《漏洞利用及渗透测试基础》实验报告

**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。

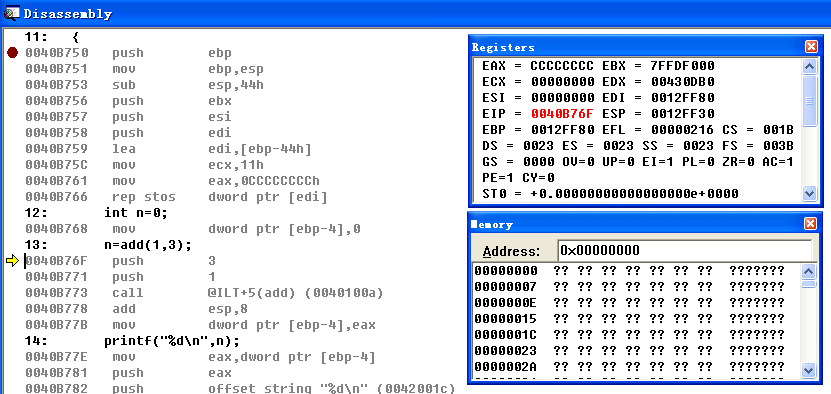
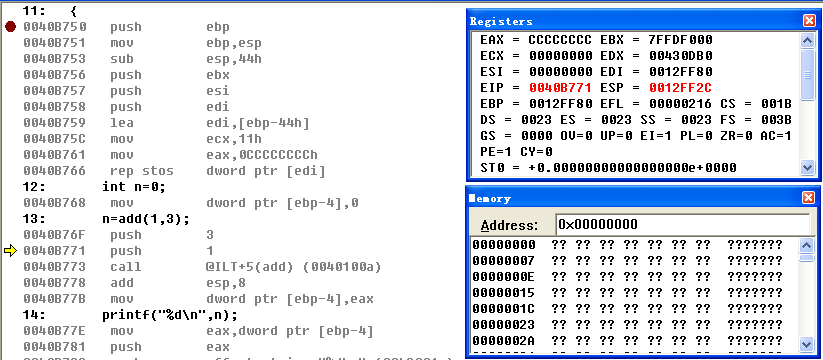
**实验过程：**

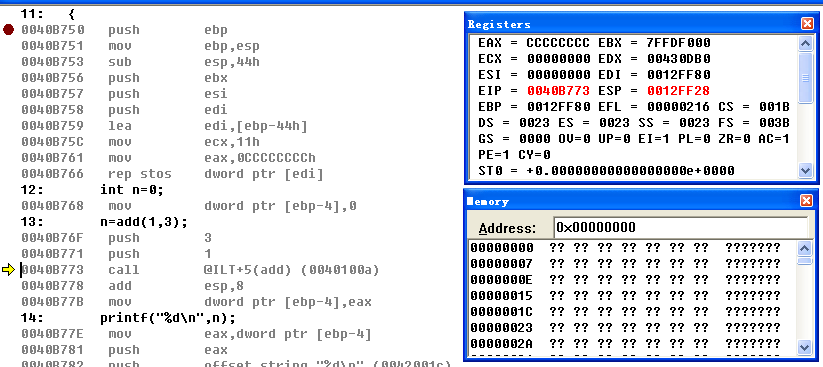
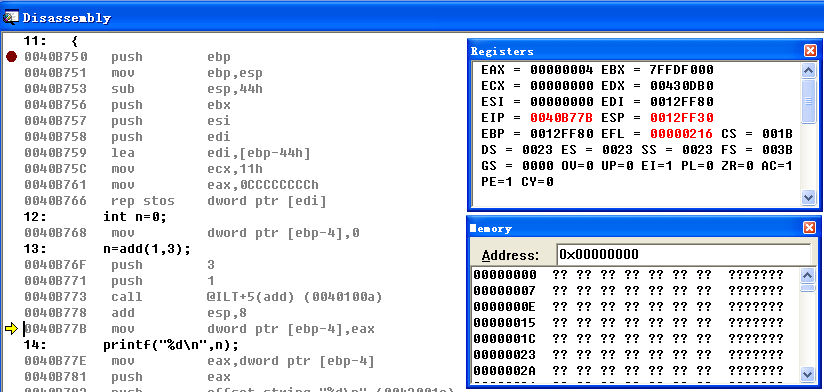
1. 进入VC反汇编

在WindowsXP系统下打开VC6新建项目和源文件并写入代码，在相应位置添加断点，F5开始调试，右键点击Go disassembly进入反汇编。

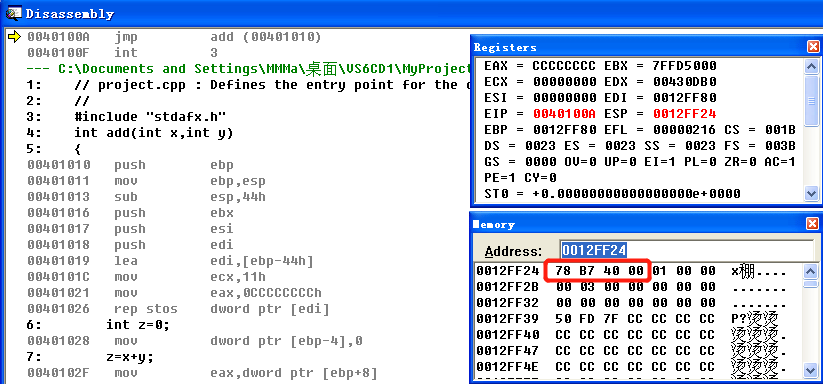
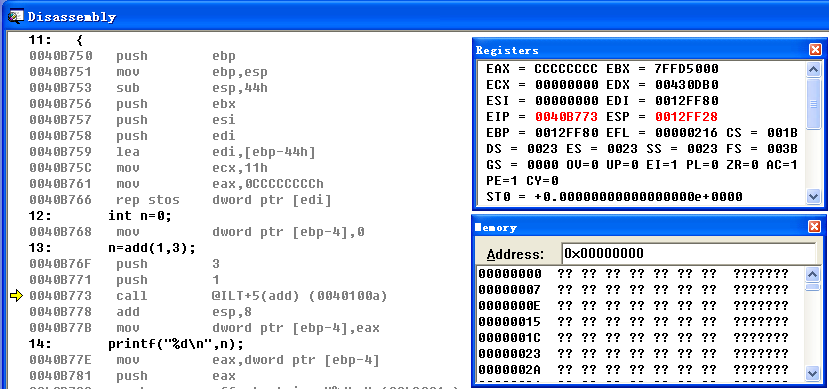
2. 观察add函数调用前后语句

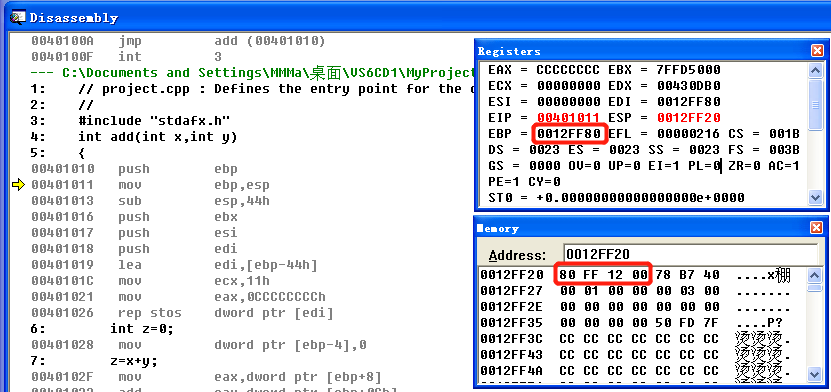
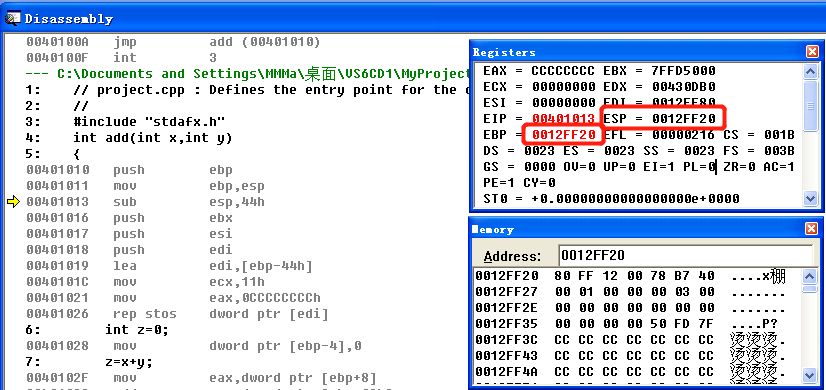
F10逐步调试，到达调用add函数的语句，调用函数之前，ESP值如图，

首先将函数的参数从左到右压入栈中，将参数3压入栈之后，如图可知ESP值减4，即栈顶指针往低地址处减了4，挪动的这块空间正好就是参数3存放的空间，之后同理将另一个参数压入栈，如图栈顶指针又减了4，即将1压入栈中。

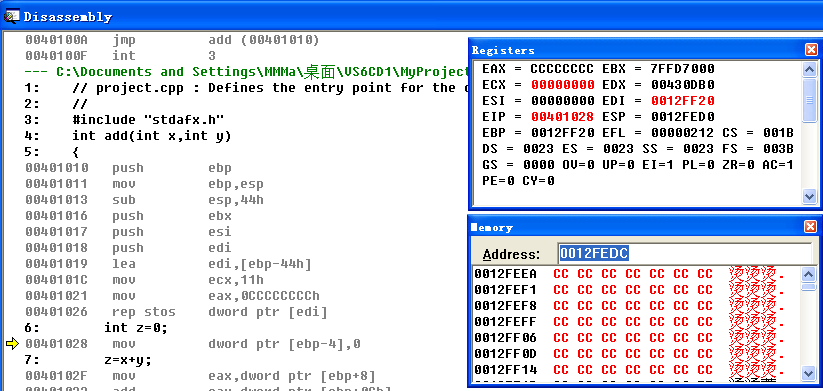
F10不跳入call语句，call完整结束之后即函数调用完成，ESP值加8，可以发现ESP的值又回到了调用add函数之前，说明函数调用结束之后的栈又退回到了调用之前的状态。

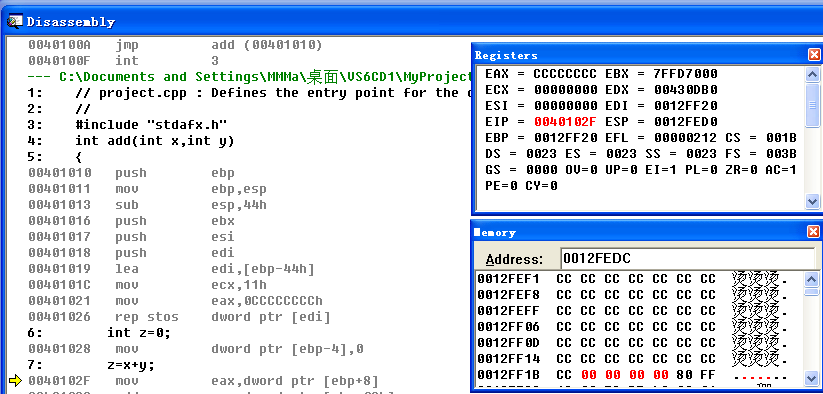
3. 观察add函数内部栈帧切换等关键汇编代码

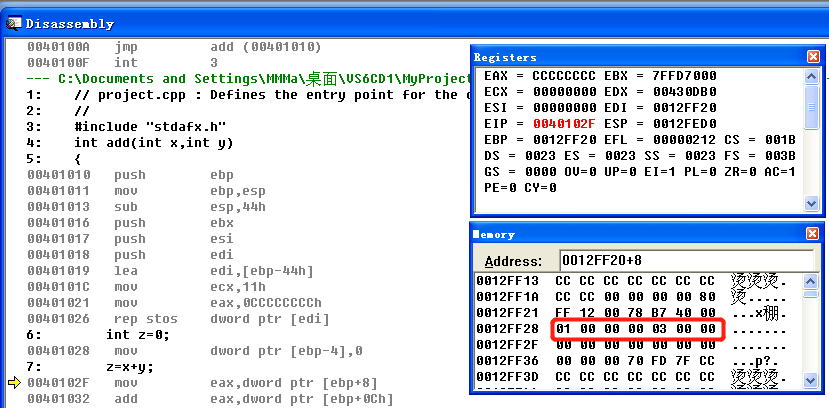
在call函数处按F11进入call函数，进入之前，EIP的值如图，为call所在地址， 进入之后可见EIP的值改变了，而ESP的值也减小，说明栈帧改变，栈中存入了东西，查看之后如图，发现内容为call之前EIP的值，即原本将顺序执行的指令地址再点击F11后跳转到了add函数的代码区，说明call函数跳转时，会先将返回地址存入栈中，这样函数结束调用时读取该值即可跳转回原本的地址，然后再跳转到要执行函数的代码区。

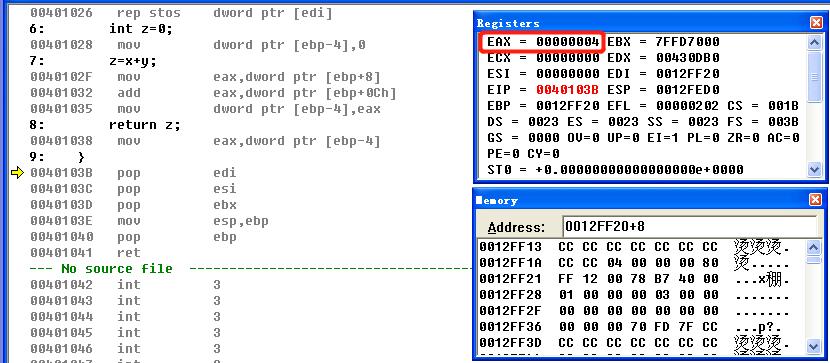
进入代码区之后，点击F11，执行了push语句将ebp压入栈中后又将esp赋值给了ebp，这一过程目的是将原本主函数的栈基地址存入栈中后，将当前栈顶esp作为栈基址，即为add函数设了栈的基址。

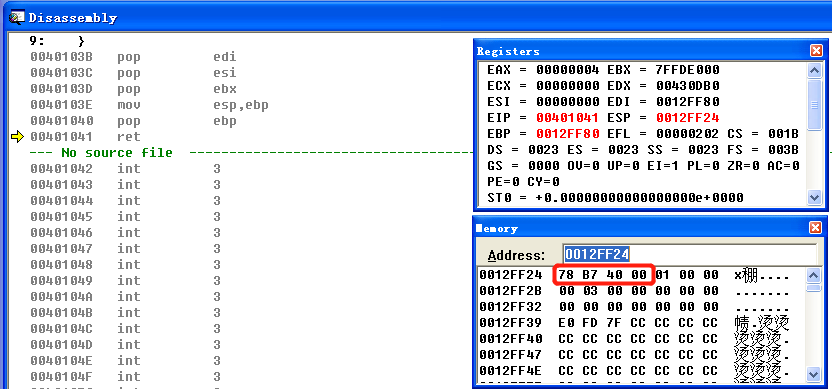
之后执行了sub语句将esp值减去44h，即为add函数开辟空间，之后连续三个push将ebx、esi、edi的值压入栈中，因此每次执行esp的值都会随之减小，之后执行了一个11h次的循环，将开辟的44h的空间初始化。根据EDI地址进入栈中也可看到栈中44h的空白栈帧区域全都初始化为CC。

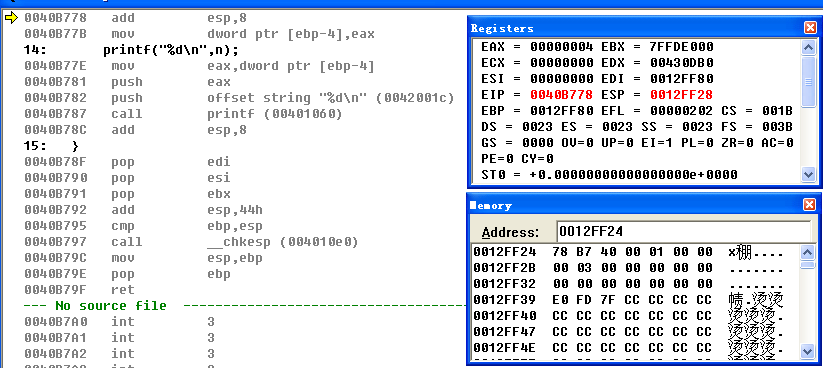


执行int z = 0，即将0存入[ebp-4]的位置，根据地址进入栈中也可看到，刚才初始化的44h的栈中，靠近ebp的一端存为了0。

之后执行z = x + y，将[ebp+8]和[ebp+0ch]的值（如图为1、3）都加到了eax中并给了刚才存的z，此时ebp作为add函数栈帧基址，ebp+为存入的参数，ebp-则为刚刚初始化的局部变量，因此这一步是将函数的参数也就是1和3都加给了z。

此时函数执行到了return z；将[ebp-4]也就是返回值z赋值给eax，即用eax来存储返回值 。

之后对应上面的三个push，执行了三个pop，将edi、esi、ebx以及esp都改回调用函数之前的值，最后到了最终的ret指令，此时根据esp地址查看栈内容，发现esp所在栈帧存储得正是最初的地址，F11之后发现这正是EIP的值，而ESP的值也回到了调用函数之前的值，至此add函数调用完成，而栈帧和寄存器也都回到了调用之前的状态，



**心得体会：**

通过这次实验，我学到了很多东西，通过在VC6上进行反汇编实验，对汇编语言有了更深入的认知和了解，熟悉了一些指令如call和ret的用法，同时也熟悉了栈的运作原理，知晓了计算机在运行程序时栈帧的变化，同时也意识到，地址与指令在汇编语言中的重要性。

而在知识之外同样也有其他得收获，在最初配置实验环境的时候，我遇到了各种各样的困难，例如VMware的兼容性问题、从无到有的安装和激活XP、VC6的使用等等，这些问题有的很常见、有的也很奇怪冷门，而我们作为一名工科生，以后将会越来越频繁的遇到这种问题，尤其是在配置一些环境的时候。能够学会如何去想办法解决他们，在我看来这是一个非常重要的品质。