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

```
C:\>DEBUG.EXE MYC0S.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

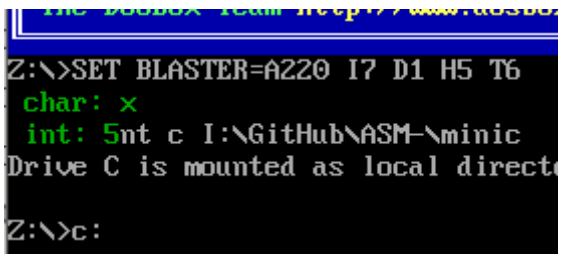
自己编写的printf

- ```
  1   void myPrintf(char *, ...);
  2
  3   main()
  4   {
  5       myPrintf(" char: %c%n int: %d", 'x',
      5);
  6   }
  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] ==
      'c')
  20              {
  21                  *(char far *)(0xb8000000
      + screenBenchmark + screenIndex) = *(char
      *)(_BP + 6 + stackIndex);
  22                  *(char far *)(0xb8000000
      + screenBenchmark + screenIndex + 1) = 2;
  ```
  ```

```
                screenIndex += 2;
                stringIndex += 2;
                stackIndex += 2;
            }
            else if (str[stringIndex + 1] == 'd')
            {
                *(char far *)(0xb8000000 + screenBenchmark + screenIndex) = *(char *)(_BP + 6 + stackIndex) + 0x30;
                *(char far *)(0xb8000000 + screenBenchmark + screenIndex + 1) = 2;

                stackIndex += 2;
                screenIndex += 2;
                stringIndex += 2;
            }
            else if (str[stringIndex + 1] == 'n')
            {
                screenBenchmark += 160;
                screenIndex = 0;
                stringIndex += 2;
            }
        }
        else
        {
            *(char far *)(0xb8000000 + screenBenchmark + screenIndex) = str[stringIndex];
            *(char far *)(0xb8000000 + screenBenchmark + screenIndex + 1) = 2;

            screenIndex += 2;
            stringIndex += 1;
        }
    }
}
```

- 

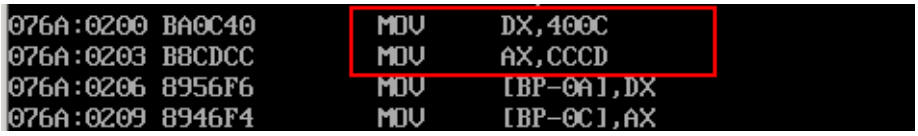
各种数据类型是通过什么返回的

float(注释中为对应的汇编代码)

```
float f(float, float);

/*
_main   proc    near
    push    bp
    mov bp,sp
    sub sp,12
*/
main()
{
        // ax=cccd dx= 400c
    float c = 2.2, a = 1.1, b = 1.1; /*
                                        mov dx,16396
                                        mov ax,-13107
                                        mov word ptr
[bp-10],dx
                                        mov word ptr
[bp-12],ax
```

下图为2.2对应的十六进制




```
                                    /* ; ?debug  L 11
                                        mov dx,16268
                                        mov ax,-13107
                                        mov word ptr
[bp-6],dx
                                        mov word ptr
[bp-8],ax
                                     ;  ?debug  L 11
                                        mov dx,16268
                                        mov ax,-13107
                                        mov word ptr
[bp-2],dx
                                        mov word ptr
[bp-4],ax
                                     */

    c = f(a, b); /*
                                        mov dx,word
ptr [bp-2]
                                        mov ax,word
ptr [bp-4]
                                        push    dx
                                        push    ax
                                        mov dx,word
ptr [bp-6]
```

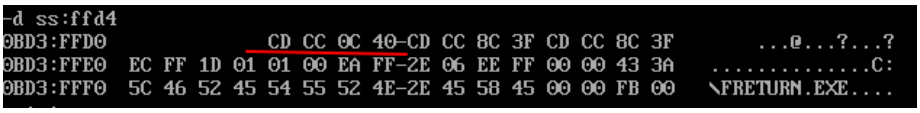
```
19                                          mov ax,word
    ptr [bp-8]
20                                          push    dx
21                                          push    ax
22                                          call    near
    ptr _f
```

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   }
```

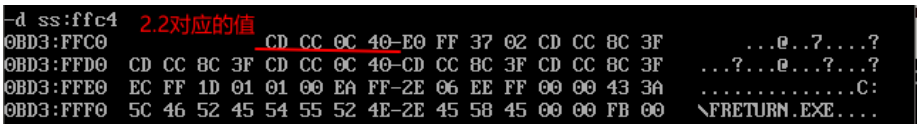
最终结果在栈中




```
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

对应汇编代码中的 `[bp-4]` 查看 `ss:ffc4` 中的值




```
                        /*
                        ;   ?debug  L 50
                            mov sp,bp
                            pop bp
                            ret
                        */
}
```

返回

结构体（注释中为对应的汇编代码）

```
struct returnStruct
{
    int i_num;
    float f_num;
    double d_num;
};

struct returnStruct f(struct returnStruct);


/*
    proc    near
    push    bp
    mov bp,sp
    sub sp,28
*/

main()

```

此时各个寄存器的值


```
{
    struct returnStruct s;
    struct returnStruct r;

    s.i_num = 1;        //  mov word ptr [bp-28],1
    s.f_num = 1.1;      /*
                            mov dx,16268
                            mov ax,-13107
```

```
                            mov word ptr [bp-24],dx
                            mov word ptr [bp-26],ax
                        */
    s.d_num = 1.11;     /*
                            mov word ptr [bp-16],16369
                            mov word ptr [bp-
18],-15729
                            mov word ptr [bp-20],23592
                            mov word ptr [bp-22],-2621
                        */

```

数据入栈后

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

```

    r=f(s);     /*
                    lea bx,word ptr [bp-14]
                    push    ss
                    push    bx
                    lea bx,word ptr [bp-28]
                    mov dx,ss
                    mov ax,bx
                    mov cx,14
                    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   NV 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   NV 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 DE FF-FD 01 6A 02 46 23 E0 FF                  j F#
```

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

```
                //  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   NV 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   NV 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 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   NV 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   NV 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  ..ff.@.z..G..@..
0BDD:FFC0  DD 0B 01 00 CD CC 8C 3F-C3 F5 28 5C 8F C2 F1 3F  .......?..(\...?
0BDD:FFD0  00 00 00 00 DE FF 70 0E-EA FF FA 01 6A 07 EA FF  ......p.....j...
```

自增后

自增前

```
                //  add sp,14
                */
}

/*
    proc    near
    push    bp
    mov bp,sp
*/
struct returnStruct f(struct returnStruct r){
    r.i_num++;      /*
                        inc word ptr [bp+4]
                    */
    r.f_num++;      /*
                        FLD dword ptr [bp+6]
FLD类似于   PUSH指令
                        FADD    qword ptr DGROUP:s@+9
FADD类似于  ADD指令
                        FSTP    dword ptr [bp+6]
FSTP类似于  POP指令
                        FWAIT
                    */
    r.d_num++;      /*
                        FLD qword ptr [bp+10]
                        FADD    qword ptr DGROUP:s@+9
                        FSTP    qword ptr [bp+10]
                        FWAIT
                    */

    return r;       /*
                        mov bx,offset DGROUP:b@
                        push    ds
                        push    bx
                        lea bx,word ptr [bp+4]
                        push    ss
```

```
                        push    bx
                        mov cx,14
                        call    far ptr SCOPY@
                        ;   ?debug  L 29
                        mov ax,offset DGROUP:b@
                        jmp short @2
                        @2:
                        ;   ?debug  L 30
                        pop bp
                        ret
                    */
}
```

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。