《软件安全》实验报告

姓名：林盛森　　学号：2312631　　班级：计科三班

一、实验名称：

IDE 反汇编实验

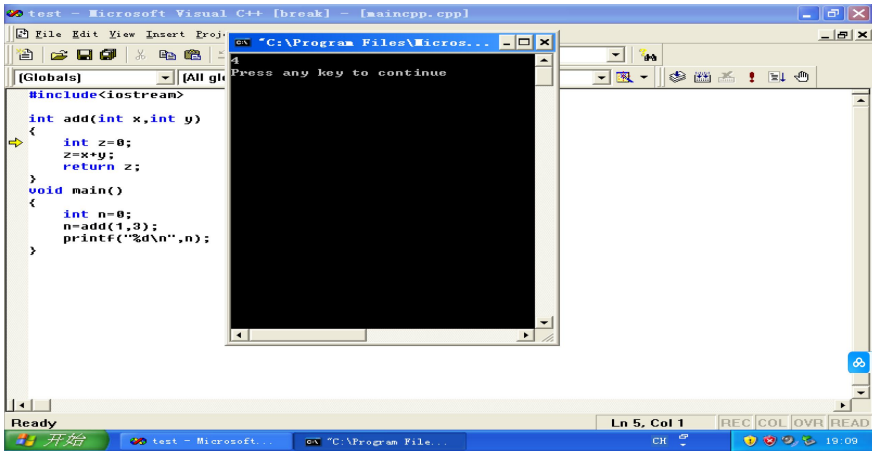
二、实验要求：

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。

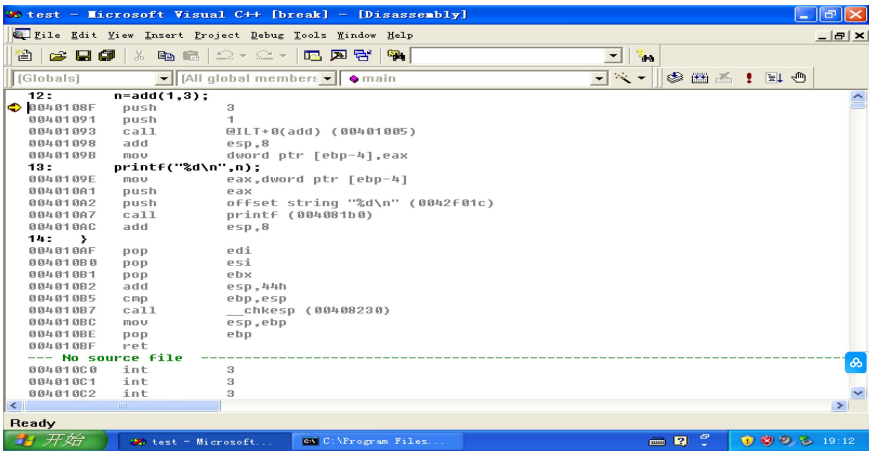
三、实验过程：

1. 进入 VC 反汇编

打开 vc6，新建一个项目，输入实验代码；



在 vc6 上运行测试代码，正常打印“4”说明没问题，继续实验；



在 n=add(1,3)处插入断电，f5 运行程序，在断点处右键点击 go to disassembly 进行反汇编，得到汇编代码。

2. 观察 add 函数调用前后语句

（1）调用前：

这段汇编代码是在 main 函数创建初期对其内部进行的相关操作，与后面在调用函数 add 内部处理操作类似，此处先不作说明；

这一句是说在栈底指针 ebp 位置处写入变量 n=0，由于栈的延伸方向是从高地址向低地址延伸，故在 ebp 的基础上偏移 4 个字节；

然后在函数调用时，按照参数从右往左入栈的顺序，以此通过 push 操作将参数 3 和 1 压入栈中，随后开始进行函数的调用；

这句就是调用 add 函数，可以看到跳转到 00401005 这个位置，然后我们按 f11 继续执

行；

```
00401005    jmp         add (00401030)
```

此处就可以看到 add 函数的入口点是在 00401030 这个位置，继续执行即可进入 add 函数内部；

（2）调用后：

```
004010AC    add         esp,8
```

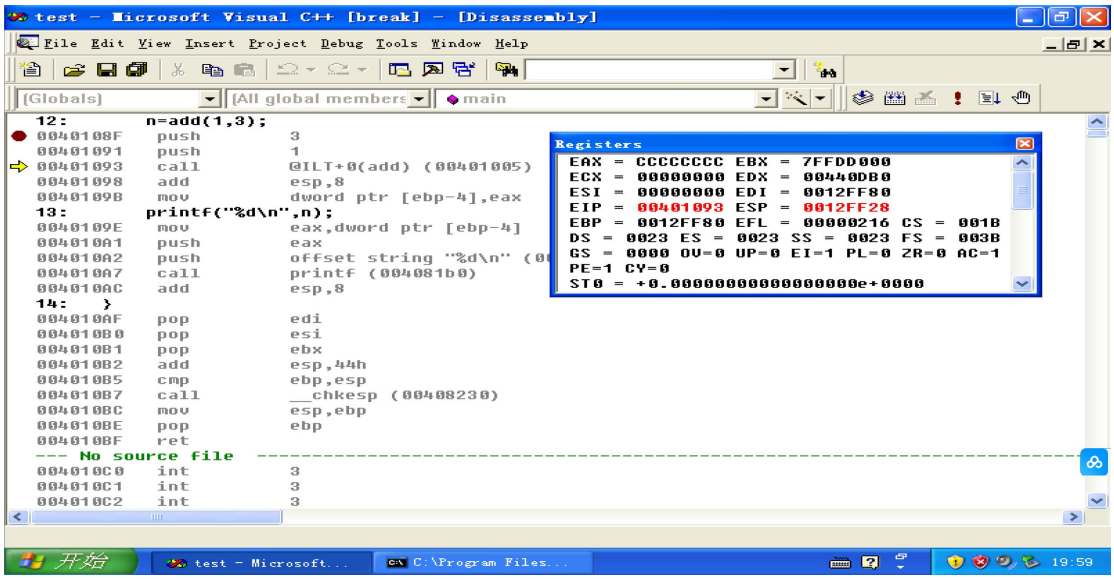
这句是给 esp 栈顶指针进行+8 的操作，即消去传入的 1 和 3 两个参数，恢复了调用前的栈指针情况；

```
00401098    mov         dword ptr [ebp-4],eax
```

add 函数返回值会传入 eax 寄存器中，这段是把 eax 中的值传给变量 n；

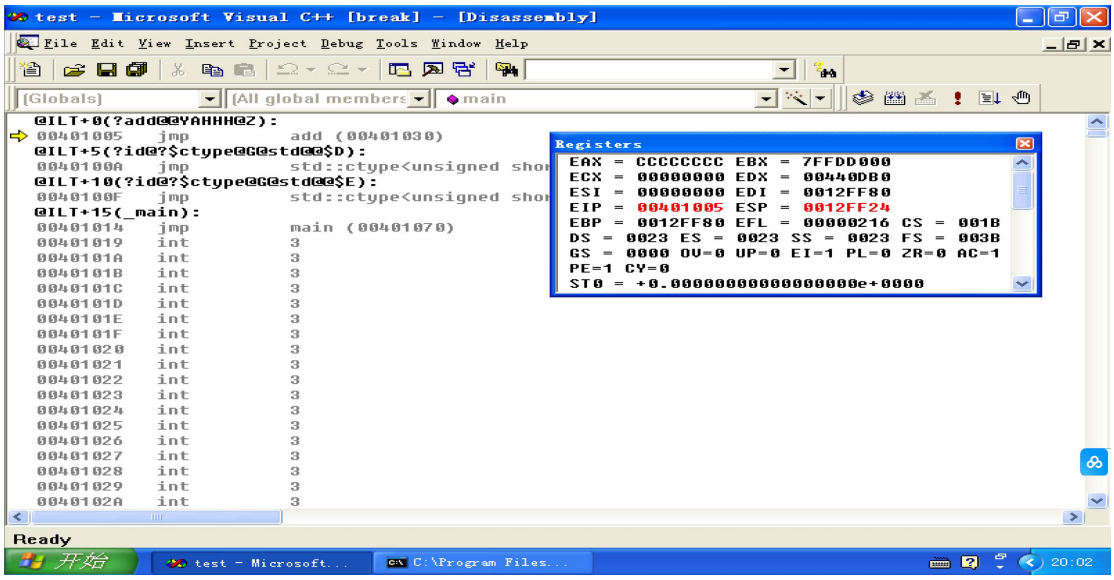
3. add 函数内部栈帧切换等关键汇编代码

（1）



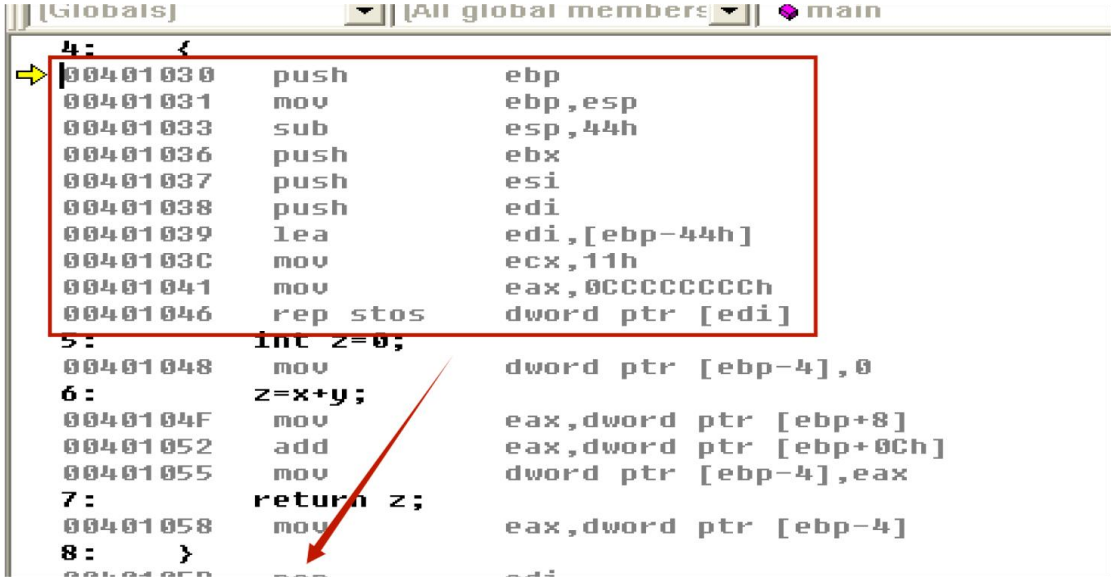
在通过 f11 逐步运行时，我们发现 esp 寄存器的值减少了 8，说明传入了两个参数；

（2）



再次点击 f11，esp 的值又少了 4，这是因为此时程序将函数的返回地址压入栈中，也就是调用处的下一条指令地址；并且 eip 的值也发生变化，这是由于 call 指令会使得 eip 的值指向程序正在运行的下一条指令地址；

（3）

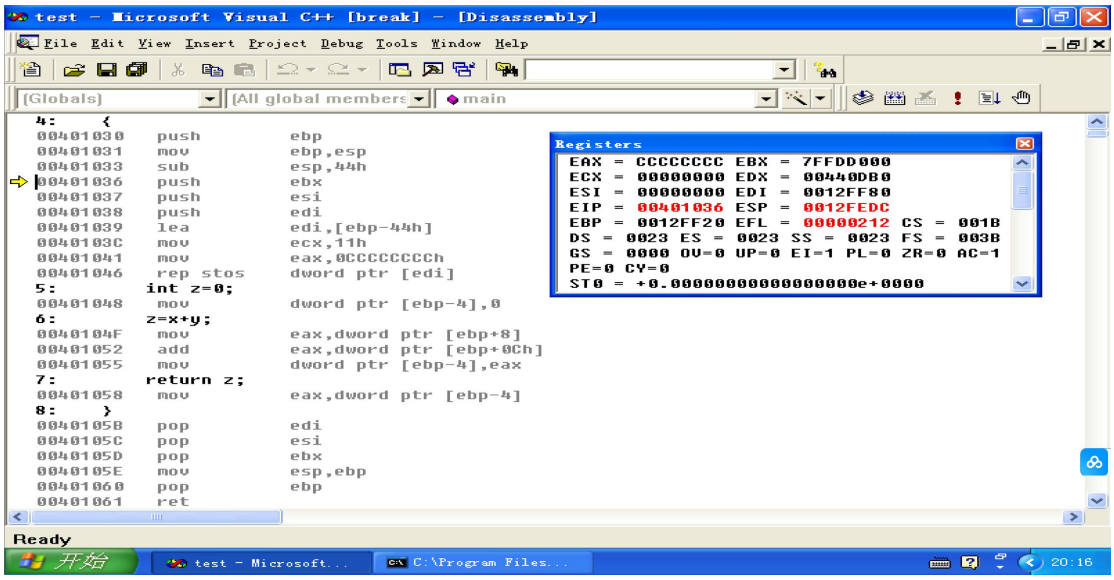


然后就进入到 add 函数内部，这段代码就是开辟空间然后还有一些寄存器的存储等，我们来具体分析一下：

push ebp 将主函数中的栈底指针地址压入栈中，

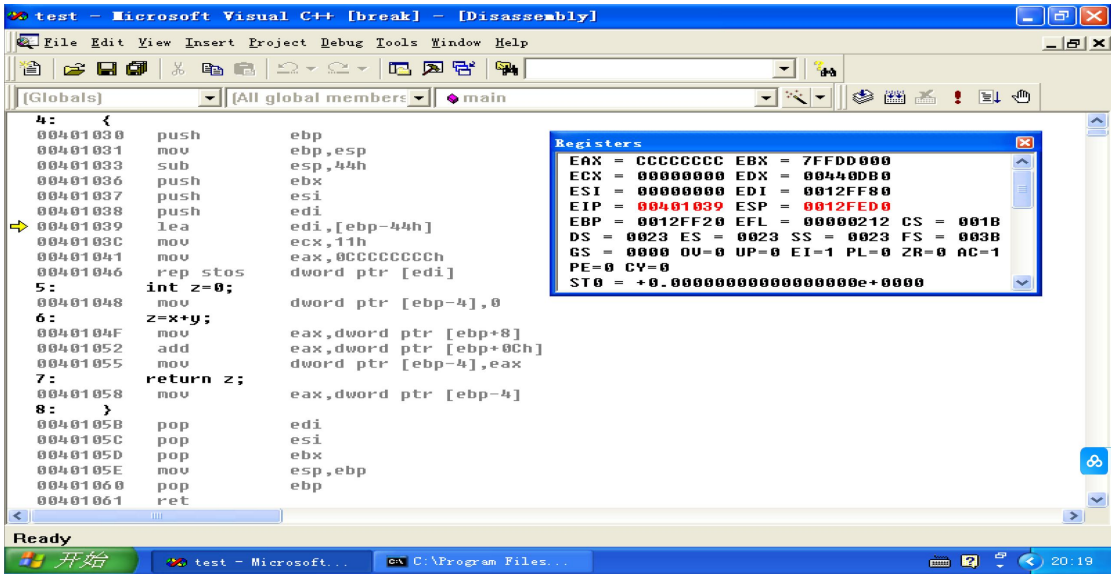
mov ebp，esp 为新栈的 esp 指针赋给 ebp 指针，作为栈的初始化，

sub esp，44h 为 add 函数分配了栈帧空间；



此时 esp 寄存器的值为 0012fedc；

（4）



push ebx，push esi，push edi 将主函数的一些寄存器的值压入栈中，从而可以在子函数里被使用且不丢失原来数据，在执行完这三句后，esp 寄存器的值变为 0012fed0；

（5）

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

接下来这四条指令，lea 指令将栈顶地址赋值给 edi，为循环计数器 ecx 赋值 11h，为 eax 寄存器赋值，rep 指令的目的是重复其上面的指令，STOS 指令的作用是将 eax 中的值拷贝到 ES:EDI 指向的地址，循环将栈区数据都初始化为 0CCCCCCCCh；

（6）

```
00401048    mov           dword ptr [ebp-4],0
```

将 z 初始化为 0；

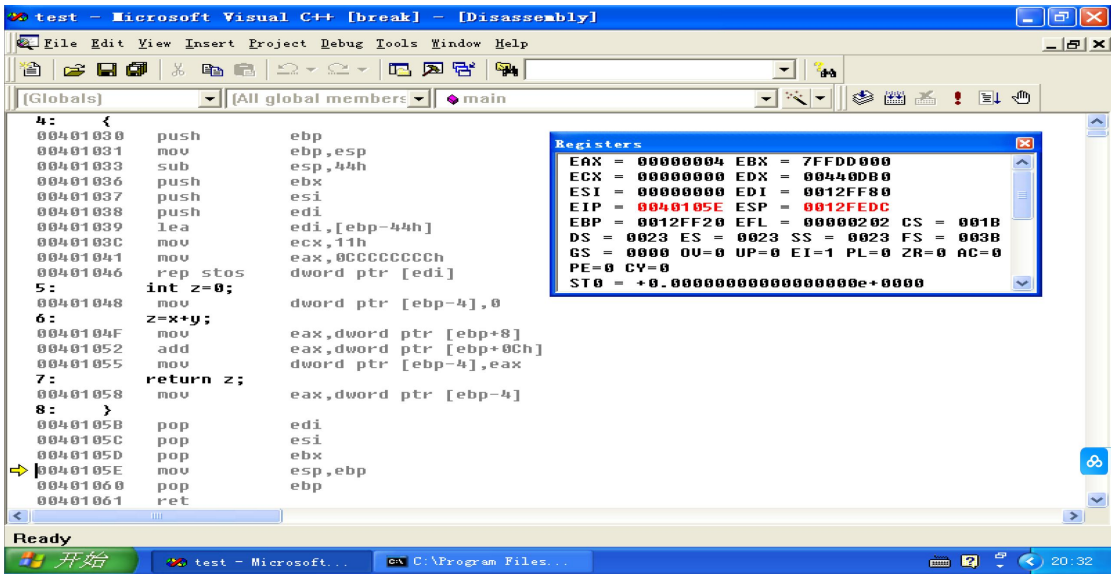
```
00401040F     mov             eax,dword ptr [ebp+8]
00401052      add             eax,dword ptr [ebp+0Ch]
00401055      mov             dword ptr [ebp-4],eax
```

此处为加法操作：先把第二个参数 1 放到 eax 寄存器中，然后去除第二个参数 3，与 1 相加，结果存入 eax 寄存器中，最后把 eax 寄存器中的值赋值给变量 z；

```
00401058      mov             eax,dword ptr [ebp-4]
```

将变量 z 的值赋值到 eax 寄存器中；

（7）



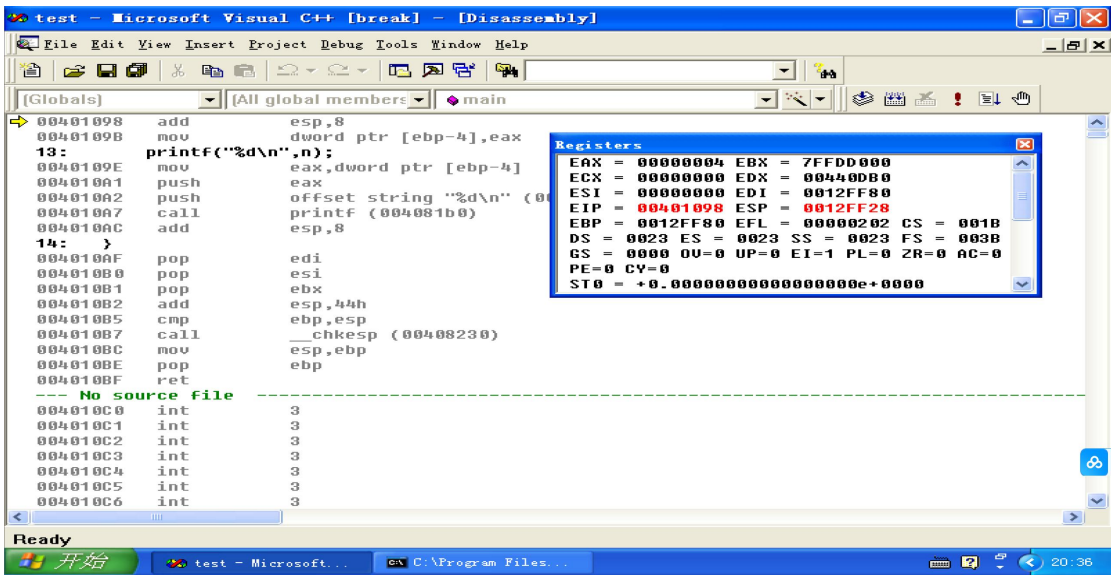
再将保存的三个寄存器弹出，可见 esp 的值恢复成了 0012fedc；

```
0040105E      mov             esp,ebp
```

然后清除开辟栈空间；

```
00401060      pop             ebp
```

把主函数的栈底指针赋值给 ebp 寄存器，



（8）

最后再通过 ret 指令，将函数的返回地址赋值给 eip 寄存器，可以看到 eip 寄存器所存指令地址恢复到了主函数中；

至此，call 指令执行完毕。

四、心得体会：

（1）通过实验，掌握了 RET 指令的用法；

（2）RET 指令实际就是执行了 Pop EIP；

（3）了解了函数调用的过程，对函数调用时函数内部栈帧切换的逻辑有了更为清晰的了解；

（4）学会使用虚拟机；

（5）此外，通过本实验，掌握了多个汇编语言的用法。