《漏洞利用及渗透测试基础》实验报告

姓名：禹相祐 学号：2312900 班级：1070

实验名称：

IDE 反汇编实验

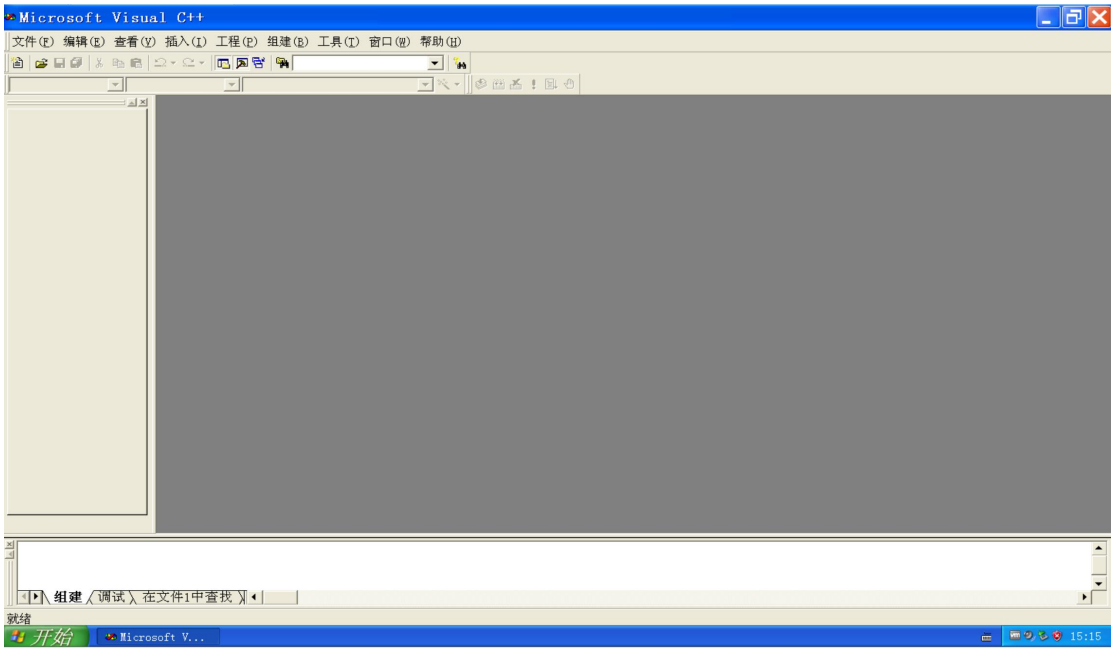
实验要求：

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：

(1)打开 VMwareStation 软件，选择虚拟机“Windows XP Professional”，然后打开 Microsoft Visual C++。



(2)新建项目，编写代码，运行并观察结果。

源代码：

homework_no1.cpp

```
#include <stdio.h>

int add(int variable_1,int variable_2)
{
    int sum=0;
    sum=variable_1+variable_2;
    return sum;
}
int main()
{
    int real_sum=0;
    real_sum=add(1,3);
    printf("%d\n",real_sum);
    system("pause");
    return 0;
}
```

运行结果：

(3)在代码 real_sum=add(1,3)处设置断点，然后进行调试，右键选择选择“Go To Disassembly”进行反汇编，得到相应的汇编代码。

底下为进入反汇编后的指令截图：

2.add 函数定义：

```
--- C:\Program Files\Microsoft Visual Studio\MyProjects\homework_no1\homework_no1.cpp
1:
2:    #include "stdafx.h"
3:    #include<stdlib.h>
4:    #include <stdio.h>
5:
6:    int add(int variable_1,int variable_2)
7:    {
00401010   push        ebp
00401011   mov         ebp,esp
00401013   sub         esp,44h
00401016   push        ebx
00401017   push        esi
00401018   push        edi
00401019   lea         edi,[ebp-44h]
0040101C   mov         ecx,11h
00401021   mov         eax,0CCCCCCCCh
00401026   rep stos    dword ptr [edi]
```

从上到下分别是：

Push ebp

// 这句保存了调用者(此程序中只有 main 调用了 add)的栈帧

Move ebp,esp

// 为 add 函数建立新的栈帧(ps:ebp 地址 > esp 地址)

Sub esp,44h

// sub 为减法 此处移动了 esp 的位置 为后面临时变量等留了 68(44h 为 16 进制数，表示 10 进制的 68)个字节的位置

Push ebx & Push esi & Push edi

// 都是为了保存寄存器现有的值，防止函数因为寄存器内存储的值的变化而发生错误，并且方便恢复原始状态。

Lea edi,[ebp-44h]

// lea 语句：表示将[ebp-44h]这个地址加载到 edi 这个寄存器内，让 edi 保存这个地址,方便后续函数操作

Mov ecx,11h

// 将 16 进制数 11(即 10 进制数 17)赋给 ecx 寄存器（ecx 寄存器通常表示计数器）

Mov eax,0CCCCCCCCh

// 同上,将值赋给 eax。但该值有特殊含义，一般用来填充未初始化的内存。

Rep stos dword ptr [edi]

// 跟上述两句连用，表示连续的将 edi 指向的 68 个字节的内存区域都填充为 0CCCCCCCCh(执行 17 次，每次 4 字节)

```
8:          int sum=0;
00401028    mov         dword ptr [ebp-4],0
9:          sum=variable_1+variable_2;
0040102F    mov         eax,dword ptr [ebp+8]
00401032    add         eax,dword ptr [ebp+0Ch]
00401035    mov         dword ptr [ebp-4],eax
10:         return sum;
00401038    mov         eax,dword ptr [ebp-4]
11:     }
0040103B    pop         edi
0040103C    pop         esi
0040103D    pop         ebx
0040103E    mov         esp,ebp
00401040    pop         ebp
00401041    ret
```

3. 观察 add 函数调用前后语句：

```
--- C:\Program Files\Microsoft Visual Studio\MyProjects\homework_no1\homework_no1.cpp  -------
12:   int main()
13:   {
0040D360   push        ebp
0040D361   mov         ebp,esp
0040D363   sub         esp,44h
0040D366   push        ebx
0040D367   push        esi
0040D368   push        edi
0040D369   lea         edi,[ebp-44h]
0040D36C   mov         ecx,11h
0040D371   mov         eax,0CCCCCCCCh
0040D376   rep stos    dword ptr [edi]
14:       int real_sum=0;
0040D378   mov         dword ptr [ebp-4],0
```

函数调用前基本同建立 add 函数时一样。

Mov dword ptr [ebp-4],0：

// 此处将 0 赋值给地址 ebp-4 处，此处存储的是变量 real_sum，用于存储求和后的结果

4.add 函数内部栈帧切换等关键汇编代码：

```
15:       real_sum=add(1,3);
0040D37F   push        3
0040D381   push        1
0040D383   call        @ILT+5(add) (0040100a)
0040D388   add         esp,8
0040D38B   mov         dword ptr [ebp-4],eax
16:       printf("%d\n",real_sum);
0040D38E   mov         eax,dword ptr [ebp-4]
0040D391   push        eax
0040D392   push        offset string "Hello World!\n" (00420f7c)
0040D397   call        printf (00401060)
0040D39C   add         esp,8
17:       system("pause");
0040D39F   push        offset string "pause" (0042001c)
0040D3A4   call        system (0040d250)
0040D3A9   add         esp,4
```

从上到下分别是：

Push 3 & Push 1:

// 让 add 函数的参数 3 and 1 从右到左依次压入栈 先压入 3 再压入 1

Call @ILT+5(add) (0040100a):

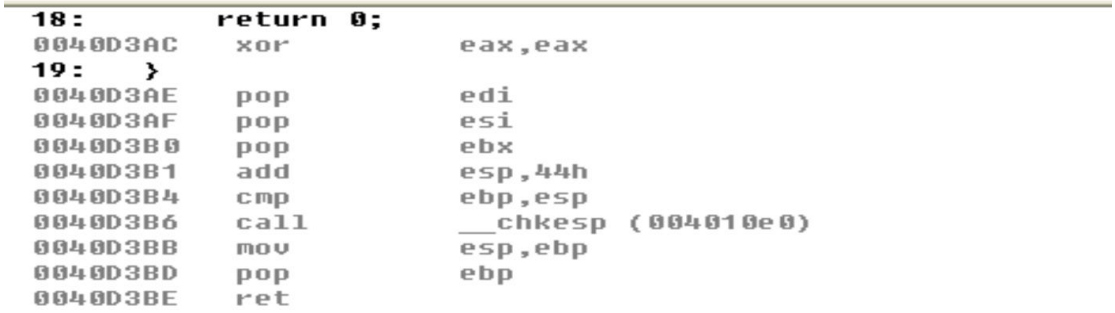
// call 语句调用 add 函数。通过跳转到 0040100a 处，再跳转到 add 函数对应的地址，实现调用

Add esp,8:

// 此处 add 函数有两个整数参数,占用了 8 个字节,当 add 函数执行完时，需要将栈指针调回执行 add 函数之前，所以将 esp 向上移动 8 个字节，恢复状态。

Mov dword ptr [ebp-4],eax:

// 将 eax 里面的值存到地址 ebp-4 处，结合上面的指令可知：此处 ebp-4 存的是代码中的变量“real_sum”，也就是用来存储 1,3 求和后结果的变量。



PS:此处为 main 函数结束后的汇编指令。

心得体会：

1. 学习到了 ebp 和 esp 是如何创建以及配合使用实现栈帧的创建与管理的；
2. 学习到了函数的参数是如何入栈以及协同运算，完成函数目标的;
3. 对 mov dword ptr[XXXX]这句指令有了更加准确的认识，不用再死记硬背。