《漏洞利用及渗透测试基础》实验报告

姓名：袁田 学号：2314022 班级：计科3班

**实验名称：**

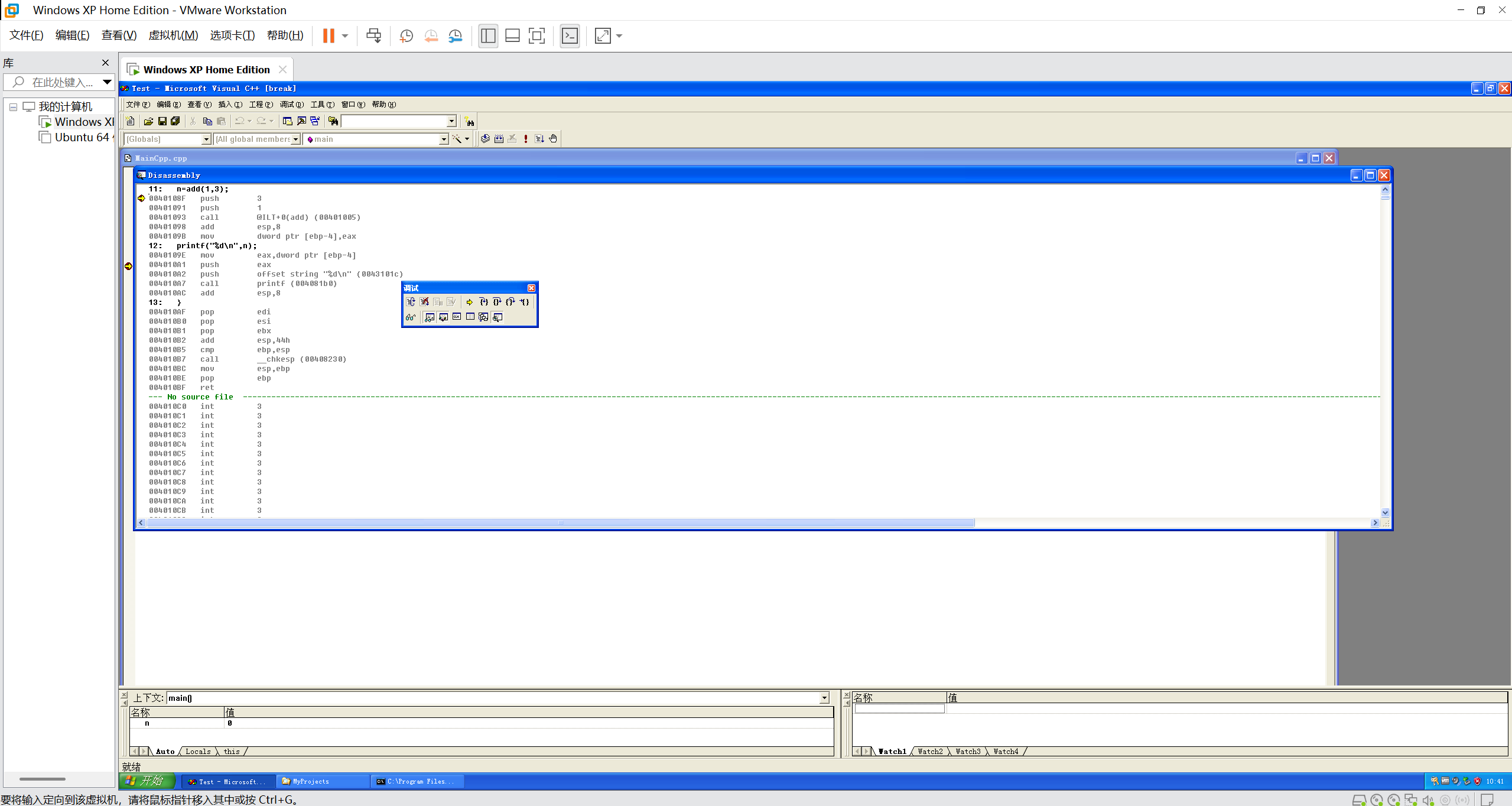
IDE反汇编实验

**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。

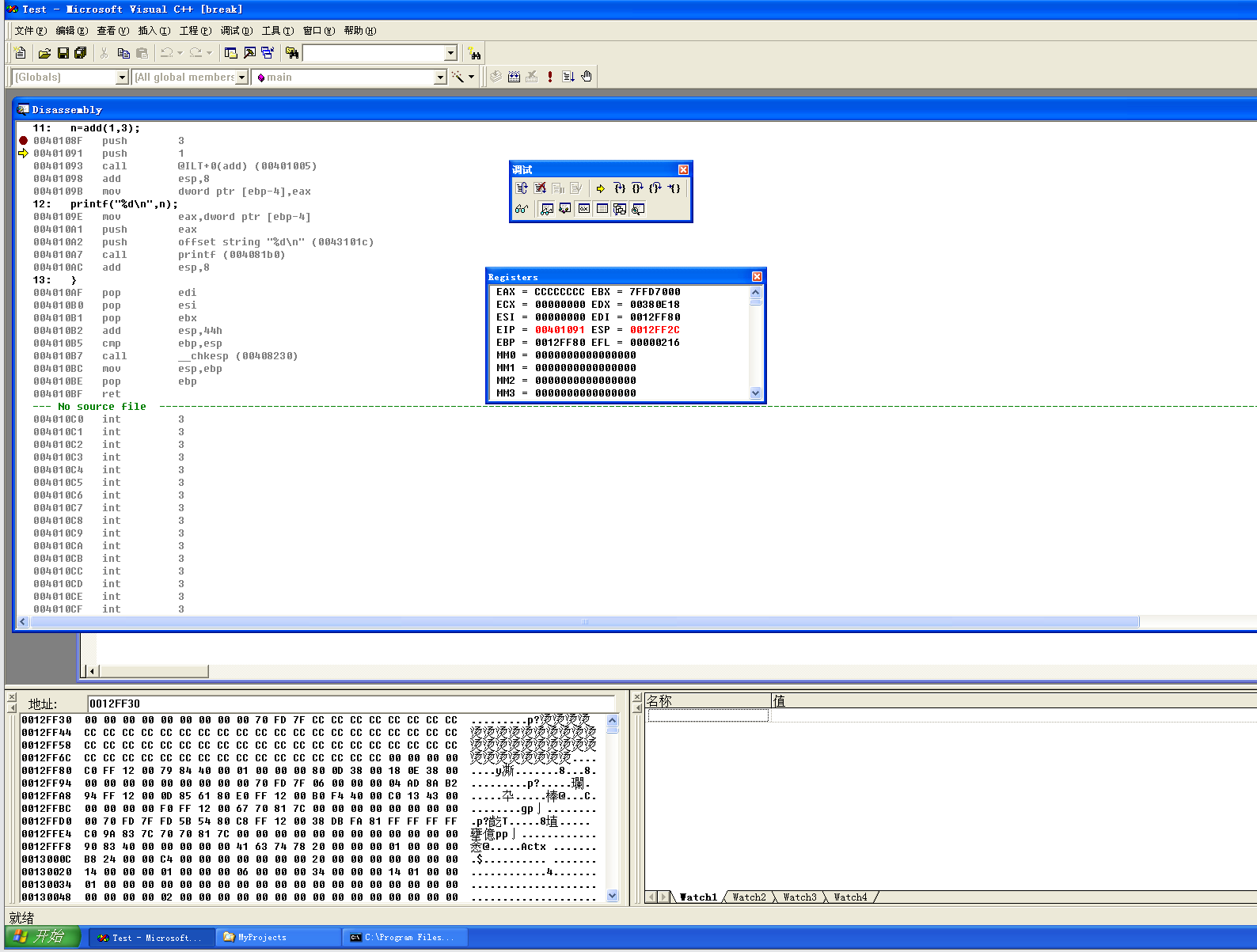
**实验过程：**

1. 进入VC反汇编



2. 观察add函数调用前后语句

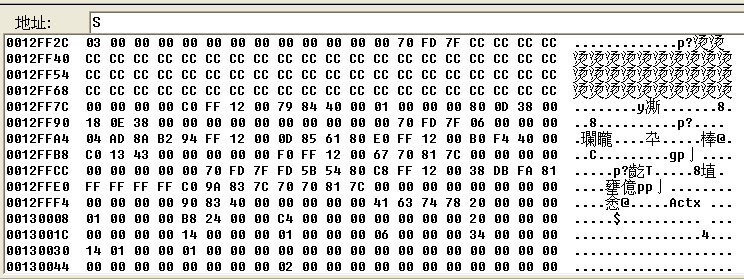
（1）函数调用时参数入栈操作下栈帧的变换



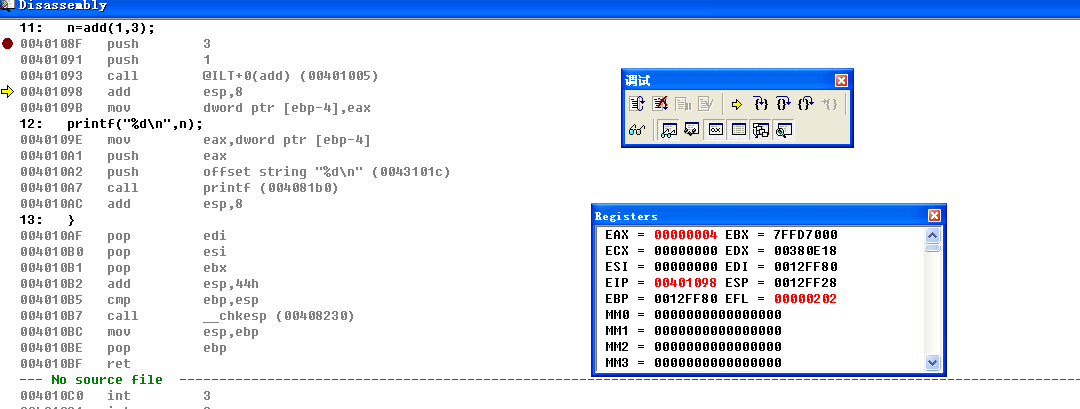
函数调用的顺序为从右至左，由于入栈操作是依次从高地址往低地址入栈

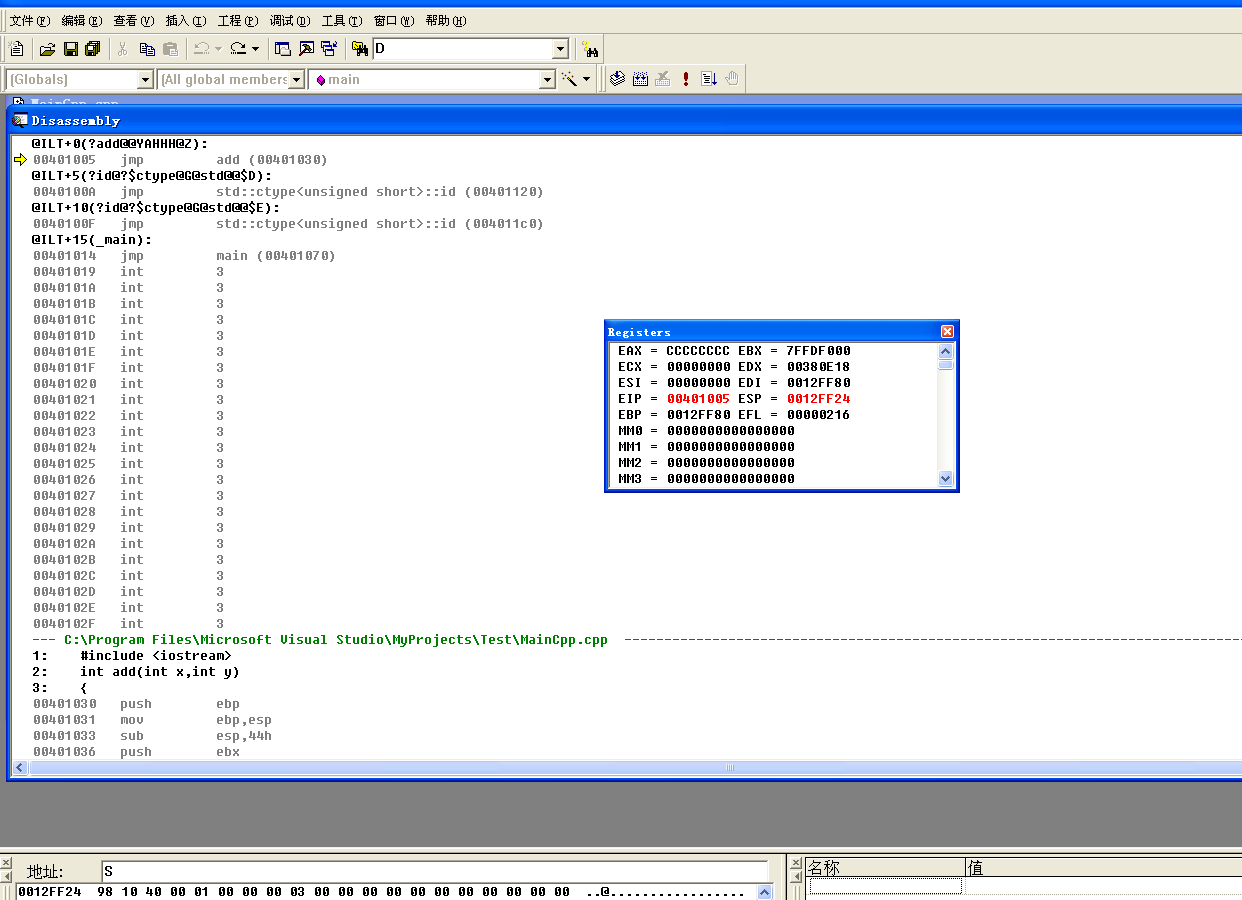
在执行了push语句后，ESP的地址变小，3入栈。1的入栈操作同理。

以下可见到，在地址0012FF2C中3入栈。



（2）执行call语句后，将返回地址压入栈，并跳转到add的入口地址进行加运算



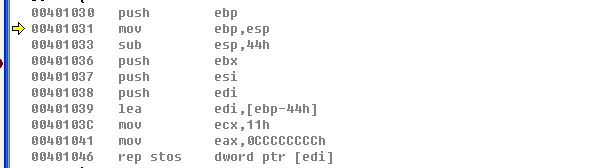


（3）执行add语句，将压入栈的两个参数清空（两个参数各自占据了4个字节）使得ESP回到参数执行前的位置，即0012FF30

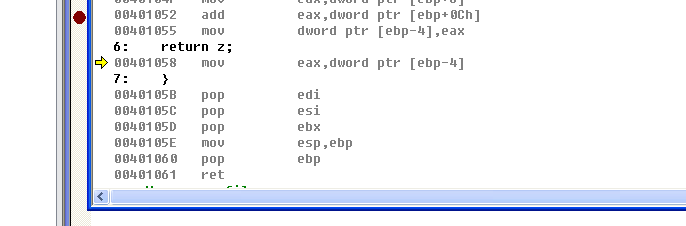


（4）将计算结果（已存到EAX中）赋值给变量n的地址[ebp-4]

3. add函数内部栈帧切换等关键汇编代码



将ebp替换为main函数栈帧中esp的值，再将esp减去44h，则得到了44h大小的内存空间进行add函数的指令进行，此处进行的栈帧的切换。



在函数结束时，进行栈帧的销毁

0040105E mov esp,ebp ;

00401060 pop ebp ;

这段语句，将esp变为目前的ebp值，相对于函数开头时，即为esp=esp+44h，回复到调用函数之前的主函数的esp值。同时利用pop语句，将之前存储的主函数的ebp值弹出，完成栈帧从调用函数到主函数的切换。

**心得体会：**

（1）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

之前在call语句执行时，由于call语句有保存返回指令地址的作用，ret语句相当于call语句的反转，用于将返回指令地址输出(EIP中保存的即为下一条指令的指令地址)，

ret指令实际就是执行了Pop EIP。使得在结束调用add函数后，继续执行主函数中后续的某些指令。

（2）此外，通过本实验，掌握了多个汇编语言的用法，例如：

call指令有两个作用，第一个作用是保存主函数中调用该函数的后的返回指令地址，第二个作用是切换栈帧到add函数的栈帧，实现函数的调用。

另外，还有关于

00401030 push ebp ;

00401031 mov ebp,esp ;

00401033 sub esp,44h ;

这段语句中，由于我们在主函数中调用函数，被调用的函数栈帧在主函数之上，所以要提前保存主函数的基址地址，即push ebp语句，随后将ebp的值变为主函数本身的esp值，并通过sub语句给该被调用函数提供栈帧(因为入栈是由高地址往低地址，因此对原有的esp做减法，同时将ebp替换为原esp的值，便可以在内存中留出空间进行函数的后续执行。