

汇编语言程序设计实验报告

目录

1	实验目的与要求.....	1
2	实验内容.....	1
3	实验过程.....	2
3.1	任务 1.....	2
3.1.1	设计思想及存储单元分配.....	2
3.1.2	流程图.....	3
3.1.3	源程序.....	4
3.1.4	实验步骤.....	9
3.1.5	实验记录与分析.....	9
4	总结与体会.....	17
	参考文献.....	18

汇编语言程序设计实验报告

1 实验目的与要求

- (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。

2 实验内容

任务 1：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）。

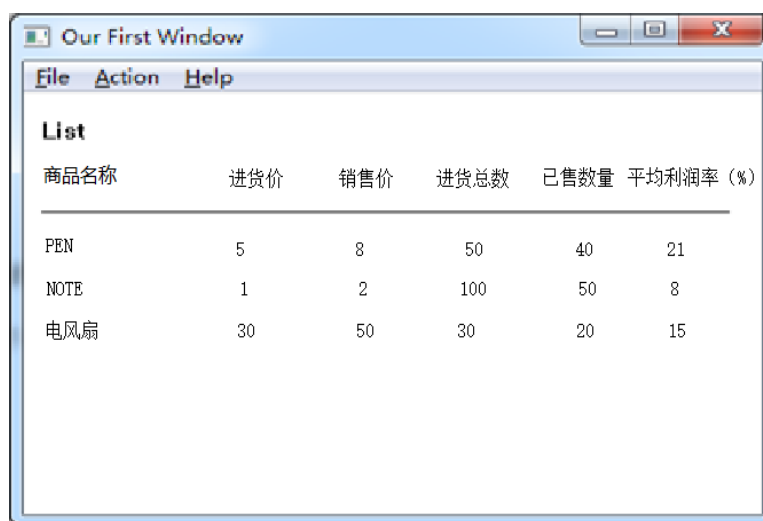


图 2.1 菜单示例

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）。

汇编语言程序设计实验报告



The screenshot shows a window titled "Our First Window" with a menu bar containing "File", "Action", and "Help". Below the menu bar is a section titled "List" containing a table with the following data:

商品名称	进货价	销售价	进货总数	已售数量	平均利润率 (%)
PEN	5	8	50	40	21
NOTE	1	2	100	50	8
电风扇	30	50	30	20	15

图 2.2 商品信息显示示意图

3 实验过程

3.1 任务 1

3.1.1 设计思想及存储单元分配

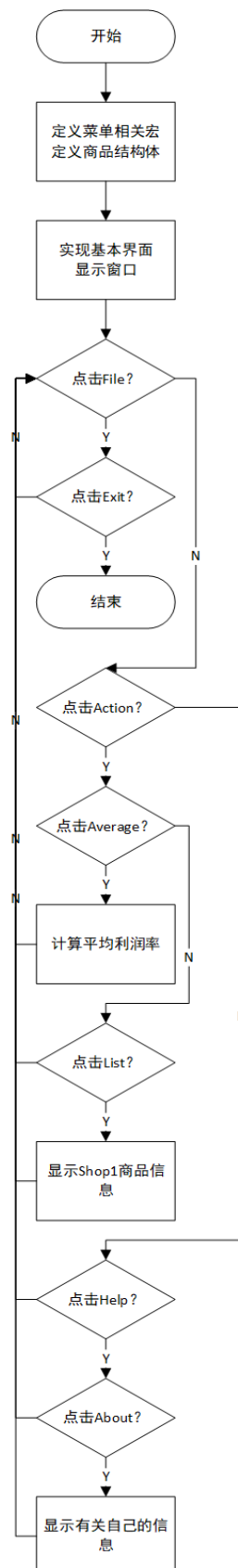
设计思想: 这次程序使用到的计算利润率子程序以及显示商品信息子程序主要设计方法在实验三任务一中, 在此不赘述。

先设计主程序, 然后再主程序中调用窗口主程序, 在窗口主程序中设计窗口信息处理程序, 同时要设计处理这些信息用到的用户处理程序。

程序中菜单栏使用库里面包含的函数可以实现。在显示文本信息时使用了 `TextOut` 函数, 其具体使用方式在源程序里给出, 在思考题中也有涉及。

汇编语言程序设计实验报告

3.1.2 流程图



汇编语言程序设计实验报告

图 3.1.1 程序流程图

3.1.3 源程序

```
.386
.model flat,stdcall
option casemap:none

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD
WndProc proto :DWORD,:DWORD,:DWORD,:DWORD
Display proto :DWORD
Rate proto :DWORD ;计算利润率
f2t10 proto ;2 进制转换十进制
radix proto

include ..\INCLUDE\menuID.INC ;菜单相关

include ..\INCLUDE\windows.inc
include ..\INCLUDE\user32.inc
include ..\INCLUDE\kernel32.inc
include ..\INCLUDE\gdi32.inc
include ..\INCLUDE\shell32.inc

includelib ..\LIB\user32.lib
includelib ..\LIB\kernel32.lib
includelib ..\LIB\gdi32.lib
includelib ..\LIB\shell32.lib

goods struct ;商品结构体
    goodsname db 10 dup(0)
    cost dw 0 ;在此定义为 dw 类型，在后面的子程序中要做相应改变
    price dw 0
    total dw 0
    sold dw 0
    profit dw 0
goods ends

.data
ClassName db 'TryWinClass',0
AppName db 'A Simple Window!',0
MenuName db 'MyMenu',0
DlgName db 'MyDialog',0
AboutMsg db 'I am student Chaofan Wang from CSIE1601',0
hInstance dd 0
CommandLine dd 0
shop1 goods <'pen',2,5,50,40,0> ;商品信息，利润率没计算时为 0
      goods <'book',10,15,20,15,0>
      goods <'kindle',58,170,20,11,0>
      goods <'bottle',16,40,25,15,0>
      goods <'switch',108,190,30,20,0>
shop2 goods <'pen',3,6,80,68,0>
      goods <'book',10,20,30,16,0>
      goods <'kindle',90,280,30,13,0>
      goods <'bottle',12,30,25,9,0>
      goods <'switch',149,280,50,39,0>
```

汇编语言程序设计实验报告

```
msg_name db 'goods name',0
msg_cost db 'cost',0
msg_price db 'price',0
msg_total db 'total',0
msg_sold db 'sold',0
msg_profit db 'profit',0
msg_shop1 db 'shop1',0
cost db '2','10','58','16','108' ;便于输出
price db '5','15','170','40','190'
total db '50','20','20','25','30'
sold db '40','15','11','15','20'
prol dd 0 ;存放 shop1 利润
buf db 10 dup(?) ;转换利润便于输出

.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, 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
```

汇编语言程序设计实验报告

```
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
                     ;;我的代码
                 .ENDIF
             .ELSEIF uMsg == WM_COMMAND
                 .IF     wParam == IDM_FILE_EXIT ;退出
                     invoke SendMessage, hWnd, WM_CLOSE, 0, 0
                 .ELSEIF wParam == IDM_ACTION_AVERAGE ;计算平均利润率
                     lea esi, shop1
                     invoke Rate, esi
                     add esi, 20
                     invoke Rate, esi
                     add esi, 20
                     invoke Rate, esi
                     add esi, 20
                     invoke Rate, esi
                     add esi, 20
                     invoke Rate, esi
                     add esi, 20
                     invoke Rate, esi
                 .ELSEIF wParam == IDM_ACTION_LIST ;显示列表
                     invoke Display, hWnd
                 .ELSEIF wParam == IDM_HELP_ABOUT ;显示信息
                     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 ;显示 shop1 信息
             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_shop1, 5
             invoke   TextOut, hdc, XX+0*XX_GAP, YY+1*YY_GAP, offset msg_name, 10
             invoke   TextOut, hdc, XX+1*XX_GAP, YY+1*YY_GAP, offset msg_cost, 4
             invoke   TextOut, hdc, XX+2*XX_GAP, YY+1*YY_GAP, offset msg_price, 5
             invoke   TextOut, hdc, XX+3*XX_GAP, YY+1*YY_GAP, offset msg_total, 5
             invoke   TextOut, hdc, XX+4*XX_GAP, YY+1*YY_GAP, offset msg_sold, 4
             invoke   TextOut, hdc, XX+5*XX_GAP, YY+1*YY_GAP, offset msg_profit, 6
             ;;
```

汇编语言程序设计实验报告

```
invoke TextOut,hdc,XX+0*XX_GAP,YY+2*YY_GAP,offset shop1[0*20].goodsname,3
invoke TextOut,hdc,XX+1*XX_GAP,YY+2*YY_GAP,offset cost,1
invoke TextOut,hdc,XX+2*XX_GAP,YY+2*YY_GAP,offset price,1
invoke TextOut,hdc,XX+3*XX_GAP,YY+2*YY_GAP,offset total,2
invoke TextOut,hdc,XX+4*XX_GAP,YY+2*YY_GAP,offset sold,2
mov ax,shop1[0*20].profit
invoke f2t10
invoke TextOut,hdc,XX+5*XX_GAP,YY+2*YY_GAP,offset buf,2
```

```
invoke TextOut,hdc,XX+0*XX_GAP,YY+3*YY_GAP,offset shop1[1*20].goodsname,4
invoke TextOut,hdc,XX+1*XX_GAP,YY+3*YY_GAP,offset cost+1,2
invoke TextOut,hdc,XX+2*XX_GAP,YY+3*YY_GAP,offset price+1,2
invoke TextOut,hdc,XX+3*XX_GAP,YY+3*YY_GAP,offset total+2,2
invoke TextOut,hdc,XX+4*XX_GAP,YY+3*YY_GAP,offset sold+2,2
mov ax,shop1[1*20].profit
invoke f2t10
invoke TextOut,hdc,XX+5*XX_GAP,YY+3*YY_GAP,offset buf,2
```

```
invoke TextOut,hdc,XX+0*XX_GAP,YY+4*YY_GAP,offset shop1[2*20].goodsname,6
invoke TextOut,hdc,XX+1*XX_GAP,YY+4*YY_GAP,offset cost+3,2
invoke TextOut,hdc,XX+2*XX_GAP,YY+4*YY_GAP,offset price+3,3
invoke TextOut,hdc,XX+3*XX_GAP,YY+4*YY_GAP,offset total+4,2
invoke TextOut,hdc,XX+4*XX_GAP,YY+4*YY_GAP,offset sold+4,2
mov ax,shop1[2*20].profit
invoke f2t10
invoke TextOut,hdc,XX+5*XX_GAP,YY+4*YY_GAP,offset buf,2
```

```
invoke TextOut,hdc,XX+0*XX_GAP,YY+5*YY_GAP,offset shop1[3*20].goodsname,6
invoke TextOut,hdc,XX+1*XX_GAP,YY+5*YY_GAP,offset cost+5,2
invoke TextOut,hdc,XX+2*XX_GAP,YY+5*YY_GAP,offset price+6,2
invoke TextOut,hdc,XX+3*XX_GAP,YY+5*YY_GAP,offset total+6,2
invoke TextOut,hdc,XX+4*XX_GAP,YY+5*YY_GAP,offset sold+6,2
mov ax,shop1[3*20].profit
invoke f2t10
invoke TextOut,hdc,XX+5*XX_GAP,YY+5*YY_GAP,offset buf,2
```

```
invoke TextOut,hdc,XX+0*XX_GAP,YY+6*YY_GAP,offset shop1[4*20].goodsname,6
invoke TextOut,hdc,XX+1*XX_GAP,YY+6*YY_GAP,offset cost+7,3
invoke TextOut,hdc,XX+2*XX_GAP,YY+6*YY_GAP,offset price+8,3
invoke TextOut,hdc,XX+3*XX_GAP,YY+6*YY_GAP,offset total+8,2
invoke TextOut,hdc,XX+4*XX_GAP,YY+6*YY_GAP,offset sold+8,2
mov ax,shop1[4*20].profit
invoke f2t10
invoke TextOut,hdc,XX+5*XX_GAP,YY+6*YY_GAP,offset buf,2
```

Display ret
 endp

Rate proc item_addr:DWORD ;入口参数为商品首地址
 push eax
 push ebx
 push ecx
 push edx
 push edi
 mov esi,item_addr
 add esi, 10
 mov ecx, [esi+6];已售数量

汇编语言程序设计实验报告

```
and ecx, 0000ffffh ;在实验三基础上, 因为已售数量为 dw, 所以将 ecx 高 16 为置 0, 以下同理
mov eax, ecx ;已售数量存入 eax
mov ecx, [esi+2] ;销售价
and ecx, 0000ffffh
imul eax, ecx ;销售价*已售数量
imul eax, 100
mov ecx, [esi] ;将进货价存入 ecx
and ecx, 0000ffffh
mov ebx, ecx ;将进货价存入 ebx 中
mov ecx, [esi+4] ;进货总数存入 ecx
and ecx, 0000ffffh
imul ebx, ecx ;进货价*进货总数
cdq
idiv ebx ;算出利润放入 eax 中
sub eax, 100 ;简化运算过程
mov pro1, eax
sub esi, 10 ;保证 esi 指向的是商品字符段
mov edi, esi
add edi, 100
xor eax, eax
add edi, 10
mov ecx, [edi+6] ;已售数量
and ecx, 0000ffffh
mov eax, ecx ;已售数量存入 eax
mov ecx, [edi+2] ;销售价
and ecx, 0000ffffh
imul eax, ecx ;销售价*已售数量
imul eax, 100
mov ecx, [edi] ;将进货价存入 ecx
and ecx, 0000ffffh
mov ebx, ecx ;将进货价存入 ebx 中
mov ecx, [edi+4] ;进货总数存入 ecx
and ecx, 0000ffffh
imul ebx, ecx ;进货价*进货总数
cdq
idiv ebx ;算出利润放入 eax 中
sub eax, 100 ;简化运算过程
;下面计算平均利润率
add eax, pro1 ;商品 1 的利润
sar eax, 1
mov [esi+18], ax;esi 为重定位到 shop1 中的商品地址
pop edi
pop edx
pop ecx
pop ebx
pop eax
ret
Rate endp

f2t10 proc ;将 ax 中存放的值转换成 10 进制
push ebx
push esi
lea esi, buf
cmp dx, 32
jne b
movsx eax, ax
b: or eax, eax
```

汇编语言程序设计实验报告

```
        jns plus
        neg eax
        mov byte ptr [esi], '-'
        inc esi
plus:    mov ebx, 10
        invoke radix
        mov byte ptr [esi], ' '
        pop esi
        pop ebx
        ret
f2t10   endp
radix   proc
        push ecx
        push edx
        xor ecx, ecx
lop1:    xor edx, edx
        div ebx
        push dx
        inc ecx
        or  eax, eax
        jnz lop1
lop2:    pop ax
        cmp al, 10
        jb  11
        add al, 7
11:      add al, 30h
        mov [esi], al
        inc si
        loop lop2
        pop edx
        pop ecx
        ret
radix   endp
end Start
```

3.1.4 实验步骤

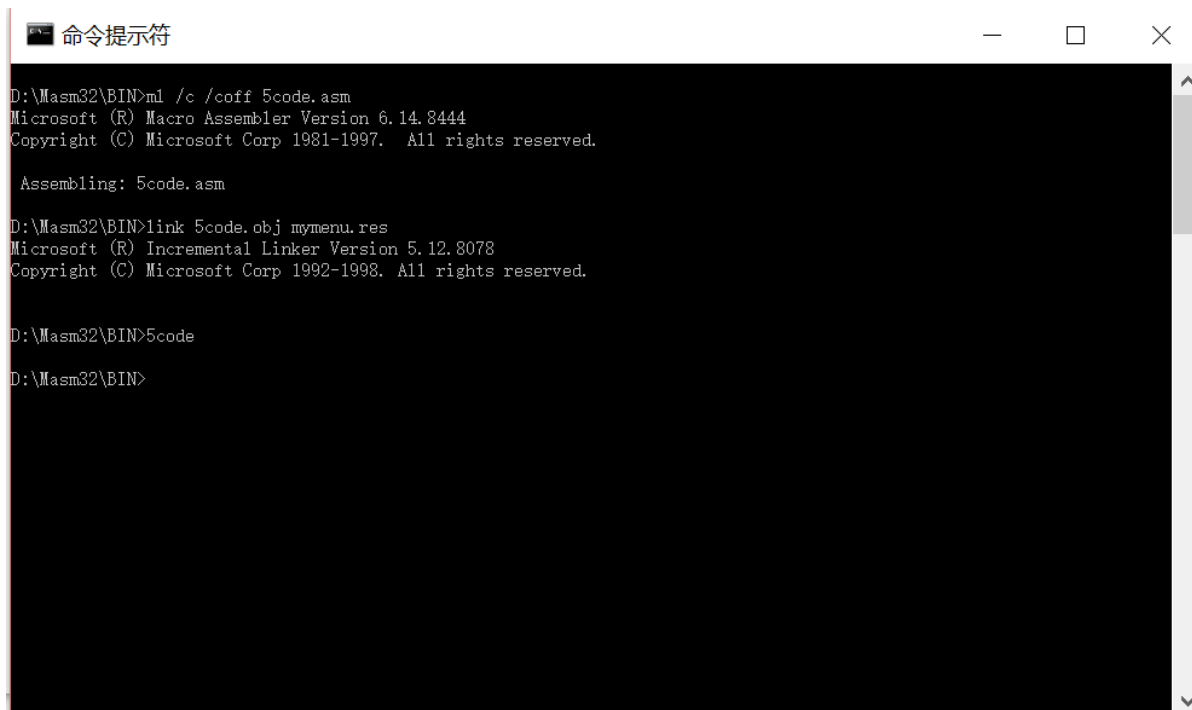
- 1.准备上机实验环境，编辑，汇编，连接文件。
- 2.运行程序，观察界面是否符合要求。
- 3.运行具体功能，观察功能是否能正确实现。在观察计算利润率功能是否正确时，打开 TD32 观察。
- 4.完成思考题。

3.1.5 实验记录与分析

- 1.实验环境条件： WINDOWS 10； MASM32 下 ML.exe,Link.exe,RC.exe; TD32.EXE。
- 2.编译文件没有问题，但是在连接时出现了问题。
错误提示为“未解析的外部符号”，这让我想到了是不是库的地址没有写成相对地址，之后改掉了这个错误后，连接成功了。
- 3.运行程序，观察窗口是否符合任务功能以的要求。

汇编语言程序设计实验报告

编译，连接文件操作如下：



```
D:\Masm32\BIN>ml /c /coff 5code.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: 5code.asm

D:\Masm32\BIN>link 5code.obj mymenu.res
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

D:\Masm32\BIN>5code

D:\Masm32\BIN>
```

图 3.1.2 编译，连接

窗口如下图，发现达到任务要求。

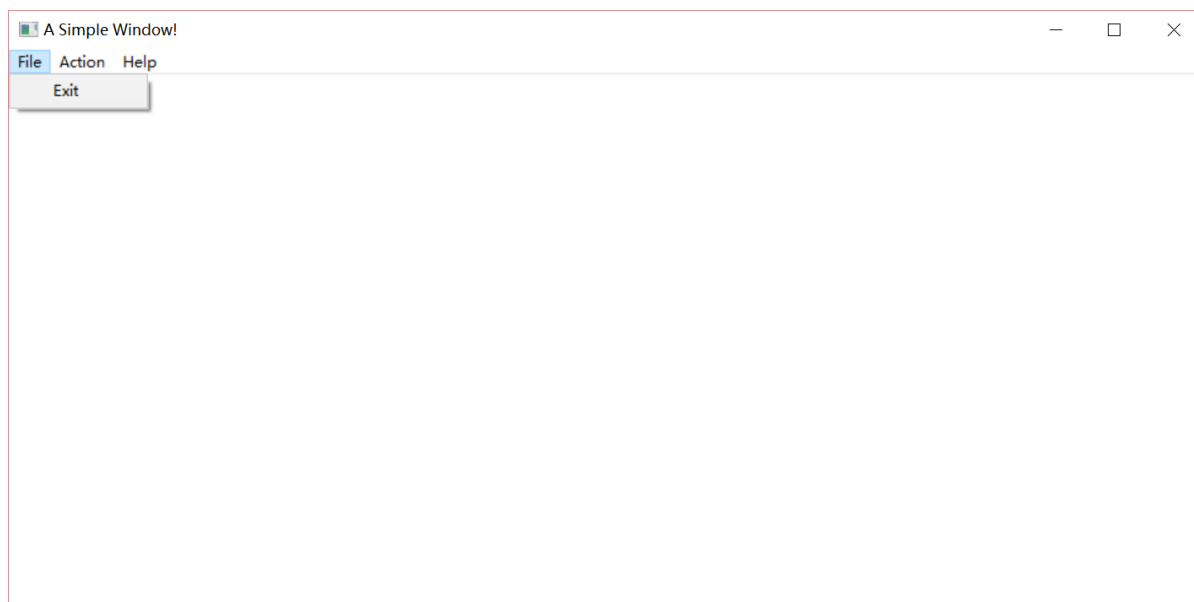


图 3.1.3 窗口

汇 编 语 言 程 序 设 计 实 验 报 告

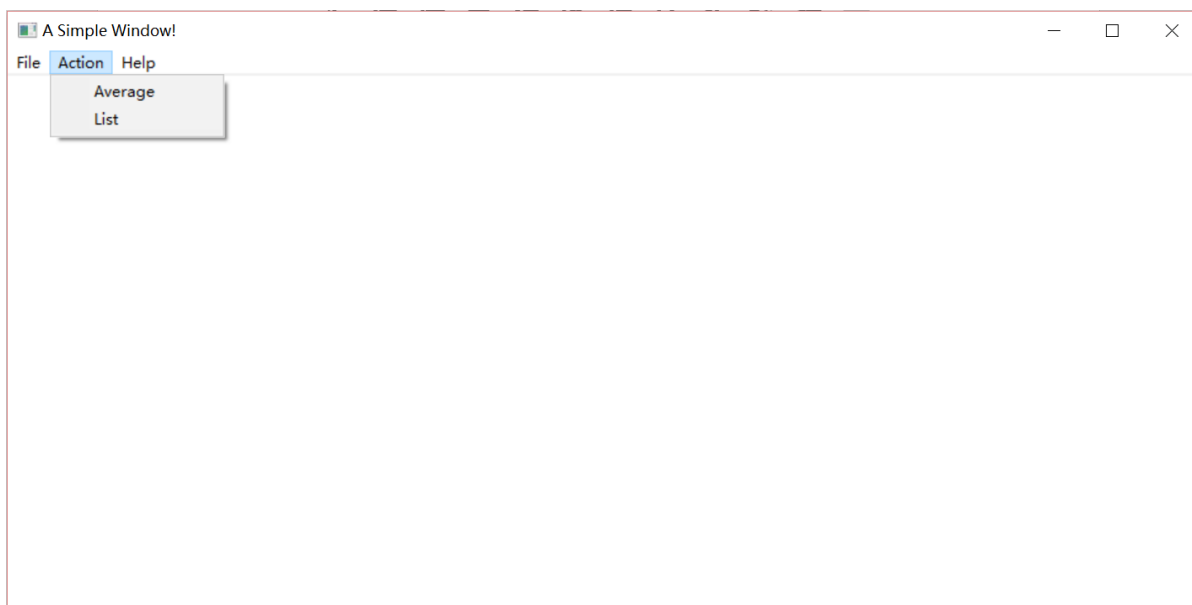


图 3.1.4 窗口

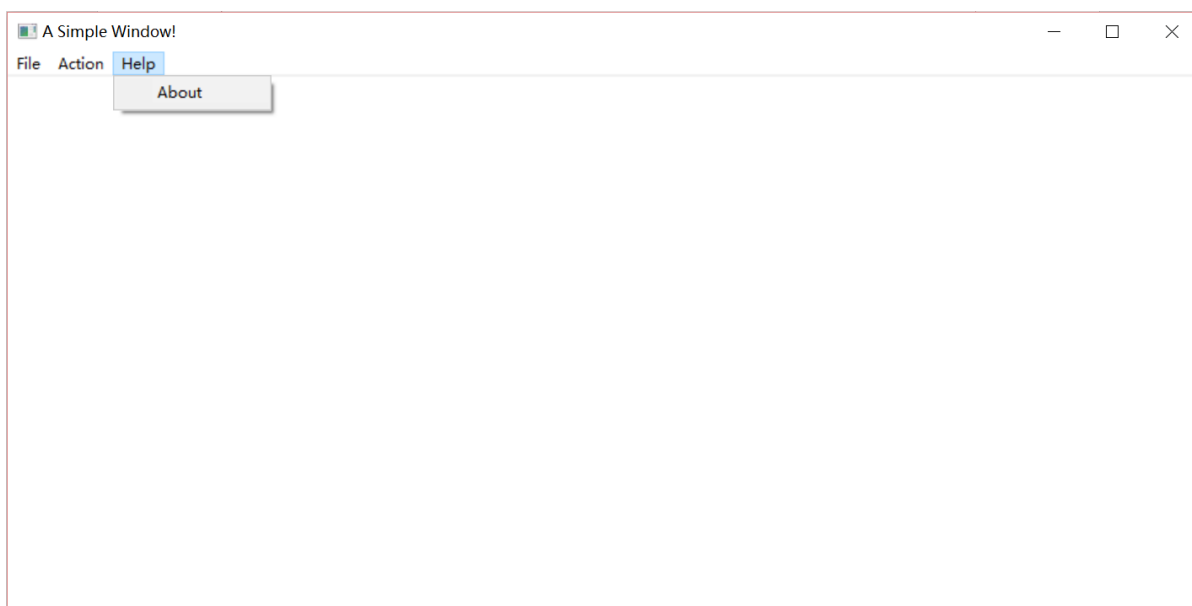


图 3.1.5 窗口

(1) 点击 Help 菜单栏下 About 按钮，发现显示一个关于自己信息的窗口。（在本人电脑上需要点击后按下 Alt 键才能弹出窗口）

汇编语言程序设计实验报告

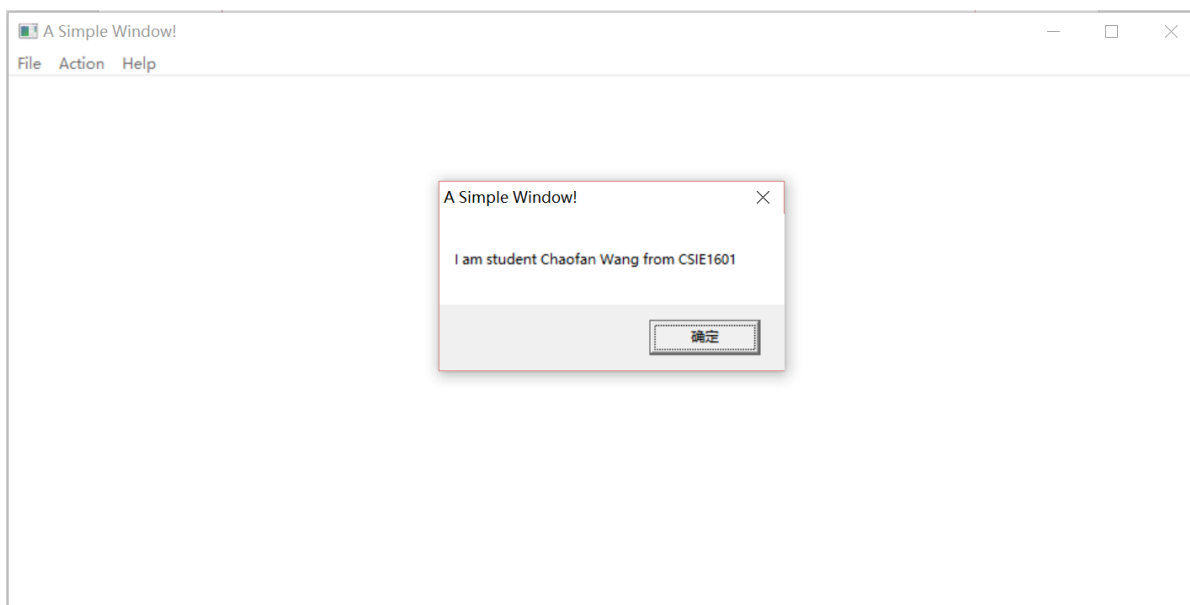


图 3.1.6 显示自己信息的窗口

4.运行 Action 菜单栏下功能。

(1) 在没有计算平均利润率之前，运行 List 功能，利润率显示为 0。

goods name	cost	price	total	sold	profit
pen	2	5	50	40	0
book	10	15	20	15	0
kindle	58	170	20	11	0
bottle	16	40	25	15	0
switch	108	190	30	20	0

图 3.1.7 没有计算平均利润率前 List 显示信息

(2) 计算平均利润率后再次运行 List 功能，发现利润率正确计算。

汇编语言程序设计实验报告

A Simple Window!					
File Action Help					
shop1					
goods name	cost	price	total	sold	profit
pen	2	5	50	40	85
book	10	15	20	15	9
kindle	58	170	20	11	47
bottle	16	40	25	15	20
switch	108	190	30	20	31

图 3.1.8 计算平均利润率后 List 显示信息

(3) 点击 Exit 按钮后，程序退出。

(4) 使用 TD32 观察计算结果，检测是否正确计算利润率。在进入计算子程序前设置断点，之后单步调试到最后结果。（这次观察的结果为 pen 的计算结果）

计算结果存放在 ax 中，下图中 ax 值为 55h，经过转换成十进制发现是 85，计算结果正确。

```
命令提示符 - td32 5code

File Edit View Run Breakpoints Data Options Window Help

-CPU Pentium Thread #629ds:306F = 6F620000

:00401629 D1F8 sar eax,1 eax 00000055 c=0
:0040162B 66894612 mov [esi+12],ax ebx 000000F0 z=0
:0040162F 5F pop edi ecx 00000050 s=0
:00401630 5A pop edx edx 00000000 b=0
:00401631 59 pop ecx esi 0040305D p=1
:00401632 5B pop ebx edi 004030CB a=0
:00401633 58 pop eax ebp 0019FD6C i=1
:00401634 C9 leave esp 0019FD58 d=0
:00401635 C20400 ret 0004 ds 002B
:00401638 53 push ebx es 002B
:00401639 56 push esi fs 0053
:0040163A 8D3580314000 lea esi,[00403180] gs 002B
:00401640 6683FA20 cmp dx,0020 ss 002B
                                cs 0023
                                eip 0040162B
:00000000 77??                                :0019FD5C 002E09BA
:00000008 77??                                :0019FD58 00401112
:00000010 77??
:00000018 77??
```

图 3.1.9 利用 TD32 观察计算结果

5. 思考题

(1) 安装 MASM32 软件包，观察 MASM32 软件包目录结构和环境配置。

在教学网站上下载的是解压版。MASM32 软件包目录用到的包括 BIN, INCLUDE, LIB, 其中 BIN 包含了 ML. EXE, LINK. EXE, RC. EXE, TD32. EXE 等工具，INCLUDE 和 LIB 则是包含各种库。

汇编语言程序设计实验报告

(2) 试对\masm32\EXAMPLE1\3DFRAMES\下的例子, 进行汇编、连接、运行和调试(TD32.EXE)。观察 WIN32 执行程序代码的特点和执行流程。体会基于窗口的应用程序所包含的四个部分之间的衔接关系。

这个例子就是一个框架, 没有任何功能。由 TD32 看到, WIN32 程序主要是利用了各种调用, 以及在代码段反汇编出来后会发现很多 push 语句, 这和上次看到的 C 语言反汇编代码类似。在主代码段中调用 WinMain, 然后再在 WinMain 中调用其他函数。

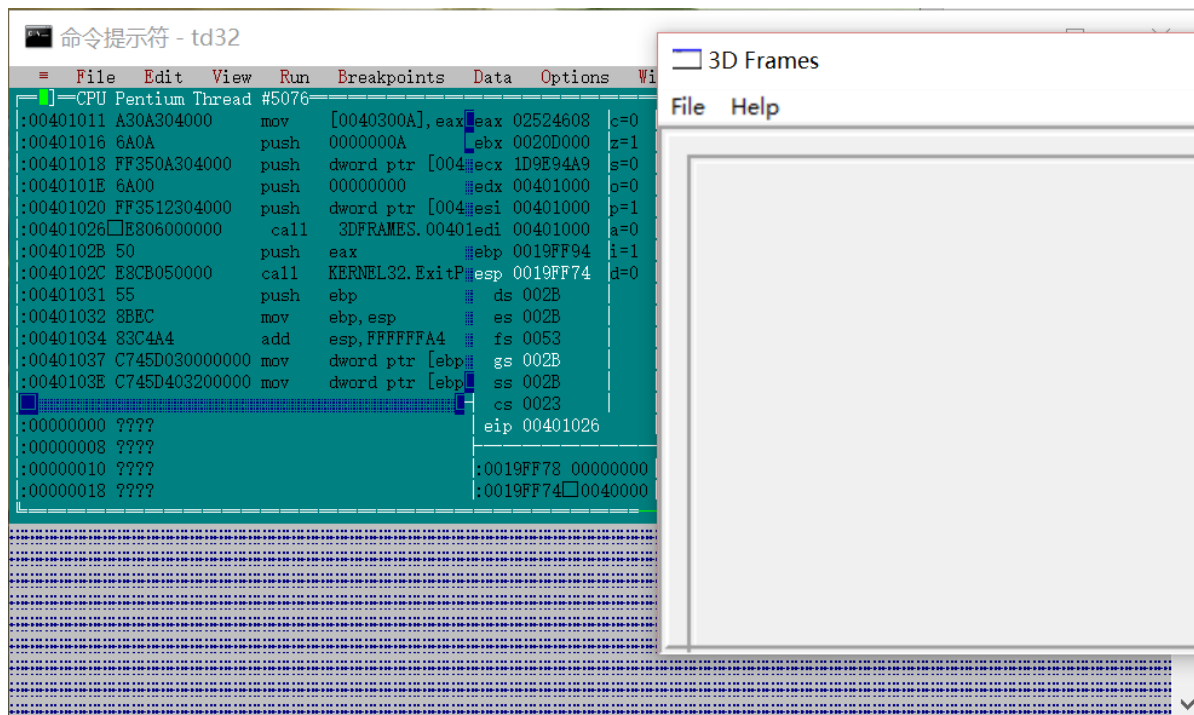


图 3.1.10 利用 TD32 观察例子

(3) 观察 TD32 与 16 位 TD 的异同。

由上面几张图可以看到, TD32 里面反汇编语句凡是出现了立即数的, 都是 32 位, 然后调用函数时, 会先出现 EXE 文件名, 然后显示函数所在位置。同时在寄存器显示区, 除了段寄存器, 其他寄存器全部扩展成 32 位寄存器显示。多了 fs, gs 段寄存器。同时在数据区看不到无关数据, 在堆栈区数据也是以 32 位显示。

(4) 调试 WIN32 程序与 16 位段程序的主要差异是什么?

调试 WIN32 程序的话, TD32 数据段没有显示, 在代码段会有很多其他调用函数, 所以要找到你自己写的函数有点困难。还有就是寄存器观察只能观察到 32 位寄存器的值, 你要对自己的函数有很清楚的了解, 才知道寄存器里的值代表着什么。

(5) 尝试使用一下汇编语言程序的源码级调试工具和方法, 与非源码级调试做个对比。

源码级调试更加接近高级语言的调试, 对我们来观看代码有帮助, 而非源码级调试则接近机器语言, 这有助于我们深入了解计算机系统是如何工作的。

(6) 用 TD32 观察代码区或数据区时, 若所观察的地址范围不是与被调试程序相关的区间, 则对应内存中的数据会因为被系统保护了而读不出来(将用?代替)。请通过修改偏移地址来改变观察的区间, 记录此现象。

使用 goto 功能, 转到了相关数据区, 发现正确显示数据。

汇编语言程序设计实验报告

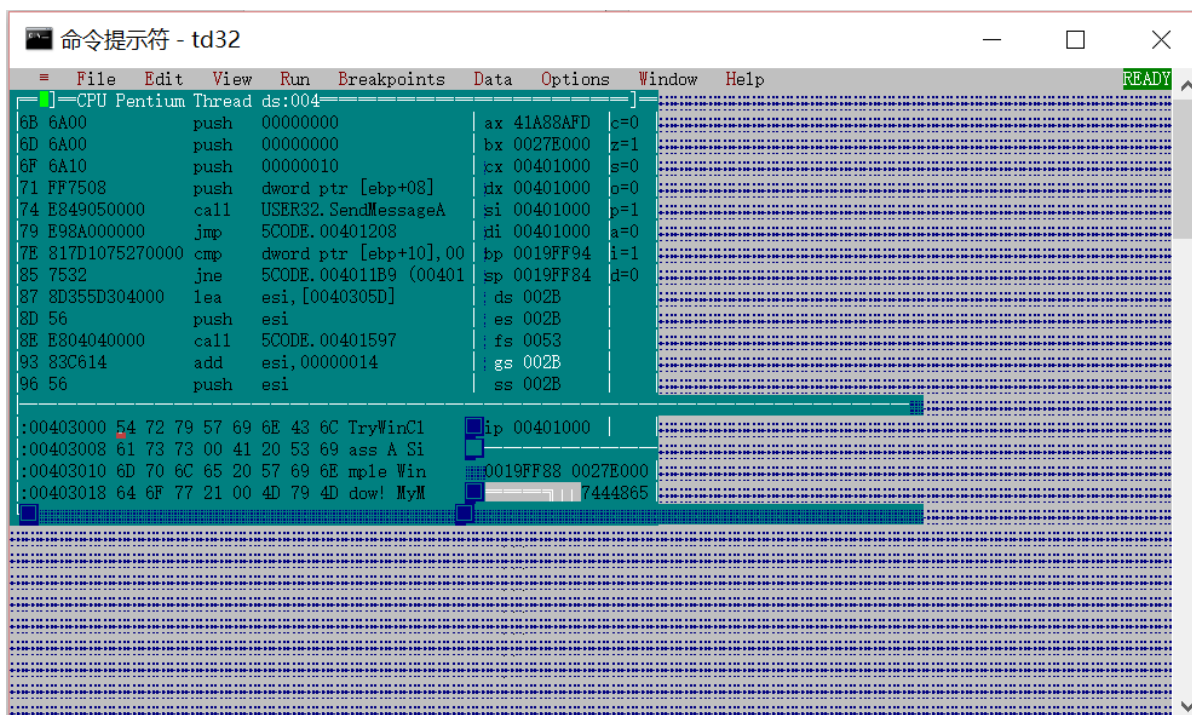


图 3.1.11 利用 TD32 观察数据区

(7) 编写和处理简单资源脚本，装入菜单，观察效果。

这个已经在实验记录里操作完成了。

(8) 观察收到的消息，记录每个菜单项或按键等操作所对应的消息信息。

在编写代码时已经定义了宏变量。

(9) 比较 DOS、Windows 输出方式，观察 Win32 程序的几种字符串输出方式所用函数的原型。

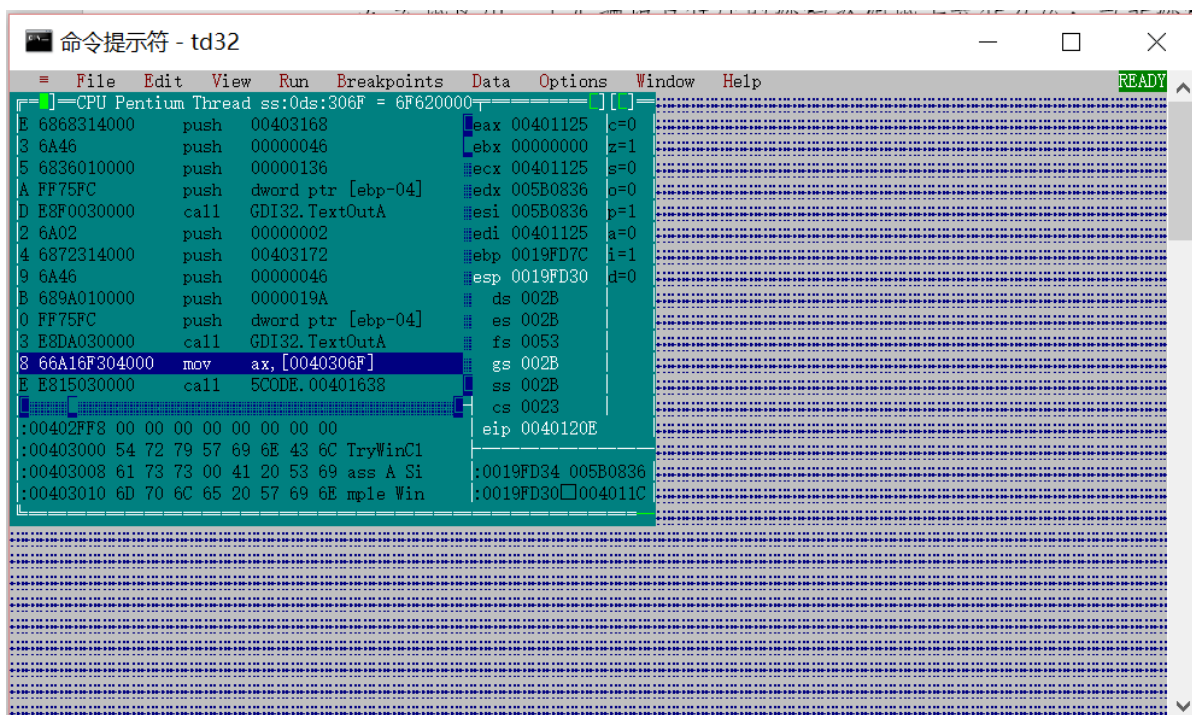
DOS 是利用九号调用输出字符串。本次实验使用的函数为 TextOut。函数原型为：

```
BOOL TextOut(  
HDC hdc, // 设备描述表句柄  
int nXStart, // 字符串的开始位置 x 坐标  
int nYStart, // 字符串的开始位置 y 坐标  
LPCTSTR lpString, // 字符串  
int cbString // 字符串中字符的个数  
);
```

(10) 观察结构变量的利润率等字段的偏移，体会结构变量优点。

发现反汇编语句将 `shop1[0*20].profit` 变成了直接寻址。而在汇编语句中的这种书写方式，可以不用去考虑偏移量，这很像高级语言，帮助我们节省了很多时间，也让代码可读性增强。

汇编语言程序设计实验报告



The screenshot shows the TD32 debugger window titled "命令提示符 - td32". The menu bar includes File, Edit, View, Run, Breakpoints, Data, Options, Window, and Help. The status bar at the top indicates "CPU Pentium Thread ss:0ds:306F = 6F620000". The main window displays assembly code with the following instructions highlighted:

```
6868314000 push 00403168
3 6A46 push 00000046
5 6836010000 push 00000136
A FF75FC push dword ptr [ebp-04]
D E8F0030000 call GDI32.TextOutA
2 6A02 push 00000002
4 6872314000 push 00403172
9 6A46 push 00000046
B 689A010000 push 0000019A
0 FF75FC push dword ptr [ebp-04]
3 E8DA030000 call GDI32.TextOutA
8 6A16F304000 mov ax, [0040306F]
E E815030000 call 5CODE.00401638
```

The register window on the right shows the following values:

Register	Value
eax	00401125
ebx	00000000
ecx	00401125
edx	005B0836
esi	005B0836
edi	00401125
ebp	0019FD7C
esp	0019FD30
ds	002B
es	002B
fs	0053
gs	002B
ss	002B
cs	0023
eip	0040120E

The instruction pointer (eip) is 0040120E. The stack pointer (esp) is 0019FD30. The instruction at address 0040120E is highlighted in blue.

图 3.1.12 利用 TD32 观察利润率字段偏移

(11) 观察简化段的效果。

简化段定义后，先是执行 push 00000000 后，再调用第一个函数。这个简化段之后的效果和完整段定义相同。

(12) 观察 Invoke 语句翻译成机器码后的特点，记录参数压栈顺序。

由上图可知，invoke 语句翻译成机器码后，先是把参数压栈，然后利用 call 指令调用函数。参数的压栈顺序为：从最后一个参数到第一个参数。

汇编语言程序设计实验报告

4 总结与体会

在本次实验中，我熟悉了 WIN32 程序的设计和调试方法。本次实验菜单的框架由老师给出，在仔细研究了例子的程序后，我自己结合上课所讲的方法，完成了本次实验。

在这次任务中，虽然要实现的功能十分简单。但由于是初次编写 win32 程序，还是遇到了一些问题。在编译过程中没有出现什么错误，但是在连接文件时却出现了问题，经过分析，发现时在导入库时没有给出路径，导致程序找不到相应文件。当修改了我的代码后，连接没有问题了。

一开始看框架里的代码时，觉得有点难以理解，但是在查阅了各种函数的用法之后，我发现这些代码其实是很简单的。在处理点击按钮的信息时，就要加入自己的代码了。通过之前了解到的知识，编写起来还是很简单的。这和之前编写 16 位程序的时候主要的区别就是感觉自己更像是在使用高级语言来写程序一样，invoke 调用函数的方法也和高级语言相类似。一些判断语句也使编写 win32 程序变得简单起来。

在定义数据时，使用了结构变量，这和 C 语言的结构体相类似，使用结构变量代替变址寻址，可以让我们在编写程序时减少考虑偏移地址的问题，而将精力放在解决问题上面，能简化我们的思维，同时这也能让我们的代码可读性增强，能让我们回过头来看代码时清楚地知道我们在写什么。

同时，在 32 位程序里使用的都是 32 位寄存器，这就对数据的长度有了要求，可以将数据定义成 dd 双字类型，如果定义成 dw 字类型，就要在子程序里做相应改变，不然就会有错误。有可能多读了一些内容，也可能将一些内容写入了其他数据区里。这都是意料之外的错误。

通过这次实验，我了解到机器语言，汇编语言，高级语言智联有着深深的联系，比如在传参数的时候，invoke 语句使用栈的方式，在 C 语言中也是如此，而且参数入栈的顺序也相同，这说明一些基本方法在汇编语言和高级语言里是通用的，高级语言都可以编译成汇编语言。

在本次试验，最大的收获就是掌握了 win32 程序的设计和调试方法，win32 程序的设计可能比 16 位程序设计多了一些便利之处。在最后使用 TD32 调试程序时，发现基本调试方法都是一样的，只是在 TD32 里面，寄存器都是使用 32 位，你得清楚知道你的程序是怎么写的，然后根据这些 32 位寄存器判断你需要查看的数据是什么，同时与调试无关的数据 TD32 会以 ? 形式出现在数据区，这要求你使用 goto 功能跳到相应数据区查看数据。总之，通过本次实验，主要是熟悉了 win32 程序的设计，调试，同时更加深入地了解汇编语言和高级语言之间的联系。

汇 编 语 言 程 序 设 计 实 验 报 告

参考文献

- [1] 王元珍、韩宗芬、曹忠升.《80X86 汇编语言程序设计》. 华中科技大学出版社:2005 年 04 月
- [2] 汇编语言教学网站-》资料下载-》案例-》win32 程序、编译和连接
- [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