

北京理工大学

本科生毕业设计（论文）

基于 MASM 的汇编程序设计

Assembly programming based on MASM

学 院：	计算机学院
专 业：	计算机科学与技术
学生姓名：	唐小娟
学 号：	1120180207
指导教师：	李元章

2021 年 5 月 14 日

原创性声明

本人郑重声明：所呈交的毕业设计（论文），是本人在指导老师的指导下独立进行研究所取得的成果。除文中已经注明引用的内容外，本文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。

特此申明。

本人签名：

日期：

年

月

日

关于使用授权的声明

本人完全了解北京理工大学有关保管、使用毕业设计（论文）的规定，其中包括：①学校有权保管、并向有关部门送交本毕业设计（论文）的原件与复印件；②学校可以采用影印、缩印或其它复制手段复制并保存本毕业设计（论文）；③学校可允许本毕业设计（论文）被查阅或借阅；④学校可以学术交流为目的，复制赠送和交换本毕业设计（论文）；⑤学校可以公布本毕业设计（论文）的全部或部分内容。

本人签名：

日期：

年

月

日

指导老师签名：

日期：

年

月

日

基于 MASM 的汇编程序设计

摘 要

本文采用 MASM，以 visual studio 2019 作为集成开发环境，选取了“大数乘法”、“文件对比”和“多重循环分析”三个实验。实验涵盖了实验目的、实验环境、实验设计、代码实现、实验结果五个部分。

我在其中掌握了基本的汇编语言程序设计技能，对底层代码的分析和优化有了进一步的理解，为我在后续的学习中打下了坚实的基础。

关键词：汇编语言与接口设计；大数乘法；文件比对；多重循环分析

Assembly programming based on MASM

Abstract

This paper uses MASM, Visual Studio 2019 as the integrated development environment, and selects three experiments: large number multiplication, file comparison and multiple loop analysis. The experiment covers five parts: the purpose of the experiment, the environment of the experiment, the design of the experiment, the implementation of the code.

I have mastered basic assembly language programming skills and further understood the analysis and optimization of the underlying code, which has laid a solid foundation for my subsequent study.

Key Words: assembly language and interface design; Multiplication of large numbers; File comparison; Multiple cycle analysis

目 录

摘 要	I
Abstract	II
第 1 章 实验目的	1
第 2 章 实验环境	2
第 3 章 大数相乘	3
3.1 实验目的	3
3.2 实验设计	3
3.2.1 程序流程	3
3.3 具体实现	3
3.3.1 计算字符串长度	3
3.3.2 逆序存放到数组	4
3.3.3 数字相乘	6
3.3.4 进位处理	8
3.3.5 逆序输出	9
3.4 实验结果展示	11
3.4.1 输入正数	11
3.4.2 输入负数	11
第 4 章 文件对比	12
4.1 实验目的	12
4.2 实验设计	12
4.2.1 程序流程	12
4.3 具体实现	13
4.3.1 建立窗口	13
4.3.2 窗口过程函数	15
4.3.3 文件对比	19
4.4 实验结果展示	22
4.4.1 两个文件相同	22
4.4.2 两个文件内容不同	24
第 5 章 多重循环分析	25
5.1 实验目的	25
5.2 程序流程	25

5.3 具体实现	25
5.3.1 多重循环的 C 语言代码	25
5.3.2 反汇编分析	26
5.3.3 重写多重循环	31
5.4 实验结果展示	33
总 结	34

第 1 章 实验目的

该实验进行汇编程序语言设计，通过实验大数乘法、文件比对、多重循环分析三个实验，掌握 80x86 的基本指令集，了解汇编程序的基本结构包括循环和分支。同时熟练运用 VS 进行汇编语言的调试，深入理解底层数据的内存分布，学习调用 windows 库函数和 C 语言库函数，将汇编语言从理论转化到实践，提高自己的编程能力。

第 2 章 实验环境

表 2-1 实验环境信息

名称	信息
操作系统	Windows 家庭中文版
IDE	Visual Studio 2019
编译环境	MASM32

第3章 大数相乘

3.1 实验目的

实现一个大数相乘的程序，输入两个大数，输出它的乘法结果。

3.2 实验设计

3.2.1 程序流程

1. 输入两个字符串；
2. 计算输入字符串的长度；
3. 逆序存放数据到数组中；
4. 进行大数相乘；
5. 对数组中的每一项进行进位处理；
6. 逆序输出结果。

3.3 具体实现

该节介绍了代码中的部分核心算法和具体代码，包括了计算字符串的长度、逆序存放、数组相乘、进位处理、逆序输出。

3.3.1 计算字符串长度

流程如下：

1. 取出字符串的地址；
2. 取出字符串地址的长度；
3. 循环取出字符串；

4. 判断是否为 0;

(a) 如果为 0，则跳出循环;

(b) 否则，继续;

5. 字符串的地址加 1，长度加 1。

具体代码如下：

```
1 ; 计算输入字符串的长度
2 ; stdcall 可以自动平衡堆栈
3 getLen proc stdcall    numstring: ptr byte , numLen: ptr DWORD
4     ; ecx 清 0，便于循环计数
5     xor ecx, ecx
6     ; esi 保存 numstring 的首地址
7     mov esi, numstring
8     ; edi 保存 numLen 的地址
9     mov edi, numLen
10 L1:
11     mov al, [esi]
12     cmp al, 0
13     jz L2
14     inc ecx
15     inc esi
16     jmp L1
17 L2:
18     mov [edi], ecx
19     ret
20 getLen endp
```

3.3.2 逆序存放到数组

逆序存放到数组中，方便在计算乘法后进位不用考虑溢出。

流程如下：

1. 取出字符串地址；
2. 取出相应数组的地址；
3. 给计数器 `ecx` 赋值长度，便于循环；
4. 取出字符；
5. 判断是否为负号；
 - (a) 如果是负号，给标记 `negNum` 加 1，由于负号是第一个字符，也跳出循环；
 - (b) 否则，给字符-“0”得到对应字符的数值；
6. 直到计数器为 0，跳出循环。

具体代码如下：

```
1 ;将得到的字符串转化为数字逆序保存在数组中
2 revNum  proc  stdcall
3     numString: ptr byte ,numArray: ptr DWORD,numLen: ptr DWORD
4     mov esi ,numArray
5     mov edi ,numLen
6     mov ecx ,[ edi ]
7     mov edi ,numString
8     xor eax ,eax
9 L1:
10    mov al ,[ edi ][ ecx -1]
11    cmp al ,45
12    jz  L2
13    sub eax ,30H
14    mov [ esi ],eax
15    add esi ,4
16    loop L1
```

```
17     ret
18 L2:
19     add negNum,1           ;此时肯定是最后一个字符串
20     mov edi,numLen
21     mov eax,[edi]
22     dec eax
23     mov [edi],eax
24     ret
25
26 revNum endp
27 \begin{lstlisting}
```

3.3.3 数字相乘

将得到数组中的每一个元素对应相乘再相加

流程如下：

1. 先进行外层循环（计数器为 esi），如果循环次数大于 num1 的长度，结束函数；
2. 进行内层循环（计数器为 edi），对 edi 清 0，取 num1[esi] 分别与 num2[edi] 相乘，相加到对应的 num3[esi+edi] 即可；（注意这里的 esi 和 edi 不考虑地址，代表数组的第几个数）

具体代码如下：

```
1 ;进行大数相乘(两层循环)
2 numMul proc stdcall
3     num1:ptr dword,num2:ptr dword,num3:ptr dword,
4     len1:dword,len2:dword,len3:ptr dword
5
6     ;第一层循环的计数
7     xor esi,esi
8     ;第二层循环的计数
```

```
9      xor edi,edi
10     mov edx,num1
11     mov ebx,num2
12     mov ecx,num3
13
14 L2:
15     ; 第一层循环
16     xor edi,edi
17     cmp esi,len1
18     ; 小于len1
19     jb L3
20     mov eax,len1
21     add eax,len2
22     sub eax,1
23     mov esi,len3
24     mov [esi],eax
25     ret
26 L3:
27     ; 第二层循环
28     mov edx,num1
29     mov eax,[edx+esi*4]
30     cmp edi,len2
31     jnb L1
32     ; 结果保存在EDX:EAX
33     mul DWORD PTR [ebx+edi*4]
34     add [ecx+edi*4],eax
35     sub edi,esi
36     inc edi
37     jmp L3
38 L1:
```

```
39     inc esi
40     jmp L2
41
42 numMul endp
```

3.3.4 进位处理

对 num3 数组中的每一个数判断是否大于 10，大于 10，则/10 向前进位，修正当前数值为%10 的结果。

流程如下：

1. 取出每一个元素值；
2. 判断是否大于 10；
 - (a) 大于 10，则修正当前元素值为%10 的结果，往前进位相应/10 的结果；
 - (b) 否则，继续执行；
3. 对最后一个值的时候要单独判断一下，如果往前进了位，那么 num3 的长度要相应加 1。

具体代码如下：

```
1 ; 进位处理
2 carry      proc      stdcall
3     numArray: ptr DWORD, numLen: ptr DWORD
4     mov esi, numArray
5     mov edi, numLen
6     xor ecx, ecx
7
8 L1:
9     cmp ecx, [edi]
10    jnb      L2
11    mov eax, [esi+ecx*4]
```

```
12      mov ebx,0AH
13      xor edx,edx
14      div ebx
15      cmp eax,0
16      jnz L3
17      inc ecx
18      jmp L1
19  L2:
20      mov eax,[esi+ecx*4]
21      cmp eax,0
22      jnz L4
23      ret
24  L3:
25      mov [esi+ecx*4],edx
26      inc ecx
27      add [esi+ecx*4],eax
28      jmp L1
29  L4:
30      mov eax,[edi]
31      add eax,1
32      mov [edi],eax
33      ret
34
35  carry endp
```

3.3.5 逆序输出

对每一个数组逆序输出，这里要注意是否有负号

具体代码如下：

```
1  printArray  proc      stdcall  numArray: ptr DWORD,numLen:DWORD
```

```
2      mov ecx ,numLen
3      mov esi ,numArray
4      cmp negNum,1
5      jz L3
6 L1:
7      mov eax ,[ esi ][ ecx*4-4]
8      pusha          ;调用C函数要保存寄存器现场
9      invoke printf , offset szNumprint ,eax
10     popa
11     loop L1
12     ret
13 L3:
14     pusha
15     invoke printf , offset szNegSign
16     popa
17     jmp L1
18
19 printArray endp
```


3.4 实验结果展示

3.4.1 输入正数

```
C:\D\大三下\汇编语言与接口技术\实验\大数相乘>bignummul.exe
Please input numbers:
1279138190283908219089038129083
3812731879013712098309128309128309183209831908
4877010955759364596247025437428335452594749432115190520553823161699236180364
C:\D\大三下\汇编语言与接口技术\实验\大数相乘>

C:\Windows\system32\cmd.exe - python
Microsoft Windows [版本 10.0.19041.867]
(c) 2020 Microsoft Corporation. 保留所有权利。

C:\Users\15827>python
Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 1279138190283908219089038129083*3812731879013712098309128309128309183209831908
4877010955759364596247025437428335452594749432115190520553823161699236180364
>>> _
```

图 3-1 验证 1

3.4.2 输入负数

```
C:\D\大三下\汇编语言与接口技术\实验\大数相乘>bignummul.exe
Please input numbers:
63831834654657644465764363371827731283271
-68321639817289371893198237198731937198273189731073
-4361095616152289305475931349822160632534691063078581308863410822273768989428418513673779783
C:\D\大三下\汇编语言与接口技术\实验\大数相乘>

C:\Windows\system32\cmd.exe - python
Microsoft Windows [版本 10.0.19041.867]
(c) 2020 Microsoft Corporation. 保留所有权利。

C:\Users\15827>python
Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 1279138190283908219089038129083*3812731879013712098309128309128309183209831908
4877010955759364596247025437428335452594749432115190520553823161699236180364
>>> 63831834654657644465764363371827731283271*-68321639817289371893198237198731937198273189731073
-4361095616152289305475931349822160632534691063078581308863410822273768989428418513673779783
>>> _
```

图 3-2 验证 2

第 4 章 文件对比

4.1 实验目的

利用 Windows 界面编程方法，实现输入两个文件，比对两个文件的内容是否相等，若相等，则弹出提示；否则指出有哪几行内容不相等

4.2 实验设计

界面设计显示：



图 4-1 界面设计

4.2.1 程序流程

1. 建立窗口显示界面，同时循环监听消息
2. 点击选择文件后，打开选择文件对话框，同时得到相应的文件路径名
3. 打开相应文件路径，读取文件内容，一行一行比对，得出结果
4. 得到结果后，弹出窗口显示

4.3 具体实现

该节介绍了代码中的部分核心算法和具体代码，包括了建立窗口，选择文件，文件对比，显示结果。

4.3.1 建立窗口

创建窗口，同时显示一系列信息到界面上。

流程如下：

1. 获得当前程序的句柄；
2. 注册窗口；
3. 建立并显示窗口；
4. 之后不断监听消息。

具体代码如下：

```
1  _WinMain                proc
2      local @stWndClass:WNDCLASSEX
3      local @stMsg:MSG
4
5      invoke GetModuleHandle, NULL
6      ; 获得当前程序的句柄
7      mov hInstance, eax
8      invoke RtlZeroMemory, addr @stWndClass, sizeof @stWndClass
9
10     ; 注册窗口
11     invoke LoadCursor, 0, IDC_ARROW
12     mov @stWndClass.hCursor, eax
13     push hInstance
14     pop @stWndClass.hInstance
15     mov @stWndClass.cbSize, sizeof WNDCLASSEX
```

```
16  mov  @stWndClass.style , CS_HREDRAW or CS_VREDRAW
17  mov  @stWndClass.lpfnWndProc , offset _ProcWinMain
18      ; 过程地址
19  mov  @stWndClass.hbrBackground , COLOR_WINDOW+1
20  mov  @stWndClass.lpszClassName , offset szClassName
21  invoke RegisterClassEx , addr @stWndClass
22
23      ; 建立并显示窗口
24  invoke CreateWindowEx , WS_EX_CLIENTEDGE, \
25                          offset szClassName , offset szCaptionMain , \
26                          WS_OVERLAPPEDWINDOW, \
27                          300, 200, 800, 400, \
28                          NULL, NULL, hInstance , NULL
29  mov  hWinMain , eax
30      ; 保存窗口句柄
31  invoke ShowWindow , hWinMain , SW_SHOWNORMAL
32      ; 显示窗口
33  invoke UpdateWindow , hWinMain
34      ; 绘制客户区
35
36      ; 消息循环
37  .while TRUE
38      invoke GetMessage , addr @stMsg , NULL, 0 , 0
39      .break .if eax == 0
40      ; 如果是WM_QUIT, 则结束循环
41      invoke TranslateMessage , addr @stMsg
42      invoke DispatchMessage , ADDR @stMsg
43      .endw
44      ret
45  _WinMain endp
```

4.3.2 窗口过程函数

循环监听消息，对于收到不同的消息，程序处理的函数不一样。

流程如下：

1. WM_PAINT: 表示绘制窗口的时候，需要执行的动作，展示窗口的一些提示信息；
2. WM_CREATED: 窗口接收到创建消息时，创建按钮和文本框；
3. WM_COMMAND: 根据 `eax` 的不同，分别执行不同的函数：
 - (a) `eax==1` 或 `eax==2`: 打开文本对话框，将选择文件路径保存到 `filepath` 中；
 - (b) `eax==3`: 进行文件比对，并弹出结果信息（注意输出行号的时候要转化成字符串形式）。

具体代码如下：

```
1 ;窗口句柄,消息标识,消息参数
2 _ProcWinMain proc uses ebx edi esi,
3     hWnd, uMsg, wParam, lParam
4     local @stPs:PAINTSTRUCT
5     local @stRect:RECT
6     local @hDc
7     local @temp:dword
8     local @buffer[1024]:byte
9     local @TextMSG[1024]:byte
10    mov eax,uMsg
11
12
13    .if eax==WM_PAINT
14        ;获取窗口设备环境句柄
```

```
15     invoke BeginPaint , hWnd, addr @stPs
16     mov @hDc, eax
17
18     ; 获取客户区大小
19     invoke GetClientRect ,hWnd,addr @stRect
20     invoke DrawText ,@hDc,addr szText ,-1,\
21         addr @stRect ,\
22         DT_CENTER
23     invoke      lstrlen , offset textout1
24     invoke TextOut ,@hDc, 200,65,offset textout1 ,eax
25     invoke      lstrlen , offset textout2
26     invoke TextOut ,@hDc, 200,145,offset textout2 ,eax
27     invoke EndPaint , hWnd, addr @stPs
28
29 ; 窗口关闭消息
30 .elseif eax==WM_CLOSE
31     invoke DestroyWindow , hWinMain
32     ; 发出WM_QUIT消息来退出循环
33     invoke PostQuitMessage , NULL
34
35
36 .elseif eax==WM_CREATE
37     invoke CreateWindowEx ,NULL,\
38         offset szbutton ,offset button1 ,\
39         WS_CHILD OR WS_VISIBLE ,\
40         50,60,120,35,\
41         hWnd,1 ,hInstance ,NULL
42
43     invoke CreateWindowEx ,NULL,\
44         offset szbutton ,offset button2 ,\
```

```
45         WS_CHILD OR WS_VISIBLE, \
46         50,140,120,35, \
47         hWnd,2,hInstance,NULL
48
49     invoke CreateWindowEx,NULL,\
50         offset szbutton,offset button3,\
51         WS_CHILD OR WS_VISIBLE,\
52         300,250,120,35,\
53         hWnd,3,hInstance,NULL
54
55     invoke CreateWindowEx,NULL,\
56         offset szedit,NULL,\
57         WS_CHILD OR WS_VISIBLE,\
58         300,65,400,35,\
59         hWnd,4,hInstance,NULL
60
61     invoke CreateWindowEx,NULL,\
62         offset szedit,NULL,\
63         WS_CHILD OR WS_VISIBLE,\
64         300,145,400,35,\
65         hWnd,5,hInstance,NULL
66
67     .elseif eax==WM_COMMAND
68         mov eax,wParam
69         .if eax == 1
70             invoke OpenFileDirectory,offset filePath1
71             invoke
72             SetDlgItemText,hWnd,4,offset filePath1
73         .elseif eax == 2
74             invoke OpenFileDirectory,offset filePath2
```

; 输出相应的文件

```
74         invoke SetDlgItemText , hWnd, 5, offset filePath2
75     .elseif eax == 3
76         invoke CompareFile , offset filePath1 , offset filePath2
77         .if lineCount==0
78             invoke MessageBoxA , NULL, offset output1 ,
79                 offset szBoxTitle , MB_OK+MB_ICONINFORMATION
80         .elseif
81             xor esi , esi
82             pusha
83             invoke crt_strcat , addr @TextMSG, offset output2
84             popa
85             .while esi < lineCount
86                 mov edi , lineSeqs[esi*4]
87                 mov @temp, edi
88                 pusha
89                 invoke crt_sprintf , addr @buffer, addr testout , @temp
90                 popa
91                 invoke crt_strcat , addr @TextMSG, addr @buffer
92                 inc esi
93             .endw
94
95             invoke MessageBoxA , NULL, addr @TextMSG,
96                 offset szBoxTitle , MB_OK+MB_ICONERROR
97         .endif
98     .endif
99
100 ; 其他默认一部分的消息
101 .else
102     invoke DefWindowProc , hWnd, uMsg, wParam , lParam
103     ret
```



```
104 .endif
105 xor eax,eax
106 ret
107
108 _ProcWinMain endp
```

4.3.3 文件对比

将得到数组中的每一个元素对应相乘再相加。

流程如下：

1. 打开文件，获得文件句柄；
2. 不断循环，直到到达文件的末尾为止；
3. 循环体内一行一行读取文件内容（`crt_fgets`），并进行比对；
 - (a) 先判断长度是否相同，不相同则一定不等
 - (b) 若长度相同，则进一步比对内容（`lstrcmp`），并保存结果

具体代码如下：

```
1 CompareFile proc stdcall , path1:ptr byte ,path2:ptr byte
2     local @pfile1:ptr FILE
3     local @pfile2:ptr FILE
4     local @readnum
5     local @temp1,@temp2
6     local @len1,@len2
7     local @idx
8
9     ; 打开文件
10    invoke crt_fopen ,path1 ,offset textOpenPermisson
11    mov @pfile1 ,eax
```

```
12
13     invoke crt_fopen , path2 , offset textOpenPermisson
14     mov @pfile2 , eax
15
16     mov @idx , 0
17
18     .while TRUE
19
20         inc @idx
21         mov @len1 , 0
22         mov @len2 , 0
23         mov @templ , 1
24         mov @temp2 , 1
25
26         ; 初始化缓冲区
27         invoke crt_memset , addr data1 , 0 , sizeof data1
28         invoke crt_memset , addr data2 , 0 , sizeof data2
29
30         invoke crt_feof , @pfile1
31         mov @templ , eax
32         invoke crt_feof , @pfile2
33         mov @temp2 , eax
34
35         mov ecx , @templ
36         mov edx , @temp2
37         .break .if ecx && edx
38
39         ; 说明没有到文件末尾
40         .if !@templ
41             invoke crt_fgets , addr data1 , 1024 , @pfile1
```

```
42         invoke lstrlen ,addr data1
43         mov @len1 ,eax
44     .endif
45
46     .if !@temp2
47         invoke crt_fgets ,addr data2 ,1024 ,@pfile2
48         invoke lstrlen ,addr data2
49         mov @len2 ,eax
50     .endif
51
52
53     mov eax ,@len1
54     mov ebx ,@len2
55     .if eax!=ebx
56         ; 长度不等当然不一样
57         mov ecx ,lineCount
58         mov ebx ,@idx
59         lea esi ,lineSeqs
60         mov [ esi+ecx*4] ,ebx
61         inc lineCount
62         .continue
63     .endif
64
65     invoke lsticmp ,addr data1 ,addr data2
66     mov @templ ,eax
67
68     .if eax
69         mov ecx ,lineCount
70         mov ebx ,@idx
71         lea esi ,lineSeqs
```

```
72             mov [esi+ecx*4],ebx
73             inc lineCount
74         .endif
75
76     .endw
77     ret
78
79 CompareFile endp
```

4.4 实验结果展示

4.4.1 两个文件相同

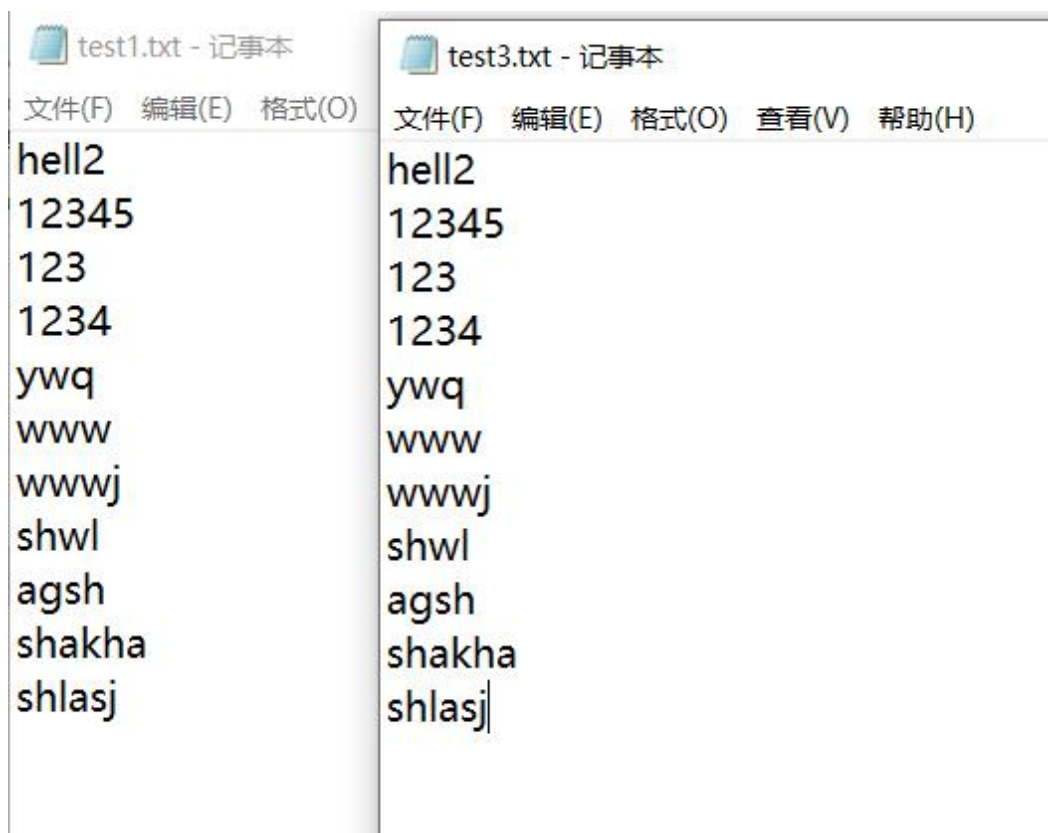


图 4-2 文件内容相同



图 4-3 文件比对结果

4.4.2 两个文件内容不同

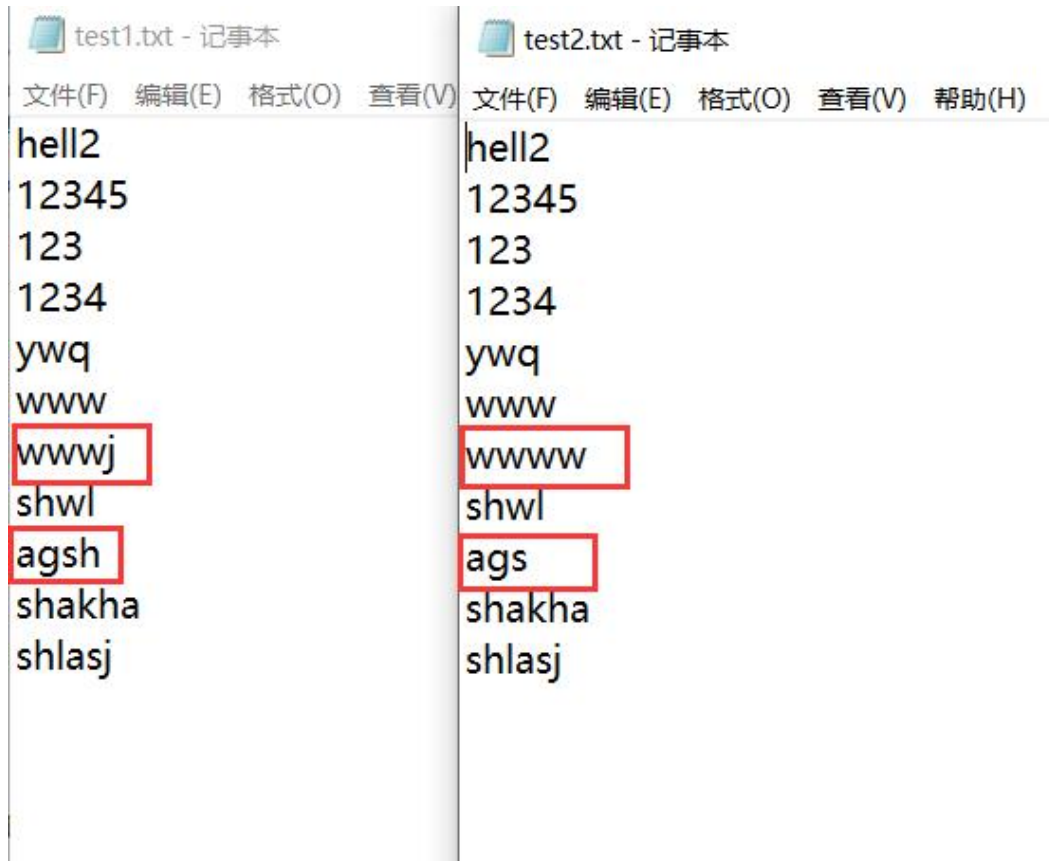


图 4-4 文件内容不同

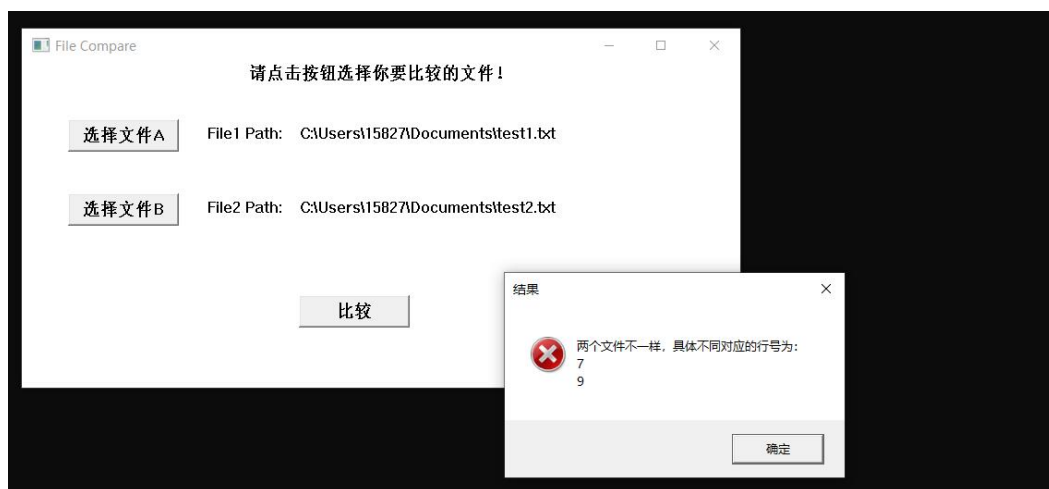


图 4-5 文件比对结果

第 5 章 多重循环分析

5.1 实验目的

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。

5.2 程序流程

1. 编写四重循环的 C 语言代码；
2. 反汇编分析语句功能；
3. 重写反汇编代码。

5.3 具体实现

该节部分主要介绍了多重循环的 C 代码反汇编的分析以及代码的重写。

5.3.1 多重循环的 C 语言代码

设置四重循环，选取 a 数组中的四个数相加输出结果，其中为了方便显示不同循环次数，对每一次循环的边界值设定不一致。

具体代码如下：

```
1 #include <stdio.h>
2 int a[10] = { 1,2,3,4,5,6,7,8,9 };
3 int main()
4 {
5     for (int i = 0; i < 2; i++)
6     {
7         for (int j = 0; j < 3; j++)
8         {
9             for (int k = 0; k < 4; k++)
```

```
10      {
11          for (int m = 0; m < 5; m++)
12          {
13              int sum = a[i] + a[j] + a[k] + a[m];
14              printf("%d\n", sum);
15          }
16      }
17  }
18  }
19  return 0;
20 }
```

5.3.2 反汇编分析

点击调试后，查看反汇编得到汇编代码，并对语句进行分析。

分析过程如下：

初始化

```
1      00C61860  push        ebp
2      00C61861  mov         ebp,esp
3      00C61863  sub         esp,0FCh
4      00C61869  push        ebx
5      00C6186A  push        esi
6      00C6186B  push        edi
7      00C6186C  lea         edi,[ebp-0FCh]
8      00C61872  mov         ecx,3Fh
9      00C61877  mov         eax,0CCCCCCCCh
10     00C6187C  rep stos    dword ptr es:[edi]
11     00C6187E  mov         ecx,offset _B5E6F96D_多重循环@cpp (0C6C003h)
```



```

12      00C61883  call
      @__CheckForDebuggerJustMyCode@4 (0C61316h)

```

1. **push ebp:** 将 ebp 压栈，保存调用函数的栈指针。
2. **mov ebp,esp:** 把 esp 的值赋给 ebp，也就是 ebp 指向栈基址
3. **sub esp,0FCh:** 为当前函数开辟局部变量空间
4. **push ebx**
5. **push esi**
6. **push edi:** 将 ebx、esi、edi 的值保存下来（压入栈中）
7. **lea edi,[ebp-0FCh]:** 取出栈中局部变量区域（大小为 0FCh=252 的那一块）的最低地址
8. **mov ecx,3Fh:** 把 63 赋给 ecx
9. **mov eax,0CCCCCCCCh:** 把 0CCCCCCCCh 赋给 eax
10. **rep stos dword ptr es:[edi]:** 将局部变量区域全部初始化为 0CCCCCCCCh（这里 $63*4=252$ ）

内部循环体

```

1      for (int i = 0; i < 2; i++)
2      ; i=0
3      00C61888  mov          dword ptr [ebp-8],0
4      00C6188F  jmp         main+3Ah (0C6189Ah)
5      00C61891  mov          eax,dword ptr [ebp-8]
6      00C61894  add          eax,1
7      00C61897  mov          dword ptr [ebp-8],eax
8      ; 判断 i 是否 <2
9      00C6189A  cmp          dword ptr [ebp-8],2
10     ; 如果不小于，跳出循环
11     00C6189E  jge          main+0D6h (0C61936h)
12     {
13         for (int j = 0; j < 3; j++)

```

```
14 ;j=0
15 00C618A4 mov dword ptr [ebp-14h],0
16 00C618AB jmp main+56h (0C618B6h)
17 00C618AD mov eax,dword ptr [ebp-14h]
18 00C618B0 add eax,1
19 00C618B3 mov dword ptr [ebp-14h],eax
20 ;j<3?
21 00C618B6 cmp dword ptr [ebp-14h],3
22 ;跳到i的循环体里
23 00C618BA jge main+0D1h (0C61931h)
24 {
25     for (int k = 0; k < 4; k++)
26 ;k=0
27 00C618BC mov dword ptr [ebp-20h],0
28 00C618C3 jmp main+6Eh (0C618CEh)
29 00C618C5 mov eax,dword ptr [ebp-20h]
30 00C618C8 add eax,1
31 00C618CB mov dword ptr [ebp-20h],eax
32 ;k<4?
33 00C618CE cmp dword ptr [ebp-20h],4
34 跳到j的循环里
35 00C618D2 jge main+0CCh (0C6192Ch)
36 {
37     for (int m = 0; m < 5; m++)
38 00C618D4 mov dword ptr [ebp-2Ch],0
39 00C618DB jmp main+86h (0C618E6h)
40 00C618DD mov eax,dword ptr [ebp-2Ch]
41 00C618E0 add eax,1
42 00C618E3 mov dword ptr [ebp-2Ch],eax
43 00C618E6 cmp dword ptr [ebp-2Ch],5
```

```

44 ; 跳到k的循环里
45 00C618EA    jge          main+0CAh (0C6192Ah)
46             {
47 int sum = a[i] + a[j] + a[k] + a[m];
48 ; 取出a数组对应的值
49 00C618EC    mov          eax,dword ptr [ebp-8]
50 00C618EF    mov          ecx,dword ptr a (0C6A000h)[eax*4]
51 00C618F6    mov          edx,dword ptr [ebp-14h]
52 00C618F9    add          ecx,dword ptr a (0C6A000h)[edx*4]
53 00C61900    mov          eax,dword ptr [ebp-20h]
54 00C61903    add          ecx,dword ptr a (0C6A000h)[eax*4]
55 00C6190A    mov          edx,dword ptr [ebp-2Ch]
56 00C6190D    add          ecx,dword ptr a (0C6A000h)[edx*4]
57 00C61914    mov          dword ptr [ebp-38h],ecx
58                                     printf("%d\n", sum);
59 00C61917    mov          eax,dword ptr [ebp-38h]
60 00C6191A    push         eax
61 00C6191B    push         offset string "%d\n" (0C67B30h)
62 00C61920    call         _printf (0C610CDh)
63 ; 平衡堆栈
64 00C61925    add          esp,8
65             }
66 00C61928    jmp          main+7Dh (0C618DDh)
67             }
68 00C6192A    jmp          main+65h (0C618C5h)
69             }
70 00C6192C    jmp          main+4Dh (0C618ADh)
71             }
72 00C61931    jmp          main+31h (0C61891h)
73             return 0;

```

```

74 00C61936 xor     eax, eax
75 }
    
```

根据上图展示的汇编代码以及相关注释，可以得到多重循环的逻辑结构。

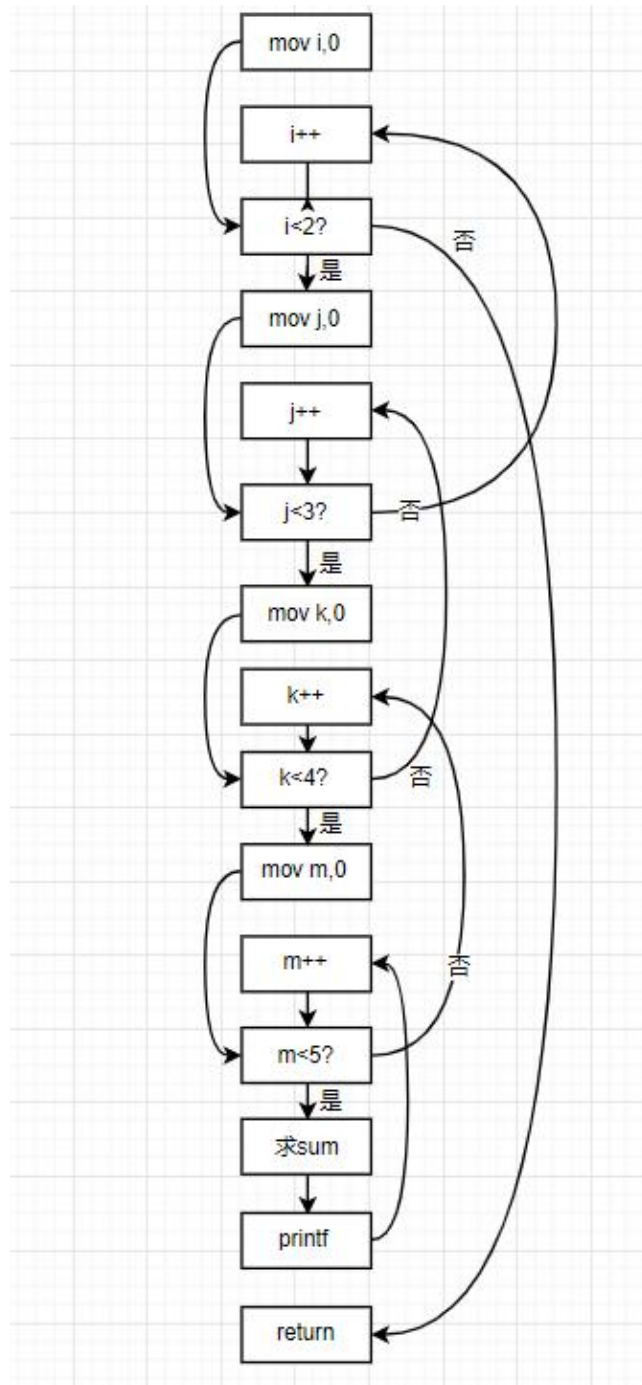


图 5-1 多重循环分析

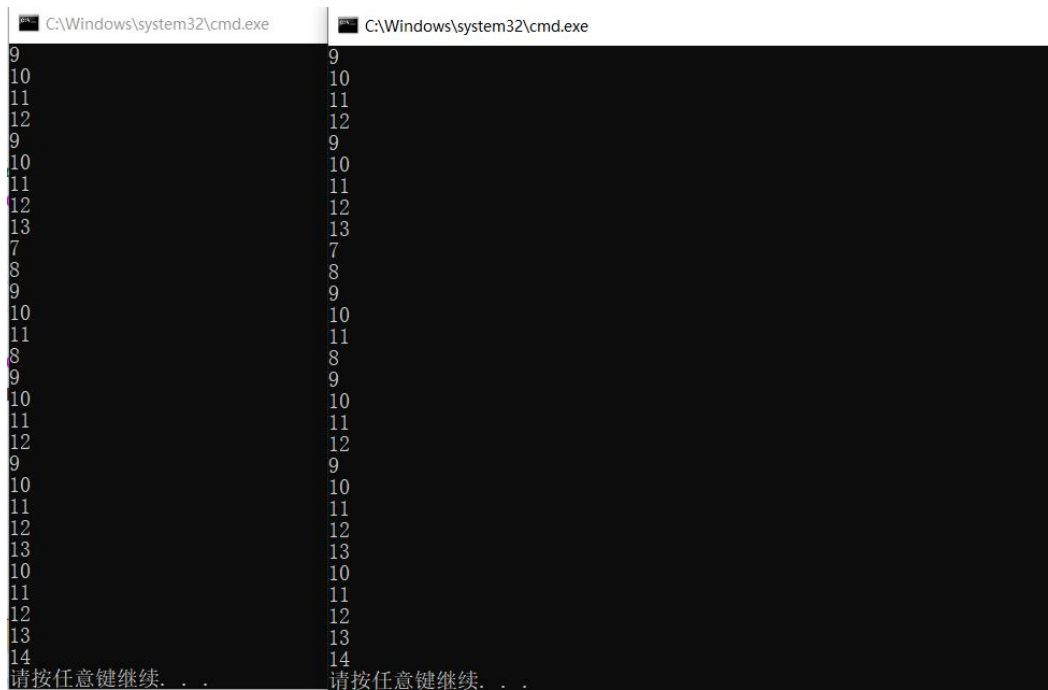
5.3.3 重写多重循环

根据上述的代码逻辑结构，重写汇编程序如下：

```
1  main  proc
2      local  @i:dword ,@j:dword ,@k:dword ,@m:dword ,@sum:dword
3      mov  @i,0
4      jmp  L1
5  L4:
6      add  @i,1
7  L1:
8      cmp  @i,2
9      jge  L2
10     mov  @j,0
11     jmp  L3
12  L6:
13     add  @j,1
14  L3:
15     cmp  @j,3
16     jge  L4
17     mov  @k,0
18     jmp  L5
19  L8:
20     add  @k,1
21  L5:
22     cmp  @k,4
23     jge  L6
24     mov  @m,0
25     jmp  L7
26  L9:
27     add  @m,1
```

```
28 L7:
29     cmp @m,5
30     jge L8
31     ; 计算sum
32     mov eax,@i
33     mov ecx,a[eax*4]
34     mov eax,@j
35     add ecx,a[eax*4]
36     mov eax,@k
37     add ecx,a[eax*4]
38     mov eax,@m
39     add ecx,a[eax*4]
40     mov @sum,ecx
41     ; 输出结果
42     invoke printf,offset printStrlen,@sum
43     jmp L9
44 L2:
45     ret
46
47 main endp
48 end main
```

5.4 实验结果展示



The image displays two side-by-side Windows command prompt windows, both titled "C:\Windows\system32\cmd.exe". Each window shows a vertical list of numbers: 9, 10, 11, 12, 9, 10, 11, 12, 13, 7, 8, 9, 10, 11, 8, 9, 10, 11, 12, 9, 10, 11, 12, 13, 10, 11, 12, 13, 14. At the bottom of each window, the text "请按任意键继续. . ." is visible. The numbers are displayed in a light blue color on a black background.

图 5-2 多重循环结果

总 结

在该实验中，我采用 MASM32 和 VS2019 完成实验了大数相乘、文件比对和 C 语言多重循环分析这三个实验，总体上难度不大，我认为相比较麻烦一点的实验是文件比对，因为要用 windows 界面编程，我对里面的一些 API 不是很熟悉，所以我采用了模块编程，先实现界面功能，再是文件比对，最后进行整合，这样一来，bug 的数量就会减少一些。

这门课的实验偏向于底层，虽然和现在流行的编程语言有些脱离，但是靠近底层能帮助我们更好的理解计算机的运行机制，为程序的进一步优化也做好了铺垫。

感谢老师这一学期的辛苦讲授，我受益良多。