王爽汇编第九章,转移指令的原理

1. 转移指令
2. 操作符offset
3. jmp指令
4. 根据位移进行转移的jmp指令
5. 插播HOOK知识
5.1. Inline Hook
5.2. Inline Hook 原理
5.3. Hook代码开发
6. 转移的目的地址在指令中的jmp指令
7. 转移地址在寄存器或内存中的jmp指令
8. jcxz指令和loop指令
8.1. jcxz指令
8.2. loop指令
9. 几种跳转指令和对应的机器码

1. 转移指令

可以修改IP，或同时可以修改CS和IP的指令统称为：转移指令

8086CPU 的 转移行为 有以下几类：

- 只修改IP时，称为段内转移，比如：jmp ax。
- 同时修改CS和IP时，称为段间转移，比如：jmp 1000:0。

由于转移指令对 IP的修改范围 不同，段内转移又分为：短转移和近转移。

- 短转移IP的修改范围为-128~127(0xFF)
- 近转移IP的修改范围为-32768~32767(0xFFFF)

8086CPU 的 转移指令 分为以下几类：

- 无条件转移指令（jmp）
- 条件转移指令（jnz jz..）
- 循环指令（loop）
- 过程
- 中断（int）

2. 操作符offset

操作符offset是伪指令，在汇编语言中由编译器处理的符号，它的功能是 取得标号的偏移地址

```
;*****************************
;     将s处的一条指令复制到s0处    *
;*****************************
assume cs:code

;>>>>>>>>>>>>>>>>>>>
;代码段
;>>>>>>>>>>>>>>>>>>>
code segment
main
s:    mov ax,bx         ;(mov ax,bx 的机器码占用两个字节)
      mov si,offset s   ;(获得第一条指令出的偏移地址 标号为s)
      mov di,offset s0  ;(获得标号s0处的偏移地址)
      mov ax,cs:[si]    ;将代码段处的机器码给ax
      mov cs:[di],ax    ;将第一条指令写入到s0处
s0:   nop;nop 机器码90h 占用一字节
      nop

      ;return 2 dos
      mov ax,4c00h
      int 21h
code ends
```

3. jmp指令

jmp为无条件转移指令，可以只修改IP，也可以同时修改CS和IP。

jmp指令要给出两种信息：

- (1) 转移的目的地址
- (2) 转移的距离（段间转移、段内短转移、段内近转移）

```
jmp short 标号
jmp near ptr 标号
jcxz 标号
loop 标号 等几种汇编指令，它们对 IP的修改是根据转移目的地址和转移起始地址之间的位移来进行的。在它们对应的机器码中不包含转移的目的地址，而包含的是到目的地址的位移距离。
```

4. 根据位移进行转移的jmp指令

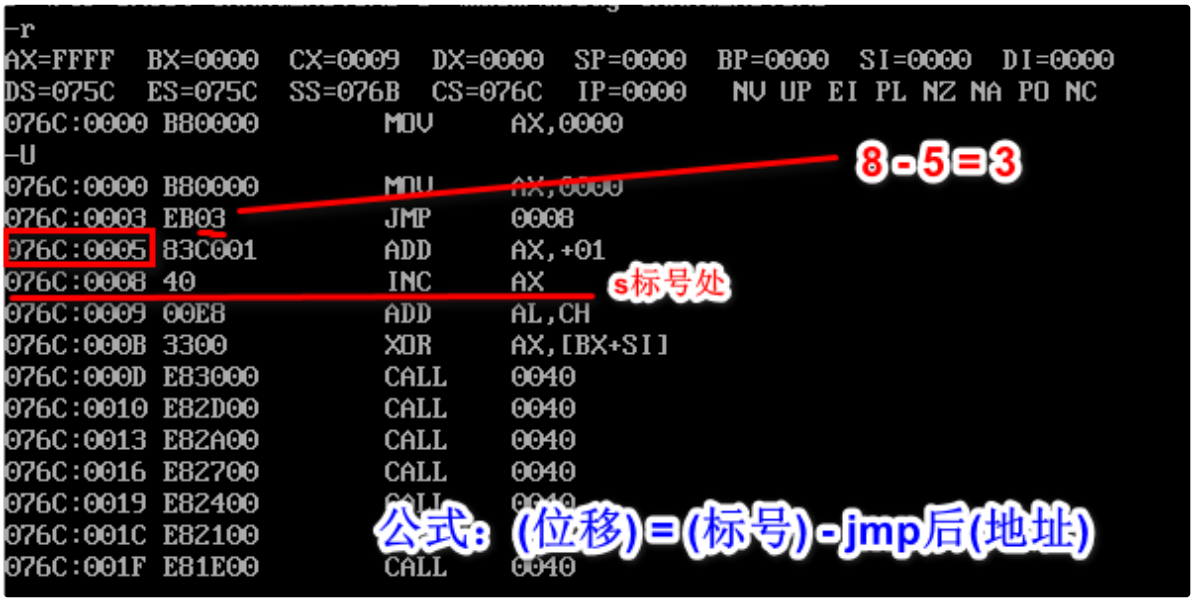
`jmp short 标号` (段内短转移)

指令“`jmp short 标号`”的功能为`(IP)=(IP)+8位位移`，转到标号处执行指令

- 8位位移 = "标号"处的地址 - jmp指令后的第一个字节的地址；
- short指明此处的位移为8位位移动；
- 8位位移的范围为(-128~127)，用补码表示。 也就是最大一个字节(0xFF)
- 8位位移由编译程序在编译时算出

```
1   assume cs:code
2
3   code segment
4   main:
5         mov ax,0
6         jmp short s; 注意s不是被翻译成了目的地址，而是根据第一条规则计算成了位移
7         add ax,1
8       s:inc ax;程序执行后,ax的值为1
9   code ends
10  end main
```

编译后标号被翻译成了位移。



CPU不需要这个目的地址就可以实现对IP的修改。这里是依据位移进行转移。

jmp short s指令的读取和执行过程如下：

- (1) (CS)=076C,(IP)=0000，执行完 `mov ax,0` 后CS:IP指向了 `EB 03` (jmp short s机器码)；
- (2) 读取指令码 `EB 03` 进入指令缓冲器；
- (3) (IP) = (IP) + 所读取指令的长度 = (IP) + 2 = 5，CS:IP指向了add ax,1；
- (4) CPU执行指令缓冲器中的指令 `EB 03` ；
- (5) 指令执行后 `IP+位移` =(IP) + 3 = 8，CS:IP(076C:0008) 指向->inc ax

5. 插播HOOK知识

这里插播点 `X86` 平台下的hook知识，看到这里我们已经知道了我们可以通过修改ip的方式来让代码跳到想要的地方执行代码。

5.1. Inline Hook

inline hook是一种通过修改机器码的方式来实现hook的技术。

在没有学汇编知识前，我们可能对Hook这种技术感到很深奥，觉得这简直是一种黑魔法，凭什么他就能把正常函数替换成我们自己的假函数。

5.2. Inline Hook 原理

当我们学习了王爽老师的汇编知识后，才明白原来在底层，CPU是根据ip寄存器来控制我们要执行的指令处的。

而学习了《王爽汇编第9章转移指令的原理》后，了解了可以通过jmp指令来修改ip寄存器，目前我们已经学习了短转移的使用，这已经足够用来学HOOK了。

```
1  #include <stdio.h>
2  #include <Windows.h>
3
4
5  //fake函数
6  void fake_demo()
7  {
8      MessageBoxA(NULL, "fake demo!");
9  }
10
11 //原函数
12 void demo()
13 {
14     MessageBoxA(NULL, "demo!", NULL, NULL);
15 }
16
17
18 int main()
19 {
20     //故意让程序暂停
21     getchar();
22
23     //执行函数
24     demo();
25 }
```

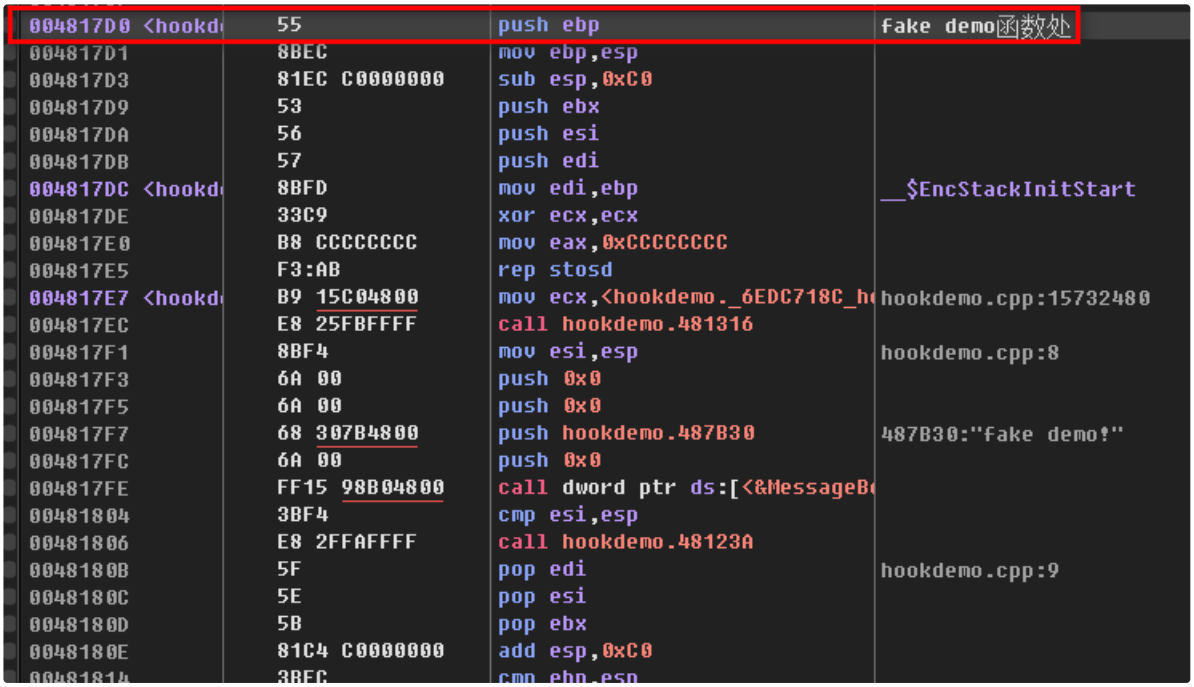
HOOK就是当我们本身要运行demo函数的时候，我们把他变成了fake_demo函数

打开OD载入程序，然后找到执行demo函数的汇编代码处，再利用我们今天刚学的短转移知识进行机器码的修改。

首先经过单步调试后，我找到了执行demo函数处的位置，我们先记下他下条指令的地址 `0x00481885` 。

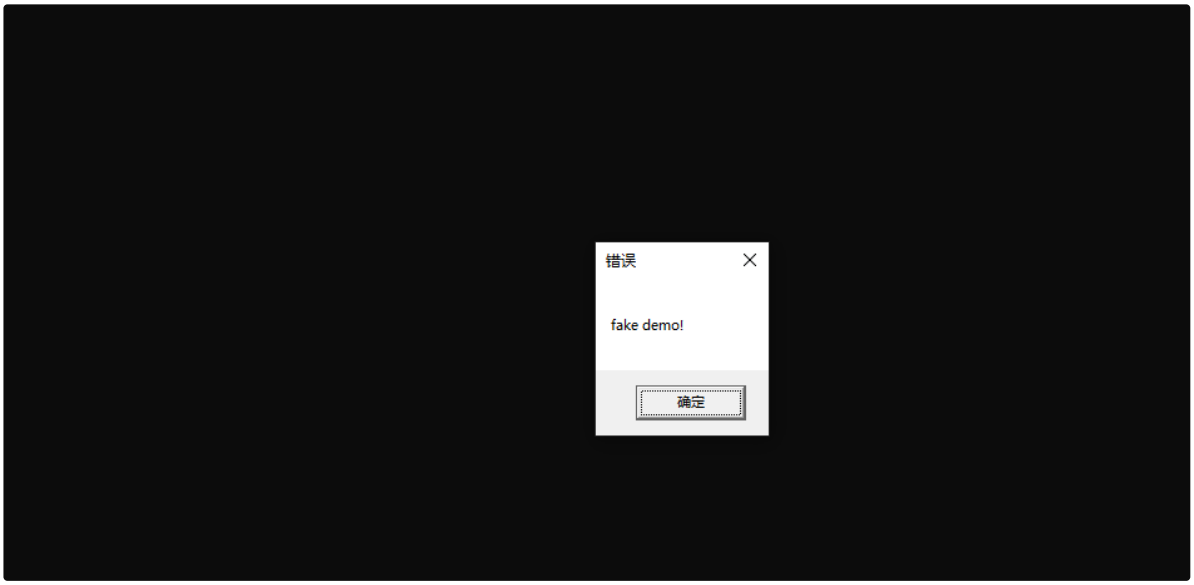
```
00481850 <hookd  55              push ebp                  hookdemo.cpp:19
00481851         8BEC            mov ebp,esp
00481853         81EC C0000000   sub esp,0xC0
00481859         53              push ebx
0048185A         56              push esi
0048185B         57              push edi
0048185C <hookd  8BFD            mov edi,ebp               __$EncStackInitStart
0048185E         33C9            xor ecx,ecx
00481860         B8 CCCCCCCC     mov eax,0xCCCCCCCC
00481865         F3:AB           rep stosd
00481867 <hookd  B9 15C04800     mov ecx,<hookdemo._6EDC718C_h  hookdemo.cpp:15732480
0048186C         E8 A5FAFFFF     call hookdemo.481316
00481871         8BF4            mov esi,esp               hookdemo.cpp:21
00481873         FF15 7CB14800   call dword ptr ds:[<&getchar>]
00481879         3BF4            cmp esi,esp
0048187B         E8 BAF9FFFF     call hookdemo.48123A
00481880         E8 E1FAFFFF     call hookdemo.481366      执行demo()函数处
00481885         33C0            xor eax,eax               hookdemo.cpp:25
00481887         5F              pop edi
00481888         5E              pop esi
00481889         5B              pop ebx
0048188A         81C4 C0000000   add esp,0xC0
00481890         3BEC            cmp ebp,esp
00481892         E8 A3F9FFFF     call hookdemo.48123A
00481897         8BE5            mov esp,ebp
```

接着我们需要找到 标号 处，也就是 `fake_demo` 函数处，也记下地址 `0x004817D0`



好了，最后的步骤就是改机器码，根据公式计算出位移： `-181(0xFFFFFF4B)`

最后来执行一遍：



5.3. Hook代码开发

在了解了Hook原理后，甚至我们用手动方式实现了HOOK后，就有了思路来代码开发了。

在此之前让我们先来认识两个函数 `WriteProcessMemory` 、 `VirtualProtect`

- WriteProcessMemory函数可以对进程的内存进行写入，这样我们就可以修改，写入机器码了。

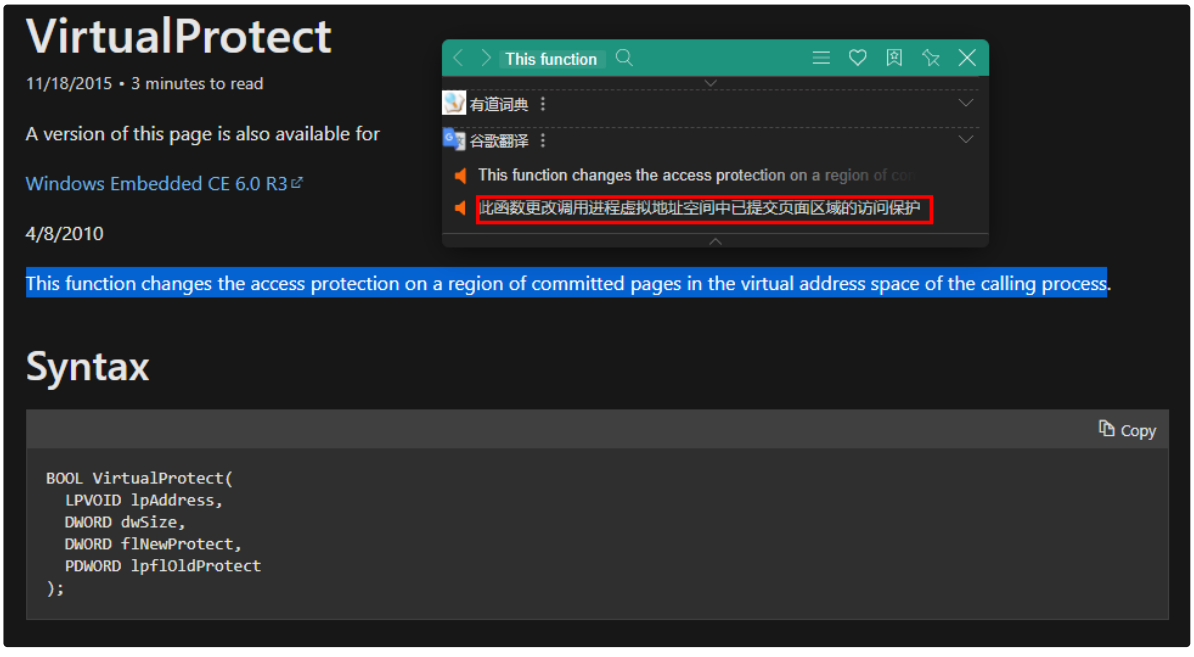
C++

此函数能写入某一进程的内存区域（直接写入会出Access Violation错误，故需此函数）。

VC++声明

```
BOOL WriteProcessMemory(
HANDLE hProcess,
LPVOID lpBaseAddress,
LPVOID lpBuffer,
DWORD nSize,
LPDWORD lpNumberOfBytesWritten
);
```

- VirtualProtect函数可以更改虚拟内存中的访问保护，在Win32中代码段具有写保护，所以我们无法直接写入，可以利用此函数修改保护后再调用WriteProcessMemory修改和写入机器码。



```
/*********************************************
 *功能：内联HOOK 函数，可以劫持函数的原本流程
 *
 *函数名：InlineHook
 *参数：(原函数地址)、(fake函数地址)
 *by：《王爽汇编笔记》
 *********************************************/
void InlineHook(DWORD dwHookAddr, LPVOID pFunAddr)
{
    BYTE jmpCode[5] = {0xE9};
    //计算偏移
    *(DWORD*)(&jmpCode[1]) = (DWORD)pFunAddr - dwHookAddr - 5;
    // 保存以前的属性用于还原
    DWORD OldProtext = 0;
    DWORD dwWritten;
    // 因为要往代码段写入数据，又因为代码段是不可写的，所以需要修改属性
    VirtualProtect((LPVOID)dwHookAddr, 5, PAGE_EXECUTE_READWRITE, &OldProtext);
    WriteProcessMemory(GetCurrentProcess(), (FARPROC)dwHookAddr, jmpCode, 5,
&dwWritten);
    VirtualProtect((LPVOID)dwHookAddr, 5, OldProtext, &OldProtext);
}

int main()
{
    //故意让程序暂停
    getchar();
    //执行函数
    InlineHook((DWORD)&demo, &fake_demo);
    demo();
}
```

6. 转移的目的地址在指令中的jmp指令

`jmp far ptr 标号`(远转移、段间转移)

- (CS) = 标号所在段的段地址;
- (IP) = 标号所在段中的偏移地址。
- far ptr 指明了指令用标号的 `段地址 和 偏移地址` 修改 `CS和IP` 。

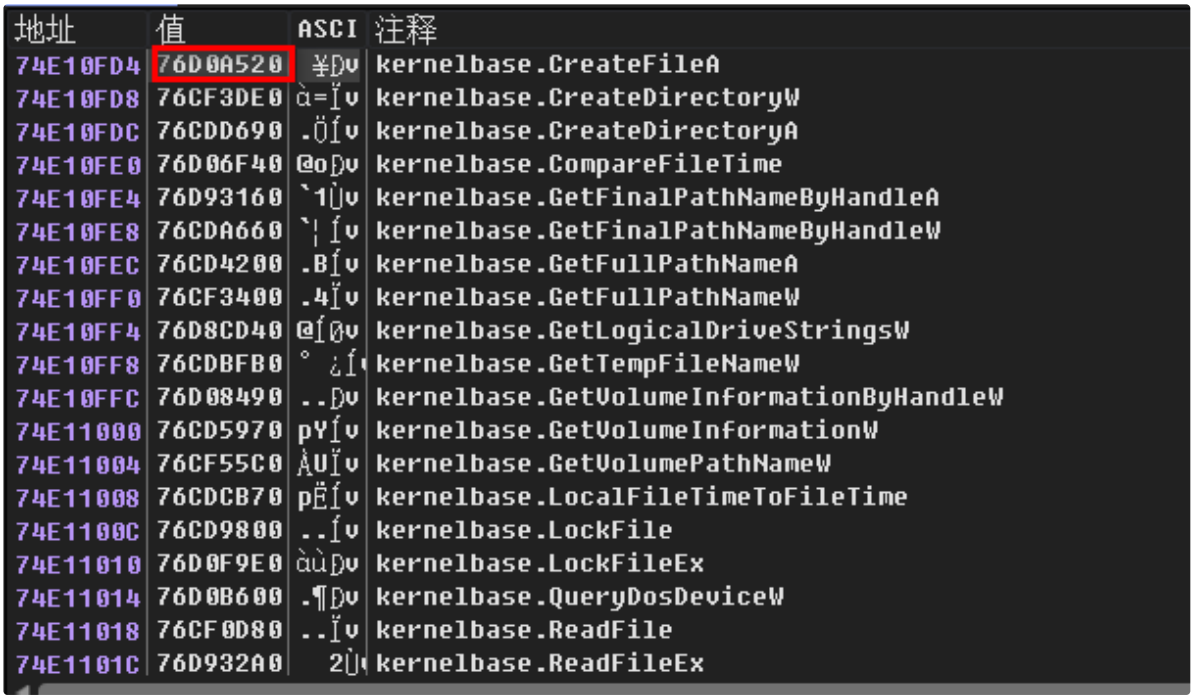
```
1   assume cs:code
2
3   code segment
4   main:
5       mov ax,0
6       mov bx,0
7       jmp far ptr s ;s被翻译成转移目的地址076C:0101
8       db 256 dup(0) ;填充256字节
9   s:  add ax,1
10      inc ax
11  code ends
12  end main
```

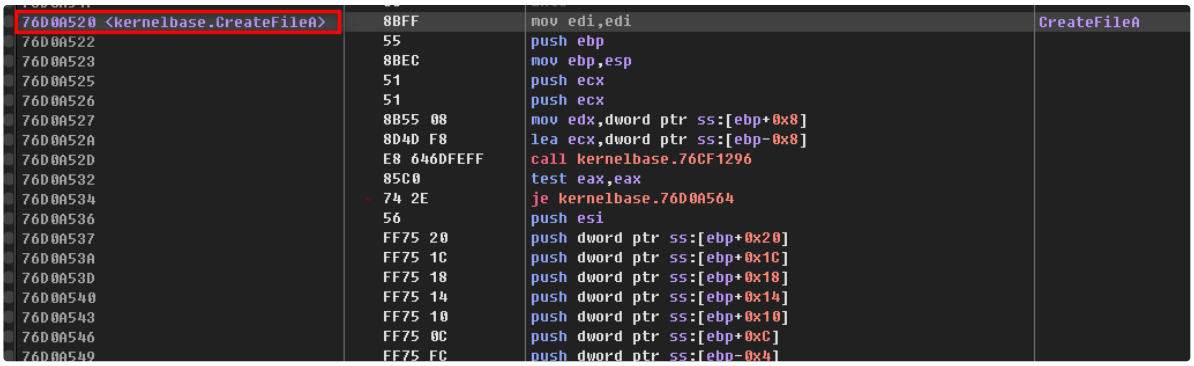
```
DOSBox 0.74-3, Cpu speed:    3000 cycles, Frameskip 0, Program:   DEBUG
D:\>if exist CHANGEAS.OBJ link CHANGEAS.OBJ; >X:\LINK.LOG


Microsoft (R) Segmented Executable Linker  Version 5.31.009
Copyright (C) Microsoft Corp 1984-1992.  All rights reserve

LINK : warning L4021: no stack segment

D:\>if exist CHANGEAS.exe c:\masm\debug CHANGEAS.exe
-U
076C:0000 B80000        MOV     AX,0000
076C:0003 BB0000        MOV     BX,0000
076C:0006 EA0B016C07    JMP     076C:010B
076C:000B 0000          ADD     [BX+SI],AL
076C:000D 0000          ADD     [BX+SI],AL
```

在 `Win32` 中的 `远转移` 可以跳到其他DLL的空间，其中关键的机器码是 `0xFF25` + 目标地址

```
004010E0    55              push ebp
004010E1    8BEC            mov ebp,esp
004010E3 <JMP.  FF25 D40FE174   jmp dword ptr ds:[<&CreateFileA>]   JMP.&CreateFileA
004010E9    0000            add byte ptr ds:[eax],al
004010EB    00C0            add al,al
004010ED    83E1 03         and ecx,0x3
004010F0    6A 00           push 0x0
004010F2    68 80000000     push 0x80
004010F7    6A 03           push 0x3
004010F9    6A 00           push 0x0
004010FB    51              push ecx
004010FC    50              push eax
004010FD    FF75 08         push dword ptr ss:[ebp+0x8]
00401100    FF15 04204000   call dword ptr ds:[<&CreateFileA>]
00401106    C9              leave
```

0x74e10fd4





7. 转移地址在寄存器或内存中的jmp指令

`jmp 16位寄存器` 功能：`IP=(16位寄存器)[段内转移]`

转移地址在内存中的jmp指令有两周格式：

- jmp word ptr [...] (段内转移)

```
1   mov ax,0123h
2   mov ds:[0],ax
3   jmp word ptr ds:[0]
4   ;ip = 0123H
```

- jmp dword ptr [...] (段间转移)

功能：从内存单元地址处开始存放着两个字，`高地址`出的字是转移目的`段地址`，`低地址`处是转移目的`偏移地址`。

1：(CS) = (内存单元地址+2)

2：(IP) = (内存单元地址)

```
mov ax,010Bh
mov ds:[0],ax
mov ax,076ch
mov ds:[2],ax
jmp dword ptr ds:[0]
;(CS) = 076C
;(IP) = 010B
;CS:IP = 076C:010B
```

8. jcxz指令和loop指令

8.1. jcxz指令

jcxz指令为有条件转移指令，所有的有条件转移指令都是短转移，在对应的机器码中包含转移的 位移 ， 而不是目的地址 。对IP的修改范围都为-128~127(0xff)。

指令格式：jcxz 标号 （如果 cx=0 ,则转移到标号处执行。）

当(cx) = 0时，(IP) = (IP) + 8位位移

- 8位位移 = "标号" 处的地址 - jcxz指令后的第一个字节的地址；
- 8位位移的范围为-128~127，用补码表示；
- 8位位移由编译程序在编译时算出。

当(cx)!=0时，程序向下执行什么都不做！

8.2. loop指令

loop指令为循环指令，所有的循环指令都是短转移，在对应的机器码中包含转移的 位移 ，而不是目的地址。

对IP的修改范围都为-128~127(0xFF)。

指令格式：loop 标号 ((cx) = (cx) - 1，如果 (cx) ≠ 0 ，转移到标号处执行)。

(cx) = (cx) - 1；如果 (cx) != 0，(IP) = (IP) + 8位位移。

- 8位位移 = ""标号""处的地址 - loop指令后的第一个字节的地址；
- 8位位移的范围为-128~127，用补码表示；
- 8位位移由编译程序在编译时算出。

如果 (cx) = 0，什么也不做（程序向下执行）。

9. 几种跳转指令和对应的机器码

注：(Win32 x86架构下)

机器码	指令	解释
0xE8	CALL	后面的四个字节是地址（近转移）
0xE9	JMP	后面的四个字节是偏移（近转移）
0xEB	JMP	后面的二个字节是偏移（近转移 8086）
0xFF15	CALL	后面的四个字节是存放地址的地址（远转移）
0xFF25	JMP	后面的四个字节是存放地址的地址（远转移）
0x68	PUSH	后面的四个字节入栈
0x6A	PUSH	后面的一个字节入栈

参考文献：

https://www.cnblogs.com/Archimedes/p/14823218.html inline hook原理和实现

https://blog.csdn.net/wzsy/article/details/17163589 几种跳转指令和对应的机器码

https://blog.csdn.net/qq_39654127/article/details/88698911 王爽《汇编语言》笔记（详细）

《王爽汇编第四版 第9章》