

《漏洞利用及渗透测试基础》实验报告

姓名：齐明杰 学号：2113997 班级：信安2班

实验名称：

IDE 反汇编实验

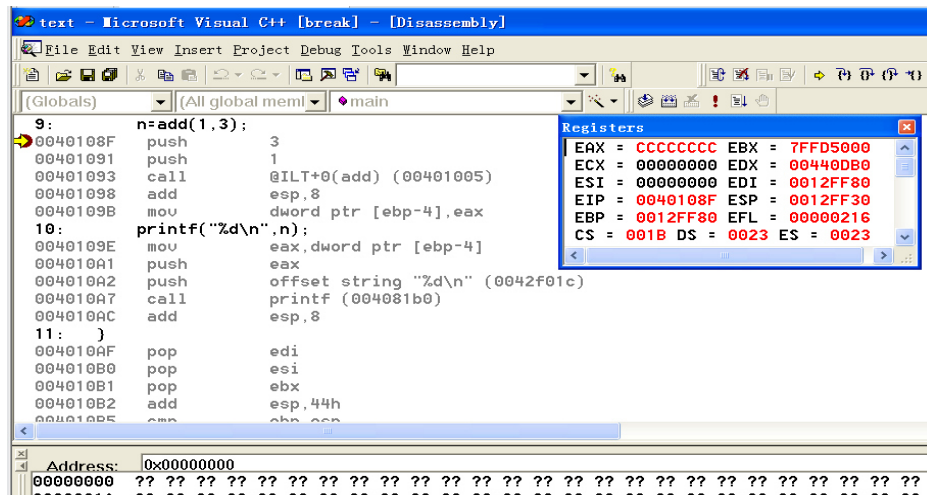
实验要求：

根据第二章示例 2-1，在 XP 环境下进行 VC6 反汇编调试，熟悉函数调用、栈帧切换、CALL 和 RET 指令等汇编语言实现，将 call 语句执行过程中的 EIP 变化、ESP、EBP 变化等状态进行记录，解释变化的主要原因。

实验过程：

1. 进入 VC 反汇编

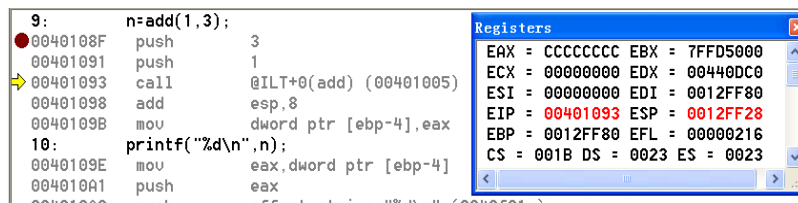
在 n=add(1,3) 语句设置断点，按下 F5 运行，然后右键” Go disassembly” 进入到反汇编窗口，如下图：



2. 观察 add 函数调用前后语句

(1) 参数入栈：add 函数的参数为 1 和 3，但首先 push 3，然后再 push 1，即参数是反向 push 入栈的。

(2) 执行 call 语句时，完成两个功能，即保存当前 eip 的值到栈中，然后跳转到所调用的函数。



执行 call 语句后(步过)，函数最终的返回值保存在寄存器 eax 中。如下图：

```

9:      n=add(1,3);
0040108F  push    3
00401091  push    1
00401093  call    @ILT+0(add) (00401005)
00401098  add     esp,8
0040109B  mov     dword ptr [ebp-4],eax
10:     printf("%d\\n",n);
0040109E  mov     eax,dword ptr [ebp-4]
004010A1  push    eax
004010A2  push    offset string "%d\\n" (0042F01C)
004010A7  call    printf (004081B0)
004010AC  add     esp,8

```

(3) 还原栈的状态：执行函数后，add esp,8 用于还原调用前 esp 的值，对应上面的两个 push。

```

9:      n=add(1,3);
0040108F  push    3
00401091  push    1
00401093  call    @ILT+0(add) (00401005)
00401098  add     esp,8
0040109B  mov     dword ptr [ebp-4],eax
10:     printf("%d\\n",n);
0040109E  mov     eax,dword ptr [ebp-4]
004010A1  push    eax
004010A2  push    offset string "%d\\n" (0042F01C)
004010A7  call    printf (004081B0)
004010AC  add     esp,8

```

(4) 保存返回值：调用 add 函数后，mov dword ptr [ebp-4],eax 将返回值赋给了 main 函数的局部变量 n 中。如下图：

```

9:      n=add(1,3);
0040108F  push    3
00401091  push    1
00401093  call    @ILT+0(add) (00401005)
00401098  add     esp,8
0040109B  mov     dword ptr [ebp-4],eax
10:     printf("%d\\n",n);
0040109E  mov     eax,dword ptr [ebp-4]
004010A1  push    eax
004010A2  push    offset string "%d\\n" (0042F01C)
004010A7  call    printf (004081B0)
004010AC  add     esp,8
11:     }
004010AF  pop     edi
004010B0  pop     esi
004010B1  pop     ebx
004010B2  add     esp,44h
004010B5  cmp     ebp,esp

```

3. add 函数内部栈帧切换等关键汇编代码

进入 add 函数的反汇编代码，如下图：

```

2:     int add(int x,int y){
00401030  push    ebp
00401031  mov     ebp,esp
00401033  sub     esp,44h
00401036  push    ebx
00401037  push    esi
00401038  push    edi
00401039  lea     edi,[ebp-44h]
0040103C  mov     ecx,11h
00401041  mov     eax,0CCCCCCCCh
00401046  rep stos dword ptr [edi]
3:     int z=0;
00401048  mov     dword ptr [ebp-4],0
4:     z=x+y;
0040104F  mov     eax,dword ptr [ebp+8]
00401052  add     eax,dword ptr [ebp+0Ch]
00401055  mov     dword ptr [ebp-4],eax

```

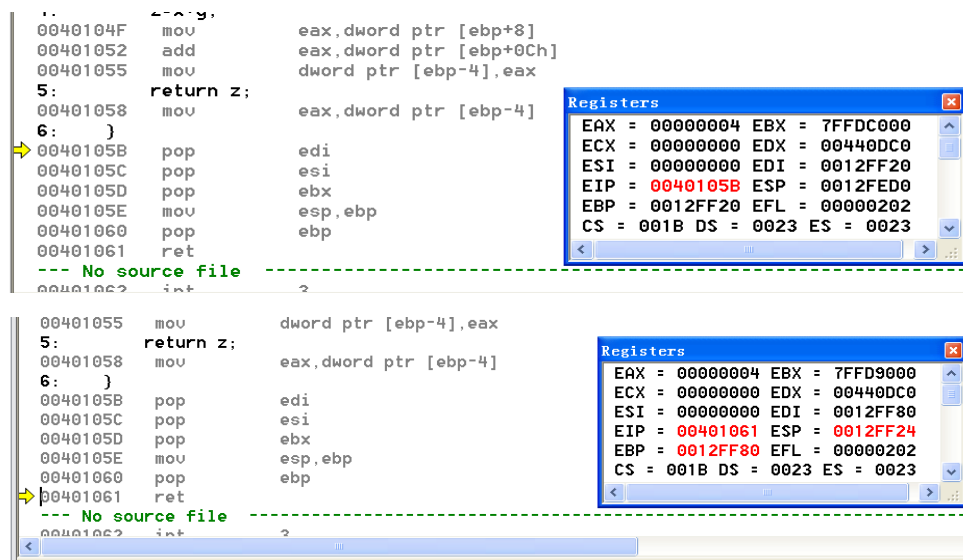
其中栈帧转移首先将 ebp 入栈，然后把 esp 的值赋给 ebp，再把 esp 抬高(即减小)，结果如下：

这几行指令完成了 $z=x+y$ 的加法操作。值得注意的是 x 先赋值给了 eax ，然后使用了 add 指令计算加法，最终结果是保存在 eax 中的。

然后将 eax 的值取出，同时三个 pop 恢复寄存器状态，对应开始的三个 $push$ 。

栈帧恢复： $mov\ esp,ebp$ ，即把当前栈帧清除；

$pop\ ebp$ ，对应函数头的 $push\ ebp$ ，即恢复到 $main$ 函数栈帧的值。



```
0040104F    mov     eax,dword ptr [ebp+8]
00401052    add     eax,dword ptr [ebp+0Ch]
00401055    mov     dword ptr [ebp-4],eax
5:      return z;
00401058    mov     eax,dword ptr [ebp-4]
6:      )
0040105B    pop     edi
0040105C    pop     esi
0040105D    pop     ebx
0040105E    mov     esp,ebp
00401060    pop     ebp
00401061    ret
```

--- No source file ---

Registers

EAX = 00000004	EBX = 7FFDC000
ECX = 00000000	EDX = 00440DC0
ESI = 00000000	EDI = 0012FF20
EIP = 0040105B	ESP = 0012FED0
EBP = 0012FF20	EFL = 00000202
CS = 001B	DS = 0023
ES = 0023	

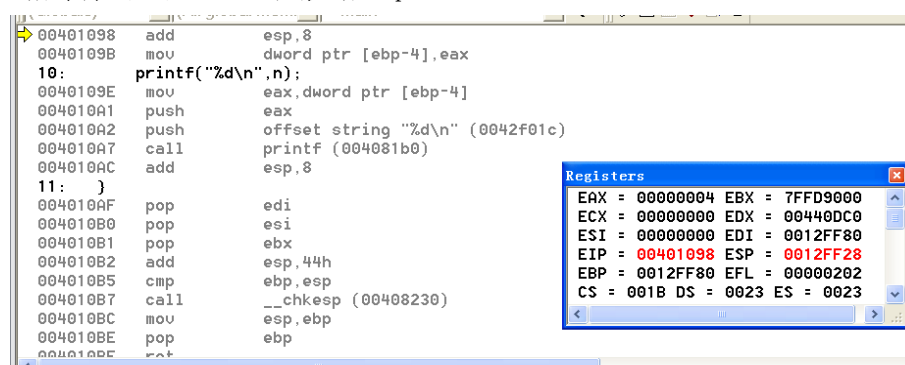
```
00401055    mov     dword ptr [ebp-4],eax
5:      return z;
00401058    mov     eax,dword ptr [ebp-4]
6:      )
0040105B    pop     edi
0040105C    pop     esi
0040105D    pop     ebx
0040105E    mov     esp,ebp
00401060    pop     ebp
00401061    ret
```

--- No source file ---

Registers

EAX = 00000004	EBX = 7FFD9000
ECX = 00000000	EDX = 00440DC0
ESI = 00000000	EDI = 0012FF80
EIP = 00401061	ESP = 0012FF24
EBP = 0012FF80	EFL = 00000202
CS = 001B	DS = 0023
ES = 0023	

之后 ret 指令弹出返回地址，赋值给 eip ：



```
00401098    add     esp,8
0040109B    mov     dword ptr [ebp-4],eax
10:     printf("%d\n",n);
0040109E    mov     eax,dword ptr [ebp-4]
004010A1    push    eax
004010A2    push    offset string "%d\n" (0042f01c)
004010A7    call    printf (004081b0)
004010AC    add     esp,8
11:     )
004010AF    pop     edi
004010B0    pop     esi
004010B1    pop     ebx
004010B2    add     esp,44h
004010B5    cmp     ebp,esp
004010B7    call    __chkesp (00408230)
004010BC    mov     esp,ebp
004010BE    pop     ebp
004010BF    ret
```

Registers

EAX = 00000004	EBX = 7FFD9000
ECX = 00000000	EDX = 00440DC0
ESI = 00000000	EDI = 0012FF80
EIP = 00401098	ESP = 0012FF28
EBP = 0012FF80	EFL = 00000202
CS = 001B	DS = 0023
ES = 0023	

回到主函数， add 函数执行结束。

心得体会：

通过实验，掌握了栈帧的切换以及函数调用的过程。

RET 指令实际就是执行了 $Pop\ EIP$

此外，通过本实验，掌握了多个汇编语言的用法，如 $rep\ stos$