《软件安全》实验报告

姓名：王众 学号：2313211 班级：计算机科学卓越班

实验名称：

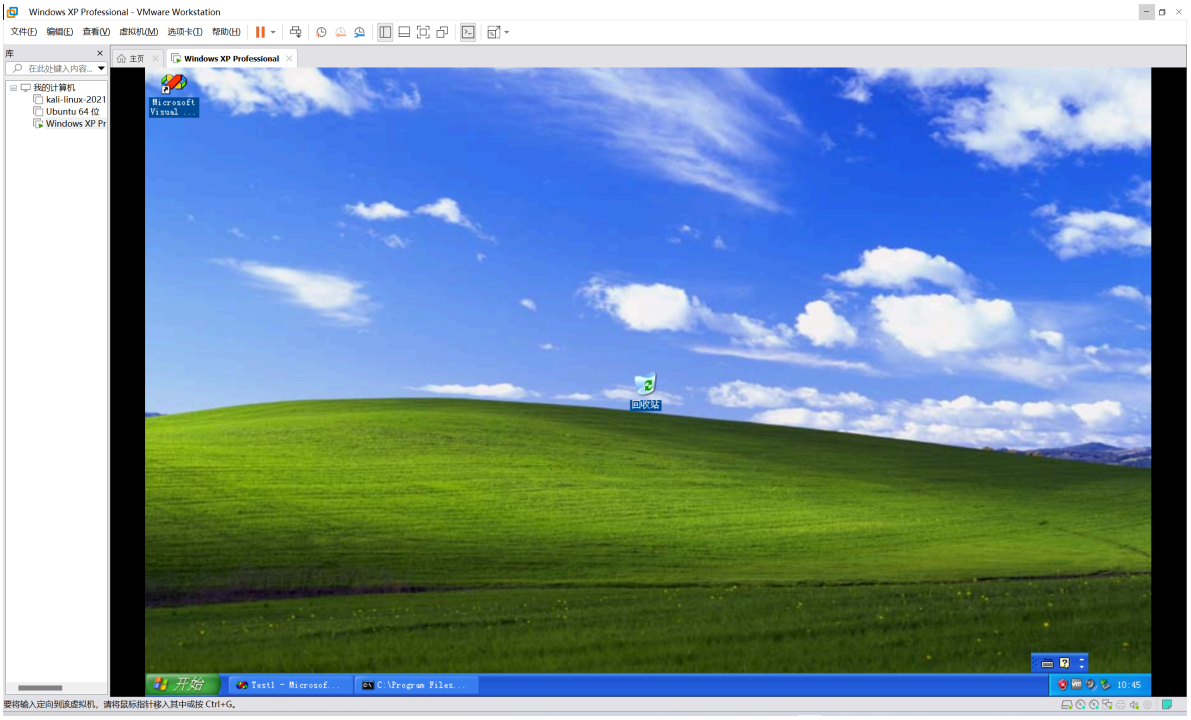
IDE反汇编实验

实验要求：

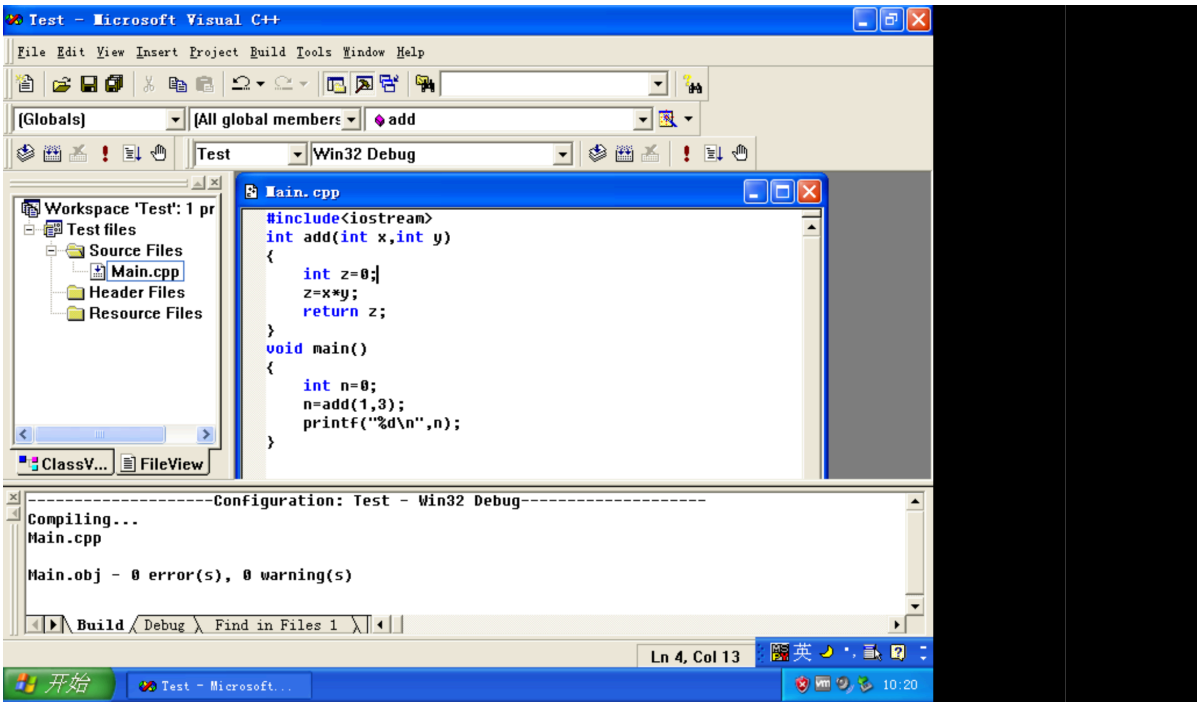
根据第二章示例2-1，在XP环境下进行VC6反汇编调试，熟悉函数调用、栈帧切换、CALL和RET指令等汇编语言实现，将call语句执行过程中的EIP变化、ESP、EBP变化等状态进行记录，解释变化的主要原因。

实验过程：

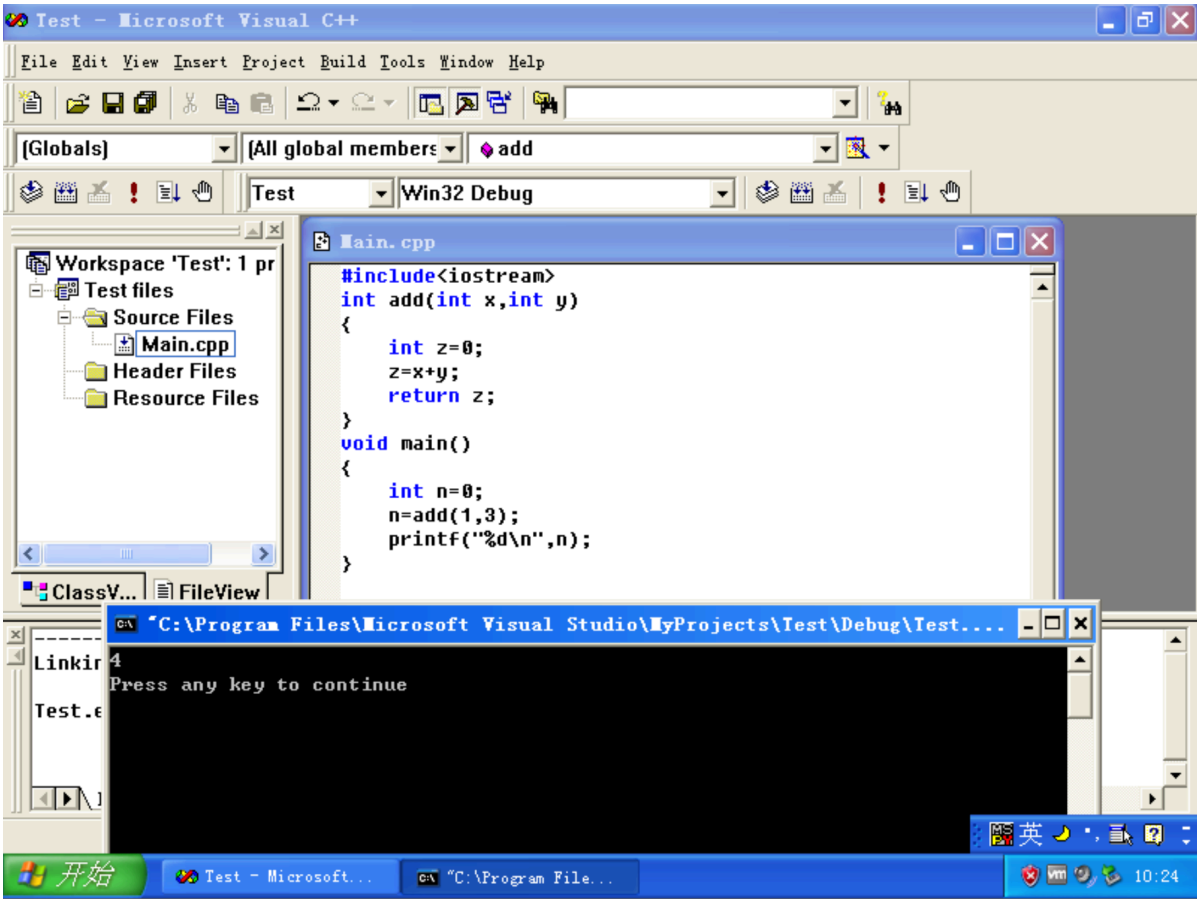
1、进入VC反汇编



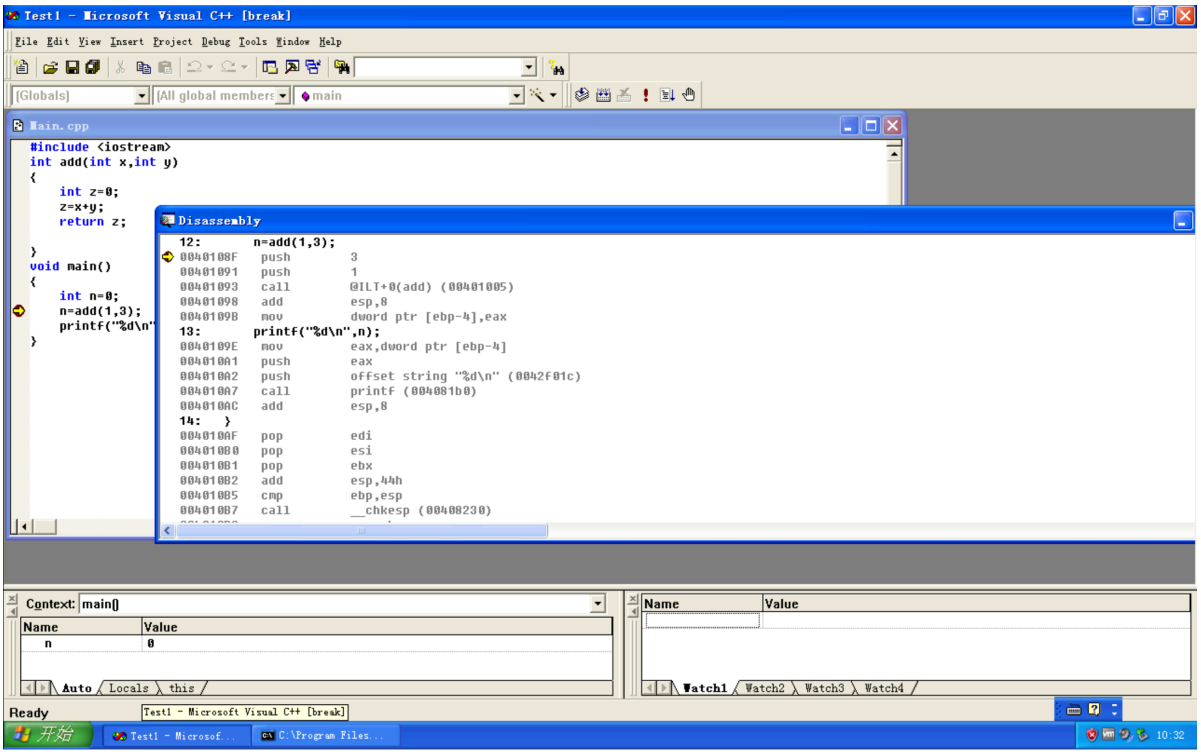
打开VM-ware workstation，进入windows XP professional软件，进入主界面。



新建project和相应的cpp文件



调试代码并查看结果输出是否正确



在代码`n=add(1,3)`处按F9打上断点，并按下F5按钮进行调试，右键选择Go To Disassembly进行反汇编，可以得到汇编代码

2.观察add函数调用前后语句

总体代码：

```
00401088   mov         dword ptr [ebp-4],0
12:       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
```

逐行分析：

调用前：

调用前的是一行mov指令

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

move指令为接下来的变量分配了一定的内存空间。`ebp-4`指将ebp寄存器抬高了4字节（栈的上面是低地址）

```
push        3
push        1
```

两条push指令，分别将3和1入栈

```
call        @ILT+0(add) (00401005)
```

调用了`call`指令，在调用`call`时程序会将下一条指令的地址（返回地址）压入栈中，然后跳转到指定的目标地址执行。

`@ILT`通常表示“Import Lookup Table”（导入查找表），这是Windows程序中用于动态链接库（DLL）函数调用的机制。

`+0`表示偏移量为0

`add`是函数名

`(00401005)`是即将跳转的地址

```
@ILT+0(?add@@YAHHH@Z):
00401005   jmp         add (00401030)
```

程序跳转到了这里,但是这里有jmp所以继续跳转

```
1:    #include <iostream>
2:    int add(int x,int y)
3:    {
00401030   push        ebp
```

程序跳转到了add函数的第一行

调用后:

```
add         esp,8
```

使用`add`语句将`esp`寄存器增加8，释放之前压入栈的两个参数。（一个参数占4个字节）

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

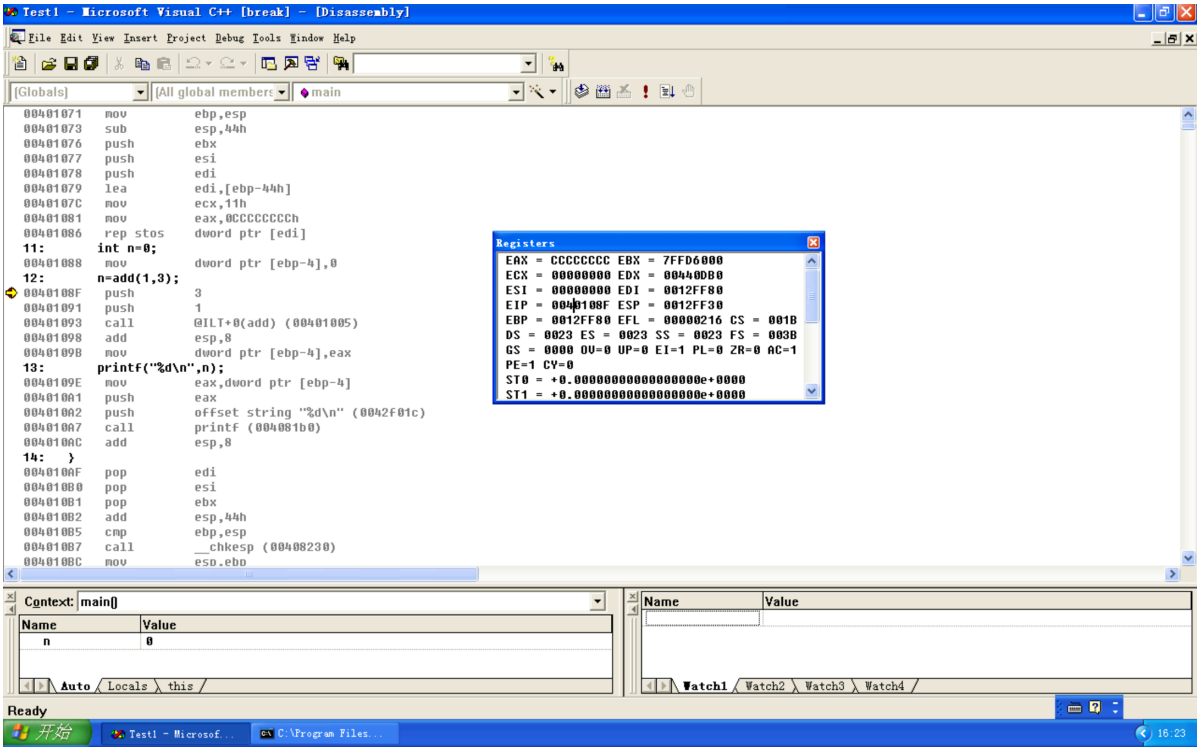
使用`eax`储存函数的返回值，即 将`add（1，3）`函数的返回值储存到`[ebp-4]`中。赋值给变量`n`

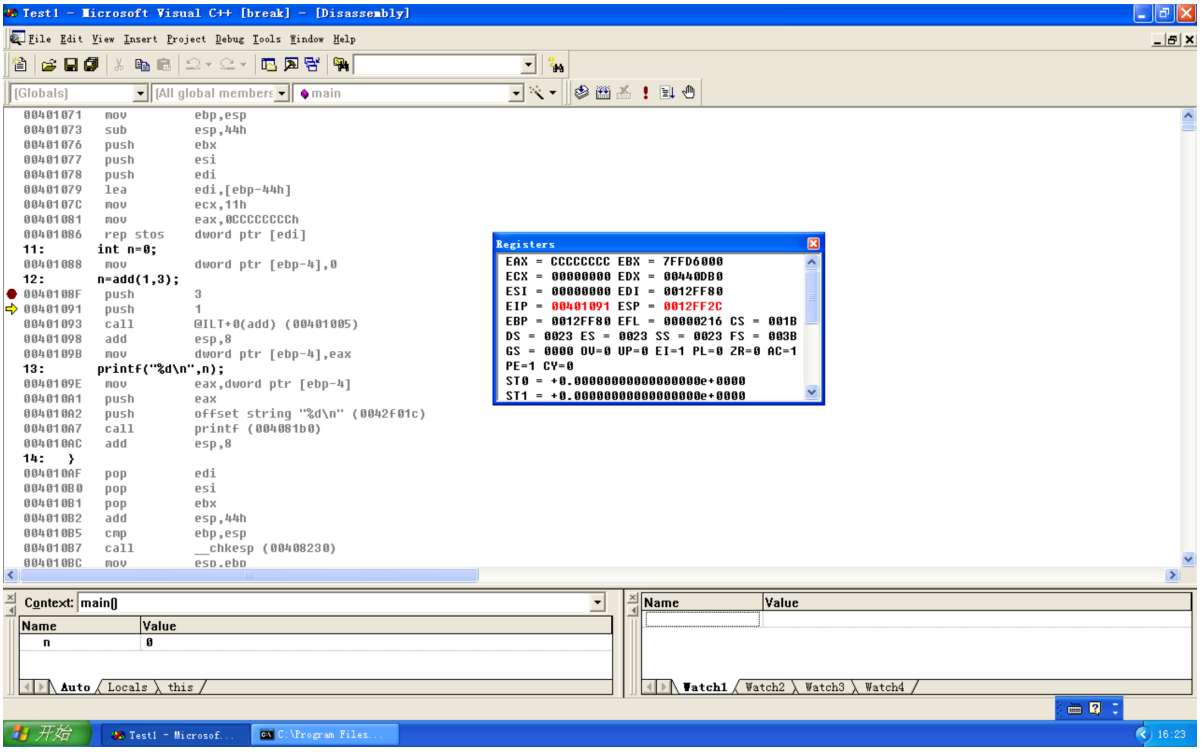
`dword`表示 32 位（4 字节）的数据，是数据大小的修饰符，用于明确操作的数据长度。

`ptr`用于明确操作的是内存地址中的数据，而不是直接操作寄存器或立即数

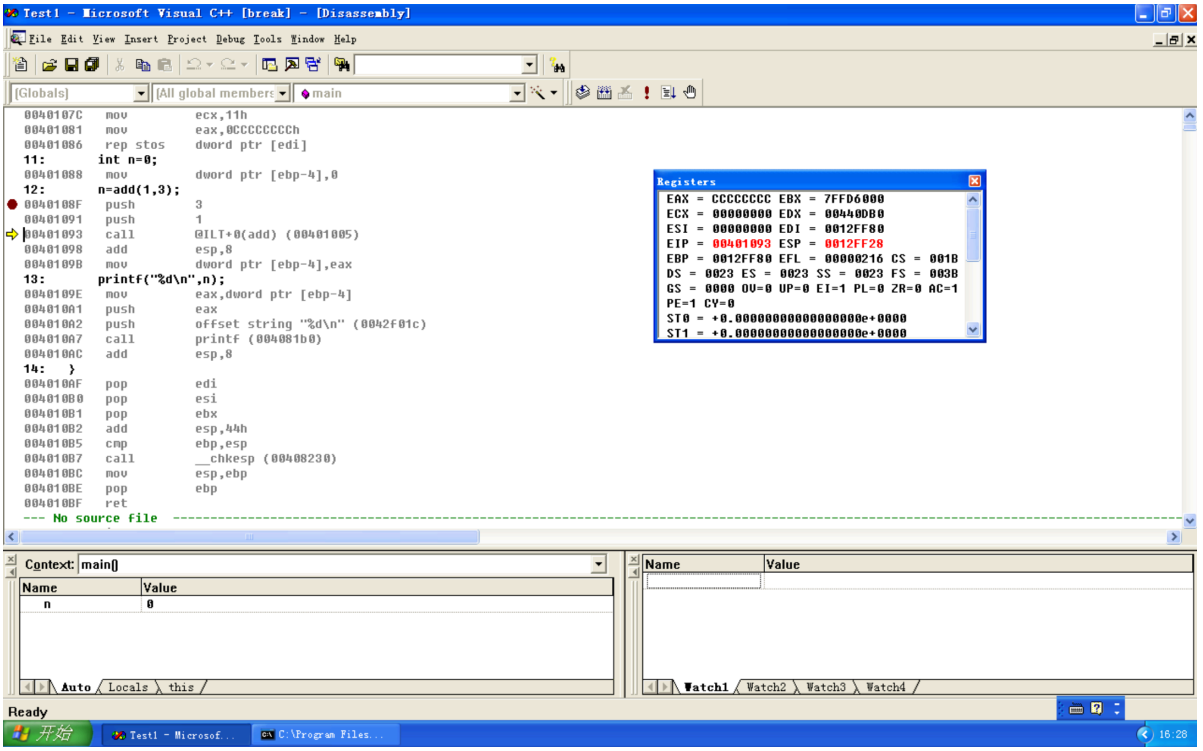
3.add函数内部栈帧切换等关键汇编代码

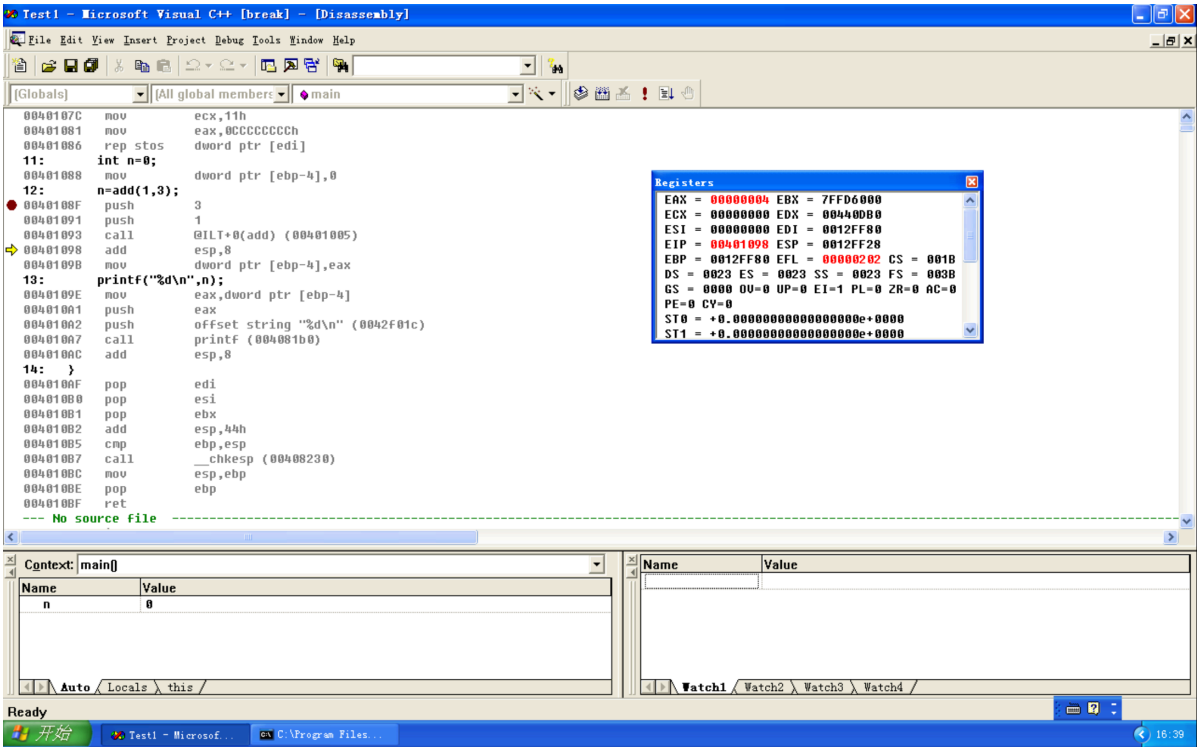
(1)首先从断点处按 F10 进行调试，观察到 esp 从 0012FF30 逐渐减去4变成 0012FF2C 再变成 0012FF28 因为入栈是从高地址向低地址增加，所以最终 esp 的值减8. EAX 的值也从 CCCCCCCC 变成结果 00000004 .



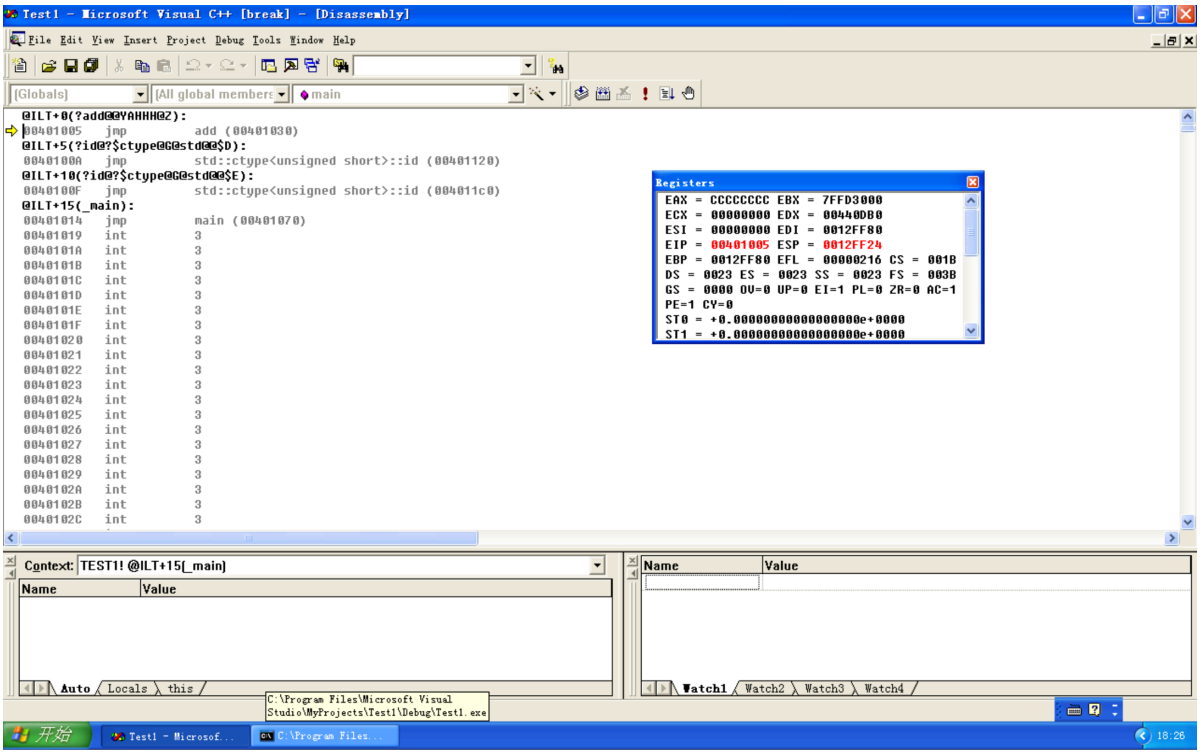


!

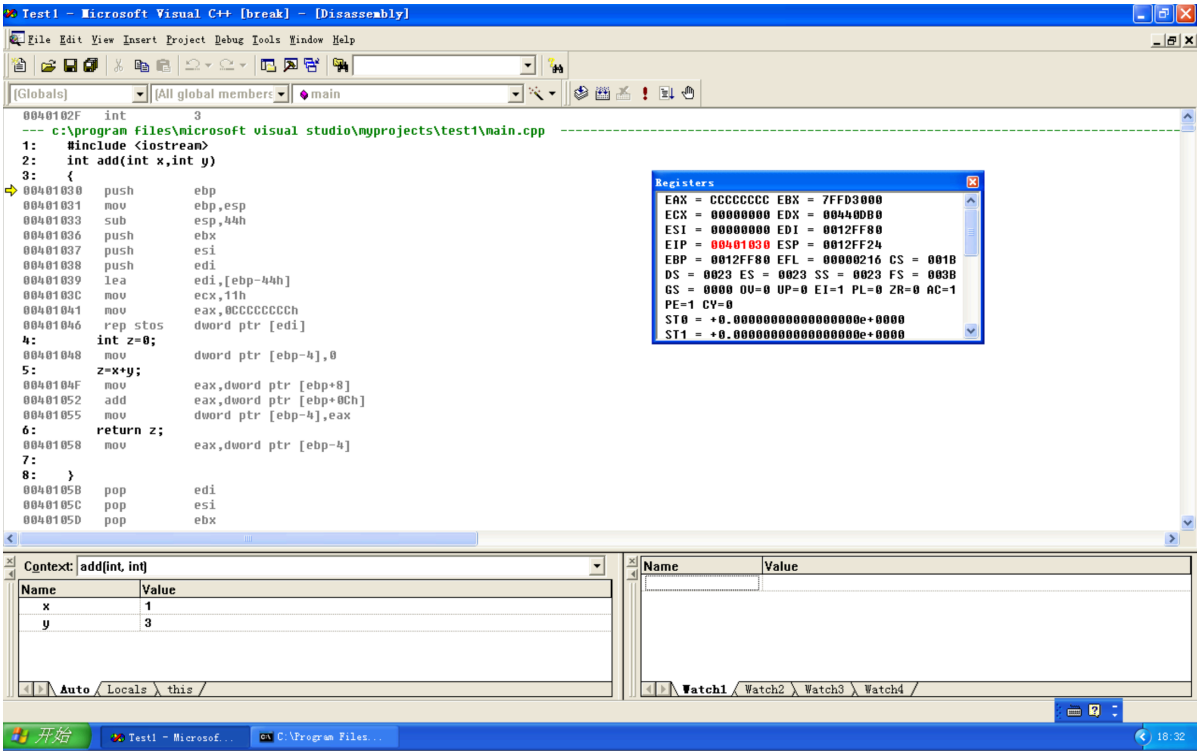




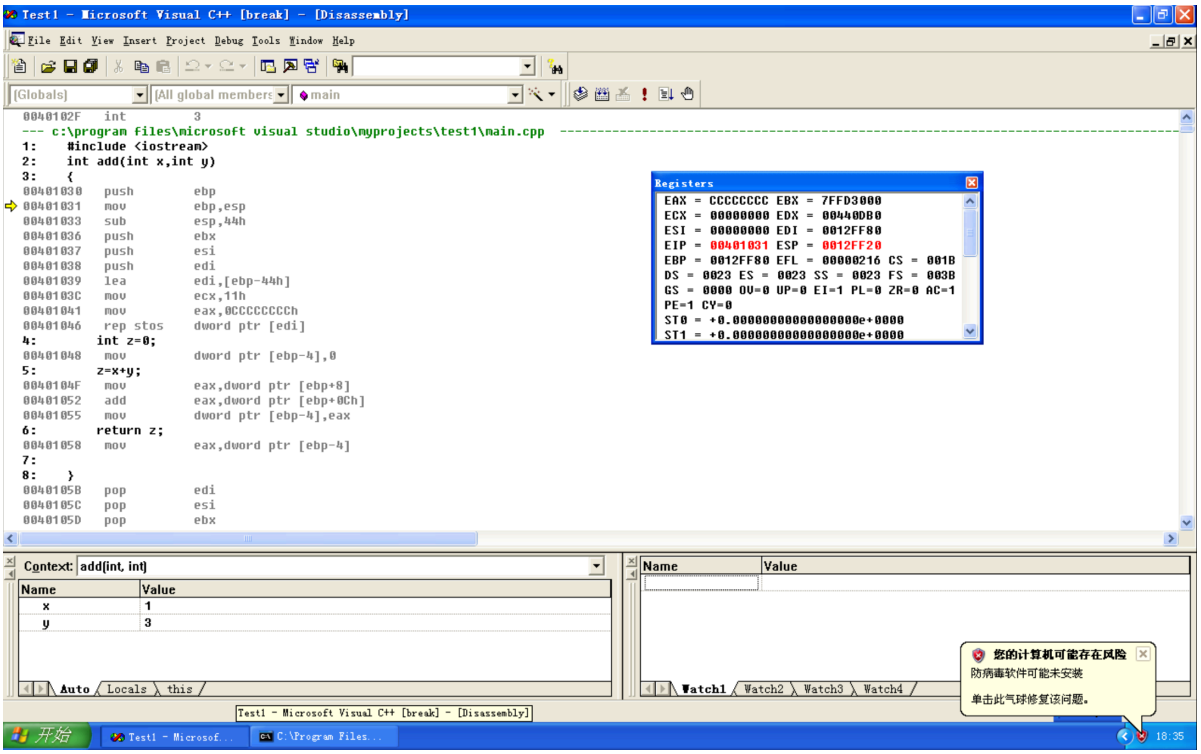
(2)在执行到 `call` 指令的时候，按 `F11`，进入到 `add` 函数。这时 `ESP` 的值又减少了4，变成了 `0012FF24`. 因为 `call` 指令分2步进行，第一步是将当前 `EIP` 的指令地址压入栈，第二步是修改 `EIP` 的值



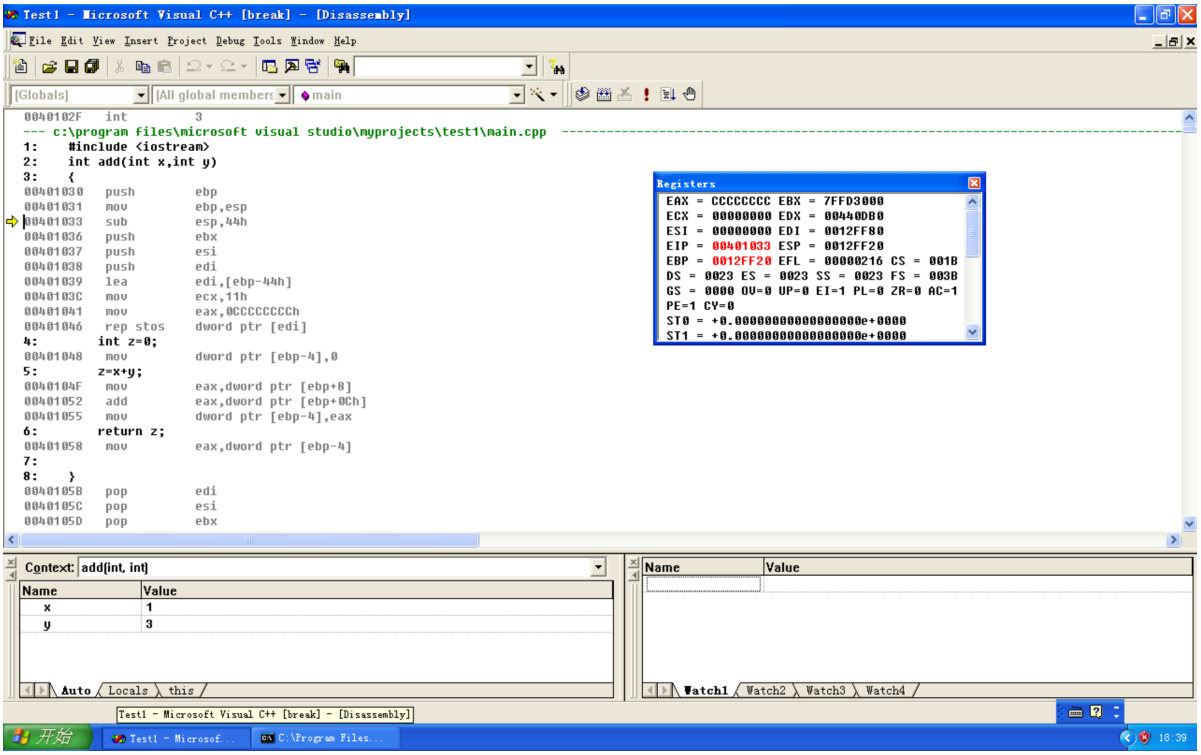
(3)再次按`F11`，进一步进行跳转，`EIP`的值变为`0041030`.



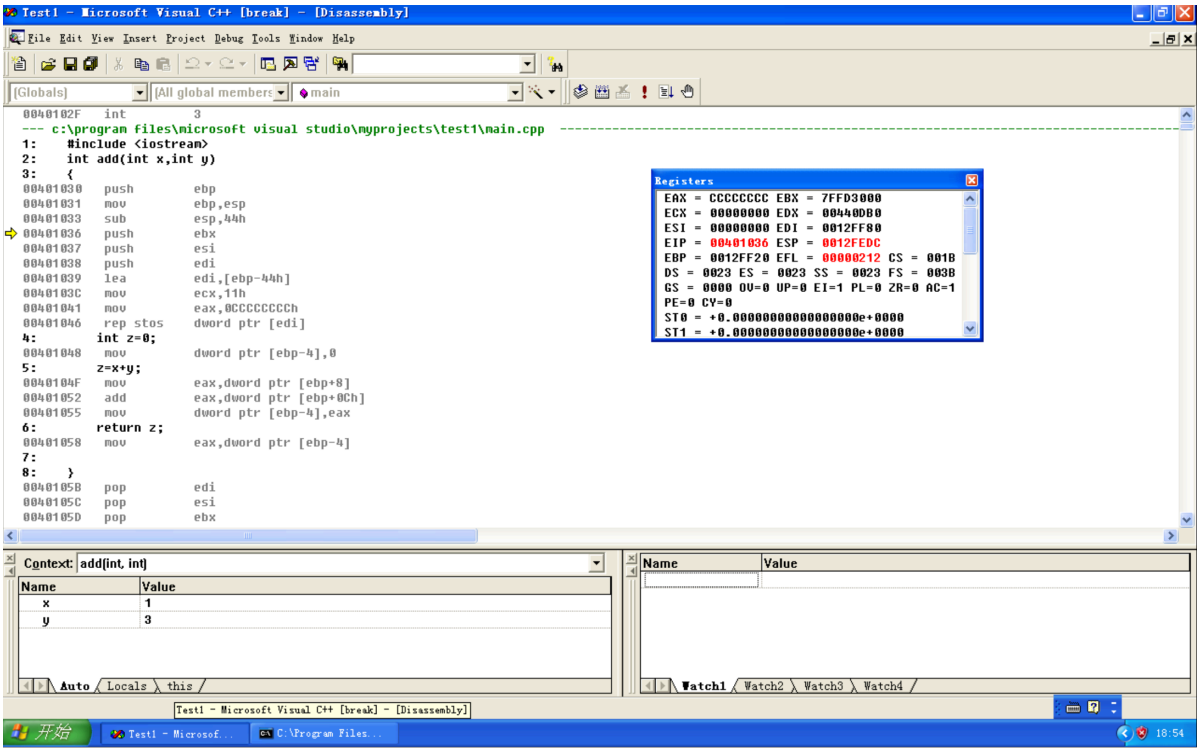
(4)进入函数后，再按`F11`.发现`ESP`又减少了4，因为进行了`ebp`的入栈操作。



(5)继续下一步 ebp 被复制为了 esp



(6) esp 被减去44h，开辟内存空间



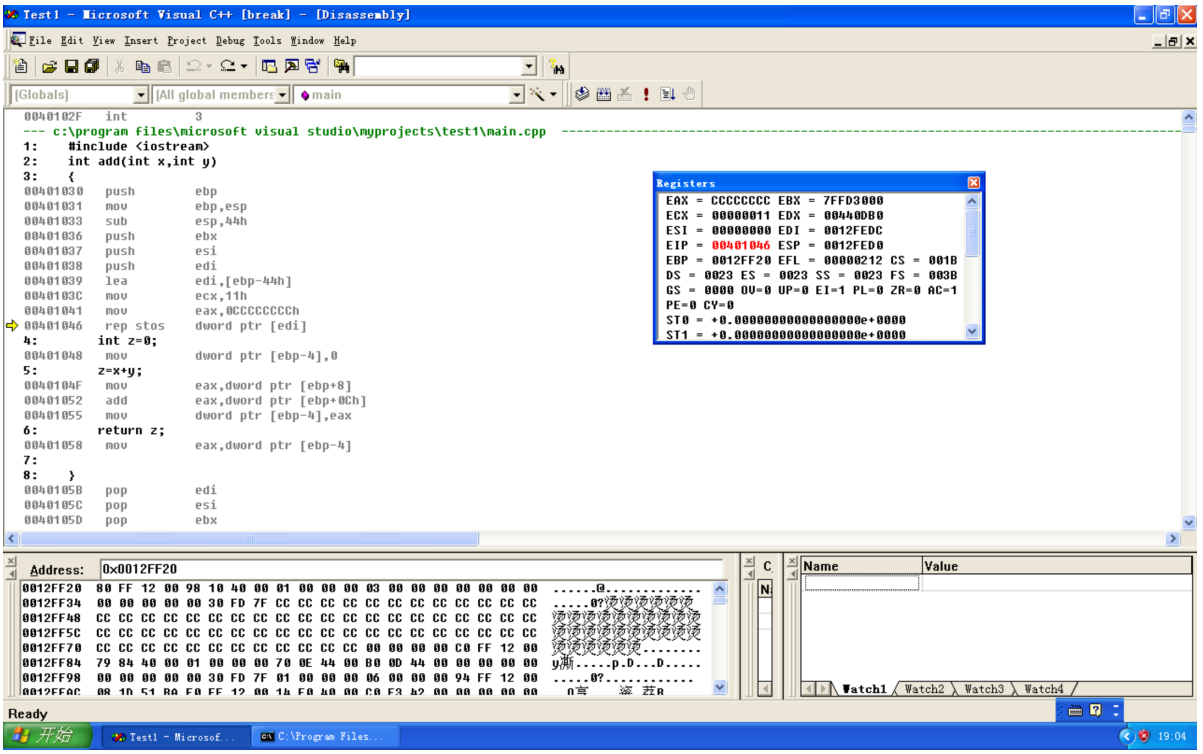
(7)3个push把主函数可能用到的寄存器的值压栈。

```
00401036   push        ebx
00401037   push        esi
00401038   push        edi
```

(8)进行初始化工作

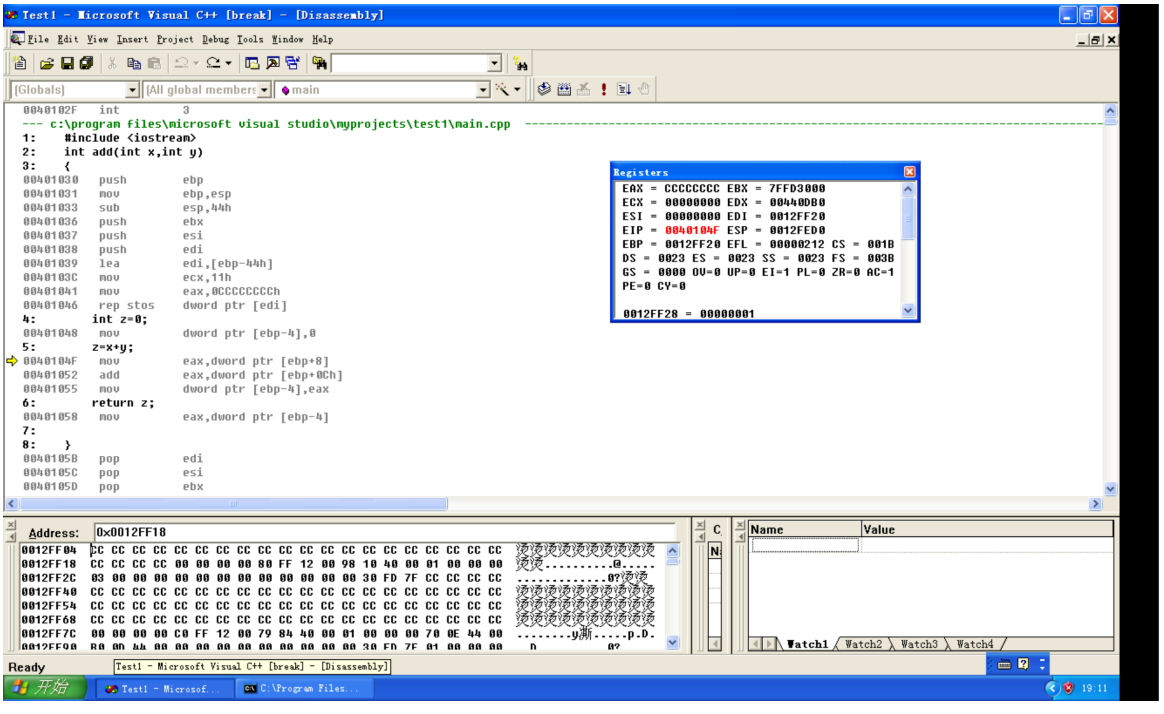
```
00401039   lea         edi,[ebp-44h]
0040103C   mov         ecx,11h
00401041   mov         eax,0CCCCCCCCh
```

(9)把局部变量最顶部的地址赋值给 `edi` ，使用ecx作为计数器并循环11h次。将44h的空间全部复制给 `0cccccch`

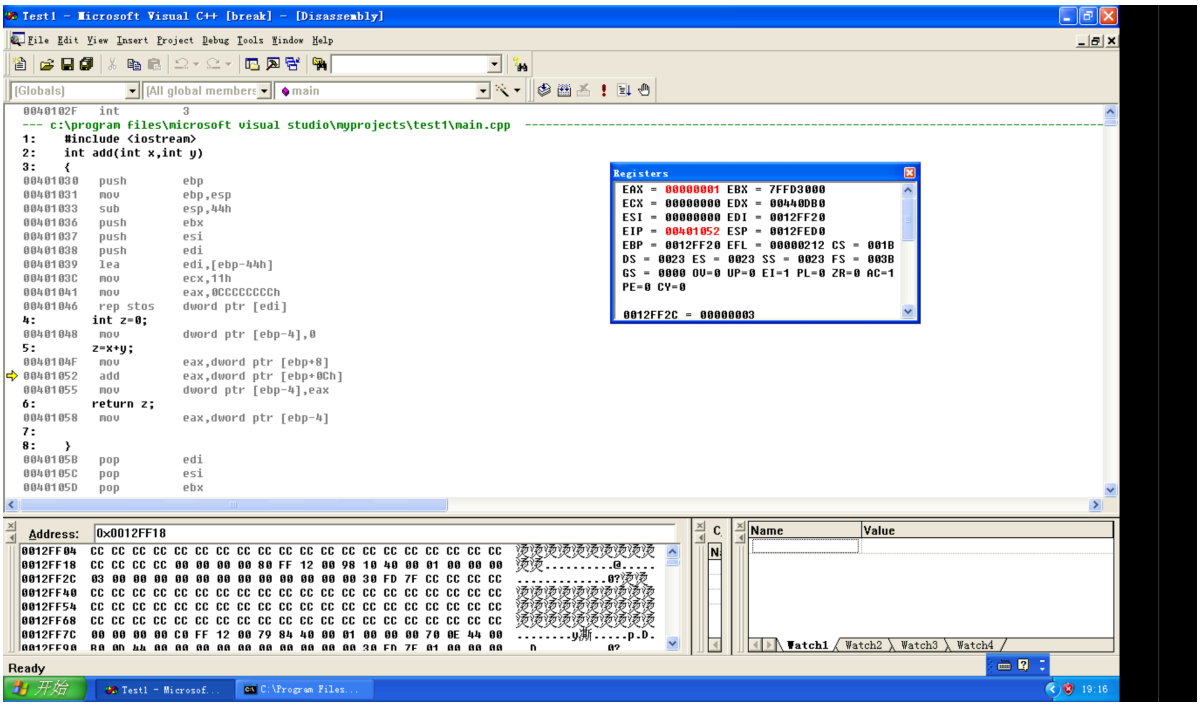


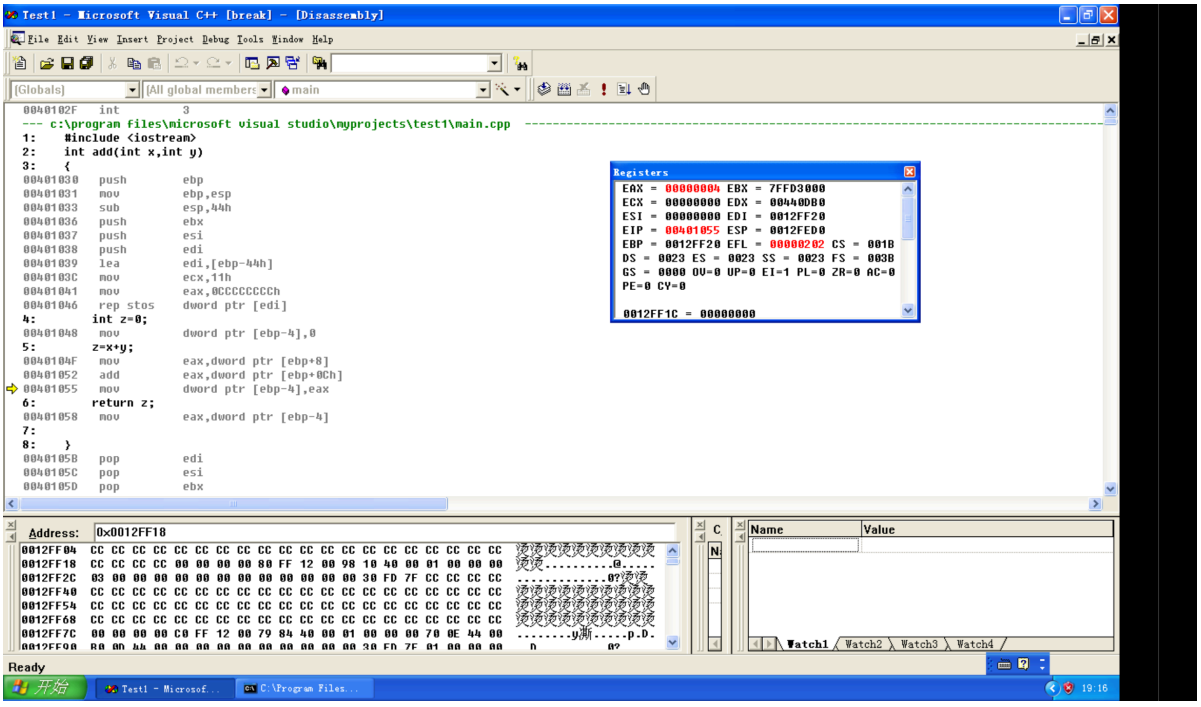
(10)把0赋值给 ebp-4

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



(11)进行加法操作， ebp+8 访问参数x， ebp+0ch 访问参数y，最后将结果保存在 EAX 中。再把返回值存在 EAX 中。

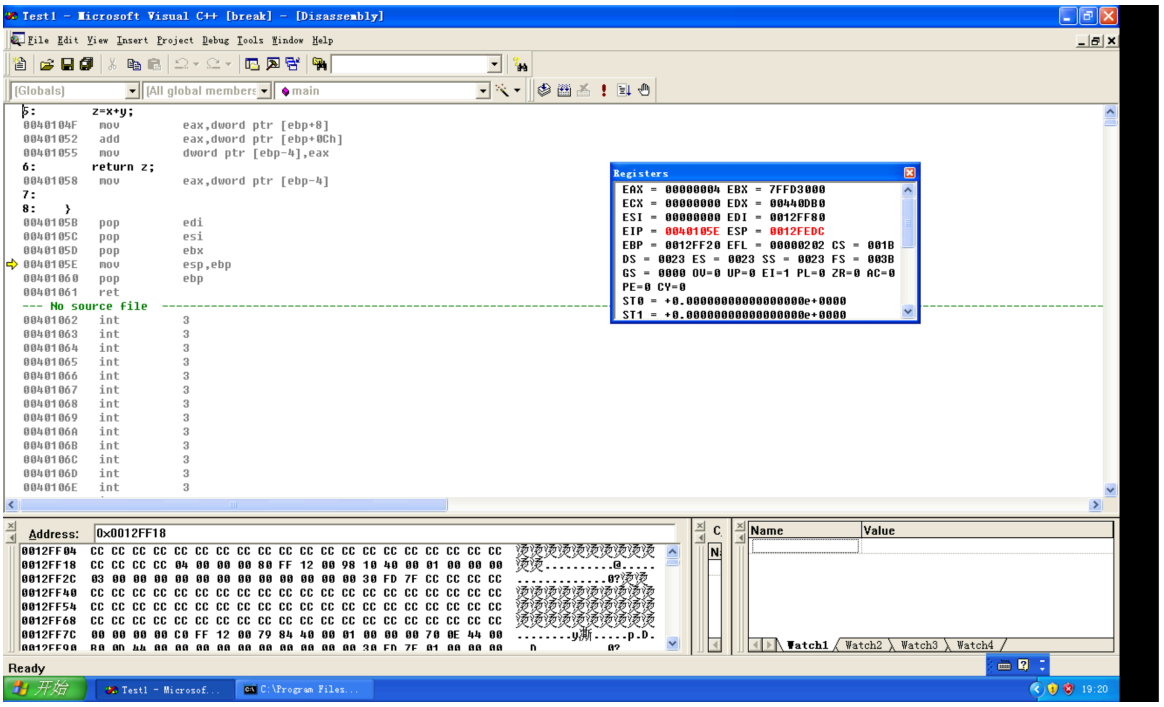




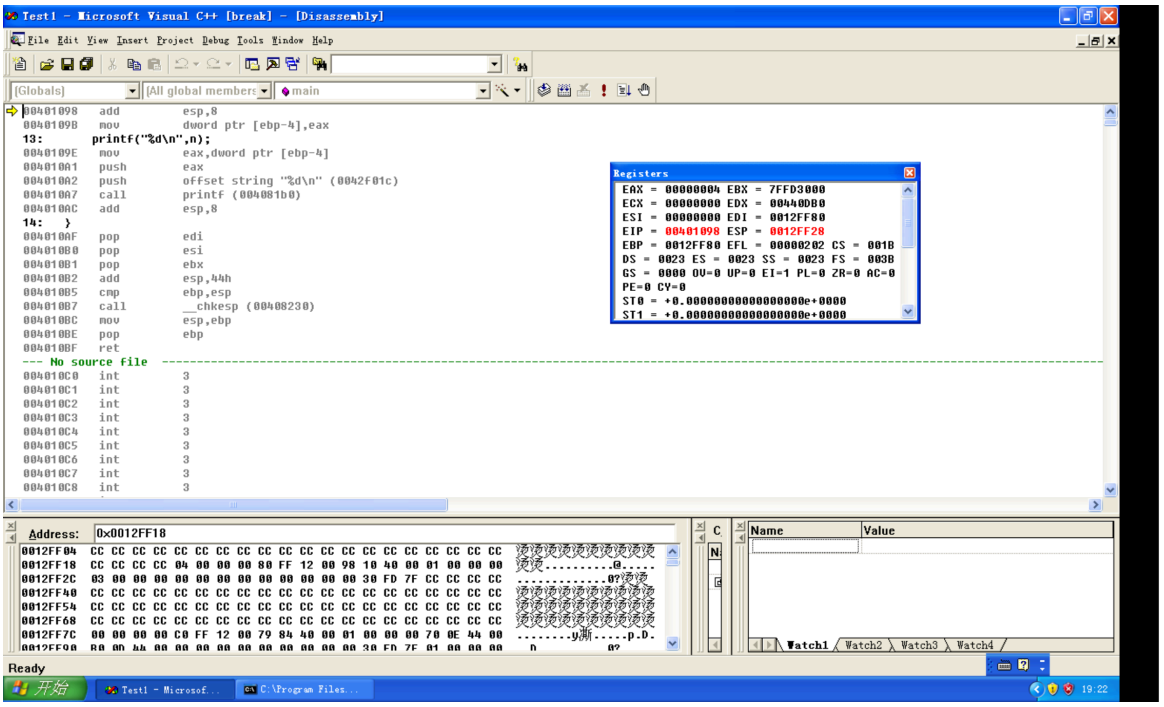
(12)上面的3个push会对应3个pop

```
0040105B    pop         edi
0040105C    pop         esi
0040105D    pop         ebx
0040105E    mov         esp,ebp
00401060    pop         ebp
```

和上面的句子相对应



(13)在`ret`语句处按`F11`，跳出函数。完成参数的出栈。




```
0012FF24 98 10 40 00 01 00 00 00 03 00 00
```

`EIP`被赋值为`00401098`整个`call`指令结束

心得体会：

在此次实验中，我学会了很多的新知识。

1.学会了如何配置VM-ware环境，并使用XP和VC6进行实验操作。编写代码，设置断点以及反汇编。

2.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。我们发现执行了`ret`指令之后，`eip`的地址变为了`00401098`

3.此外，通过本实验，我掌握了多个汇编语言的用法

4.通过反汇编，逐行对照高级语言代码和汇编指令，理解了每行高级语言代码是如何被翻译成机器指令的。

5.在学习过程中了解到了参数是如何通过栈或寄存器传递的，返回值是如何存储的（通常通过 `eax` 寄存器）。