《漏洞利用及渗透测试基础》实验报告

姓名：陈嘉乐 学号：2111090 班级：1121

**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

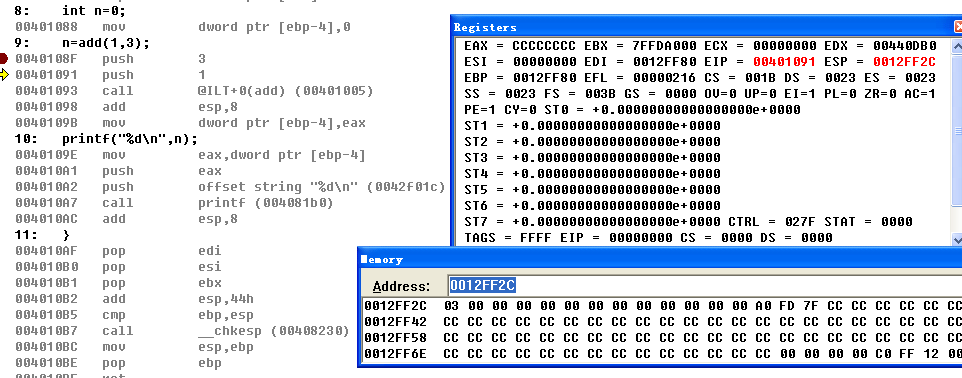
打开虚拟机的xp系统后，在vc6中创建win32的控制台文件，添加源文件后将代码复制到源文件中，debug无误后，在n=add(1,3)处设置断点，进行调试，并调出registers和memory消息栏，并在代码处右边点击go to disassembly将c++代码进入反汇编页面，从而观察汇编语句的调用。



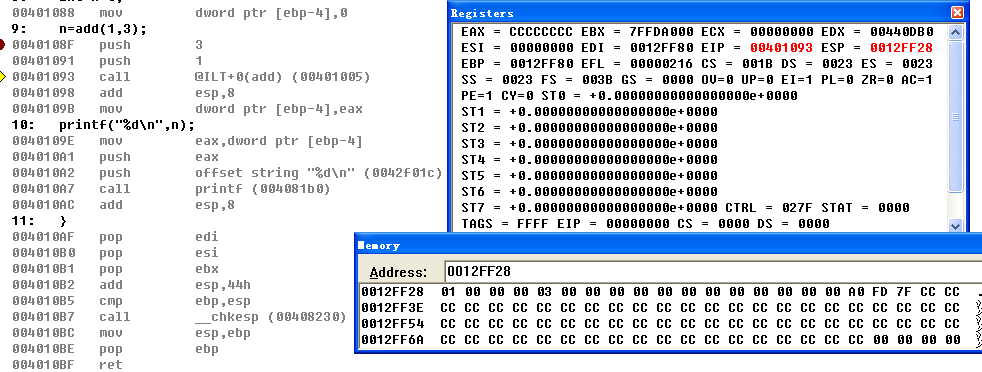
2. 观察add函数调用前后语句

在add函数调用前会将函数所用的参数按照从右向左的顺序依次进行入栈

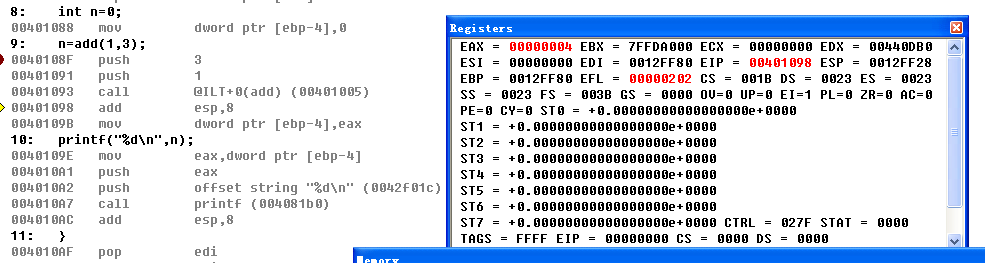
对于push 3操作执行F10后我们可以看到esp即栈顶指针向上减小了4，将此时的esp的地址执行查找可以看到是值3进行了入栈操作。



同理F10 push 1语句后，栈顶esp依旧向上减小4，查找地址得到的值为1，即1入栈

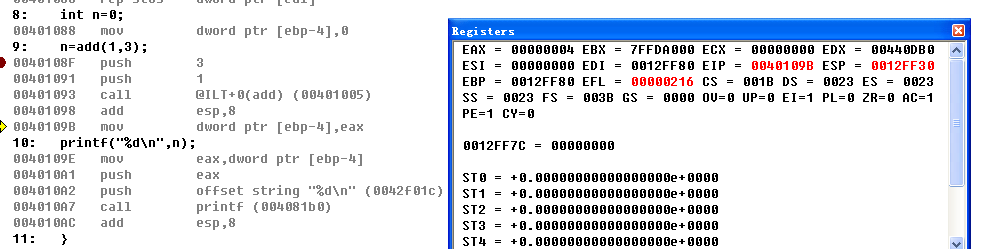


此时在函数add调用前的步骤参数入栈便执行结束，将调用函数直接执行后得到的参数：

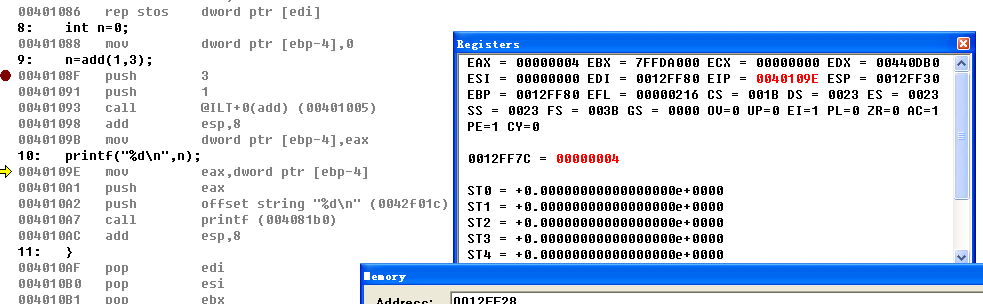


esp和ebp的值均为调用前的地址，但eax的值变成了00000004，即1+3

函数调用后还要将入栈的参数执行出栈，add esp，8即将eip的地址返回到了执行n=add(1,3)语句前的状态，即完全的还原了栈帧

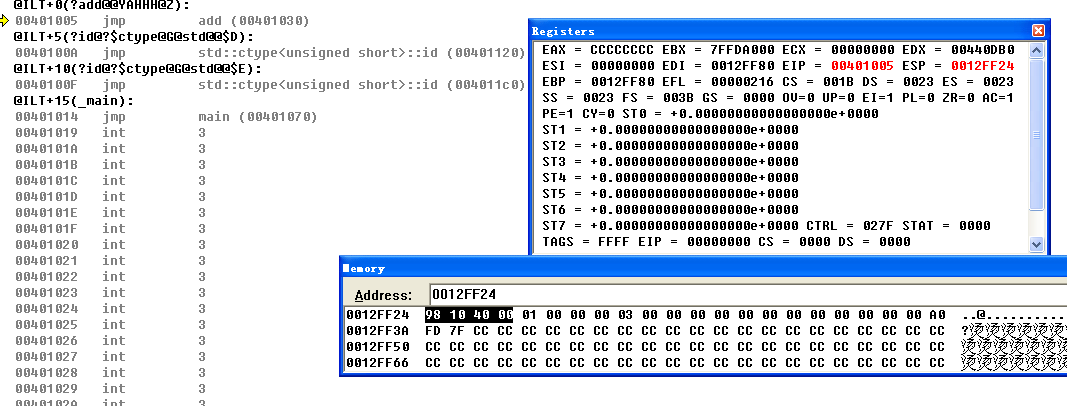


最后一步的mov指令便是将得到的结果eax的4赋值给n，对于n在定义时即为ebp-4，即0012FF7C，通过mov指令将结果4赋值给n，便完成了整个add函数的调用。

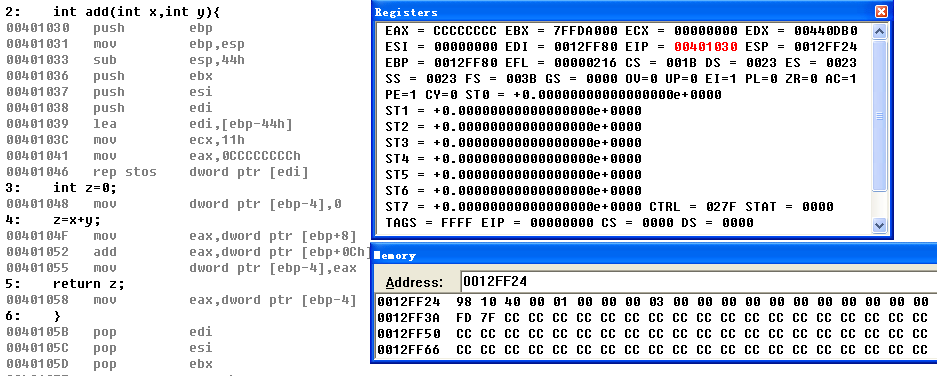


3. add函数内部栈帧切换等关键汇编代码

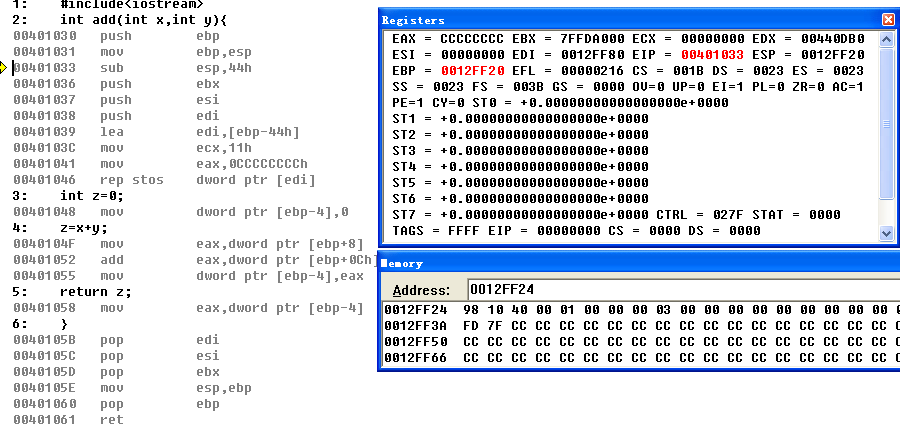
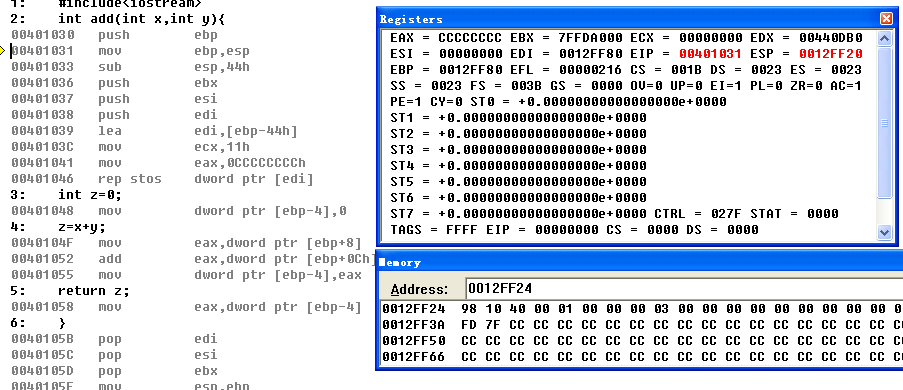
那对于add函数内部还有着很多的步骤，通过F11进入函数内部一步一步执行，第一步后我们得到了下一条运行的指令并不是00401098而是00401005，同时esp为0012FF24，我们查找esp该地址存储的内容，发现为98 10 40 00通过计算机自己的字节存储顺序我们可以发现这个值便是之前add函数下一条将要执行的语句的地址00401098，即这个步骤便是函数的返回地址入栈。



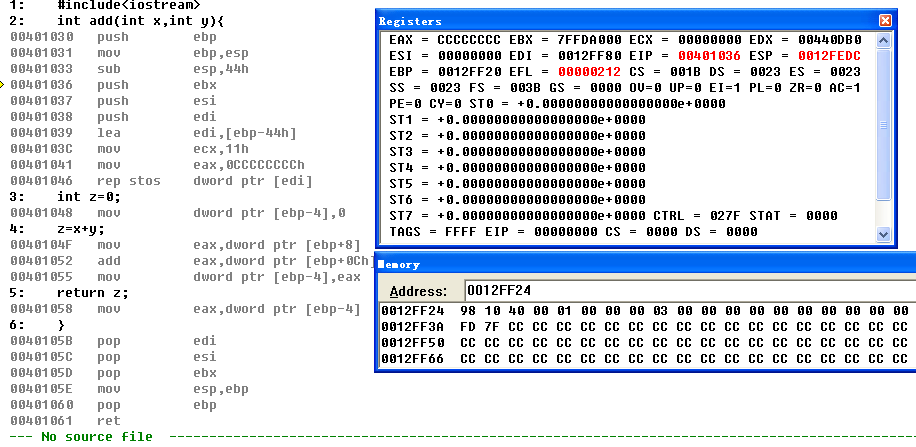
再次F11便进入了add函数代码段的执行



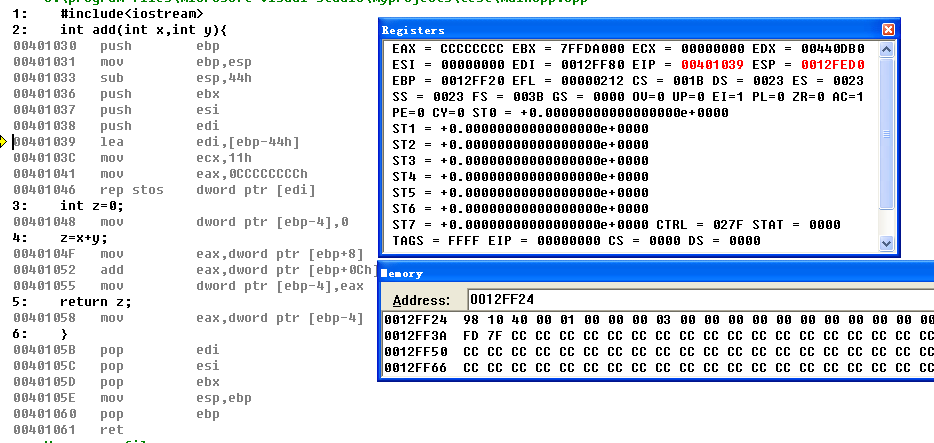
首先是ebp的压栈，这是为了下一步的更改ebp的值，再将esp的值赋值给ebp，这是对于add这个函数的调用重新的为其创建了一个基址，便于更好的调用；



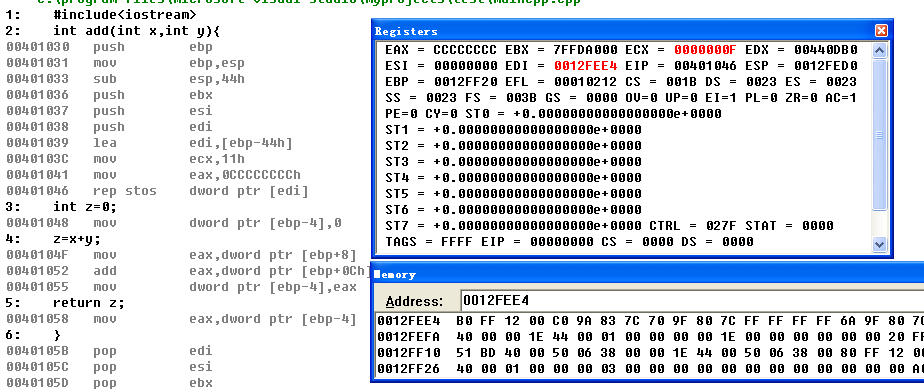
接下来esp减去44h，即栈顶向上走，为add这个函数开辟栈帧；



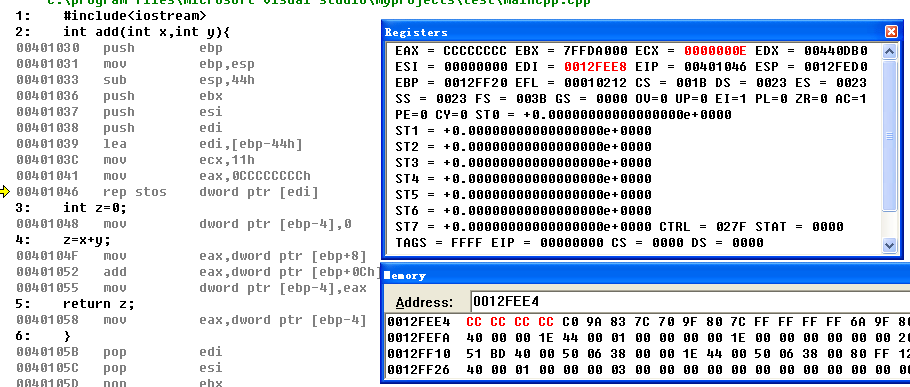
接下来便是将主函数可能会用到的一些寄存器的值进行入栈储存；



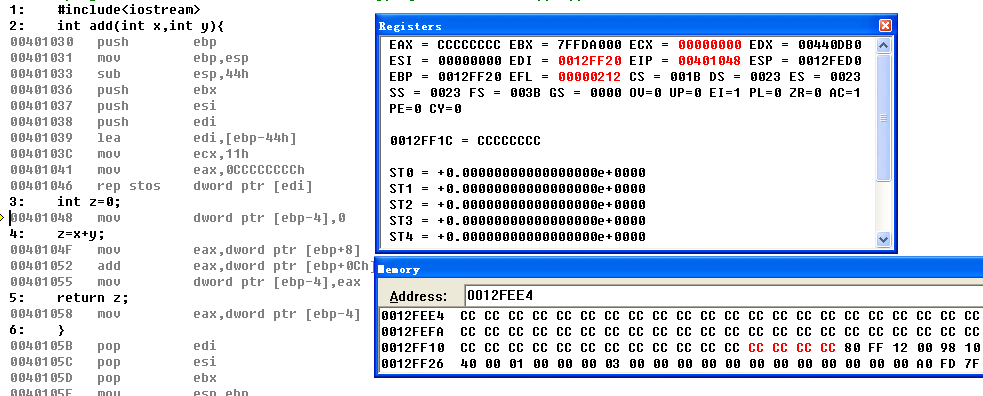
然后便是对开辟出的44h的空间进行初始化，ecx是计数寄存器，11h便是初始化的步骤循环11h次，每个都初始化为0CCCCCCCCh；



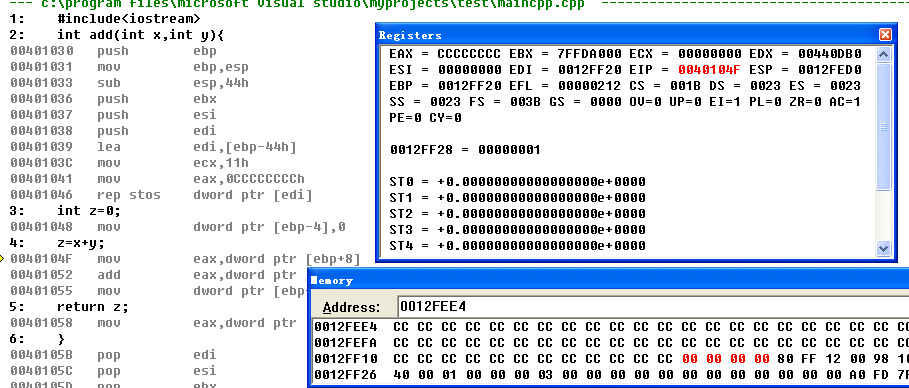
在ret出执行F11时可以看到每一次循环的变化，我们查找下一次的目标地址edi：0012FEE4，再F11执行一次循环后，我们可以看到该地址的值被初始化成了CC CC CC CC



当循环到ecx变成0时，循环便终止了，此时可以看到整个的内存区域便全部的初始化成了CC；

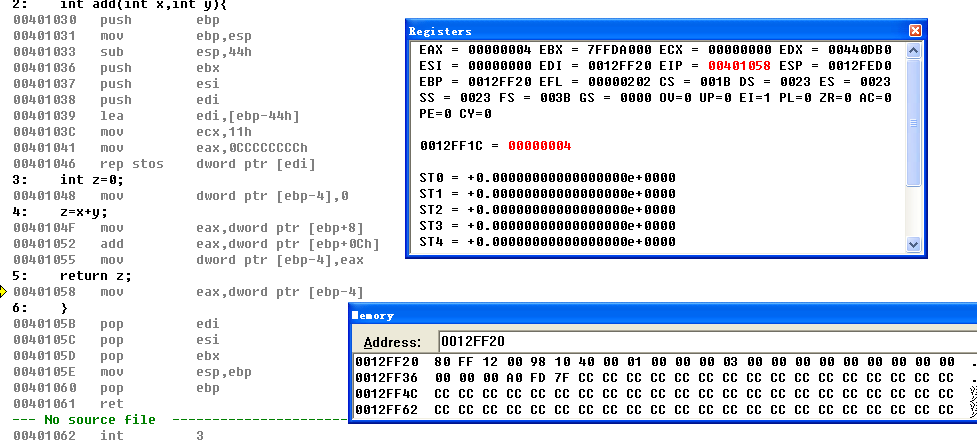
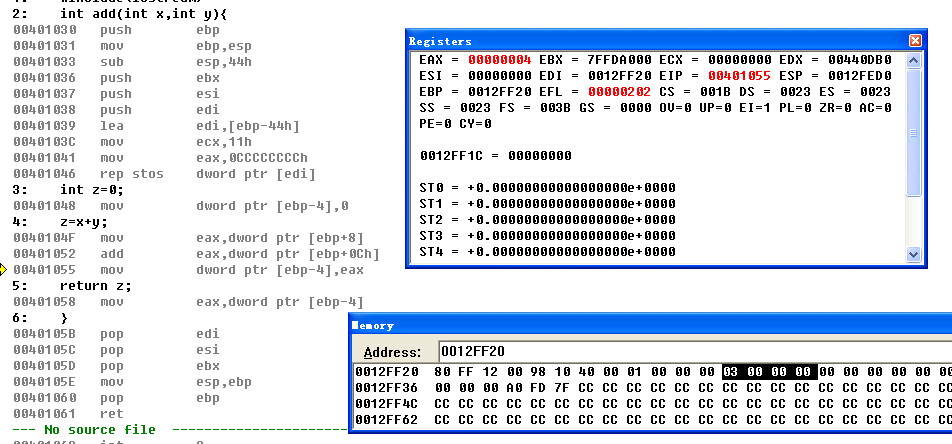
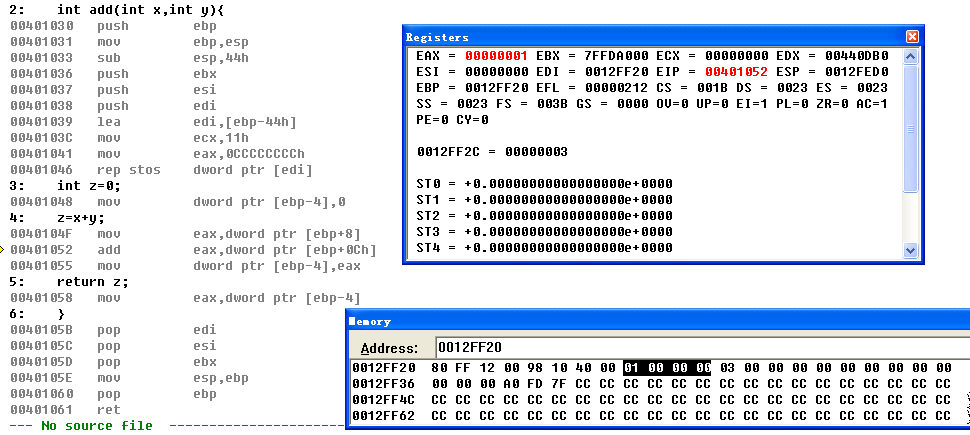


下一步便是将0赋值给ebp-4，效果为：

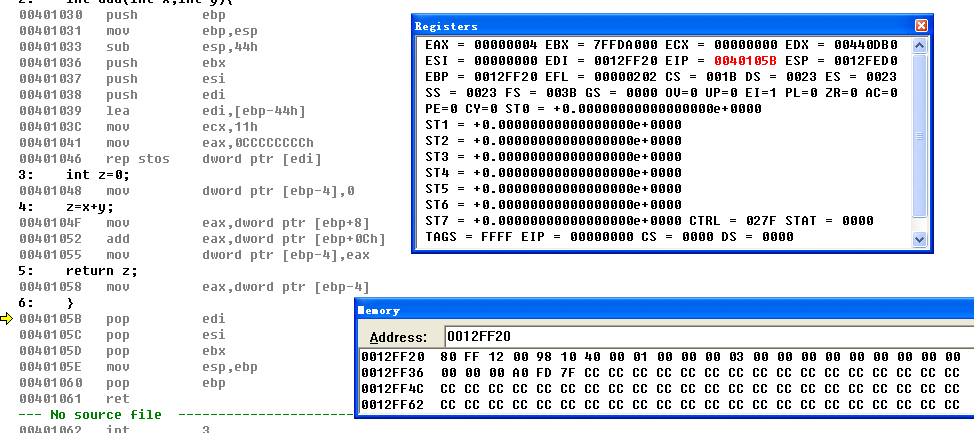


看到下一个便是ebp，即80 FF 12 00

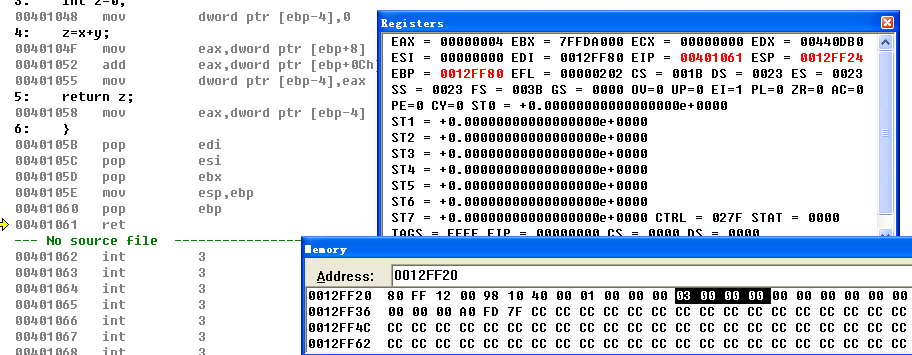
接下来便是将之前入栈的参数执行加法并将结果赋值给z；



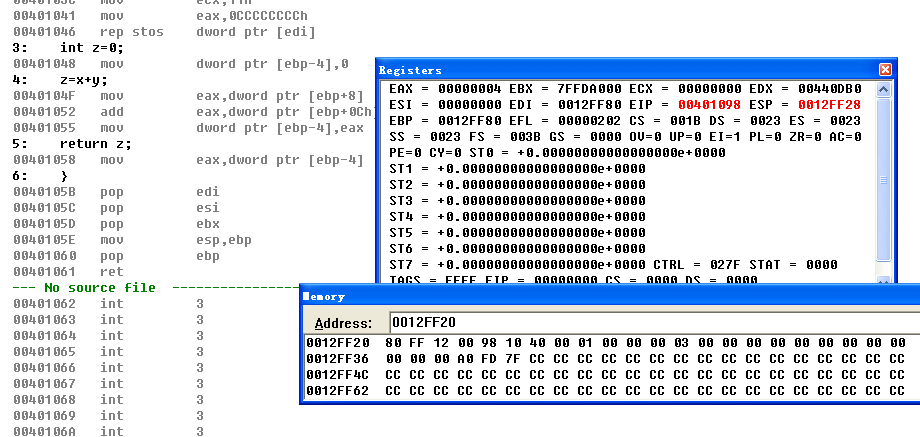
add函数的内部代码段便执行结束了，我们要将得到的值赋值给eax，然后eax在返回给调用的函数。这里并不是函数的结束。



对于函数，add本身要分配栈帧，在函数完成后也要还原栈帧，对于入栈的寄存器也要依次的弹出，对于重新创建的基址，也要依次的还原，将ebp赋值给esp，并将ebp弹出，并完全的还原到了，进入add代码段之前的状态。



当执行ret时add函数的调用便是真正的结束，执行一次F11



由于esp之前存的是函数的返回地址，再执行F11后下一条执行的指令便是返回地址00401098即add esp,8；即执行入栈参数1,3的栈帧还原，那么add函数便调用结束了。

同时我们知道之前函数将调用执行的结果存在了eax中，那么最后将eax的值赋值给ebp-4即变量n，便完成了整个n=add(1,3)的执行。

（此处根据实际操作过程，留下具体操作步骤、附加一些自己的理解，即可）

**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