《漏洞利用及渗透测试基础》实验报告

姓名：田晋宇　　学号：2212039　　班级：物联网班

实验名称：

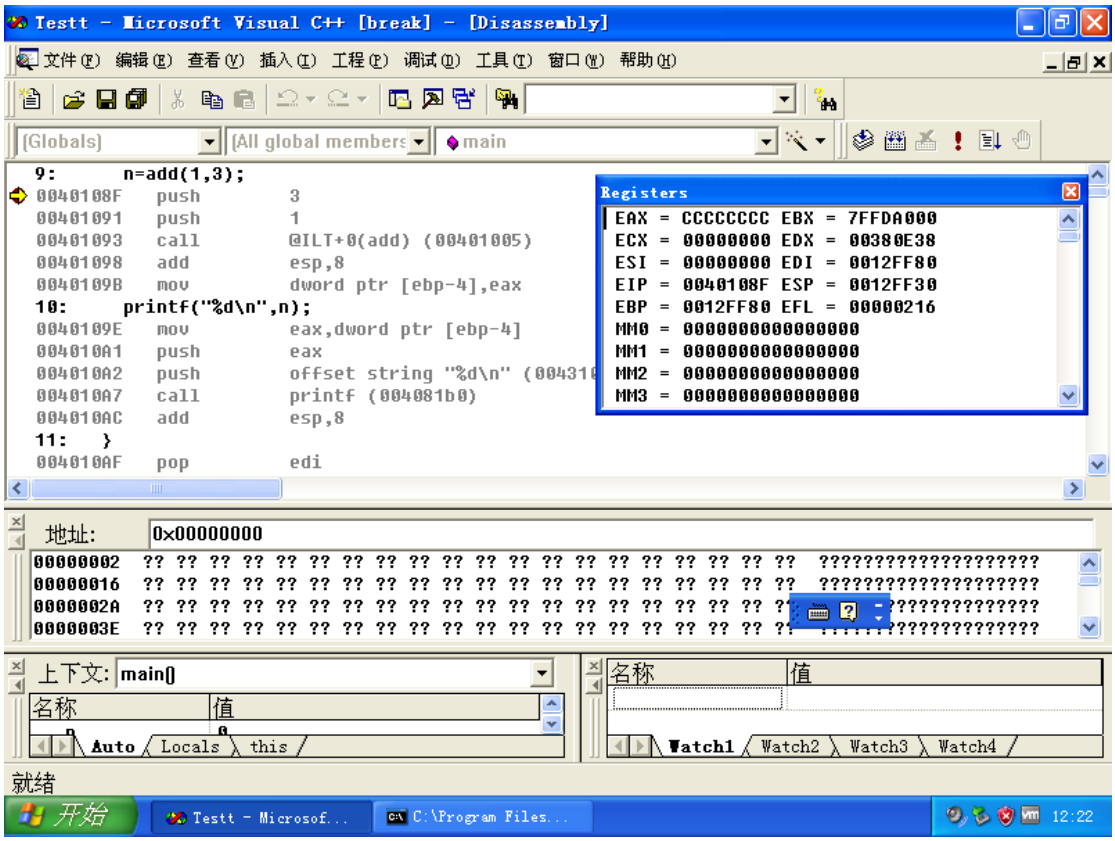
IDE 反汇编实验

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



2. 观察 add 函数调用前后语句

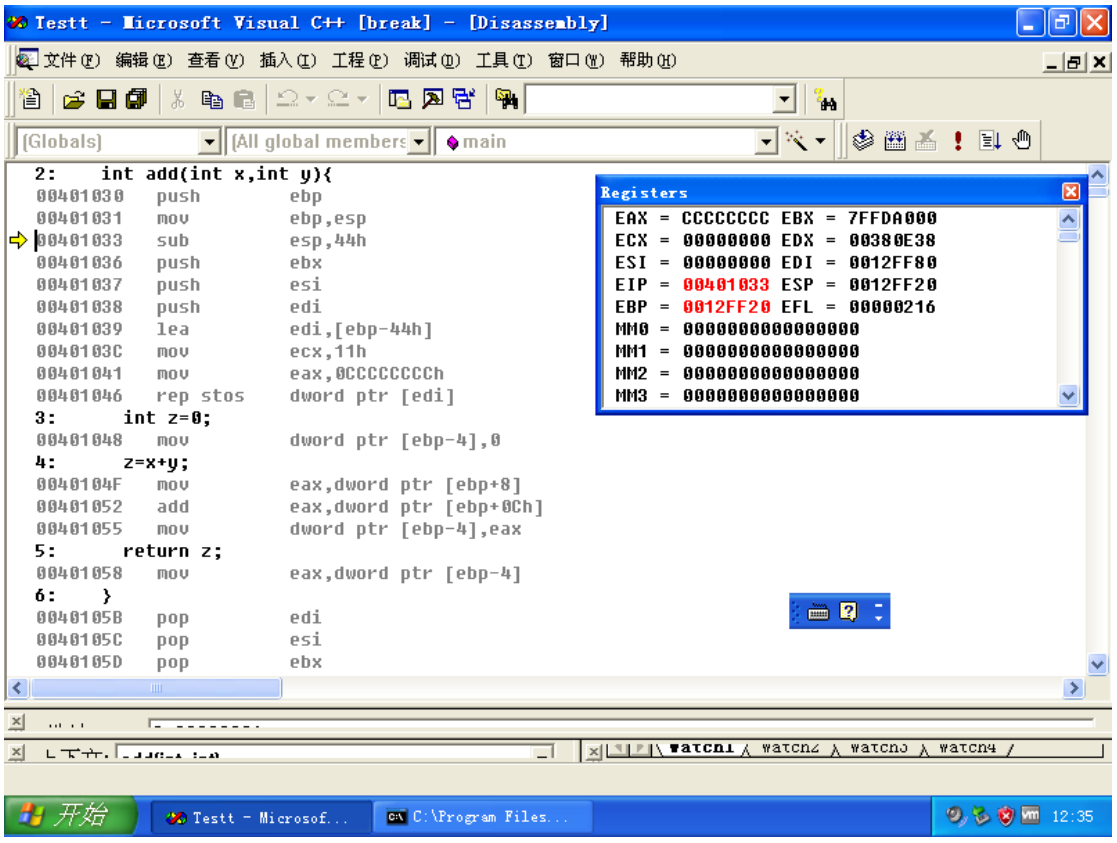
add 函数调用前，参数从右至左入栈，call 指令进入函数内部，add 指令使 esp 恢复到初始状态，最后利用 eax 将值赋给 ebp-4 即 n。

```
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
```

```
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
5:      return z;
00401058   mov         eax,dword ptr [ebp-4]
6:    }
```

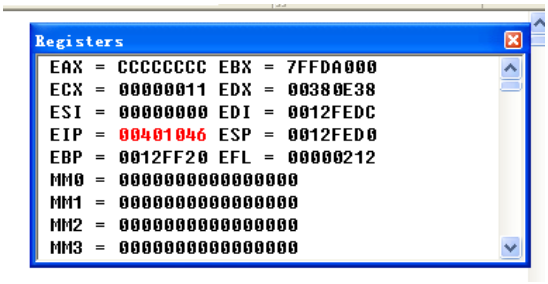
3. add 函数内部栈帧切换等关键汇编代码

在 add 函数内部，首先将 ebp 寄存器压入栈底，接着将 esp 的值赋给 ebp，可以看到 esp 和 ebp 的值已经相等：

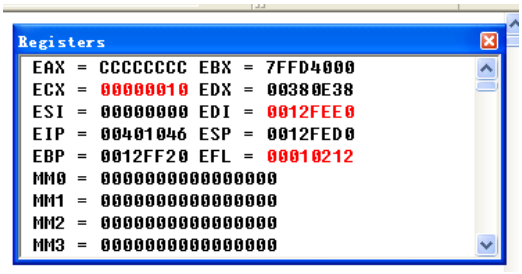


接着用 sub 指令，使 esp 寄存器上移，开辟一个 44 大小的空间，接下来的的三个 push 指令将函数调用过程中可能用到的变量压入栈中，接下来的几行语句对抬高的栈空间进行初始化，ebp-44h 即开辟占空间的起始地址，ecx 计数器意味着要进入循环，通过下方 rep 指令进入循环，每循环一次，ecx 计数器会减一，同时将一个 0CCCCCCCCh 压入栈中。

初始状态：



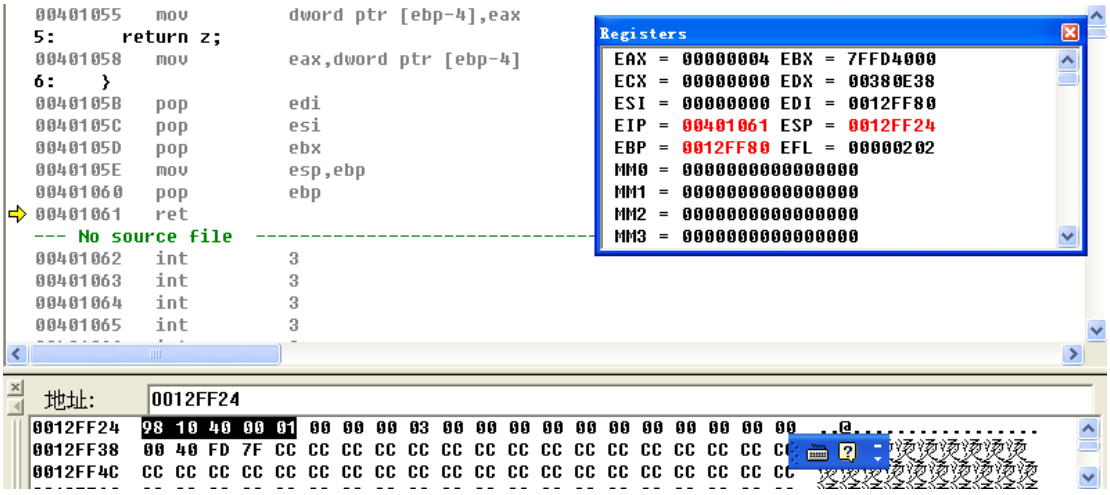
一次循环之后：



之后的 ebp+8 和 ebp+0Ch 是之前入栈的参数，通过 mov 和 add 指令完成加法操作。最终将 z 的值赋给承载返回值地址的 eax 寄存器。

```
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
```

最后完成的是函数调用结束的操作，三条 pop 指令对应之前的 oush 指令，ret 为函数真正调用阶数的指令，此时 EIP 寄存器已被赋值为 ESP 栈顶指针，即主函数返回地址。



ret 指令之后 EIP 寄存器被赋值为主函数返回地址：

心得体会：

通过实验，掌握了 RET 指令的用法；

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