

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实验**

**实验名称： 模块化程序设计**

**实验时间： 2018-4-20，18:30-21:50 实验地点： 南一楼804室14号实验台**

**指导教师： 李海波 专业班级：计算机科学与技术201607班**

**学 号： U201614700 姓 名： 王亚宁**

**同组学生： 无 报告日期： 2018年4月22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：

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**

[实验目的与要求 1](#_Toc512945077)

[实验内容 1](#_Toc512945078)

[实验过程 2](#_Toc512945079)

[任务1 2](#_Toc512945080)

[1. 设计思想及存储单元分配 2](#_Toc512945081)

[2. 流程图 2](#_Toc512945082)

[3. 源程序 7](#_Toc512945083)

[4. 实验步骤 27](#_Toc512945084)

[5. 实验记录 27](#_Toc512945085)

[任务2 29](#_Toc512945086)

[1. 设计思想及存储单元分配 30](#_Toc512945087)

[2. 流程图 30](#_Toc512945088)

[3. 源程序 31](#_Toc512945089)

[4. 实验步骤 44](#_Toc512945090)

[5. 实验记录 44](#_Toc512945091)

[体会 47](#_Toc512945092)

[参考文献 47](#_Toc512945093)

# 实验目的与要求

1. 掌握子程序设计的方法与技巧，熟悉子程序的参数传递方法和调用原理；

2. 掌握宏指令、模块化程序的设计方法;

3. 掌握较大规模程序的合作开发与调试方法；

4. 掌握汇编语言程序与C语言程序混合编程的方法；

5. 熟悉C编译器的基本优化方法;

6. 了解C语言编译器的命名方法，主、子程序之间参数传递的机制。

# 实验内容

任务一：宏与子程序设计，进一步修改与增强实验一任务四的**网店商品信息管理程序**的功能；

1. 调整后的功能三的描述

（1）首先显示一个功能菜单（格式自行定义。若是未登录状态，只显示菜单“1”和“6”）:

1=查询商品信息，2=修改商品信息，3=计算平均利润率，

4=计算利润率排名，5=输出全部商品信息，6=程序退出。

输入1-6的数字进入对应的功能。

（2）查询商品信息

提示用户输入要查询的商品名称。若未能在第一个网店中找到该商品，重新提示输入商品名称。若只输入回车，则回到功能三（1）。

找到该商品之后，按照：“SHOP1，商品名称，销售价，进货总数，已售数量”顺序显示该商品的信息，同时还要将“SHOP2”中该商品的信息也显示出来。显示之后回到功能三（1）。

（3）修改商品信息

提示用户输入要修改信息的商品名称（先指定网店名称）。[若把接下来的处理步骤写成子程序，则网店名称和商品名称（或其偏移地址）就是子程序的入口参数，是否找到、是否是回车或者修改成功的信息是出口参数]。若未能在指定网店中找到该商品，重新提示输入网店名称和商品名称。若只输入回车，则回到功能三（1）。

找到该商品之后，按照：进货价，销售价，进货总数的次序，逐一先显示原来的数值，然后输入新的数值（若输入有错，则重新对该项信息进行显示与修改。若直接回车，则不修改该项信息）。

如：进货价：25》24 //符号“》”仅作为分隔符，也可以选择其他分隔符号

销售价：46》5A6 //输入了非法数值，下一行重新显示和输入

销售价：46》56

进货总数：30》 //直接回车时，对这项信息不做修改

当对三项信息都处理完毕后，回到功能三（1）。

（4）计算平均利润率

首先计算SHOP1中第一个商品的利润率PR1，然后在SHOP2网店中寻找到该商品，也计算其利润率PR2。最后求出该商品的平均利润率APR=(PR1+PR2)/2，并保存到SHOP1的利润率字段中。重复上述步骤，依次将每个商品的平均利润率计算出来。回到功能三（1）。

（5）计算利润率排名

对SHOP2中的每个商品按照平均利润率的大小排名，排名信息存放到SHOP2中商品的利润率字段中。回到功能三（1）。

（6）输出全部商品信息

将SHOP1和SHOP2中的所有商品信息显示到屏幕上，包括平均利润率和排名（替代了商品原有的利润率字段）。具体的显示格式自行定义（可以分网店显示，也可以按照商品排名显示，等等，显示方式可以作为子程序的入口参数）。回到功能三（1）。

2.其他要求

（1）两人一组，一人负责包括菜单显示、程序退出在内的主程序，以及菜单中的功能（1）和（2）；另一人负责菜单中的功能（3）、（4）和（5）。各自汇编自己的模块，设计测试方法，测试通过；然后把自己的模块交给对方，各自把对方的程序整合到自己的程序中，连接生成一个程序，再进行整体调试。

注意，在每个模块的开始，注明编写者的名字以及同组同学的名字。整合到一起时，要注意删掉自己测试时额外加的代码，若有重复的模块（如：两个人都会使用进制转换子程序，各自模块中可能都有相同的进制转换程序），也需要去掉重复的部分。

建议分组方法：按照学号顺序依次两人一组，若班级人数为奇数，则最后三人一组（其中两人的分工是相同的，第三人只需要选择其中一个同学的模块与自己模块整合即可）。

（2）排名的基本要求是按照平均利润率从高到低计算名次，也可以考虑按照指定字段（比如已售数量等）排名。相同平均利润率时排名相同，下一个相邻平均利润率的名次应该是排名在前的所有商品种类“和”的下一个数值。

（3）将9号和10号DOS系统功能调用定义成宏指令并调用。功能（1）-（5）应尽量采用子程序方式实现。需要借鉴书上的进制转换程序：十进制转二进制的子程序F10T2和二进制转十进制的子程序F2T10。

任务二：在C语言程序中调用 汇编语言实现的函数。对于任务1的程序进行改造，主控程序、以及输入输出较多的某一个功能（如功能（1）、（2）、（5）中的某一个）用C语言实现，其他功能用独立的汇编语言子程序的方式实现； 在C语言程序中调用汇编语言子程序。

# 实验过程

## 任务1

### 设计思想及存储单元分配

实验一要求对功能三继续进行改进，设计思想：

1.菜单显示的要求：根据用户的授权情况显示对应的菜单，在用户输入了菜单中的选项后将字符串转换为数字，保存在AL中返回给调用者，然后进入到对应的子程序中。

2.选项1的设计：输入商品名并在商店一中搜索商品，如果搜索失败直接结束，成功则显示信息。

3.选项2的设计：输入商店名和商品名作为参数，传参到modify函数中，调用modify函数。

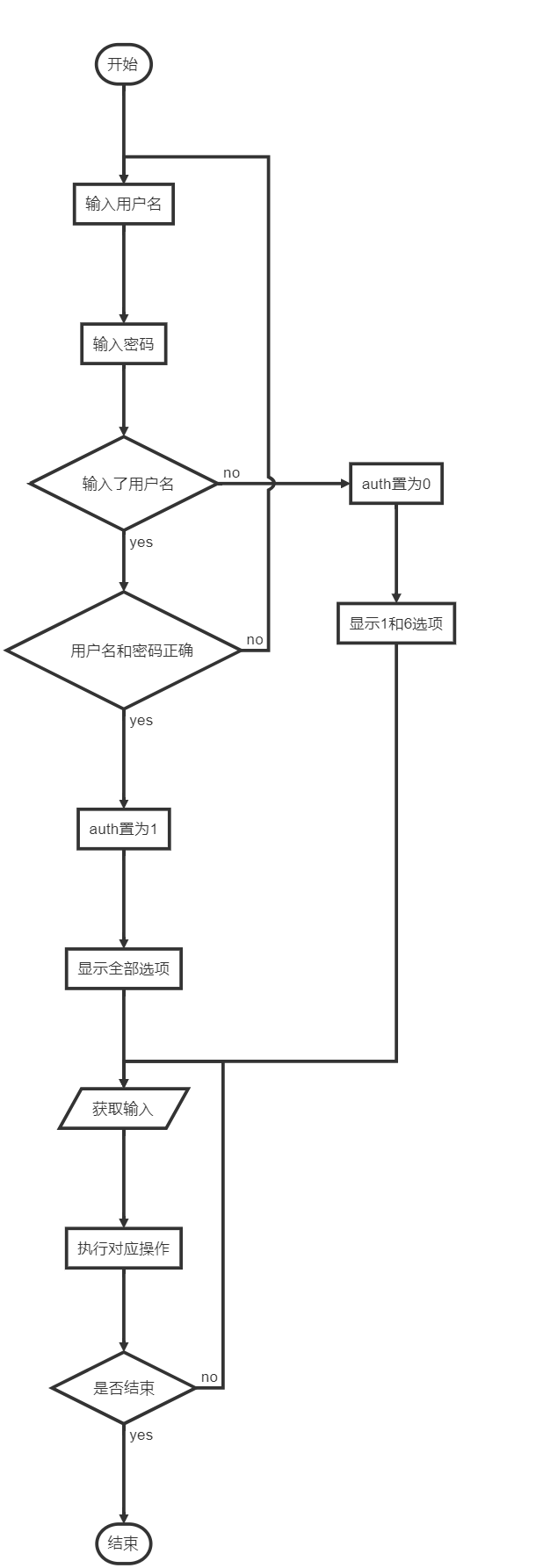
4.选项3的设计：使用上次实验中的计算全部商品利润的子程序即可。

5.选项4的设计：使用辅助数组，存储每个商品的偏移地址，在排序时根据所包含地址的商品利润率对地址排序，由此达到有序。

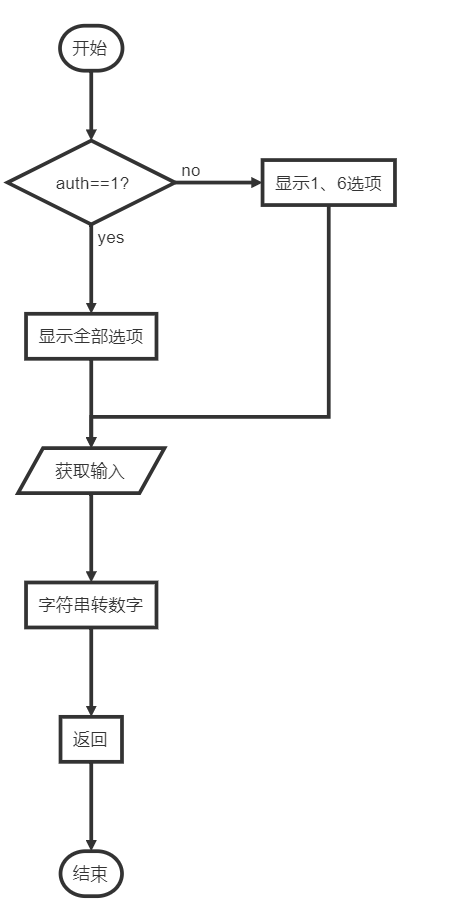
6.选项5的设计：直接遍历所有商品并输出其中的信息即可。

### 流程图

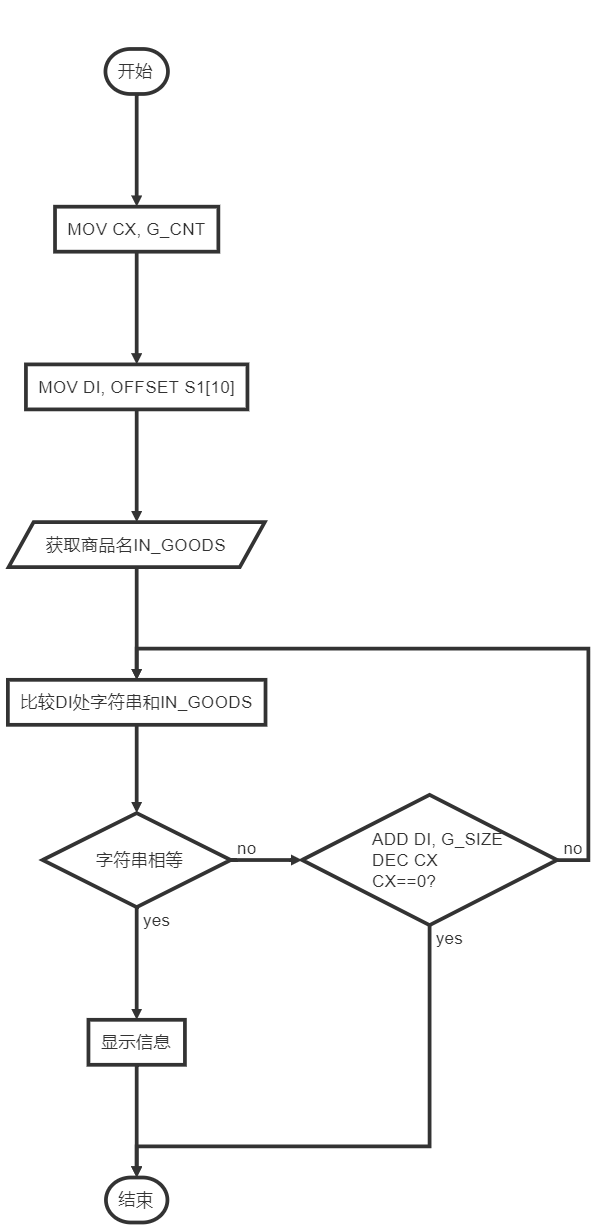
大致的流程图：



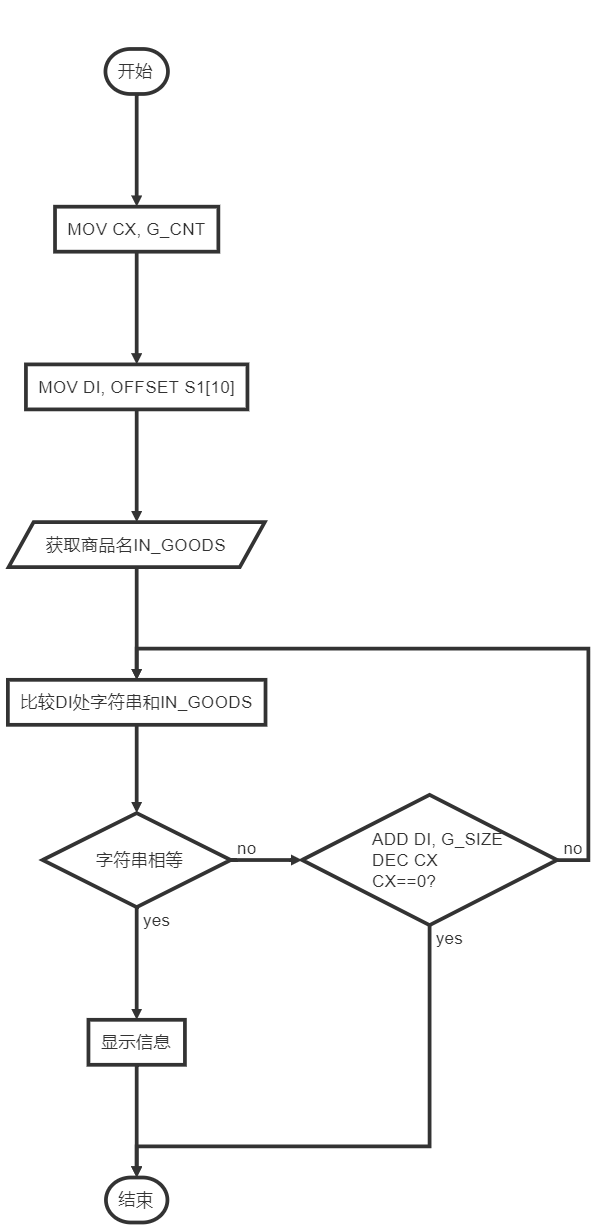
菜单的显示流程图：



显示商品信息的流程图：



修改商品信息的流程图：



求所有利润率的流程图：



### 源程序

第一个文件：

INCLUDE MACRO.LIB

NAME mission1

EXTRN CALCU\_ALL:NEAR, SORT\_PROP:NEAR, PRINT\_ALLP:NEAR, PRINT\_ONE:NEAR

PUBLIC S1, S2, IDXES, SHIFTLINE, PRINT\_NUM

PUBLIC \_ADD, G\_PRO, G\_COST, ADD1, ADD2

PUBLIC G\_SIZE, G\_CNT, S\_CNT, S\_SIZE

.386

DATA1 SEGMENT USE16 PARA PUBLIC 'D1'

NOTICE\_NAME DB 'Please enter user name: $'

NOTICE\_PWD DB 'Please enter user password: $'

NOTICE\_GOODS DB 'Please enter goods name: $'

IN\_GOODS DB 20, ?, 20 DUP(0)

BNAME DB 'yaning', 4 DUP(0); 用户名

BPASS DB 'passwd', 4 DUP(0); 密码

IN\_NAME DB 20, ?, 20 DUP(0)

IN\_PWD DB 20, ?, 20 DUP(0)

G\_CNT = 5

S1 DB 'SHOP1', 5 DUP(0)

BAG1 DB 'BAG', 7 DUP(0)

DW 12, 30, 1000, 5, ? ; 利润率未计算

G\_SIZE = $ - BAG1

GA1 DB 'PEN', 7 DUP(0)

DW 35, 56, 70, 25, ? ; 利润率未计算

GA2 DB 'BOOK', 6 DUP(0)

DW 12, 30, 25, 5, ? ; 利润率未计算

GAN DB G\_CNT - 3 DUP('Temp-Value', 15, 0, 20, 0, 30, 0, 2, 0, ?, ?); 其他商品暂时未知

S\_SIZE = $ - S1

S2 DB 'SHOP2', 5 DUP(0) ;网店名称，用0结束

BAG2 DB 'BAG', 7 DUP(0)

DW 12, 30, 2000, 2, ? ; 利润率未计算

GB1 DB 'PEN', 7 DUP(0) ;商品名称

DW 35, 50, 30, 24, ? ;利润率还未计算

GB2 DB 'BOOK', 6 DUP(0) ; 商品名称

DW 12, 28, 20,15, ? ;利润率还未计算

GBN DB G\_CNT - 3 DUP('Temp-Value', 15, 0, 20, 0, 30, 0, 2, 0, ?, ?)

AUTH DB 0; 标志是否通过验证

GG\_INDEX DD 2 DUP(0); 一共两家店,存放查询的物品的地址在内存中

G\_INDEX DD 0

SHIFTLINE DB 13, 10, '$'

S\_CNT = 2

S\_TEMP\_CNT DB 0

G\_TEMP\_CNT DB 0

SG\_INDEX DD 0

S\_INDEX DD 0

G\_COST DD 0

G\_PRO DD 0

TIME DD 0

; 菜单字符串

query DB "1. Query goods.$", 24 DUP(0); len 13

modify DB "2. Modify goods.$", 23 DUP(0); len 14

calcu\_pro DB "3. Calcu average profile rate.$", 9 DUP(0);len 28

sort\_pro DB "4. Sort the profile rate.$", 14 DUP(0); len 23

print\_all DB "5. Print all goods's information.$", 6 DUP(0); len 31

exit DB "6. Exit the program.$", 19 DUP(0); len 18

\_ADD DW 0

NUM DD 0

; 菜单函数中输入选项使用

STRLEN DW 0

; 输入商店名

NOTICE\_SHOP DB 'Please enter shop name: $'

IN\_SHOP DB 10, ?, 10 DUP(0)

; 地址存放，字符串比较使用

ADD1 DW 0

ADD2 DW 0

; 选项一的shop1、shop2

SHOP1\_STR DB 'SHOP1$'

SHOP2\_STR DB 'SHOP2$'

; 获取数字的字符串缓冲区

IN\_NUM DB 20, ?, 20 DUP(0)

; 选项二的地址参数

ADD3 DW 0

ADD4 DW 0

MYBP DW 0

; 输出提示

COST DB 'Buy cost: $'

PRICE DB 'Sale price: $'

BUY\_CNT DB 'Buy count: $'

; 排序函数中使用的地址数组

IDXES DW G\_CNT DUP(0)

MAX DW 0

RANK DW 0

; 输出全部信息函数使用的提示信息

PRINT\_SHOP1 DB 'In shop1: $'

PRINT\_SHOP2 DB 'IN shop2: $'

PRINT\_BUY DB 'in price: $'

PRINT\_SELL DB 'sell price: $'

PRINT\_BUY\_CNT DB 'buy count: $'

PRINT\_SELL\_CNT DB 'sell count: $'

PRINT\_PRO DB 'profile: $'

PRINT\_RANK DB 'rank: $'

; print\_one数据区

PRINT\_ONE\_BP DW 0

DATA1 ENDS

STACK SEGMENT USE16 PARA STACK 'STACK'

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT USE16 PARA PUBLIC 'CODE'

ASSUME CS:CODE, SS:STACK, DS:DATA1

START:

MOV AX, DATA1

MOV DS, AX

; code after this

FUNC1: ; 功能1开始

WRITE <OFFSET NOTICE\_NAME>

READ <OFFSET IN\_NAME>

CRLF

WRITE <OFFSET NOTICE\_PWD>

READ <OFFSET IN\_PWD>

CRLF

FUNC2: ;功能2开始

; 用户名输入长度为0？

MOV AUTH, 0

CMP IN\_NAME[1], 0

; 调到功能3

JZ FUNC3

; 长度为1，检测是否是q

CMP IN\_NAME[1], 1

; 不是1跳check\_name

JG CHECK\_NAME

CMP IN\_NAME[2], 'q'

; 输入为q，退出

JZ QUIT

CHECK\_NAME:

; 检查用户名

; 已知用户名长度为6，作为已知量使用

CMP IN\_NAME[1], 6

; 不为6错误

JNZ FUNC1

MOV CX, 6

LEA EBX, OFFSET IN\_NAME[2]

LEA EDX, OFFSET BNAME

LOOP\_NAME: ; 偷懒从高到低检查

MOV AH, [EBX]

MOV AL, [EDX]

CMP AH, AL

JNZ FUNC1

INC EBX

INC EDX

LOOP LOOP\_NAME

; 密码也是6位

CMP IN\_PWD[1], 6

JNZ FUNC1

MOV CX, 6

LEA EBX, OFFSET IN\_PWD

ADD EBX, 2

LEA EDX, OFFSET BPASS

LOOP\_PWD:

MOV AH, [EBX]

MOV AL, [EDX]

CMP AH, AL

; 密码不同跳功能1

JNZ FUNC1

INC EBX

INC EDX

LOOP LOOP\_PWD

; 通过用户名和密码检测

MOV AUTH, 1

; 进入功能3

FUNC3:

; 显示菜单

CALL SHOW\_MENU

; 返回值是AL中的选项

; 合法就向下找对应的选项

CMP AL, 1

JE OPT\_1

CMP AL, 2

JE OPT\_2

CMP AL, 3

JE OPT\_3

CMP AL, 4

JE OPT\_4

CMP AL, 5

JE OPT\_5

CMP AL, 6

JE OPT\_6

JMP FUNC3

OPT\_1:

CALL QUERYP

JMP FUNC3

OPT\_2:

WRITE <OFFSET NOTICE\_SHOP>

READ <OFFSET IN\_SHOP>

CRLF

PUSH OFFSET IN\_SHOP[2]

WRITE <OFFSET NOTICE\_GOODS>

READ <OFFSET IN\_GOODS>

CRLF

PUSH OFFSET IN\_GOODS[2]

CALL MODIFYP

JMP FUNC3

OPT\_3:

CALL CALCU\_ALL

JMP FUNC3

OPT\_4:

CALL SORT\_PROP

JMP FUNC3

OPT\_5:

CALL PRINT\_ALLP

JMP FUNC3

OPT\_6:

JMP QUIT

FUNC4:

; 时间计算一下

CALL GET\_MS

SUB EAX, TIME

; print参数压栈

PUSH EAX

; 显示运行时间

CALL PRINT\_NUM

CRLF

; 原来的出栈

MOV AX, [SI + 18]

; 检测利润率范围

CMP AL, 90

JGE SHOWA

CMP AL, 50

JGE SHOWB

CMP AL, 20

JGE SHOWC

CMP AL, 0

JGE SHOWD

JMP SHOWF

SHOWA:

MOV DL, 'A'

JMP SHOW

SHOWB:

MOV DL, 'B'

JMP SHOW

SHOWC:

MOV DL, 'C'

JMP SHOW

SHOWD:

MOV DL, 'D'

JMP SHOW

SHOWF:

MOV DL, 'F'

SHOW:

MOV AH, 2

INT 21H

; 换行

CRLF

JMP FUNC1

QUIT:

;stop here

MOV AH, 4CH

INT 21H

; 函数：获取时间戳，以ms为单位，

; 参数: 无

; 返回值：EAX:

GET\_MS PROC

PUSH EBX

PUSH ECX

PUSH EDX

MOV AH, 2CH

INT 21H

XOR EAX, EAX

XOR EBX, EBX

; 获取小时

MOV BL, CH

ADD EAX, EBX

;; 转为分钟

IMUL EAX, 60

; 获取分钟

MOV BL, CL

ADD EAX, EBX

; 转为s

IMUL EAX, 60

; 获取10ms

MOV BL, DH

ADD EAX, EBX

IMUL EAX, 100

MOV BL, DL

ADD EAX, EBX

; 转为ms

IMUL EAX, 10

POP EDX

POP ECX

POP EBX

RET

GET\_MS ENDP

; 输出函数：输出参数的十进制表示

; 参数：一个双字的数值

; 返回值：无

; 注意：

PRINT\_NUM PROC NEAR

POP BP

POP DWORD PTR NUM

PUSH EAX

PUSH EDX

PUSH ECX

PUSH EBX

MOV EBX, 10

MOV EAX, NUM

XOR CX, CX

CDQ

CMP EAX, 0

JGE L2

PUSH EAX

PUSH EDX

; 输出负号，正数不输出

MOV DL, '-'

MOV AH, 2

INT 21H

POP EDX

POP EAX

NEG EAX

L2:

CDQ

IDIV EBX

MOV DH, 0

ADD DL, '0'

PUSH DX

; 还原EDX的实际情况

INC CX

CMP EAX, 0

JG L2

; 栈里一共ECX个数字可用

L3:

POP DX

MOV AH, 2

INT 21H

LOOP L3

; 恢复环境

POP EBX

POP ECX

POP EDX

POP EAX

PUSH BP

RET

PRINT\_NUM ENDP

; 输出菜单函数

; 参数：无

; 返回值：输入的菜单选项，存放在AL中

SHOW\_MENU PROC

PUSH BX

PUSH CX

PUSH DX

PUSH DI

WRITE <OFFSET query>

CRLF

CMP AUTH, 1

JNZ STR6

MOV DI, OFFSET modify

MOV CX, 4

SHOW\_MENU\_LOOP1:

WRITE <DI>

CRLF

ADD DI, 40

LOOP SHOW\_MENU\_LOOP1

STR6:

WRITE <OFFSET exit>

CRLF

; 输入选项部分

CALL GET\_NUM

CRLF

;恢复环境

POP DI

POP DX

POP CX

POP BX

RET

SHOW\_MENU ENDP

; 二进制转10进制函数？

; 参数：双字数字，字符串首址

; 返回值：无

F2TSTR PROC

POP BP

; 获取字符串偏移地址

POP WORD PTR \_ADD

; 获取数字

POP DWORD PTR NUM

PUSH EAX

PUSH EBX

PUSH ECX

PUSH EDX

PUSH DI

; 首先检测正负号

MOV EAX, NUM

MOV DI, \_ADD

MOV EBX, 10

; EAX符号拓展出来

CMP EAX, 0

JGE POSITIVE

MOV BYTE PTR [DI], '-'; 负号

INC DI

; 转为相反数

NEG EAX

POSITIVE:

XOR EDX, EDX

; EDX:EAX除以10, 得到余数存放在EDX中, EAX中为商

IDIV EBX

ADD DX, '0'

PUSH DX

INC ECX

;MOVSX EAX, AX

CMP EAX, 0

JG POSITIVE

F2TSTR\_LOOP1:

POP DX

MOV [DI], DL

INC DI

; 将栈中字符全部出栈

LOOP F2TSTR\_LOOP1

; 不要忘了$

MOV BYTE PTR [DI], '$'

POP DI

POP EDX

POP ECX

POP EBX

POP EAX

PUSH BP

RET

F2TSTR ENDP

; 字符串转数字函数

; 参数：字符串地址一个字，字符串长度一个字

; 返回值：转换为的数字，EAX中存放，溢出不管,错误返回-1

FSTRT2 PROC

POP BP

; 弹出参数

POP WORD PTR STRLEN

POP WORD PTR \_ADD

; 保存寄存器

PUSH EBX

PUSH ECX

PUSH EDX

PUSH DI

; 置零

XOR EAX, EAX

XOR EBX, EBX

XOR ECX, ECX

XOR EDX, EDX

; 开始计算

; 首先判断正负号

MOV DI, \_ADD

MOV CX, WORD PTR STRLEN

CMP BYTE PTR [DI], '-'

JNZ FSTRT2\_LOOP1

; 正负号是否占据字符位置

INC DI

DEC CX

FSTRT2\_LOOP1:

MOV BL, [DI]

SUB BL, '0'

CMP BL, 0

JL FSTRT2\_ERR

CMP BL, 9

JG FSTRT2\_ERR

; 拓展BX

MOVZX EBX, BL

IMUL EAX, 10

CDQ

ADD EAX, EBX

INC DI

LOOP FSTRT2\_LOOP1

SUB DI, STRLEN

; 判断正负，负责变号

CMP BYTE PTR [DI], '-'

JNZ FSTRT2\_EXIT

NEG EAX

JMP FSTRT2\_EXIT

FSTRT2\_ERR:

MOV EAX, -1

FSTRT2\_EXIT:

POP DI

POP EDX

POP ECX

POP EBX

PUSH BP

RET

FSTRT2 ENDP

; 查询商品函数

; 参数：无（参数在此函数中输入）

; 返回值：无（直接在此函数中输出）

QUERYP PROC

; 需要使用的寄存器先压栈保存环境

PUSH EAX

PUSH EBX

PUSH ECX

PUSH EDX

PUSH DI

; 输出提示

WRITE <OFFSET NOTICE\_GOODS>

; 输入商品名

READ <OFFSET IN\_GOODS>

CRLF

; 开始在商店一寻找商品

MOV CX, G\_CNT

MOV DI, OFFSET S1[10]

XOR EBX, EBX

MOV BL, IN\_GOODS[1]

; EBX移动到输入名下一个位置

ADD EBX, OFFSET IN\_GOODS[2]

; 给它加个尾巴

MOV BYTE PTR [EBX], '$'

QUERY\_LOOP1:

PUSH DI

PUSH OFFSET IN\_GOODS[2]

CALL CMP\_STR

; 查看输入是否与已有商品名相同

CMP AL, 0

JZ END\_QUERY\_LOOP1

; 转移到下一个物品

ADD DI, G\_SIZE

LOOP QUERY\_LOOP1

JMP NO\_GOODS

END\_QUERY\_LOOP1:

; 存在商品就好办了

; DI此时存放的是商品在第一个商店中的位置

; 输出商品的信息“SHOP1，商品名称，销售价，进货总数，

;已售数量”顺序显示该商品的信息，同时还要将“SHOP2”中该商品的信息也显示出来。

WRITE <OFFSET SHOP1\_STR>

COMMA

WRITE <OFFSET IN\_GOODS[2]>

COMMA

; 得出销售价

MOV AX, [DI + 12]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

COMMA

; 进货总数

MOV AX, [DI + 14]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

COMMA

; 已售总量

MOV AX, [DI + 16]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

CRLF

; 显示商店2中方物品信息

ADD DI, S\_SIZE

WRITE <OFFSET SHOP2\_STR>

COMMA

WRITE <OFFSET IN\_GOODS[2]>

COMMA

; 得出销售价

MOV AX, [DI + 12]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

COMMA

; 进货总数

MOV AX, [DI + 14]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

COMMA

; 已售总量

MOV AX, [DI + 16]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

CRLF

; 没有此商品

NO\_GOODS:

; 函数结束

POP EAX

POP EBX

POP ECX

POP EDX

POP DI

RET

QUERYP ENDP

; 修改商店物品的信息

; 参数：商店名和物品名字符串所在地址

; 返回值，无

MODIFYP PROC

; 首先获取参数

POP MYBP

; ADD3是第一个参数，是商品名吧

POP WORD PTR ADD3

POP WORD PTR ADD4

PUSH EAX

PUSH EBX

PUSH ECX

PUSH DI

; 首先比较商店

MOV DI, OFFSET S1

PUSH DI

PUSH ADD4

CALL CMP\_STR

; 是这个shop进入下环节

CMP AL, 0

JZ FIND\_GOODS

MOV DI, OFFSET S2

PUSH DI

PUSH ADD4

CALL CMP\_STR

CMP AL, 0

; 进入下环节

JZ FIND\_GOODS

; 没进入下环节说明没有此商店

JMP NO\_SHOP

; 寻找商品

FIND\_GOODS:

; 哪个商店？在DI中，不需要关心

; 移动到第一个物品处

ADD DI, 10

; 开始寻找物品名

; 将商品数量输入到CX中

MOV CX, G\_CNT

LOOP\_GOODS:

PUSH DI

PUSH WORD PTR ADD3

; 比较字符串是否相同

CALL CMP\_STR

CMP AL, 0

JZ FOUND\_GOOD

; 移动到下一个物品处

ADD DI, G\_SIZE

LOOP LOOP\_GOODS

JMP NO\_GOODS\_1

FOUND\_GOOD:

; 改变进货价

CHG\_COST:

WRITE <OFFSET COST>

MOV AX, [DI + 10]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

; 逗号分隔开

COMMA

; 尝试获取数字

READ <OFFSET IN\_NUM>

CRLF

CMP BYTE PTR IN\_NUM[1], 0

JE CHG\_PRICE

PUSH OFFSET IN\_NUM[2]

MOV AL, IN\_NUM[1]

MOV AH, 0

PUSH AX

CALL FSTRT2

CMP AX, -1

JE CHG\_COST

; 将得到的数字的低两字节放入位置

MOV [DI + 10], AX

CHG\_PRICE:

;同等逻辑

WRITE <OFFSET PRICE>

MOV AX, [DI + 12]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

COMMA

; 尝试获取数字

READ <OFFSET IN\_NUM>

CRLF

CMP IN\_NUM[1], 0

JE CHG\_CNT

PUSH OFFSET IN\_NUM[2]

MOV AL, IN\_NUM[1]

MOV AH, 0

PUSH AX

CALL FSTRT2

CMP AX, -1

JE CHG\_PRICE

MOV [DI + 12], AX

;; 修改进货数量

CHG\_CNT:

WRITE <OFFSET BUY\_CNT>

MOV AX, [DI + 14]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

COMMA

; 尝试获取数字

READ <OFFSET IN\_NUM>

CRLF

CMP BYTE PTR IN\_NUM[1], 0

JE NO\_GOODS\_1

PUSH OFFSET IN\_NUM[2]

MOV AL, IN\_NUM[1]

MOV AH, 0

PUSH AX

CALL FSTRT2

CMP AX, -1

JE CHG\_CNT

MOV [DI + 14], AX

NO\_GOODS\_1:

NO\_SHOP:

; 结束

POP DI

POP ECX

POP EBX

POP EAX

PUSH MYBP

RET

MODIFYP ENDP

; 比较两个字符串

; 参数：两个字符串的偏移地址

; 返回值AL: 1:不想等，0：相等

; 以第一个字符串结尾为准，调用时注意

CMP\_STR PROC NEAR

POP BP

POP WORD PTR ADD1

POP WORD PTR ADD2

PUSH EBX

PUSH ECX

PUSH EDX

PUSH DI

XOR EBX, EBX

XOR EDX, EDX

MOV BX, ADD2

MOV DX, ADD1

MOV AL, 1

CMP\_STR\_LOOP1:

MOV CH, [EBX]

MOV CL, [EDX]

CMP CH, CL

JNZ END\_CMP\_STR

INC EBX

INC EDX

CMP BYTE PTR [EBX], 0

JNZ CMP\_STR\_LOOP1

MOV AL, 0

END\_CMP\_STR:

POP DI

POP EDX

POP ECX

POP EBX

PUSH BP

CMP\_STR ENDP

RET

; 获取一个数字

; 参数无

; 返回值：输入的数字EAX，数字过大将会被截断

; 注意：请求失败返回-1，所以返回-1有可能是输入的-1也肯能是请求失败

GET\_NUM PROC

READ <OFFSET IN\_NUM>

PUSH OFFSET IN\_NUM[2]

MOV AH, 0

MOV AL, IN\_NUM[1]

PUSH AX

; 解析交给函数去做

CALL FSTRT2

RET

GET\_NUM ENDP

CODE ENDS

END START

第二个文件：

INCLUDE MACRO.LIB

EXTRN S1: BYTE, S2: BYTE, IDXES: WORD, SHIFTLINE: BYTE, PRINT\_NUM:NEAR

EXTRN \_ADD: WORD, G\_PRO: DWORD, G\_COST: DWORD, ADD1: WORD, ADD2: WORD

EXTRN G\_SIZE:ABS, G\_CNT:ABS, S\_CNT:ABS, S\_SIZE:ABS

PUBLIC CALCU\_ALL, SORT\_PROP, PRINT\_ALLP, PRINT\_ONE

.386

DATA1 SEGMENT USE16 PARA PUBLIC 'D1'

; 获取数字的字符串缓冲区

IN\_NUM DB 20, ?, 20 DUP(0)

; 选项二的地址参数

ADD3 DW 0

ADD4 DW 0

MYBP DW 0

; 输出提示

COST DB 'Buy cost: $'

PRICE DB 'Sale price: $'

BUY\_CNT DB 'Buy count: $'

; 排序函数中使用的地址数组

MAX DW 0

RANK DW 0

; 输出全部信息函数使用的提示信息

PRINT\_SHOP1 DB 'In shop1: $'

PRINT\_SHOP2 DB 'IN shop2: $'

PRINT\_BUY DB 'in price: $'

PRINT\_SELL DB 'sell price: $'

PRINT\_BUY\_CNT DB 'buy count: $'

PRINT\_SELL\_CNT DB 'sell count: $'

PRINT\_PRO DB 'profile: $'

PRINT\_RANK DB 'rank: $'

; print\_one数据区

PRINT\_ONE\_BP DW 0

; print\_num

NUM DD 0

; pint\_name

NAME\_ADD DW 0

NAME\_BP DW 0

DATA1 ENDS

STACK SEGMENT USE16 PARA STACK 'STACK'

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT USE16 PARA PUBLIC 'CODE'

ASSUME CS:CODE, SS:STACK, DS:DATA1

; 函数：计算一个商品的平均利润率

; 参数：第一个商店里的物品的偏移地址

; 返回值：利润率AX

CALCUG\_PRO PROC

POP BP

POP WORD PTR \_ADD

PUSH EBX

PUSH ECX

PUSH EDX

PUSH SI

MOV DWORD PTR G\_PRO, 0

MOV SI, WORD PTR \_ADD

MOV CX, S\_CNT

L1:

PUSH DWORD PTR G\_PRO

;PUSH DWORD PTR G\_COST

XOR EAX, EAX

XOR EBX, EBX

XOR EDX, EDX

MOV AX, [SI + 10]

MOV BX, [SI + 14]

IMUL BX

; 80x86低位在前

; MOVSX EAX, AX; 拓展AX符号位

MOV WORD PTR G\_COST, AX

MOV WORD PTR G\_COST[2], DX

MOV AX, [SI + 12]

MOV BX, [SI + 16]

IMUL BX

;MOVSX EAX, AX

MOV WORD PTR G\_PRO, AX

MOV WORD PTR G\_PRO[2], DX

MOV EAX, G\_PRO

MOV EBX, G\_COST

SUB EAX, EBX

IMUL EAX, 100

CDQ

IDIV EBX; 得出结果

POP DWORD PTR G\_PRO

ADD G\_PRO, EAX

ADD SI, S\_SIZE

LOOP L1

; 恢复现场并返回

MOV EAX, G\_PRO; AX为低位

CDQ

MOV EBX, S\_CNT

IDIV EBX

POP SI

POP EDX

POP ECX

POP EBX

PUSH BP

RET

CALCUG\_PRO ENDP

; 获取所有商品平均利润率函数

; 参数：无

; 返回值：无

CALCU\_ALL PROC

PUSH AX

PUSH DI

MOV CX, G\_CNT

MOV DI, OFFSET S1[10]

CALCU\_ALL\_LOOP1:

PUSH DI

CALL CALCUG\_PRO

MOV [DI + 18], AX

ADD DI, G\_SIZE

LOOP CALCU\_ALL\_LOOP1

POP DI

POP AX

RET

CALCU\_ALL ENDP

; 平均利润率排名函数

; 我选择使用选择排序

; 参数：第一个商品的位置

; 返回值：无

SORT\_PROP PROC

PUSH EAX

PUSH EBX

PUSH ECX

PUSH EDX

PUSH DI

PUSH SI

MOV CX, G\_CNT

MOV BX, OFFSET IDXES

MOV DI, OFFSET S1[10]

; 获取所有的商品偏移地址

GET\_IDX\_LOOP:

MOV [BX], DI

ADD DI, G\_SIZE

; 每次后移两字节

ADD BX, 2

LOOP GET\_IDX\_LOOP

; 获取地址完毕，对地址对应商品排序

; 通过对利润率排名，将排名结果放到shop2对应的利润字段

; RANK存储排名, 排名从1开始吧

MOV RANK, 1

; CX控制外层循环

MOV CX, G\_CNT

; BX指向第一个物品偏移地址的地址

MOV BX, OFFSET IDXES

SORT\_OUT\_LOOP:

PUSH CX

PUSH BX

; AX存放已知最大利润率

MOV AX, -1

; DI获取商品地址

MOV DI, [BX]

; 获取此商品利润率

MOV AX, [DI + 18]

MOV MAX, BX

; 需要从BX下一个物品处开始循环

; CX现在的数值就是剩余的商品数量加1

SORT\_IN\_LOOP:

DEC CX

CMP CX, 0

JE NOT\_BIG

ADD BX, 2

MOV DI, [BX]

MOV DX, [DI + 18]

CMP DX, AX

; 此商品比最大的利润值小到NOT\_BIG

JLE NOT\_BIG

; 否则此商品就是最大的，

MOV AX, DX

; 获取地址在数组中的地址

MOV MAX, BX

NOT\_BIG:

CMP CX, 0

JG SORT\_IN\_LOOP

; 结束，交换

POP BX

; 将最大的与现在的下标所属元素交换

MOV AX, [BX]

MOV DI, MAX

MOV DI, [DI]

MOV [BX], DI

MOV DI, MAX

MOV [DI], AX

MOV DI, [BX]

MOV AX, RANK

INC WORD PTR RANK

ADD DI, S\_SIZE

MOV [DI + 18], AX

ADD BX, 2

POP CX

LOOP SORT\_OUT\_LOOP

POP SI

POP DI

POP EDX

POP ECX

POP EBX

POP EAX

RET

SORT\_PROP ENDP

; 输出全部信息函数

; 参数：无

; 返回值：无

; 将SHOP1和SHOP2中的所有商品信息显示到屏幕上，包括平均利润率和排名（替代了商品原有的利润率字段）。

;具体的显示格式自行定义（可以分网店显示，也可以按照商品排名显示，等等，显示方式可以作为子程序的入口参数）。

PRINT\_ALLP PROC

PUSH EBX

PUSH ECX

PUSH DI

; 商品数量

MOV CX, G\_CNT

; BX指向shop1中商品

; DI指向shop2中商品

MOV BX, OFFSET S1[10]

MOV DI, OFFSET S2[10]

; 开始循环吧

PRINT\_ALL\_LOOP1:

PUSH BX

PUSH DI

CALL PRINT\_ONE

ADD BX, G\_SIZE

ADD DI, G\_SIZE

LOOP PRINT\_ALL\_LOOP1

POP DI

POP ECX

POP EBX

RET

PRINT\_ALLP ENDP

; 输出一个商品

; 参数：商品在两个商店中的偏移

; 返回值：无

PRINT\_ONE PROC

POP PRINT\_ONE\_BP

POP ADD1

POP ADD2

PUSH EAX

PUSH EBX

PUSH DI

MOV BX, ADD2

MOV DI, ADD1

PUSH BX

CALL PRINT\_NAME

WRITE <OFFSET PRINT\_SHOP1>

WRITE <OFFSET PRINT\_BUY>

MOV AX, [BX + 10]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

COMMA

WRITE <OFFSET PRINT\_SELL>

MOV AX, [BX + 12]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

COMMA

WRITE <OFFSET PRINT\_BUY\_CNT>

MOV AX, [BX + 14]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

COMMA

WRITE <OFFSET PRINT\_SELL\_CNT>

MOV AX, [BX + 16]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

CRLF

WRITE <OFFSET PRINT\_SHOP2>

WRITE <OFFSET PRINT\_BUY>

MOV AX, [DI + 10]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

COMMA

WRITE <OFFSET PRINT\_SELL>

MOV AX, [DI + 12]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

COMMA

WRITE <OFFSET PRINT\_BUY\_CNT>

MOV AX, [DI + 14]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

COMMA

WRITE <OFFSET PRINT\_SELL\_CNT>

MOV AX, [DI + 16]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

CRLF

; 输出利润率

WRITE <OFFSET PRINT\_PRO>

MOV AX, [BX + 18]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

COMMA

; 输出排名

WRITE <OFFSET PRINT\_RANK>

MOV AX, [DI + 18]

MOVSX EAX, AX

PUSH EAX

CALL PRINT\_NUM

CRLF

POP DI

POP EBX

POP EAX

PUSH PRINT\_ONE\_BP

RET

PRINT\_ONE ENDP

; 用来显示程序中以0结尾的字符串，可以用来显示商品名和商店名

; 参数：字单位的地址

; 返回值：无

PRINT\_NAME PROC

POP NAME\_BP

POP NAME\_ADD

PUSH AX

PUSH BX

PUSH DX

MOV BX, NAME\_ADD

NAME\_LOOP:

MOV DL, [BX]

MOV AH, 2

INT 21H

INC BX

CMP BYTE PTR [BX], 0

JNZ NAME\_LOOP

POP DX

POP BX

POP AX

PUSH NAME\_BP

RET

PRINT\_NAME ENDP

CODE ENDS

END

其中包含的宏库：

; 输出字符串的宏

; 参数：字符串偏移地址

; 返回值：无

WRITE MACRO A

PUSH AX

PUSH DX

MOV DX, A

MOV AH, 9

INT 21H

POP DX

POP AX

ENDM

; 读入字符串宏

; 参数：缓冲区偏移地址

; 返回值：无

READ MACRO A

PUSH DX

PUSH AX

MOV DX, A

MOV AH, 10

INT 21H

POP AX

POP DX

ENDM

; 换行宏

; 参数：无

; 返回值：无

CRLF MACRO

WRITE <OFFSET SHIFTLINE>

ENDM

; 输出逗号

; 参数：无

; 返回值：无

COMMA MACRO

PUSH DX

PUSH AX

MOV DL, ','

MOV AH, 2

INT 21H

POP AX

POP DX

ENDM

### 实验步骤

1. 打开dosbox挂载磁盘，切换到对应目录下，输入masm mission1编译模块1，masm module编译模块2。
2. 输入link mission1 module连接两个obj文件生成可执行文件mission1.exe。
3. 输入 mission1运行程序。
4. 程序会提示输入用户名和密码，输入后如果用户名和密码都正确将会进入下一步。
5. 程序会显示一个运行菜单界面，里面有6个选项，分别对应题目中的6个功能。
6. 输入对应的数字，程序会执行对应的子程序并显示其结果。

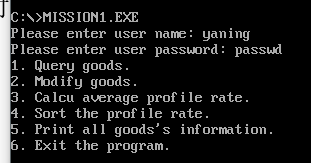
### 实验记录

实验记录如下：

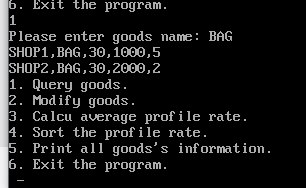
1. 本次实验是一个人完成的，我选择先使用单文件将程序写完确认可以运行后，再将文件拆分成两个文件分别编译连接生成可执行程序。
2. 首先遇到的困难是第四个选项，对所有的商品根据利润率从高到低排序，其实排序算法的思想很简单，但是我一直都是使用的高级语言，并没有使用汇编语言写过排序，于是我选择使用了最简单的选择排序，先使用g++得到选择排序的汇编代码研究一下，自己再根据机器生成的代码写出自己的代码，最终还是写出了排序的功能。
3. 第二个遇到的困难是文件分开后如何连接到一起的方法我先是按照书上给出的代码段和数据段的定义方法，连接文件时发生了错误，根据同学的建议将外部程序的定义改成了FAR类型的，编译连接成功后运行时发现程序在调用时跳转到了一个很远的地方，而且代码也明显和我的没有关系，于是再次修改将两个文件的代码段修改为一个，FAR更改为NEAR，这时程序可以跳转到正确的地方，而且代码也是正确的，再次调试发现功能还是有问题，在运行过选项3后显示菜单时使用的字符串会被修改，很明显发生了数据覆盖，而且功能5的输出都是乱码，检查时我发现我将输出字符串的方式简化为了宏，但是宏替换时并不会检查你的数据段是哪一个，而调用中断时是以DS为，再次将两者的数据段也修改为一个，运行时和一个文件时的结果相同。

运行时的截图。

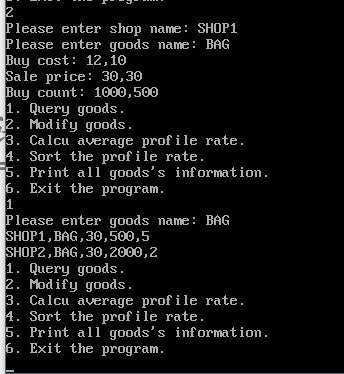
登陆和显示菜单：



功能1使用：

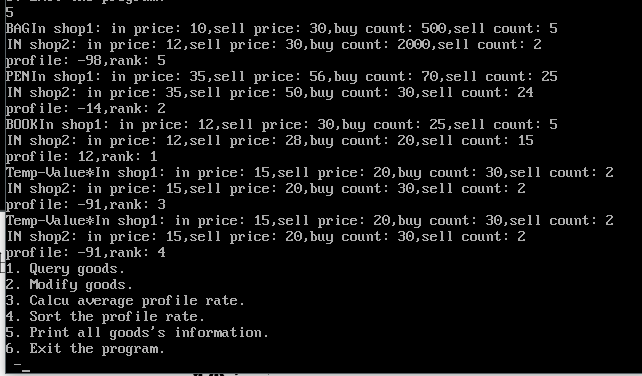


功能2的使用截图：



运行后修改成功。

计算利润率后排序并输出：



根据从高到低排序可知排序结果正确。

## 任务2

在C语言程序中调用汇编语言实现的函数

对于任务1的程序进行改造，主控程序、以及输入输出较多的某一个功能（如功能（1）、（2）、（5）中的某一个）用C语言实现，其他功能用独立的汇编语言子程序的方式实现； 在C语言程序中调用汇编语言子程序

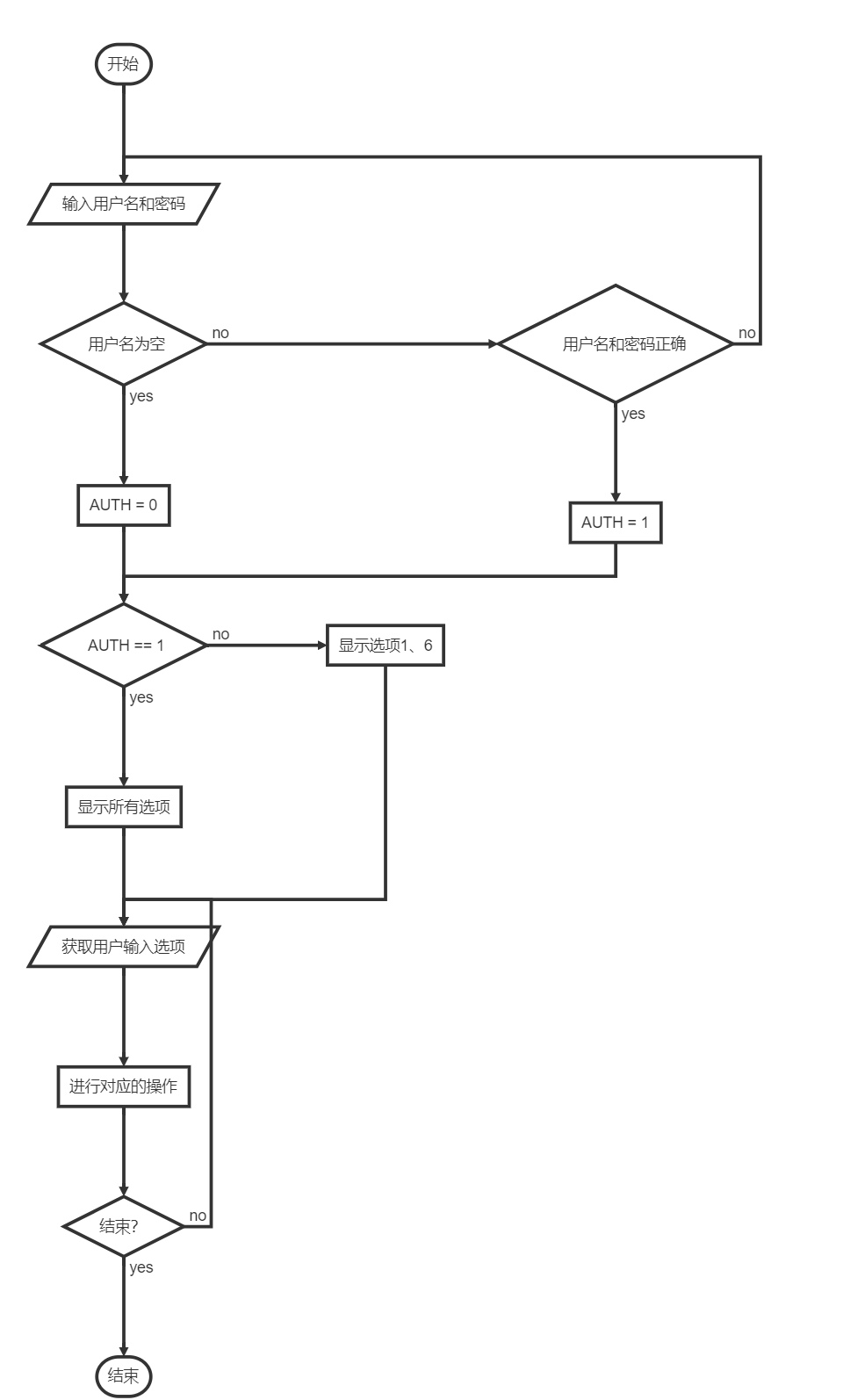
### 设计思想及存储单元分配

本次实验主要考察的是汇编和c相互调用的使用。主要难点在于环境的搭建和接口的声明，环境搭建好后需要熟悉变量的声明，如何在c中调用变量。

设计思想：程序中的c部分主要完成了用户确认和密码确认，输出比较多的是功能5，输出所有的商品信息，这个函数我也是使用c语言编写的。其余模块和函数都是使用汇编编写的。

### 流程图

C语言的主控程序对程序的控制流程图：



### 源程序

C语言的源程序：

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define G\_CNT 4

#define G\_SIZE 20

#define S\_CNT 2

#define S\_SIZE 90

#ifdef \_\_cplusplus

extern "C" {

#endif

// 汇编中变量声明

extern char AUTH, S1;

// 汇编模块声明

extern int SHOW\_MENU();

extern void QUERYP(char \*goods);

extern void MODIFYP(char \*shop, char \*goods);

extern void CALCU\_ALL();

extern void SORT\_PROP();

#ifdef \_\_cplusplus

}

#endif

void prints(char \*s) {

printf("%s", s);

}

int scans(char \*s) {

s[0] = 0;

scanf("%s", s);

int num = strlen(s);

return num;

}

// 功能5，输出全部的商品信息

void print\_allp() {

// 获取两个商店中的商品首地址

char \*goods1 = &S1 + 10;

char \*goods2 = &S1 + S\_SIZE + 10;

for (int i = 0; i < G\_CNT; ++i) {

printf("%s: \n", goods1);

// 获取数据首地址

short \*data1 = (short\*)(goods1 + 10);

short \*data2 = (short\*)(goods2 + 10);

printf("In shop1: ");

printf("sell price: %d, ", data1[1]);

printf("buy count: %d, ", data1[2]);

printf("sell count: %d.\n", data1[3]);

printf("In shop2: ");

printf("sell price: %d, ", data2[1]);

printf("buy count: %d, ", data2[2]);

printf("sell count: %d.\n", data2[3]);

printf("profile: %d%%, ", data1[4]);

printf("rank: %d.\n", data2[4]);

goods1 += G\_SIZE;

goods2 += G\_SIZE;

}

}

void PRINT\_NUM(int num) {

printf("%d", num);

}

int main(void) {

char name[] = "yaning", passwd[] = "passwd";

char in\_name[100] = { 0, }, in\_passwd[100] = { 0, };

while (1) {

printf("Plase enter user name: ");

scanf("%s", in\_name);

printf("Please enter passwd: ");

scanf("%s", in\_passwd);

if (strlen(in\_name) == 0) {

AUTH = 0;

break;

}

if (strcmp(in\_name, name) == 0 && strcmp(in\_passwd, passwd) == 0) {

AUTH = 1;

break;

}

}

char in\_shop[20] = { 0, }, in\_goods[20] = { 0, };

int choice = 0;

while (choice != 6) {

choice = SHOW\_MENU();

switch (choice) {

case 1:

printf("Please enter goods name: ");

scanf("%s", in\_goods);

QUERYP(in\_goods);

break;

case 2:

printf("Enter shop name: ");

scanf("%s", in\_shop);

printf("Enter goods name: ");

scanf("%s", in\_goods);

MODIFYP(in\_shop, in\_goods);

break;

case 3:

CALCU\_ALL();

break;

case 4:

SORT\_PROP();

break;

case 5:

print\_allp();

break;

case 6:

break;

default:

break;

}

}

return 0;

}

汇编程序段代码：

.386

.model flat, c

.stack 400

.data

; 声明为共享数据

public AUTH, S1

G\_CNT = 4

S1 DB 'SHOP1', 5 DUP(0)

BAG1 DB 'BAG', 7 DUP(0)

DW 12, 30, 1000, 5, ? ; 利润率未计算

G\_SIZE = $ - BAG1

GA1 DB 'PEN', 7 DUP(0)

DW 35, 56, 70, 25, ? ; 利润率未计算

GA2 DB 'BOOK', 6 DUP(0)

DW 12, 30, 25, 5, ? ; 利润率未计算

GAN DB G\_CNT - 3 DUP('TempValue', 0, 15, 0, 20, 0, 30, 0, 2, 0, ?, ?); 其他商品暂时未知

S\_SIZE = $ - S1

S2 DB 'SHOP2', 5 DUP(0) ;网店名称，用0结束

BAG2 DB 'BAG', 7 DUP(0)

DW 12, 30, 2000, 2, ? ; 利润率未计算

GB1 DB 'PEN', 7 DUP(0) ;商品名称

DW 35, 50, 30, 24, ? ;利润率还未计算

GB2 DB 'BOOK', 6 DUP(0) ; 商品名称

DW 12, 28, 20,15, ? ;利润率还未计算

GBN DB G\_CNT - 3 DUP('TempValue', 0, 15, 0, 20, 0, 30, 0, 2, 0, ?, ?)

AUTH DB 0; 标志是否通过验证

GG\_INDEX DD 2 DUP(0); 一共两家店,存放查询的物品的地址在内存中

G\_INDEX DD 0

SHIFTLINE DB 13, 10, 0

S\_CNT = 2

S\_TEMP\_CNT DB 0

G\_TEMP\_CNT DB 0

SG\_INDEX DD 0

S\_INDEX DD 0

G\_COST DD 0

G\_PRO DD 0

TIME DD 0

; 菜单字符串

query DB "1. Query goods.", 0, 24 DUP(0); len 13

modify DB "2. Modify goods.", 0, 23 DUP(0); len 14

calcu\_pro DB "3. Calcu average profile rate.", 0, 9 DUP(0);len 28

sort\_pro DB "4. Sort the profile rate.", 0, 14 DUP(0); len 23

print\_all DB "5. Print all goods's information.", 0, 6 DUP(0); len 31

exit DB "6. Exit the program.", 0, 19 DUP(0); len 18

; 选项一的shop1、shop2

SHOP1\_STR DB 'SHOP1', 0

SHOP2\_STR DB 'SHOP2', 0

; 获取数字的字符串缓冲区

IN\_NUM DB 20, ?, 20 DUP(0)

MYBP DD 0

; 输出提示

COST DB 'Buy cost: ', 0

PRICE DB 'Sale price: ', 0

BUY\_CNT DB 'Buy count: ', 0

; 排序函数中使用的地址数组

IDXES DW G\_CNT DUP(0)

MAX DW 0

RANK DW 0

\_comma DB ",", 0

.code

; 函数共享

public SHOW\_MENU, QUERYP, MODIFYP, SORT\_PROP, CALCU\_ALL

; C中的函数

; 80x86里的中断全部GG了，只能使用c里的printf和scanf了

PRINT\_NUM proto c, num : DWORD

prints proto c, s : dword

scans proto c, s : dword

; 输出字符串

; 参数：字符串地址

; 返回值：无

WRITE proc A:dword

push eax

invoke prints, A

pop eax

ret

WRITE endp

; 读入字符串

; 参数：缓冲区地址

; 返回值：无

READ proc A:dword

push eax

push ebx

mov ebx, A

add ebx, 2

invoke scans, ebx

mov [ebx - 1], al

pop ebx

pop eax

ret

READ endp

; 换行函数

; 参数：无

; 返回值：无

CRLF proc

invoke WRITE, offset SHIFTLINE

ret

CRLF endp

; 输出逗号

; 参数：无

; 返回值：无

COMMA proc

invoke write, offset \_comma

ret

COMMA endp

; 比较两个字符串

; 参数：两个字符串的偏移地址

; 返回值AL: 1:不想等，0：相等

; 以第一个字符串结尾为准，调用时注意

CMP\_STR PROC ADD1:DWORD, ADD2:DWORD

PUSH EBX

PUSH ECX

PUSH EDX

PUSH EDI

XOR EBX, EBX

XOR EDX, EDX

MOV EBX, ADD1

MOV EDX, ADD2

MOV AL, 1

CMP\_STR\_LOOP1:

MOV CH, [EBX]

MOV CL, [EDX]

CMP CH, CL

JNZ END\_CMP\_STR

INC EBX

INC EDX

CMP BYTE PTR [EBX], 0

JNZ CMP\_STR\_LOOP1

MOV AL, 0

END\_CMP\_STR:

POP EDI

POP EDX

POP ECX

POP EBX

RET

CMP\_STR ENDP

; 字符串转数字函数

; 参数：字符串地址两个字，字符串长度一个字

; 返回值：转换为的数字，EAX中存放，溢出不管,错误返回-1

FSTRT2 PROC \_ADD:DWORD, STRLEN:WORD

; 弹出参数

; 保存寄存器

PUSH EBX

PUSH ECX

PUSH EDX

PUSH EDI

; 置零

XOR EAX, EAX

XOR EBX, EBX

XOR ECX, ECX

XOR EDX, EDX

; 开始计算

; 首先判断正负号

MOV EDI, \_ADD

MOV CX, WORD PTR STRLEN

CMP BYTE PTR [EDI], '-'

JNZ FSTRT2\_LOOP1

; 正负号是否占据字符位置

INC EDI

DEC CX

FSTRT2\_LOOP1:

MOV BL, [EDI]

SUB BL, '0'

CMP BL, 0

JL FSTRT2\_ERR

CMP BL, 9

JG FSTRT2\_ERR

; 拓展BX

MOVZX EBX, BL

IMUL EAX, 10

CDQ

ADD EAX, EBX

INC EDI

LOOP FSTRT2\_LOOP1

SUB DI, STRLEN

; 判断正负，负责变号

CMP BYTE PTR [EDI], '-'

JNZ FSTRT2\_EXIT

NEG EAX

JMP FSTRT2\_EXIT

FSTRT2\_ERR:

MOV EAX, -1

FSTRT2\_EXIT:

POP EDI

POP EDX

POP ECX

POP EBX

RET

FSTRT2 ENDP

; 获取一个数字

; 参数无

; 返回值：输入的数字EAX，数字过大将会被截断

; 注意：请求失败返回-1，所以返回-1有可能是输入的-1也肯能是请求失败

GET\_NUM PROC

invoke READ, OFFSET IN\_NUM

MOV AH, 0

MOV AL, IN\_NUM[1]

; 解析交给函数去做

invoke FSTRT2, OFFSET IN\_NUM[2], AX

RET

GET\_NUM ENDP

; 输出菜单函数

; 参数：无

; 返回值：输入的菜单选项，存放在AL中

SHOW\_MENU PROC

PUSH EBX

PUSH ECX

PUSH EDX

PUSH EDI

invoke WRITE, OFFSET query

invoke CRLF

CMP AUTH, 1

JNZ STR6

MOV EDI, OFFSET modify

MOV ECX, 4

SHOW\_MENU\_LOOP1:

push ecx

invoke WRITE, EDI

invoke CRLF

ADD EDI, 40

pop ECX

LOOP SHOW\_MENU\_LOOP1

STR6:

invoke WRITE, OFFSET exit

invoke CRLF

; 输入选项部分

invoke GET\_NUM

invoke CRLF

;恢复环境

POP EDI

POP EDX

POP ECX

POP EBX

RET

SHOW\_MENU ENDP

; 查询商品函数

; 参数：无（参数在此函数中输入）

; 返回值：无（直接在此函数中输出）

QUERYP PROC IN\_GOODS : DWORD

; 需要使用的寄存器先压栈保存环境

PUSH EAX

PUSH EBX

PUSH ECX

PUSH EDX

PUSH EDI

; 开始在商店一寻找商品

MOV ECX, G\_CNT

MOV EDI, OFFSET S1[10]

MOV EBX, IN\_GOODS

QUERY\_LOOP1:

invoke CMP\_STR, EDI, EBX

; 查看输入是否与已有商品名相同

CMP AL, 0

JZ END\_QUERY\_LOOP1

; 转移到下一个物品

ADD EDI, G\_SIZE

LOOP QUERY\_LOOP1

JMP NO\_GOODS

END\_QUERY\_LOOP1:

; 存在商品就好办了

; EDI此时存放的是商品在第一个商店中的位置

; 输出商品的信息“SHOP1，商品名称，销售价，进货总数，

;已售数量”顺序显示该商品的信息，同时还要将“SHOP2”中该商品的信息也显示出来。

invoke WRITE, EDI

invoke CRLF

invoke WRITE, OFFSET SHOP1\_STR

invoke COMMA

; 得出销售价

MOV AX, [EDI + 12]

MOVSX EAX, AX

invoke PRINT\_NUM, EAX

invoke COMMA

; 进货总数

MOV AX, [EDI + 14]

MOVSX EAX, AX

invoke PRINT\_NUM, EAX

invoke COMMA

; 已售总量

MOV AX, [EDI + 16]

MOVSX EAX, AX

invoke PRINT\_NUM, EAX

invoke CRLF

; 显示商店2中方物品信息

ADD EDI, S\_SIZE

invoke WRITE, OFFSET SHOP2\_STR

invoke COMMA

; 得出销售价

MOV AX, [EDI + 12]

MOVSX EAX, AX

invoke PRINT\_NUM, EAX

invoke COMMA

; 进货总数

MOV AX, [EDI + 14]

MOVSX EAX, AX

invoke PRINT\_NUM, EAX

invoke COMMA

; 已售总量

MOV AX, [EDI + 16]

MOVSX EAX, AX

invoke PRINT\_NUM, EAX

invoke CRLF

; 没有此商品

NO\_GOODS:

; 函数结束

POP EAX

POP EBX

POP ECX

POP EDX

POP EDI

RET

QUERYP ENDP

; 修改商店物品的信息

; 参数：商店名和物品名字符串所在地址

; 返回值，无

MODIFYP PROC ADD3:DWORD, ADD4:DWORD

PUSH EAX

PUSH EBX

PUSH ECX

PUSH EDI

; 首先比较商店

MOV EBX, OFFSET S1

invoke CMP\_STR, EBX, ADD3

; 是这个shop进入下环节

CMP AL, 0

JZ FIND\_GOODS

MOV EBX, OFFSET S2

invoke CMP\_STR, EBX, ADD3

CMP AL, 0

; 进入下环节

JZ FIND\_GOODS

; 没进入下环节说明没有此商店

JMP NO\_SHOP

; 寻找商品

FIND\_GOODS:

; 哪个商店？在EDI中，不需要关心

; 移动到第一个物品处

ADD EBX, 10

; 开始寻找物品名

; 将商品数量输入到CX中

MOV ECX, G\_CNT

LOOP\_GOODS:

invoke CMP\_STR, EBX, ADD4

CMP AL, 0

JZ FOUND\_GOOD

; 移动到下一个物品处

ADD EBX, G\_SIZE

LOOP LOOP\_GOODS

JMP NO\_GOODS\_1

FOUND\_GOOD:

; 改变进货价

CHG\_COST:

invoke WRITE, OFFSET COST

MOV AX, [EBX + 10]

MOVSX EAX, AX

invoke PRINT\_NUM, EAX

; 逗号分隔开

invoke COMMA

; 尝试获取数字

invoke READ, OFFSET IN\_NUM

CMP BYTE PTR IN\_NUM[1], 0

JE CHG\_PRICE

MOV AL, IN\_NUM[1]

MOV AH, 0

invoke FSTRT2, OFFSET IN\_NUM[2], AX

CMP AX, -1

JE CHG\_COST

; 将得到的数字的低两字节放入位置

MOV [EBX + 10], AX

CHG\_PRICE:

;同等逻辑

invoke WRITE, OFFSET PRICE

MOV AX, [EBX + 12]

MOVSX EAX, AX

invoke PRINT\_NUM, EAX

invoke COMMA

; 尝试获取数字

invoke READ, OFFSET IN\_NUM

CMP IN\_NUM[1], 0

JE CHG\_CNT

MOV AL, IN\_NUM[1]

MOV AH, 0

invoke FSTRT2, OFFSET IN\_NUM[2], AX

CMP AX, -1

JE CHG\_PRICE

MOV [EBX + 12], AX

;; 修改进货数量

CHG\_CNT:

invoke WRITE, OFFSET BUY\_CNT

MOV AX, [EBX + 14]

MOVSX EAX, AX

invoke PRINT\_NUM, EAX

invoke COMMA

; 尝试获取数字

invoke READ, OFFSET IN\_NUM

CMP BYTE PTR IN\_NUM[1], 0

JE NO\_GOODS\_1

MOV AL, IN\_NUM[1]

MOV AH, 0

invoke FSTRT2, OFFSET IN\_NUM[2], AX

CMP AX, -1

JE CHG\_CNT

MOV [EBX + 14], AX

NO\_GOODS\_1:

NO\_SHOP:

; 结束

POP EDI

POP ECX

POP EBX

POP EAX

RET

MODIFYP ENDP

; 函数：计算一个商品的平均利润率

; 参数：第一个商店里的物品的偏移地址

; 返回值：利润率AX

CALCUG\_PRO PROC \_ADD:DWORD

PUSH EBX

PUSH ECX

PUSH EDX

PUSH ESI

MOV G\_PRO, 0

MOV ESI, \_ADD

MOV ECX, S\_CNT

L1:

PUSH G\_PRO

XOR EAX, EAX

XOR EBX, EBX

XOR EDX, EDX

MOV AX, [ESI + 10]

MOV BX, [ESI + 14]

IMUL AX, BX

; 80x86低位在前

MOVSX EAX, AX; 拓展AX符号位

MOV G\_COST, EAX

MOV AX, [ESI + 12]

MOV BX, [ESI + 16]

IMUL AX, BX

MOVSX EAX, AX

MOV G\_PRO, EAX

MOV EAX, G\_PRO

MOV EBX, G\_COST

SUB EAX, EBX

IMUL EAX, 100

CDQ

IDIV EBX; 得出结果

POP DWORD PTR G\_PRO

ADD G\_PRO, EAX

ADD ESI, S\_SIZE

LOOP L1

; 恢复现场并返回

MOV EAX, G\_PRO

CDQ

MOV EBX, S\_CNT

IDIV EBX

POP ESI

POP EDX

POP ECX

POP EBX

RET

CALCUG\_PRO ENDP

; 获取所有商品平均利润率函数

; 参数：无

; 返回值：无

CALCU\_ALL PROC

PUSH EAX

PUSH EDI

MOV ECX, G\_CNT

MOV EDI, OFFSET S1[10]

CALCU\_ALL\_LOOP1:

invoke CALCUG\_PRO, EDI

MOV [EDI + 18], AX

ADD EDI, G\_SIZE

LOOP CALCU\_ALL\_LOOP1

POP EDI

POP EAX

RET

CALCU\_ALL ENDP

; 平均利润率排名函数

; 我选择使用选择排序

; 参数：无

; 返回值：无

SORT\_PROP PROC

PUSH EAX

PUSH EBX

PUSH ECX

PUSH EDX

PUSH DI

PUSH SI

MOV CX, G\_CNT

MOV EBX, OFFSET IDXES

MOV EDI, OFFSET S1[10]

; 获取所有的商品偏移地址

GET\_IDX\_LOOP:

MOV [EBX], DI

ADD DI, G\_SIZE

; 每次后移两字节

ADD BX, 2

LOOP GET\_IDX\_LOOP

; 获取地址完毕，对地址对应商品排序

; 通过对利润率排名，将排名结果放到shop2对应的利润字段

; RANK存储排名, 排名从1开始吧

MOV RANK, 1

; CX控制外层循环

MOV CX, G\_CNT

; BX指向第一个物品偏移地址的地址

MOV EBX, OFFSET IDXES

SORT\_OUT\_LOOP:

PUSH CX

PUSH BX

; AX存放已知最大利润率

MOV AX, -1

; DI获取商品地址

MOV DI, [EBX]

; 获取此商品利润率

MOV AX, [EDI + 18]

MOV MAX, BX

; 需要从BX下一个物品处开始循环

; CX现在的数值就是剩余的商品数量加1

SORT\_IN\_LOOP:

DEC CX

CMP CX, 0

JE NOT\_BIG

ADD BX, 2

MOV DI, [EBX]

MOV DX, [EDI + 18]

CMP DX, AX

; 此商品比最大的利润值小到NOT\_BIG

JLE NOT\_BIG

; 否则此商品就是最大的，

MOV AX, DX

; 获取地址在数组中的地址

MOV MAX, BX

NOT\_BIG:

CMP CX, 0

JG SORT\_IN\_LOOP

; 结束，交换

POP BX

; 将最大的与现在的下标所属元素交换

MOV AX, [EBX]

MOV DI, MAX

MOV DI, [EDI]

MOV [EBX], DI

MOV DI, MAX

MOV [EDI], AX

MOV DI, [EBX]

MOV AX, RANK

INC WORD PTR RANK

ADD DI, S\_SIZE

MOV [EDI + 18], AX

ADD BX, 2

POP CX

LOOP SORT\_OUT\_LOOP

POP SI

POP DI

POP EDX

POP ECX

POP EBX

POP EAX

RET

SORT\_PROP ENDP

end

### 实验步骤

1. 环境搭建：本次饰演的环境使用Visual Studio community 2017, version 15.6.7，搭建过程主要如下：打开VS，选中Visual C++，选择新建一个Empty Project，项目命名为ASM，点击OK新建项目，项目建立好后打开View-Solution Explorer，右键项目名称找到Build Dependencies-Build Customizations，选中masm，点击OK，将项目文件添加到Source Files里面，点击Build-Build Solution生成解决方案，即可生成可执行文件。
2. 运行程序可以直接点击Debug-Start Without Debugging，即可运行程序。
3. 程序运行时操作与上次的任务相同，不过这次是在cmd里运行，而不是在DosBox中了。

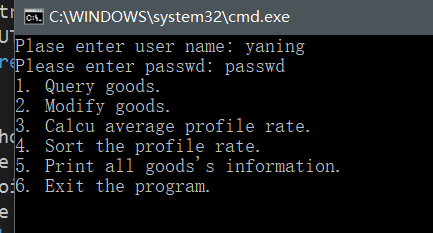
### 实验记录

本次实验中遇到的问题记录：

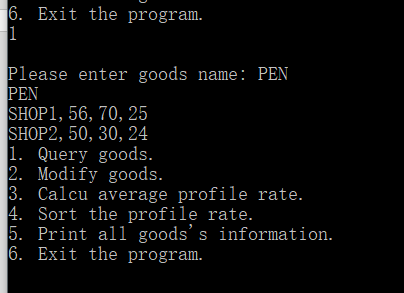
1. 输入用户名为空时，运行strlen获取字符串长度的返回值并不是0，经过VS调试发现是因为char数组在不进行初始化时它的内容是不确定的，也就是说scanf输入空字符串时并不会在开头添加一个0表示结束，由此导致输入用户名位空出现问题，只需在char数组定义时进行初始化即可，随后运行正确。
2. 修改数据时发生了错误：进入功能2时，我对第一个数据修改，第二个不想修改时运行结果显示我的第二个数据也被修改为与第一个数据相同了，Debug后发现还是字符串在第二次输入为空字符串时第一次的数据没有修改，和第一个的错误一样，在scanf前对字符串首字符置为0即可防止错误发生。
3. C语言如何调用汇编中的函数问题：传参只需要按照参数的顺序依次在C语言中声明即可，传参也是一样。
4. 如何使用汇编调用C语言的函数进行字符串输出：由于32位下的汇编不能使用中断进行数据的输入输出，只能调用C语言中的printf和scanf函数进行输出，调用时传参就是传的字符串的首字符地址，也就是汇编中对一个内存数据进行OFFSET操作的结果，理解了这个机制之后传参就会方便很多。

程序的运行记录如下：

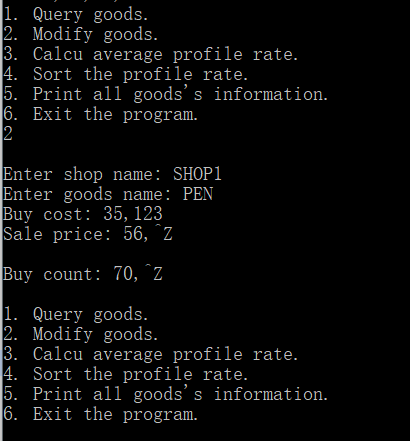
输入用户名和密码运行截图：



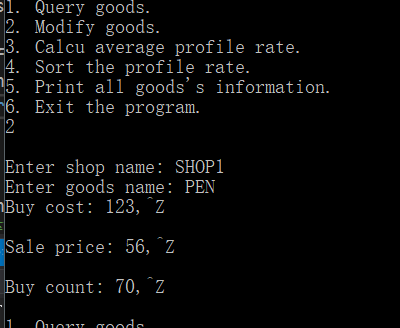
功能1示范：



功能2示范：

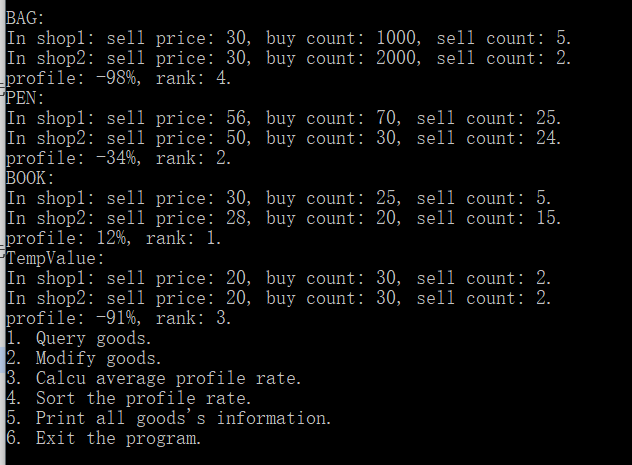


输出信息确认修改是否正确：



可见Buy cost已被修改。

功能3和4运行没有回显，运行后运行功能5示范：



输出信息正确，排名也没有问题，利润率正确。

# 体会

任务一的难点主要是对程序分块后的编写，培养的是模块化的变成思想，我在实验中遇到的困难主要就是程序编写好后的编译和连接，在单文件时没有出现的问题可以得到很好的显露。主要就是代码段和数据段的定义，如何定义外部变量，如何使用masm和link对多文件程序进行编译连接。在编写子程序时也 学到了如何使用栈传参，如何编写接口的文档让别人更好的复用自己的程序，充分的测试也是很有必要的。

任务二不需要编写太多的新代码，主要是学习如何在C语言中使用会变得模块和在汇编中使用C语言的模块。由于在32位模式下80x86的中断不能使用，那么就必须使用C语言中的printf和scanf，在C中对scanf和printf进行封装，封装为scans和prints，接受字符串首地址参数，只进行字符串输出，在汇编中对函数进行声明，使用invoke进行调用即可，比直接使用int 21h的方法方便了一些，也更加安全，不过我也遇到了一些比较坑的地方，就比如输入空字符串时如果上次的不为空那么字符串中的数据就还是上一次的输入数据，会导致数据错误，在scanf前对数据修改一下即可。

# 参考文献

[1] 王元珍等.80x86汇编语言程序设计.版本(第1版)

[2] 王晓虹等.汇编语言程序设计教程.版本(第1版)