i;                   /*记录下字符串的长度*/
355                 break;                   /*跳出for循环*/
356             }
357         }
358         time();
359         printstring(message12);           /*打印message12字符串*/
360         cin_cmd();                       /*显示命令提示符，并接收输入的指令*/
361     }
362 }
363 else{
364     printstring(message_error);          /*打印message_error字符串*/
365     printstring(message12);             /*打印message12字符串*/
366     cin_cmd();                          /*显示命令提示符，并接收输入的指令*/
367     if(string[0]=='y') continue;         /*如果输入的字符串的首位字符是'y'，继续运行程序*/
368 }
369 }
370 }
371
289 set_clock_interrupt proc              ;设置时钟中断 08h
290     cli                                ;清除IF标志，使芯片屏蔽可屏蔽中断
291     push es                            ;把es压栈
292     push ax                            ;把ax压栈
293     xor ax,ax                          ;把ax置0
294     mov es,ax                          ;把ax的值存进es寄存器中
295     ;save the vector
296     mov ax,word ptr es:[20h]           ;把时钟中断入口es:20h存到ax中 (08h*4=20h)
297     mov word ptr [clock_vector],ax     ;把ax存进clock_vector向量中，保存原来的值
298     mov ax,word ptr es:[22h]           ;把键盘中断入口es:22h存到ax中 (08h*4+2=22h)
299     mov word ptr [clock_vector+2],ax   ;把ax存进clock_vector+2向量中，保存原来的值
300     ;fill the vector
301     mov word ptr es:[20h],offset Paint;设置时钟中断向量的偏移地址，当中断触发时自动调用Paint函数
302     mov ax,cs                          ;把cs存进ax中
303     mov word ptr es:[22h],ax           ;设置时钟中断向量的段地址
304     pop ax                             ;把ax弹出栈
305     pop es                             ;把es弹出栈
306     sti                                ;设置IF标志，使芯片不屏蔽可屏蔽中断
307     ret                                ;返回
308 set_clock_interrupt endp              ;函数定义结束

```

以上为止，c 函数的编写就结束了，每一条语句的具体功能都在代码上有注释（如上）。

在这一个 c 程序中实现的其实是等于前一次实验的引导扇区的功能，输出提示字符，然后提供输入

命令的功能和显示信息的功能。

完成 c 文件后，就要着手去写和它相呼应的汇编文件了。

首先还是先要用 extrn 标识符去定义一些在 c 文件里面的变量，作用是与 c 文件里面一样的，都是关联两个文件，使这些变量可以互用。要注意，在这里定义变量的时候要在变量名前加一个下划线“\_”，因为 c 文件编译成 obj 文件的时候，变量名前都会加上“\_”（具体原因我不清楚），因此要加上才能关联上对应的变量。

```
1 ;汇编模块
2 extrn _main:near ;使用extrn调用c模块中的main函数
3 extrn _cal_pos:near ;使用extrn调用c模块中的cal_pos函数
4 extrn _c_paint:near ;使用extrn调用c模块中的c_paint函数
5 extrn _pos:near ;使用extrn调用c模块中的pos变量
6 extrn _ch:near ;使用extrn调用c模块中的ch变量
7 extrn _x:near ;使用extrn调用c模块中的x变量
8 extrn _y:near ;使用extrn调用c模块中的y变量
9 extrn _offset_prog:near ;使用extrn调用c模块中的offset_prog变量
10 extrn _data1:near ;使用extrn调用c模块中的data变量
11 extrn _is_ouch:near ;使用extrn调用c模块中的is_ouch变量
12 extrn _c_ouch:near ;使用extrn调用c模块中的c_ouch函数
13 extrn _c_rotate:near ;使用extrn调用c模块中的c_rotate函数
```

汇编代码跟实验 3 的内容差不多，只不过加上了以下的中断代码：

```
35 clock_vector dw 0,0 ;时钟向量
36 keyboard_vector dw 0,0 ;键盘向量
37
38 public _setcursor ;定义设置光标的函数
39 _setcursor proc ;子程序定义伪指令
40     push ax ;此处分别将ax, bx, dx按顺序压栈，用于保护数据
41     push bx ;由于函数中的中断操作将会用到这三个寄存器
42     push dx ;因此先将这几个寄存器里的原始数据压栈保护
43     mov ah,02h ;功能号设置为02h，光标定位
44     mov dh,byte ptr [_x] ;设置dh为x变量的值，表示起始的行数
45     mov dl,byte ptr [_y] ;设置dl为y变量的值，表示起始的列数
46     mov bh,0 ;bh设置为0，表示第0页
47     int 10h ;设置中断号为10h，进行中断
48     pop dx ;中断调用完成后恢复各个寄存器中的原始值
49     pop bx ;因此按顺序弹出dx, bx, ax的原始值
50     pop ax ;分别放回原处
51     ret ;返回
52 _setcursor endp ;子函数的定义结束
```

```

54 public _printchar ;定义打印一个字符的函数
55 _printchar proc ;子程序定义伪指令
56     push ax ;此处分别将ax, es, bp, bx按顺序压栈, 用于保护数据
57     push es ;由于函数中的中断操作将会用到这四个寄存器
58     push bp ;因此先将这几个寄存器里的原始数据压栈保护
59     push bx ;
60     call _setcursor ;调用c模块中的setcursor函数, 放置光标
61     mov bp, sp ;把栈顶位置寄存器赋值给bp寄存器
62     mov ax, 0b800h ;把显存起始位置存到ax寄存器中
63     mov es, ax ;把ax寄存器的值存到段地址
64     mov al, byte ptr [bp+10] ;把传入函数的数据存到al寄存器中, 表示要输出的字符
65     mov ah, 0fh ;把0fh存进ah寄存器中, 表示要输出的字符的样式为白字黑底
66     mov bx, word ptr [_pos] ;把pos变量存进bx寄存器中, 代表输出字符的坐标
67     mov word ptr es:[bx], ax ;在屏幕上打印该字符
68     inc word ptr [_y] ;y变量加1(横向坐标加1)
69     call near ptr _cal_pos ;调用cal_pos函数, 用于计算新的坐标
70     pop bx ;中断调用完成后恢复各个寄存器中的原始值
71     pop bp ;因此按顺序弹出bx, bp, es, ax的原始值
72     pop es ;分别放回原处
73     pop ax ;
74     ret ;返回
75 _printchar endp ;子函数的定义结束

247 set_keyboard_interrupt proc ;设置键盘中断 09h
248     cli ;清除IF标志, 使芯片屏蔽可屏蔽中断
249     push es ;把es压栈
250     push ax ;把ax压栈
251     xor ax, ax ;把ax置0
252     mov es, ax ;把ax的值存进es寄存器中
253     ;save the vector
254     mov ax, word ptr es:[24h] ;把键盘中断入口es:24h存到ax中 (09h*4=24h)
255     mov word ptr [keyboard_vector], ax ;把ax存进keyboard_vector向量中, 保存原来的值
256     mov ax, word ptr es:[26h] ;把键盘中断入口es:26h存到ax中 (09h*4+2=26h)
257     mov word ptr [keyboard_vector+2], ax ;把ax存进keyboard_vector+2向量中, 保存原来的值
258     ;fill the vector
259     mov word ptr es:[24h], offset Ouch ;设置键盘中断向量的偏移地址, 当中断触发时自动调用Ouch函数
260     mov ax, cs ;把cs存进ax中
261     mov word ptr es:[26h], ax ;设置键盘中断向量的段地址
262     ;用于中断服务C程序的变量
263     mov word ptr [_is_ouch], 1 ;改变is_ouch变量, 初始化标志
264     pop ax ;把ax弹出栈
265     pop es ;把es弹出栈
266     sti ;设置IF标志, 使芯片不屏蔽可屏蔽中断
267     ret ;返回
268 set_keyboard_interrupt endp ;函数定义结束

270 re_keyboard_interrupt proc ;恢复键盘中断
271     cli ;清除IF标志, 使芯片屏蔽可屏蔽中断
272     push es ;把es压栈
273     push ax ;把ax压栈
274     xor ax, ax ;把ax置0
275     mov es, ax ;把ax的值存进es寄存器中
276     mov ax, word ptr [keyboard_vector] ;把原先保存的键盘中断向量原来的值存进ax中
277     mov word ptr es:[24h], ax ;恢复键盘中断
278     mov ax, word ptr [keyboard_vector+2] ;把原先保存的键盘中断向量原来的值存进ax中
279     mov word ptr es:[26h], ax ;恢复键盘中断
280     ;用于中断服务C程序的变量
281     mov word ptr [_is_ouch], 0 ;重置初始化标志
282     pop ax ;把ax弹出栈
283     pop es ;把es弹出栈
284     sti ;设置IF标志, 使芯片不屏蔽可屏蔽中断
285     ret ;返回
286 re_keyboard_interrupt endp ;函数定义结束

```



```

310 re_clock_interrupt proc ;恢复时钟中断
311     cli ;清除IF标志，使芯片屏蔽可屏蔽中断
312     push es ;把es压栈
313     push ax ;把ax压栈
314     xor ax,ax ;把ax置0
315     mov es,ax ;把ax的值存进es寄存器中
316     mov ax,word ptr [clock_vector] ;把原先保存的时钟中断向量原来的值存进ax中
317     mov word ptr es:[20h],ax ;恢复时钟中断
318     mov ax,word ptr [clock_vector+2] ;把原先保存的时钟中断向量原来的值存进ax中
319     mov word ptr es:[22h],ax ;恢复时钟中断
320     pop ax ;把ax弹出栈
321     pop es ;把es弹出栈
322     sti ;设置IF标志，使芯片不屏蔽可屏蔽中断
323     ret ;返回
324 re_clock_interrupt endp ;函数定义结束

```

```

359 Ouch proc ;定义Ouch函数
360     cli ;清除IF标志，使芯片屏蔽可屏蔽中断
361     push es ;依次将下列寄存器压栈，因为要使用到这些寄存器，保护原值
362     push si
363     push di
364     push ax
365     push bx
366     push cx
367     push dx
368     push bp
369     push ds
370
371     call near ptr _c_ouch ;调用c_rotate函数，当键盘有输入时打印Ouch!
372
373     in al,60h ;读缓冲区
374
375     mov al,20h ;把20h存到al寄存器中(al=EOI)
376     out 20h,al ;发送EOI到主芯片
377     out 0a0h,al ;发送EOI到从芯片
378
379     pop ds ;依次将下列寄存器弹出栈，恢复原值
380     pop bp
381     pop dx
382     pop cx
383     pop bx
384     pop ax
385     pop di
386     pop si
387     pop es
388     sti ;设置IF标志，使芯片不屏蔽可屏蔽中断
389     iret ;中断返回
390 Ouch endp

```

326	Paint proc	;定义Paint函数
327	cli	;清除IF标志,使芯片屏蔽可屏蔽中断
328	push es	;依次将下列寄存器压栈,因为要使用到这些寄存器,保护原值
329	push si	
330	push di	
331	push ax	
332	push bx	
333	push cx	
334	push dx	
335	push bp	
336	push ds	
337		
338	call near ptr _c_rotate	;调用c_rotate函数,打印横杠
339		
340	call near ptr _c_paint	;调用c_paint函数,在屏幕周围打印字符
341		
342	mov al,20h	;把20h存到al寄存器中(al=EOI)
343	out 20h,al	;发送EOI到主芯片
344	out 0a0h,al	;发送EOI到从芯片
345		
346	pop ds	;依次将下列寄存器弹出栈,恢复原值
347	pop bp	
348	pop dx	
349	pop cx	
350	pop bx	
351	pop ax	
352	pop di	
353	pop si	
354	pop es	
355	sti	;设置IF标志,使芯片不屏蔽可屏蔽中断
356	iret	;中断返回
357	Paint endp	;函数定义结束

392	public _put_color_char	;输出一个指定位置的指定颜色的字符
393	_put_color_char proc	;子程序定义伪指令
394	mov bp,sp	;把栈的原地址存到bp中
395	push es	;把以下寄存器压栈
396	push ax	;
397	push bx	;
398		
399	mov ax,0b800h	;显存初始位置存进ax
400	mov es,ax	;把ax存进es
401	mov ax,word ptr [bp+4]	;把传入的第二个变量(x坐标)存到ax中
402	mov bx,80	;计算横坐标
403	mul bx	;
404	add ax,word ptr [bp+6]	;把传入的第三个变量(y坐标)存到ax中
405	mov bx,2	;计算纵坐标
406	mul bx	;
407	mov bx,ax	;得到相对位置
408	mov ax,word ptr [bp+2]	;把传入的第一个变量(字符)存到ax中
409	mov byte ptr es:[bx],al	;把字符存进该位置
410	mov ax,word ptr [bp+8]	;把传入的第四个变量(颜色)存到ax中
411	mov byte ptr es:[bx+1],al	;设置颜色
412		
413	pop bx	;把以下寄存器弹出栈
414	pop ax	;
415	pop es	;
416	ret	;
417	_put_color_char endp	;子程序定义结束



33 号中断：

```

419 int33h proc                ;子程序定义伪指令(33号中断)
420     cli                    ;清除IF标志，使芯片屏蔽可屏蔽中断
421     push es                 ;把es压栈
422     push ax                 ;把ax压栈
423     xor ax,ax               ;把ax置0
424     mov es,ax               ;把ax的值存进es寄存器中
425     mov word ptr es:[0cch],offset interrupt33h;设置中断向量的偏移地址，当中断触发时自动调用interrupt33h函数
426     mov ax,cs               ;把cs存进ax
427     mov word ptr es:[0ceh],ax ;设置中断向量的段地址
428     pop ax                  ;把ax弹出栈
429     pop es                  ;把es弹出栈
430     sti                     ;设置IF标志，使芯片不屏蔽可屏蔽中断
431     ret                     ;返回
432 int33h endp                ;子程序定义结束

```

```

434 ;以下是输出的信息
435 message1_33h db 'Interrupt 33h success'
436 message1_33h_1 equ $-message1_33h
437 message2_33h db 'Coded by zongjiaxi'
438 message2_33h_1 equ $-message2_33h
439 message3_33h db '15352461'
440 message3_33h_1 equ $-message3_33h
441 interrupt33h proc          ;子程序定义伪指令
442     cli                    ;清除IF标志，使芯片屏蔽可屏蔽中断
443     push es                 ;依次将下列寄存器压栈，因为要使用到这些寄存器，保护原值
444     push si
445     push di
446     push ax
447     push bx
448     push cx
449     push dx
450     push bp
451     push ds
452
453     mov ax,cs               ;段地址储存
454     mov es,ax               ;放到es中
455     mov ax,1301h            ;功能号、光标
456     mov bx,0001h            ;颜色
457     mov dx,0408h            ;行，列
458     mov cx,message1_33h_1    ;长度
459     mov bp,offset message1_33h ;字符串起始位置
460     int 10h                 ;调用10号中断
461     ;以下基本相同
462     mov ax,cs
463     mov es,ax
464     mov ax,1301h
465     mov bx,0001h;bl颜色
466     mov dx,050dh;行，列
467     mov cx,message2_33h_1
468     mov bp,offset message2_33h
469     int 10h

```

```

470
471     mov ax,cs
472     mov es,ax
473     mov ax,1301h
474     mov bx,0001h;bl颜色
475     mov dx,0608h;行,列
476     mov cx,message3_33h_1
477     mov bp,offset message3_33h
478     int 10h
479
480     mov al,20h                ;把20h存到al寄存器中(al=EOI)
481     out 20h,al                ;发送EOI到主芯片
482     out 0a0h,al               ;发送EOI到从芯片
483
484     pop ds                    ;依次将下列寄存器弹出栈,恢复原值
485     pop bp
486     pop dx
487     pop cx
488     pop bx
489     pop ax
490     pop di
491     pop si
492     pop es
493     sti                      ;设置IF标志,使芯片不屏蔽可屏蔽中断
494     iret                    ;中断返回
495 interrupt33h endp

```

### 34 号中断

```

496
497 int34h proc
498     cli                      ;清除IF标志,使芯片屏蔽可屏蔽中断
499     push es                  ;把es压栈
500     push ax                  ;把ax压栈
501     xor ax,ax                ;把ax置0
502     mov es,ax                ;把ax的值存进es寄存器中
503     mov word ptr es:[0d0h],offset interrupt34h
504     mov ax,cs
505     mov word ptr es:[0d2h],ax
506     pop ax                   ;把ax弹出栈
507     pop es                   ;把es弹出栈
508     sti                      ;设置IF标志,使芯片不屏蔽可屏蔽中断
509     ret                      ;返回
510 int34h endp

```

```

511
512 message1_34h db 'Interrupt 34h success'
513 message1_34h_1 equ $-message1_34h
514 message2_34h db 'Coded by zhonglingshan'
515 message2_34h_1 equ $-message2_34h
516 message3_34h db '15352443'
517 message3_34h_1 equ $-message3_34h
518 interrupt34h proc
519     cli                                ;清除IF标志，使芯片屏蔽可屏蔽中断
520     push es                            ;依次将下列寄存器压栈，因为要使用到这些寄存器，保护原值
521     push si
522     push di
523     push ax
524     push bx
525     push cx
526     push dx
527     push bp
528     push ds
529
530     mov ax,cs
531     mov es,ax
532     mov ax,1301h
533     mov bx,0002h;b1颜色
534     mov dx,042eh;行, 列
535     mov cx,message1_34h_1
536     mov bp,offset message1_34h
537     int 10h

```

```

538
539     mov ax,cs
540     mov es,ax
541     mov ax,1301h
542     mov bx,0002h;b1颜色
543     mov dx,0534h;行, 列
544     mov cx,message2_34h_1
545     mov bp,offset message2_34h
546     int 10h
547
548     mov ax,cs
549     mov es,ax
550     mov ax,1301h
551     mov bx,0002h;b1颜色
552     mov dx,062eh;行, 列
553     mov cx,message3_34h_1
554     mov bp,offset message3_34h
555     int 10h
556
557     mov al,20h                        ;把20h存到al寄存器中(al=EOI)
558     out 20h,al                        ;发送EOI到主芯片
559     out 0a0h,al                       ;发送EOI到从芯片

```

```

560
561     pop ds                ;依次将下列寄存器弹出栈，恢复原值
562     pop bp
563     pop dx
564     pop cx
565     pop bx
566     pop ax
567     pop di
568     pop si
569     pop es
570     sti                  ;设置IF标志，使芯片不屏蔽可屏蔽中断
571     iret                 ;中断返回
572 interrupt34h endp

```

35 号中断：

```

573
574 int35h proc
575     cli                  ;清除IF标志，使芯片屏蔽可屏蔽中断
576     push es              ;把es压栈
577     push ax              ;把ax压栈
578     xor ax,ax            ;把ax置0
579     mov es,ax            ;把ax的值存进es寄存器中
580     mov word ptr es:[0d4h],offset interrupt35h
581     mov ax,cs
582     mov word ptr es:[0d6h],ax
583     pop ax               ;把ax弹出栈
584     pop es               ;把es弹出栈
585     sti                  ;设置IF标志，使芯片不屏蔽可屏蔽中断
586     ret                  ;返回
587 int35h endp

```

```

588
589 message1_35h db 'Interrupt 35h success'
590 message1_35h_1 equ $-message1_35h
591 message2_35h db 'Coded by zhouchancheng'
592 message2_35h_1 equ $-message2_35h
593 message3_35h db '15352448 I Love Girls'
594 message3_35h_1 equ $-message3_35h
595 interrupt35h proc
596     cli                  ;清除IF标志，使芯片屏蔽可屏蔽中断
597     push es              ;依次将下列寄存器压栈，因为要使用到这些寄存器，保护原值
598     push si
599     push di
600     push ax
601     push bx
602     push cx
603     push dx
604     push bp
605     push ds
606
607     mov ax,cs
608     mov es,ax
609     mov ax,1301h
610     mov bx,0003h;bl颜色
611     mov dx,1008h;行，列
612     mov cx,message1_35h_1
613     mov bp,offset message1_35h
614     int 10h

```

```

615
616     mov ax,cs
617     mov es,ax
618     mov ax,1301h
619     mov bx,0003h;b1颜色
620     mov dx,110dh;行,列
621     mov cx,message2_35h_1
622     mov bp,offset message2_35h
623     int 10h
624
625     mov ax,cs
626     mov es,ax
627     mov ax,1301h
628     mov bx,0003h;b1颜色
629     mov dx,1208h;行,列
630     mov cx,message3_35h_1
631     mov bp,offset message3_35h
632     int 10h
633
634     mov al,20h                ;把20h存到al寄存器中(al=EOI)
635     out 20h,al                ;发送EOI到主芯片
636     out 0a0h,al               ;发送EOI到从芯片
637
638     pop ds                    ;依次将下列寄存器弹出栈,恢复原值
639     pop bp
640     pop dx
641     pop cx
642     pop bx
643     pop ax
644     pop di
645     pop si
646     pop es
647     sti                      ;设置IF标志,使芯片不屏蔽可屏蔽中断
648     iret                    ;中断返回
649 interrupt35h endp

```

36号中断:

```

650
651 int36h proc
652     cli                      ;清除IF标志,使芯片屏蔽可屏蔽中断
653     push es                  ;把es压栈
654     push ax                  ;把ax压栈
655     xor ax,ax                ;把ax置0
656     mov es,ax                ;把ax的值存进es寄存器中
657     mov word ptr es:[0d8h],offset interrupt36h
658     mov ax,cs
659     mov word ptr es:[0dah],ax
660     pop ax                    ;把ax弹出栈
661     pop es                    ;把es弹出栈
662     sti                      ;设置IF标志,使芯片不屏蔽可屏蔽中断
663     ret                      ;返回
664 int36h endp

```

```

665
666 message1_36h db 'Interrupt 36h success'
667 message1_36h_1 equ $-message1_36h
668 message2_36h db 'Final show'
669 message2_36h_1 equ $-message2_36h
670 message3_36h db 'All success'
671 message3_36h_1 equ $-message3_36h
672 interrupt36h proc
673     cli                                ;清除IF标志，使芯片屏蔽可屏蔽中断
674     push es                            ;依次将下列寄存器压栈，因为要使用到这些寄存器，保护原值
675     push si
676     push di
677     push ax
678     push bx
679     push cx
680     push dx
681     push bp
682     push ds
683
684     mov ax,cs
685     mov es,ax
686     mov ax,1301h
687     mov bx,0004h;bl颜色
688     mov dx,102eh;行,列
689     mov cx,message1_36h_1
690     mov bp,offset message1_36h
691     int 10h
692
693     mov ax,cs
694     mov es,ax
695     mov ax,1301h
696     mov bx,0004h;bl颜色
697     mov dx,1134h;行,列
698     mov cx,message2_36h_1
699     mov bp,offset message2_36h
700     int 10h

```

```

701
702     mov ax,cs
703     mov es,ax
704     mov ax,1301h
705     mov bx,0004h;bl颜色
706     mov dx,122eh;行,列
707     mov cx,message3_36h_1
708     mov bp,offset message3_36h
709     int 10h
710
711     mov al,20h                        ;把20h存到al寄存器中(al=EOI)
712     out 20h,al                        ;发送EOI到主芯片
713     out 0a0h,al                      ;发送EOI到从芯片
714
715     pop ds                            ;依次将下列寄存器弹出栈，恢复原值
716     pop bp
717     pop dx
718     pop cx
719     pop bx
720     pop ax
721     pop di
722     pop si
723     pop es
724     sti                              ;设置IF标志，使芯片不屏蔽可屏蔽中断
725     iret                             ;中断返回
726 interrupt36h endp

```



至此，与 c 文件关联的汇编文件也完成了，这两个文件编译并关联后就是这次实验的“内核”了。

接下来，就要编写调用 33h，34h，35h，36h 四个中断程序的用户程序了。在上面的代码可以看出，这四个代码的功能是在屏幕的左上、左下、右上、右下的位置输出四个信息，用户程序在内核中调用。

代码如下：

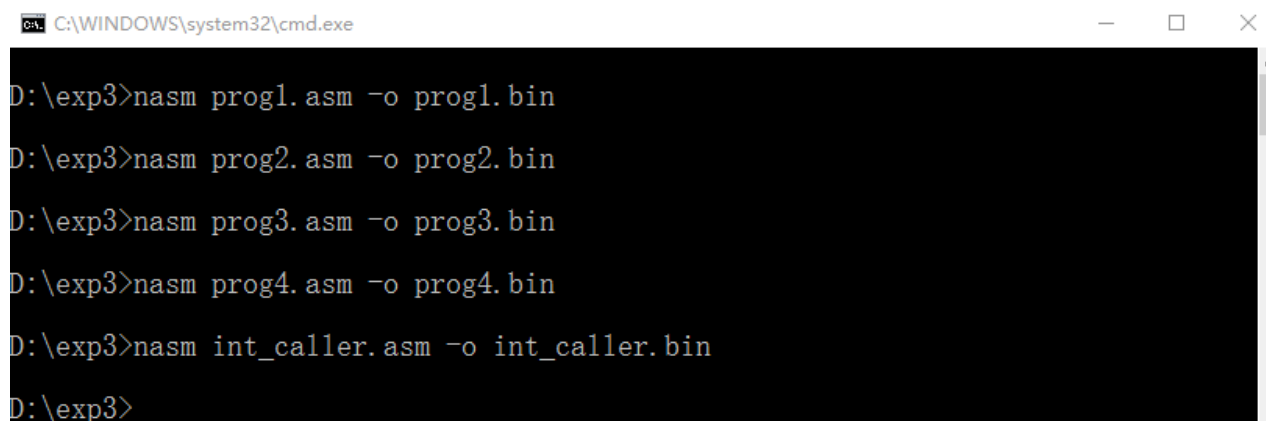
```
1  org 0b100h                ;程序偏移量为0b100h
2  Start:                    ;
3      mov ax,cs              ;把cs寄存器的值(代码段地址)存到ax寄存器中
4      mov ds,ax              ;把ax寄存器的值存到ds寄存器中
5      mov es,ax              ;把ax寄存器的值存到es寄存器中
6      int 33h                ;调用33号中断
7      int 34h                ;调用34号中断
8      int 35h                ;调用35号中断
9      int 36h                ;调用36号中断
10     mov cx,0aaffh           ;以下为延时
11 delay:
12     push cx
13     mov cx,0ffffh
14 delay2:
15     loop delay2
16     pop cx
17     loop delay
18     ret                    ;返回
19
20 times 1022-($-$$) db 0
21 db 0x55,0xaa
```

把这个用户程序编译成 bin 文件后和四个用户程序一起合并到 loader1.bin 中就完成了。

至此，所有的程序就已经完成了，真是一个很痛苦的过程！

接下来就是编译的过程了，跟实验 3 的时候一样的编译方法：

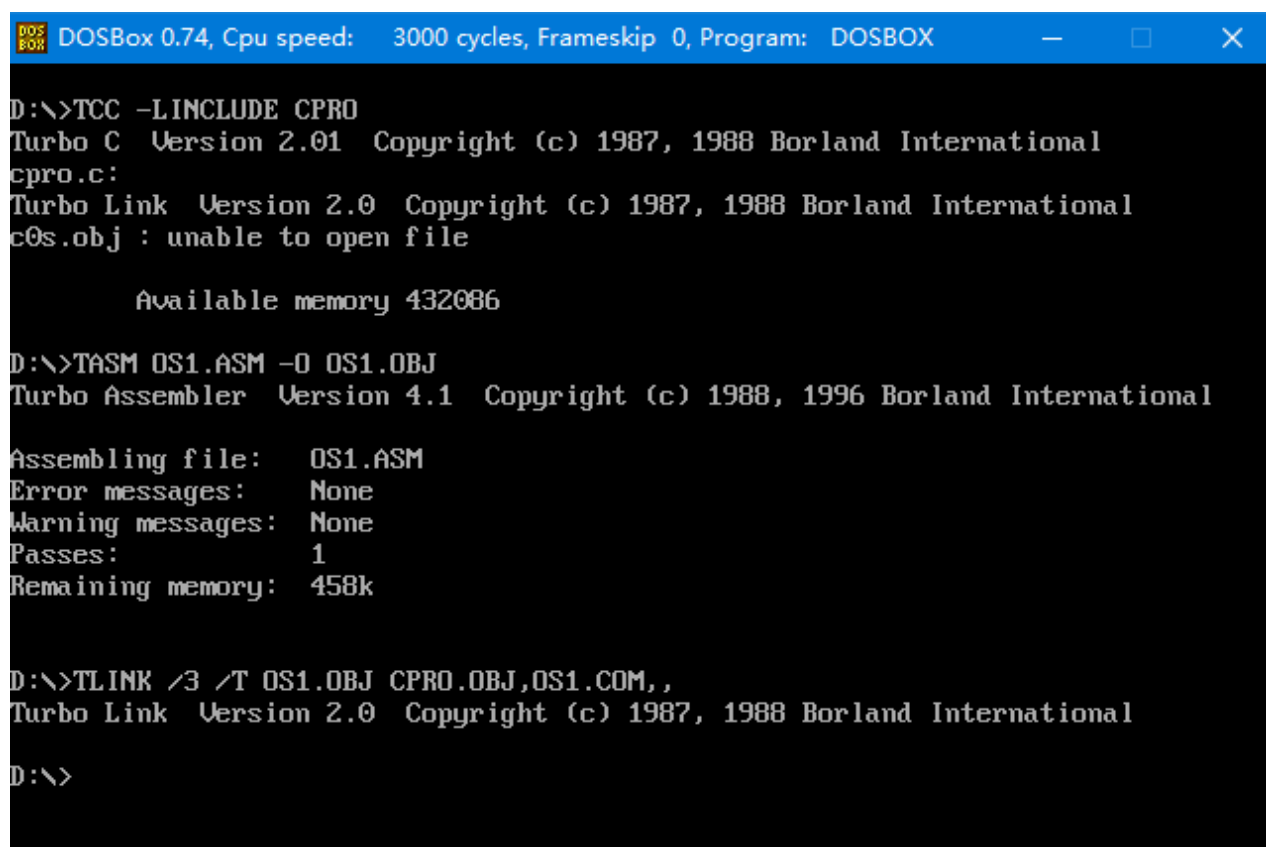
首先使用 nasm 编译出 5 个用户程序：



```
C:\WINDOWS\system32\cmd.exe

D:\exp3>nasm prog1.asm -o prog1.bin
D:\exp3>nasm prog2.asm -o prog2.bin
D:\exp3>nasm prog3.asm -o prog3.bin
D:\exp3>nasm prog4.asm -o prog4.bin
D:\exp3>nasm int_caller.asm -o int_caller.bin
D:\exp3>
```

然后使用 DOSBOX 编译内核的汇编文件和 c 文件，再连接起来：



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

D:\>TCC -LINCLUDE CPRO
Turbo C Version 2.01 Copyright (c) 1987, 1988 Borland International
cpro.c:
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International
c0s.obj : unable to open file

        Available memory 432086

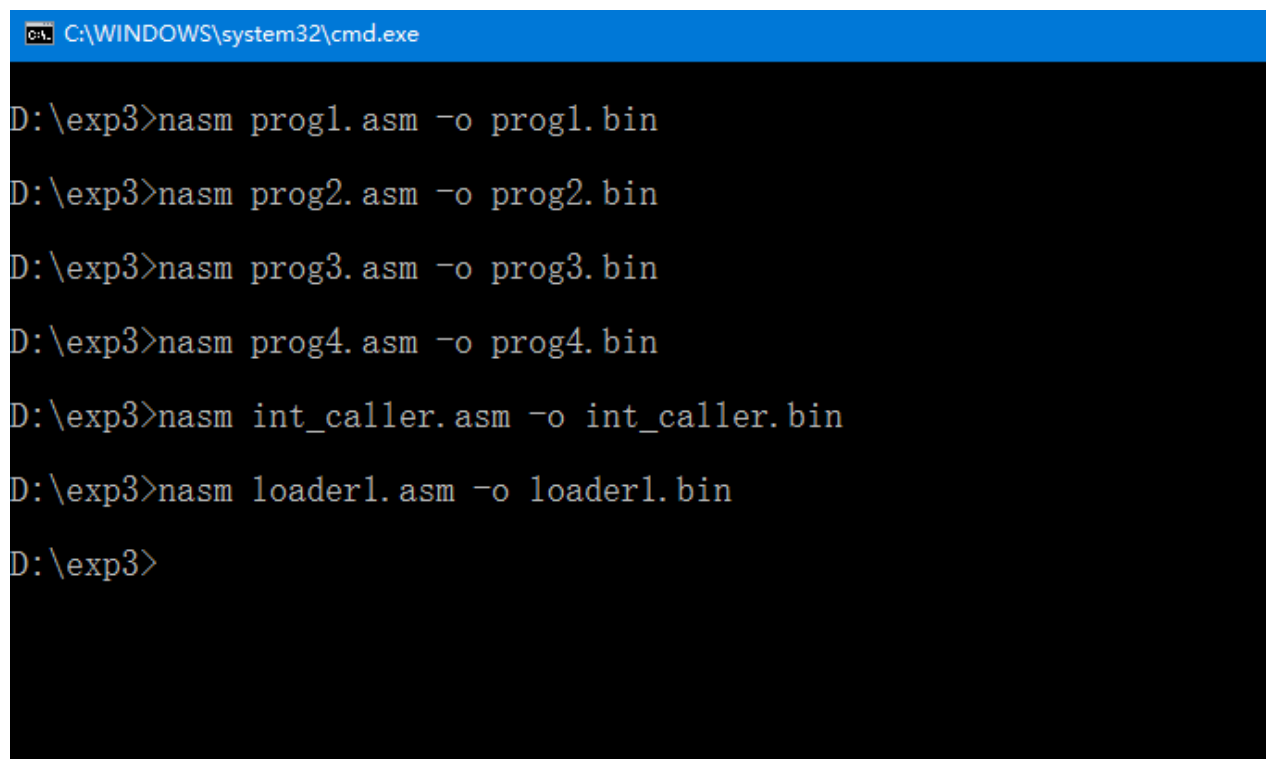
D:\>TASM OS1.ASM -O OS1.OBJ
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file:  OS1.ASM
Error messages:  None
Warning messages:  None
Passes: 1
Remaining memory: 458k

D:\>TLINK /3 /T OS1.OBJ CPRO.OBJ,OS1.COM,,
Turbo Link Version 2.0 Copyright (c) 1987, 1988 Borland International

D:\>
```

最后再编译引导程序：



```
C:\WINDOWS\system32\cmd.exe

D:\exp3>nasm prog1.asm -o prog1.bin
D:\exp3>nasm prog2.asm -o prog2.bin
D:\exp3>nasm prog3.asm -o prog3.bin
D:\exp3>nasm prog4.asm -o prog4.bin
D:\exp3>nasm int_caller.asm -o int_caller.bin
D:\exp3>nasm loader1.asm -o loader1.bin
D:\exp3>
```

```

Z
ZWelcome to our team's system!
ZYou can input some legal instructions to active the functions
Z    If you want to run the program by Batching,input 'batch'
Z        For example: batch 1 2 3 4
Z    If you want to run the program by Time-Sharing,input 'time'
Z        For example: time 1 2 3 4
Z    If you want to know the information of the program,input 'show'
Z        For example: show
Z    If you want to run the default instructions,input 'default'
Z        For example: default
Z    If you want to run the interrupt test program,input 'interrupt'
Z        For example: interrupt
Z
ZOrz >_
Z
Z
Z

```

OS Running!

然后输入 batch 1 2 3 4 指令，批处理实现用户程序的调用，在进入后按下键盘，会发现屏幕有 "Ouch!" 字样出现：

- 23 -



- 24 -

[illegible]

[illegible]



输入 interrupt ,运行调用中断的用户程序,可以看到在屏幕的四个位置都显示了一些我们小组成员的信息,而且按下键盘按键还能打印 Ouch!,这个中断程序的延时大概为 8 秒左右:

```

Welcome to our team's system!
You can input some legal instructions to active the functions
  If you want to run the program by Batching,input 'batch'
    For example: batch 1 2 3 4
  If you want to run the program by Time-Sharing,input 'time'
    For example: time 1 2 3 4
  If you want to know the information of the program,input 'show'
    For example: show
  If you want to run the default instructions,input 'default'
    For example: default
  If you want to run the interrupt test program,input 'interrupt'
    For example: interrupt

Orz >interrupt_

OS Running/

Ouch!Ouch!Ouch!Ouch!Ouch!_

Interrupt 33h success
      Coded by zongjiayi
15352461

Interrupt 34h success
      Coded by zhonglingshan
15352443

Interrupt 35h success
      Coded by zhouchancheng
15352448 I Love Girls

Interrupt 36h success
      Final show
All success
```

至此,本次实验的程序功能已经全部展示完毕。

## 【实验总结】

**宗嘉希 15352461**：首先吧，这一次的实验花费了我挺多的时间，比前三次的实验所花费的时间都要多，因为我没有去系统地学习 x86 汇编，因此对于内核、引导程序、用户程序等等在内存中是如何分配空间的，因此在本次的实验中遇到了很多困难并且花费了很多的时间去解决。现在就在此说一下遇到的困难吧：首先，老师给了我们一个能在屏幕中间利用中断去打印 ascii 码字符的程序，但是我不明白为什么在我的虚拟机上无法运行这一个程序，运行的结果就是只在屏幕中央显示初始化时的数字，中断过程完全没有，这个问题我烦恼了很久，大概搞了一周才发现原来是我的内存位置没有设置好，老师给的代码里面写的是 org 100h，就是把代码定位到偏移量为 100h 的内存位置，然而我通过查阅资料发现，应该是 7c00h，才可以让中断正常运行，因为 7c00h 的内存位置是引导程序的起始位置。所以，我就解决了这个一开始就遇到的大问题。后来，我把中断加入到了内核中，也就是 c 语言和汇编语言组成的内核中，但是又有一个新的问题出现了，就是这个中断它不会在内核中正常运行，这个也是花费了我很多的时间去找解决的方法（没办法，汇编语言没学好），后来才发现原来引导程序必须要放在 org 8100h 的位置。后来，在实验快要完成的时候又有一个新问题，就是无法正常运行用户程序，好像卡死了一样，这个只花了我一天就找出问题来了，因为一开始的时候我是直接使用我自己编写的实验三的源代码去添加终端的内容来做实验四的，但是实验三的代码中，复杂度比不上实验四的，因此，内核占据的空间变大

了，变成了 14 个扇区了，但是原来读取的扇区数只有 8 个，因此会出现读取错误，无法正常运行用户程序。还有一些小的问题，比如说现实用户程序信息的时候，发现屏幕什么字符串都没有了，原来是因为刚好字符串太多，到了下一页的开头，什么都看不到，因此在现实之前先输出几行字符串，把页面压到第二页再显示信息就可以了（挺愚蠢的方法）。当所有问题都解决了之后实验也就完成了，真的感到非常的高兴，虽然很累，但是心中有了很舒缓的感觉。这一次的实验让我发现了在汇编中对内存的掌握是多末的重要。希望能在以后的操作系统课上学习到更多的东西！

**钟凌山 15352443：**这次的操作系统实验的主要内容是进一步了解系统中断的运行机制、分类和调用方法，并用系统中断功能实现操作系统中的一些复杂功能。本次实验主要用到了时钟中断和键盘输入中断，前者实现操作系统默认状态中和用户程序中字母在屏幕上的滚动显示，以及表示操作系统正在正常运行的“-”、“|”等字符的循环显示；后者则用户程序运行时，在键盘有输入的情况下在屏幕上方依次打出“Ouch!”字样。

这些功能虽然只是上次实验的改版，但是却依然耗费了我们大量的时间去编写和调试，主要的原因还是因为对系统中断不是非常熟悉。

**周禅城 15352448：**这次实验真的很难很难啊！！！！实验 4 之难，难于上青天！这次实验主要是设置中断与编写中断服务程序，主要指时钟中断与键盘中断。这次实验的过程可谓是一路坎坷。首先，在认真听了老师的理论和实验课之后，我们认为我们对中断的理解应该还是比较透彻了，实验应该不会太难完成。可是，残酷的现实啊，我们还是太天真。首先，我们想跑一跑老师 ppt 上的代码，了解一下基本框架，但是一开始就遇到了玄学问题。我们将老师的代码编译后在 DOSBOX 上能跑，到了 VMware 上中断就不发挥作用，整个操作系统卡在中断那里不动。对于这个问题，我们小组讨论了两三天也不知道怎么解决。最后不得已求助老师。老师说那是因为 VMware 没有初始化中断芯片，然后给了我们一段初始化中断芯片的代码。我们把这段代码加进去之后，发现竟然还是不能跑！！！！不能跑！！！！真的是不

努力一下，你都不知道什么叫做绝望。最后经过我们的通力合作，花了大概一周时间，终于发现是内存存放位置的问题，剩下的就比较好解决了。真的不得不说。关键时候还是要靠自己学习，希望我们以后能越做越好吧！

附录：

### **源代码：**

用户程序：

prog1.asm

prog2.asm

prog3.asm

prog4.asm

int\_caller.asm

内核代码：

os1.asm

cpro.c

引导扇区代码：

loader1.asm

### **编译文件：**

prog1.bin

prog2.bin

prog3.bin

prog4.bin

int\_caller.bin

os1.obj

cpro.obj

os1.com

loader1.bin

### **镜像文件：**

144mb.img