

0622-研究试验

修改c0s.obj

按书上的汇编代码编译完生成的c0s.obj

- 测试函数(1)

```
1
2 f()
3 {
4     *(char far *) (0xb8000000 + 160 * 10 + 2 * 40) =
5     'a';
6     *(char far *) (0xb8000000 + 160 * 10 + 2 * 40 + 1)
7     = 2;
8 }
9
10 main(){
11     int a=0;
12     int b=0;
13     f();
14 }
```

```
C:\>DEBUG.EXE MYCOS.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 BB00B8      MOV     BX,B800
0772:0015 8EC3             MOV     ES,BX
0772:0017 BB9006      MOV     BX,0690
0772:001A 26             ES:
0772:001B C60761      MOV     BYTE PTR [BX],61
0772:001E BB00B8      MOV     BX,B800
```

- 测试函数(2)

```
1
2 f();
3
4 main(){
5     int a=0;
6     int b=0;
7     f();
8 }
```

```

9
10 f()
11 {
12     *(char far *) (0xb8000000 + 160 * 10 + 2 * 40) =
    'a';
13     *(char far *) (0xb8000000 + 160 * 10 + 2 * 40 + 1)
    = 2;
14 }

```

```

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 83EC04      SUB     SP,+04
0772:0018 C746FC0000  MOV     WORD PTR [BP-04],0000
0772:001D C746FE0000  MOV     WORD PTR [BP-02],0000

```

可以看到测试函数1是先调用 `f()` 而测试函数2先调用 `main()` 还是没思路

自己编写的printf

- ```

1 void myPrintf(char *, ...);
2
3 main()
4 {
5 myPrintf(" char: %c\n int: %d", 'x',
6 5);
7 }
8 void myPrintf(char *str, ...)
9 {
10 int stackIndex = 0;
11 int stringIndex = 0;
12 int screenIndex = 0;
13 int screenBenchmark = 160 * 10;
14
15 while (str[stringIndex] != 0)
16 {
17 if (str[stringIndex] == '%')
18 {
19 if (str[stringIndex + 1] ==
20 'c')
21 {
22 *(char far *) (0xb8000000
23 + screenBenchmark + screenIndex) = *(char
24 *) (_BP + 6 + stackIndex);
25 *(char far *) (0xb8000000
26 + screenBenchmark + screenIndex + 1) = 2;

```

```

23 screenIndex += 2;
24 stringIndex += 2;
25 stackIndex += 2;
26 }
27 else if (str[stringIndex + 1]
== 'd')
28 {
29 *(char far *)(0xb8000000
+ screenBenchmark + screenIndex) = *(char
*)(_BP + 6 + stackIndex) + 0x30;
30 *(char far *)(0xb8000000
+ screenBenchmark + screenIndex + 1) = 2;
31
32 stackIndex += 2;
33 screenIndex += 2;
34 stringIndex += 2;
35 }
36 else if (str[stringIndex + 1]
== 'n')
37 {
38 screenBenchmark += 160;
39 screenIndex = 0;
40 stringIndex += 2;
41 }
42 }
43 else
44 {
45 *(char far *)(0xb8000000 +
screenBenchmark + screenIndex) =
str[stringIndex];
46 *(char far *)(0xb8000000 +
screenBenchmark + screenIndex + 1) = 2;
47
48 screenIndex += 2;
49 stringIndex += 1;
50 }
51 }
52 }

```

•

```

The DOSBOX Team http://www.dosbox.org/
Z:\>SET BLASTER=A220 I7 D1 H5 T6
char: x
int: 5
nt c I:\Github\ASM\minic
Drive C is mounted as local direct
Z:\>c:

```

各种数据类型是通过什么返回的

float( 注释中为对应的汇编代码 )

```

1 float f(float, float);
2
3 /*
4 _main proc near
5 push bp
6 mov bp,sp
7 sub sp,12
8 */
9 main()
10 {
11 // ax=cccd dx= 400c
12 float c = 2.2, a = 1.1, b = 1.1; /*
13 mov dx,16396
14 mov ax,-13107
15 mov word ptr
16 [bp-10],dx
17 mov word ptr
18 [bp-12],ax

```

下图为2.2对应的十六进制

```

076A:0200 BA0C40 MOV DX,400C
076A:0203 B8CDCC MOV AX,CCCD
076A:0206 8956F6 MOV [BP-0A],DX
076A:0209 8946F4 MOV [BP-0C],AX

```

```

1 /* ; ?debug L 11
2 mov dx,16268
3 mov ax,-13107
4 mov word ptr
5 [bp-6],dx
6 mov word ptr
7 [bp-8],ax
8 ; ?debug L 11
9 mov dx,16268
10 mov ax,-13107
11 mov word ptr
12 [bp-2],dx
13 mov word ptr
14 [bp-4],ax
15
16 */
17
18 c = f(a, b); /*
19
20 mov dx,word
21 ptr [bp-2]
22
23 mov ax,word
24 ptr [bp-4]
25
26 push dx
27 push ax
28 mov dx,word
29 ptr [bp-6]

```

```

19 mov ax,word
 ptr [bp-8]
20 push dx
21 push ax
22 call near
 ptr _f

```

跳转到 \_f 函数（在下一段）执行，然后返回到此处在执行

[点击跳转](#)

```

1 /* add sp,8
2 FSTP dword
 ptr [bp-12]
3 FWAIT
4 */
5 }

```

最终结果在栈中

```

-d ss:ffd4
0BD3:FFD0 CD CC 0C 40-CD CC 8C 3F CD CC 8C 3F ...@...?...?
0BD3:FFE0 EC FF 1D 01 01 00 EA FF-2E 06 EE FF 00 00 43 3A C:
0BD3:FFF0 5C 46 52 45 54 55 52 4E-2E 45 58 45 00 00 FB 00 \PRETURN.EXE...

```

```

1
2 /*
3 _f proc near
4 push bp
5 mov bp,sp
6 sub sp,4
7 */
8 float f(float a, float b)
9 {
10 float ab = a + b; /*
11 FLD dword ptr [bp+4]
12 FLD dword ptr [bp+8]
13 FADD
14 FSTP dword ptr [bp-4]
15 FWAIT
16 */
17
18 return ab; /*
19 FLD dword ptr [bp-4]
20 jmp short @2
21 @2: */

```

返回前的各个寄存器的值

```

AX=CCCD BX=0634 CX=000F DX=3F8C SP=FFC4 BP=FFC8 SI=003A DI=0627
DS=0BD3 ES=0BD3 SS=0BD3 CS=076A IP=0261 NU UP EI NG NZ NA PO NC
076A:0261 8BE5 MOV SP,BP

```

对应汇编代码中的 `[bp-4]` 查看 `ss:ffc4` 中的值

```
-d ss:ffc4 2.2对应的值
0BD3:FFC0 CD CC 0C 40-E0 FF 37 02 CD CC 8C 3F ...@..7...?
0BD3:FFD0 CD CC 8C 3F CD CC 0C 40-CD CC 8C 3F CD CC 8C 3F ...?...@...?...?
0BD3:FFE0 EC FF 1D 01 01 00 EA FF-2E 06 EE FF 00 00 43 3A C:
0BD3:FFF0 5C 46 52 45 54 55 52 4E-2E 45 58 45 00 00 FB 00 \FRETURN.EXE....
```

```
1 /*
2 ; ?debug L 50
3 mov sp,bp
4 pop bp
5 ret
6 */
7 }
```

返回

结构体（注释中为对应的汇编代码）

```
1 struct returnStruct
2 {
3 int i_num;
4 float f_num;
5 double d_num;
6 };
7
8 struct returnStruct f(struct returnStruct);
9
10
11 /*
12 proc near
13 push bp
14 mov bp,sp
15 sub sp,28
16 */
17
18 main()
19
```

此时各个寄存器的值

```
AX=0000 BX=0654 CX=000D DX=B605 SP=FFC2 BP=FFDE SI=003A DI=0647
DS=0BDD ES=0BDD SS=0BDD CS=076A IP=0200 NU UP EI NG NZ NA PO NC
```

```
1 {
2 struct returnStruct s;
3 struct returnStruct r;
4
5 s.i_num = 1; // mov word ptr [bp-28],1
6 s.f_num = 1.1; /*
7 mov dx,16268
8 mov ax,-13107
```

```

9 mov word ptr [bp-24],dx
10 mov word ptr [bp-26],ax
11 */
12 s.d_num = 1.11; /*
13 mov word ptr [bp-16],16369
14 mov word ptr [bp-
15 18],-15729
16 mov word ptr [bp-20],23592
17 mov word ptr [bp-22],-2621
18 */

```

数据入栈后

```
0BDD:FFC0 01 00 CD CC 8C 3F-C3 F5 28 5C 8F C2 F1 3F
```

```

1
2 r=f(s); /*
3 lea bx,word ptr [bp-14]
4 push ss
5 push bx
6 lea bx,word ptr [bp-28]
7 mov dx,ss
8 mov ax,bx
9 mov cx,14
10 call far ptr SPUSH@

```

SPUSH@执行前后对应寄存器的变化

```

AX=FFC2 BX=FFC2 CX=000E DX=0BDD SP=FFBE BP=FFDE SI=003A DI=0647
DS=0BDD ES=0BDD SS=0BDD CS=076A IP=0234 NU UP EI NG NZ NA PO NC
076A:0234 9AE61C6A07 CALL 076A:1CE6
-p
AX=0BDD BX=0239 CX=0000 DX=076A SP=FFB0 BP=FFDE SI=003A DI=0647
DS=0BDD ES=0BDD SS=0BDD CS=076A IP=0239 NU UP EI PL ZR NA PE NC

```

可以看到栈顶指针变了然后看对应的栈空间

```

0BDD:FFB0 01 00 CD CC 8C 3F C3 F5-28 5C 8F C2 F1 3F D0 FF?..(\...?..
0BDD:FFC0 DD 0B 01 00 CD CC 8C 3F-C3 F5 28 5C 8F C2 F1 3F?..(\...?
0BDD:FFD0 00 00 00 00 00 00 00 FF-FD 01 60 07 46 73 F0 FFi Es

```

可以看到 SPUSH@ 的功能是 把 保存的数据再在栈空间中复制一份

```

1 // call near ptr _f

```

\_f 执行前后对应寄存器的变化

```

AX=0BDD BX=0239 CX=0000 DX=076A SP=FFB0 BP=FFDE SI=003A DI=0647
DS=0BDD ES=0BDD SS=0BDD CS=076A IP=0239 NU UP EI PL ZR NA PE NC
076A:0239 E83700 CALL 0273
-p
AX=05B4 BX=FFB0 CX=0000 DX=076A SP=FFB0 BP=FFDE SI=003A DI=0647
DS=0BDD ES=0BDD SS=0BDD CS=076A IP=023C NU UP EI PL ZR NA PE NC

```

可以看到ax 值改变，然后查看对应内存空间的值

```
-d ds:05b4
0BDD:05B0 02 00 66 66-06 40 E2 7A 14 AE 47 E1
0BDD:05C0 00 40 00 00 00 00 00 00-00 00 00 00 00 00 00
```

可以发现和在call near ptr\_f中最后 `lea bx,word ptr [bp+4]` 执行完后的栈空间的数据一样

```
AX=0BDD BX=05B4 CX=0000 DX=076A SP=FFA8 BP=FFAC SI=003A DI=0647
DS=0BDD ES=0BDD SS=0BDD CS=076A IP=029C NU UP EI PL NZ NA PO NC
076A:029C 8D5E04 LEA BX,[BP+04] SS:FFB0=0002
-t
AX=0BDD BX=FFB0 CX=0000 DX=076A SP=FFA8 BP=FFAC SI=003A DI=0647
DS=0BDD ES=0BDD SS=0BDD CS=076A IP=029F NU UP EI PL NZ NA PO NC
076A:029F 16 PUSH SS
-d ss:ffa8
0BDD:FFA0 B4 05 DD 0B DE FF 3C 02 <
0BDD:FFB0 02 00 66 66 06 40 E2 7A-14 AE 47 E1 00 40 D0 FF 自增后 ff .e.z..G..e..
0BDD:FFC0 DD 0B 01 00 CD CC 8C 3F-C3 F5 2B 5C 8F C2 F1 3F 自增前 ...?...(\...?
0BDD:FFD0 00 00 00 00 DE FF 70 0F-EA FF FA 01 6A 07 FA FF ...?P...?...
```

```
1 // add sp,14
2 */
3 }
4
5 /*
6 proc near
7 push bp
8 mov bp,sp
9 */
10 struct returnStruct f(struct returnStruct r){
11 r.i_num++; /*
12 inc word ptr [bp+4]
13 */
14 r.f_num++; /*
15 FLD dword ptr [bp+6]
16 FLD类似于 PUSH指令
17 FADD qword ptr DGROUP:s@+9
18 FADD类似于 ADD指令
19 FSTP dword ptr [bp+6]
20 FSTP类似于 POP指令
21 FWAIT
22 */
23 r.d_num++; /*
24 FLD qword ptr [bp+10]
25 FADD qword ptr DGROUP:s@+9
26 FSTP qword ptr [bp+10]
27 FWAIT
28 */
29 return r; /*
30 mov bx,offset DGROUP:b@
31 push ds
32 push bx
33 lea bx,word ptr [bp+4]
34 push ss
```



```
33 push bx
34 mov cx,14
35 call far ptr SCOPY@
36 ; ?debug L 29
37 mov ax,offset DGROUP:b@
38 jmp short @2
39 @2:
40 ; ?debug L 30
41 pop bp
42 ret
43 */
44 }
```

可以发现，结构体返回是把结果复制到内存空间中然后返回该内存空间的首地址。