### 

**Course Lab Report**

**Course Name: Assembly Language Programming Experiment**

**Experiment Name: Experiment 5 WIN32 Programming**

**Experiment time: 2017-05-14 , 14 :30-18 : 00 Experiment location: No. 90 test bench , Room 804, South 1st Floor**

**Instructor: Li Haibo**

**Professional class: school handover 201601 class**

**Student ID: U2016 12696 Name: Chen Haorui**

**Students in the same group: None Date of report: May 15, 2018**

**statement of originality**

I solemnly declare that the content of this report is independently completed by me, and the references to viewpoints, methods, data and literature have been pointed out in the text. Except for the content cited in the text, this report does not contain any other individual or collective published works or achievements, and there is no plagiarism or plagiarism.

Hereby declare!

Student signature:

Date : 2017. 5.15

performance evaluation

|  |  |  |
| --- | --- | --- |
| Experiment completion quality score (70 points) (experimental steps are clear, detailed and in-depth, experimental records are true and complete, etc.) | Report writing quality score (30 points) (report specification, complete, smooth, detailed, etc.) | Total score (100 points) |
|  |  |  |

Instructor's signature:

date:

**Table of contents**

[1](#_Toc515200104)  [Experimental purpose and requirements](#_Toc515200104)  [2](#_Toc515200104)

[2](#_Toc515200108)  [Experimental content](#_Toc515200108)  [2](#_Toc515200108)

[3](#_Toc515200112)  [Experimental process](#_Toc515200112)  [3](#_Toc515200112)

[3.1](#_Toc515200113)  [Task 1](#_Toc515200113)  [3](#_Toc515200113)

[3.1.1](#_Toc515200114)  [Design idea and storage unit allocation](#_Toc515200114)  [3](#_Toc515200114)

[3.1.2](#_Toc515200115)  [Flowchart](#_Toc515200115)  [3](#_Toc515200115)

[3.1.3](#_Toc515200116)  [Source program](#_Toc515200116)  [4](#_Toc515200116)

[3.1.4](#_Toc515200117)  [Experimental Step](#_Toc515200117)  [5](#_Toc515200117)

[3.1.5](#_Toc515200118)  [Experimental records and analysis](#_Toc515200118)  [5](#_Toc515200118)

[4](#_Toc515200119)  [Summary and experience](#_Toc515200119)  [10](#_Toc515200119)

[Reference](#_Toc515200120)  [15](#_Toc515200120)

# Experimental purpose and requirements

The main purpose and requirements of this experiment are the following 3 points. All tasks will be carried out around these 3 points. I hope you can check whether you have met these goals and requirements afterwards.

1. Familiar with the design and debugging methods of WIN32 programs;
2. Familiar with functions such as INVOKE, structure variables, and simplified segment definitions in macro assembly language;
3. Further understand some relations among machine language, assembly language, high-level language, implementation mode and protection mode.

# Experimental content

Write a window-based WIN32 program to realize the average profit rate calculation and product information display functions of the **product information management program of the online store (referring to some practices from Experiment 3), and the specific requirements are described as follows.**

Function 1: Write a window-based menu frame for WIN32 programs, with the following drop-down menu items:

File Action Help

Exit Average About

list

Click the Exit option under the menu File to end the program; click the option About under the menu Help, and a message box will pop up, displaying personal information, similar to that shown in Figure 2.1 . Click the options Average and List under the menu Action to realize the functions of calculating the average profit rate or displaying all product information of SHOP1 respectively (see the description of function 2 for details).



Figure 2.1 Menu Example

Function 2: It is required to use structural variables to store relevant information of commodities. The number of commodities is defined at least 5 types.

When ordering the menu item Average, calculate the average profit rate of all commodities according to the method of Experiment 3. Use TD32 to observe the calculation results.

all the product information of SH OP1 in the window . The specific display format can be defined by yourself, you can refer to the style in Figure 2.2 (Chinese is not required).



Figure 2.2 Schematic diagram of commodity information display

# experiment procedure

## task 1

### Design Thought and Storage Unit Allocation

in the menu , which are File, Action and Help. Among them, there is an Exit function under File , and the program can be exited after selection; there are two functions of A verage and List under Action , and A verage is selected to calculate all commodities The average profit margin, select List to list all commodity information of SH OP1 in the window ; there is A bout function under Help , and the developer information will be displayed after selection .

1. storage unit allocation

First define the cmodty commodity structure, which contains the byte string goodname (commodity name) with a length of 10 , dw type in\_price (purchase price), dw type out\_price (selling price), dw type in\_num (purchase quantity), dw type out\_num (already sales quantity) and dw type interest (current profit).

SHOP1 , SHOP2: two cmodty structure arrays with a length of 5 , and the commodities in the two arrays are the same.

OLD\_INT : Double word variable , the first two bytes store the old IP, and the last two bytes store the old CS.

hInstance , CommandLine : double-word type variable, the value is 0 , as the parameter of the API function .

ClassName , AppName , MenuName , DlgName , AboutMsg : Byte string, information saved in window title or dialog box.

msg\_name , msg\_iprice , msg\_oprice , msg\_inum , msg\_onum , msg\_interest : byte string, storage .

Other variables have been accounted for in Experiment 3 .

2. Register allocation

varies from function to function .

### flow chart,

Figure 3.1.1 is Task 1 Flowchart of a Win32 window program .

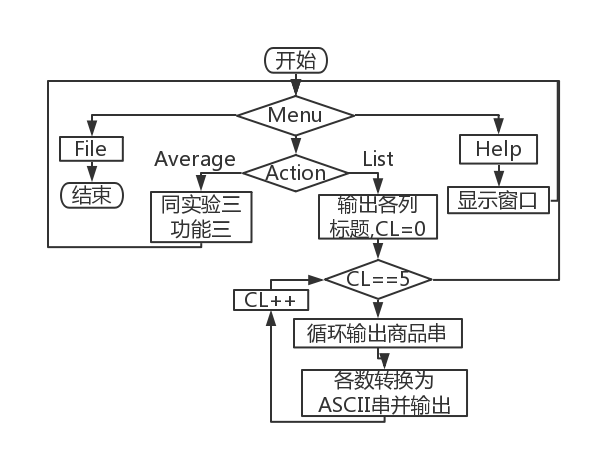


Figure 3.1.1 Win32 window program flow chart

### source program

M enuID. 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

cmdty 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

cmdty ends

getymacro \_

INC ECX

IMUL ECX,YY\_GAP

ADD ECX,YY

ENDM

IO MACROADS,NM

LEA DX,ADS

MOV AH, NM

INT 21H

ENDM

INTR MACROT ; The parameter is part of the register name

MOV SI,DS:[E&T&+10] ; Purchase price

IMUL SI,DS:[E&T&+14] ; Purchase quantity

MOV AX,DS:[E&T&+12] ;Sales price

IMUL AX,DS:[E&T&+16]; Sold Quantity

SUB AX, SI

IMUL HKU

IDIV SI

ENDM

FND MACROCOUT, PIN, A, B, N; The parameter is the return label when it succeeds

MOV SI,0

LP&A&: ; Get the current product string length in store 1 and save it in AX(AL)

CMPBYTE PTR DS:[EDI+ESI],0

JEBRK&N&

INC SI

JMPLP&A&

BRK&N&:

MOV AX,SI

MOV CH,M ; CH is responsible for the traversal of store two

LEA EBX,PIN

LP&B&:

CALL CMPST ; Compare whether the two commodity strings are the same, if not, AH is 0 (default value)

ORAH, AH

JNZCOUT

ADDEBX,20 ; EBX moves to the next product in store 2, continue to compare

DECCH

JNZLP&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 cmdty <'PEN',35,56,70,45,?>

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

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

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

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

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

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

cmdty <'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,'$' ;line feed, carriage return

PR1 DW? ; profit address in store one

M DB5 ; Number of valid commodities

TAD DB90DUP(0); Temporary array storing ranking and offset addresses

TNM DB4DUP(0); Temporary array for storing decimal numbers

HKU DW100

.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

Y Y 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 is responsible for the traversal of store one

LEA EDI, SHOP1

LPI:

XOR ESI,ESI

LPI\_A: ;SI gives the string length of the product

MOV DL,[EDI+ESI]

CMP DL,0

JEBRK1

INC ESI

JMP LPI\_A

BRK1:

PUSHAD

getty

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 ; Convert to decimal and output ASCII characters

PUSHAD

IMUL ESI,XX\_GAP

ADD ESI,XX

getty

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

JZBRK2 ; Commodities have not been traversed

ADD EDI,12

JMP LPI

BRK2:

ret

Display endp

MRADIX PROCUSES CX EDI

XOR ECX,ECX

LEA EDI, TNM

LPQ:

XOR DX,DX

DIV BX

PUSH DX

INC CX ; Save digits

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 ; the current character is stored in a temporary string

INC EDI

LOOP LPR

RET

MRADIX ENDP

CMPST PROC ; compare string function

MOV SI,0

LPE:

MOV DH,DS:[EDI+ESI]

CMP DH,DS:[EBX+ESI]

JNEBRK9

INCSI

CMPSI,AX

JNELPE

INC SI

CMPBYTE PTR DS:[EBX+ESI],0; judge whether BX is also traversed

JNEBRK9

INCAH ; same string

BRK9:

RET

CMPST ENDP

PART3 PROCUSES AX CX SI BP

MOV CL,M ; CL is responsible for the traversal of store one

LEA EDI, SHOP1

LPC: ;The previous values of all registers can be invalidated

FND BRK10,SHOP2,A,B,11 ; cyclically compare the strings pointed to by BP and BX, if they are the same, go to BRK11

BRK10: ; Update the average profit rate of each product (CL, EDI, EBX are occupied)

INTRDI ; Calculate the profit margin of the item pointed to by EDI

MOV DS:[EDI+18],AX;Save EDI profit

INTRBX ; Calculate the profit margin of the commodity pointed to by EBX

ADD AX,DS:[EDI+18]; Add the two profits

CWD

SAR AX,1 ; average profit

MOV DS:[EDI+18],AX ;The average profit is stored in the profit field of SHOP1

DEC CL

ORCL,CL

JZBAR5

ADD EDI,20 ; BP moves to the next item in store one

JMPLPC

BAR5:

RET

PART3 ENDP

end Start

### Experimental procedure

1. Install MASM 32 and prepare the experimental environment.

2. Based on the sample window program framework , modify the relevant parameters in menuID.inc , t1.rc and t1.asm , and modify the menu as shown in Figure 2.1.

3. Based on the framework of the previous step , extract function 3 and function 5 of experiment 3 and modify them . Function 3 corresponds to the function of A average, and function 5 corresponds to the function of List, which are respectively called by Average and List under Action , and modify t1 at the same time Related call parameters in .asm . Repeat the compilation and linking process until the executable file is successfully generated.

4. Use TD32 to debug step by step until the correct result can be output when List is called .

### Experiment Recording and Analysis

1. Install MASM32 and prepare the experimental environment.

2. On the basis of the sample program, modify the variables in the source file and the menu display mode in the .rc resource file . After consulting with classmates , you know that the '&' inserted in the directory pass in the rc file is used to activate the Alt shortcut key Invocation, the symbol should precede the letter that will be paired with Alt and is case- insensitive . Compile and connect , and find an error as shown in Figure 3.1.2 . After flipping through the book, I found that the header file in .inc has not been modified.

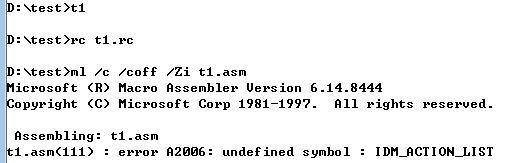


Figure 3.1.2 Assembly error

3. After modifying the constant definition in the header file of the .inc header file , put var.bat and nmake.exe into the source file directory according to the example file cbox , and directly use var and nmake to compile and connect , and find an error as shown in Figure 3.1.3 . Based on this, I understand that makefile is a configuration file when compiling and connecting , not a file generated after compiling and connecting . Put the makefile into the directory , modify the name of the target file and try to compile and connect again , and no exception occurs during the process .

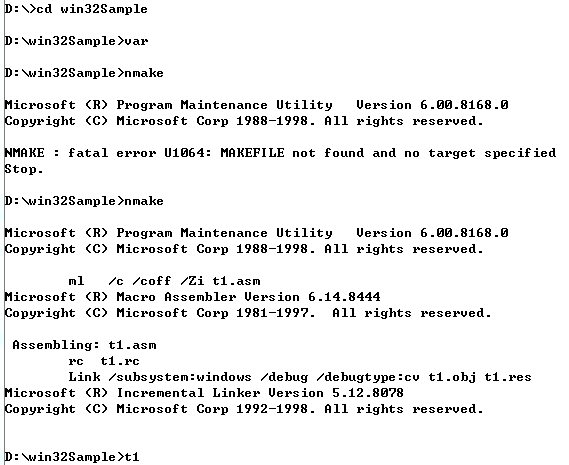


Figure 3.1.3 Use var to nmake to try to compile and connect

4. Enter t1, run A bout and observe the subdirectories under Action , the results are shown in Figure 3.1.4 and 3.1.5 , which proves that the directory modification is successful.

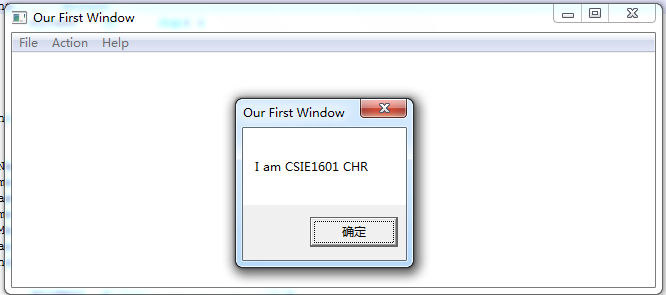


Figure 3.1.4 Run About

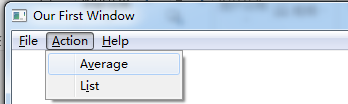


Figure 3. 1.5 Observe subdirectories under Action

5. Functions 3 and 5 of Experiment 3 were adapted according to the ideas in the experimental steps and then called by A verage and List respectively . There was no error in compiling and linking , but both crashed when executing A verage and List . Using TD32 single-step debugging function 3, it is found that the memory is shown in Figure 3.1.6 when the execution reaches ret , that is, the five profits are all calculated correctly, but after returning to the WndProc function , an error crash is reported, and the error address prompted is the same as the current address of ebp , such as As shown in Figure 3.1.7 . Guess that ebp is related to the call of the function , so you should try to avoid using ebp in your own program .

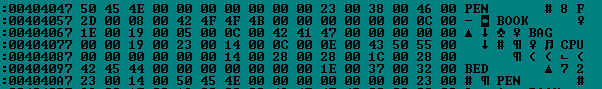


Figure 3.1.6 Data calculation is normal

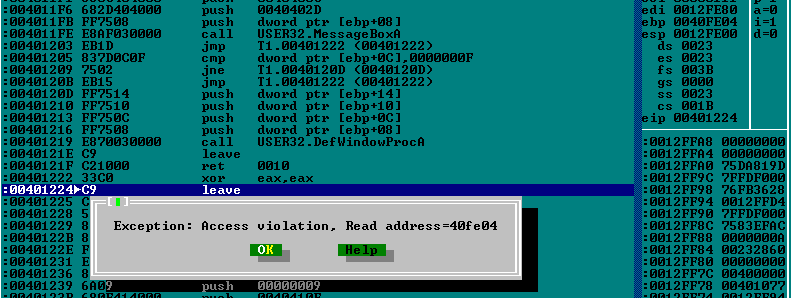


Figure 3.1.7 crashes after returning, guess e bp cannot be used

6. Replace ebp with edi , recompile and connect , and find that the Average function is executed normally, but the List program still crashes. After single-step debugging, it is found that an error is reported at the system function call , as shown in Figure 3.1.8 . It shows that it is no longer suitable to use system function calls for output in Win32 programming , and W indows API functions should be used instead .

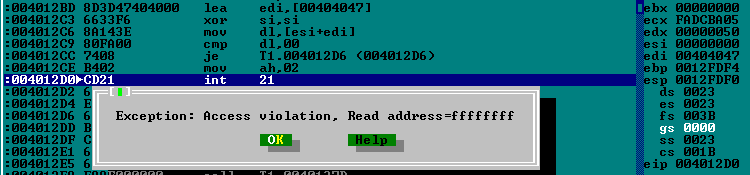


Figure 3.1.8 The interrupt handler cannot be used

7. When re-debugging, I found that an error was reported at the position shown in Figure 3.1.9 , because the error position was not the statement I wrote and there was no output in the window when the error occurred. I thought the error position was in the TextOut function , so I had no clue at all. Ask the teacher for help here. The following is the solution process (refer to Figure 3.1.10) : F8 single-step execution: the API call at 004012 E0 , and found that no error was reported at this time . This proves that the TextOut function is not wrong, or that the parameter configuration of the invoke function is not wrong here. Continue to debug downwards , you can see that there are different codes in the middle of the push before the TextOut call at 00401319 , and a push to 16 -bit register is performed at 0040401312 , and the parameters of the second call to TextOut are compared , found that here is a direct operation on the memory variable, so try to store this memory variable into a register and then use it as a parameter .

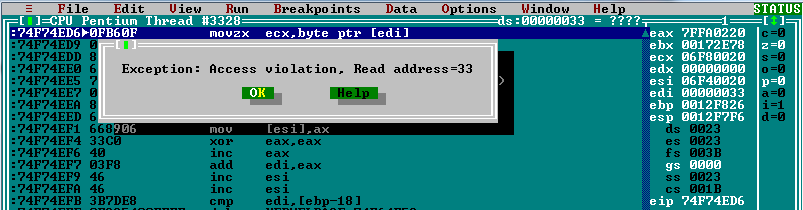
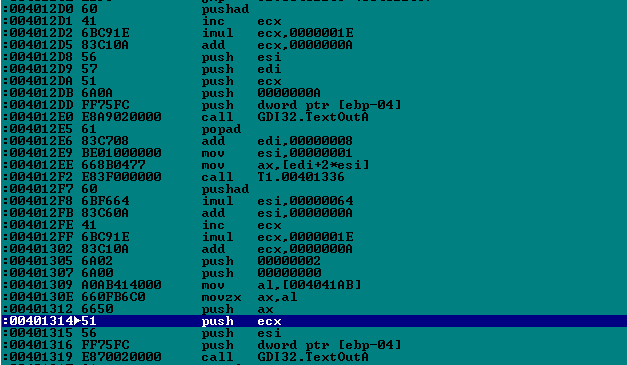


Figure 3.1.9 The interrupt handler cannot be used



interferes with the push of 16-bit registers

8. Compile, connect and run again , and find that the output is successful , as shown in Figure 3.1.11 , but there is still a problem with the calculation of the profit rate.

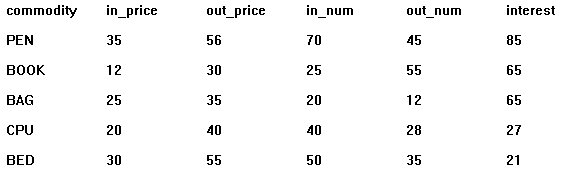


Figure 3.1.11 Data writing error

After the single-step inspection of TD32 , it was found that EBP was changed to EDI by mistake . In the CMPST function , EBX was also changed to EDI. As a result , when comparing strings , it changed to compare with itself, and EBX always stayed at store 2. At the first commodity , resulting in an error in the calculation of the profit margin. The corrected result is shown in Figure 3.1.12 . In addition, comment out .ELSEIF uMsg == WM\_PAINT in the Winproc function , that is, no longer judge this situation , and you can keep the current display when dragging the window.

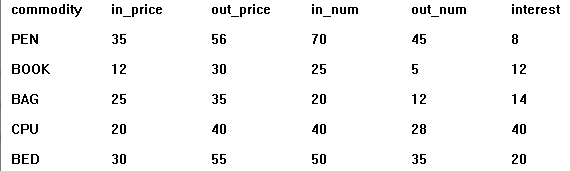


Figure 3.1.12 Profit margin returns to normal

# Summary and experience

This experiment brought me into contact with assembly programming under Win 32 for the first time . Through this experiment, I have some understanding of the Win 32 assembly environment, the method of compiling and linking to generate executable files , and the use of API functions , especially the TextoutA function . The use of interrupt handlers is no longer allowed under the bit , and there are restrictions on the use of registers (such as EBP) . I am familiar with the calling relationship of .inc files , .rc files , and .asm files .

In the process of observing the teacher helping to debug the program, I have gained two experiences . One is that sometimes it is difficult for us to know the cause of the problem due to limited knowledge , but this does not prevent us from trying to adjust the program through a troublesome but feasible way , such as this time the TNM The address is first assigned to the 32-bit register and then the register is used as a parameter. This may appear redundant in a 16 -bit environment; but it is necessary in a 32-bit environment . As for why the memory variable is set as the parameter of TextoutA , there will be 5 steps of unclear operations, which need further study . The second is to face strange and repeated codes during single-step debugging. You can try to compare them with each other . For example, when TextoutA outputs the product name, everything is normal, but there is a problem with the number after the output. It is also a black box inside invoke In the case of , the parameter is likely to be in the configuration of the actual parameter . At this time, you can focus on comparing the transfer process of parameters, so that it is more likely to locate the problem.

Thinking questions ( not involved in the experimental steps ) :

under M asm32 is shown in Figure 4.1 and 4.2 .

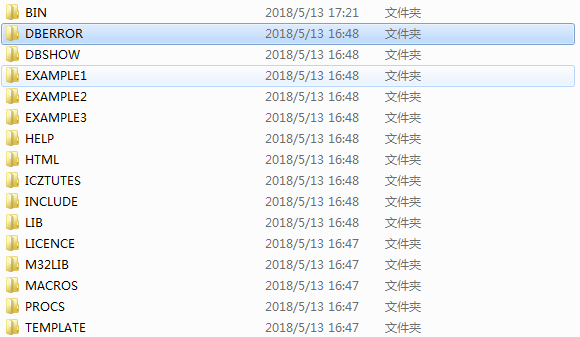




Figure 4.1 & Figure 4.2 Memory - resident program nesting

2. As can be seen from Figure 4.2, when a win32 program contains API functions, there will be a large number of nested calls in the execution process , intensive jumps and use of the stack, making it very difficult to debug single-step. It can be seen from the code structure : the operating system first executes the " main program " , and the "main program" obtains information related to this program ( handle , command line parameter address) and then calls the " window main program " Winmain , "window main program" Configure window parameters, register and create window classes and load various resources such as menus , and then enter the " window message processing program " WndProc according to the obtained information ( GetMessage ) , and call the " user processing program " according to the type of message received during the window process " The corresponding function in " completes the function.

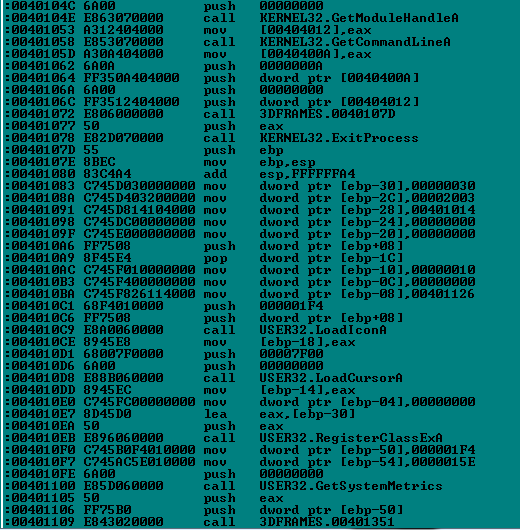


Figure 4.2 3dframes function disassembly statement

3. As can be seen from Figure 4.3 and Figure 4.4 , the cs, ds, and ss windows in TD32 no longer mark the segment name + 16- bit address but directly connect the 32-bit address after the colon , and the register window no longer provides 16-bit segments and 32-bit segments The switching function of the general register group and eip can only observe its 32-bit address. The CPU in TD32 is the actual CPU configuration of the computer , and the virtual machine CPU configuration in TD . The same point is that the window layout is the same, the cs window still has four columns of addresses, disassembly statements, instructions, and operands; the ds window still displays addresses , memory values and ASC characters ; the ss window still displays addresses and memory values.

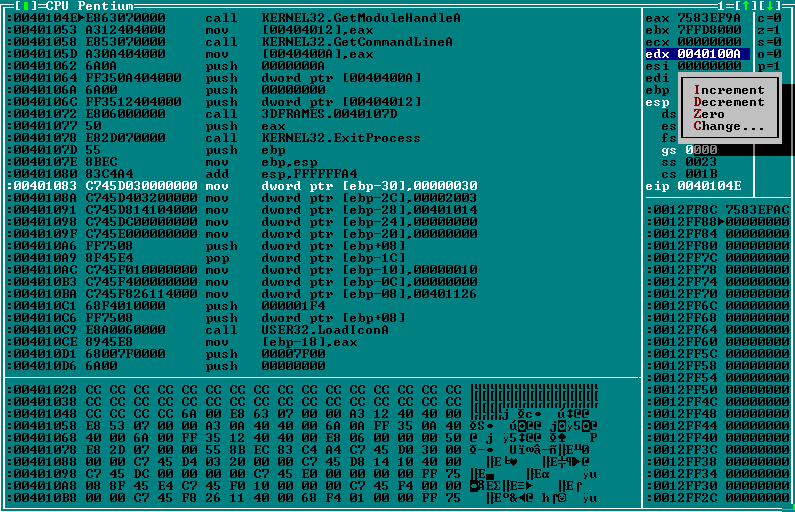


Figure 4.3 TD32 window

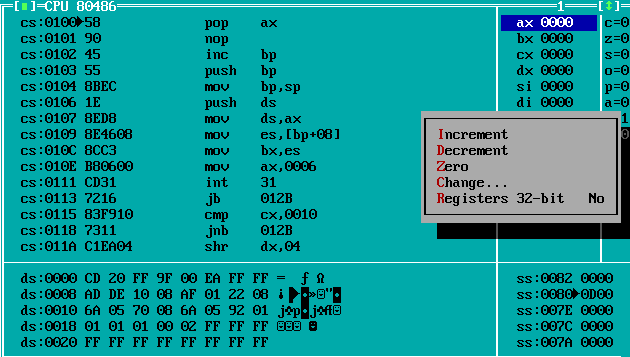


Figure 4.4 TD window

Win32 API template is applied , it is difficult to understand the API function-related codes after being expanded in TD , so developers should focus on the logical correctness of the functional functions they write and the correctness of the use of API functions and parameter configuration.

5. Configure the win32\_sett file and win32\_vcdebug file , compile the assembly source file c box.asm and the resource file cbox.rc in the console , and re-import the cobx.inc that was deleted by mistake . Connect cobx.obj and cbox.res to form cbox.exe, the whole process is shown in Figure 4.5 . Open cbox.exe in vs2015 , and then open cbox.asm, the interface is shown in Figure 4.6 . It can be seen that the VS interface has no assistance in displaying the structure of the assembly language , and you need to install the highlight plug-in for configuration . However, compared to disassembling all winAPI functions in TD32 , the display is still much more concise, and there is no need to be disturbed by the program segments in the window template .

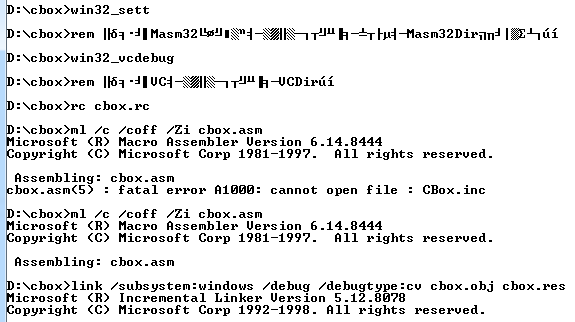


Figure 4.5 The whole process of compilation, assembly and connection

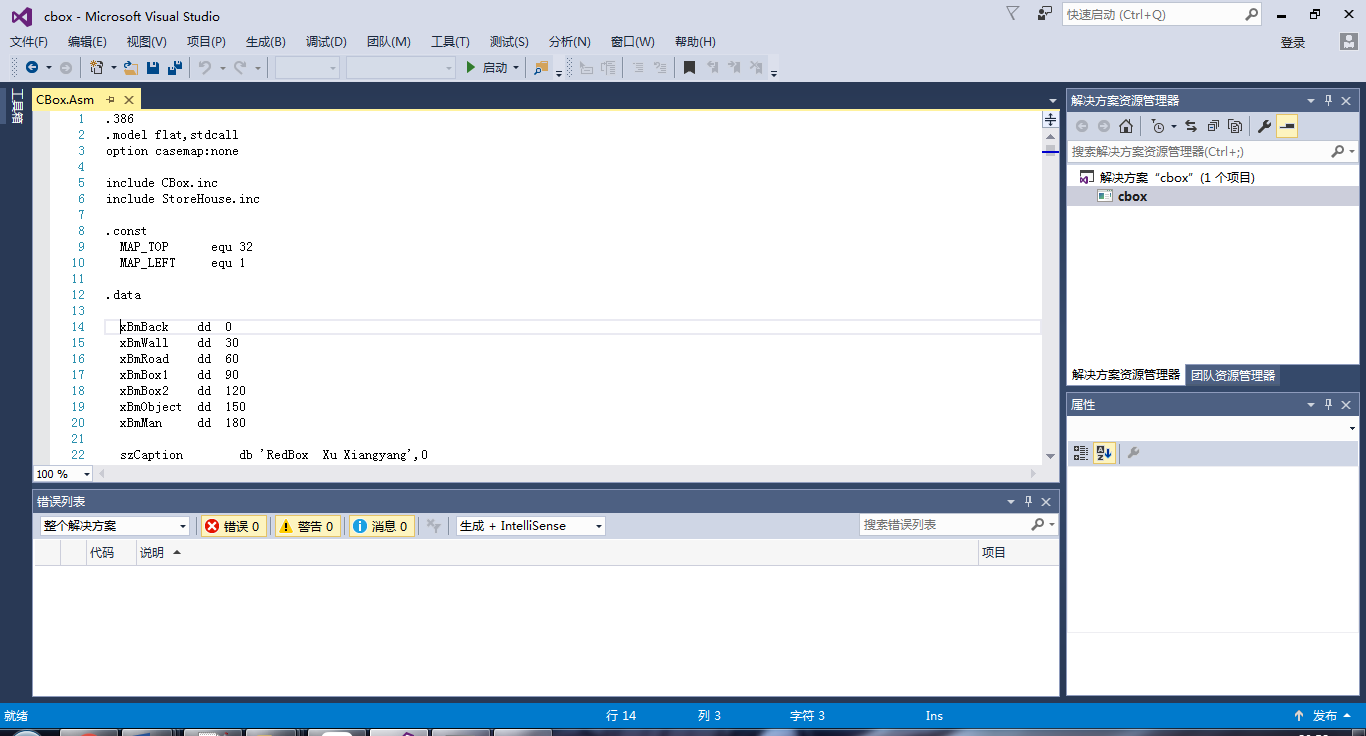


Figure 4.6 c box.asm shows in vs 2015

6. On the basis of 5 , use TD to open cbox.exe , turn to the data segment , and you can directly see the part of the memory space whose offset address is 0 is protected and cannot be read /displayed as ? No. , the same phenomenon can be seen with the window of the ss segment, as shown in Figure 4.7 .

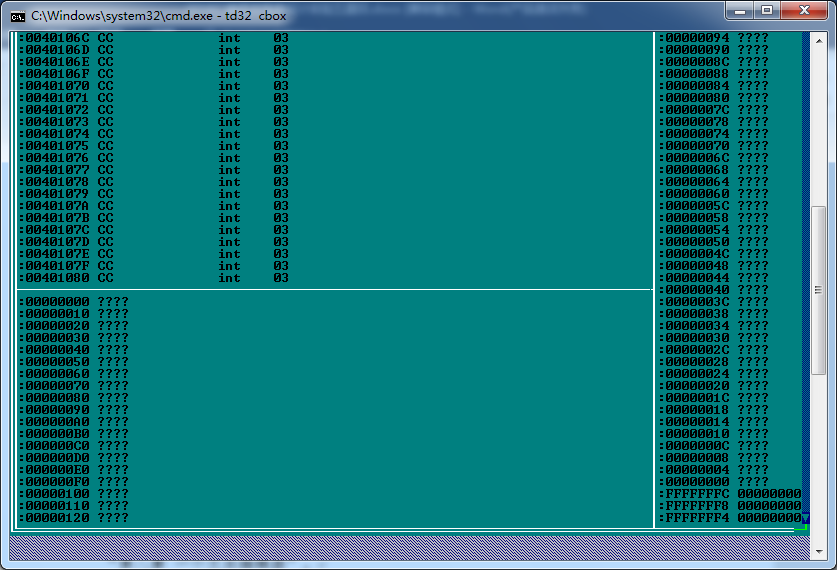


Figure 4.7 Memory Protection

7. See experiment records 1 -4.

9. In DOS, the output adopts No. 2 call ( character ) or No. 10 call ( string with the terminator $ ) , and the string operation in Chapter 5 can also be used; Win32 needs to use W indows API function for output , in In this experiment the most commonly used TextOut function, in addition to the M essageBoxA function and so on . The parameters of the two functions include the handle and pointer to the first address of the string . M essageBoxA also needs the style of the message box; TextOut also needs the coordinates of the initial output position and the string length.

12. Take invoke TextOut, hdc, XX+0\*XX\_GAP, ECX, EDI, ESI in Figure 4.8 as an example (XX= 10, XX\_GAP=100 ) , after executing PUSHY , press ESI for each parameter of TextOut from right to left , EDI, ECX, XX+0\*XX\_GAP=10, hdc= [EBP-04] are pushed onto the stack in sequence .

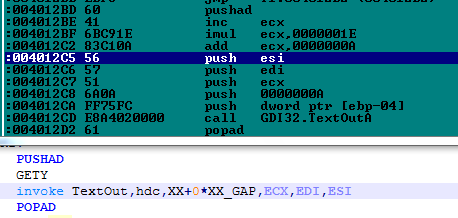


Figure 4.8 Expansion of the invoke function

# references

[1] Xu Xiangyang, "80X86 Assembly Language Programming Hands-on Guide", "Chapter 7 MASM32 Environment", "Chapter 8 A Text Editor".

[2] Assembly language teaching website-"data download-"case-"win32 program, compilation and connection

The operating instructions among them give several methods of compiling and linking to generate execution programs.

[3] Assembly language teaching website - "data download -" books - "source level debugging of Win32 assembler

The operating instructions among them give several methods of compiling and linking to generate execution programs.

[4] MSDN (Microsoft Developer Network), Help on Windows API.