

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	设计思想及存储单元分配	2
3.2	流程图	3
3.3	源程序	5
3.4	实验步骤	12
3.5	实验记录与分析	12
4	总结与体会.....	17
	参考文献	18

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

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推荐度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	Recommendation	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 5.1 所示。点菜单 Action 下的选项 Recommendation、List 将分别实现计算推荐度或显示 SHOP 所有商品信息的功能（详见功能二的描述）。

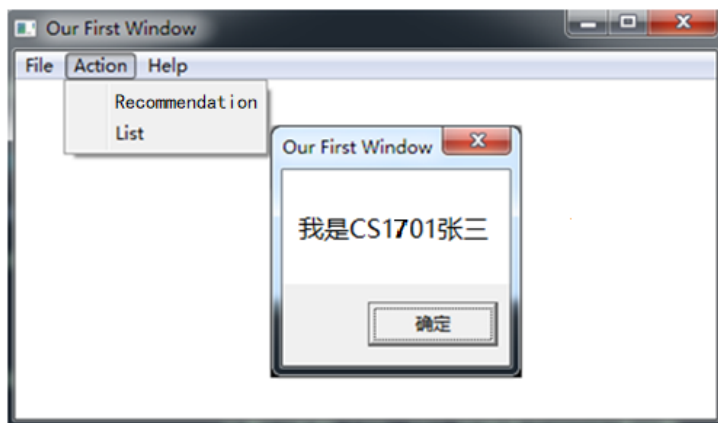


图 1-1 菜单示例

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。

(1) 点菜单项 Recommendation 时，按照实验三的方法计算所有商品的推荐度。用 TD32 观察计算结果。

(2) 点菜单项 List 时，要求能在窗口中列出 SHOP 的所有商品的信息。具体显示格式自行定义，可以参照图 5.2 的样式（不要求用中文）。

汇编语言程序设计实验报告



商品名称	折扣	进货价	销售价	进货总数	已售数量	推荐度
PEN	4	5	8	50	40	21
NOTE	8	1	2	100	50	8
电风扇	5	30	50	30	20	15

图 1-2 商品信息显示示意图

3 实验过程

3.1 设计思想及存储单元分配

1. 编写资源文件（.rc 文件），装入菜单。具体做法为在 File 主菜单下面装入 Exit，Action 主菜单下面装入 Recommendation 和 List，Help 主菜单下面装入 About。（用 POPUP 语句装入主菜单，用 MENUITEM 语句装入下拉选项）在源文件中，使用 invoke LoadMenu, hInst, 600 载入菜单。

2. 源程序用到的是 WIN32 的标准框架，即以下四个部分：主程序、窗口主程序、窗口消息处理程序以及用户处理程序。设计思想是先编写数据段，然后按照 WIN32 框架编写代码段。具体操作如以下 5 点所示：

1) 在数据段中录入所有商品的信息以及其他需要用到的变量、数据以及输出的字符串。

2) 主程序：先得到应用程序的句柄，获取参数指令，然后调用窗口主程序。

3) 窗口主程序：先将窗口初始化，设定窗口风格，获取窗口过程的入口地址；然后再装载菜单和图标等资源；然后进入消息循环，判断是否返回（eax 是否为 0），若不返回，则从键盘接收并转换为消息，然后将消息分发到窗口消息处理程序，若要返回则设置返回码。

4) 窗口消息处理程序：接收并判断收到信息的种类，执行功能。若为毁坏窗口的信息，则退出。其余情况下，若为 IDM_File_Exit 消息，则关闭窗口；若为 IDM_Acion_Recommendation 消息，则分别调用实验三 CALCREDER 计算所有商品的推荐度；若为 IDM_Action_List 信息，则调用 Display 子程序输出所有商品信息；若为 IDM_Help_About 信息，则在小窗口中显示个人信息。如果不是本程序要处理的信息，作缺省处理。

5) Display 子程序设计思想：首先设定 List 中的每一项之间的行距 x_jump 与列距 y_jump，设定 List 的边框间距 x_axis 和 y_axis。然后用 TextOut 函数输出 List 中的值，遇到要输出数字的场合我们需要利用 TRANS2 子程序将其变为十进制字符串输出。

3.2 流程图

1) 主程序流程图如下：

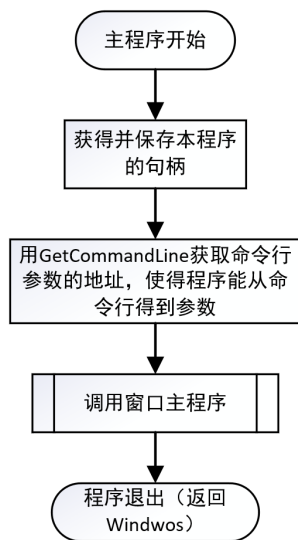


图 3-1 主程序流程图

2) 窗口主程序流程图如下：

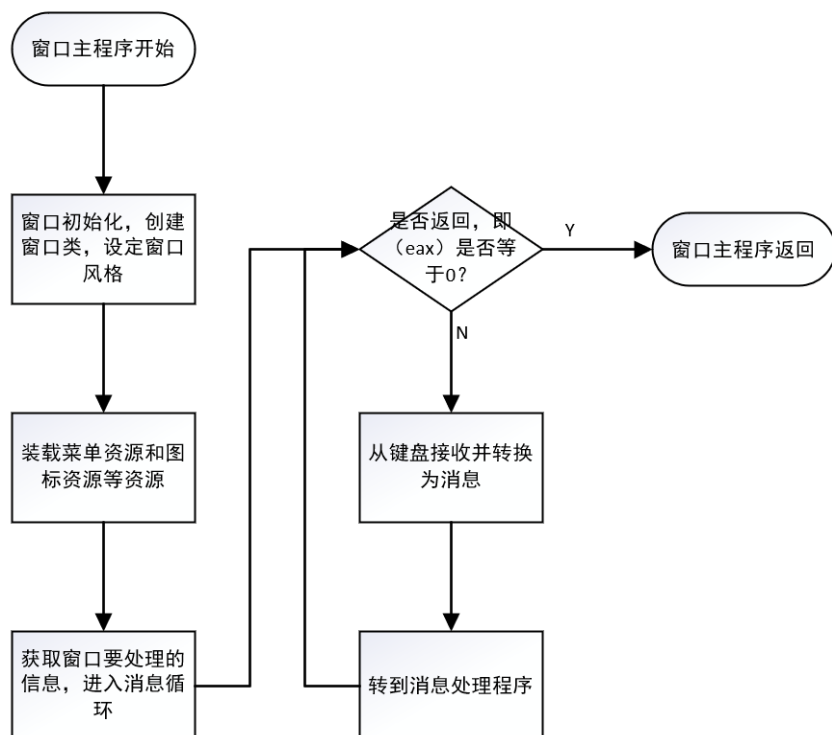


图 3-2 窗口主程序流程图

汇编语言程序设计实验报告

3) 窗口处理程序流程图如下:

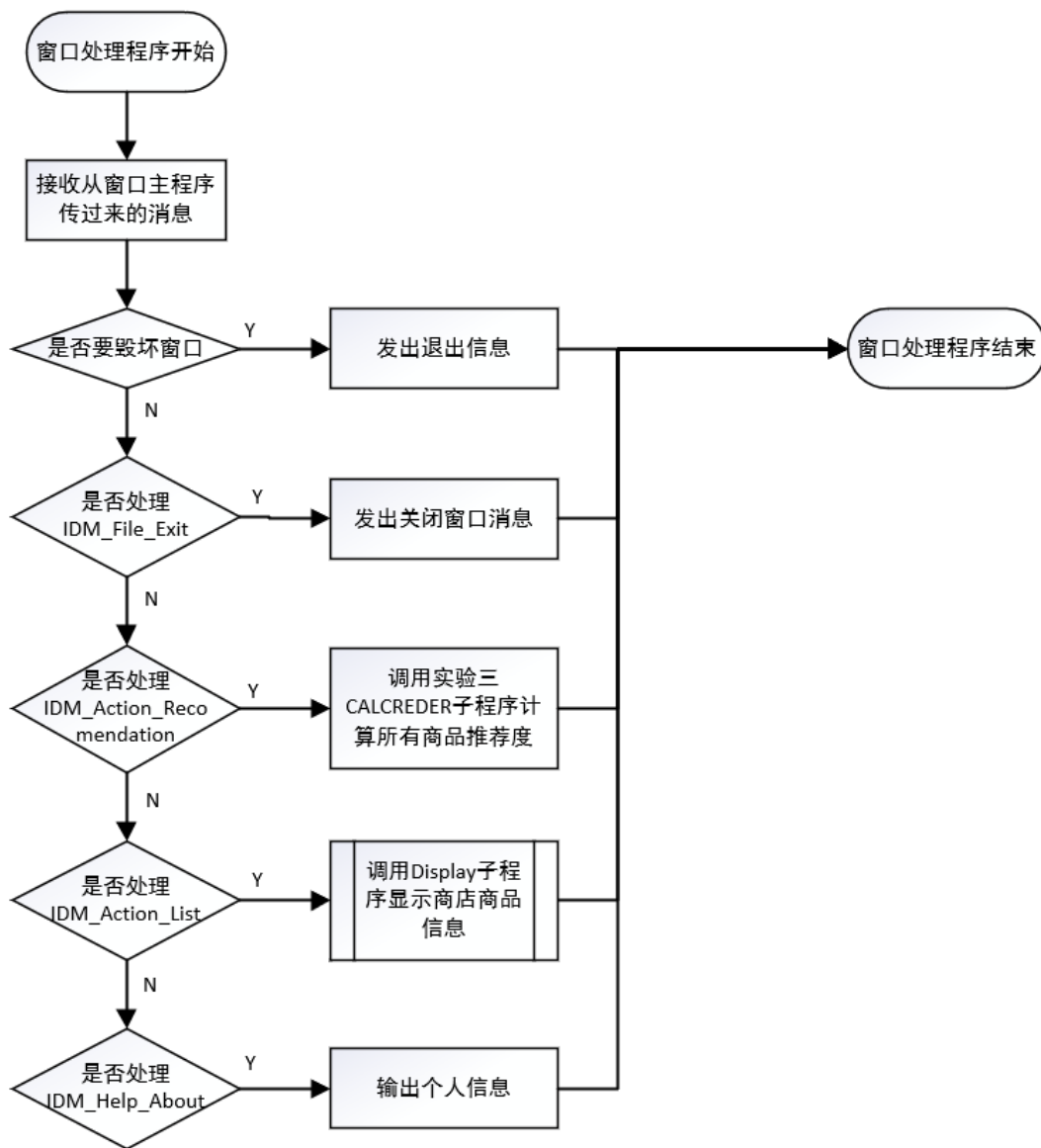


图 3-3 窗口处理程序流程图

汇编语言程序设计实验报告

4) Display 子程序流程图如下:

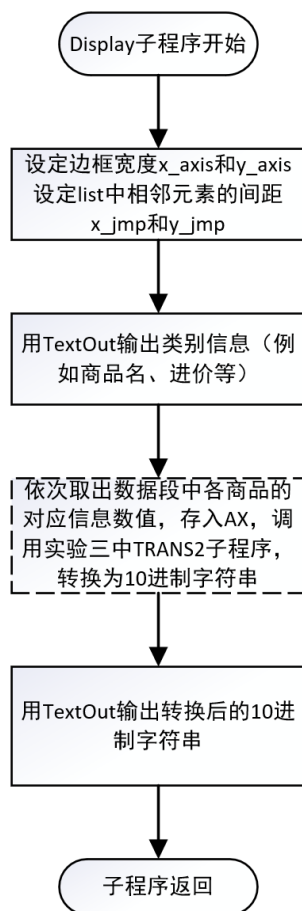


图 3-4 Display 子程序流程图

3.3 源程序

1) .rc 文件程序如下:

```
#define IDM_File_Exit 10001
#define IDM_Action_Recommendation 10101
#define IDM_Action_List 10102
#define IDM_Help_About 10201

MyMenu MENU
BEGIN
    POPUP "&File" ;装载 File
    BEGIN
        MENUITEM "&Exit", IDM_File_Exit
    END
    POPUP "&Action" ;装载 Action
    BEGIN
        MENUITEM "&Recommendation", IDM_Action_Recommendation
        MENUITEM "&List", IDM_Action_List
    END
    POPUP "&Help" ;装载 Help
    BEGIN
        MENUITEM "&About", IDM_Help_About
```

汇编语言程序设计实验报告

END
END

2) .inc 文件程序如下:

```
IDM_File_Exit equ 10001
IDM_Action_Recommendation equ 10101
IDM_Action_List equ 10102
IDM_Help_About equ 10201
```

3) .asm 文件程序如下:

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

WinMain proto:dword,:dword,:dword,:dword
WndProc proto:dword,:dword,:dword,:dword
Display proto:dword
CALCREFER proto
TRANS2 proto

; include 5.inc
; include user32.inc
; include gdi32.inc
; include kernel32.inc
; include shell32.inc
; include windows.inc

; includelib user32.lib
; includelib gdi32.lib
; includelib kernel32.lib
; includelib shell32.lib

include \masm32\task5\5.inc
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
include \masm32\include\gdi32.inc
include \masm32\include\shell32.inc

includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\gdi32.lib
includelib \masm32\lib\shell32.lib

item struct
    iname db 10 dup(0)
    discount db 0
    input dw 0
    selling dw 0
    initem dw 0
    sellyet dw 0
    reco dw 0
item ends
```

汇编语言程序设计实验报告

```
.data
    out_name db 'item_name',0
    out_discount db 'discount',0
    out_inprice db 'in_price',0
    out_sellprice db 'sell_price',0
    out_initem db 'in_item',0
    out_sell db 'sell_item',0
    out_rec db 'recommendation',0
    out_about db 'ACM1701 WL DING',0 ;about 的内容
    windowtype db 'TryWinClass',0 ;窗口类名
    windowname db 'shop of dwl',0 ;窗口抬头
    commodity item<'PEN',10,10,25,120,80,?> ;折扣,进货价,售货价,进货总数,已售数量,推荐度
    item<'BOOK',9,12,30,25,5,?>
    item<'NOTE',8,30,50,40,20,?>
    item<'PENCIL',10,5,8,10,5,?>
    BUF1 db 10 dup(0)
    TEMP1 dd 0
    TEMP2 dd 0
    TEMP3 dd 0
    N equ 5
    hInstance dd 0
    CommandLine dd 0
    menuname db 'MyMenu',0
```

.code

start:

```
    invoke GetModuleHandle, NULL;获得并保存本程序的句柄
    mov hInstance, eax
    invoke GetCommandLine
    mov CommandLine, eax
    invoke WinMain, hInstance, NULL, CommandLine, SW_SHOWDEFAULT;调用窗口主程序
    invoke ExitProcess, eax;退出本程序, 返回 Windows
```

;窗口主程序

```
WinMain proc hInst:dword, hPreInst:dword, CmdLine:dword, CmdShow:dword
    local wc:WNDCLASSEX;创建主窗口时所需要的信息由该结构说明
    local msg:MSG;消息结构变量用于存放获取的信息
    local hWnd:HWND;存放窗口句柄
```

;给 WNDCLASSEX 结构变量 wc 的各字段赋值

```
    mov wc.cbSize, SIZEOF WNDCLASSEX;WNDCLASSEX 结构类型的字节数
    mov wc.style, CS_HREDRAW or CS_VREDRAW;窗口风格
    mov wc.lpfnWndProc, OFFSET WndProc;本窗口过程的入口地址
    mov wc.cbClsExtra, NULL;不用自定义数据则不需要 OS 预留空间, 置 NULL
    mov wc.cbWndExtra, NULL;同上
    push hInst;本应用程序句柄->wc.hInstance
    pop wc.hInstance
    mov wc.hbrBackground, COLOR_WINDOW+1;窗口的背景设为白色
    mov wc.lpszMenuName, OFFSET menuname;菜单名
    mov wc.lpszClassName, OFFSET windowtype;窗口类名
    invoke LoadIcon, NULL, IDI_APPLICATION;装入系统默认图标
    mov wc.hIcon, eax;保存图标的句柄
    mov wc.hIconSm, 0;窗口不带小图标
    invoke LoadCursor, NULL, IDC_ARROW;装入系统默认的光标
    mov wc.hCursor, eax;保存光标句柄
```


汇编语言程序设计实验报告

```
invoke RegisterClassEx, ADDR wc;注册窗口类
invoke CreateWindowEx, NULL, ADDR windowtype, ;建立 windowtype 类窗口
ADDR windowname, ;窗口标题地址
WS_OVERLAPPEDWINDOW+WS_VISIBLE, ;创建可显示的窗口
CW_USEDEFAULT, CW_USEDEFAULT, ;窗口左上角坐标默认值
CW_USEDEFAULT, CW_USEDEFAULT, ;窗口宽度, 高度默认值
NULL, NULL, ;无父窗口, 无菜单
hInst, NULL;本程序句柄, 无参数传递给窗口
mov hWnd, eax;保存窗口的句柄
invoke ShowWindow, hWnd, SW_SHOWNORMAL
invoke UpdateWindow, hWnd
StartLoop:;进入消息循环
invoke GetMessage, ADDR msg, NULL, 0, 0;从 windows 获取消息
cmp eax, 0;如果 eax 不为 0, 则转换并分发消息
je ExitLoop;如果 eax 为 0, 则转 exitloop
invoke TranslateMessage, ADDR msg;从键盘接受按键并转换为消息
invoke DispatchMessage, ADDR msg;将消息分发到窗口的消息处理程序
jmp StartLoop;再循环获取消息
ExitLoop:
    mov eax, msg.wParam;设置返回码
    ret
WinMain endp

;窗口消息处理程序
WndProc proc hWnd:dword, uMsg:dword, wParam:dword, lParam:dword
local hdc:HDC;存放设备上下文句柄
.if uMsg==WM_DESTROY;收到的是销毁窗口信息
    invoke PostQuitMessage, NULL;发出退出消息
.elseif uMsg==WM_KEYDOWN
.    if wParam==VK_F1
        invoke MessageBox, hWnd, ADDR out_about, ADDR windowname, 0
    .endif
.elseif uMsg==WM_COMMAND
.    if wParam==IDM_File_Exit
        invoke SendMessage, hWnd, WM_CLOSE, 0, 0
.    elseif wParam==IDM_Action_Recommendation
        mov ebx, offset commodity
        invoke CALCREFER
        mov ebx, offset commodity[1*21]
        invoke CALCREFER
        mov ebx, offset commodity[2*21]
        invoke CALCREFER
        mov ebx, offset commodity[3*21]
        invoke CALCREFER
.    elseif wParam==IDM_Action_List
        invoke Display, hWnd
.    elseif wParam==IDM_Help_About
        invoke MessageBox, hWnd, ADDR out_about, ADDR windowname, 0
    .endif
.else
    invoke DefWindowProc, hWnd, uMsg, wParam, lParam;不是本程序要处理的消息, 作其他缺省处理
    ret
.endif
mov eax, 0
ret
WndProc endp
```

汇编语言程序设计实验报告

```
;输出信息函数
Display proc hWnd:dword
    x_axis equ 8
    y_axis equ 10
    x_jump equ 100
    y_jump equ 30
    size_of equ sizeof(item)
    local hdc:HDC
    invoke GetDC, hWnd
    mov hdc, eax

    invoke TextOut, hdc, x_axis+0*x_jump, y_axis+0*y_jump, OFFSET out_name, 9
    invoke TextOut, hdc, x_axis+1*x_jump, y_axis+0*y_jump, OFFSET out_discount, 8
    invoke TextOut, hdc, x_axis+2*x_jump, y_axis+0*y_jump, OFFSET out_inprice, 8
    invoke TextOut, hdc, x_axis+3*x_jump, y_axis+0*y_jump, OFFSET out_sellprice, 10
    invoke TextOut, hdc, x_axis+4*x_jump, y_axis+0*y_jump, OFFSET out_initem, 7
    invoke TextOut, hdc, x_axis+5*x_jump, y_axis+0*y_jump, OFFSET out_sell, 9
    invoke TextOut, hdc, x_axis+6*x_jump, y_axis+0*y_jump, OFFSET out_rec, 14

    invoke TextOut, hdc, x_axis+0*x_jump, y_axis+1*y_jump, OFFSET commodity[0*21].iname, 3
    mov al, commodity[0*size_of].discount
    mov ah, 0
    invoke TRANS2
    invoke TextOut, hdc, x_axis+1*x_jump, y_axis+1*y_jump, OFFSET BUF1, 2
    mov ax, commodity[0*size_of].input
    invoke TRANS2
    invoke TextOut, hdc, x_axis+2*x_jump, y_axis+1*y_jump, OFFSET BUF1, 2
    mov ax, commodity[0*size_of].selling
    invoke TRANS2
    invoke TextOut, hdc, x_axis+3*x_jump, y_axis+1*y_jump, OFFSET BUF1, 2
    mov ax, commodity[0*size_of].initem
    invoke TRANS2
    invoke TextOut, hdc, x_axis+4*x_jump, y_axis+1*y_jump, OFFSET BUF1, 2
    mov ax, commodity[0*size_of].sellyet
    invoke TRANS2
    invoke TextOut, hdc, x_axis+5*x_jump, y_axis+1*y_jump, OFFSET BUF1, 2
    mov ax, commodity[0*size_of].reco
    invoke TRANS2
    invoke TextOut, hdc, x_axis+6*x_jump, y_axis+1*y_jump, OFFSET BUF1, 2

    invoke TextOut, hdc, x_axis+0*x_jump, y_axis+2*y_jump, OFFSET commodity[1*21].iname, 4
    mov al, commodity[1*size_of].discount
    mov ah, 0
    invoke TRANS2
    invoke TextOut, hdc, x_axis+1*x_jump, y_axis+2*y_jump, OFFSET BUF1, 1
    mov ax, commodity[1*size_of].input
    invoke TRANS2
    invoke TextOut, hdc, x_axis+2*x_jump, y_axis+2*y_jump, OFFSET BUF1, 2
    mov ax, commodity[1*size_of].selling
    invoke TRANS2
    invoke TextOut, hdc, x_axis+3*x_jump, y_axis+2*y_jump, OFFSET BUF1, 2
    mov ax, commodity[1*size_of].initem
    invoke TRANS2
    invoke TextOut, hdc, x_axis+4*x_jump, y_axis+2*y_jump, OFFSET BUF1, 2
    mov ax, commodity[1*size_of].sellyet
    invoke TRANS2
    invoke TextOut, hdc, x_axis+5*x_jump, y_axis+2*y_jump, OFFSET BUF1, 2
```

汇编语言程序设计实验报告

```
mov ax,commodity[1*size_of].reco
invoke TRANS2
invoke TextOut,hdc,x_axis+6*x_jump,y_axis+2*y_jump,OFFSET BUF1,2

invoke TextOut,hdc,x_axis+0*x_jump,y_axis+3*y_jump,OFFSET commodity[2*21].iname,4
mov al,commodity[2*size_of].discount
mov ah,0
invoke TRANS2
invoke TextOut,hdc,x_axis+1*x_jump,y_axis+3*y_jump,OFFSET BUF1,1
mov ax,commodity[2*size_of].input
invoke TRANS2
invoke TextOut,hdc,x_axis+2*x_jump,y_axis+3*y_jump,OFFSET BUF1,2
mov ax,commodity[2*size_of].selling
invoke TRANS2
invoke TextOut,hdc,x_axis+3*x_jump,y_axis+3*y_jump,OFFSET BUF1,2
mov ax,commodity[2*size_of].initem
invoke TRANS2
invoke TextOut,hdc,x_axis+4*x_jump,y_axis+3*y_jump,OFFSET BUF1,2
mov ax,commodity[2*size_of].sellyet
invoke TRANS2
invoke TextOut,hdc,x_axis+5*x_jump,y_axis+3*y_jump,OFFSET BUF1,2
mov ax,commodity[2*size_of].reco
invoke TRANS2
invoke TextOut,hdc,x_axis+6*x_jump,y_axis+3*y_jump,OFFSET BUF1,2

invoke TextOut,hdc,x_axis+0*x_jump,y_axis+4*y_jump,OFFSET commodity[3*21].iname,5
mov al,commodity[3*size_of].discount
mov ah,0
invoke TRANS2
invoke TextOut,hdc,x_axis+1*x_jump,y_axis+4*y_jump,OFFSET BUF1,2
mov ax,commodity[3*size_of].input
invoke TRANS2
invoke TextOut,hdc,x_axis+2*x_jump,y_axis+4*y_jump,OFFSET BUF1,2
mov ax,commodity[3*size_of].selling
invoke TRANS2
invoke TextOut,hdc,x_axis+3*x_jump,y_axis+4*y_jump,OFFSET BUF1,2
mov ax,commodity[3*size_of].initem
invoke TRANS2
invoke TextOut,hdc,x_axis+4*x_jump,y_axis+4*y_jump,OFFSET BUF1,2
mov ax,commodity[3*size_of].sellyet
invoke TRANS2
invoke TextOut,hdc,x_axis+5*x_jump,y_axis+4*y_jump,OFFSET BUF1,2
mov ax,commodity[3*size_of].reco
invoke TRANS2
invoke TextOut,hdc,x_axis+6*x_jump,y_axis+4*y_jump,OFFSET BUF1,2
ret
Display endp
```

； 计算推荐度： 入口参数 ebx 存放首地址

```
CALCREFER PROC USES EAX EBX ECX EDX EDI ESI
```

```
mov esi, ebx
```

```
MOV AX, [esi+17]
```

```
CMP AX, [esi+15]
```

```
JNL ENDCALC
```

； 推荐度 = (进货价*(2*进货数量)+已售数量*实际销售价格)*128/实际销售价格*2*进货数量

； 计算进货价*进货数量*2

```
MOVZX EAX, word ptr [esi+15]
```

汇编语言程序设计实验报告

```
SHL EAX, 1
MOVZX ECX, word ptr [esi+11]
MUL ECX
MOV TEMP1, EAX
; 计算实际销售价格
MOVZX EAX, byte ptr [esi+10]
MOV BX, [esi+13]
MOVZX EDX, BX
MUL EDX
MOV EBX, 10
XOR EDX, EDX
DIV EBX
MOV TEMP2, EAX
; 计算分母, 实际销售价格*2*进货数量
MOVZX EBX, word ptr [esi+15]
MUL EBX
SHL EAX, 1
MOV TEMP3, EAX
; 计算已售数量*实际售价
MOVZX EBX, word ptr [esi+17]
MOV EAX, TEMP2
MUL EBX

;两部分加法
ADD EAX, TEMP1
; 乘以 128
SHL EAX, 7
;除法
MOV EBX, TEMP3
XOR EDX, EDX
DIV EBX
; 存放推荐度
MOV [esi+19], AX
ENDCALC:
RET
CALCREFER ENDP

;待转化数放在 AX 中, 输出在 BUF1 中
TRANS2 PROC
    PUSH EBX      ;保护现场
    PUSH ECX
    PUSH EDX
    PUSH ESI
    MOV BX, 10 ;等会 BX 作为被除数 (因为是要转化成 10 进制)
    MOV ECX, 0 ;计数器清零
    LEA ESI, BUF1 ;将 BUF1 的地址给 SI, 等会通过变址寻址把相应的数据放在 BUF1 中
    OR AX, AX ;AX 为正数是直接进入后面的转换
    JNS LOP1
    NEG AX ;AX 为负数时先转变成正数
    MOV BYTE PTR [ESI], '-' ;此时应该先存放一个负号
    INC ESI
LOP1:
    XOR DX, DX ;通过异或将 DX 清零, 这种清零方式应该更快一些 (可以试着验证一下)
    DIV BX ;除 10 取余, 二进制转化为 10 进制的正常操作
    PUSH DX ;将余数进栈 (因为顺序是反的, 第一个余数应该是转换后的进制的最后一位, 此时进行一次进出栈转换顺序)
    INC CX
```

汇编语言程序设计实验报告

```
OR AX, AX      ;AX 为 0 时跳出循环
JNZ LOP1
LOP2:
POP AX          ;将之前的余数出栈
ADD AL, 30H     ;将余数加 30H 变成相应的 ASCII 码
MOV [ESI], AL   ;将余数出栈
INC ESI
LOOP LOP2       ;计数器减一并且判断是否跳出循环
MOV BYTE PTR [ESI], ' ' ;在存放空间的最后加入一个字符串结束符，方便等会字符串的输出
POP ESI         ;保护现场
POP EDI
POP ECX
POP EBX
RET
TRANS2 ENDP

end start
```

3.4 实验步骤

1. 安装 MASM32 软件包，录入源程序所示的代码。（包括资源文件.rc、源文件.asm 和.inc 文件）
2. 在 cmd 下用 MASM32 对文件进行编译和连接，若报错，修改程序错误，若没有错误，则生成得到可执行文件。
3. 测试程序功能，主要包括：执行程序正常生成窗口；点击 File 中的 Exit 能正常退出窗口；点击 Help 中的 About，能正常弹出个人信息；点击 Action 中的 List 能显示商品信息；点击 Recommendation 正常计算推荐度，再次点击 List 能够显示。
4. 用 TD32 观察所编写程序的代码区与数据区，总结其特点，比较其与 16 位的 td 有什么异同。
5. 总结 WIN32 程序与 16 位段程序的主要差异是什么？
6. 观察 Invoke 语句翻译成机器码之后的特点，观察程序参数压栈。
7. 单步跟踪到调用系统 API 函数的位置，观察相关代码的特点。

3.5 实验记录与分析

1. 编译连接过程中没有出现报错。
2. 测试程序功能
 - 1) 执行程序正常生成窗口

正常生成窗口截图如下：

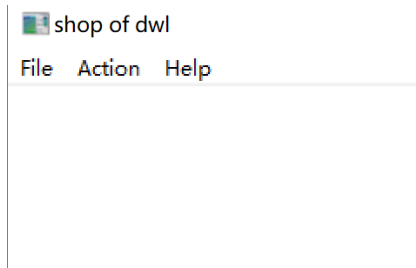


图 3-5 生成 WIN32 窗口

汇编语言程序设计实验报告

从图中我们可以看出，按照设想生成了 WIN32 窗口。窗口名为 shop of dwl，菜单包括 File、Action 和 Help 三个选项。

- 2) 点击 File 中的 Exit 能够正常退出窗口
- 3) 点击 Help 中的 About 能看到我的个人信息

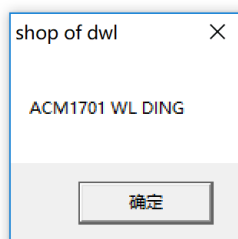
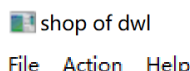


图 3-6 点击 About 显示个人信息

- 4) 点击 Action 中的 List 显示商品信息如下：



item_name	discount	in_price	sell_price	in_item	sell_item	recommendation
PEN	10	10	25	12	80	0
BOOK	9	12	30	25	5	0
NOTE	8	30	50	40	20	0
PENCI	10	5	8	10	5	0

图 3-7 点击 Action 中的 List 显示商品信息

由图，我们正常显示了每个商品的名称、折扣、进价、售价、进货数和销售数，这里推荐度还没有被计算所以为 0。

- 5) 点击 Action 中的 Recommendation 计算推荐度，然后再点击 List 显示

汇编语言程序设计实验报告

shop of dwl

File Action Help

item_name	discount	in_price	sell_price	in_item	sell_item	recommendation
PEN	10	10	25	12	80	93
BOOK	9	12	30	25	5	69
NOTE	8	30	50	40	20	12
PENCI	10	5	8	10	5	11

图 3-8 计算并显示推荐度

如图，运行了 Recommendation 选项之后，List 中 Recommendation 选项不再是 0，变成了其真实数值，计算成功。

3. 用 td32 打开编写的文件，观察其代码区与数据区的特点，比较其与 16 位有什么异同。

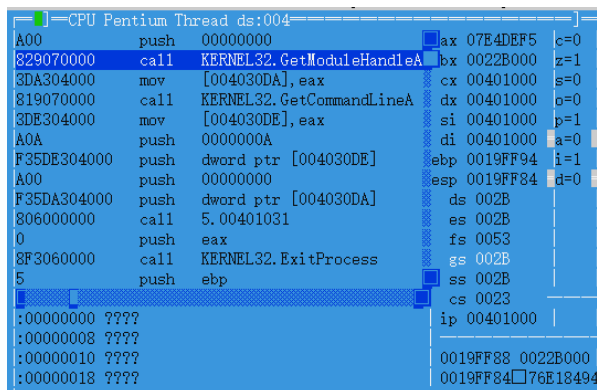


图 3-9 用 td32 调试编写的程序

由图中我们可以看出，用 TD32 对程序进行调试时，程序的偏移地址都是 32 位的，而且数据段中，我们能看到例如 ??? 这样的数据，通过改变偏移地址查看数据段我们能发现，在我们程序用到的数据区域中，它是有显示的，但是在我们程序不用的区域内存中，它是被保护起来的，即用 ??? 表示，为的是不让人读出其中的数据。

除了数据和地址的位数不一样之外，TD32 基本上与 TD16 保持一致的内容形式与操作方式。

4. 总结 WIN32 程序与 16 位段程序的主要差异

1) WIN32 程序偏移地址都是 32 位，用的是 32 位寄存器，而 16 位段程序偏移地址一般是 16 位，一般用 16 位寄存器，WIN32 程序更长的偏移地址和位数更多的寄存器证明 WIN32 程序能承载更长的程序以及更大的数据，即能承载更多的功能。

2) WIN32 程序比 16 位段程序拥有更多的库资源以及其他可扩展的资源。例如，WIN32 程序可以添加资源文件，这样就可以实现菜单、图标、对话框等功能，功能比 16 位段程序更加丰富与强大，并且 WIN32 程序装载的一些库文件，例如 USER32.LIB 等等都能扩展 WIN32 程序的功能，这些功能是 16 位段程序没有的。

5. 观察 Invoke 语句翻译成机器码之后的特点，观察程序参数压栈。

汇编语言程序设计实验报告

打开 TD32，观察 Invoke 语句，截图如下：

```
:004010D6 8945B0      mov     [ebp-50],eax      bx 0022B000 z=1
:004010D9 6A01        push    00000001         ecx 00401000 s=0
:004010DB FF75B0      push    dword ptr [ebp]  edx 00401000 o=0
:004010DE E82F060000 call    USER32.ShowWin  esi 00401000 p=1
:004010E3 FF75B0      push    dword ptr [ebp]  edi 00401000 a=0
:004010E6 E833060000 call    USER32.UpdateW  ebp 0019FF94 i=1
:004010EB 6A00        push    00000000         esp 0019FF84 d=0
:004010ED 6A00        push    00000000         ds 002B
:004010EF 6A00        push    00000000         es 002B
:004010F1 8D45B4      lea     eax,[ebp-4C]      fs 0053
:004010F4 50          push    eax              gs 002B
:004010F5 E8EE050000 call    USER32.GetMess  ss 002B
:00000000 ????                          cs 0023
:00000008 ????                          eip 00401000
:00000010 ????                          :0019FF88 0022B000
:00000018 ????                          :0019FF84 76E1849
```

图 3-10 Invoke 语句在 td32 中的截图

我们可以发现，在 TD32 中，所有的 Invoke 语句都被翻译成了 call 语句，且其机器码有如下特点：其一，Invoke 语句机器码都是 10 位的；其二，Invoke 语句机器码的最高两位都是 E8；其三，Invoke 语句机器码最低四位都是 0000。

下面单步执行，发现执行完 invoke 语句之后，程序会将原始的数据段地址压栈，下面给出了两组例子：

第一组例子，第一幅图为单步执行到 invoke 语句，第二幅图为再执行一步，程序将原始地址压栈（图中方框可能是系统显示等因没有显示出来，其符号为↓）

```
命令提示符 - td32 5
:00401045 C745D81A114000 mov    dword ptr [ebp],bx 002E6000 z=0
:0040104C C745DC00000000 mov    dword ptr [ebp],ecx 836F0347 s=0
:00401053 C745E000000000 mov    dword ptr [ebp],edx 00401000 o=0
:0040105A FF7508      push    dword ptr [ebp]  esi 00401000 p=0
:0040105D 8F45E4      pop     dword ptr [ebp]  edi 00401000 a=0
:00401060 C745F006000000 mov    dword ptr [ebp],ebp 0019FF6C i=1
:00401067 C745F4E2304000 mov    dword ptr [ebp],esp 0019FF14 d=0
:0040106E C745F858304000 mov    dword ptr [ebp],ds 002B
:00401075 68007F0000      push    00007F00         es 002B
:0040107A 6A00        push    00000000         fs 0053
:0040107C E873060000 call    USER32.LoadIco  gs 002B
:00401081 8945E8      mov     [ebp-18],eax      ss 002B
:00000000 ????                          cs 0023
:00000008 ????                          eip 0040107C
:00000010 ????                          :0019FF18 00007F00
:00000018 ????                          :0019FF14 00000000
```

图 3-11 程序将数据地址压栈示意图 1

```
命令提示符 - td32 5
:004016E2 FF253C204000 jmp     [0040203C]        bx 002E6000 z=0
:004016E8 FF2538204000 jmp     [00402038]        cx 836F0347 s=0
:004016EE FF2528204000 jmp     [00402028]        dx 00401000 o=0
:004016F4 FF2518204000 jmp     [00402018]        si 00401000 p=0
:004016FA FF251C204000 jmp     [0040201C]        di 00401000 a=0
:00401700 FF2520204000 jmp     [00402020]        bp 0019FF6C i=1
:00401706 FF2524204000 jmp     [00402024]        sp 0019FF10 d=0
:0040170C FF2548204000 jmp     [00402048]        ds 002B
:00401712 FF252C204000 jmp     [0040202C]        es 002B
:00401718 FF2530204000 jmp     [00402030]        fs 0053
:0040171E FF2534204000 jmp     [00402034]        gs 002B
:00401724 FF250C204000 jmp     [0040200C]        ss 002B
:00000000 ????                          cs 0023
:00000008 ????                          eip 004016F4
:00000010 ????                          :0019FF14 00000000
:00000018 ????                          :0019FF10 0040108
```

图 3-12 程序将数据地址压栈示意图 2

第二组例子，同样的，仿照上述例子的形式，我们再给出一组 invoke 语句以及其压栈的

汇编语言程序设计实验报告

示意图。

命令提示符 - td32 5

```
File Edit View Run Breakpoints Data Options Wi
CPU Pentium Thread #33468
:0040103E C745D403000000 mov dword ptr [ebp+eax, 00683BE0 c=1
:00401045 C745D81A114000 mov dword ptr [ebp+ebx, 003CC000 z=0
:0040104C C745DC00000000 mov dword ptr [ebp+ecx, E8A16DB5 s=0
:00401053 C745E000000000 mov dword ptr [ebp+edx, 00401000 o=0
:0040105A FF7508 push dword ptr [ebp+esi, 00401000 p=0
:0040105D 8F45E4 pop dword ptr [ebp+edi, 00401000 a=0
:00401060 C745F006000000 mov dword ptr [ebp+ebp, 0019FF6C i=1
:00401067 C745F4E2304000 mov dword ptr [ebp+esp, 0019FF14 d=0
:0040106E C745F858304000 mov dword ptr [ebp+ds, 002B
:00401075 68007F0000 push 00007F00 es 002B
:0040107A 6A00 push 00000000 fs 0053
:0040107C E873060000 call USER32.LoadIconA gs 002B
:00401081 8945E8 mov [ebp-18], eax ss 002B
:00000000 ???? eip 0040107C
:00000008 ????
:00000010 ????
:00000018 ????
:0019FF18 00007F00
:0019FF14 00000000
```

图 3-13 程序将数据地址压栈示意图 3

命令提示符 - td32 5

```
File Edit View Run Breakpoints Data Options Wi
CPU Pentium Thread ds:00402018 = 764BE9C0
:004016F4 FF2518204000 jmp [00402018] eax 00683BE0 c=1
:004016FA FF251C204000 jmp [0040201C] ebx 003CC000 z=0
:00401700 FF2520204000 jmp [00402020] ecx E8A16DB5 s=0
:00401706 FF2524204000 jmp [00402024] edx 00401000 o=0
:0040170C FF2548204000 jmp [00402048] esi 00401000 p=0
:00401712 FF252C204000 jmp [0040202C] edi 00401000 a=0
:00401718 FF2530204000 jmp [00402030] ebp 0019FF6C i=1
:0040171E FF2534204000 jmp [00402034] esp 0019FF10 d=0
:00401724 FF250C204000 jmp [0040200C] ds 002B
:0040172A FF2508204000 jmp [00402008] es 002B
:00401730 FF2510204000 jmp [00402010] fs 0053
:00401736 FF2500204000 jmp [00402000] gs 002B
:0040173C 0000 add [eax], al ss 002B
:00000000 ???? cs 0023
:00000008 ???? eip 004016F4
:00000010 ????
:00000018 ????
:0019FF14 00000000
:0019FF10 0040108
```

图 3-14 程序将数据地址压栈示意图 4

下图中一系列的 jmp 都记录着参数被压栈的信息，截图如下：

命令提示符 - td32 5

```
File Edit View Run Breakpoints Data Options Wi
CPU Pentium Thread ds:00402030 = 764CA780
:004016D0 FF254C204000 jmp [0040204C] ax 00683BE0 c=1
:004016D6 FF2540204000 jmp [00402040] bx 003CC000 z=0
:004016DC FF2544204000 jmp [00402044] cx E8A16DB5 s=0
:004016E2 FF253C204000 jmp [0040203C] dx 00401000 o=0
:004016E8 FF2538204000 jmp [00402038] si 00401000 p=0
:004016EE FF2528204000 jmp [00402028] di 00401000 a=0
:004016F4 FF2518204000 jmp [00402018] bp 0019FF6C i=1
:004016FA FF251C204000 jmp [0040201C] sp 0019FF10 d=0
:00401700 FF2520204000 jmp [00402020] ds 002B
:00401706 FF2524204000 jmp [00402024] es 002B
:0040170C FF2548204000 jmp [00402048] fs 0053
:00401712 FF252C204000 jmp [0040202C] gs 002B
:00401718 FF2530204000 jmp [00402030] ss 002B
:00000000 ???? cs 0023
:00000008 ???? eip 004016F4
:00000010 ????
:00000018 ????
:0019FF14 00000000
:0019FF10 0040108
```

图 3-15 参数压栈截图

汇编语言程序设计实验报告

6. 单步跟踪到调用系统 API 函数的位置，观察相关代码的特点。

我们通过单步调试跟踪到调用窗口主程序的语句，即如下所示截图的语句：

`invoke WinMain,hInstance,NULL,CommandLine,SW_SHOWDEFAULT;`调用窗口主程序

图 3-16 源程序中调用系统 API 函数的语句

下面我们通过 TD32 单步调试，跟踪到这一语句，如下截图：

:00401016	6A0A	push	0000000A
:00401018	FF35DE304000	push	dword ptr [004
:0040101E	6A00	push	00000000
:00401020	FF35DA304000	push	dword ptr [004
:00401026	E806000000	call	5.00401031

图 3-17 TD32 中系统 API 函数的显示

可以发现，该语句是先将几个内存地址和几个内存中的数值压栈，最后再用一个 `call` 调用。值得注意的是，这里 `call` 后面的数值很特殊，是一个带小数点的数字，为 5.00401031，这有可能就是调用系统 API 函数时代码的一大特点。

4 总结与体会

通过这次实验，我对基于 WIN32 编程实现一个窗口程序有了更深入的理解与认识，并且通过实验了解并掌握了 WIN32 编程的方法与 WIN32 程序的特点，同时感受到了 WIN32 编程相对于 16 位段程序而言更丰富的功能与更强大的处理能力。实验中也进行了单步调试，了解了 TD32 的调试方法以及 WIN32 编程语句翻译成机器码的特点等等。

实验中运用了 WIN32 实现窗口程序的标准框架，即以下四个部分：主程序、窗口主程序、窗口消息处理程序、用户处理程序，实验中的重点与难点便是构建这个框架。在这个过程中，我对于一些函数有了实践性的认识，例如运用最多的 `Invoke` 函数以及 `TextOut` 函数，同时也通过编程过程中了解到 WIN32 窗口程序框架中一些特点，比如对于段的伪指令的定义，不需要像是 16 位程序那样定义，只需要定义 `.code`、`.data` 以及 `.stack` 即可。同时 WIN32 程序支持结构体的定义，这样能够更方便我们存储商店商品数据以及调用，我们只需要通过结构体的点运算即可调用商品数据，不必自己计算偏移值。

实验另一块就是运用 TD32 查看并调试程序。通过实验我们发现 TD32 和 16 位的 `td` 操作方法基本是一致的。但是 TD32 窗口有其自己的明显特点，就是数据段中的无关数据（与本程序无关的内存区域）是被用？？？？保护起来的。通过单步调试我们通过观察发现 TD32 中都用的是 32 位的寄存器，代码段偏移值是 32 位，同时发现了调用 `invoke` 语句机器码的特点，调用系统 API 函数的代码的特点等等。

汇 编 语 言 程 序 设 计 实 验 报 告

参考文献

- [1] 王元珍 曹忠升 韩宗芬 编著. 80X86 汇编语言程序设计. 武汉: 华中科技大学出版社, 2005 年 4 月. 238 页-270 页