

《软件安全》实验报告

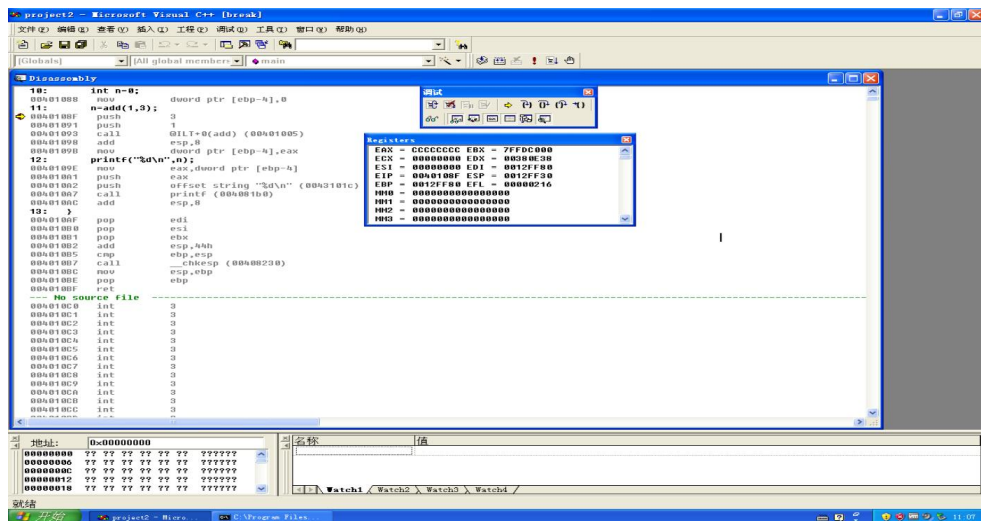
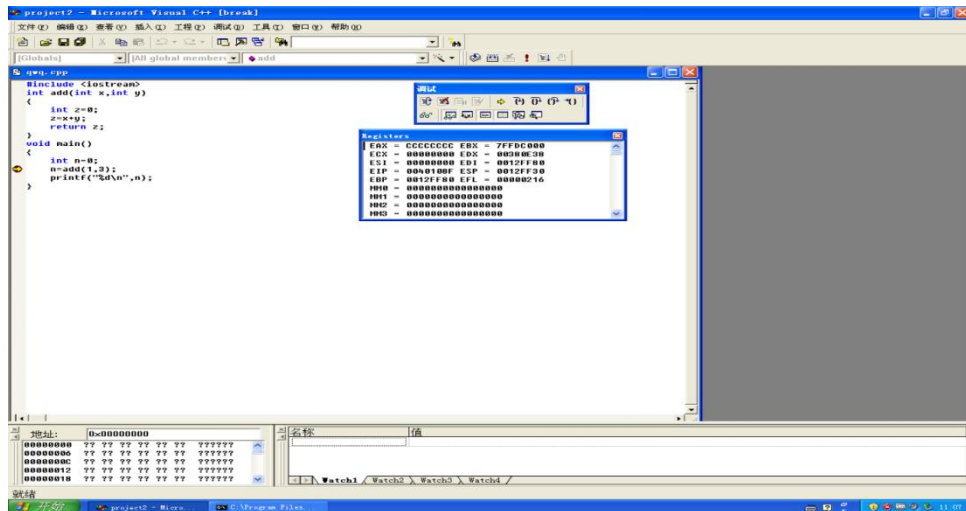
姓名：王昱 学号：2212046 班级：信息安全班

一、实验名称： IDE 反汇编实验

二、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。

三、实验过程： 1. 进入 VC 反汇编

安装好 Windows XP 环境后，下载 VC 6.0 并打开,创建好项目后将代码粘贴到程序中并在函数调用处设置断点，进入反汇编。



2. 观察 add 函数调用前后语句



```
Disassembly
10:      int n=0;
00401088      mov     dword ptr [ebp-4],0
11:      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
12:      printf("%d\n",n);
```

(1) 变量 n 初始化。

初始的时候，ebp 是主函数栈帧的基址，而 n 是主函数的一个局部变量，而栈是向低地址增长，所以[ebp-4]就是变量 n 的地址，符合栈的结构。因此 mov dword ptr [ebp-4],0 的作用就是将 0 的值赋值给 n。

(2) 参数入栈

由于参数是从右向左入栈，所以先 push 3，再 push 1。



```
ECX = 00000000 EDI = 00380E38
ESI = 00000000 EDI = 0012FF80
EIP = 00401093 ESP = 0012FF28
EBP = 0012FF80 EFL = 00000216
```

观察两次 push 之后的 ESP 变化，从 0012FF30 到 0012FF2C 再到 0012FF28，ESP 是栈指针寄存器，参数入栈操作，ESP 向低地址增长。



```
地址: 0012FF28
0012FF28 01 00 00 00 03 00 .....
0012FF2F 00 00 00 00 00 00
```

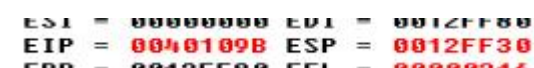
观察内存变化，push 之后 3 和 1 依次存在内存中。

(3) 调用 CALL 指令和恢复栈帧



```
00401093      call     @ILT+0(add) (00401005)
00401098      add     esp,8
```

首先 call 调用，调用后 ESP 的地址并没有变化。之后执行 add esp,与前面的入栈操作相对应，此时栈恢复到了函数调用之前的状态。



```
ESI = 00000000 EDI = 0012FF80
EIP = 0040109B ESP = 0012FF30
EBP = 0012FF80 EFL = 00000016
```

(4) 将函数执行后的返回值赋值给 n



```
0040109B      mov     dword ptr [ebp-4],eax
```

使用 mov 指令将 eax 寄存器的值，即相加后的结果赋值给 ptr [ebp-4],而[ebp-4]是 n 的地址表达式。这样就完成了整个函数执行。再按 F5 就会返回最初的界面。

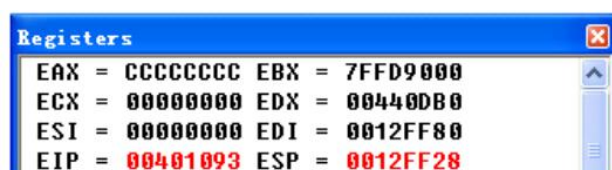
3. add 函数内部栈帧切换等关键汇编代码

(1)参数入栈

首先是参数由右向左入栈。

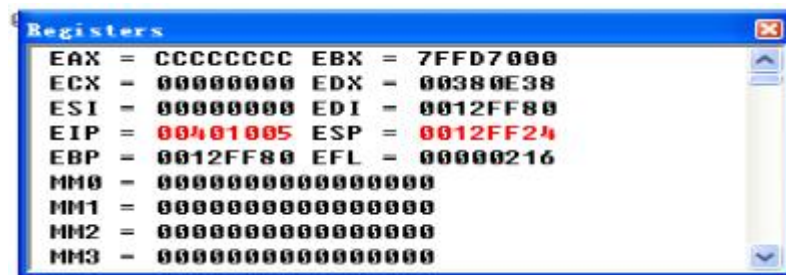
(2)返回地址入栈

遇到 call 指令之后进行返回地址入栈的操作，call 指令执行前的寄存器如下：



```
Registers
EAX = CCCCCCCC EBX = 7FFD9000
ECX = 00000000 EDI = 00400B00
ESI = 00000000 EDI = 0012FF80
EIP = 00401093 ESP = 0012FF28
```

而 call 的下一条指令即函数的返回地址是：00401098
 执行了 call 指令后 EIP 发生变化，而此时栈里存入了函数的返回地址。



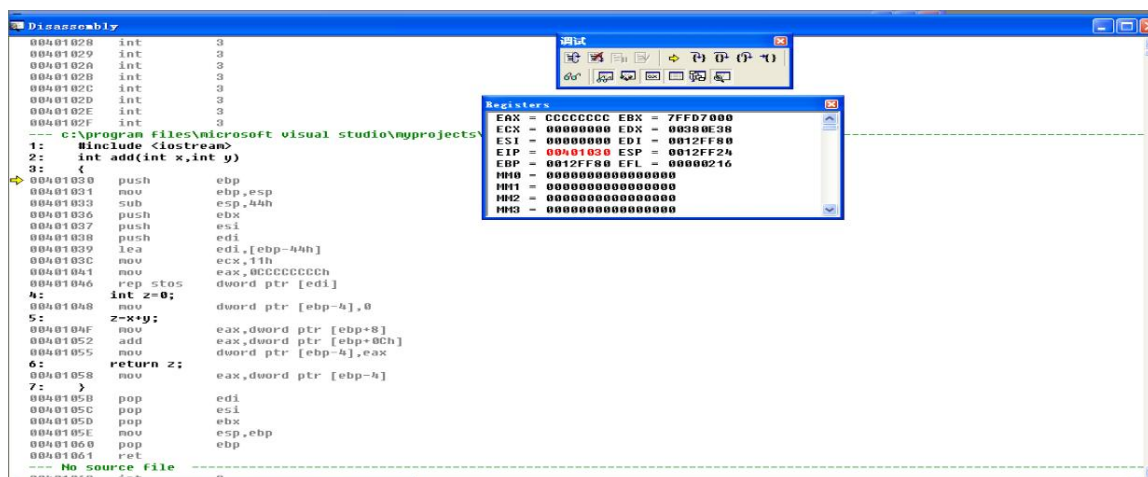
此时的 ESP 地址所存的值：



可以看到是 00401098，即函数的返回地址，完成了返回地址入栈的操作。

(3) 跳转到 add 函数

通过 step into(F11)进入到 add 函数的代码区：



(4) 函数实现具体过程



进入 add 函数内部后，首先将主函数的 EBP（0012FF80）入栈，存到 ESP（0012FF20）中，以便在函数结束后返回时恢复栈帧;将 ESP 的值赋给 EBP 作为 add 函数的基地址，将 main 函数栈帧切换到 add 函数栈帧。



(可以看到地址 0012FF20 储存着 0012FF80)



之后 ESP 减去 44h，相当于栈向上增长 44h，为 add 函数开辟新的栈帧。并且三条 push 指令分别把函数可能用到的寄存器入栈。

```
00401039 lea     edi,[ebp-44h]
0040103C mov     ecx,11h
00401041 mov     eax,0CCCCCCCCh
00401046 rep stos dword ptr [edi]
```

初始化刚才开辟的栈帧，将[ebp-44h]新空间的起始地址赋值给 edi；ecx 是计数寄存器，与循环配套使用，ecx 的值是 11h，说明循环 11h 次，共初始化 44h 的空间。

0012FF2E	00 00 00 00 00 00
0012FF34	00 00 00 00 00 70p
0012FF3A	FD 7F CC CC CC CC	?
0012FF40	CC CC CC CC CC CC	?
0012FF46	CC CC CC CC CC CC	?
0012FF4C	CC CC CC CC CC CC	?
0012FF52	CC CC CC CC CC CC	?
0012FF58	CC CC CC CC CC CC	?
0012FF5E	CC CC CC CC CC CC	?
0012FF64	CC CC CC CC CC CC	?
0012FF6A	CC CC CC CC CC CC	?
0012FF70	CC CC CC CC CC CC	?
0012FF76	CC CC CC CC CC CC	?
0012FF7C	00 00 00 00 C0 FF

可以看到循环将空白区域初始化。

```
00401040 rep stos dword ptr [edi]
4: int z=0;
00401048 mov     dword ptr [ebp-4],0
5: z=x+y;
0040104F mov     eax,dword ptr [ebp+8]
00401052 add     eax,dword ptr [ebp+0Ch]
00401055 mov     dword ptr [ebp-4],eax
```

mov dword ptr [ebp-4],0

首先是函数的局部变量 z 初始化，局部变量是后入栈的，所以是 ebp-4。

mov eax,dword ptr [ebp+8]

add eax,dword ptr [ebp+0Ch]

mov dword ptr [ebp-4],eax

由于函数参数是先入栈的，所以要 ebp+8，ebp+0Ch，相加后赋值给 z。

(5)函数返回及恢复栈帧

mov eax,dword ptr [ebp-4]

把要返回的局部变量 z 的值赋给 eax，eax 的一个重要作用就是储存返回值。

0040105B pop edi

0040105C pop esi

0040105D pop ebx

对应这三个寄存器在函数开始时候的 push，现将他们按相反的顺序弹出栈。

0040105E mov esp,ebp

恢复原函数的栈顶

00401060 pop ebp

恢复原函数的栈底

00401061 ret

当要执行 ret 指令前，ESP（0012FF24）的值是 00401098，即函数返回地址。

Address:	0x0012FF24									
0012FF24	98	10	40	00	01	00	00	00	03	..@.....
0012FF2D	00	00	00	00	00	00	00	00	00
0012FF36	00	00	00	A0	FD	7F	CC	CC	CC	...狙.烫.
0012FF3F	CC	CC	CC	CC	CC	CC	CC	CC	CC	烫烫烫烫.
0012FF48	CC	CC	CC	CC	CC	CC	CC	CC	CC	烫烫烫烫.
0012FF51	CC	CC	CC	CC	CC	CC	CC	CC	CC	烫烫烫烫.
0012FF5A	CC	CC	CC	CC	CC	CC	CC	CC	CC	烫烫烫烫.
0012FF63	CC	CC	CC	CC	CC	CC	CC	CC	CC	烫烫烫烫.
0012FF6C	CC	CC	CC	CC	CC	CC	CC	CC	CC	烫烫烫烫.
0012FF75	CC	CC	CC	CC	CC	CC	CC	00	00	烫烫烫...
0012FF7E	00	00	C0	FF	12	00	79	84	40y.斯
0012FF87	00	01	00	00	00	70	0F	44	00n.D.

执行后 EIP 变为返回地址，返回到主函数。

EIP = 00401098

add esp,8

mov dword ptr [ebp-4],eax

最后，ESP 和 EBP 完全恢复未调用函数的栈的状态。并将函数返回值从 eax 赋给 main 函数的变量 n。由此整个函数运行结束。

四、心得体会：

- 1.通过实验，掌握了 CALL 和 RET 指令的用法：call 和 ret 指令都是转移指令，通过 EIP 的入栈和出栈来实现了当前状态的保存和函数的跳转。
- 2.学会了 Windows XP 环境的配置，对软件的使用更加熟悉。
- 3.对汇编代码的基本语法和调试有了初步的认识和了解。
- 4.本次实验通过实验视频的教导，观察了函数每一步寄存器和内存地址的变化，对汇编语言的理解更加深入透彻。