### 

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实验**

**实验名称： 实验五 WIN32编程**

**实验时间： 2017-05-14，14：30-18：00 实验地点： 南一楼804室90号实验台**

**指导教师： 李海波**

**专业班级：校交 201601班**

**学 号： U201612696 姓 名： 陈淏睿**

**同组学生： 无 报告日期： 2018年 5 月 15日**

**原创性声明**

  本人郑重声明：本报告的内容由本人独立完成，有关观点、方法、数据和文献等的引用已经在文中指出。除文中已经注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品或成果，不存在剽窃、抄袭行为。

特此声明！

学生签名：

日期：2017.5.15

成绩评定

|  |  |  |
| --- | --- | --- |
| 实验完成质量得分（70分）（实验步骤清晰详细深入，实验记录真实完整等） | 报告撰写质量得分（30分）（报告规范、完整、通顺、详实等） | 总成绩（100分） |
|  |  |  |

指导教师签字：

                     日 期：

**目录**

[1 实验目的与要求 2](#_Toc515200104)

[2 实验内容 2](#_Toc515200108)

[3 实验过程 3](#_Toc515200112)

[3.1 任务1 3](#_Toc515200113)

[3.1.1 设计思想及存储单元分配 3](#_Toc515200114)

[3.1.2 流程图 3](#_Toc515200115)

[3.1.3 源程序 4](#_Toc515200116)

[3.1.4 实验步骤 5](#_Toc515200117)

[3.1.5 实验记录与分析 5](#_Toc515200118)

[4 总结与体会 10](#_Toc515200119)

[参考文献 15](#_Toc515200120)

# 实验目的与要求

本次实验的主要目的与要求有下面3点，所有的任务都会围绕这3点进行，希望大家事后检查自己是否达到这些目的与要求。

1. 熟悉WIN32程序的设计和调试方法；
2. 熟悉宏汇编语言中INVOKE、结构变量、简化段定义等功能；
3. 进一步理解机器语言、汇编语言、高级语言之间以及实方式、保护方式之间的一些关系。

# 实验内容

编写一个基于窗口的WIN32程序，实现**网店商品信息管理程序**的平均利润率计算及商品信息显示的功能（借鉴实验三的一些做法），具体要求如下描述。

功能一：编写一个基于窗口的WIN32程序的菜单框架，具有以下的下拉菜单项：

File Action Help

Exit Average About

List

点菜单File下的Exit选项时结束程序；点菜单Help下的选项About，弹出一个消息框，显示本人信息，类似图2.1所示。点菜单Action下的选项Average、List将分别实现计算平均利润率或显示SHOP1所有商品信息的功能（详见功能二的描述）。



图2.1 菜单示例

功能二：要求采用结构变量存放商品的相关信息。商品数至少定义5种。

点菜单项Average时，按照实验三的方法计算所有商品的平均利润率。用TD32观察计算结果。

点菜单项List时，要求能在窗口中列出SHOP1的所有商品的信息。具体显示格式自行定义，可以参照图2.2的样式（不要求用中文）。



图2.2 商品信息显示示意图

# 实验过程

## 任务1

### 设计思想及存储单元分配

菜单中共有三个主目录，分别为File,Action与Help.其中File下有Exit功能，选中后退出程序；Action下有Average和List两个功能，选Average计算所有商品的平均利润率，选List在窗口中列出SHOP1的所有商品的信息;Help下有About功能，选中后显示开发者信息。

1. 存储单元分配

先定义cmodty商品结构，其中包含长为10的字节串goodname（商品名称），dw型in\_price（进货价），dw型out\_price（售价），dw型in\_num（进货数量），dw型out\_num（已售数量）和dw型interest（当前利润）。

SHOP1，SHOP2：两长为5的cmodty结构数组，两数组中商品相同。

OLD\_INT：双字变量，前两个字节存储旧IP, 后两个字节存储旧CS。

hInstance，CommandLine：双字类型变量，值均为0，作为API函数的参数。

ClassName，AppName，MenuName，DlgName，AboutMsg：字节串，保存在窗口标题或对话框中信息。

msg\_name，msg\_iprice，msg\_oprice，msg\_inum，msg\_onum，msg\_interest：字节串，存储。

其它变量已在实验三中说明。

2.寄存器分配

各函数中对寄存器的使用情况均不相同。

### 流程图，

图3.1.1是任务1 Win32窗口程序的流程图。

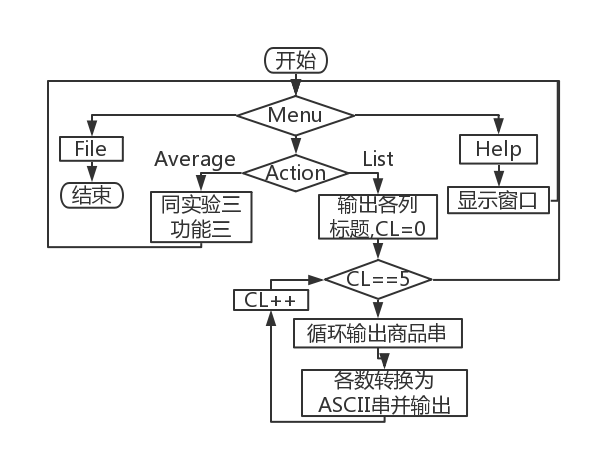


图3.1.1 Win32窗口程序流程图

### 源程序

MenuID.inc

IDM\_FILE\_EXIT equ 10001

IDM\_ACTION\_LIST equ 10101

IDM\_ACTION\_AVERAGE equ 10102

IDM\_HELP\_ABOUT equ 10201

T1.rc

#define IDM\_FILE\_EXIT 10001

#define IDM\_ACTION\_LIST 10101

#define IDM\_ACTION\_AVERAGE 10102

#define IDM\_HELP\_ABOUT 10201

MyMenu MENU

BEGIN

POPUP "&File"

BEGIN

MENUITEM "E&xit",IDM\_FILE\_EXIT

END

POPUP "&Action"

BEGIN

MENUITEM "A&verage",IDM\_ACTION\_AVERAGE

MENUITEM "L&ist",IDM\_ACTION\_LIST

END

POPUP "&Help"

BEGIN

MENUITEM "A&bout",IDM\_HELP\_ABOUT

END

END

t1.asm

.386

.model flat,stdcall

option casemap:none

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD

WndProc proto :DWORD,:DWORD,:DWORD,:DWORD

Display proto :DWORD

PART3 PROTO

include menuID.INC

include windows.inc

include user32.inc

include kernel32.inc

include gdi32.inc

include shell32.inc

includelib user32.lib

includelib kernel32.lib

includelib gdi32.lib

includelib shell32.lib

cmodty struct

goodname db 10 dup(0)

in\_price dw 0

out\_price dw 0

in\_num dw 0

out\_num dw 0

interest dw 0

cmodty ends

GETY MACRO

INC ECX

IMUL ECX,YY\_GAP

ADD ECX,YY

ENDM

IO MACRO ADS,NM

LEA DX,ADS

MOV AH,NM

INT 21H

ENDM

INTR MACRO T ;参数为寄存器名的一部分

MOV SI,DS:[E&T&+10] ;进货价

IMUL SI,DS:[E&T&+14] ;进货数量

MOV AX,DS:[E&T&+12] ;销售价

IMUL AX,DS:[E&T&+16] ;已售数量

SUB AX,SI

IMUL HKU

IDIV SI

ENDM

FND MACRO COUT,PIN,A,B,N ;参数为成功时的返回标号

MOV SI,0

LP&A&: ;获得商店一中当前商品串长，存入AX(AL)

CMP BYTE PTR DS:[EDI+ESI],0

JE BRK&N&

INC SI

JMP LP&A&

BRK&N&:

MOV AX,SI

MOV CH,M ;CH负责商店二的遍历

LEA EBX,PIN

LP&B&:

CALL CMPST ;比较两商品串是否相同,不同时AH为0（默认值）

OR AH,AH

JNZ COUT

ADD EBX,20 ;EBX移向商店二中下一商品，继续比较

DEC CH

JNZ LP&B&

ENDM

.data

ClassName db 'TryWinClass',0

AppName db 'Our First Window',0

MenuName db 'MyMenu',0

DlgName db 'MyDialog',0

AboutMsg db 'I am CSIE1601 CHR',0

hInstance dd 0

CommandLine dd 0

SHOP1 cmodty <'PEN',35,56,70,45,?>

cmodty <'BOOK',12,30,25,5,?>

cmodty <'BAG',25,35,20,12,?>

cmodty <'CPU',20,40,40,28,?>

cmodty <'BED',30,55,50,35,?>

SHOP2 cmodty <'PEN',35,50,30,24,?>

cmodty <'BOOK',12,28,20,15,?>

cmodty <'BAG',20,35,42,35,?>

cmodty <'CPU',10,35,30,12,?>

cmodty <'BED',20,45,60,30,?>

msg\_name db 'commodity',0

msg\_iprice db 'in\_price',0

msg\_oprice db 'out\_price',0

msg\_inum db 'in\_num',0

msg\_onum db 'out\_num',0

msg\_interest db 'interest',0

TIP\_12 DB ' $'

TIP\_9 DB 0AH,0DH,'$' ;换行，回车

PR1 DW ? ;商店一中的利润地址

M DB 5 ;有效商品个数

TAD DB 90 DUP(0) ;存放排名与偏移地址的临时数组

TNM DB 4 DUP(0) ;存放十进制数的临时数组

HKU DW 100

.code

Start: invoke GetModuleHandle,NULL

mov hInstance,eax

invoke GetCommandLine

mov CommandLine,eax

invoke WinMain,hInstance,NULL,CommandLine,SW\_SHOWDEFAULT

invoke ExitProcess,eax

;;

WinMain proc hInst:DWORD,hPrevInst:DWORD,CmdLine:DWORD,CmdShow:DWORD

LOCAL wc:WNDCLASSEX

LOCAL msg:MSG

LOCAL hWnd:HWND

invoke RtlZeroMemory,addr wc,sizeof wc

mov wc.cbSize,SIZEOF WNDCLASSEX

mov wc.style, CS\_HREDRAW or CS\_VREDRAW

mov wc.lpfnWndProc, offset WndProc

mov wc.cbClsExtra,NULL

mov wc.cbWndExtra,NULL

push hInst

pop wc.hInstance

mov wc.hbrBackground,COLOR\_WINDOW+1

mov wc.lpszMenuName, offset MenuName

mov wc.lpszClassName,offset ClassName

invoke LoadIcon,NULL,IDI\_APPLICATION

mov wc.hIcon,eax

mov wc.hIconSm,0

invoke LoadCursor,NULL,IDC\_ARROW

mov wc.hCursor,eax

invoke RegisterClassEx, addr wc

INVOKE CreateWindowEx,NULL,addr ClassName,addr AppName,\

WS\_OVERLAPPEDWINDOW,CW\_USEDEFAULT,\

CW\_USEDEFAULT,CW\_USEDEFAULT,CW\_USEDEFAULT,NULL,NULL,\

hInst,NULL

mov hWnd,eax

INVOKE ShowWindow,hWnd,SW\_SHOWNORMAL

INVOKE UpdateWindow,hWnd

;;

MsgLoop: INVOKE GetMessage,addr msg,NULL,0,0

cmp EAX,0

je ExitLoop

INVOKE TranslateMessage,addr msg

INVOKE DispatchMessage,addr msg

jmp MsgLoop

ExitLoop: mov eax,msg.wParam

ret

WinMain endp

WndProc proc hWnd:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

LOCAL hdc:HDC

.IF uMsg == WM\_DESTROY

invoke PostQuitMessage,NULL

.ELSEIF uMsg == WM\_KEYDOWN

.IF wParam == VK\_F1

;;your code

.ENDIF

.ELSEIF uMsg == WM\_COMMAND

.IF wParam == IDM\_FILE\_EXIT

invoke SendMessage,hWnd,WM\_CLOSE,0,0

.ELSEIF wParam == IDM\_ACTION\_LIST

invoke Display,hWnd

.ELSEIF wParam == IDM\_ACTION\_AVERAGE

invoke PART3

.ELSEIF wParam == IDM\_HELP\_ABOUT

invoke MessageBox,hWnd,addr AboutMsg,addr AppName,0

.ENDIF

; .ELSEIF uMsg == WM\_PAINT

;;redraw window again

.ELSE

invoke DefWindowProc,hWnd,uMsg,wParam,lParam

ret

.ENDIF

xor eax,eax

ret

WndProc endp

Display proc hWnd:DWORD

XX equ 10

YY equ 10

XX\_GAP equ 100

YY\_GAP equ 30

LOCAL hdc:HDC

invoke GetDC,hWnd

mov hdc,eax

invoke TextOut,hdc,XX+0\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_name,9

invoke TextOut,hdc,XX+1\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_iprice,8

invoke TextOut,hdc,XX+2\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_oprice,9

invoke TextOut,hdc,XX+3\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_inum,6

invoke TextOut,hdc,XX+4\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_onum,7

invoke TextOut,hdc,XX+5\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_interest,8

XOR ECX,ECX ;CL负责商店一的遍历

LEA EDI,SHOP1

LPI:

XOR ESI,ESI

LPI\_A: ;SI给出商品串长

MOV DL,[EDI+ESI]

CMP DL,0

JE BRK1

INC ESI

JMP LPI\_A

BRK1:

PUSHAD

GETY

invoke TextOut,hdc,XX+0\*XX\_GAP,ECX,EDI,ESI

POPAD

ADD EDI,8

MOV ESI,1

LPI\_B:

MOV AX,DS:[EDI+2\*ESI]

MOV EBX,10

CALL MRADIX ;转换为十进制并输出ASCII字符

PUSHAD

IMUL ESI,XX\_GAP

ADD ESI,XX

GETY

PUSH EDX

LEA EDX,TNM

invoke TextOut,hdc,ESI,ECX,EDX,EBX

POP EDX

POPAD

INC ESI

CMP ESI,6

JNE LPI\_B

INC CX

CMP CX,5

JZ BRK2 ;商品未遍历完

ADD EDI,12

JMP LPI

BRK2:

ret

Display endp

MRADIX PROC USES CX EDI

XOR ECX,ECX

LEA EDI,TNM

LPQ:

XOR DX,DX

DIV BX

PUSH DX

INC CX ;保存位数

OR AX,AX

JNZ LPQ

MOV EBX,ECX

LPR:

POP AX

CMP AL,10

JB BRK23

ADD AL,7

BRK23:

ADD AL,30H

MOV [EDI],AL ;当前字符存入临时串

INC EDI

LOOP LPR

RET

MRADIX ENDP

CMPST PROC ;比较字符串函数

MOV SI,0

LPE:

MOV DH,DS:[EDI+ESI]

CMP DH,DS:[EBX+ESI]

JNE BRK9

INC SI

CMP SI,AX

JNE LPE

INC SI

CMP BYTE PTR DS:[EBX+ESI],0 ;判断BX是否同样遍历完成

JNE BRK9

INC AH ;字符串相同

BRK9:

RET

CMPST ENDP

PART3 PROC USES AX CX SI BP

MOV CL,M ;CL负责商店一的遍历

LEA EDI,SHOP1

LPC: ;之前所有寄存器的值可作废

FND BRK10,SHOP2,A,B,11 ;循环比较BP与BX指向的串，相同时转到BRK11

BRK10: ;更新各商品均利润率（CL,EDI,EBX被占用）

INTR DI ;计算EDI所指向的商品的利润率

MOV DS:[EDI+18],AX ;保存EDI利润

INTR BX ;计算EBX所指向的商品的利润率

ADD AX,DS:[EDI+18] ;两利润相加

CWD

SAR AX,1 ;求平均利润

MOV DS:[EDI+18],AX ;平均利润存入SHOP1的利润字段

DEC CL

OR CL,CL

JZ BAR5

ADD EDI,20 ;BP移向商店一中下一商品

JMP LPC

BAR5:

RET

PART3 ENDP

end Start

### 实验步骤

1. 安装MASM32，准备上机实验环境。

2.以样例窗口程序框架为基础，修改menuID.inc,t1.rc与t1.asm中相关参数，修改菜单成为图2.1所示。

3. 以上一步的框架为基础，抽出实验三的功能三与功能五并修改，功能三对应Average功能，功能五对应List功能，分别被Action下的Average与List调用，同时修改t1.asm中相关调用参数。不断重复汇编，连接过程直至成功生成可执行文件。

4. 用TD32单步调试直至调用List时可输出正确结果。

### 实验记录与分析

1. 安装MASM32,准备上机实验环境。

2. 在样例程序基础之上修改源文件中的变量和.rc资源文件中的菜单显示方式，请教同学后知道了rc文件中目录传中插入的‘&’是用于激活Alt快捷键调用，该符号应加在将与Alt搭配的字母之前且不区分大小写。编译并连接，发现报错如图3.1.2所示。翻书后发现.inc中的头文件尚未跟着修改。

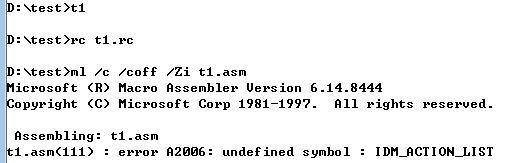


图3.1.2汇编报错

3.修改.inc头文件头文件中的常量定义后，仿照示例文件cbox将var.bat与nmake.exe放入源文件目录，直接用var与nmake汇编连接，发现报错如图3.1.3所示。据此我明白makefile是汇编连接时的配置文件而非汇编连接后生成的文件。将makefile放入目录，修改目标文件名后再次尝试编译连接，过程中无异常出现。

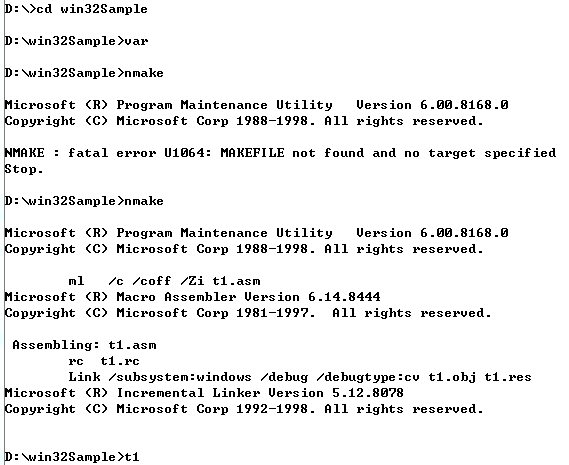


图3.1.3用var于nmake尝试编译连接

4. 进入t1，运行About并观察Action下子目录，结果如图3.1.4与3.1.5所示，证明目录修改成功。

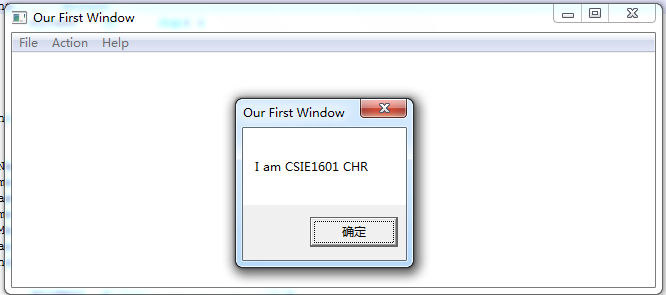


图3.1.4运行About

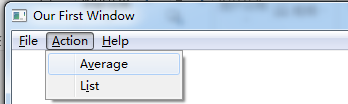


图3.1.5观察Action下子目录

5. 将实验三的功能三与功能五按照实验步骤中的思路改编后分别被Average和List调用，编译连接中无错误，但在执行Average与List时均崩溃。用TD32单步调试功能三，发现执行至ret时内存如图3.1.6所示，即五个利润均计算正确，但返回WndProc函数后报错崩溃，且提示的报错地址与ebp当前地址相同，如图3.1.7所示。猜想ebp与函数的调用有关，故在自己的程序中应尽量避免使用ebp。

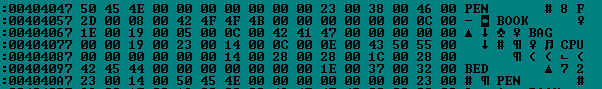


图3.1.6数据计算正常

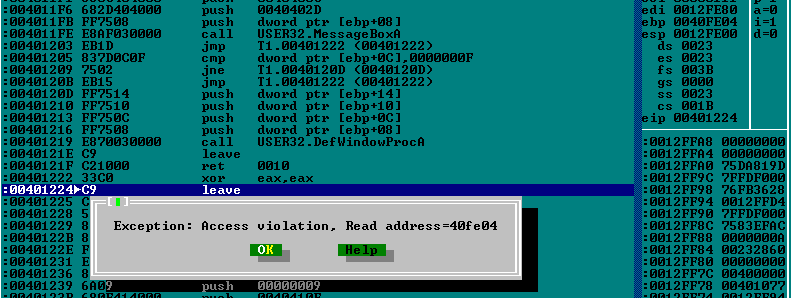


图3.1.7返回后崩溃，猜想ebp不能用

6. 将ebp均用edi代替，重新编译连接，发现Average功能执行正常，但List程序仍崩溃。单步调试后发现在系统功能调用处报错，如图3.1.8所示。说明Win32编程中不再适合用系统功能调用进行输出，应改用Windows API函数。

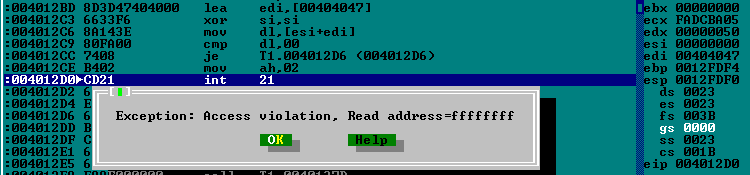


图3.1.8中断处理程序无法使用

7. 重新调试时发现在如图3.1.9所示位置报错，因为出错位置并非我写的语句且出错时窗口中并无输出，我以为出错位置在TextOut函数里，因此完全没有头绪，在此处向老师请求了帮助。下面是解决过程（参考图3.1.10）：F8单步执行过：004012E0处的API调用，发现此时并未报错。由此证明TextOut函数并未出错，或者说此处invoke函数的参数配置并未出错。继续向下调试，可以看到在：00401319处的TextOut调用前的push中间出现了不同的代码，0040401312处进行了一次对16位寄存器的push，由第二次对TextOut的调用的参数对比，发现此处是对存储器变量直接操作，因此尝试将这个存储器变量存入寄存器再作为参数。

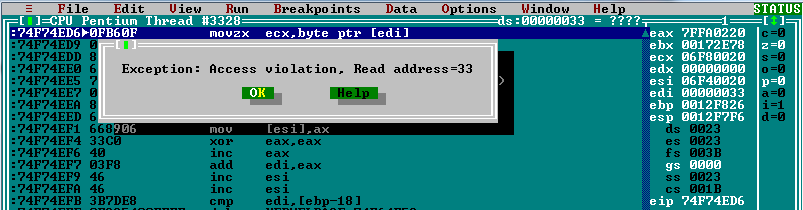


图3.1.9中断处理程序无法使用

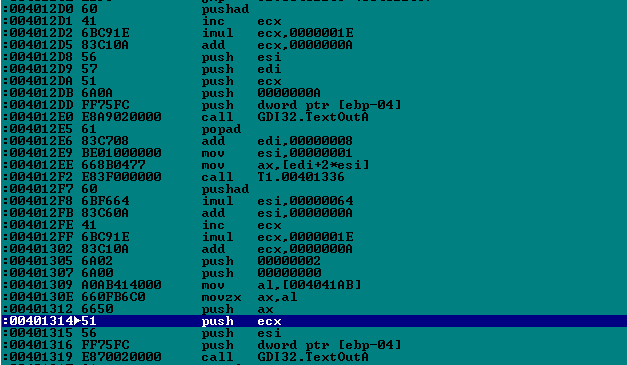


图3.1.10对16位寄存器的push造成了干扰

8. 再次汇编连接并运行，发现输出成功，如图3.1.11所示，但对利润率的计算仍有问题。

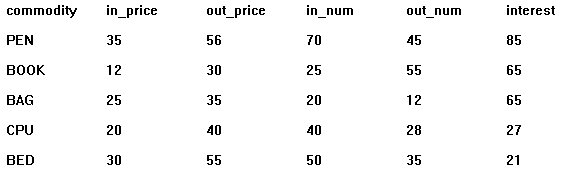


图3.1.11数据书写错误

9.TD32单步检查后发现前面讲EBP改为EDI时误操作，在CMPST函数中将EBX也改成了EDI，导致比较字符串时变成了和自身比较，EBX始终停留在商店二第一个商品处，导致利润率计算错误。改正后结果如图3.1.12所示。另外，在Winproc函数中注释掉.ELSEIF uMsg == WM\_PAINT，即不再对此种情况进行判断，即可在拖动窗口时保留当前显示。

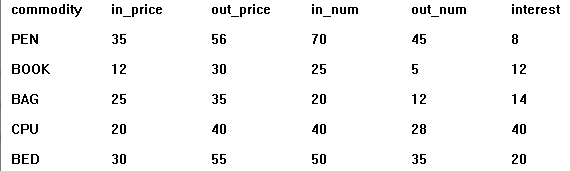


图3.1.12利润率恢复正常

# 总结与体会

本次实验使我第一次接触到了Win 32下的汇编编程。通过本次试验，我对Win 32汇编的环境，编译连接生成可执行文件的方法以及API函数尤其是TextoutA函数的使用有了一些了解，对32位下段的定义增强了熟悉，知道了32位下不再允许使用中断处理程序，对寄存器（比如EBP）的使用有了限制，熟悉了.inc文件，.rc文件和.asm文件的调用关系。

在观察老师帮忙调试程序的过程中，我得到的经验有两点。一是有的时候我们由于知识所限很难知道出现该问题的原因是什么，但这并不妨碍我们通过尝试虽然麻烦一点但一定可行的道路来调通程序，就比如这次将TNM的地址先赋给32位寄存器再将寄存器作为参数。在16位环境下这样可能显得多余；但在32位环境下这却是必要操作。至于为什么把存储器变量设为TextoutA的参数时会产生5步看不出头绪的操作则需要进一步学习了。二是在单步调试的时候面对陌生的，重复出现的代码可以尝试相互对比的方法，比如TextoutA输出商品名的时候一切正常，但输出之后的数时就有问题，在invoke内部同样是黑箱的情况下，参数很有可能出在实参的配置上。这时就可重点对比参数的传递过程，从而更有可能定位问题所在。

思考题（实验步骤中未涉及的）：

1.由图4.1和4.2显示Masm32下的程序清单。

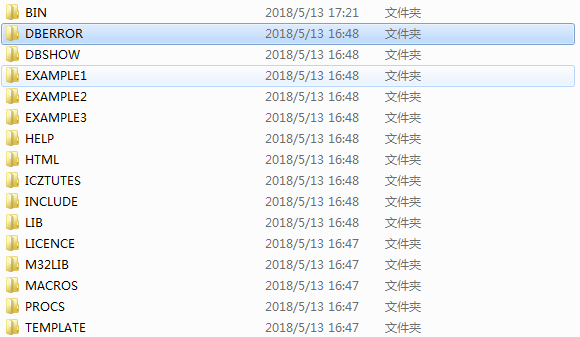




图4.1&图4.2内存驻留程序嵌套

2.由图4.2可见，win32程序中当含有API函数时在执行流程中会出现大量嵌套的调用，密集的跳转以及对栈的使用，使得对其单步调试变得十分困难。从代码结构中可以看出：操作系统首先执行“主程序”，“主程序”获得与本程序相关的信息（句柄，命令行参数地址）后调用“窗口主程序”Winmain，“窗口主程序”配置窗口参数，注册并创建窗口类并装载菜单等各种资源，之后根据获得的信息（GetMessage）进入“窗口消息处理程序”WndProc,在窗口过程中根据收到的消息种类调用“用户处理程序”中的相应函数完成功能。

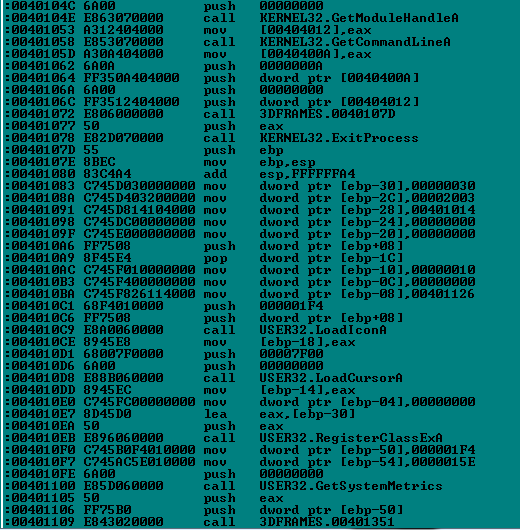


图4.2 3dframes函数反汇编语句

3.由图4.3和图4.4可见，TD32中cs,ds,ss窗口不再标注段名+16位地址而是直接在冒号后接32位地址，寄存器窗口不再提供16位段与32位段的切换功能，对通用寄存器组与eip只能观察其32位地址，TD32中CPU为电脑实际CPU配置，而TD中为虚拟机CPU配置。相同点在于窗口布局相同，cs窗口仍有地址，反汇编语句，指令，操作数四列；ds窗口仍显示地址，内存值与ASC字符；ss窗口仍显示地址与内存值。

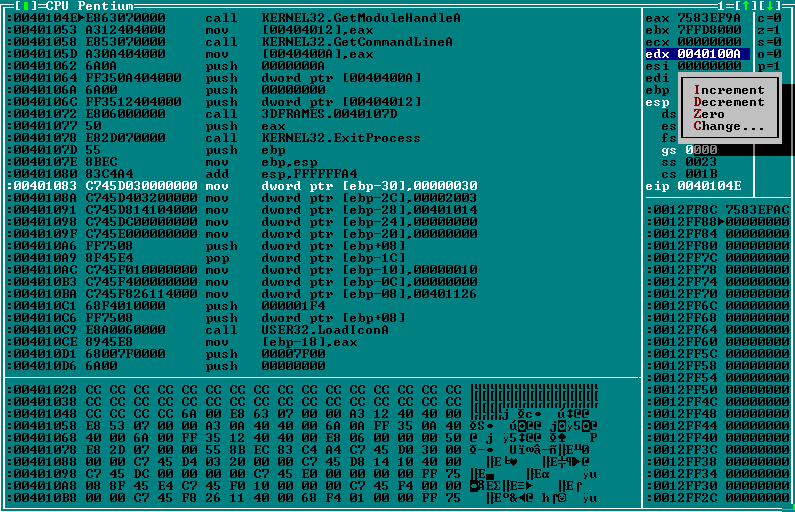


图4.3 TD32窗口

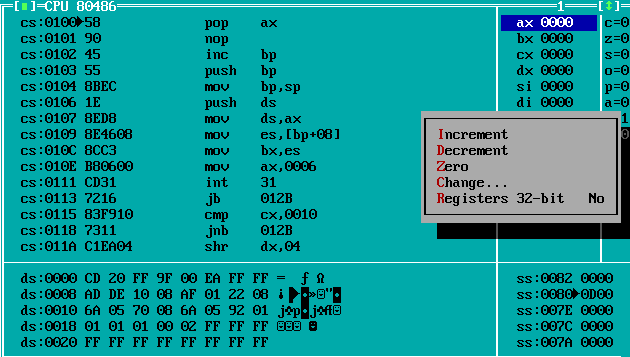


图4.4 TD窗口

4.当套用Win32 API模板时，API函数相关代码在TD中展开后理解存在困难，因此开发者应重点关注自己编写的功能函数的逻辑正确性与对API函数使用及参数配置的正确性。

5.配置win32\_sett文件与win32\_vcdebug文件，在控制台中编译汇编源文件cbox.asm和资源文件cbox.rc，期间重新导入了误删的cobx.inc。连接cobx.obj和cbox.res形成cbox.exe，全过程如图4.5所示。在vs2015中打开cbox.exe,再打开cbox.asm，界面如图4.6所示。可以看到VS界面对汇编语言并无显示结构上的辅助，需要自己安装高亮插件进行配置。但相比TD32中将winAPI函数全部反汇编，在显示上还是要简洁不少，且不用被窗口模板中的程序段干扰。

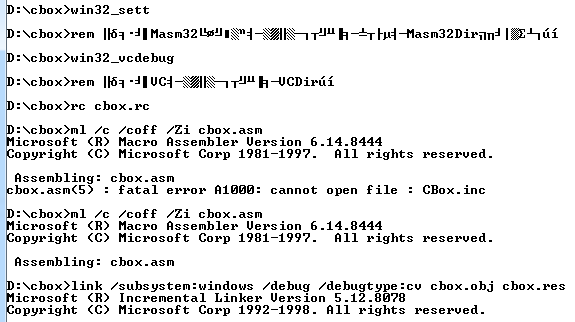


图4.5编译汇编连接全过程

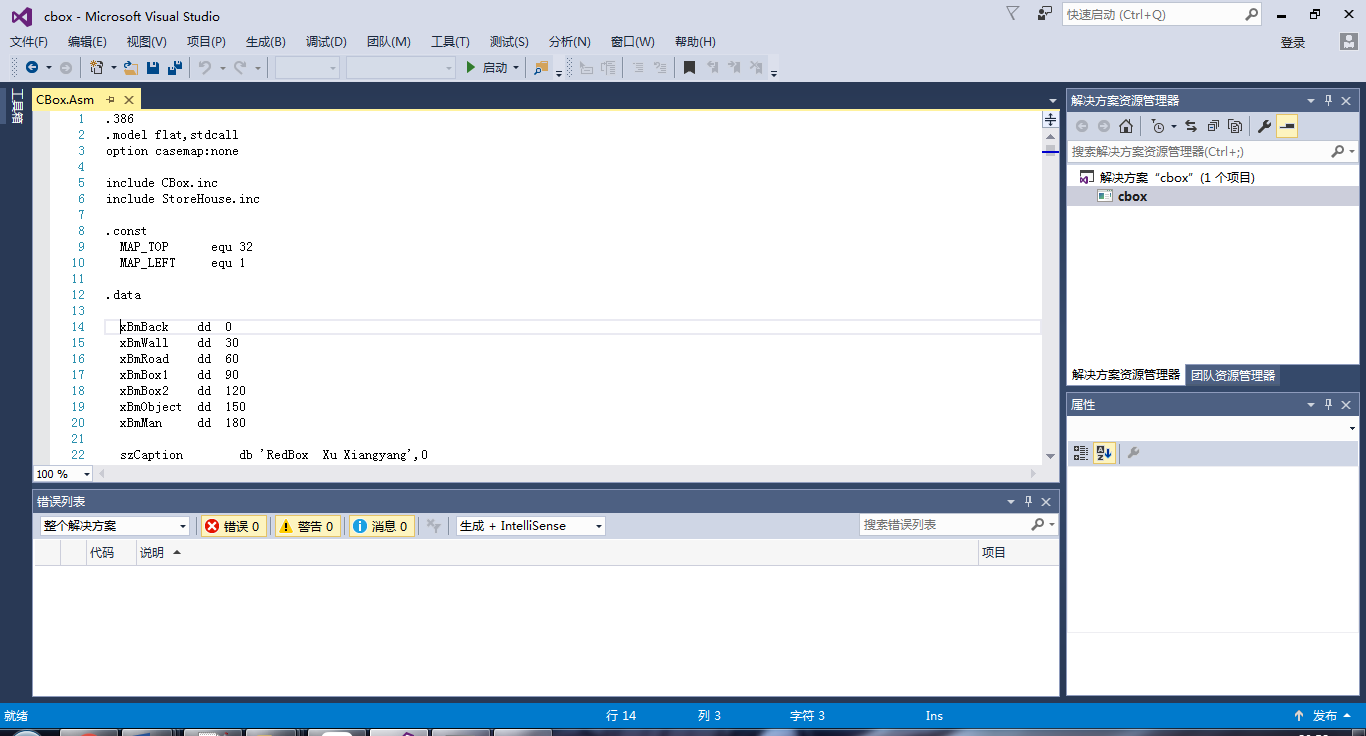


图4.6 cbox.asm在vs2015中显示

6.在5的基础上用TD打开cbox.exe，翻到数据段可直接看到偏移地址为0开始的部分内存空间因被保护而读不出来/显示为？号，用ss段的窗口查看也有同样的现象，如图4.7所示。

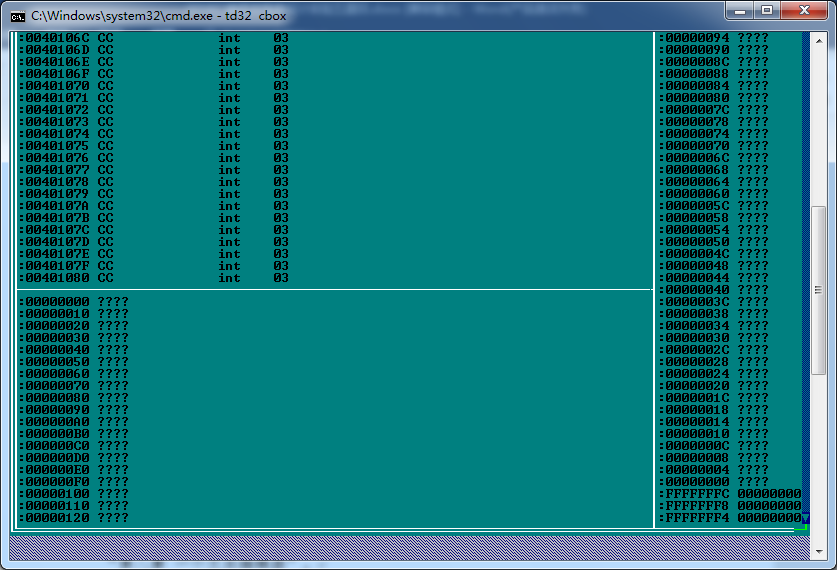


图4.7内存保护

7.见实验记录1-4.

9.DOS中输出采用2号调用（字符）或10号调用（有终止符$的串），也可采用第五章中的字符串操作；Win32中需要用Windows API函数进行输出，在本实验中最常使用TextOut函数，除此之外还有MessageBoxA函数等。两函数参数均含句柄与串首址指针，MessageBoxA另需要消息框样式；TextOut另需要起始输出位置坐标及串长。

12.以图4.8中的invoke TextOut,hdc,XX+0\*XX\_GAP,ECX,EDI,ESI为例（XX=10,XX\_GAP=100），在执行完PUSHY后，TextOut各参数从右向左按ESI,EDI,ECX,XX+0\*XX\_GAP=10,hdc=[EBP-04]的顺序依次被压栈。

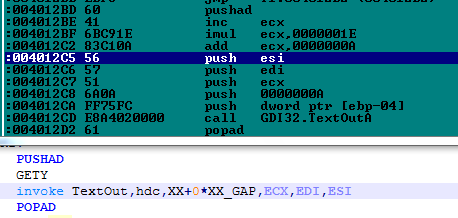


图4.8对invoke函数展开

# 参考文献

[1]许向阳，《80X86汇编语言程序设计上机指南》“第七章 MASM32环境”、“第八章 一个文本编辑器”。

[2] 汇编语言教学网站-》资料下载-》案例-》win32程序、编译和连接

其中的操作说明，给出了几种编译和链接生成执行程序的方法。

[3] 汇编语言教学网站-》资料下载-》书籍-》Win32汇编程序的源码级调试

其中的操作说明，给出了几种编译和链接生成执行程序的方法。

[4] MSDN（Microsoft Developer Network）， 有关 Windows API 的帮助。