《软件安全》实验报告

姓名：胡博浩 学号：2212998 班级：信息安全

**实验名称：**

IDE反汇编实验

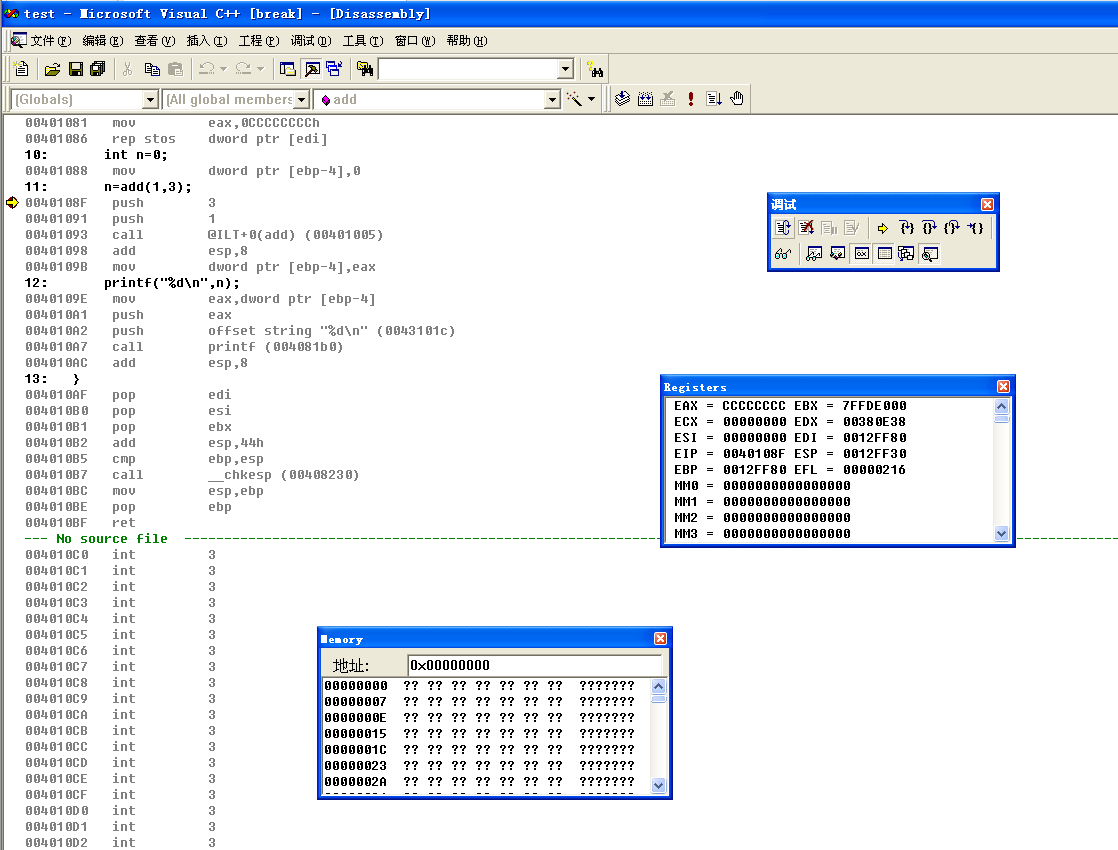
**实验要求：**

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

在主函数的n=add(1,3)语句处设置一个断点，按F5进入调试状态，然后右键选“Go disassembly”进入到反汇编窗口，如图：



2. 观察add函数调用前后语句

1）在add函数调用前，首先将add函数所需要的两个参数从右向左依次压入栈中。

0040108F push 3

00401091 push 1

2）通过call指令调用add函数，此时还做了两件事情，将EIP中的下一条指令的地址入栈即保存返回地址，再跳转到add函数所在的代码块中。

此时栈区状态为：



3）执行call语句（步过），函数的返回值保存在寄存器eax中。

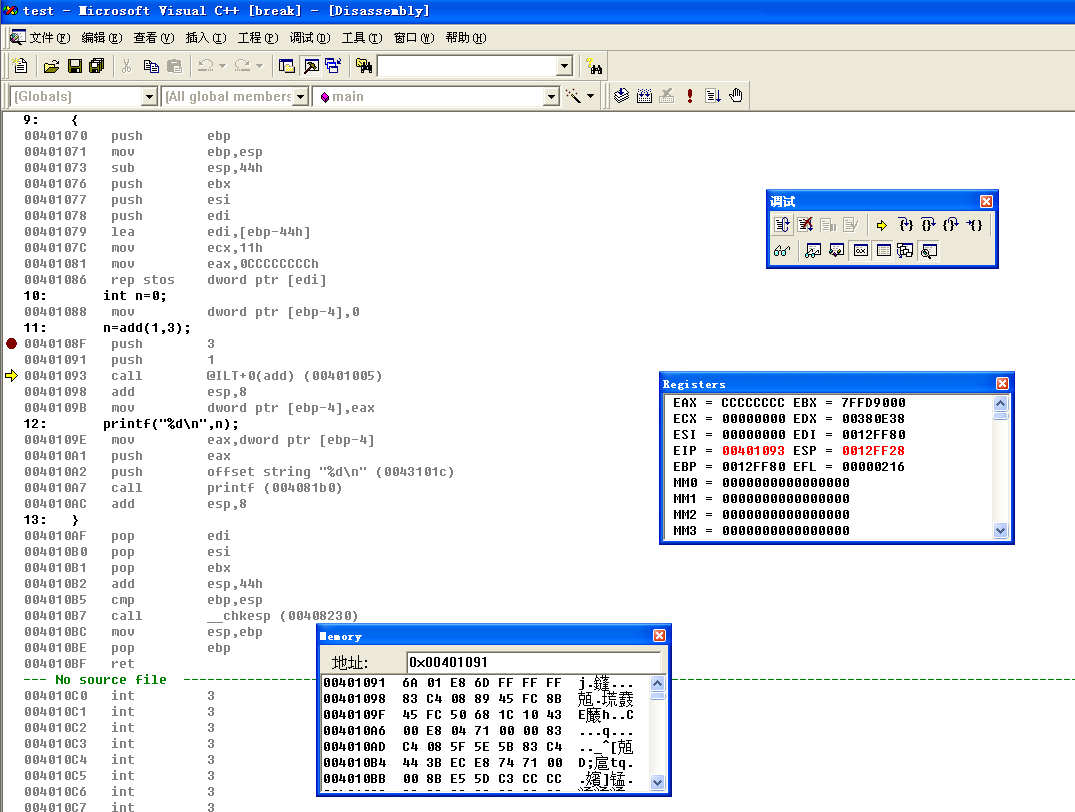
4）add函数调用完毕后，恢复栈状态，将ESP恢复到调用add函数之前的状态，然后将函数返回值从EAX寄存器中转移到局部变量n所在的地址处。

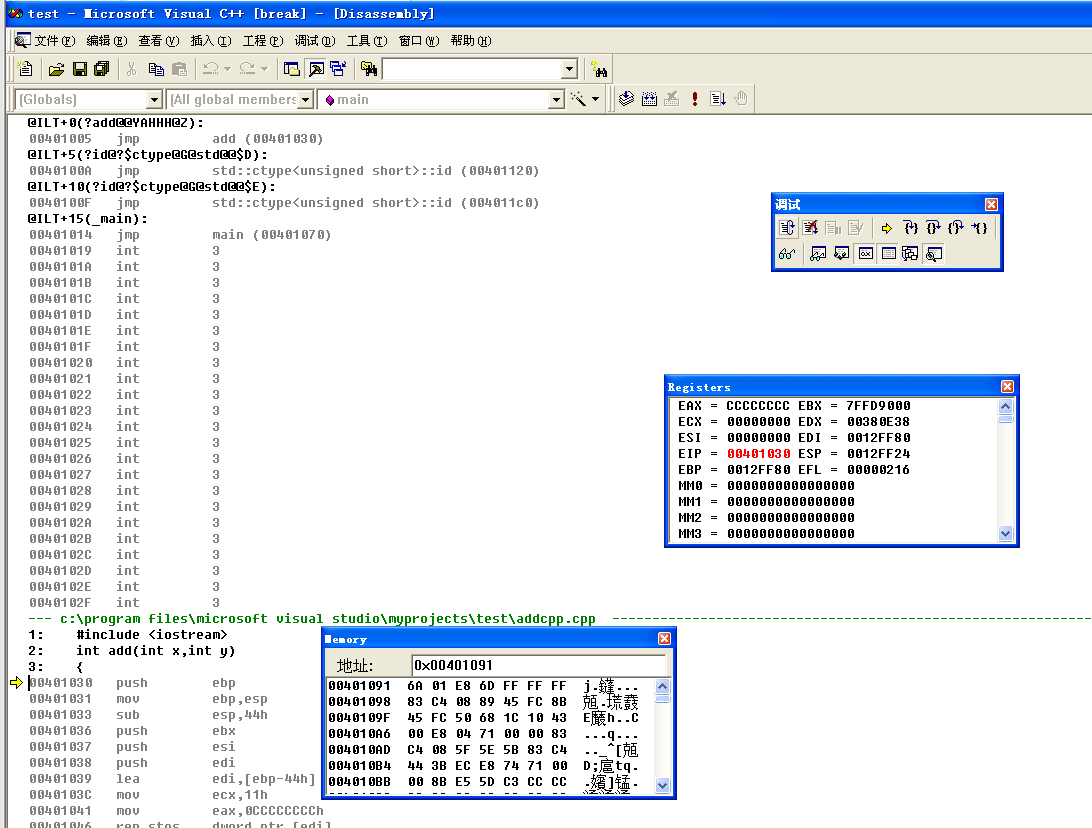
00401098 add esp,8

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

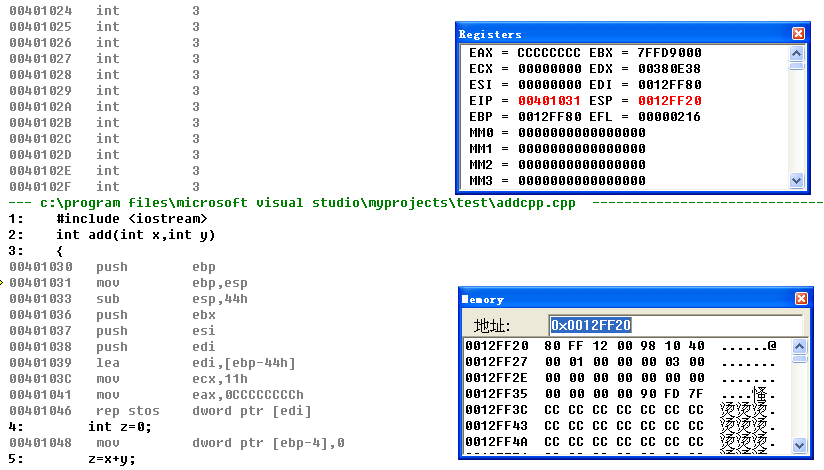
3. add函数内部栈帧切换等关键汇编代码

1）在call指令执行时，发生了两件事情，首先将EIP中的返回地址入栈，ESP-4，然后设置EIP的值，实现从main函数到add函数的跳转。





2）进入add函数内部后，首先将原始的EBP入栈，以便在函数结束后返回时恢复栈帧。



3）把esp的值赋给ebp，再把esp抬高(即减小)，开辟add函数的栈帧。

00401031 mov ebp,esp

00401033 sub esp,44h

4）将外部使用的寄存器状态保存。

00401036 push ebx

00401037 push esi

00401038 push edi

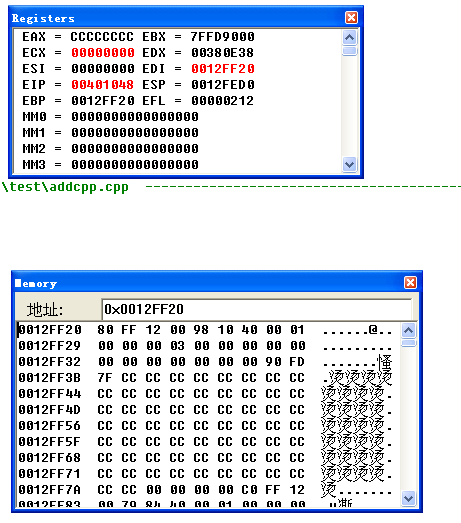
5）对栈分配的内存进行循环赋值，完成栈帧的转化。

00401039 lea edi,[ebp-44h]

0040103C mov ecx,11h

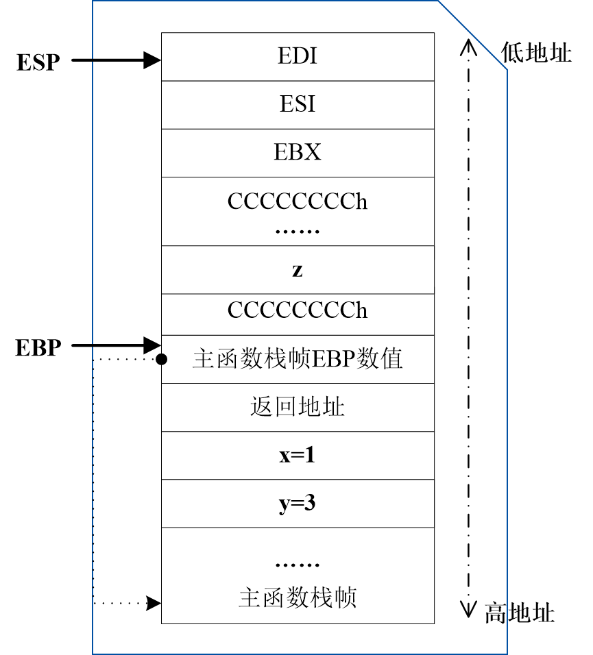
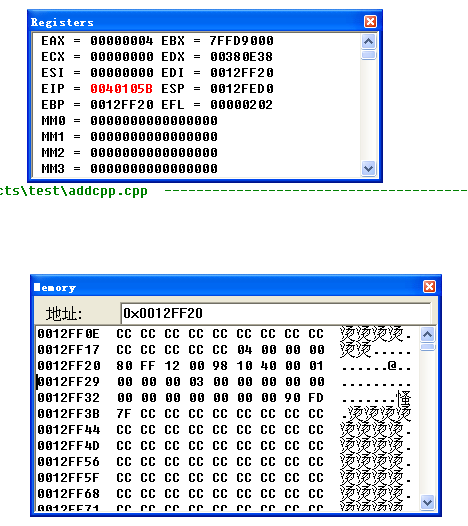
00401041 mov eax,0CCCCCCCCh

00401046 rep stos dword ptr [edi]



6）完成加法操作： x先赋值给eax，然后使用了add指令计算加法。

在add函数执行完成之后，return z；这时将函数的返回值从局部变量中转移到寄存器EAX中，实现函数结果的返回。



此时栈帧如图所示。

7）清理函数开辟的栈帧，将原始的寄存器信息出栈。

0040105B pop edi

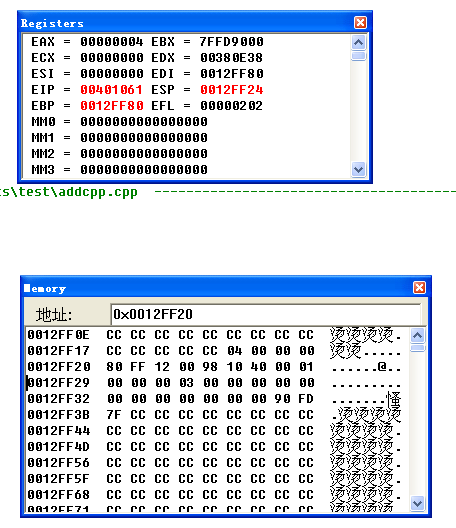
0040105C pop esi

0040105D pop ebx

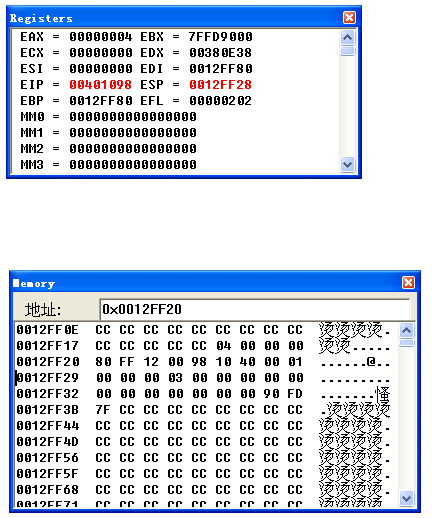
通过mov esp,ebp 清除函数开辟的栈帧，此时esp和ebp的值相同，都指向原始栈帧的栈底EBP。

然后将原始栈帧的栈底EBP的值出栈，恢复了原始的EBP状态。

00401060 pop ebp



8）ret指令完成了两件事情，将当前的栈顶ESP中保存原始EIP的值出栈到EIP中，然后跳回到调用函数的代码块中。



**心得体会：**

1. 通过本次实验，了解了函数调用过程中栈帧的调整以及函数调用的过程。
2. 清楚的认识到了不同的寄存器的作用，如EIP，ESP，EBP
3. CALL指令和RET指令，其实就是通过EIP的入栈和出栈来实现的。
4. 掌握了多个汇编语言的用法，比如说rep stos