北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

汇编语言与接口技术实验报告

学　　院：计算机学院

专　　业：计算机科学与技术

班　　级：07111701

学　　号：1120172118

姓　　名：邵雨洁

任课教师：李元章

2020 年 6 月 18 日

第1章 大数相乘

1.1 实验目的

1）学习汇编语言基本知识，掌握基本的汇编能力

2）学习控制台界面的汇编程序的编写

3）分析大数相乘问题，实现正确高效的代码编写

1.2 实验内容

大数相乘。要求实现两个十进制大整数的相乘(100位以上)，输出乘法运算结果。

1.3. 实验原理

两个100位以上的大数相乘，无法直接的通过MUL实现，在本次实验中所采用的基本思想是，模拟竖式计算的过程，然后将A的每一位与B的每一位进行相乘，步步累加。

1.4. 实验过程

1.4.1 流程图

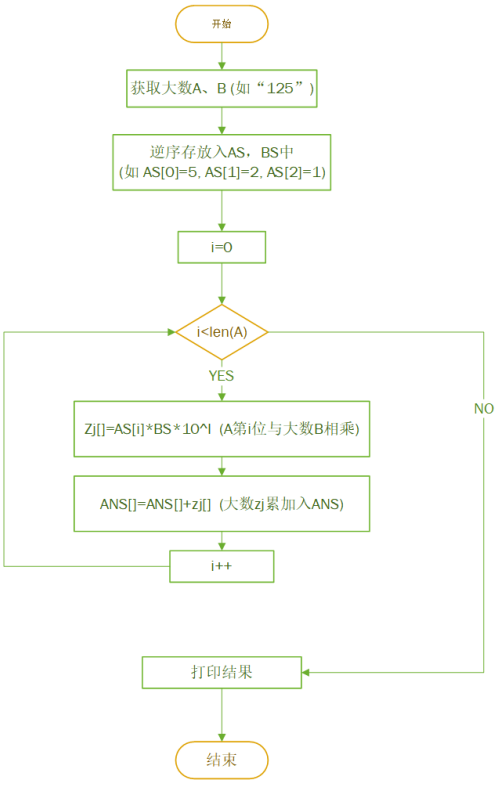

图1-1 整体流程

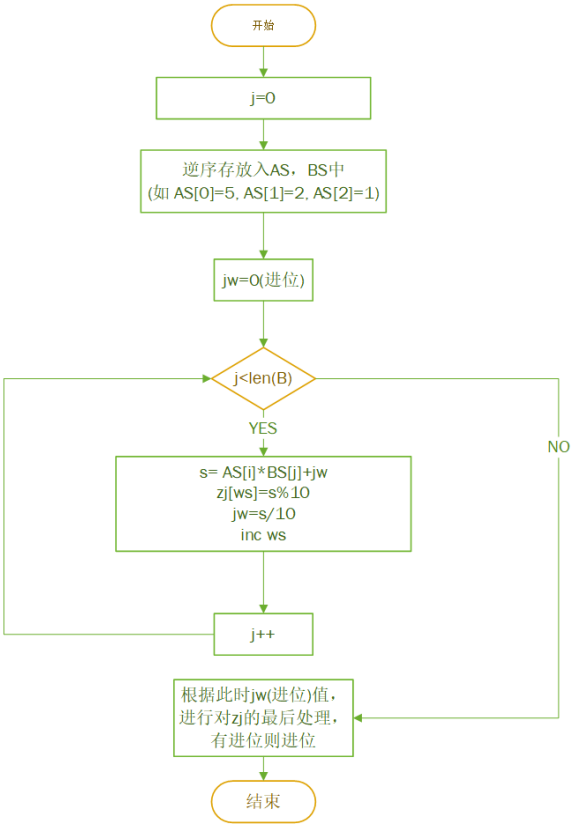

图1-2　zj[]=AS[i]*BS

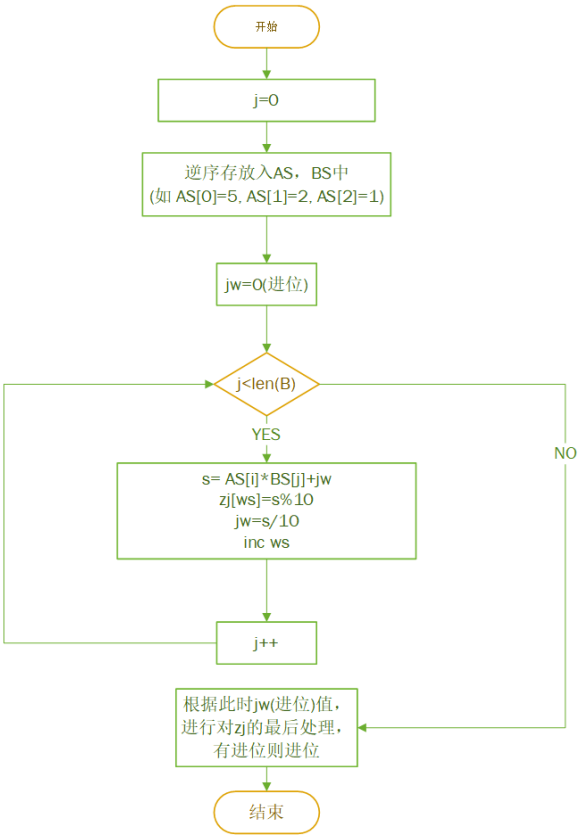

图1-3 ANS[]=ANS[]+zj[]

1.4.2 打印结果

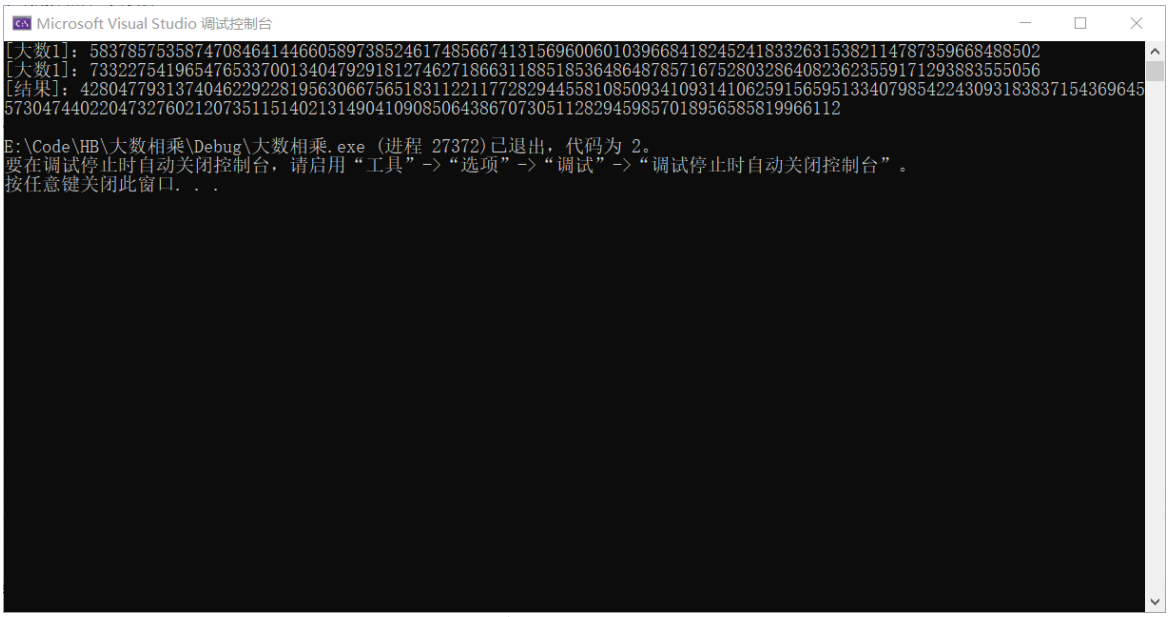

图1-4 结果打印

1.5. 实验总结

本次实验实现了第一个较为复杂，具有具体功能的汇编代码的编写，问题本身比较简单，思路清晰，但是在一些细节方面也要小心。在汇编实现过程中存在的问题主要有一下几点：

1）源操作数和目标操作数类型不匹配

2）对于指令没有认真回顾，导致可以通过一条指令实现的功能通过几条指令实现

3）对于符号，offset等认识不深刻，printf的使用比较僵化

4）指令本身格式出错，要一条一条排查，才能找到错误，一般都是格式、类似问题

在完成整个任务后，对以上存在的几个问题都有了改进。

通过这次编程实验，我对于编写汇编代码更加熟练，对汇编语言的基本知识有了更深的理解。

第2章 Windows界面文本比对

2.1 实验目的

1. 学习控制台界面的汇编，理解创建窗口的基本步骤，学会使用文本框、

按钮等基础控件

2. 学习对文件的基本操作

2.2 实验内容

Windows界面风格实现两个文本文件内容的比对。若两文件内容一样，输出相应提示；若两文件不一样，输出对应的行号。

2.3 实验步骤

2.3.1 窗口创建

主要步骤可以分为一下六步。

(1)获取程序句柄：执行成功，返回模块句柄，失败返回零。

```
invoke GetModuleHandle,NULL
```

(2)注册窗口类：定义了窗口一些主要属性。

```
; 注册窗口类
invoke LoadCursor,0,IDC_ARROW
mov @stWndClass.hCursor,eax                    ;窗口光标
push hInstance
pop @stWndClass.hInstance                      ;所属的实例句柄
mov @stWndClass.cbSize,sizeof WNDCLASSEX       ;结构的字节数
mov @stWndClass.style,CS_HREDRAW or CS_VREDRAW  ;类风格
mov @stWndClass.lpfnWndProc,offset _ProcWinMain ;窗口过程地址
mov @stWndClass.hbrBackground,COLOR_BTNFACE + 1 ;背景色;COLOR_WINDOW
mov @stWndClass.lpszClassName,offset szClassName;类名字符串的地址
invoke RegisterClassEx,addr @stWndClass
```

(3)建立窗口

```
; 建立并显示窗口
invoke CreateWindowEx,NULL, ;dwExStyle 风格
        offset szClassName,offset szCaptionMain,;窗口类名和标题
        WS_OVERLAPPEDWINDOW,;窗口的两个参数dwStyle和dwExStyle决定了窗口的外形和行为
        400,200,600,400,;水平，垂直，高，宽
        NULL,NULL,hInstance,NULL     ;hWndParent：窗口所属的父窗口
                                     ;hMenu：窗口上要出现的菜单的句柄
                                     ;hInstance：模块句柄，和注册窗口类时一样，指定了窗
                                         口所属的程序模块
                                     ;lpParam：指针，指向一个欲传给窗口的参数，该参数可
                                       在WM_CREATE消息中被获取
```

(4)显示窗口

```
invoke ShowWindow,hWinMain,SW_SHOWNORMAL
```

(5)刷新

```
invoke UpdateWindow,hWinMain
```

(6)消息循环

```
    ; 消息循环
  .while TRUE
    invoke GetMessage,addr @stMsg,NULL,0,0
    .break .if eax == 0
    invoke TranslateMessage,addr @stMsg
    invoke DispatchMessage,addr @stMsg
  .endw
```

2.3.2 文本框、按钮创建

与创建窗口使用的函数相同——CreateWindowEx，在具体参数上有变化。

2.3.4 文本比对

通过GetWindowText获取文本框中的文件地址，用CreateFile获取文件句柄，ReadFile读取文件全部内容保存到缓冲区，然后即可进行文件对比。

因为要输出不同行的行数，所以注意到换行占两个字节，前一个值为13，后一个值为10，通过这一点可以得知是否为一行。

具体比较的思路为以File1为基准，扫描，line记录当前行号，如果位数超过了File2,输出不同的行数；如果File1读完了，但File2仍有数据，输出不同的行数；如果出现不同，输出不同行数；如果相同，输出相同。

以MessageBox弹出结果信息，借助wsprintf将line转换为字符串，以在消息框中输出。

2.4 结果展示

2.4.1 主界面

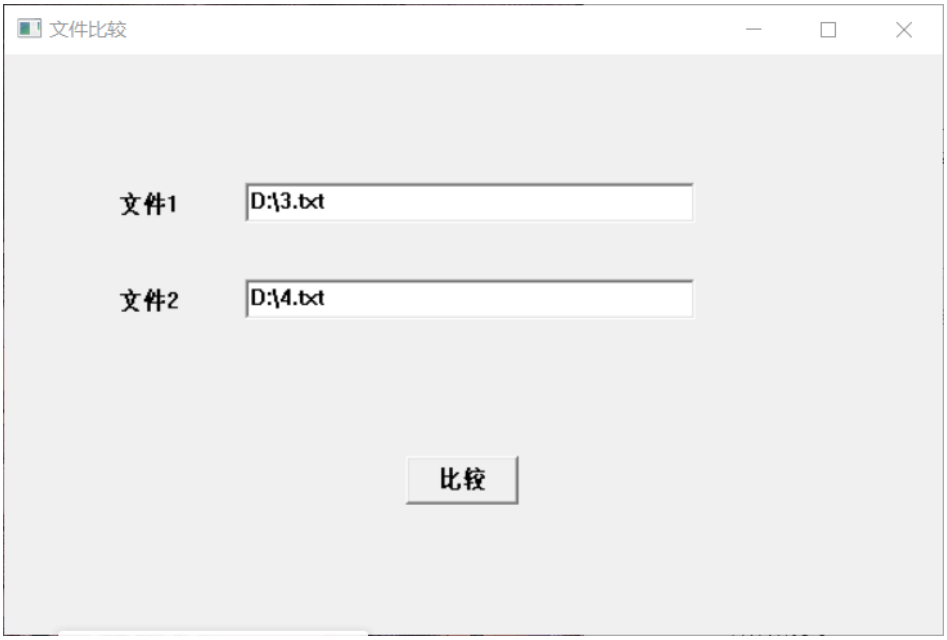

图 2-1 主界面

2.4.2 对比-相同

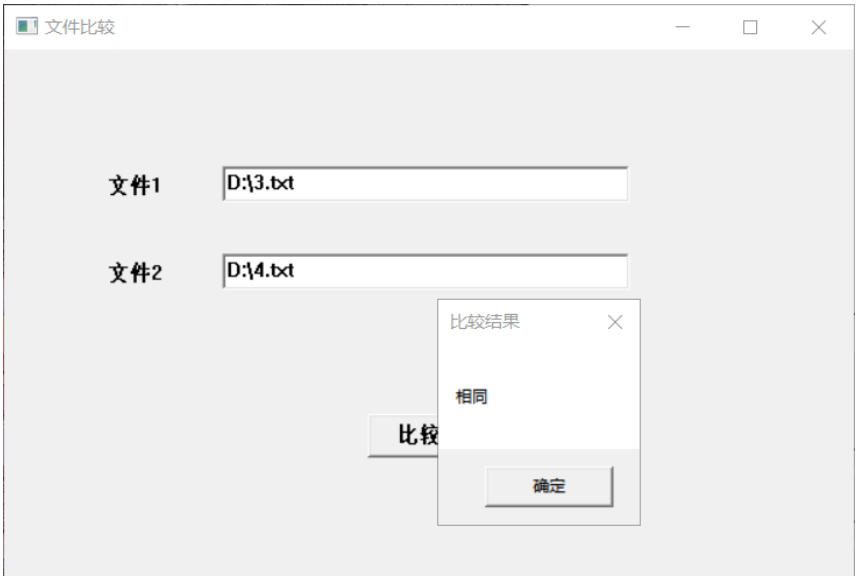

图2-2 结果-相同

图 2-3 对比文件

2.4.3 对比-不同

图2-4 结果-不同

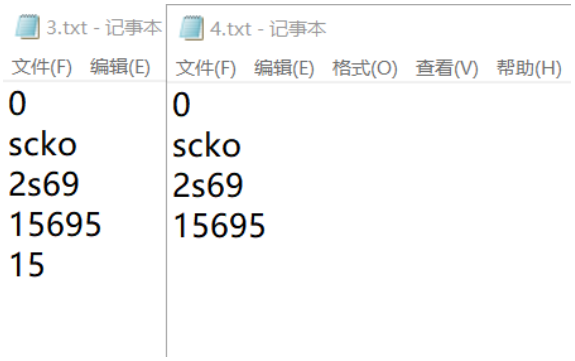

图 2-5 对比文件(从第一行开始，3.txt 第四行有换行，4.txt 第四行无换行，所有认为第四行起不同)

2.5 实验总结

本次实验主要学习了Windows界面的汇编程序编写，对于整体的框架是参考了网上一些现有的代码，然后自己进行了理解。代码中的一些API在操作系统课程设计中学习使用过，所以比较熟悉，有的虽然是第一次使用，但是思想方法是相类似的，所以在最初的困惑之后，能较好的理解创建窗口的一系列操作。

第二个问题是对于文件的读取，通过文本框可以获取文件的路径。最初是通过右击桌面的文件，查看属性来获得路径，意外的是，并不能通过属性里的位置找到文件，在找了好久之后才发现问题在此，并没有找到之所以会这样的原因。

本次实验的收获很多，让我对Window界面编程有了基础的认识，熟悉了基本流程，锻炼了汇编程序编写的能力。

第3章 C语言多重循环的反汇编

3.1 实验目的

1)学习阅读反汇编程序，了解程序的实现机理，学习编写高效率的程序

2)学习汇编循环结构的编写

3.2 实验内容

C语言编写多重循环程序（大于3重），查看其反汇编码，分析各条语句功能，并采用汇编语言重写相同功能程序。

3.3 实验步骤

3.3.1 C语言多重循环程序编写

编写了简单的具有四重循环的程序，每重循环的首尾打印层数信息。

```c
#include<stdio.h>
int main() {
    int a = 0;
    int i, j, k, w;
    for (i = 0; i <= 1; i++) {
        printf("第一层开始\n");
        for (k = 0; k <= i; k++) {
            printf("  第二层开始\n");
            for (j = 0; j <= k; j++) {
                printf("    第三层开始\n");
                for (w = 0; w <= j; w++) {
                    a = a+1;
                    printf("      第四层\n");

                }
                printf("    第三层结束\n");
            }
            printf("  第二层结束\n");
        }
        printf("第一层结束\n");
    }
}
```

图3-1 C语言多重循环程序

```
Microsoft Visual Studio 调试控制台
第一层开始
  第二层开始
    第三层开始
      第四层
    第三层结束
  第二层结束
第一层结束
第一层开始
  第二层开始
    第三层开始
      第四层
    第三层结束
  第二层结束
  第二层开始
    第三层开始
      第四层
    第三层结束
    第三层开始
      第四层
      第四层
    第三层结束
  第二层结束
第一层结束

E:\Code\C&C++\loop\Debug\loop.exe (进程 33956)已退出，代码为 0。
```

图3-2 打印结果

3.3.2 反汇编代码分析

(1) 首尾部分

```
        --- E:\Code\C&C++\loop\源.cpp ----------------------------------------------------
#include<stdio.h>
int main() {
00FD4E60  push        ebp                     ;ebp入栈
00FD4E61  mov         ebp,esp                 ;保留当前栈顶指针
00FD4E63  sub         esp,0FCh                ;临时变量的位置
00FD4E69  push        ebx
00FD4E6A  push        esi
00FD4E6B  push        edi                     ;保存寄存器状态
00FD4E6C  lea         edi,[ebp-0FCh]          ;获取临时变量缓存区的地址
00FD4E72  mov         ecx,3Fh                 ;每次处理4个字节，所有对于0FCh,有(0FCh/4)=03Fh
00FD4E77  mov         eax,0CCCCCCCCh          ;与int 3中断有关
00FD4E7C  rep stos    dword ptr es:[edi]      ;重复执行，对整个临时变量区进行初始化，ecx次
00FD4E7E  mov         ecx,offset _43A689A6_源@cpp (0FDC003h)        ;vs添加的调试指令
00FD4E83  call        @__CheckForDebuggerJustMyCode@4 (0FD1208h)    ;vs添加的调试指令
```

……（主体见下一部分）

```
;退出，恢复此前edi,esi,ebx寄存器的值
00FD4F71  pop         edi
00FD4F72  pop         esi
00FD4F73  pop         ebx             ;出栈，还原寄存器的初值
00FD4F74  add         esp,0FCh        ;清除栈内容
00FD4F7A  cmp         ebp,esp
00FD4F7C  call        __RTC_CheckEsp (0FD1212h);检查esp值是否恢复
00FD4F81  mov         esp,ebp
00FD4F83  pop         ebp
00FD4F84  ret
```

(2) 主体部分

```
        int a = 0;
00FD4E88  mov         dword ptr [a],0         ;将0赋值给a
        int i, j, k, w;
        for (i = 0; i <= 1; i++) {
00FD4E8F  mov         dword ptr [i],0         ;第一层循环初始化
00FD4E96  jmp         main+41h (0FD4EA1h)     ;跳转去第一层循环条件判断
00FD4E98  mov         eax,dword ptr [i]       ;第一层循环内部执行一次后，在跳转到此处
00FD4E9B  add         eax,1                   ;执行每次循环体结束后的i++
00FD4E9E  mov         dword ptr [i],eax
00FD4EA1  cmp         dword ptr [i],1         ;判断是否满足第一层循环,满足进入循环：00FD4EAB
00FD4EA5  jg          main+10Fh (0FD4F6Fh)    ;不满足条件跳出第一层循环
            printf("第一层开始\n");
00FD4EAB  push        offset string "\xb5\xda\xd2\xbb\xb2\xe3\xbf\xaa\xca\xbc\n" (0FD7CE0h)
00FD4EB0  call        _printf (0FD1375h)      ;第一层循环内部
00FD4EB5  add         esp,4                   ;到栈下一个位置，准备给k赋值
            for (k = 0; k <= i; k++) {
00FD4EB8  mov         dword ptr [k],0         ;第二层循环初始化
00FD4EBF  jmp         main+6Ah (0FD4ECAh)     ;跳转去第二层循环条件判断
00FD4EC1  mov         eax,dword ptr [k]       ;第二层循环内部执行一次后，在跳转到此处
```

```
00FD4EC4  add         eax,1                     ;执行每次循环体结束后的k++
00FD4EC7  mov         dword ptr [k],eax
00FD4ECA  mov         eax,dword ptr [k]
00FD4ECD  cmp         eax,dword ptr [i]    ;判断是否满足第二层循环条件,满足进入循环:00FD4ED6
                for (k = 0; k <= i; k++) {
00FD4ED0  jg          main+0FDh (0FD4F5Dh)    ;不满足跳出第二层循环，执行第一层下一条语句
                     printf("  第二层开始\n");
00FD4ED6  push        offset string "\xb5\xda\xd2\xbb\xb2\xe3" (0FD7BCCh)
00FD4EDB  call        _printf (0FD1375h)
00FD4EE0  add         esp,4                     ;到栈下一个位置，准备给j赋值
                     for (j = 0; j <= k; j++) {
00FD4EE3  mov         dword ptr [j],0           ;第三层循环初始化
00FD4EEA  jmp         main+95h (0FD4EF5h)       ;跳转去第三层循环条件判断
00FD4EEC  mov         eax,dword ptr [j]         ;第三层循环内部执行一次后，在跳转到此处
00FD4EEF  add         eax,1                     ;执行每次循环体结束后的j++
00FD4EF2  mov         dword ptr [j],eax
00FD4EF5  mov         eax,dword ptr [j]         ;判断是否满足第三层循环条件,满足进入循
环:00FD4EFD
00FD4EF8  cmp         eax,dword ptr [k]
00FD4EFB  jg          main+0EBh (0FD4F4Bh)   ;不满足跳出第三层循环，执行第二层下一条语句
                        printf("    第三层开始\n");
00FD4EFD  push        offset string "    \xb5\xda\xc8\xfd\xb2\xe3\xbf\xaa\xca\xbc\n"
(0FD7BE0h)
00FD4F02  call        _printf (0FD1375h)
00FD4F07  add         esp,4                     ;到栈下一个位置，准备给w赋值
                        for (w = 0; w <= j; w++) {
00FD4F0A  mov         dword ptr [w],0           ;第四层循环初始化
00FD4F11  jmp         main+0BCh (0FD4F1Ch)    ;跳转去第四层循环条件判断
00FD4F13  mov         eax,dword ptr [w]         ;第四层循环内部执行一次后，在跳转到此处
00FD4F16  add         eax,1                     ;w++
00FD4F19  mov         dword ptr [w],eax
00FD4F1C  mov         eax,dword ptr [w]
00FD4F1F  cmp         eax,dword ptr [j];判断是否满足第四层循环条件,满足进入循环:00FD4F24
00FD4F22  jg          main+0DCh (0FD4F3Ch)   ;不满足跳出第三层循环，执行第三层下一条语句
                              a = a+1;
00FD4F24  mov         eax,dword ptr [a]         ;第四层循环内容
00FD4F27  add         eax,1
00FD4F2A  mov         dword ptr [a],eax
                              printf("      第四层\n");
00FD4F2D  push        offset string "      \xb5\xda\xcb\xc4\xb2\xe3" (0FD7CD0h)
00FD4F32  call        _printf (0FD1375h)
00FD4F37  add         esp,4

                        }
00FD4F3A  jmp         main+0B3h (0FD4F13h)   ;执行完第四层一次循环,跳到循环体后的w++
                        printf("    第三层结束\n");
00FD4F3C  push        offset string "    \xb5\xda\xc8\xfd\xb2\xe3\xbd\xe1\xca\xf8\n"
(0FD7CECh)  ;跳出第四层循环后，进入此处，执行第三层循环的下一条语句
00FD4F41  call        _printf (0FD1375h)
00FD4F46  add         esp,4
                     }
00FD4F49  jmp         main+8Ch (0FD4EECh)   ;执行完第三层一次循环,跳到循环体后的j++
                     printf("  第二层结束\n");
00FD4F4B  push        offset string "  \xb5\xda\xb6\xfe\xb2\xe3\xbd\xe1\xca\xf8\n"
(0FD7E20h)  ;第三层循环结束，跳到此处执行第二层下一条语句
```

```
00FD4F50  call        _printf (0FD1375h)
00FD4F55  add         esp,4
                }
00FD4F58  jmp         main+61h (0FD4EC1h)    ;执行完第二层一次循环,跳到循环体后的k++
            printf("第一层结束\n");
00FD4F5D  push        offset string "\xb5\xda\xd2\xbb\xb2\xe3\xbd\xe1\xca\xf8\n" (0FD7E30h)
                                             ;第二层循环结束，跳到此处执行第一层下一条语句
00FD4F62  call        _printf (0FD1375h)
00FD4F67  add         esp,4
        }
00FD4F6A  jmp         main+38h (0FD4E98h)    ;执行完第一层一次循环,跳到循环体后的k++
}
00FD4F6F  xor         eax,eax                ;寄存器清零

}
```

3.3.3 汇编代码编写(见5Multicycle.asm)

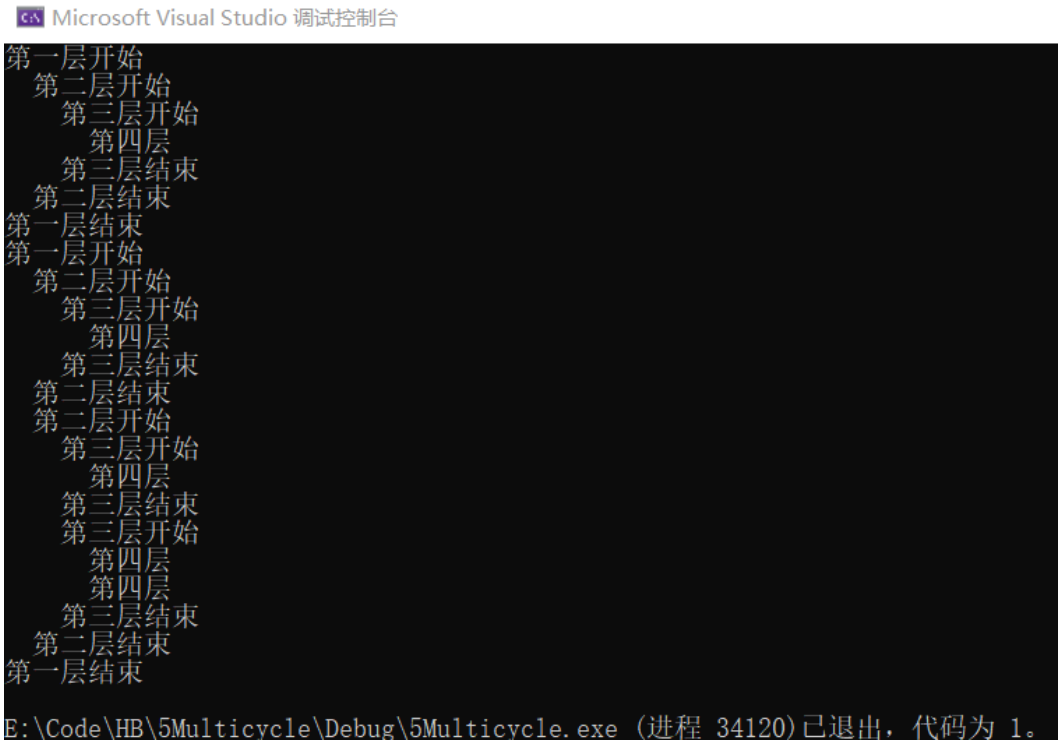

图3-3 汇编打印结果

3.4 实验总结

本次实验中主要是对C语言的反汇编进行了分析阅读，主体部分和自己的汇编思路并没有太多差别，所有比较好理解，主要是对于一个函数的开头入栈、结尾出栈的操作理解存在一些问题，对其功能性的认识比较到位。

本次的反汇编分析和汇编代码的编写针对的都是循环语句，循环语气其实也没有什么特别之处，在理解好其逻辑顺序，明白下一步要做些什么，就很清晰。并且，在编译原理中我们已经有过类似的操作，所有还是比较简单的。