

1.求2^3

```
1  assume cs:code
2
3  code segment
4      mov ax,2
5      add ax,ax
6      add ax,ax
7
8      mov ax,4c00h
9      int 21h
10 code ends
11 end
```

2.loop 标号

- 每次执行到loop指令的时候，都去看一下cx寄存器中的值是否为0 如果为0，就向下继续执行其他的命令，如果不是，就返回到标号的位置

```
1  assume cs:code
2
3  code segment
4      mov ax,2
5      mov cx,11
6      s:add ax,ax
7      loop s
8      mov ax,4c00h
9      int 21h
10 code ends
11 end
```

3.把dw 1h,2h,3h,4h定义的数据，逆序存放

- 思路解决 想把数据存放到栈中，然后在利用栈的特性先进后出，把数据弹出 要给栈段设置存储空间（就直接在代码段设置一个空的占用空间）

```
1  assume cs:code
2
3  code segment
4      dw 1h,2h,3h,4h
5      dw 8 dup(0)
6      start:
7          mov bx,0
8          mov cx,4
9          mov ax,cs
10         mov ss,ax
11         mov sp,20h
12
```

```

13
14     s:push cs:[bx] ;把数据压栈
15     add bx,2 ;因为操作的是字，所以需要加2
16     loop s
17
18     mov cx,4
19     mov bx,0
20     s2:pop cs:[bx]
21     add bx,2
22     loop s2
23
24
25
26     mov ax,4c00h
27     int 21h
28
29 code ends
30 end start

```

4.把数据段的adcd数据转换成大写的

```

1  assume cs:code,ds:data
2  data segment
3      db 'abcd'
4  data ends
5
6  code segment
7  start:mov ax,data
8          mov ds,ax ;给数据段赋值
9          mov cx,5
10         mov bx,0
11         s:mov al,[bx] ;把数据段的数据取出来
12         and al,11011111B ;al和11011111B做与运算，运算的结果赋值给al
13         mov [bx],al
14         inc bx
15         loop s
16         mov ax,4c00h
17         int 21h
18     code ends
19     end star

```

5.字母转换成大写的

- 实现中出现的问题 二重循环都需要cx作为计数器，但是内部循环改变cx后，外重循环就乱套了 解决 在栈中保存外层循环的数据，需要的时候栈取出

```

1  assume cs:code,ss:stack
2
3  stack segment
4      db 0,0,0,0,0,0,0,0

```

```

5  stack ends
6
7  code segment
8      db 'abc'
9      db 'def'
10     start:
11         mov ax,stack
12         mov ss,ax
13         mov sp,8
14         mov dx,0
15
16
17         mov cx,2
18     s1:push cx ;把cx计数器数据保存到栈中
19
20         mov cx,3
21         mov bx,0
22     s:mov al,cs:[dx+bx]
23         and al,11011111b
24         mov cs:[dx+bx],al
25         inc bx
26         loop s
27
28         add dx,16
29         pop cx ;弹出栈中的数据,赋值给cx
30         loop s1
31
32         mov ax,4c00h
33         int 21h
34
35
36 code ends
37 end start

```

6.程序在运行的时候，将s处的指令复制到s0处

```

1  assume cs:code
2  code segment
3      s:mov,ax
4      mov si,offset s
5      mov di,offset s0
6      mov ax,cs:[si]
7      mov cs:[di],ax ;应为cs:[di],对应数据段,所以必须使用ax作为中间数据,赋值
8      s0:nop
9      nop
10 code ends
11 end

```

7.用栈来传递参数

- 把参数放到栈中，需要的时候，利用bp（这个寄存器默认是ss），去除栈中保存的数据 问题 把dw定义的数的3次方，放到dd定义的数据中

```

1  assume cs:code,ds:data
2
3  data segment
4      dw 1,2,3,4,5,6,7,8
5      dd 0,0,0,0,0,0,0,0
6  data ends
7  stack segment
8      db 16 dup(0)
9  stack ends
10
11 code segment
12     mov si,0
13     mov di,16
14     mov cx,8 ;需要循环8次
15
16     mov ax,stack
17     mov ss,ax
18     mov sp,16;初始化栈段
19
20     mov ax,data
21     mov ds,ax ;初始化数据段
22
23     s:push [si] ;参数压栈, 等待cube调用这个参数
24     call cube
25
26
27
28
29
30     add si,2 ;定义的数据是字, 所以计算一个后加2
31     add di,4 ;定义的数据是双字, 所以计算一个后加4
32     loop s
33     cube:
34     push bp ;把bp暂时征用, 保存bp之前的值
35     mov ax,[sp+4] ;获取栈中保存的数据, sp默认 ss:[bp],又因为call和push bp, 都用了栈。
36                     ;所以要从栈中的第4个数据才是传递的参数
37     mul bp
38     mul bp
39
40     mov [di],dx ;高位的数据
41     mov [di+2],ax ;低位的数据
42     pop bp ;归还bp
43     ret 4
44
45     mov ax,4c00h
46     int 21h
47
48 code ends
49 end start

```

8.完成128位数的相加

```

1  assume cs:code,ds:data
2
3  data segment
4      dw 1h,2h,3h,4h,5h,6h,7h,8h
5      dw 1h,2h,3h,4h,5h,6h,7h,8h
6  data ends
7  stack segment
8      db 16 dup(0)
9  stack ends
10
11 code segment
12     start:
13         mov si,0
14         mov di,16
15
16
17         mov ax,data
18         mov ds,ax;初始化数据段
19
20         mov ax,stack
21         mov ss,stack
22         mov sp,16;初始化栈段
23
24
25
26
27         call add128
28
29         mov ax,4c00h
30         int 21h
31
32     add128:
33         push cx
34         push ax
35         push si
36         push di
37
38         mov cx,8
39         sub ax,ax ;把ax归0,也把CF归0,不能用mov,ax,0,这样不能保证CF是0
40
41         mov ax,[si]
42         add [di],ax
43
44     s:inc si
45         inc si
46         inc di ;不能用 add di,2这样就可能破坏了CF原有的值,而inc不会影响CF
47         inc di
48
49         mov ax,[si]
50         adc [di],ax
51         loop s
52
53         pop di

```

```

54     pop si
55     pop ax
56     pop cx
57
58     ret
59
60
61
62
63 code ends
64 end start

```

9.找出data数据中，等于8的个数

```

1  assume cs:code,ds:data
2
3  data segment
4      db 1,2,3,4,5,6,7,8
5  data ends
6  stack segment
7      db 16 dup(0)
8  stack ends
9
10 code segment
11     start:
12         mov ax,data
13         mov ds,ax
14
15
16     •   mov ax,stack
17     •   mov ss,ax
18     •   mov sp,16
19     •   mov ax,0
20     •
21     •   mov si,0
22     •   mov cx,8
23     •
24     •   s:cmp byte ptr [si],8
25     •
26     •   jne next
27     •   inc ax
28     •   next:
29     •   inc si
30     •   loop s
31     •
32     •   mov ax,4c00h
33     •   int 21h
34
35 code ends
36 end start

```

10.直接操控显存，在屏幕正中央显示 白底蓝字welcome to masm!

```

1  assume cs:code,ds:data
2
3  data segment
4      db 'welcome to masm!'
5  data ends
6
7  code segment
8      start:
9          mov ax,data
10         mov ds,ax
11
12
13     •      mov ax,0B800H ;显存首地址
14     •      mov ss,ax;让ss指向显存的首地址
15
16     •
17
18     •      mov cx,16
19     •      mov si,0
20     •      mov bp,1980 ;显存总共25行，每行160字节。12*160+80-16=1980。(显存显示一个字符需要一
    个字的大小)
21     •      s:mov al,[si] ;把字符送入低位
22     •      mov ah,71h ;把字符属性送入高位。71h, 代表白底蓝字
23     •      mov [bp],ax
24     •      add bp,2
25     •      add si,1
26     •      loop s
27     •
28     •      mov ax,4c00h
29     •      int 21h
30
31 code ends
32 end start

```

11.在屏幕中，把数据2B用字符显示出来

- 思想 先定义一个0-E的字符表 然后把当前的数据作为偏移量，去表中查找

```

1  assume cs:code
2  code segment
3      table db '0123456789ABCDE'
4
5  start:
6      call showbyte
7
8  mov ax,4c00h
9  int 21h
10 showbyte:
11 mov al,2bh
12 mov ah,al
13 mov cl,4
14 shr ah,cl ;右移4位，只保留高4位的数据

```

```

15 and al,00001111b;只保留低4位的数据
16
17 mov bh,0
18 mov bl,ah
19 mov ah,table[bx] ;把数字2变成字符2
20
21 •
22
23 mov bl,al
24 mov bh,0
25 mov al,table[bx] ;把数字B变成字符B
26
27 mov bx,0B800H ;显存首地址
28 mov es,bx;让ss指向显存的首地址
29
30 mov bh,71h
31 mov bl,ah
32
33 mov es:[1980],bx
34 mov bl,al
35 mov es:[1982],bx
36 ret
37
38 •
39 code ends
40 end start

```

12.除法错误，运行自己编写的中断处理程序

```

1  assume cs:code
2
3  code segment
4      start:
5      ;安装程序
6      mov si,offset do0
7      mov ax,cs
8      mov ds,ax
9
10     mov di,200h
11     mov ax,0
12     mov es,ax
13
14
15
16     mov cx,offset do0end-offset do0
17     cld
18     rep movsb
19     ;设置中断表的数据
20     mov ax,0
21     mov es,ax
22     mov ax,200h
23     mov es:[0],ax

```



```

24     mov ax,0
25     mov es:[2],ax
26
27     ;触发触发中断
28     mov ax,1000
29     mov bh,0
30     div bh
31     mov ax,4c00h
32     int 21h
33
34     ;除法中断程序，完成显示overflow!
35     do0: jmp do0start
36     db 'Overflow!';不能把定义的数据当做指令执行了
37     do0start:
38     mov si,202h
39     mov ax,0b800h
40     mov es,ax
41
42     • mov di,1980
43     • mov cx,9
44     •
45     • s: mov al,cs:[si]
46     • mov es:[di],al
47     • inc si
48     • add di,2
49     • loop s
50     • mov ax,4c00h
51     • int 21h
52     do0end: nop
53     code ends
54
55     end start

```

13.出现int 7ch中断，然后执行我自己的程序。最终在屏幕显示 HELLO

```

1  assume cs:code,ds:data
2  data segment
3      db 'hello',0
4  data ends
5
6  code segment
7      start:
8      ;安装程序
9      mov ax,cs
10     mov ds,ax
11     mov si,offset do0
12
13
14     mov ax,0
15     mov es,ax
16     mov ax,200h
17     mov di,ax

```

```

18
19 mov cx,offset do0end-offset do0
20 cld
21 rep movsb
22 ;设置中断表的数据
23 mov ax,0
24 mov es,ax
25 mov ax,200h
26 mov es:[7ch*4],ax
27 mov ax,0
28 mov es:[7ch*4+2],ax
29
30 int 7ch
31
32 mov ax,4c00h
33 int 21h
34
35 do0:
36     push cx
37     push bx
38     push ds
39     push es
40     push si
41     push di
42
43
44 mov bx,0
45 mov si,bx
46 mov bx,1980
47 mov di,bx
48 mov bx,0B800H
49 mov es,bx
50 mov bx,data
51 mov ds,bx
52
53 s:mov ah,0
54 mov cl,[si]
55 jcxz change
56 and byte ptr [si],11011111b
57 mov bl,[si]
58 mov es:[di],bl
59 add di,2
60
61 inc si
62 jmp s
63
64 change:
65     pop di
66     pop si
67     pop es
68     pop ds
69     pop bx
70     pop cx

```

```

71     iret
72 do0end:nop
73 code ends
74
75 end start

```

14. 获取系统时间

<p>INT 20H 一终止程序运行</p> <p>INT 21H 一功能调用</p> <p>INT 22H 一终止处理程序的地址</p> <p>INT 23H 一Ctrl+C 处理程序</p> <p>INT 24H 一致命错误处理程序</p> <p>INT 25H 一读磁盘扇区(忽略逻辑结构)</p> <p>INT 26H 一写磁盘扇区(忽略逻辑结构)</p> <p>INT 27H 一终止, 并驻留在内存</p> <p>INT 28H 一DOS 空闲</p> <p>INT 2FH 一多重中断服务</p> <p>INT 33H 一鼠标功能</p>	<p>DOS 中断类 (INT 21H 中断)</p> <ol style="list-style-type: none"> 1、字符功能调用类(Character-Oriented Function) 2、目录控制功能(Directory-Control Function) 3、磁盘管理功能(Disk-Management Function) 4、文件操作功能(File Operation Function) 5、文件操作功能(FCB)(File Operation Function) 6、记录操作功能(Record Function) 7、记录操作功能(FCB)(Record Function) 8、内存分配功能(Memory-Allocation Function) 9、系统功能(System Function) 10、进程控制功能(Process-Control Function) 11、时间和日期功能(Time and Date Function) 	<p>功能 2AH</p> <p>功能描述: 取系统日期</p> <p>入口参数: AH = 2AH</p> <p>出口参数: CX = 年 (1980~2099), DH = 月 (1~12), DL = 日 (1~31) AL = 星期几 (0=Sun, 1=Mon. ...)</p>
--	--	---

```

1  assume cs:code
2
3  code segment
4  start:
5      mov ah,2Ah
6      int 21h
7
8      nop
9      mov ax,4c00h
10     int 21h
11 code ends
12 end start

```

- 执行结果

```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
AX=2AFF BX=0000 CX=000A DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0002  NV UP EI PL NZ NA PO NC
076A:0002 CD21          INT     21
-t

AX=2AFF BX=0000 CX=000A DX=0000 SP=FFFA BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=F000 IP=14A0  NV UP DI PL NZ NA PO NC
F000:14A0 FB          STI
-t

AX=2AFF BX=0000 CX=000A DX=0000 SP=FFFA BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=F000 IP=14A1  NV UP EI PL NZ NA PO NC
F000:14A1 FE38          ???    [BX+SI]    DS:0000=CD
-t

AX=2A01 BX=0000 CX=07E5 DX=0104 SP=FFFA BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=F000 IP=14A5  NV UP EI PL NZ NA PO NC
F000:14A5 CF          IRET
-t

AX=2A01 BX=0000 CX=07E5 DX=0104 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0004  NV UP EI PL NZ NA PO NC
076A:0004 90          NOP

```

15.访问当前月份

背景知识

当前时间在CMOS RAM中用6个字节存放

内容	秒		分		时			日	月	年
地址	00	01	02	03	04	05	06	07	08	09

(这6个信息的长度都为1个字节)

例：今天5月15日

1	5	0	5
0010	0101	0000	0101

时间信息用BCD码存放

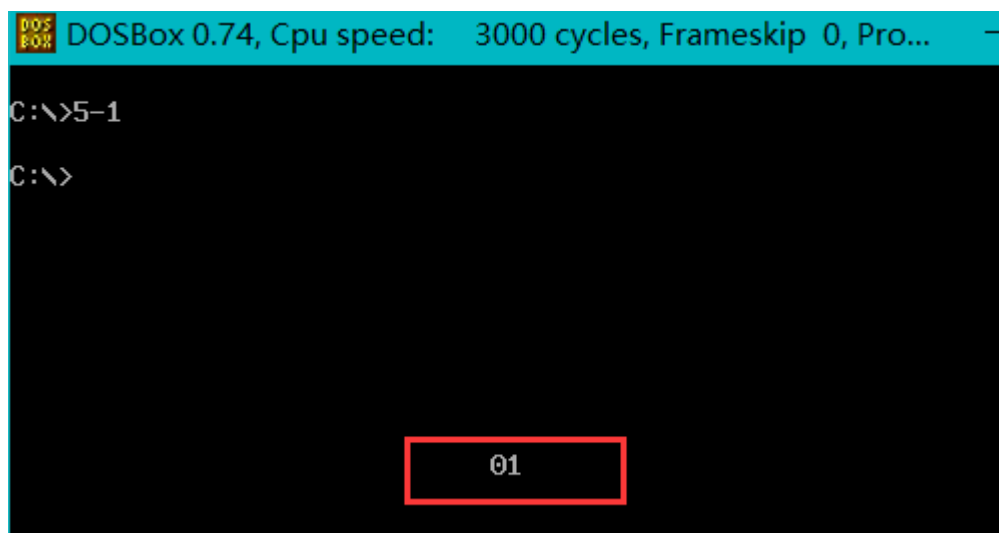
数码： 0 1 2 3 4 5 6 7 8 9

BCD码：0000 0001 0010 0011 0100 0101 0110 0111 1000 1001

```

1  assume cs:code
2
3  code segment
4  start:
5      ;需要读取8号地址的信息
6      mov al,8
7      out 70h,al
8      in al,71h
9
10     ;把十位和各位信息分开
11     mov ah,al
12     mov cl,4
13     shr ah,cl
14     and al,00001111b
15     ;转换为ASCII码
16     add ah,30h
17     add al,30h
18     ;操作显存, 显示信息
19     mov bx,0B800H
20     mov es,bx
21     mov byte ptr es:[1980],ah
22     mov byte ptr es:[1982],al
23
24     mov ax,4c00h
25     int 21h
26
27 code ends
28 end start

```



16.屏幕从a-z依次显示，按下esc就改变颜色

```

1  assume cs:code,ss:stack,ds:data
2
3  data segment
4      dw 0,0
5  data ends
6  stack segment

```

```

7      db 128 dup(0)
8  stack ends
9  code segment
10 start:
11      mov ax,stack
12      mov ss,ax
13      mov sp,128
14      mov ax,data
15      mov ds,ax
16
17      ;改变9号中断例程入口
18      mov ax,0
19      mov es,ax
20      push es:[9*4]
21      pop ds:[0]
22      push es:[9*4+2]
23      pop ds:[2]
24      ;保存, 9号中断例程的地址信息后, 转到自己的中断例程中
25      mov word ptr es:[9*4],offset int9
26      mov es:[9*4+2],cs
27
28
29
30      ;显示a~z
31      mov ax,0B800H
32      mov es,ax
33      mov ah,'a'
34      s:mov es:[160*12+40*2],ah
35      call delay
36      inc ah
37      cmp ah,'z'
38      jna s
39
40      ;回复原来中断例程的地址
41      mov ax,0
42      mov es,ax
43      push ds:[0]
44      pop es:[9*4]
45      push ds:[2]
46      pop es:[9*4+2]
47
48      mov ax,4c00h
49      int 21h
50
51      ;按esc切换颜色,这个中断例程,不是在代码中调用的。他是发生外部中断时候调用的。(键盘敲入等
   时候, 调用)
52      int9:
53      push ax
54      push bx
55      push es
56
57      in al,60h;读取当前键入扫描码
58

```

```

59      ;模拟调用int 9中断。
60      pushf
61      pushf
62      pop  bx
63      and  ah,11111100b;修改IF,TF值为0
64      push  bx
65      popf
66      call dword ptr ds:[0];call相当于把cs,ip入栈的操作做了
67
68      cmp  al,1  ;esc的扫描码是1, 查看是不是esc
69      jna  int9end
70      mov  bx,0B800H
71      mov  es,bx
72      inc  byte ptr es:[160*12+80+1]
73  int9end:
74      pop  es
75      pop  bx
76      pop  ax
77      iret
78
79
80
81      ;延迟效果
82  delay:
83      push  cx
84      push  bx
85      mov  cx,0FFFFH
86      mov  bx,3
87  s2:nop
88      cmp  cx,1
89      je  setcx
90      loop s2
91
92  setcx:
93      cmp  bx,1
94      je  delayend
95      mov  cx,0FFFFH
96      dec  bx
97      jmp  s2
98  delayend:pop  bx
99      pop  cx
100
101      ret
102      mov  ax,4c00h
103      int  21h
104
105  code ends
106  end start

```

17.修改int 9中断例程，按下f1切换屏幕字体颜色

```

1  assume cs:code,ss:stack,ds:data

```

```

2 ;修改int 9中断例程, 按下f1切换屏幕字体颜色
3 data segment
4     dw 0,0
5 data ends
6 stack segment
7     db 128 dup(0)
8 stack ends
9 code segment
10 start:
11     mov ax,stack
12     mov ss,ax
13     mov sp,128
14     mov ax,data
15     mov ds,ax
16     ;安装程序
17     mov ax,cs
18     mov ds,ax
19     mov ax,offset int9
20     mov si,ax
21
22     mov ax,0
23     mov es,ax
24     mov ax,204h
25     mov di,ax
26
27     mov cx,offset int9end-offset int9
28     rep movsb
29
30     ;保存原来程序的地址,并且更新新的中断例程入口
31     cli
32     mov ax,0
33     mov es,ax
34     push es:[9*4]
35     pop es:[200h]
36     push es:[9*4+2]
37     pop es:[202h]
38
39     mov word ptr es:[9*4],200h
40     mov word ptr es:[9*4+2],0
41     sti
42     mov ax,4c00h
43     int 21h
44
45
46
47     ;修改int9中断例程
48     int9:
49     push ax
50     push bx
51     push cx
52     push es
53
54     pushf

```

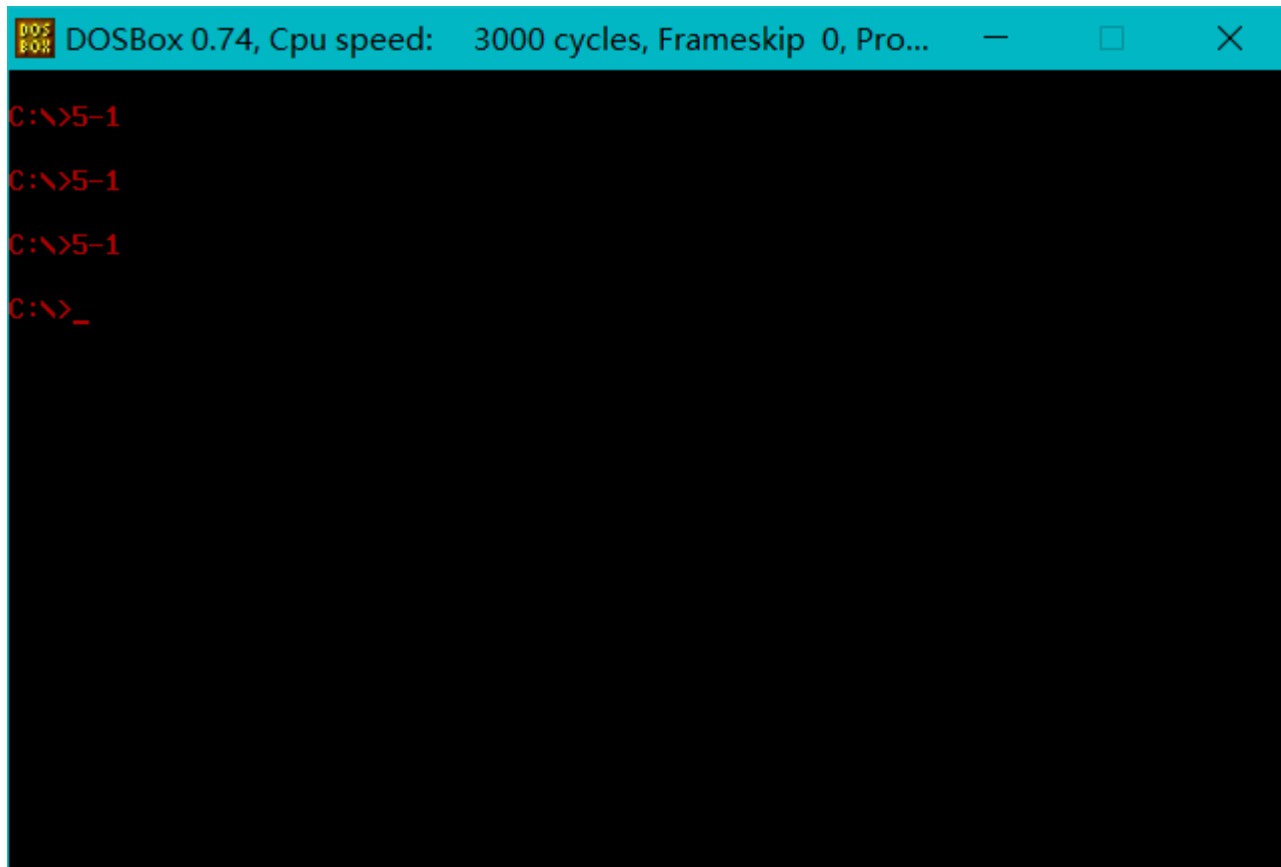


```

55     pushf
56     pop  bx
57     and  bh,11111100b
58     push bx
59     popf
60
61     mov  ax,0
62     mov  es,ax
63     in  al,60h;保存键入的扫描码
64
65
66
67     call dword ptr es:[200h]
68
69     cmp  al,3bh
70     jne  done
71     mov  ax,0B800H
72     mov  es,ax
73     mov  bx,1
74     mov  cx,160*25
75  s:
76     inc  byte ptr es:[bx]
77     add  bx,2
78     loop s
79  done:pop  es
80     pop  cx
81     pop  bx
82     pop  ax
83     iret
84  int9end:nop
85
86
87
88
89
90
91  code ends
92  end start

```

18.实现，按下 r,g,b键，屏幕字体颜色变红，绿，蓝



```
1  assume cs:code,ss:stack,ds:data
2  data segment
3      dw 0,0
4  data ends
5  stack segment
6      db 128 dup(0)
7  stack ends
8  code segment
9  start:
10     mov ax,stack
11     mov ss,ax
12     mov sp,128
13     mov ax,data
14     mov ds,ax
15
16     mov ah,0
17     int 16h
18
19     mov ah,1
20     cmp al,'r'
21     je red
22     cmp al,'g'
23     je green
24     cmp al,'b'
25     je blue
26     jmp sret
27
28     red:shl ah,1
```

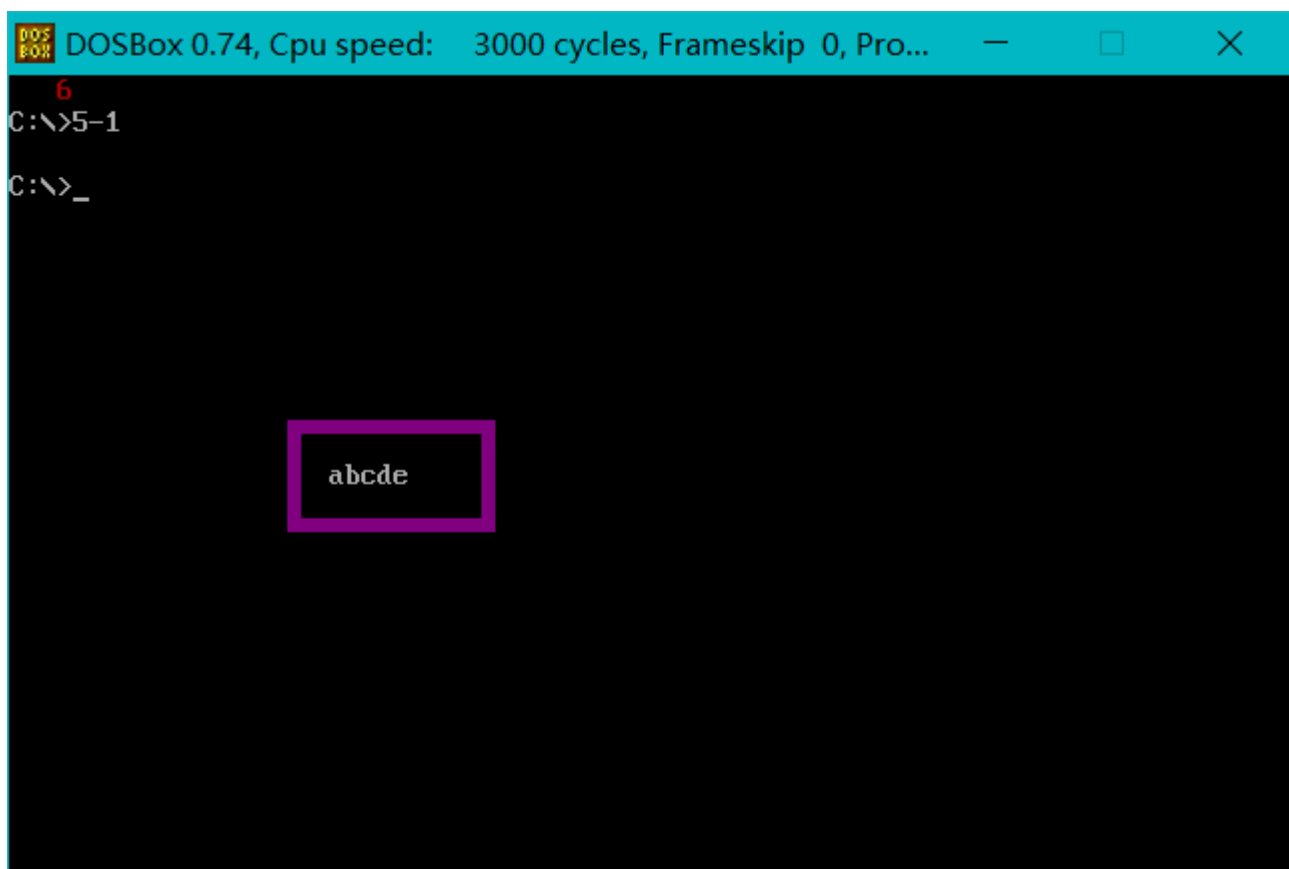
```

29     green:shl ah,1
30     blue:mov bx,0B800H
31     mov es,bx
32     mov bx,1
33     mov cx,160*25
34         ;先把RGB这三个设置为0
35     s:and byte ptr es:[bx],11111000b
36         ;把自己设置的颜色，赋值上去
37     or byte ptr es:[bx],ah
38     add bx,2
39     loop s
40
41
42     sret:mov ax,4c00h
43     int 21h
44
45
46
47
48
49 code ends
50 end start

```

19.字符串的输入

- 按回车键：删除一个字符
- 按字符键：输入一个字符
- 按回车键：字符输入结束，在字符最后添加一个0，然后显示字符



```

1  assume cs:code,ss:stack,ds:data
2
3  data segment
4      db 32 dup(0)
5      top db 0
6  data ends
7  stack segment
8      db 128 dup(0)
9  stack ends
10 code segment
11 start:
12     mov ax,stack
13     mov ss,ax
14     mov sp,128
15     mov ax,data
16     mov ds,ax
17
18     mov si,0
19     mov dh,12
20     mov dl,20
21     call getstr
22     return:
23     mov ax,4c00h
24     int 21h
25
26 getstr:
27     push ax
28 getstrs:
29     mov ah,0
30     int 16h
31
32     cmp al,20h;获取到的字符（不用扫描码），查看是不是字符
33     jb nochar
34
35     mov ah,0
36     call charstack;是字符，就调用函数，字符入栈
37     jmp getstrs
38
39 nochar:
40     cmp ah,0eh;退格键扫描码
41     je backspace
42     cmp ah,1ch;回车键的扫描码
43     je enters
44     jmp getstrs
45 backspace:
46     mov ah,1
47     call charstack
48     jmp getstrs
49
50 enters:
51     mov al,0
52     mov ah,0
53     call charstack;结束输入，把0字符入栈

```

```

54
55     mov ah,2;显示字符串
56     call charstack
57
58
59
60     pop ax
61     ret;getstr结束
62
63
64 charstack:
65     jmp short charstart
66     chartable db '0123456789ABCDE'
67     table dw charpush,charpop,charshow
68 charstart:
69     push bx
70     push dx
71     push di
72     push es
73     push cx
74
75     cmp ah,2
76
77     ja sret ;大于2, 就不是调用。直接返回
78
79     mov bl,ah
80     mov bh,0
81     add bx,bx
82     jmp word ptr table[bx]
83
84 charpush:
85     mov bx,0B800H
86     mov es,bx
87     ;输出当前敲的是哪个键
88     mov byte ptr es:[2],al
89     mov byte ptr es:[3],00000100b
90
91     mov bl,top
92     mov bh,0
93     mov [si+bx],al
94     add top,1
95
96
97     mov bl,top;测试 top有没有+1
98     mov bh,0
99     ;显示当前有几个字符串, top的值
100    mov bl,chartable[bx]
101    mov byte ptr es:[6],bl
102    mov byte ptr es:[7],00000100b
103
104    jmp sret
105
106 charpop:

```

```

107         cmp top,0
108         je sret
109         dec top
110         mov bl,top
111         mov bh,0
112         mov al,[si+bx]
113         ;测试有没有调用这个
114         ;mov bx,0B800H
115         ;mov es,bx
116         ;mov byte ptr es:[160*2],'o'
117         ;mov byte ptr es:[160*2+1],00000100b
118         jmp sret
119
120 charshow:
121         cmp top,0
122         je sret;没有字符串了,直接回去
123         mov bx,0B800H
124         mov es,bx
125         mov bx,0
126         mov di,12*160+40
127
128         mov cl,top
129         mov ch,0
130
131         s1:mov al,[bx]
132         mov es:[di],al
133         add di,2
134         inc bx
135
136
137         loop s1
138
139 sret:
140         pop cx
141         pop es
142         pop di
143         pop dx
144         pop bx
145
146         ret
147
148
149
150 code ends
151 end start

```

20.让计算机唱歌

```

1  assume cs:code,ss:stack,ds:data
2
3  data segment
4      ;音乐频谱

```

```

5     mus_freq dw 262,262,262,196,330,330,330,262
6     dw 262,330,392,392,349,330,294
7     dw 294,330,349,349,330,294,330,262
8     dw 262,330,294,196,247,294,262,-1
9     ;每个应付延续时间
10    mus_time dw 3 dup(12,12,25,25),12,12,50
11    dw 3 dup(12,12,25,25),12,12,50
12 data ends
13 stack segment
14     db 100 dup(0)
15 stack ends
16 code segment
17 start:
18     mov ax,stack
19     mov ss,ax
20     mov sp,100
21     mov ax,data
22     mov ds,ax
23     lea si,mus_freq
24     lea di,mus_time
25 play:
26     mov dx,[si]
27     cmp dx,-1
28     je end_play
29     call sound
30     add si,2
31     add di,2
32     jmp play
33 end_play:
34     mov ax,4c00h
35     int 21h
36
37 sound:
38     push ax
39     push dx
40     push cx
41
42     ;把频率传送给, 8253芯片, 下面的设置, 都是固定的
43     mov al,0b6h
44     out 43h,al
45     mov dx,12h
46     mov ax,34dch
47     div word ptr [si];si里面就是音乐频率, 这个是自己给的
48     out 42h,al
49     mov al,ah
50     out 42h,al
51
52
53     ;控制8255芯片, 把扬声器打开
54     in al,61h
55     mov ah,al;保存当前扬声器的状态, 以便稍后恢复
56     or al,3
57     out 61h,al

```

```
58
59     ;设置扬声器开启时长
60
61     mov dx,[di]
62 wait1:
63     mov cx,20000
64 delay:
65     nop
66     loop delay
67     dec dx;这里用的是双重循环。dx判断外重循环多少次
68     jnz wait1
69     ;关闭扬声器, 恢复扬声器的值
70     mov al,ah
71     out 61h,al
72
73     pop cx
74     pop dx
75     pop ax
76     ret
77
78 code ends
79 end start
```