

0608-研究试验4-宣讲会研究报告-尹忠恩

不使用 main 函数

编写程序 f.c

```
1 f()
2 {
3     *(char far *) (0xb8000000 + 160 * 10 + 2 * 40)
    = 'a';
4     *(char far *) (0xb8000000 + 160 * 10 + 2 * 40 +
    1) = 2;
5 }
```

1. 将程序保存在 minc 目录下 编译链接

1. 在连接过程中会出现问题

o



2. 提示出的缺少 `_main`

3. 可能与 `C0S` 有关

2. 用学习汇编是的 `link.exe` 对 `tc.exe` 生成的 `f.obj` 进行连接生成 `f.exe`

1. 由图中可见总共由 **303** 个字节

○  F.OBJ

文件类型: 3D Object (.OBJ)

打开方式: 3D 查看器 更改(C)...

位置: F:\GitHub\ASM-\minic

大小: 303 字节 (303 字节)

占用空间: 0 字节

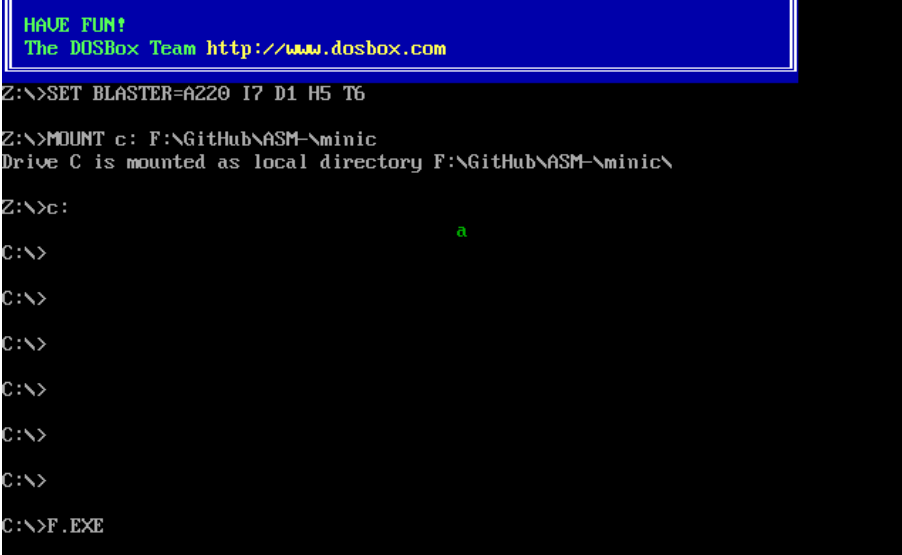
创建时间: 2020年6月8日, 17:01:32

修改时间: 2020年6月8日, 17:01:32

访问时间: 2020年6月8日, 17:01:32

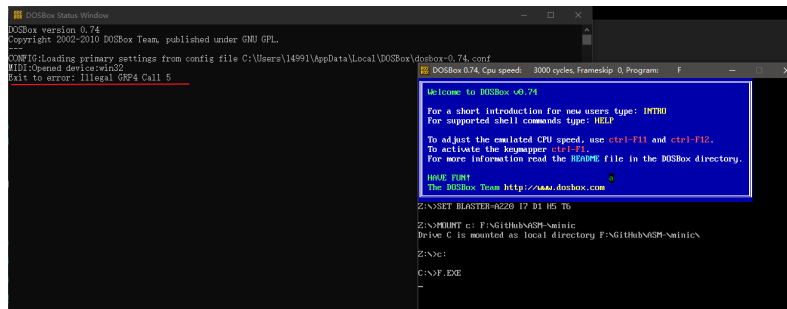
属性: ☐ 只读(R) ☐ 隐藏(H) 高级(D)...

2. 函数 `f` 实现了其功能 在屏幕中打印出 **a**

○ 

```
HAVE FUN!  
The DOSBox Team http://www.dosbox.com  
  
Z:\>SET BLASTER=A220 I7 D1 H5 T6  
  
Z:\>MOUNT c: F:\GitHub\ASM-\minic  
Drive C is mounted as local directory F:\GitHub\ASM-\minic\  
  
Z:\>c:  
  
C:\>  
C:\>  
C:\>  
C:\>  
C:\>  
C:\>  
C:\>  
C:\>F.EXE
```

- 但是函数不能正常结束 Dosbox 会卡死后然后退出



3. debug 后可见 f 函数的偏移地址为0

```

Z:\>MOUNT c: F:\GitHub\ASM\minic
Drive C is mounted as local directory F:\GitHub\ASM\minic\

Z:\>c:

C:\>DEBUG.EXE F.EXE
-u
076A:0000 55          PUSH    BP
076A:0001 8BEC          MOV     BP,SP
076A:0003 BB00B8      MOV     BX,BB00
076A:0006 BEC3          MOV     ES,BX
076A:0008 BB9006      MOV     BX,0690
076A:000B 26          ES:
076A:000C C60761      MOV     BYTE PTR [BX],61
076A:000F BB00B8      MOV     BX,BB00
076A:0012 BEC3          MOV     ES,BX
076A:0014 BB9106      MOV     BX,0691
076A:0017 26          ES:
076A:0018 C60702      MOV     BYTE PTR [BX],02
076A:001B 5D          POP     BP
076A:001C C3          RET
076A:001D 0000      ADD     [BX+SI],AL
076A:001F 0000      ADD     [BX+SI],AL
- ;

```

3. 写一个 m.c 在编译连接

```

1 | main()
2 | {
3 |     *(char far *) (0xb8000000 + 160 * 10 + 2 * 40)
   | = 'a';
4 |     *(char far *) (0xb8000000 + 160 * 10 + 2 * 40 +
   | 1) = 2;
5 | }

```

1. 由图可见代码总长为 **4280** 个字节

	M.EXE
文件类型:	应用程序 (.EXE)
描述:	M.EXE
位置:	F:\GitHub\ASM-\minic
大小:	4.17 KB (4,280 字节)
占用空间:	8.00 KB (8,192 字节)
创建时间:	2020年6月8日, 17:25:31
修改时间:	2020年6月8日, 17:25:31
访问时间:	2020年6月8日, 17:25:31
属性:	<input type="checkbox"/> 只读(R) <input type="checkbox"/> 隐藏(H) 高级(D)...

2. 由图可见 `m.exe` 可以正常返回 因为在执行玩 `m.exe` 后还可以在命令行输入新的命令

```

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6
Z:\>MOUNT c: F:\GitHub\ASM-\minic
Drive C is mounted as local directory F:\GitHub\ASM-\minic\
Z:\>c:
C:\>tc M.C
C:\>M.EXE
C:\>M.EXE
C:\>

```

3. `m.exe` 的汇编代码中 `main` 的偏移地址为 `01fa` 而 `f.exe` 中 `f` 的偏移地址为 `0000`

```

076A:01FA 55          PUSH    BP
076A:01FB 8BEC          MOV     BP,SP
076A:01FD BB00B8      MOV     BX,B800
076A:0200 8EC3          MOV     ES,BX
076A:0202 BB9006      MOV     BX,0690
076A:0205 26             ES:
076A:0206 C60761      MOV     BYTE PTR [BX],61
076A:0209 BB00B8      MOV     BX,B800
076A:020C 8EC3          MOV     ES,BX
076A:020E BB9106      MOV     BX,0691
076A:0211 26             ES:
076A:0212 C60702      MOV     BYTE PTR [BX],02
076A:0215 5D             POP     BP
076A:0216 C3             RET
076A:0217 C3             RET
076A:0218 55          PUSH    BP
076A:0219 8BEC          MOV     BP,SP

```

```

Z:\>MOUNT c: F:\GitHub\ASM-\minic
Drive C is mounted as local directory F:\GitHub\ASM-\minic\

Z:\>c:

C:\>DEBUG.EXE F.EXE
~u
076A:0000 55          PUSH    BP
076A:0001 8BEC          MOV     BP,SP
076A:0003 BB00B8      MOV     BX,B800
076A:0006 8EC3          MOV     ES,BX
076A:0008 BB9006      MOV     BX,0690
076A:000B 26             ES:
076A:000C C60761      MOV     BYTE PTR [BX],61
076A:000F BB00B8      MOV     BX,B800
076A:0012 8EC3          MOV     ES,BX
076A:0014 BB9106      MOV     BX,0691
076A:0017 26             ES:
076A:0018 C60702      MOV     BYTE PTR [BX],02
076A:001B 5D             POP     BP
076A:001C C3             RET
076A:001D 0000          ADD     [BX+SI],AL
076A:001F 0000          ADD     [BX+SI],AL
~ ;

```

4. debug 对 m.exe 进行跟踪

1. 跳转到main函数开始地址

```

AX=0000 BX=0691 CX=0011 DX=9C68 SP=FFEA BP=FFF2 SI=003A DI=0229
DS=07C1 ES=B800 SS=07C1 CS=076A IP=011D  NU UP EI PL ZR NA PE NC
076A:011D 50          PUSH    AX
~q
C:\>DEBUG.EXE M.EXE
~u 01fa
076A:01FA 55          PUSH    BP
076A:01FB 8BEC          MOV     BP,SP
076A:01FD BB00B8      MOV     BX,B800
076A:0200 8EC3          MOV     ES,BX
076A:0202 BB9006      MOV     BX,0690
076A:0205 26             ES:
076A:0206 C60761      MOV     BYTE PTR [BX],61
076A:0209 BB00B8      MOV     BX,B800
076A:020C 8EC3          MOV     ES,BX
076A:020E BB9106      MOV     BX,0691
076A:0211 26             ES:
076A:0212 C60702      MOV     BYTE PTR [BX],02
076A:0215 5D             POP     BP
076A:0216 C3             RET
076A:0217 C3             RET
076A:0218 55          PUSH    BP
076A:0219 8BEC          MOV     BP,SP
~ ;

```

2. 执行到 ret 后 跳转到了 076a: 011d

```

-g 0216
AX=0000 BX=0691 CX=0011 DX=AAE8 SP=FFEB BP=FFF2 SI=003A DI=0229
DS=07C1 ES=B800 SS=07C1 CS=076A IP=0216 NU UP EI PL ZR NA PE NC
076A:0216 C3 RET
-t
AX=0000 BX=0691 CX=0011 DX=AAE8 SP=FFEA BP=FFF2 SI=003A DI=0229
DS=07C1 ES=B800 SS=07C1 CS=076A IP=011D NU UP EI PL ZR NA PE NC
076A:011D 50 PUSH AX
-;

```

- 由图可以看到 在 07fa:011a 中调用 main

```

-u 0110
076A:0110 8800 MOV [BX+SI],AL
076A:0112 FF368600 PUSH [0086]
076A:0116 FF368400 PUSH [0084]
076A:011A E8DD00 CALL 01FA
076A:011D 50 PUSH AX
076A:011E E8F700 CALL 0218

```

- 在执行完划红线的ret 然后一直按p 会找到程序的返回指令为 int 21

```

AX=4C00 BX=0691 CX=0000 DX=1060 SP=FFE0 BP=FFE0 SI=002F DI=0229
DS=07C1 ES=B800 SS=07C1 CS=076A IP=0156 NU UP EI PL ZR NA PE NC
076A:0156 CD21 INT 21
-p
Program terminated normally

```

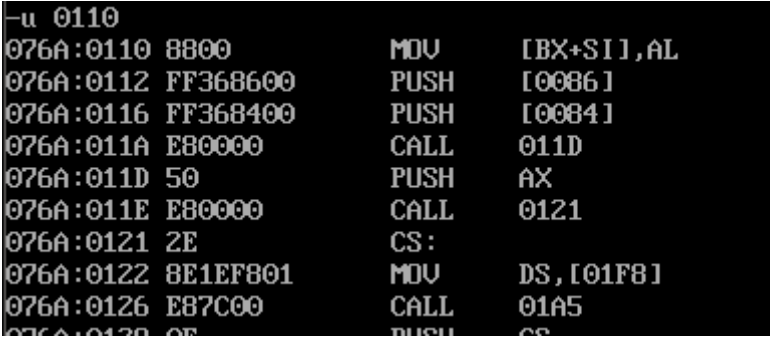
5. 思考问题

- main函数的调用指令和程序返回指令应该是c0s中对应的代码中调用的
- 没有main 函数，提示错误信息中由“c0s”的相关信息，因该是 tc.exe 把 在搭建环境时所需要的 文件 和 用户的 obj 生成 .exe 而不单单是一个 c0s.obj
- 调用 main 函数的指令 因该来自于 c0s.obj
- 用 link.exe 连接 c0s.obj
- debug 后的 c0s.exe

```

076A:0000 BAB07 MOV DX,078A
076A:0003 2E CS:
076A:0004 8916F801 MOV [01F8],DX
076A:0008 B430 MOV AH,30
076A:000A CD21 INT 21
1. 076A:000C 8B2E0200 MOV BP,[0002]
076A:0010 8B1E2C00 MOV BX,[002C]
076A:0014 8EDA MOV DS,DX
076A:0016 A39200 MOV [0092],AX
076A:0019 8C069000 MOV [0090],ES
076A:001D 891E8C00 MOV [008C],BX

```

2.  The screenshot shows assembly code for c0s.exe. It starts with a label -u at address 0110. The code includes instructions like MOV [BX+SI],AL, PUSH [0086], PUSH [0084], CALL 011D, PUSH AX, CALL 0121, CS:, MOV DS,[01F8], and CALL 01A5.
3. 可以看见基本结构和由用户生成的包含main 函数的基本结构相似

6. 用 `link.exe` 对 `c0s.obj` 连接生成 `c0s.exe`

c0s.exe	m.exe
<pre> C:\>DEBUG.EXE C0S.EXE -u 076A:0000 BABA07 MOV DX,078A 076A:0003 2E CS: 076A:0004 8916F801 MOV [01F8],DX 076A:0008 B430 MOV AH,30 076A:000A CD21 INT 21 076A:000C 8B2E0200 MOV BP,[0002] 076A:0010 8B1E2C00 MOV BX,[002C] 076A:0014 8EDA MOV DS,DX 076A:0016 A39200 MOV [0092],AX 076A:0019 8C069000 MOV [0090],ES 076A:001D 891E8C00 MOV [008C],BX 076A:001D 891E8C00 MOV [008C],BX -u 076A:0021 892EAC00 MOV [00AC],BP 076A:0025 C7069600FFFF MOV WORD PTR [0096],FFFF 076A:002B E83401 CALL 0162 076A:002E C43E8A00 LES DI,[008A] 076A:0032 8BC7 MOV AX,DI 076A:0034 8BD8 MOV BX,AX 076A:0036 B9FF7F MOV CX,7FFF 076A:0039 26 ES: 076A:003A 813D3837 CMP WORD PTR [DI],3738 076A:003E 7519 JNZ 0059 076A:0040 26 ES: 076A:0041 8B5502 MOV DX,[DI+02] 076A:0041 8B5502 MOV DX,[DI+02] -u 076A:0044 80FA3D CMP DL,3D 076A:0047 7510 JNZ 0059 076A:0049 80E6DF AND DH,DF 076A:004C FF069600 INC WORD PTR [0096] 076A:0050 80FE59 CMP DH,59 076A:0053 7504 JNZ 0059 076A:0055 FF069600 INC WORD PTR [0096] 076A:0059 F2 REPNZ 076A:005A AE SCASB 076A:005B E361 JCXZ 00BE 076A:005D 43 INC BX 076A:005E 26 ES: 076A:005F 3B05 CMP [DI],AL 076A:0061 75D6 JNZ 0039 076A:0063 80CD80 OR CH,80 </pre>	<pre> C:\>DEBUG.EXE M.EXE -u 076A:0000 BAC107 MOV DX,07C1 076A:0003 2E CS: 076A:0004 8916F801 MOV [01F8],DX 076A:0008 B430 MOV AH,30 076A:000A CD21 INT 21 076A:000C 8B2E0200 MOV BP,[0002] 076A:0010 8B1E2C00 MOV BX,[002C] 076A:0014 8EDA MOV DS,DX 076A:0016 A39200 MOV [0092],AX 076A:0019 8C069000 MOV [0090],ES 076A:001D 891E8C00 MOV [008C],BX 076A:001D 891E8C00 MOV [008C],BX -u 076A:0021 892EAC00 MOV [00AC],BP 076A:0025 C7069600FFFF MOV WORD PTR [0096],FFFF 076A:002B E83401 CALL 0162 076A:002E C43E8A00 LES DI,[008A] 076A:0032 8BC7 MOV AX,DI 076A:0034 8BD8 MOV BX,AX 076A:0036 B9FF7F MOV CX,7FFF 076A:0039 26 ES: 076A:003A 813D3837 CMP WORD PTR [DI],3738 076A:003E 7519 JNZ 0059 076A:0040 26 ES: 076A:0041 8B5502 MOV DX,[DI+02] 076A:0041 8B5502 MOV DX,[DI+02] -u 076A:0044 80FA3D CMP DL,3D 076A:0047 7510 JNZ 0059 076A:0049 80E6DF AND DH,DF 076A:004C FF069600 INC WORD PTR [0096] 076A:0050 80FE59 CMP DH,59 076A:0053 7504 JNZ 0059 076A:0055 FF069600 INC WORD PTR [0096] 076A:0059 F2 REPNZ 076A:005A AE SCASB 076A:005B E361 JCXZ 00BE 076A:005D 43 INC BX 076A:005E 26 ES: 076A:005F 3B05 CMP [DI],AL 076A:0061 75D6 JNZ 0039 076A:0063 80CD80 OR CH,80 </pre>

由上图可知代码开头部分两者的开始部分都相同

7. 对比 m.exe 和 c0s.exe 中调用main函数的call指令的偏移地址的后10条指令

c0s.exe				m.exe			
076A:011A	E80000	CALL	011D	076A:011A	E8DD00	CALL	01FA
076A:011D	50	PUSH	AX	076A:011D	50	PUSH	AX
076A:011E	E80000	CALL	0121	076A:011E	E8F700	CALL	0218
076A:0121	2E	CS:		076A:0121	2E	CS:	
076A:0122	8E1EF801	MOV	DS,[01F8]	076A:0122	8E1EF801	MOV	DS,[01F8]
076A:0126	E87C00	CALL	01A5	076A:0126	E87C00	CALL	01A5
076A:0129	0E	PUSH	CS	076A:0129	0E	PUSH	CS
076A:012A	FF169601	CALL	[0196]	076A:012A	FF16A201	CALL	[01A2]
076A:012E	33C0	XOR	AX,AX	076A:012E	33C0	XOR	AX,AX
076A:0130	8BF0	MOV	SI,AX	076A:0130	8BF0	MOV	SI,AX
076A:0132	B92F00	MOV	CX,002F	076A:0132	B92F00	MOV	CX,002F
076A:0135	90	NOP		076A:0135	90	NOP	
076A:0136	FC	CLD		076A:0136	FC	CLD	
076A:0137	0204	ADD	AL,[SI]	076A:0137	0204	ADD	AL,[SI]
076A:0139	80D400	ADC	AH,00	076A:0139	80D400	ADC	AH,00

由上图可知前10行有3行调用的不同 第一个为调用main函数然后返回到 c0s.obj 中的 **076a:011d**，其余的call应该也是完成相关任务后返回 c0s.obj

8. 改写 c0s.obj

1. 编写c0s.obj

```
1  assume cs:code
2  data segment
3
4  DB 128 dup(0)
5
6  data ends
7
8  code segment
9  start:
10     mov ax,data
11     mov ds,ax
12     mov ss,ax
13     mov sp,128
14
15     call s
```



```

16
17     mov ax,4c00h
18     int 21h
19
20     s:
21 code ends
22 end start

```

2. 用 `masm` 生成 `c0s.obj`

3. 在 `tc` 中打开 `f.c` 编译 连接 后可以正常生成 `f.exe`

```

Z:\>c:
C:\>MASM.EXE COS.ASM: 1
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987. All rights reserved.

51766 + 464778 Bytes symbol space free

0 Warning Errors
0 Severe Errors

C:\>tc F.C 2
C:\>F.EXE 3

```

9 debug f.exe

```

0772:0000 BB6A07      MOV     AX,076A
0772:0003 8ED8           MOV     DS,AX
0772:0005 8ED0           MOV     SS,AX
0772:0007 BC8000      MOV     SP,0080
0772:000A E80500      CALL    0012
0772:000D B8004C      MOV     AX,4C00
0772:0010 CD21           INT     21
0772:0012 55           PUSH    BP
0772:0013 8BEC           MOV     BP,SP
0772:0015 BB00B8      MOV     BX,B800
0772:0018 8EC3           MOV     ES,BX
0772:001A BB9006      MOV     BX,0690
0772:001D 26           ES:
0772:001E C60761      MOV     BYTE PTR [BX],61

```

```

0772:001E C60761      MOV     BYTE PTR [BX],61
-u
0772:0021 BB00B8      MOV     BX,B800
0772:0024 8EC3        MOV     ES,BX
0772:0026 BB9106      MOV     BX,0691
0772:0029 26           ES:
0772:002A C60702      MOV     BYTE PTR [BX],02
0772:002D 5D           POP     BP
0772:002E C3           RET
0772:002F 00FB        ADD     BL,BH
0772:0031 52           PUSH    DX
0772:0032 0802        OR      [BP+SI],AL
0772:0034 1900        SBB     [BX+SI],AX
0772:0036 0000        ADD     [BX+SI],AL
0772:0038 06           PUSH    ES
0772:0039 0019        ADD     [BX+DI],BL
0772:003B 0000        ADD     [BX+SI],AL
0772:003D 0001        ADD     [BX+DI],AL
0772:003F 0001        ADD     [BX+DI],AL

```

可以看到程序正确的调用了 `f()`

疑问：

为什么汇编代码中仅仅调用了 `s` 然后 `s` 什么都没用，连接完后就把 `f` 函数自动填充过去到 `s` 处

10 编写新的 `f.c`

```

1 | #define Buffer ((char *)*(int far *)0x02000000)
2 | f()
3 | {
4 |     Buffer = 0;
5 |     Buffer[10] = 0;
6 |     while (Buffer[10] != 8)
7 |     {
8 |         Buffer[Buffer[10]] = 'a' + Buffer[10];
9 |         Buffer[10]++;
10 |    }
11 | }

```

debug 后的 `f.exe`

```
Z:\>MOUNT c: F:\GitHub\ASM-\minic
Drive C is mounted as local directory F:\GitHub\ASM-\minic\
```

```
Z:\>c:
```

```
C:\>tc F.C
```

```
C:\>DEBUG.EXE F.EXE
```

```
-u
0772:0000 B86A07      MOV     AX,076A
0772:0003 8ED8          MOV     DS,AX
0772:0005 8ED0          MOV     SS,AX
0772:0007 BC8000      MOV     SP,0080
0772:000A E80500      CALL    0012
0772:000D B8004C      MOV     AX,4C00
0772:0010 CD21          INT     21
0772:0012 55            PUSH    BP
0772:0013 8BEC          MOV     BP,SP
0772:0015 BB0002      MOV     BX,0200
0772:0018 8EC3          MOV     ES,BX
0772:001A 33DB          XOR     BX,BX
0772:001C 26            ES:
0772:001D C7070000      MOV     WORD PTR [BX],0000
- ; _
```

```
0772:001D C7070000      MOV     WORD PTR [BX],0000
-u
0772:0021 BB0002      MOV     BX,0200
0772:0024 8EC3          MOV     ES,BX
0772:0026 33DB          XOR     BX,BX
0772:0028 26            ES:
0772:0029 8B1F          MOV     BX,[BX]
0772:002B C6470A00      MOV     BYTE PTR [BX+0A],00
0772:002F EB3C          JMP     006D
0772:0031 BB0002      MOV     BX,0200
0772:0034 8EC3          MOV     ES,BX
0772:0036 33DB          XOR     BX,BX
0772:0038 26            ES:
0772:0039 8B1F          MOV     BX,[BX]
0772:003B 8A470A      MOV     AL,[BX+0A]
0772:003E 0461          ADD     AL,61
0772:0040 BB0002      MOV     BX,0200
- ;
```

```
0772:0040 BB0002      MOV     BX,0200
-u
0772:0043 8EC3          MOV     ES,BX
0772:0045 33DB          XOR     BX,BX
0772:0047 26            ES:
0772:0048 8B1F          MOV     BX,[BX]
0772:004A 50            PUSH    AX
0772:004B 53            PUSH    BX
0772:004C BB0002      MOV     BX,0200
0772:004F 8EC3          MOV     ES,BX
0772:0051 33DB          XOR     BX,BX
0772:0053 26            ES:
0772:0054 8B1F          MOV     BX,[BX]
0772:0056 8A470A      MOV     AL,[BX+0A]
0772:0059 98            CBW
0772:005A 5B            POP     BX
0772:005B 03D8          ADD     BX,AX
0772:005D 58            POP     AX
0772:005E 8807          MOV     [BX],AL
0772:0060 BB0002      MOV     BX,0200
- ; _
```

```

0772:0063 8EC3      MOV     ES,BX
0772:0065 33DB      XOR     BX,BX
0772:0067 26        ES:
0772:0068 8B1F      MOV     BX,[BX]
0772:006A FE470A    INC     BYTE PTR [BX+0A]
0772:006D BB0002    MOV     BX,0200
0772:0070 8EC3      MOV     ES,BX
0772:0072 33DB      XOR     BX,BX
0772:0074 26        ES:
0772:0075 8B1F      MOV     BX,[BX]
0772:0077 807F0A0B  CMP     BYTE PTR [BX+0A],0B
0772:007B 75B4      JNZ     0031
0772:007D 5D        POP     BP
0772:007E C3        RET
0772:007F 00FB      ADD     BL,BH
0772:0081 52        PUSH    DX
0772:0082 0B02      OR      [BP+SI],AL
- ;

```

可以看到编译连结完debug后的代码逻辑和汇编代码逻辑基本一致

感悟

C0S.OBJ的功能就是进行相关初始化，申请资源、设置DS、SS等寄存器，并且在MAIN函数结束后进行相关资源的释放、环境恢复等工作，随后调用DOS的21H程序返回。