# 课 程 实 验 报 告

**实验名称： 汇编实验一至五**

**实验地点： 南一楼804室**

**指导教师： 张勇**

**专业班级：计算机科学与技术201706班**

**学 号： U2015414556 姓 名： 梁一飞**

**同组学生： 无 报告日期： 2019年 5 月 10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： 纸质版再签名

日期：2017.3.17

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

[课 程 实 验 报 告 0](#_Toc9973761)

[实验一 编程基础 3](#_Toc9973762)

[实验内容 4](#_Toc9973763)

[实验过程 6](#_Toc9973764)

[1.1 任务1 6](#_Toc9973765)

[1.2 任务2 9](#_Toc9973766)

[1.3 任务3 11](#_Toc9973767)

[1.4 任务1-3总结与体会 13](#_Toc9973768)

[1.5 任务4 14](#_Toc9973769)

[1.6 任务4总结与体会 23](#_Toc9973770)

[实验二 程序优化 23](#_Toc9973771)

[实验内容 24](#_Toc9973772)

[实验过程 26](#_Toc9973773)

[2.1 任务1 26](#_Toc9973774)

[2.2 任务2 38](#_Toc9973775)

[2.3. 总结与体会 47](#_Toc9973776)

[实验三 模块化程序设计 47](#_Toc9973777)

[实验内容 48](#_Toc9973778)

[实验过程 54](#_Toc9973779)

[3.1任务1 54](#_Toc9973780)

[3.2 任务1总结与体会 77](#_Toc9973781)

[3.3 任务2 77](#_Toc9973782)

[3.4 任务2 总结与体会 90](#_Toc9973783)

[实验四 中断与反跟踪 90](#_Toc9973784)

[实验内容 91](#_Toc9973785)

[实验过程 93](#_Toc9973786)

[4.1 任务1 93](#_Toc9973787)

[4.2 任务2 96](#_Toc9973788)

[4.3 任务3 99](#_Toc9973789)

[4.4 任务4 101](#_Toc9973790)

[4.5 任务5 139](#_Toc9973791)

[4.6 总结与体会 141](#_Toc9973792)

[实验五 WIN32编程。 141](#_Toc9973793)

[实验内容 142](#_Toc9973794)

[实验过程 145](#_Toc9973795)

[5.1 实验步骤 145](#_Toc9973796)

[5.2 源程序 145](#_Toc9973797)

[5.3 实验记录与分析 156](#_Toc9973798)

[5.4 具体函数实现 159](#_Toc9973799)

[5.5 总结与体会 159](#_Toc9973800)

# 实验一 编程基础

**实验名称： 编程基础**

**实验地点： 南一楼804室**

**指导教师： 张勇**

**专业班级：计算机科学与技术201706班**

**学 号： U2015414556 姓 名： 梁一飞**

**同组学生： 无 报告日期： 2019年 3 月 14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： 纸质版再签名

日期：2017.3.17

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：

**实验目的与要求**

1. 掌握汇编源程序编辑工具、汇编程序、连接程序、调试工具TD的使用；
2. 理解数、符号、寻址方式等在计算机内的表现形式；
3. 理解指令执行与标志位改变之间的关系；
4. 熟悉常用的DOS功能调用；
5. 熟悉分支、循环程序的结构及控制方法，掌握分支、循环程序的调试方法；
6. 加深对转移指令及一些常用的汇编指令的理解。

## 实验内容

**任务1.** 《80X86汇编语言程序设计》教材中 P31的 1.14题。

要求：(1) 直接在TD中输入指令，完成两个数的求和、求差的功能。求和/差后的结果放在(AH)中。

(2) 请事先指出执行指令后(AH)、标志位 SF、OF、CF、ZF的内容。

(3) 记录上机执行后的结果，与（2）中对应的内容比较。

**任务2. 《80X86汇编语言程序设计》教材中 P45的 2.3题。**

要求：（1）分别记录执行到“MOV CX，10”和“INT 21H”之前的(BX), (BP),(SI),(DI)各是多少。

（2）记录程序执行到退出之前数据段开始40个字节的内容，指出 程序运行结果是否与设想的一致。

任务3. 《80X86汇编语言程序设计》教材中 P45的 2.4题的改写。

要求：(1) 实现的功能不变，但对数据段中变量访问时所用到的变址寄存器采用32位寄存器。

(2) 记录程序执行到退出之前数据段开始40个字节的内容，检查 程序运行结果是否与设想的一致。

(3)在TD代码窗口中观察并记录机器指令代码在内存中的存放形式，并与TD中提供的反汇编语句及自己编写的源程序语句进行对照，也与任务2做对比。（相似语句记录一条即可，重点理解机器码与汇编语句的对应关系，尤其注意操作数寻址方式的编码形式，比如寄存器间接寻址、变址寻址、32位寄存器与16位寄存器编码的不同、段前缀在代码里是如何表示的等）。

（4）观察连续存放的二进制串在反汇编成汇编语言语句时，从不同字节位置开始反汇编，结果怎样？理解 IP/EIP指明指令起始位置的重要性。

任务4. 设计实现一个网店商品信息管理的程序。

1、实验背景

有一个老板在网上开了1个网店SHOP，网店里有n种商品销售。每种商品的信息包括：商品名称（10个字节，数据段中定义时，名称不足部分补0），折扣（字节类型，取值0~10；0表示免费赠送，10表示不打折，1~9为折扣率；实际销售价格=销售价\*折扣/10），进货价(字类型)，销售价（字类型），进货总数（字类型），已售数量（字类型），推荐度【=（进货价/实际销售价格+已售数量/(2\*进货数量)）\*128，字类型】。老板管理网店信息时需要输入自己的名字（10个字节，数据段中定义时，不足部分补0）和密码（6个字节，数据段中定义时，不足部分补0），登录后可查看商品的全部信息；顾客（无需登录）可以查看网店中每个商品除了进货价以外的信息。

例如：

BNAME DB ‘ZHANG SAN’,0 ；老板姓名（必须是自己名字的拼音）

BPASS DB ‘test’，0，0 ；密码

N EQU 30

SNAME DB ‘SHOP’,0 ；网店名称，用0结束

GA1 DB ‘PEN’, 7 DUP(0) ，10 ；商品名称及折扣

DW 35，56，70，25，？ ；推荐度还未计算

GA2 DB ‘BOOK’, 6 DUP(0) ，9 ；商品名称及折扣

DW 12，30，25，5，？ ；推荐度还未计算

GAN DB N-2 DUP( ‘Temp-Value’,8，15，0，20，0，30，0，2，0，？，？) ;除了2个已经具体定义了的商品信息以外，其他商品信息暂时假定为一样的。

2、功能一：提示并输入登录用户的姓名与密码

（1）使用9号DOS系统功能调用，先后分别提示用户输入姓名和密码(第一行显示将要访问的网店名称)。

（2）使用10号DOS系统功能调用，分别输入姓名和密码。输入的姓名字符串放在以in\_name为首址的存储区中，密码放在以in\_pwd为首址的存储区中，进入功能二的处理。

（3）若输入姓名时只是输入了回车，则将0送到AUTH字节变量中，跳过功能二，进入功能三；若在输入姓名时仅仅输入字符q，则程序退出。

3、功能二：登录信息认证

（1）使用循环程序结构，比较姓名是否正确。若不正确，则跳到（3）。

（2）若正确，再比较密码是否相同，若不同，跳到（3）。

（3）若名字或密码不对，则提示登录失败，并回到“功能一（1）”的位置，提示并重新输入姓名与密码。

（4）若名字和密码均正确，则将1送到AUTH变量中，进到功能三。

**提示：**字符串比较时，当采用输入串的长度作为循环次数时，若因循环次数减为0而终止循环，则还要去判断网店中定义的字符串的下一个字符是否是结束符0，若是，才能确定找到了（这样做是为了避免输入的字符串仅仅是数据段中所定义字符串的子集的误判情况）。

4、功能三：计算指定商品的推荐度。

（1）提示用户输入要查询的商品名称。若未能找到该商品，重新提示输入商品名称。若只输入回车，则回到功能一（1）。

（2）判断登录状态，若是已经登录的状态，转到（3）。否则，转到（4）。

（3）在下一行显示该商品的名称，然后回到功能一（1）。

（4）计算该商品的推荐度， 然后进入功能四。

**要求尽量避免溢出**。

**提示：**使用循环程序结构，注意寻址方式的灵活使用。结果只保留整数部分。

5、功能四：将功能三计算的推荐度进行等级判断，并显示判断结果。

（1）等级显示方式：若推荐度大于100，显示“A”；大于50，显示“B”；大于10，显示“C”；其他，显示“F”。

**提示：**使用分支程序结构，采用2号DOS系统功能调用显示结果。

（2）使用转移指令回到“功能一（1）”处（提示并输入姓名和密码）。

## 实验过程

### 1.1 任务1

#### 1.1.1实验步骤

1. 准备上机实验环境。

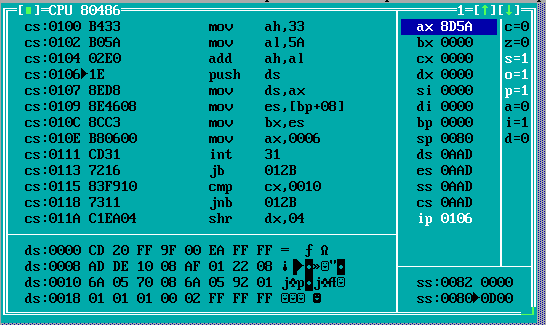
2. 在TD的代码窗口中的当前光标下输入第一个运算式对应的两个8位数值对应的指令语句MOV AH,0110011B；MOV AL,1011010B；ADD AH,AL(SUB AH,AL)；观察代码区显示的内容与自己输入字符之间的关系；然后确定CS:IP指向的是自己输入的第一条指令的位置，单步执行三次，观察寄存器内容的变化，记录标志寄存器的结果。重复上述步骤，完成所有计算

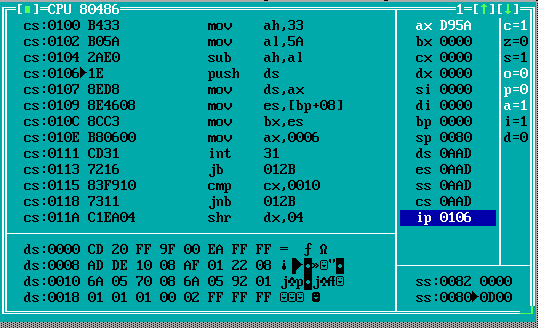
预计计算结果如下表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | AH | SF | OF | ZF | CF |
| 1-求和 | 8D | 1 | 1 | 0 | 0 |
| 1-求差 | D9 | 1 | 0 | 0 | 1 |
| 2-求和 | 7A | 0 | 1 | 0 | 1 |
| 2-求差 | 34 | 0 | 0 | 0 | 0 |
| 3-求和 | 09 | 0 | 0 | 0 | 1 |
| 3-求差 | C2 | 1 | 1 | 0 | 1 |

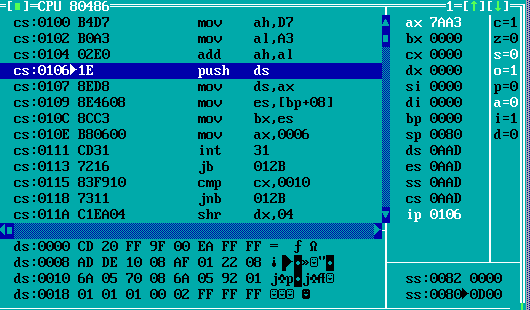
#### 1.1.2实验记录与分析

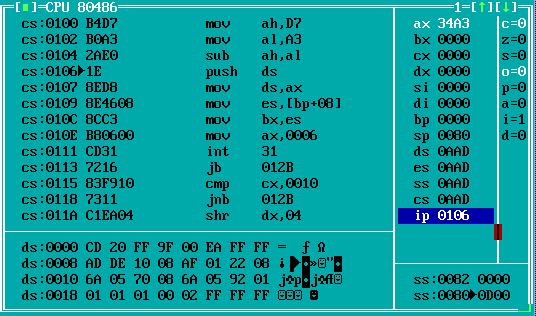
1.x1=+110011B x2=+011010B

求和：

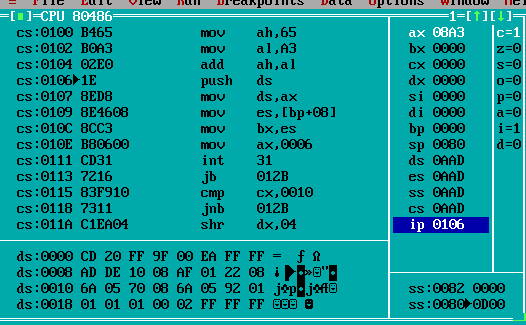
求差：

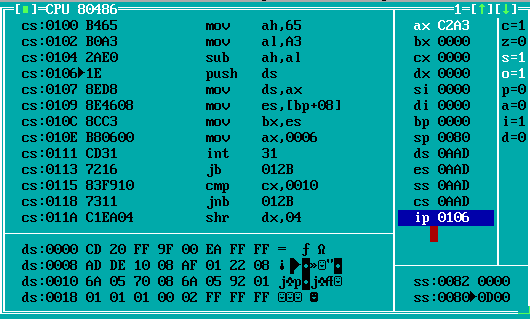
2.x1=-0101001B x2=-1011101B

求和：

求差：

3.x1=+1100101B x2=-1011101B

求和：

求差：

由结果可知，与预期相符

### 1.2 任务2

#### 1.2.1 实验步骤

1.用记事本输入源代码，并命名未ASM格式

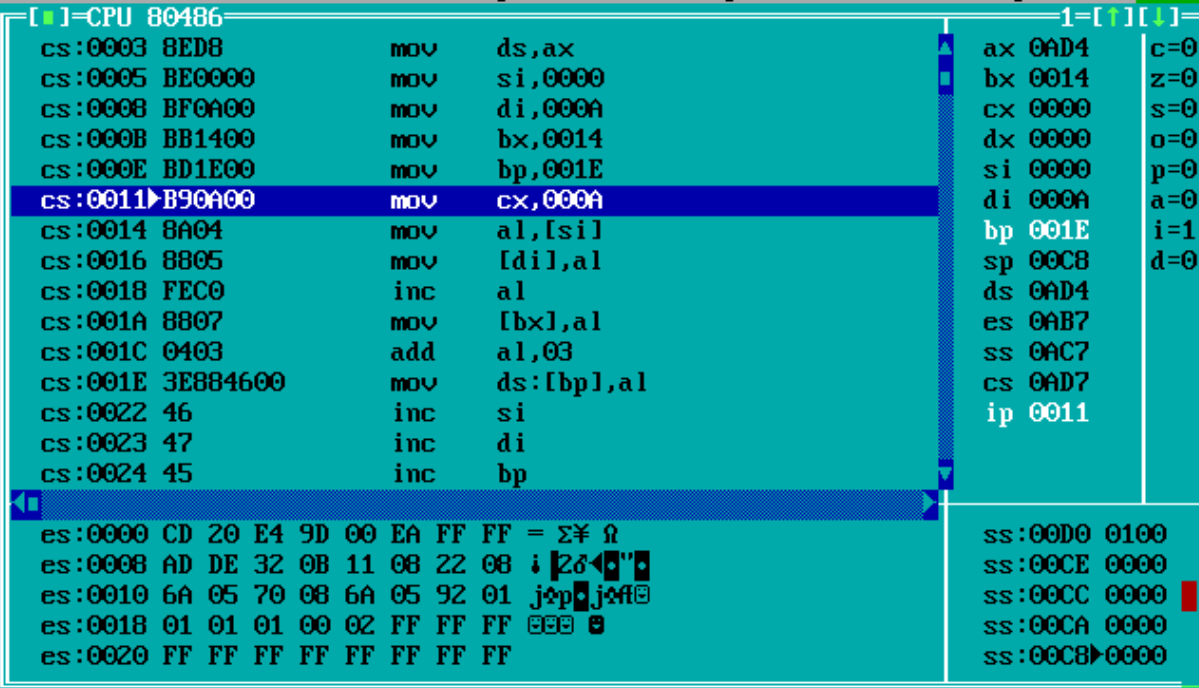
2.打开DOSBOX，配置好，然后输入MASM task2.asm（源代码文件名）生成OBJ文件，然后LINK TASK2.OBJ 生成EXE文件

3.使用TD TASK2.EXE进入调试页面观察要记录的数据

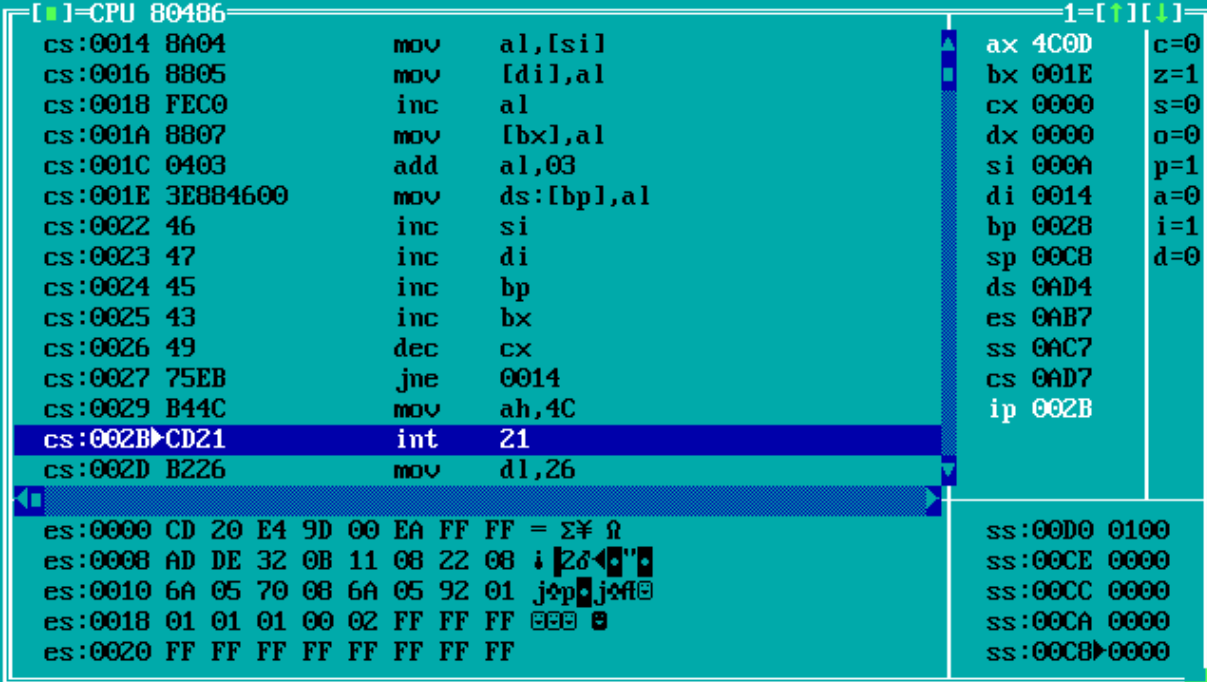
4.按F8分步执行观察记录第一问，按F2在INT 21处设置断点，按F9直接执行，并将ES段GOTO到DS:0000段观察记录第二问。

#### 1.2.2 实验记录与分析

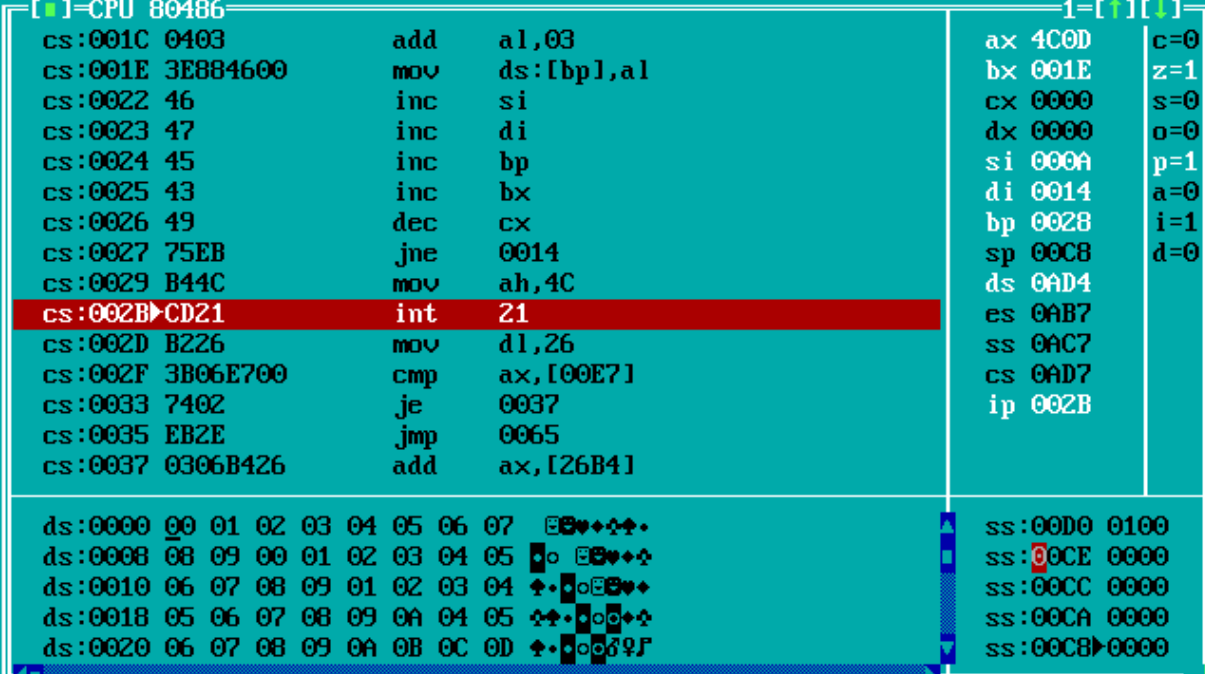
1.



执行到MOV CX,10 前的BX,BP,SI,DI分别为：0014、001E、0000、000A



执行到INT 21 前的BX,BP,SI,DI分别为：001E、0028、000A、0014

2. 

前40个数据段为：00 01 02 03 04 05 06 07 08 09 00 01 02 03 04 05 06 07 08 09 01 02 03 04 05 06 07 08 09 0A 04 05 06 07 08 09 0A 0B 0C 0D

正好对应BUF1 BUF2 BUF3 BUF4，与预期相同。

### 1.3 任务3

#### 1.3.1 实验步骤

1.首先修改实验2的源程序，将16为寄存器改为32位寄存器。由于32位寄存器低16位与16位寄存器相同，改为变址寻址方式时，为使功能不变，故将其偏移量置为0，故采用BUF1[]形式，修改的代码段如下：

MOV AL,BUF1[ESI]

MOV BUF1[EDI],AL

INC AL

MOV BUF1[EBX],AL

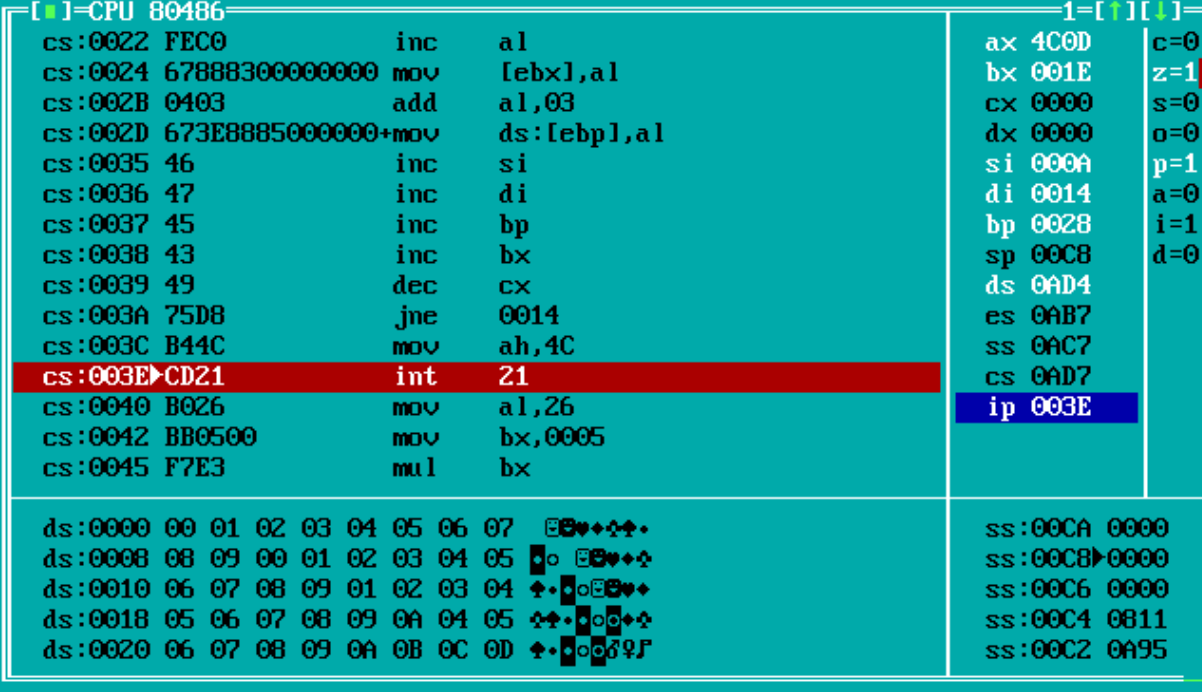
ADD AL,3

MOV DS:BUF1[EBP],AL

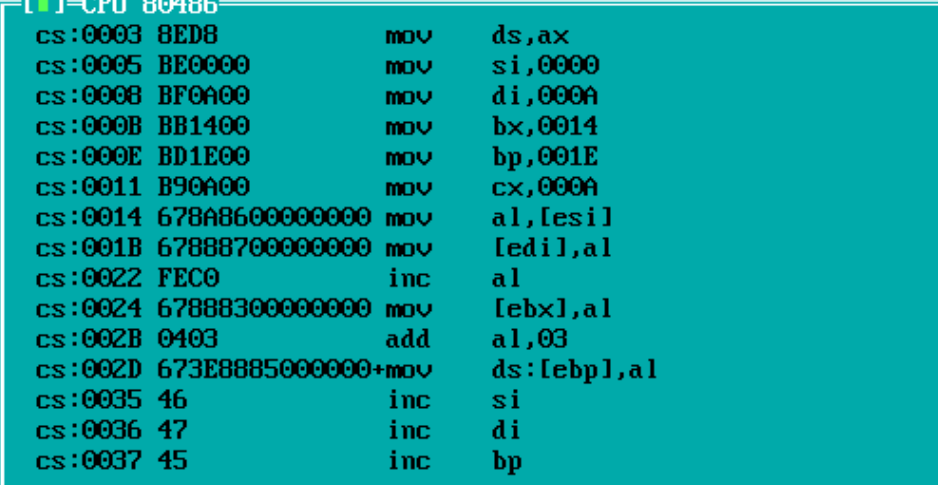
其余代码段同二

2.操作步骤同任务二

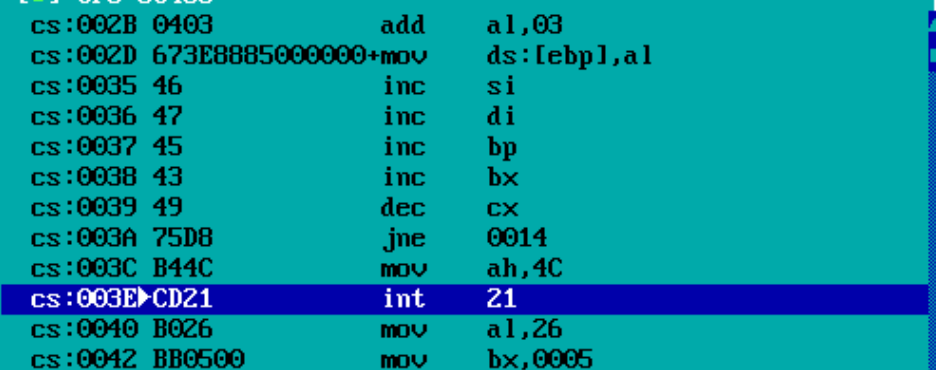
#### 1.3.2 实验记录与分析



可见数据段开始40个字节的内容与任务二一致，因为BUF1,BUF2,BUF3,BUF4并未改变



与2对比可见，在将16位寄存器改为32位寄存器的反汇编段，寄存器字节数明显变多，且每执行一句的偏移量也增加。



在执行完毕后，偏移地址比2明显增加，其余和2相同。

将反汇编语言与自己的代码进行对比得出以下结论：

1.反汇编语句的指令名称与源程序相同，寄存器名称也相同

2.反汇编语言中，输入的数据都成了进制形式，如add al,3 c成了 add al,03

3.字符串的名称成了偏移地址，如BUF1[ESI]显示的为[ESI]，因为BUF1的偏移地址为0，所以被省略了

4.LOPA和START也成了偏移地址，如JAE LOPA 成了 JNE 0014

### 1.4 任务1-3总结与体会

通过本次实验，我熟悉了DOSBOX的操作环境，学会了汇编语言的编译、运行以及调试的方法。通过对调试工具的运用，我对数、符号、寻址方式在计算机中的表示形式有了更深的理解，同时初步认识了汇编语言和反汇编语言的区别及联系。

在任务一的实验中，在我输入负数后，在调试工具中注意到计算机是以补码形式表示的，以前都是理论知识，现在有了一个直观的认识，并且在单步执行观察时，了解了源操作数和目的操作数的区别。

在任务二的实验中，我掌握了汇编程序从编写，到编译生成OBJ文件，再到生成可执行文件，再到调试的全部过程，在一开始编译时，我遇到了一个错误，通过返回信息找到了错误所在，是一个拼写错误，然后及时改正了，这也让我掌握了如何看错误返回信息来找到问题所在

通过任务三改变寻址方式以及采用不同的寄存器与任务二进行对比，我对计算机的寻址方式有了更深的认识，同时这两个任务也让我掌握了CHANGE 和GOTO 的功能，以及如何单步执行以及设置断点进行分段执行。

### 1.5 任务4

#### 1.5.1 设计思想及储存单元分配

1. 设计思想  
   数据段：用户名字符串  
    密码字符串  
    提示语句字符串  
    登陆状态AUTH  
    商品信息  
    用户输入用户名储存区：11字节  
    用户输入密码储存区：7字节  
    用户查找商品储存区：11字节  
   代码段：  
   模块一：登陆模块：  
    用户根据提示输入用户名，密码

根据输入的不同后续执行不同操作  
模块二：商品模块：  
 用户根据提示输入商品名  
 用双重循环方式在商品信息部分找到相应商品并显示  
模块三：推荐度计算模块：  
 根据公式计算推荐度并显示；  
在计算时使用的是字乘，字除操作，这种操作对结果的位数有一定限制，因此计算公式应该修正为：  
 128\*进价/实际售价+64\*销售量/进货量

若进价小于实际售价且销售量小于进货量则可以保证除法结果不会有溢出；

储存单元分配  
CS:CODE DS:DATA SS:STACK

变量定义:  
商品信息格式如下：  


输入储存区格式如下：  
IN\_NAME DB 11

DB ?

DB 11 DUP(0)

IN\_PWD DB 7

DB ?

DB 7DUP(0)

IN\_GA DB 10

DB ?

DB 7DUP(0)

#### 1.5.2 流程图

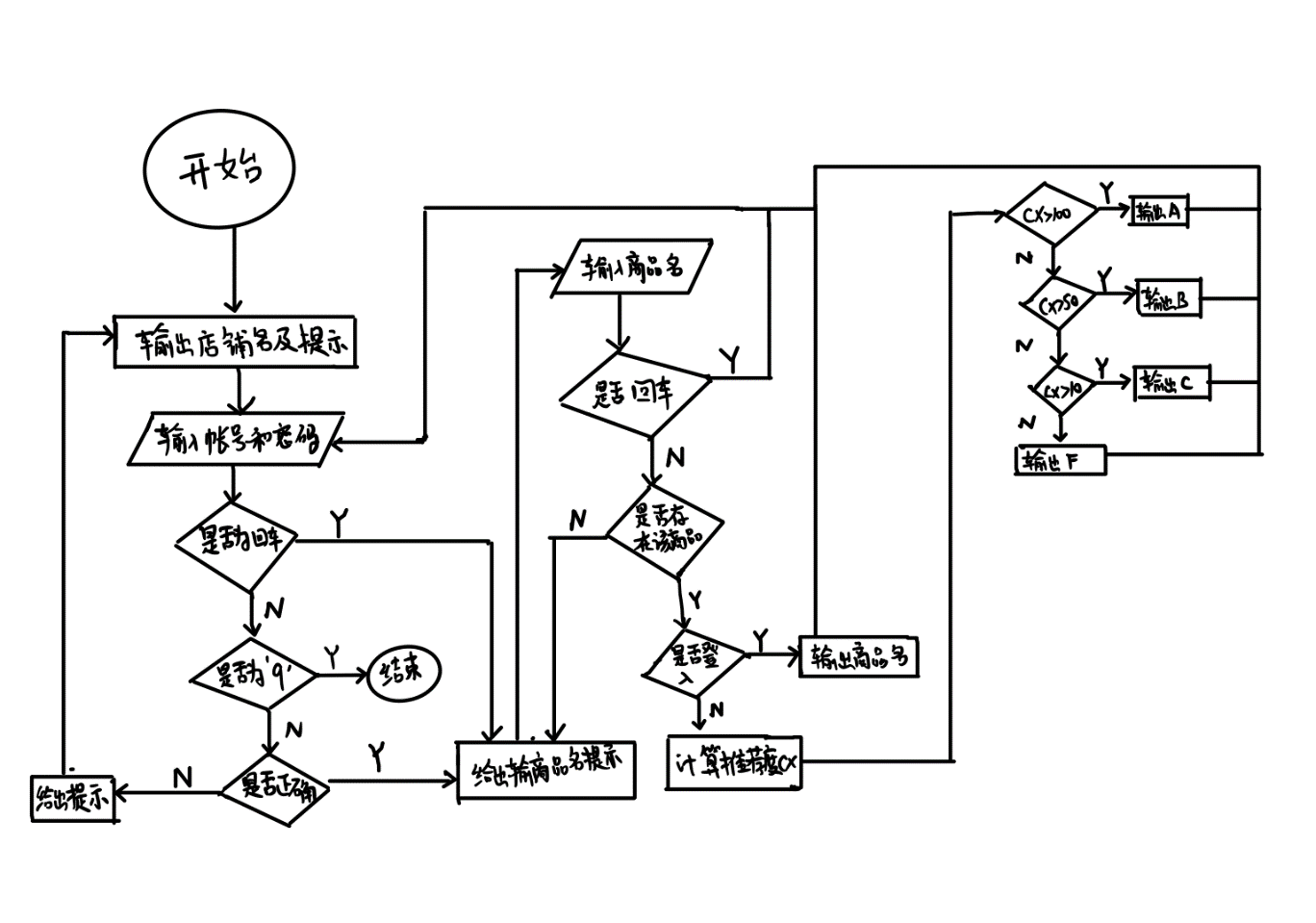


图4.1.程序流程图

#### 1.5.3 源程序

.386

DATA SEGMENT USE16

BNAME DB 'LIANGYIFEI' ;老板姓名

BPASS DB '991026' ;密码

SNAME DB 'FEISHOP',0DH,0AH,'$' ;网店名称，用0结束

N EQU 30

TAG DB '$'

POINT DB 'print username and password:',0DH,0AH,'$'

TIP DB 'MISMATCH!',0DH,0AH,'$'

TIP2 DB 'NOT FOUND!TRY AGAIN!',0DH,0AH,'$'

POINT2 DB 'Tell me what you want:',0DH,0AH,'$'

TAG2 DB 0DH,0AH,'$'

IN\_NAME DB 11

DB ?

DB 11 DUP(0)

IN\_PWD DB 7

DB ?

DB 7 DUP(0)

IN\_GA DB 11

DB 0

DB 11 DUP(0)

AUTH DB ?

I DB 0

J DB 0

GA1 DB 'PEN',7 DUP(0) ,10 ;商品名称及折扣

DW 35,56,70,25,? ;推荐度还未计算

GA2 DB 'BOOK', 6 DUP(0) ,9 ;商品名称及折扣

DW 12,30,25,5,? ;推荐度还未计算

GAN DB N-2 DUP('Temp-Value',8,15,0,20,0,30,0,2,0,?,?)

DATA ENDS

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT USE16

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

START:MOV AX,DATA

MOV DS,AX

LEA DX,SNAME

MOV AH,9

INT 21H

FUN1: LEA DX,POINT

MOV AH,9

INT 21H

LEA DX,IN\_NAME

MOV AH,10

INT 21H

LEA DX,TAG2

MOV AH,9

INT 21H

MOV CL,IN\_NAME[1]

CMP CL,0

JE T1

CMP CL,1

JE ISQ

JMP IPW

ISQ: MOV CL,IN\_NAME[2]

CMP CL,'q'

JE QUIT

IPW: LEA DX,IN\_PWD

MOV AH,10

INT 21H

LEA DX,TAG2

MOV AH,9

INT 21H

FUN2\_1: MOV CL,IN\_NAME[1]

CMP CL,10

JNE T2

MOV DI,10

JMP LOPA1

FUN2\_2: MOV CL,IN\_PWD[1]

CMP CL,6

JNE T2

MOV DI,6

JMP LOPA2

LOPA1:MOV CL,BNAME[DI-1]

CMP CL,IN\_NAME[DI+1]

JNE T2

DEC DI

CMP DI,0

JNE LOPA1

JMP FUN2\_2

LOPA2:MOV CL,BPASS[DI-1]

CMP CL,IN\_PWD[DI+1]

JNE T2

DEC DI

CMP DI,0

JNE LOPA2

MOV AUTH,1

FUN3\_1:LEA DX,POINT2

MOV AH,9

INT 21H

LEA DX,IN\_GA

MOV AH,10

INT 21H

LEA DX,TAG2

MOV AH,9

INT 21H

MOV CL,IN\_GA[1]

CMP CL,0

JE FUN1

MOVZX AX,CL

MOV DI,OFFSET IN\_GA

ADD DI,AX

ADD DI,2

MOV AL,0

MOV [DI],AL

MOV SI,OFFSET GA1

SUB SI,21

MOV DI,OFFSET IN\_GA

MOV I,N

JMP LOPA3

FUN3\_2:CMP AUTH,1

JNE FUN4

MOV CL,IN\_GA[1]

MOV BX,0

PRINT: ADD CL,-1

JS PRIN

MOV AH,02H

MOV DL,[SI][BX]

INT 21H

ADD BX,1

JMP PRINT

PRIN: LEA DX,TAG2

MOV AH,9

INT 21H

JMP CLEAR

LOPA3:ADD I,-1

JS T3

ADD SI,21

MOV CL,9

LOPA4:MOV BL,CL

MOV AL,2[DI+BX]

CMP AL,0[SI][BX]

JNE LOPA3

ADD CL,-1

JS FUN3\_2

JMP LOPA4

FUN4:MOV AX,WORD PTR [SI+11]

MOV BX,0080H

MUL BX

MOV BX,WORD PTR [SI+13]

DIV BX

MOV BX,000AH

MUL BX

MOV BL,[SI+10]

MOV BH,00H

DIV BX

MOV CX,AX

MOV AX,WORD PTR [SI+17]

MOV BX,0040H

MUL BX

MOV BX,WORD PTR [SI+15]

DIV BX

ADD AX,CX

MOV WORD PTR [SI+19],AX

CMP AX,100

JA ALEVEL

CMP AX,50

JA BLEVEL

CMP AX,10

JA CLEVEL

JMP FLEVEL

ALEVEL: LEA DX,'A'

MOV AH,2

INT 21H

LEA DX,TAG2

MOV AH,9

INT 21H

JMP CLEAR

BLEVEL: LEA DX,'B'

MOV AH,2

INT 21H

LEA DX,TAG2

MOV AH,9

INT 21H

JMP CLEAR

CLEVEL: LEA DX,'C'

MOV AH,2

INT 21H

LEA DX,TAG2

MOV AH,9

INT 21H

JMP CLEAR

FLEVEL: LEA DX,'F'

MOV AH,2

INT 21H

LEA DX,TAG2

MOV AH,9

INT 21H

JMP CLEAR

T1: MOV AUTH,0

JMP FUN3\_1

T2: LEA DX,TIP

MOV AH,9

INT 21H

JMP FUN1

T3: LEA DX,TIP2

MOV AH,9

INT 21H

MOV AX,0

MOV BX,0

MOV CX,0

MOV DX,0

MOV SI,0

MOV DI,0

JMP FUN3\_1

CLEAR:MOV AX,0

MOV BX,0

MOV CX,0

MOV DX,0

MOV SI,0

MOV DI,0

JMP FUN1

QUIT: MOV AH,4CH

INT 21H

CODE ENDS

END START

#### 1.5.4 实验步骤

1. 准备环境
2. 汇编，链接，运行
3. 测试游客输入：  
   在用户名输入处直接输入回车；  
   查找商品“PEN”
4. 测试商店老板输入：  
   输入正确用户名，密码  
   查找商品“BOOK”
5. 测试错误输入：  
   输入错误用户名；  
   输入正确用户名，错误密码  
   输入正确用户名，空密码  
   输入正确用户名，密码；查找商品“PENCEIL”（不存在的商品）

6. 根据测试中出现的错误完善程序

#### 1.5.5 实验记录与分析

1.游客输入：

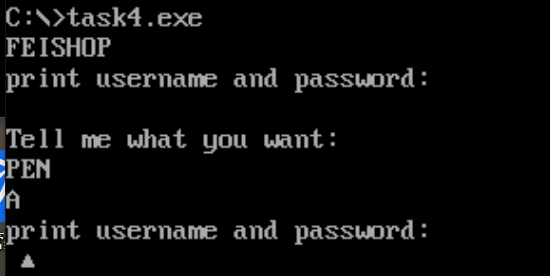


图4.2.游客输入

2.老板输入：

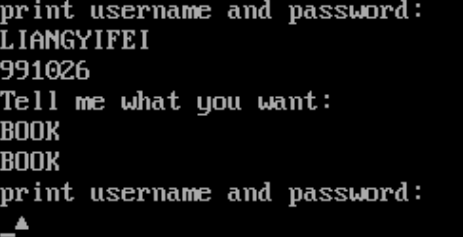


图4.3.老板输入

3.错误输入：

|  |  |
| --- | --- |
| 1.账号正确，密码错误 |  |
| 2.账号错误，密码正确 |  |
| 3.账号密码都错误 |  |
| 4.查找不存在的商品 |  |

根据以上结果可以看出程序的功能运行是正确的，但是在运行时还是出现了一些问题，问题如下：

1.在刚开始通过双重循环匹配商品时，无论输入什么都是没有对应的商品

2.在改正第一个问题后，匹配第一个商品可以输出正确的结果，之后就一直输出错误的结果或者程序直接崩溃

通过TD工具单步执行后，我发现了问题所在，并予以了改正

#### 1.5.6 程序修改

第一个问题：在TD窗口单步调试时，我先找到了IN\_GA的内存地址，观察里面储存的数据，我发现，在我输入PEN时，里面在E后面还存储了一个回车符，而我的匹配算法是从末尾一直往前比，在遇到回车符时，就会出现不匹配的情况，于是根据这个问题，我在输入后加了几行代码，将0覆盖回车符的位置，于是这个问题得到了解决

第二个问题：依旧是通过TD窗口单步调试，我观察到在输入第一个商品名并输出正确的结果后，寄存器里面的内容并没有清空，而在我输入第二个商品名时，第一个商品送到寄存器里的内容也会发生作用，所以会导致第二个商品出现一些错误，所以每次输出后我都让它跳转到一个叫做CLEAR的标号，里面的代码的作用是清空寄存器的内容，然后再跳回来，这样，问题便得到了解决

### 1.6 任务4总结与体会

任务4是第一个用汇编语言写的中型程序，在编写的时候，遇到了许多问题，第一个问题就是输入输出问题，在刚开始时，我输入和输出后面都没有加上回车和换行符号，所以导致UI界面非常的不美观，在第一次运行程序时及时进行了补充。然后在登入状态下要输出商品的名字，又令我感到为难，一开始打算用9号功能输出，可是我要输出的IN\_GA里面并没有$符号，于是放弃这个想法，改用2号功能单字符输出，我先用SI获取了IN\_GA的偏移地址，然后用CL获取长度，再用BX控制循环，用基址变址寻址方式找到要输出的商品名的第一个字符，于是顺利输出了商品名。第二个问题就是如何匹配商品名，在这个部分，我知道了在输入时，回车符号也会读进变量里面，这给我的匹配带了大麻烦，一开始我怀疑是我的匹配算法有问题，但是经过细致的推导，确信匹配算法没有问题，于是只好用TD工具进行调试，终于发现我存储商品名的变量里面多了一个回车符号，于是我将0送入回车符号的位置，问题得到解决。第三个问题就是计算推荐度，在计算推荐度时，用报告中给出的原始公式形式会存在很大的误差，因为汇编语言除法运算的余数部分以我现在的知识水平还不太好处理，于是对公式进行了优化，这样在一定的范围内，误差就可以忽略不计了。还有一些其他的小问题，例如MOV语句的两个寄存器不匹配，或者加减乘除的两个数据类型不匹配，在编译报错后，我根据返回的错误行一一进行了修改。程序刚设计好时，还是出现了一些问题，例如没有清空缓存区导致程序只能‘一次’正确运行，为了让程序尽善尽美，我又对此对代码进行了优化。总而言之，这次实验里出现的种种问题，都让我对汇编语言的认识深入了一层。

# 实验二 程序优化

**实验名称： 程序优化**

**实验地点： 南一楼804室**

**指导教师： 张勇**

**专业班级：计算机科学与技术201706班**

**学 号： U2015414556 姓 名： 梁一飞**

**同组学生： 无 报告日期： 2019年 3 月 28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： 纸质版再签名

日期：2017.3.17

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：

**实验目的与要求**

了解程序计时的方法以及运行环境对程序执行情况的影响。

熟悉汇编语言指令的特点，掌握代码优化的基本方法。

## 实验内容

**任务1. 观察多重循环对CPU计算能力消耗的影响**

应用场景介绍：以实验一任务4的背景为基础，只要有一个顾客访问网店中的商品，系统就需要计算一遍所有商品的推荐度（**本次实验都要按照此需求计算推荐度**），然后再处理顾客实际购买的商品的信息。现假设在双十一零点时，SHOP网店中的“Bag”商品共有m件，有m个顾客几乎同时下单购买了该商品。请模拟后台处理上述信息的过程并观察执行的时间。

上述场景的后台处理过程，可以理解为在同一台电脑上有m个请求一起排队使用实验一任务4的程序。为了观察从第1个顾客开始进入购买至第m个顾客购买完毕之间到底花费了多少时间，我们让实验一任务4的功能三调整后的代码重复执行m次，通过计算这m次循环执行前和执行后的时间差，来感受其影响。功能三之外的其他功能不纳入到这m次循环体内（但可以保留不变）。

**调整后的功能三的描述：**

（1）提示用户输入要购买的商品名称（比如“Bag”）。【此后可插入计时、循环】

（2）计算SHOP中所有商品的推荐度。

（3）在SHOP中找到顾客购买的商品（比如“Bag”，若未能找到该商品，回到（1）重新输入。若只输入回车，则回到功能一（1））。

（4）判断该商品已售数量是否大于等于进货总数，若是，则回到功能一（1），否则将已售数量加1。 【循环控制，计时结束】

（5）回到功能三（1）。

请按照上述设想修改实验一任务4的程序，并将m和n值尽量取大（比如大于1000，具体数值依据实验效果来改变，逐步增加到比较明显的程度，比如秒级的时间间隔。另外，也可以把定义“Bag”的位置放在所有商品的最后，使得搜索它的时间变长），以得到较明显的效果。

**提示:** 学校汇编教学网站的软件下载中提供了显示当前时间“秒和百分秒”的子程序。若在m次循环前调用一下该子程序，m次循环执行完之后再调用一下该子程序，就能在屏幕上观察并感受到执行循环前后的时间差（时间差值需要自行手工计算，当然，你也可以选用网站上另一个计时程序，它是可以帮你计算好差值的）。注意，由于虚拟机环境下CPU会被分时调度，故该时间差值会因计算机运行环境、状态以及虚拟机的设置参数的不同而不同。

**任务2.** 对任务1中的汇编源程序进行优化

优化工作包括代码长度的优化和执行效率的优化，本次优化的重点是执行效率的优化。请通过优化m次循环体内的程序，使程序的执行时间尽可能减少10%以上（注意，在编写任务1的程序时，尽量不要考虑代码优化的问题）。

**优化方法提示：**首先是通过选择执行速度较快的指令来提高性能，比如，把乘除指令转换成移位指令、加法指令等；其次，内循环体中每减少一条指令，就相当于减少了m\*n条指令的执行时间，需要仔细斟酌；第三，在寻址方式中尽量把16位寄存器换成32位寄存器，能有**更多的机会和技巧**提高指令执行效率。**注意：由于本课程关注的是指令级别的影响，因此，请大家不要只考虑纯算法级别的优化。**

## 实验过程

### 2.1 任务1

#### 2.1.1设计思想

在实验一任务4的基础上，我做了如下修改：1.原实验四的功能四为计算商品推荐度，在找到商品后进行，现在改为找商品前进行，原来为计算一个商品，现在为计算所有商品，由一个循环控制 2.原实验四为只找一次商品，现在改为找M次，M为自定义次数，且每次找到商品后则跳转至计算全部商品推荐度的模块。修改后的程序框图如下图所示：

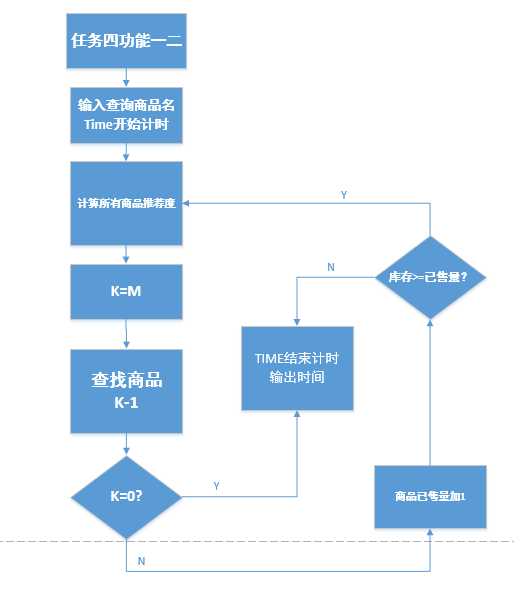


图1.程序流程图

其中计算所有商品的推荐度为一个N重循环，由一个变量I控制，N为自定义数字。

#### 2.1.2 源程序

.386

DATA SEGMENT USE16

BNAME DB 'LIANGYIFEI' ;老板姓名

BPASS DB '991026' ;密码

SNAME DB 'FEISHOP',0DH,0AH,'$' ;网店名称，用0结束

N EQU 1000

M EQU 1000

TAG DB '$'

POINT DB 'print username and password:',0DH,0AH,'$'

TIP DB 'MISMATCH!',0DH,0AH,'$'

TIP2 DB 'NOT FOUND!TRY AGAIN!',0DH,0AH,'$'

POINT2 DB 'Tell me what you want:',0DH,0AH,'$'

TAG2 DB 0DH,0AH,'$'

IN\_NAME DB 11

DB ?

DB 11 DUP(0)

IN\_PWD DB 7

DB ?

DB 7 DUP(0)

IN\_GA DB 11

DB 0

DB 11 DUP(0)

AUTH DB ?

I DW 0

J DB 0

K DW 0

GA1 DB 'PEN',7 DUP(0) ,10 ;商品名称及折扣

DW 35,56,70,25,? ;推荐度还未计算

GA2 DB 'BOOK', 6 DUP(0) ,9 ;商品名称及折扣

DW 12,30,25,5,? ;推荐度还未计算

GAN DB N-3 DUP('Temp-Value',8,15,0,20,0,30,0,2,0,?,?)

GA3 DB 'BAG',7 DUP(0), 8

DW 15,20,20000,2,?

DATA ENDS

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT USE16

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

START:MOV AX,DATA

MOV DS,AX

LEA DX,SNAME

MOV AH,9

INT 21H

FUN1: LEA DX,POINT

MOV AH,9

INT 21H

LEA DX,IN\_NAME

MOV AH,10

INT 21H

LEA DX,TAG2

MOV AH,9

INT 21H

MOV CL,IN\_NAME[1]

CMP CL,0

JE T1

CMP CL,1

JE ISQ

JMP IPW

ISQ: MOV CL,IN\_NAME[2]

CMP CL,'q'

JE QUIT

IPW: LEA DX,IN\_PWD

MOV AH,10

INT 21H

LEA DX,TAG2

MOV AH,9

INT 21H

FUN2\_1: MOV CL,IN\_NAME[1]

CMP CL,10

JNE T2

MOV DI,10

JMP LOPA1

FUN2\_2: MOV CL,IN\_PWD[1]

CMP CL,6

JNE T2

MOV DI,6

JMP LOPA2

LOPA1:MOV CL,BNAME[DI-1]

CMP CL,IN\_NAME[DI+1]

JNE T2

DEC DI

CMP DI,0

JNE LOPA1

JMP FUN2\_2

LOPA2:MOV CL,BPASS[DI-1]

CMP CL,IN\_PWD[DI+1]

JNE T2

DEC DI

CMP DI,0

JNE LOPA2

MOV AUTH,1

FUN3\_1:LEA DX,POINT2

MOV AH,9

INT 21H

MOV AX,0

LEA DX,IN\_GA

MOV AH,10

INT 21H

LEA DX,TAG2

MOV AH,9

INT 21H

MOV AX,0

CALL TIMER;开始计时

MOV CL,IN\_GA[1]

CMP CL,0

JE FUN1

MOVZX AX,CL

MOV DI,OFFSET IN\_GA

ADD DI,AX

ADD DI,2

MOV AL,0

MOV [DI],AL

CMP AUTH,1

JE FUN3\_3

MOV K,M

FUN3\_4:ADD K,-1

JS TIME

JMP FUN5

FUN3\_3:MOV SI,OFFSET GA1

SUB SI,21

MOV DI,OFFSET IN\_GA

MOV I,N

JMP LOPA3

FUN3\_2:CMP AUTH,1

JNE FUN4

MOV CL,IN\_GA[1]

MOV BX,0

PRINT: ADD CL,-1

JS PRIN

MOV AH,02H

MOV DL,[SI][BX]

INT 21H

ADD BX,1

JMP PRINT

PRIN: LEA DX,TAG2

MOV AH,9

INT 21H

JMP CLEAR

LOPA3:ADD I,-1

JS T3

ADD SI,21

MOV CL,9

LOPA4:MOV BL,CL

MOV AL,2[DI+BX]

CMP AL,0[SI][BX]

JNE LOPA3

ADD CL,-1

JS FUN3\_2

JMP LOPA4

FUN4:MOV AX,WORD PTR[SI+19]

CMP AX,100

JA ALEVEL

CMP AX,50

JA BLEVEL

CMP AX,10

JA CLEVEL

JMP FLEVEL

FUN4\_1:MOV AX,0

MOV AX,WORD PTR [SI+15]

SUB AX,WORD PTR[SI+17]

JS TIME

ADD WORD PTR [SI+17],1

MOV AX,0

JMP FUN3\_4

TIME: MOV AX,1

CALL TIMER

JMP FUN3\_1

FUN5:MOV SI,OFFSET GA1;计算全部商品的推荐度

ADD SI,-21

MOV I,N

LOPA5:ADD I,-1

JS FUN3\_3

ADD SI,21

MOV AX,WORD PTR [SI+11]

MOV BX,0080H

MUL BX

MOV BX,WORD PTR [SI+13]

DIV BX

MOV BX,000AH

MUL BX

MOV BL,[SI+10]

MOV BH,00H

DIV BX

MOV CX,AX

MOV AX,WORD PTR [SI+17]

MOV BX,0040H

MUL BX

MOV BX,WORD PTR [SI+15]

DIV BX

ADD AX,CX

MOV WORD PTR [SI+19],AX

MOV AX,0

MOV BX,0

MOV CX,0

JMP LOPA5

ALEVEL: LEA DX,'A'

MOV AH,2

INT 21H

LEA DX,' '

MOV AH,2

INT 21H

JMP FUN4\_1

BLEVEL: LEA DX,'B'

MOV AH,2

INT 21H

LEA DX,' '

MOV AH,2

INT 21H

JMP FUN4\_1

CLEVEL: LEA DX,'C'

MOV AH,2

INT 21H

LEA DX,' '

MOV AH,2

INT 21H

JMP FUN4\_1

FLEVEL: LEA DX,'F'

MOV AH,2

INT 21H

LEA DX,' '

MOV AH,2

INT 21H

JMP FUN4\_1

T1: MOV AUTH,0

JMP FUN3\_1

T2: LEA DX,TIP

MOV AH,9

INT 21H

JMP FUN1

T3: LEA DX,TIP2

MOV AH,9

INT 21H

MOV AX,0

MOV BX,0

MOV CX,0

MOV DX,0

MOV SI,0

MOV DI,0

MOV BX,11

MOV SI ,OFFSET IN\_GA

CLEAR2:MOV BYTE PTR[SI+BX],0

ADD BX,-1

CMP BX,0

JNE CLEAR2

MOV SI,0

JMP FUN3\_1

CLEAR:MOV AX,0

MOV BX,0

MOV CX,0

MOV DX,0

MOV SI,0

MOV DI,0

MOV BX,11

MOV SI, OFFSET IN\_GA

CLEAR1: MOV BYTE PTR[SI+BX],0

ADD BX,-1

CMP BX,0

JNE CLEAR1

MOV SI,0

JMP FUN1

QUIT: MOV AH,4CH

INT 21H

;时间计数器(ms),在屏幕上显示程序的执行时间(ms)

;使用方法:

; MOV AX, 0 ;表示开始计时

; CALL TIMER

; ... ... ;需要计时的程序

; MOV AX, 1

; CALL TIMER ;终止计时并显示计时结果(ms)

;输出: 改变了AX和状态寄存器

TIMER PROC

PUSH DX

PUSH CX

PUSH BX

MOV BX, AX

MOV AH, 2CH

INT 21H ;CH=hour(0-23),CL=minute(0-59),DH=second(0-59),DL=centisecond(0-100)

MOV AL, DH

MOV AH, 0

IMUL AX,AX,1000

MOV DH, 0

IMUL DX,DX,10

ADD AX, DX

CMP BX, 0

JNZ \_T1

MOV CS:\_TS, AX

\_T0: POP BX

POP CX

POP DX

RET

\_T1: SUB AX, CS:\_TS

JNC \_T2

ADD AX, 60000

\_T2: MOV CX, 0

MOV BX, 10

\_T3: MOV DX, 0

DIV BX

PUSH DX

INC CX

CMP AX, 0

JNZ \_T3

MOV BX, 0

\_T4: POP AX

ADD AL, '0'

MOV CS:\_TMSG[BX], AL

INC BX

LOOP \_T4

PUSH DS

MOV CS:\_TMSG[BX+0], 0AH

MOV CS:\_TMSG[BX+1], 0DH

MOV CS:\_TMSG[BX+2], '$'

LEA DX, \_TS+2

PUSH CS

POP DS

MOV AH, 9

INT 21H

POP DS

JMP \_T0

\_TS DW ?

DB 'Time elapsed in ms is '

\_TMSG DB 12 DUP(0)

TIMER ENDP

CODE ENDS

END START

#### 2.1.3 实验步骤

1. 准备环境
2. 汇编，链接，运行
3. 测试不会售空输入
4. 测试售空输入
5. 取M=1000,N=300；M=800,N=500；M=800,N=700；M=1000,N=1000；4组值分别测试查找“BAG”的用时。
6. 不同运行环境的影响:CPU速度为3000Hz,5000Hz,7000Hz,10000Hz的4种运行环境，测试查找“BAG”的用时，寻找用时和CPU速度的关系.
7. 尝试多种时间获得方式

#### 2.1.4 实验记录与分析

1.在M=1000,N=1000时，将商品库存量改为20000，测试不会售空输入，结果如下：



图1.不会售空测试

其中的A为商品推荐度，可根据A的数量来判断循环次数是否正确，经过验证，是正确的

2.售空测试

在M=1000,N=1000时，将商品库存改为100，测试售空输入，结果如下：

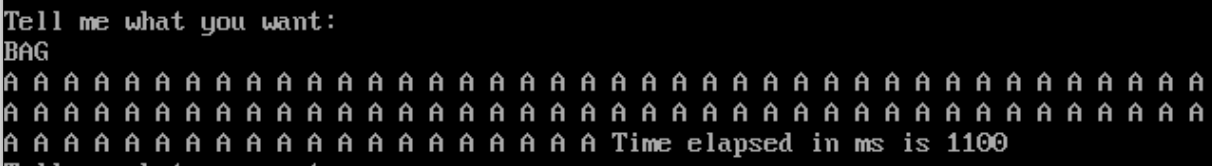


图2.售空测试

可见，在售空的情况下，总循环次数等于商品剩余库存量

3.分别取M=1000,N=300；M=800,N=500；M=800,N=700；M=1000,N=1000；4组值分别测试查找“BAG”的用时。

表1.M、N测试

|  |  |  |
| --- | --- | --- |
| M | N | 用时 |
| 1000 | 300 |  |
| 1000 | 1000 |  |
| 800 | 500 |  |
| 800 | 800 |  |

作图分析如下：

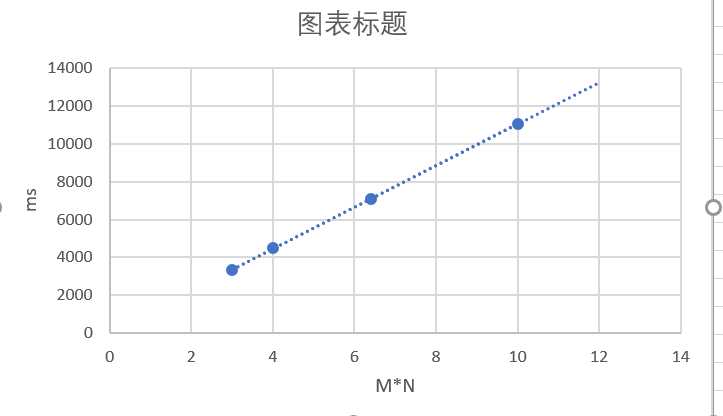


图3.时间关系图

可见，M\*N与时间为线性关系，符合时间复杂度O（M\*N）

4.取 CPU速度为3000Hz,5000Hz,7000Hz,10000Hz的4种运行环境，测试查找“BAG”的用时，寻找用时和CPU速度的关系.

表2.CPU速度与用时关系

|  |  |
| --- | --- |
| CPU | 用时 |
| 3000HZ |  |
| 5000HZ |  |
| 7000HZ |  |
| 10000HZ |  |

作图分析如下：

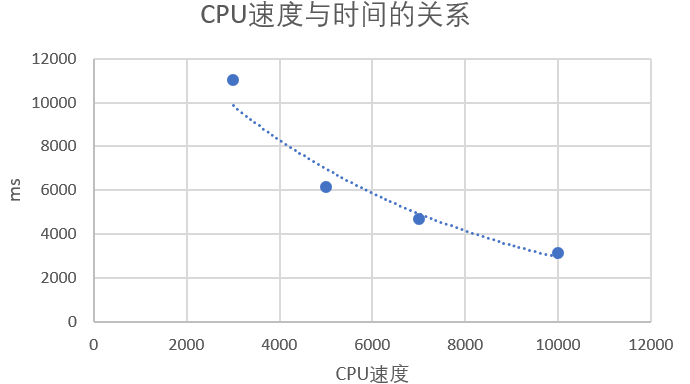


图4.CPU速度与时间关系图

由图可见，CPU速度与时间成反比。当用TD调试工具运行时，时间仅为120ms，CPU主频为264.7KHZ,故TD调试工具与虚拟器的CPU并不相同。

5.采取不同的方法获取时间

（1）汇编网站所提供的timer子程序，精度为ms

（2）用RDTSC命令，将64位机器周期读入EDX:EAX中，然后入栈记录数据。进入循环前以及循环结束后分别使用一次，做差后得到两次调用时的机器周期差。又由图3.1.5可知



图5. DOSBox基础参数图示

DOSBox0.74版本的虚拟机CPU主频为3000Hz，于是机器周期差除以3000即可得到循环用时精度为ms

（3）访问40H:4CH处的数据，该数据每55ms增加1，因此将得到的数据做差后乘55即可得到耗时，精度为0.1s。这种时间获得方式最高可计算时间差为14.08s，对于测试来说，缺陷太大。

### 2.2 任务2

#### 2.2.1 设计思想

我主要用了以下两种方法优化时间效率：

1.将控制循环的变量改为用寄存器控制：由于本次实验总共有两个大循环：查找循环，计算循环。而这两个循环我分别使用了两个内存变量来控制，如果改为寄存器，将更加节省时间，可惜的是在程序设计过程中，我用到了AX,BX,CX,DX,DI,SI六个寄存器，已经无法空出两个多余的寄存器用来控制循环，因为我没有用到栈，所以我把控制查找循环的变量改为用BP寄存器来控制，计算循环仍然由一个内存变量控制。

2.优化推荐度计算过程：由于我推荐度计算中的乘法都是由MUL语句执行，所以我改为了移位来优化时间效率。将乘128改为了左移7位，乘64改为了左移6位。

3.其他优化：去掉了内存中省下来的变量，优化了部分语句，如取消的DEC DI后的CMP DI,0 ，直接用JS语句控制跳转，另外省略了不必要的输出：每次计算后输出物品推荐度。

#### 2.2.2 源程序

.386

DATA SEGMENT USE16

BNAME DB 'LIANGYIFEI' ;老板姓名

BPASS DB '991026' ;密码

SNAME DB 'FEISHOP',0DH,0AH,'$' ;网店名称，用0结束

N EQU 1000

M EQU 1000

TAG DB '$'

POINT DB 'print username and password:',0DH,0AH,'$'

TIP DB 'MISMATCH!',0DH,0AH,'$'

TIP2 DB 'NOT FOUND!TRY AGAIN!',0DH,0AH,'$'

POINT2 DB 'Tell me what you want:',0DH,0AH,'$'

TAG2 DB 0DH,0AH,'$'

IN\_NAME DB 11

DB ?

DB 11 DUP(0)

IN\_PWD DB 7

DB ?

DB 7 DUP(0)

IN\_GA DB 11

DB 0

DB 11 DUP(0)

AUTH DB ?

I DW 0

K DW 0

GA1 DB 'PEN',7 DUP(0) ,10 ;商品名称及折扣

DW 35,56,70,25,? ;推荐度还未计算

GA2 DB 'BOOK', 6 DUP(0) ,9 ;商品名称及折扣

DW 12,30,25,5,? ;推荐度还未计算

GAN DB N-3 DUP('Temp-Value',8,15,0,20,0,30,0,2,0,?,?)

GA3 DB 'BAG',7 DUP(0), 8

DW 15,20,20000,2,?

DATA ENDS

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT USE16

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

START:MOV AX,DATA

MOV DS,AX

LEA DX,SNAME

MOV AH,9

INT 21H

FUN1: LEA DX,POINT

MOV AH,9

INT 21H

LEA DX,IN\_NAME

MOV AH,10

INT 21H

LEA DX,TAG2

MOV AH,9

INT 21H

MOV CL,IN\_NAME[1]

CMP CL,0

JE T1

CMP CL,1

JE ISQ

JMP IPW

ISQ: MOV CL,IN\_NAME[2]

CMP CL,'q'

JE QUIT

IPW: LEA DX,IN\_PWD

MOV AH,10

INT 21H

LEA DX,TAG2

MOV AH,9

INT 21H

FUN2\_1: MOV CL,IN\_NAME[1]

CMP CL,10

JNE T2

MOV DI,10

JMP LOPA1

FUN2\_2: MOV CL,IN\_PWD[1]

CMP CL,6

JNE T2

MOV DI,6

JMP LOPA2

LOPA1:MOV CL,BNAME[DI-1]

CMP CL,IN\_NAME[DI+1]

JNE T2

DEC DI

JNS LOPA1

JMP FUN2\_2

LOPA2:MOV CL,BPASS[DI-1]

CMP CL,IN\_PWD[DI+1]

JNE T2

DEC DI

JNS LOPA2

MOV AUTH,1

FUN3\_1:LEA DX,POINT2

MOV AH,9

INT 21H

MOV AX,0

LEA DX,IN\_GA

MOV AH,10

INT 21H

LEA DX,TAG2

MOV AH,9

INT 21H

MOV AX,0

CALL TIMER;开始计时

MOV CL,IN\_GA[1]

CMP CL,0

JE FUN1

MOVZX AX,CL

MOV DI,OFFSET IN\_GA

ADD DI,AX

ADD DI,2

MOV AL,0

MOV [DI],AL

CMP AUTH,1

JE FUN3\_3

MOV K,M

FUN3\_4:ADD K,-1

JS TIME

JMP FUN5

FUN3\_3:MOV SI,OFFSET GA1

SUB SI,21

MOV DI,OFFSET IN\_GA

MOV BP,N

JMP LOPA3

FUN3\_2:CMP AUTH,1

JNE FUN4

MOV CL,IN\_GA[1]

MOV BX,0

PRINT: ADD CL,-1

JS PRIN

MOV AH,02H

MOV DL,[SI][BX]

INT 21H

ADD BX,1

JMP PRINT

PRIN: LEA DX,TAG2

MOV AH,9

INT 21H

JMP CLEAR

LOPA3:ADD BP,-1

JS T3

ADD SI,21

MOV CL,9

LOPA4:MOV BL,CL

MOV AL,2[DI+BX]

CMP AL,0[SI][BX]

JNE LOPA3

ADD CL,-1

JS FUN3\_2

JMP LOPA4

FUN4: MOV AX,0

MOV AX,WORD PTR [SI+15]

SUB AX,WORD PTR[SI+17]

JS TIME

ADD WORD PTR [SI+17],1

MOV AX,0

JMP FUN3\_4

TIME: MOV AX,1

CALL TIMER

JMP FUN3\_1

FUN5:MOV SI,OFFSET GA1;计算全部商品的推荐度

ADD SI,-21

MOV BP,N

LOPA5:ADD BP,-1

JS FUN3\_3

ADD SI,21

MOV AX,WORD PTR [SI+11]

SHL AX,7

MOV DX,0

MOV BX,WORD PTR [SI+13]

DIV BX

MOV BX,000AH

MUL BX

MOV BL,[SI+10]

MOV BH,00H

DIV BX

MOV CX,AX

MOV AX,WORD PTR [SI+17]

SHL AX,6

MOV BX,WORD PTR [SI+15]

DIV BX

ADD AX,CX

MOV WORD PTR [SI+19],AX

MOV AX,0

MOV BX,0

MOV CX,0

JMP LOPA5

T1: MOV AUTH,0

JMP FUN3\_1

T2: LEA DX,TIP

MOV AH,9

INT 21H

JMP FUN1

T3: LEA DX,TIP2

MOV AH,9

INT 21H

MOV AX,0

MOV BX,0

MOV CX,0

MOV DX,0

MOV SI,0

MOV DI,0

MOV BX,11

MOV SI ,OFFSET IN\_GA

CLEAR2:MOV BYTE PTR[SI+BX],0

ADD BX,-1

CMP BX,0

JNE CLEAR2

MOV SI,0

JMP FUN3\_1

CLEAR:MOV AX,0

MOV BX,0

MOV CX,0

MOV DX,0

MOV SI,0

MOV DI,0

MOV BX,11

MOV SI, OFFSET IN\_GA

CLEAR1: MOV BYTE PTR[SI+BX],0

ADD BX,-1

CMP BX,0

JNE CLEAR1

MOV SI,0

JMP FUN1

QUIT: MOV AH,4CH

INT 21H

;时间计数器(ms),在屏幕上显示程序的执行时间(ms)

;使用方法:

; MOV AX, 0 ;表示开始计时

; CALL TIMER

; ... ... ;需要计时的程序

; MOV AX, 1

; CALL TIMER ;终止计时并显示计时结果(ms)

;输出: 改变了AX和状态寄存器

TIMER PROC

PUSH DX

PUSH CX

PUSH BX

MOV BX, AX

MOV AH, 2CH

INT 21H ;CH=hour(0-23),CL=minute(0-59),DH=second(0-59),DL=centisecond(0-100)

MOV AL, DH

MOV AH, 0

IMUL AX,AX,1000

MOV DH, 0

IMUL DX,DX,10

ADD AX, DX

CMP BX, 0

JNZ \_T1

MOV CS:\_TS, AX

\_T0: POP BX

POP CX

POP DX

RET

\_T1: SUB AX, CS:\_TS

JNC \_T2

ADD AX, 60000

\_T2: MOV CX, 0

MOV BX, 10

\_T3: MOV DX, 0

DIV BX

PUSH DX

INC CX

CMP AX, 0

JNZ \_T3

MOV BX, 0

\_T4: POP AX

ADD AL, '0'

MOV CS:\_TMSG[BX], AL

INC BX

LOOP \_T4

PUSH DS

MOV CS:\_TMSG[BX+0], 0AH

MOV CS:\_TMSG[BX+1], 0DH

MOV CS:\_TMSG[BX+2], '$'

LEA DX, \_TS+2

PUSH CS

POP DS

MOV AH, 9

INT 21H

POP DS

JMP \_T0

\_TS DW ?

DB 'Time elapsed in ms is '

\_TMSG DB 12 DUP(0)

TIMER ENDP

CODE ENDS

END START

#### 2.2.3 实验步骤

1.准备环境

2.汇编，链接，运行

3.取M=1000,N=300；M=800,N=500；M=800,N=700；M=1000,N=1000；4组值分别测试查找“BAG”的用时，与优化前作对比

4.不同运行环境的影响:CPU速度为3000Hz,5000Hz,7000Hz,10000Hz的4种运行环境，测试查找“BAG”的用时，与优化前作对比

#### 2.2.4 实验记录与分析

1.在M=1000，N=1000的情况下，运行程序，得到结果如图：

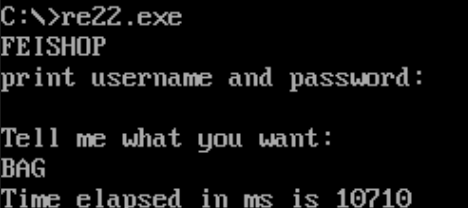


图6.优化结果图

得到的优化效率为：2.9%

2.改变M和N,测试时间

表3.优化时间表

|  |  |  |
| --- | --- | --- |
| M | N | 时间 |
| 1000 | 300 |  |
| 1000 | 1000 |  |
| 800 | 500 |  |
| 800 | 800 |  |

作对比图分析：

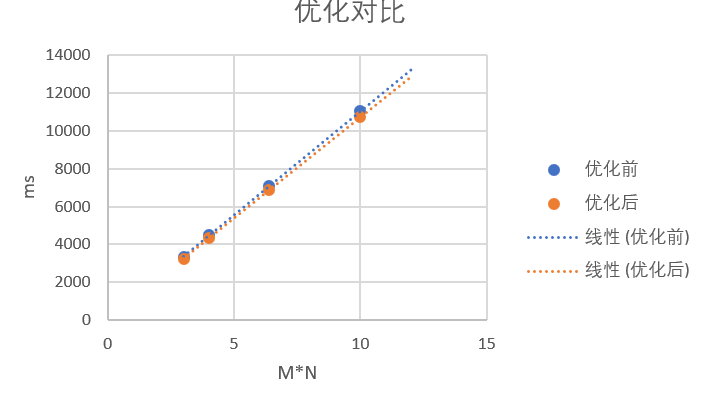


图7.优化对比图

可见，优化效率虽然不大，但是优化是稳定的。

3.改变CPU主频，测试时间

|  |  |
| --- | --- |
| CPU | 用时 |
| 3000HZ |  |
| 5000HZ |  |
| 7000HZ |  |
| 10000HZ |  |

做出对比图如下：

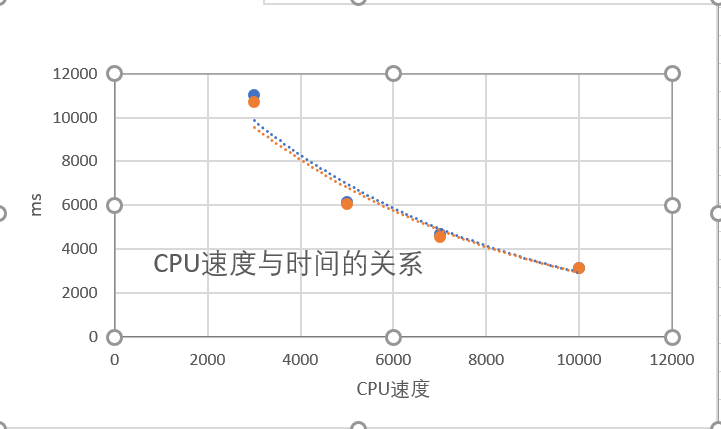


图8.优化前后对比图

由图可以看出，在CPU主频非常高的情况下，优化的时间可以忽略不记。

### 2.3. 总结与体会

在本次实验过程中，由于任务一是在上次实验任务四的基础上完成的，所以很快便做好的，主要问题在任务二的优化上，一开始我将乘法都改成移位，但是随之而来造成了程序崩溃，在仔细研究了乘除法功能并在TD工具里查看AX,DX寄存器的内容，我发现了问题。原先使用MUL语句进行运算时，由于AX寄存器足够储存乘法运算后的结果，所以DX中的数据被置为0，接下来进行除法运算时，就不会发生错误。但是改为移位后，AX的内容虽然和直接用MUL的结果一样，但是DX寄存器里的内容却并没有发生改变，故接下来在进行除法运算时，计算机会认为DX里储存的是乘法运算的高位，所以除法就发生了错误，于是，在移位语句后，我加了一条MOV DX,0 ，顺利解决了这个问题。第二个问题就是将控制循环的变量改为寄存器，这个就让我有些后悔在动手写上次实验的任务四时没有对寄存器进行合理的规划，导致6个基本的寄存器都被使用了，再进行修改的化会影响到许多地方，于是只好将栈的寄存器BP拿来计数，因为全程没有用到栈，优化效果也不是很高，仅仅优化了60ms，移位则优化了300ms。本次实验的优化效率不是很高，我认为问题出在我没有对程序用到的寄存器进行合理的安排，导致想要优化时没有寄存器可用。这也提醒了我，以后动手写汇编代码时，先提前安排好每个寄存器的功能，留出多余的寄存器，以便于优化。

# 实验三 模块化程序设计

**实验名称： 模块化程序设计**

**实验地点： 南一楼804室**

**指导教师： 张勇**

**专业班级：计算机科学与技术201706班**

**学 号： U2015414556 姓 名： 梁一飞**

**同组学生： 胡晨风 报告日期： 2019年 4 月 12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： 纸质版再签名

日期：2017.3.17

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：

**实验目的和要求**

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；

掌握宏指令、模块化程序的设计方法;

掌握较大规模程序的合作开发与调试方法；

掌握汇编语言程序与C语言程序混合编程的方法；

了解C编译器的基本优化方法;

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1 宏与子程序设计 （尽量在第一次4个课内学时阶段完成，上机前实验报告应完成该任务中除实验记录与分析、总结与体会之外的内容。注意引用实验一中未改变部分的流程图、源代码，但要注意子程序有其自身的表示法要求）

进一步修改与增强实验一任务4的**网店商品信息管理程序**的功能，主要调整功能三。

**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=计算推荐度，

4=计算推荐度排名，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）。（思考一下模糊查询如何实现）

找到该商品之后，按照：“商品名称，折扣，销售价，进货总数，已售数量，推荐度”顺序显示该商品的信息。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。[若把接下来的处理步骤写成子程序，则商品名称（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：折扣，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：折扣：9》8 //符号“》”仅作为分隔符，也可以选择其他分隔符号

进货价：25》24

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

当对这些信息都处理完毕后，回到功能三（1）。

（4）计算推荐度

从头到尾依次将每个商品的推荐度计算出来。回到功能三（1）。

（5）计算推荐度排名

对SHOP中的每个商品按照推荐度的大小排名，排名信息可以存放到自行定义的一组结构变量中。回到功能三（1）。

（6）输出全部商品信息

将SHOP中的所有商品信息显示到屏幕上，包括排名。具体的显示格式自行定义（可以按照存放次序显示，也可以按照商品推荐度排名的次序显示，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。

实验报告中只需要描述自己负责的相关功能的设计思想、流程图、源程序。但在设计思想中要描述整体框架（包括整体的模块结构图、功能模块与子程序之间的对应关系等）和分工说明（包括模块的分配，两人协商一致的函数名、变量名等信息）。实验步骤和记录中要描述自己功能的实现与测试以及与同组模块整合后的联调与测试。

**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增加的代码，若有重复的模块（如：因两个人都会使用进制转换程序，导致各自模块中可能都有相同的进制转换子程序），也需要去掉重复的部分。

**建议分组方法：**按照学号（或前后左右相邻座位号）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推荐度从高到低计算名次，也可以考虑按照指定字段（比如已售数量等）排名。相同推荐度排名相同，下一个相邻推荐度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（可以复用网站上相关的代码）。

**上述任务1中，需要解决的问题提示：**

在TD中跟踪到子程序内部有几种方法？在TD中观察子程序调用和返回时堆栈的变化。若执行RET前把栈顶的数值改掉，那么RET执行后程序返回到何处？

注意观察FAR、NEAR类型子程序的RET指令的机器码有何不同？观察FAR类型子程序被调用时堆栈的变化情况。对一个NEAR类型子程序强制使用FAR调用（即CALL FAR PTR 子程序名）会怎样？反之，对一个FAR类型的子程序（子程序可以与主程序在同一个代码段，也可以在不同的代码段）强制使用NEAR调用又会怎样？

通过把一个模块拆成多个模块或反之，体会子程序和模块化程序设计的方法，体会模块调用关系图、子程序功能说明、输入/输出说明在程序设计中的作用。注意 “功能”描述、“模块”描述、“子程序”说明等的区别与联系。

观察不同模块的可合并段合并后变量偏移地址的变化情况。观察不同段在内存里的放置次序。体会模块间段的定义及其对应的装配方法。

在编程中使用不同的子程序参数传递方法来编写子程序。

观察模块间的参数的传递方法，包括公共符号的定义和外部符号的引用，若符号名不一致或类型不一致会有什么现象发生？

通过TD观察宏指令在执行程序中的替换和扩展，解释宏和子程序的调用有何不同。

如何使菜单等显示信息显示得更漂亮一点？

EXTRN说明语句放在.386之前或者之后有什么区别？

EXTRN说明的变量的段与段寄存器的关联关系（ASSUME伪指令所表达的信息）是否能带入到本模块中？如果不能带入，是否可以通过加段前缀的方法来解决？

如何利用宏功能使汇编语言的程序变得更加直观易读？

例：下面是一个利用宏功能直观化后的完整代码段程序，请写出对应的宏定义，并模仿该方式对自己编写的某段程序进行类似的改写。

StartProgram code，data，stack，start

Initial\_ds

GetStringTo BUF

DisplayStringFrom BUF

ExitToDOS

EndProgram code，start

任务2：在C语言程序中调用汇编语言实现的函数

（主要在第二次4个课内学时阶段完成。上机前实验报告应完成该任务中除实验记录与分析、总结与体会之外的内容。若流程图没有实质性改变，只需说明引用任务1的流程图即可；源程序给出完整C语言程序，汇编语言源程序只需给出调整部分的代码）

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。

提示：本任务不分组，但要利用任务1自己整合后的结果。

**上述任务2中，需要解决的问题提示：**

1.在**不同**的C语言开发环境中实现与汇编语言程序的混合编程，其操作方法有可能是**不同**的。请大家选择比较简单或自己熟悉的C语言开发环境并查找相关的资料完成本实验（建议用BC31，其功能与操作方法相对比较简单）。

2.在实验报告中，比较详细的给出你的开发环境及其实现方法。

3.观察C语言编译器中对各种符号的命名规则（指编译器内部可以识别的命名规则，比如，符号名前面是否加下划线“\_”，等），主、子程序之间参数传递的机制，通过堆栈传递参数后堆栈空间回收的方法（要设计一个有多个参数需要传递的C函数）。

4.对混合编程形成的执行程序，用调试工具观察由C语言形成的程序代码与由汇编语言形成的程序代码之间的相互关系，包括段、偏移的值，汇编指令访问C的变量时是如何翻译的，等。

5. C语言与汇编语言的混合编程有两种形式：一种就是C语言的模块与汇编语言的模块组合到一个工程里，编译连接成一个程序；另一种就是在C语言程序中直接嵌入汇编语言指令语句。请尝试在C语言源程序中不合理地嵌入汇编语言的指令语句，达到破坏C语言程序的正确性的目的。比如，在连续的几条C语言语句中间（尤其是在几行计算公式对应的C语句的中间）加入一条修改AX寄存器（或DS等其他寄存器）的汇编指令语句，若AX的内容在此处本不该被修改，则可观察到破坏了C语言程序正确性的效果（该项实验表明：在C语言程序中，若不考虑上下语句翻译成怎样的机器码而随意嵌入汇编指令语句时，有可能存在出错的风险）。

6.观察C编译器的优化策略对代码的影响。通过实际观察与分析，记录本实验中汇编语言程序的效率会优于C语言程序的实例（至少给出一处的观察结果）。

7.通过调试混合编程的程序，体会与纯粹汇编语言编写的程序的调试过程的差异。

8.通过本次实验，希望大家明白：不同的编程语言是可以协同解决一个问题的，而且可以利用不同语言的特点来更好地解决问题；利用汇编语言的知识，能够更好地理解高级语言的内部处理原理与策略，为编写更好的C语言程序、用好C编译器提供支持。

**参考文献：**

[1]许向阳，《80X86汇编语言程序设计上机指南》“第十一章 汇编语言程序与C程序的连接”。

资料说明：

第十一章 汇编语言程序与C程序的连接.doc（从教学网站的实验指导中下载）

C 程序调用 汇编语言函数 操作说明.doc（从教学网站的实验指导中下载）。

操作步骤与多模块编程时建立一个项目（或工程，PROJECT）相同，只是选择模块时除了C语言的模块，还有汇编语言的模块（源程序或OBJ文件）。

**注意**：C编译器对C文件扩展名为\*.C和\*.CPP的处理策略有差异，因此，大家在参考网站上资料时，一定要注意所给出的例子的文件扩展名是什么。

BC3.1 或 TC3.0下混合编程的方法与上述“【1】【2】”中介绍的类似，但还需要注意一些区别。首先是C编译器需要外挂一个汇编程序以帮助处理汇编语言的语句，一般会使用TASM，它与MASM基本一致，语法规则只有少许差异。**教学网站中可以下载一个带DOS虚拟机的完整BC31软件，包括C编译器，TASM，TD等**。其次，命名规则有点不同。比如，若想在C语言程序中直接嵌入汇编语言语句，该软件就不需要在ASM关键词前加下划线。另外，若想C编译器对嵌入的汇编语言语句不报错，还需要在C语言程序开始之处加上说明语句#pragma inline，例如：

**#include <stdio.h>**

**#pragma inline**

**int main()**

**{ int count;**

**asm mov count,5 //嵌入的汇编语言语句**

**printf("count=%d",count);**

**return 0;**

**}**

## 实验过程

### 3.1任务1

本次实验我和胡晨风一组，我负责设计主控程序及功能1、2，胡晨风负责设计功能3、4、5。

#### 3.1.1设计思想

在实验一任务4的基础上，我做了如下修改： 将计算推荐度的功能4提前，在输入账号密码前直接计算好全部商品的推荐度，这样做的目的是:在查询商品时要显示推荐度，为了防止输出一堆乱码，首先计算好商品的推荐度。然后在输入账号密码后，根据是否登陆显示不同的菜单，然后根据选择的功能，执行相应的函数，程序总体流程如下：

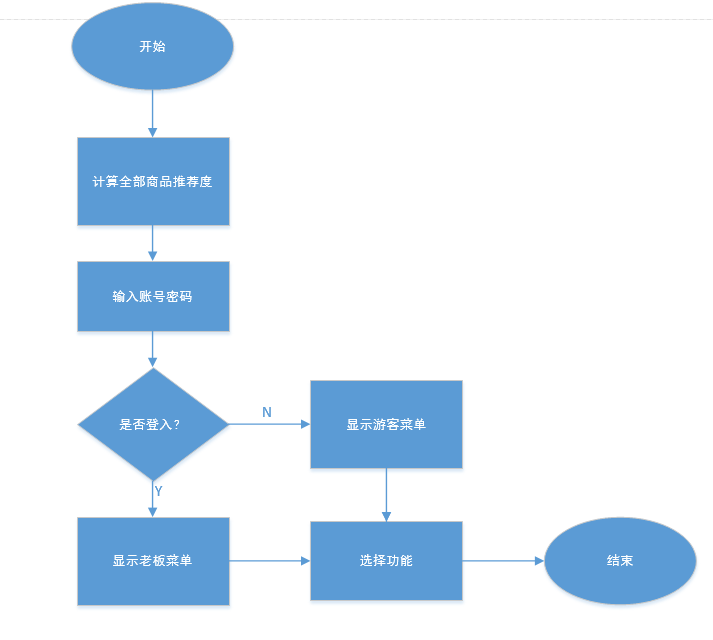


图1.程序流程图

对于功能一：查询商品，我的设计思想如下：首先用户输出商品名，调用一个查找函数来查找商品，入口参数为商品的偏移地址和储存用户输入的商品的变量的偏移地址，分别储存在SI,DI中，出口参数为SI，为找到商品的偏移地址。找到之后，根据SI，分别显示商品的若干信息，需要调用子程序F2T10，将商品若干信息以十进制的形式输出，子程序流程图如下：

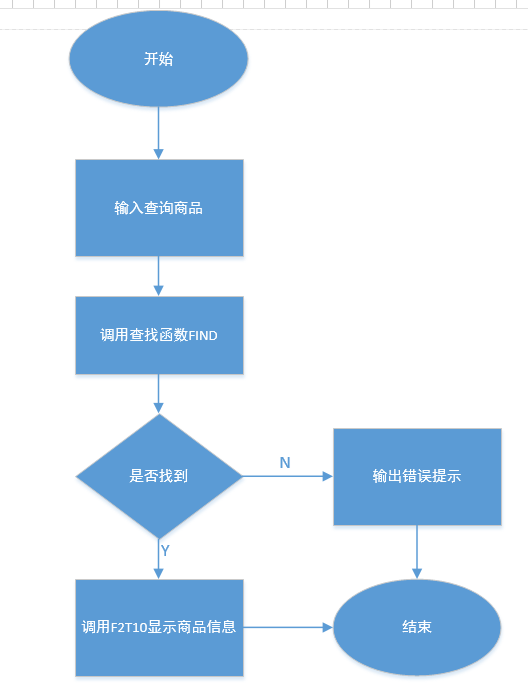


图2.功能一流程图

对于功能二：修改商品信息，我的设计思想是：首先用户输入商品名，然后调用函数FIND找到该商品，之后调用F2T10显示折扣度和符号‘>’,再调用F10T2将用户输入的修改信息转换成二进制存储到对应的内存地址，F10T2可以判断用户输入是否合法，如果输入错误，则会跳转到折扣度这一步重新输入，输入正确则跳转到下一步，修改进货价，依次类推，依次修改销售价和进货总量，子程序流程图如下：

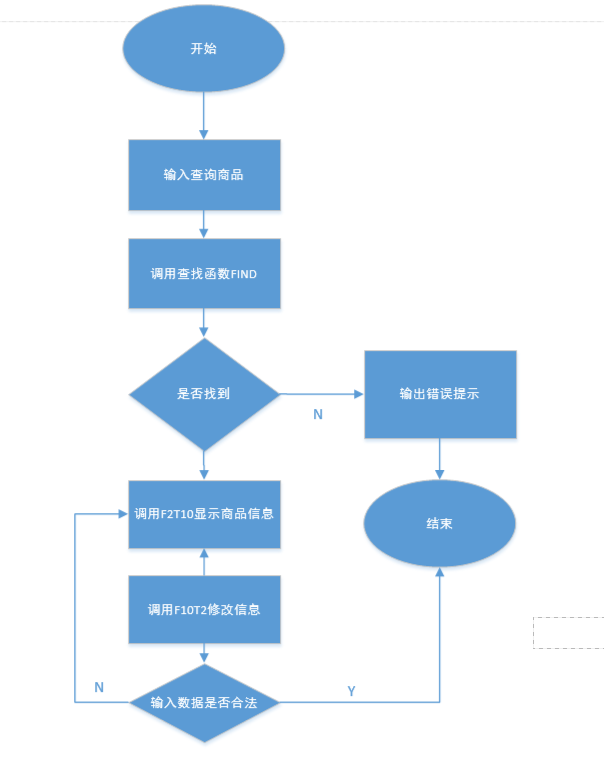


图3.功能二流程图

设计好后，链接子程序，首先将两个程序的数据段PUBLIC为一个数据段‘DATA’，并将F2T10 PUBLIC，将功能三四五声明EXTERN进行调用，在功能三四五里做相应修改，编译后，将两文件LINK起来，就得到了EXE文件。

#### 3.1.2 源程序

**Re3:**

EXTERN FUNC3 :FAR

EXTERN FUNC4 :FAR

EXTERN FUNC5 :FAR

PUBLIC F2T10

PUBLIC RDAIX

.386

DATA SEGMENT USE16

LEAP DW 80 DUP(0)

BNAME DB 'LIANGYIFEI' ;老板姓名

BPASS DB '991026' ;密码

SNAME DB 'FEISHOP',0DH,0AH,'$' ;网店名称，用0结束

BUF DB 12 DUP(?)

N EQU 30

POINT DB 'print username and password:',0DH,0AH,'$'

TIP DB 'MISMATCH!',0DH,0AH,'$'

TIP2 DB 'NOT FOUND!TRY AGAIN!',0DH,0AH,'$'

POINT2 DB 'Tell me what you want:',0DH,0AH,'$'

POINT3 DB 'Tell me what you choose:',0DH,0AH,'$'

POINT4 DB 'ERROR!',0DH,0AH,'$'

TAG DB 0DH,0AH,'$'

SIGN DB ?

DAT DW 0AH

TIP3 DB 'Welcome BOSS!According the following tips to choose:',0DH,0AH,'1.Query product information',0DH,0AH,'2.Modify product information',0DH,0AH,'3.Calculate the recommendation',0DH,0AH,'4.Recommendation ranking',0DH,0AH,'5.All product information',0DH,0AH,'6.Exit',0DH,0AH,'$'

TIP4 DB 'Welcome to FEISHOP!According the following tips to choose:',0DH,0AH,'1.Query product information',0DH,0AH,'6.Exit',0DH,0AH,'$'

IN\_NAME DB 11

DB ?

DB 11 DUP(0)

IN\_PWD DB 7

DB ?

DB 7 DUP(0)

IN\_GA DB 11

DB ?

DB 11 DUP(0)

AUTH DB ?

I DW 0

GA1 DB 'PEN',7 DUP(0) ,10 ;商品名称及折扣

DW 35,56,70,25,? ;推荐度还未计算

GA2 DB 'BOOK', 6 DUP(0) ,9 ;商品名称及折扣

DW 12,30,25,5,? ;推荐度还未计算

GAN DB N-2 DUP('Temp-Value',8,15,0,20,0,30,0,2,0,?,?)

NUM DB 10

DB ?

DB 10 DUP(0)

DATA ENDS

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT USE16

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

START:MOV AX,DATA

MOV DS,AX

LEA DX,SNAME

MOV AH,9

INT 21H

MOV SI,OFFSET GA1;计算全部商品的推荐度

CALL CACUL

FUN1: LEA DX,POINT

MOV AH,9

INT 21H

LEA DX,IN\_NAME

MOV AH,10

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

MOV CL,IN\_NAME[1]

CMP CL,0

JE T1

CMP CL,1

JE ISQ

JMP IPW

ISQ: MOV CL,IN\_NAME[2]

CMP CL,'q'

JE QUIT

IPW: LEA DX,IN\_PWD

MOV AH,10

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

FUN2\_1: MOV CL,IN\_NAME[1]

CMP CL,10

JNE T2

MOV DI,10

JMP LOPA1

FUN2\_2: MOV CL,IN\_PWD[1]

CMP CL,6

JNE T2

MOV DI,6

JMP LOPA2

LOPA1:MOV CL,BNAME[DI-1]

CMP CL,IN\_NAME[DI+1]

JNE T2

DEC DI

CMP DI,0

JNE LOPA1

JMP FUN2\_2

LOPA2:MOV CL,BPASS[DI-1]

CMP CL,IN\_PWD[DI+1]

JNE T2

DEC DI

CMP DI,0

JNE LOPA2

MOV AUTH,1

FUN3\_1:CMP AUTH,1

JZ PRIN

LEA DX,TIP4

MOV AH,9

INT 21H

JMP FUN3\_2

PRIN:LEA DX,TIP3

MOV AH,9

INT 21H

FUN3\_2:LEA DX,POINT3

MOV AH,9

INT 21H

MOV AH,1

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

CMP AL,'1'

JZ F1C

CMP AL,'2'

JZ F2C

CMP AL,'3'

JZ F3C

CMP AL,'4'

JZ F4C

CMP AL,'5'

JZ F5C

CMP AL,'6'

JZ F6C

WRO: LEA DX,POINT4

MOV AH,9

INT 21H

JMP FUN3\_2

F1C: CALL FUNC1

CALL CLEAR

JMP FUN3\_1

F2C: CMP AUTH,0

JE WRO

CALL FUNC2

CALL CLEAR

JMP FUN3\_1

F3C: CMP AUTH,0

JE WRO

MOV SI,OFFSET GA1

CALL FUNC3

LEA DX,TAG

MOV AH,9

INT 21H

JMP FUN3\_1

F4C: CMP AUTH,0

JE WRO

MOV SI,OFFSET GA1

CALL FUNC4

JMP FUN3\_1

F5C: CMP AUTH,0

JE WRO

MOV SI,OFFSET GA1

CALL FUNC5

LEA DX,TAG

MOV AH,9

INT 21H

JMP FUN3\_1

F6C: CALL CLEAR

JMP FUN1

FUNC1 PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

PUSH SI

PUSH DI

LEA DX,POINT2

MOV AH,9

INT 21H

LEA DX,IN\_GA

MOV AH,10

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

MOV CL,IN\_GA[1]

CMP CL,0

JE F1\_3

MOV DI,OFFSET IN\_GA

MOV SI,OFFSET GA1

CALL FIND

F1\_2: MOV CL,IN\_GA[1]

MOV BX,0

PRINTNAME: ADD CL,-1

JS PRINTVALUE

MOV AH,02H

MOV DL,[SI][BX]

INT 21H

ADD BX,1

JMP PRINTNAME

PRINTVALUE: LEA DX,' '

MOV AH,2

INT 21H

MOV AX,0

MOV AL,[SI+10]

CALL F2T10

MOV AX,WORD PTR[SI+11]

CALL F2T10

MOV AX,WORD PTR[SI+13]

CALL F2T10

MOV AX,WORD PTR[SI+15]

CALL F2T10

MOV AX,WORD PTR[SI+17]

CALL F2T10

MOV AX,WORD PTR[SI+19]

CMP AX,100

JA ALEVEL

CMP AX,50

JA BLEVEL

CMP AX,10

JA CLEVEL

JMP FLEVEL

ALEVEL: LEA DX,'A'

MOV AH,2

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

JMP F1\_3

BLEVEL: LEA DX,'B'

MOV AH,2

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

JMP F1\_3

CLEVEL: LEA DX,'C'

MOV AH,2

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

JMP F1\_3

FLEVEL: LEA DX,'F'

MOV AH,2

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

F1\_3: POP DI

POP SI

POP DX

POP CX

POP BX

POP AX

RET

FUNC1 ENDP

FUNC2 PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

PUSH SI

PUSH DI

LEA DX,POINT2

MOV AH,9

INT 21H

LEA DX,IN\_GA

MOV AH,10

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

MOV CL,IN\_GA[1]

CMP CL,0

JE F2\_6

MOV DI,OFFSET IN\_GA

MOV SI,OFFSET GA1

CALL FIND

F2\_2: MOV AX,0

MOV AL,[SI+10]

CALL F2T10

LEA DX,'>'

MOV AH,2

INT 21H

LEA DX,NUM

MOV AH,10

INT 21H

MOV DI,OFFSET NUM

ADD DI,2

MOV CL,NUM[1]

CALL F10T2

CMP DI,-1

JE F2\_2

MOV [SI+10],AL

LEA DX,TAG

MOV AH,9

INT 21H

F2\_3: MOV AX,[SI+11]

CALL F2T10

LEA DX,'>'

MOV AH,2

INT 21H

LEA DX,NUM

MOV AH,10

INT 21H

MOV DI,OFFSET NUM

ADD DI,2

MOV CL,NUM[1]

CALL F10T2

CMP DI,-1

JE F2\_3

MOV [SI+11],WORD PTR AX

LEA DX,TAG

MOV AH,9

INT 21H

F2\_4: MOV AX,[SI+13]

CALL F2T10

LEA DX,'>'

MOV AH,2

INT 21H

LEA DX,NUM

MOV AH,10

INT 21H

MOV DI,OFFSET NUM

ADD DI,2

MOV CL,NUM[1]

CALL F10T2

CMP DI,-1

JE F2\_4

MOV [SI+13],WORD PTR AX

LEA DX,TAG

MOV AH,9

INT 21H

F2\_5: MOV AX,[SI+15]

CALL F2T10

LEA DX,'>'

MOV AH,2

INT 21H

LEA DX,NUM

MOV AH,10

INT 21H

MOV DI,OFFSET NUM

ADD DI,2

MOV CL,NUM[1]

CALL F10T2

CMP DI,-1

JE F2\_5

MOV [SI+15],AL

LEA DX,TAG

MOV AH,9

INT 21H

F2\_6: POP DI

POP SI

POP DX

POP CX

POP BX

POP AX

RET

FUNC2 ENDP

T1: MOV AUTH,0

JMP FUN3\_1

T2: LEA DX,TIP

MOV AH,9

INT 21H

JMP FUN1

T3: LEA DX,TIP2

MOV AH,9

INT 21H

JMP FUN3\_1

FIND PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

MOV CL,IN\_GA[1]

MOVZX AX,CL

ADD DI,AX

ADD DI,2

MOV AL,0

MOV [DI],AL

SUB SI,21

MOV DI,OFFSET IN\_GA

MOV I,N

LOPA3:ADD I,-1

JS T3

ADD SI,21

MOV CL,9

LOPA4:MOV BL,CL

MOV AL,2[DI+BX]

CMP AL,0[SI][BX]

JNE LOPA3

ADD CL,-1

JS FEND

JMP LOPA4

FEND: POP DX

POP CX

POP BX

POP AX

RET

FIND ENDP

CLEAR PROC

MOV AX,0

MOV BX,0

MOV CX,0

MOV DX,0

MOV SI,0

MOV DI,0

MOV BX,11

MOV SI, OFFSET IN\_GA

CLEAR1: MOV BYTE PTR[SI+BX],0

ADD BX,-1

CMP BX,0

JNE CLEAR1

MOV SI,0

RET

CLEAR ENDP

CACUL PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

ADD SI,-21

MOV I,N

LOPA5:ADD I,-1

JS CAUEND

ADD SI,21

MOV AX,WORD PTR [SI+11]

MOV BX,0080H

MUL BX

MOV BX,WORD PTR [SI+13]

DIV BX

MOV BX,000AH

MUL BX

MOV BL,[SI+10]

MOV BH,00H

DIV BX

MOV CX,AX

MOV AX,WORD PTR [SI+17]

MOV BX,0040H

MUL BX

MOV BX,WORD PTR [SI+15]

DIV BX

ADD AX,CX

MOV WORD PTR [SI+19],AX

MOV AX,0

MOV BX,0

MOV CX,0

JMP LOPA5

CAUEND: POP DX

POP CX

POP BX

POP AX

RET

CACUL ENDP

QUIT: MOV AH,4CH

INT 21H

F10T2 PROC FAR

PUSH DX

PUSH BX

MOV AX,0

MOV SIGN,0

MOV BL,[DI]

CMP BL,'+'

JZ A

CMP BL,'-'

JNE NEXT2

MOV SIGN,1

A:

DEC CX

JZ ERR

NEXT1:

INC DI

MOV BL,[DI]

NEXT2:

CMP BL,'0'

JB ERR

CMP BL,'9'

JA ERR

SUB BL,30H

MOV BH,0

MUL DAT

JO ERR

ADD AX,BX

JC ERR

DEC CX

JNZ NEXT1

CMP SIGN,1

JNE QQ

NEG AX

QQ:

POP BX

POP DX

RET

ERR:

MOV DI,-1

JMP QQ

F10T2 ENDP

F2T10 PROC FAR

PUSH EBX

PUSH SI

LEA SI,BUF

CMP DX,32

JE B

MOVSX EAX,AX

B: OR EAX,EAX

JNS PLUS

NEG EAX

MOV BYTE PTR [SI],'-'

INC SI

PLUS: MOV EBX,10

CALL RDAIX

MOV BYTE PTR [SI],'$'

LEA DX,BUF

MOV AH,9

INT 21H

LEA DX,' '

MOV AH,2

INT 21H

POP SI

POP EBX

RET

F2T10 ENDP

RDAIX PROC FAR

PUSH CX

PUSH EDX

XOR CX,CX

LOP1:XOR EDX,EDX

DIV EBX

PUSH DX

INC CX

OR EAX,EAX

JNZ LOP1

LOP2:POP AX

CMP AL,10

JB L1

ADD AL,7

L1: ADD AL,30H

MOV [SI],AL

INC SI

LOOP LOP2

POP EDX

POP CX

RET

RDAIX ENDP

CODE ENDS

END START

**Re32：**

PUBLIC FUNC3,FUNC4,FUNC5

EXTERN F2T10:FAR

EXTERN RDAIX:FAR

.386

STACK1 SEGMENT USE16 STACK

DB 200 DUP(0)

STACK1 ENDS

DATA SEGMENT USE16 PARA PUBLIC 'DATA'

BUF DB 12 DUP(?)

N EQU 30

M EQU 60

PAI DW N DUP(0)

PAIZ DW N DUP(0)

DATA ENDS

CODE1 SEGMENT USE16

ASSUME CS:CODE1, DS:DATA, SS:STACK1

FUNC3 PROC FAR

PUSH DI

PUSH EAX

PUSH EBX

PUSH ECX

PUSH EDX

MOV DI,N

TJD:MOVZX EAX,WORD PTR 11[SI]

SAL EAX,7

MOV EDX,0

MOVZX EBX,WORD PTR 13[SI]

DIV EBX

MOV EBX,0000000AH

MUL EBX

MOVZX EBX,BYTE PTR 10[SI]

MOV EDX,0

DIV EBX

MOV ECX,EAX

MOVZX EAX,WORD PTR 17[SI]

SAL EAX,6

MOV EDX,0

MOVZX EBX,WORD PTR 15[SI]

DIV EBX

ADD EAX,ECX

MOV 19[SI],AX

ADD SI,21

DEC DI

CMP DI,0

JNZ TJD

POP EDX

POP ECX

POP EBX

POP EAX

POP DI

RET

FUNC3 ENDP

FUNC4 PROC FAR

PUSH DI

PUSH AX

PUSH EBX

PUSH ECX

MOV DI,0

LOOP4: MOV AX,WORD PTR 19[SI]

MOV PAI[DI],AX

ADD SI,21

ADD DI,2

CMP DI,M

JNZ LOOP4

MOV DI,1

LOOP1: MOV EBX,1

MOV ECX,0

MOV AX,PAI

LOOP2: CMP AX,PAI[EBX\*2]

JNB NEXT2

NEXT1: MOV AX,PAI[EBX\*2]

MOV ECX,EBX

NEXT2: ADD EBX,1

CMP EBX,N

JNZ LOOP2

MOV EBX,ECX

MOV PAIZ[EBX\*2],DI

MOV PAI[EBX\*2],0

INC DI

CMP DI,N+1

JNZ LOOP1

POP ECX

POP EBX

POP AX

POP DI

RET

FUNC4 ENDP

;入口参数

; SI 第一个商品的地址

;出口参数 显示所有商品的信息

FUNC5 PROC FAR

PUSH DI

PUSH DX

PUSH BX

PUSH AX

MOV DI,0

LOOP5:

MOV DX,0AH

MOV AH,2

INT 21H

MOV DX,0DH

MOV AH,2

INT 21H

MOV BX,0

LOOP3:

MOV AL,BYTE PTR [BX+SI]

CMP AL,0

JZ K

MOV DL,AL

MOV AH,2

INT 21H

INC BX

CMP BX,10

JNZ LOOP3

K: MOV DX,' '

MOV AH,2

INT 21H

MOV AL,BYTE PTR 10[SI]

MOV AH,0

MOV DX,16

CALL F2T10

MOV DL,' '

MOV AH,2

INT 21H

MOV AX,WORD PTR 11[SI]

MOV DX,16

CALL F2T10

MOV DL,' '

MOV AH,2

INT 21H

MOV AX,WORD PTR 13[SI]

MOV DX,16

CALL F2T10

MOV DL,' '

MOV AH,2

INT 21H

MOV AX,WORD PTR 15[SI]

MOV DX,16

CALL F2T10

MOV DL,' '

MOV AH,2

INT 21H

MOV AX,WORD PTR 17[SI]

MOV DX,16

CALL F2T10

MOV DL,' '

MOV AH,2

INT 21H

MOV AX,WORD PTR 19[SI]

MOV DX,16

CALL F2T10

MOV DL,' '

MOV AH,2

INT 21H

MOV AX,PAIZ[DI]

MOV DX,16

CALL F2T10

ADD DI,2

ADD SI,21

CMP DI,60

JNZ LOOP5

POP AX

POP BX

POP DX

POP DI

RET

FUNC5 ENDP

CODE1 ENDS

end

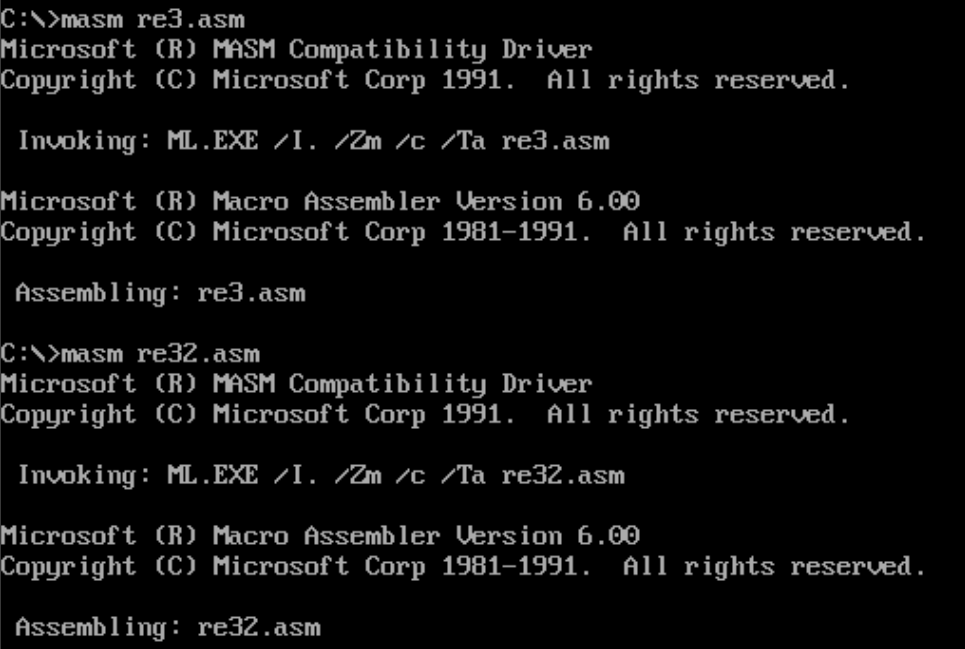
#### 3.1.3 实验步骤

1. 准备环境
2. 汇编，链接，运行
3. 测试未登录状态的界面与功能以及错误输入
4. 测试登陆状态的界面与功能以及错误输入

#### 3.1.4 实验记录与分析

1.编译、链接、运行

步骤如图所示：



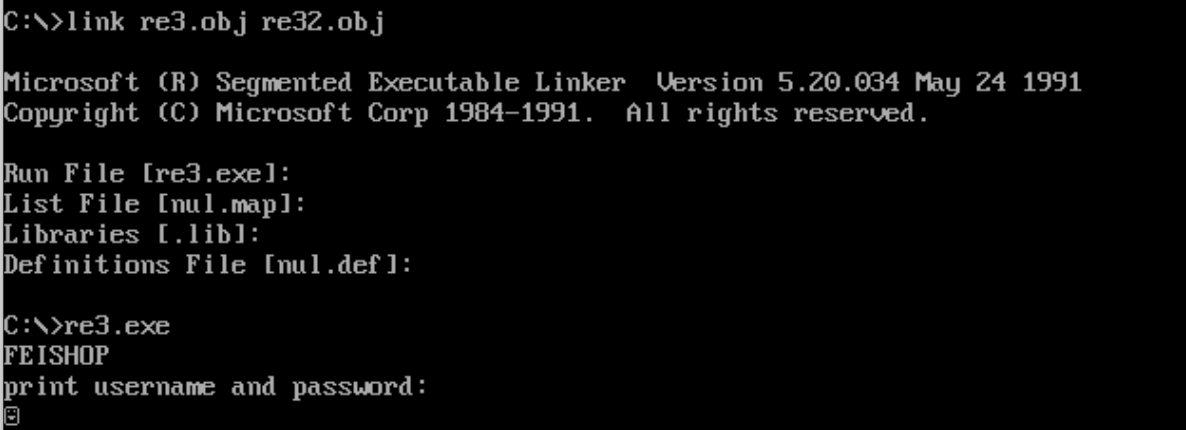


图4.编译、链接、运行图

2.测试未登录状态

界面如图所示：

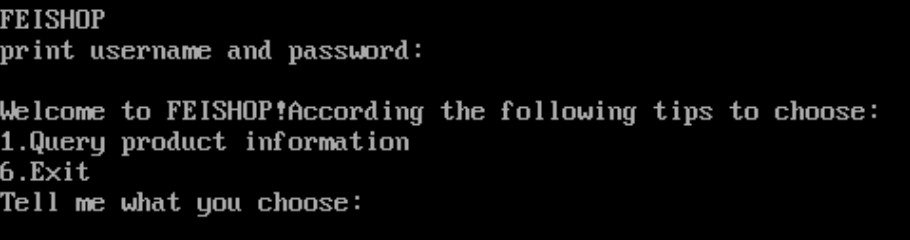


图5.编译、链接、运行图

测试功能一，查询商品PEN：

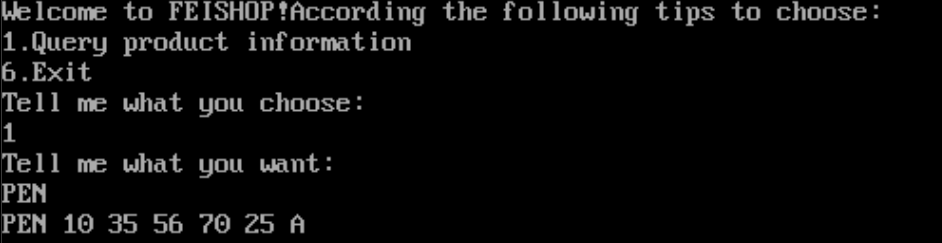


图6.测试图

输入2，看是否有错误提示：

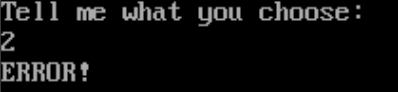


图7.测试图

3.测试登陆状态

登陆界面图：

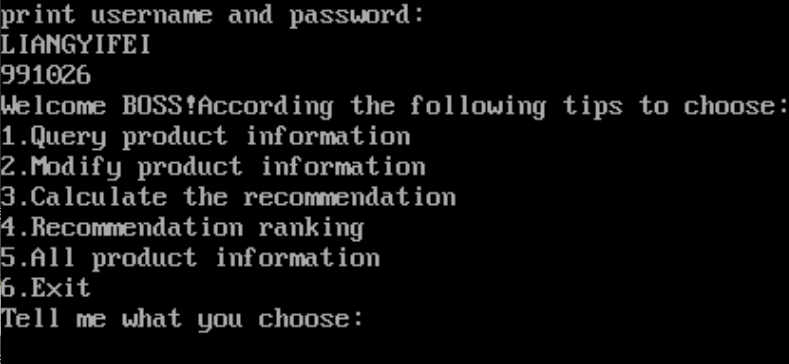
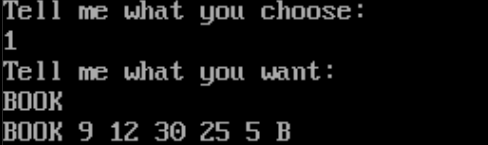
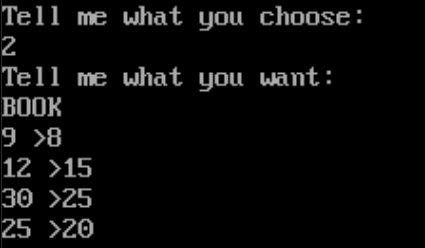
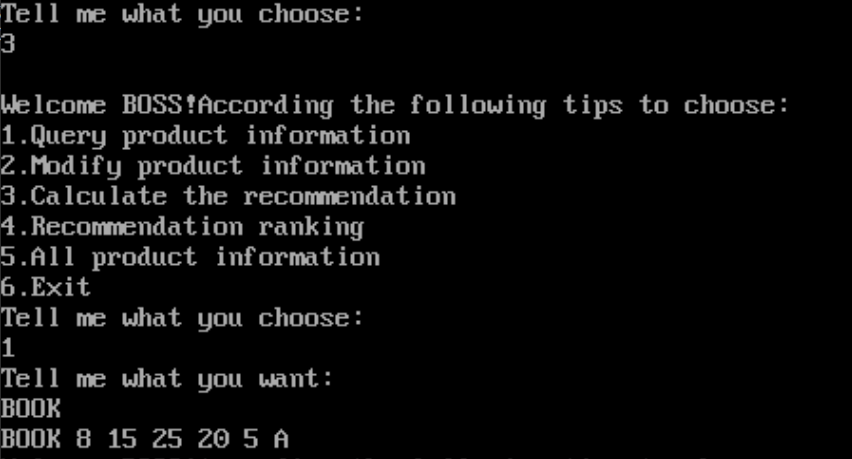


图8.登陆界面图

测试功能一，查询商品BOOK，测试功能二,修改商品BOOK信息，然后测试功能三计算推荐度，再重新查询商品BOOK。然后测试功能4，计算排名，再用功能5，显示商品排名信息：







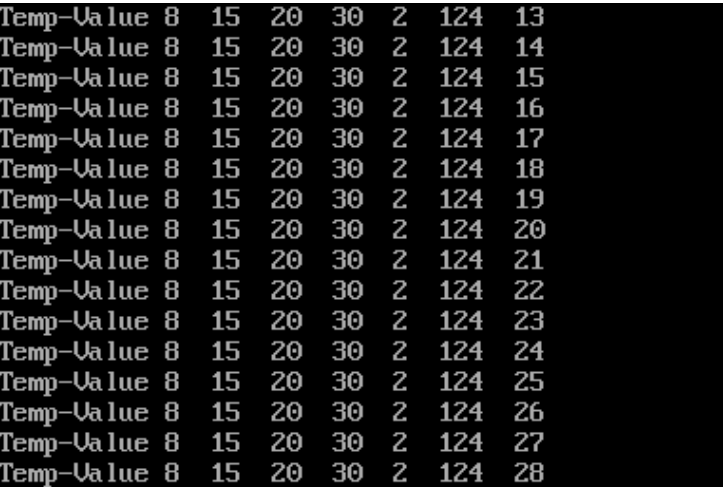
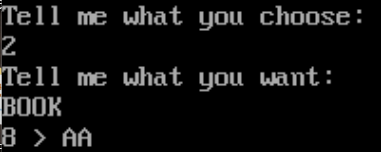


图9.测试

对于功能二，测试错误输入：



此时光标会回到‘>’后，可以重新进行输入覆盖掉错误输入

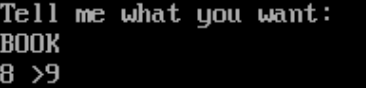


图10.测试

### 3.2 任务1总结与体会

本次实验的主要问题均出在将两个程序链接到一起上，在实验过程中出现了如下问题：1.在对子程序进行封装时，我先将各寄存器入栈，但是在结束出栈时没有按入栈顺序的反顺序进行出栈，导致下一个功能会出现莫名其妙的错误，在修改好顺序后，这个问题得到了解决。

2.在对两个ASM文件进行链接时，部分函数和变量忘记PUBLIC和EXTERN，导致无法运行，在仔细检查后得到解决。

3.两个ASM文件的内存段没有整合成一个内存段，导致在调用函数修改偏移地址里的内容时，同时修改了两个内存段的内容，输出了一些乱码，在两个内存段分别加上PARA PUBLIC ‘DATA’，于是问题得到解决。

4.一开始在功能3、4、5上加了一句NAME WAN2，虽然未出现报错，但是无法调用，于是在TD调试工具里按F7进入子程序，发现进不去，然后去掉了NAME WAN2，问题得到了解决。

通过这次实验，我掌握了汇编模块化编程的方法，这使我的代码量减少了，程序简明易读了，再也不像以前编程序那样JMP来JMP去了。

### 3.3 任务2

#### 3.3.1 设计思想

首先将主控程序改为C语言实现，主控程序主要包括登陆和菜单选择界面。剩下的用汇编实现，本次实验在visual studio 2017中进行，在开始实验时，首先按汇编网站上的文件配备好了实验环境。

#### 3.3.2 源程序

**Re3.asm:**

.386

.model flat, c

option casemap:none

extrn printf: near

extrn scanf: near

extrn strncmp: near

public FUNC1

public FUNC2

public FUNC3

public FUNC4

public FUNC5

includelib user32.lib

includelib kernel32.lib

goodStruct struct

good\_name DB 10 dup(0)

good\_len DB 0

good\_sale DW 10

good\_in\_price DW ?

good\_out\_price DW ?

good\_all\_count DW ?

good\_sale\_count DW ?

good\_recommand DW ?

good\_sort DW 1

goodStruct ends

.DATA

N EQU 30 ;number

numOfBag EQU 1000

ga1 goodStruct <'PEN', 3, 10, 35, 56, 70, 25, ?>

ga2 goodStruct <'BOOK', 4, 9, 12, 30, 25, 5, ?>

goodStruct N-2 dup(<'EARASER', 3, 7, 35, 56, 70, 25, ?>)

checkname DB 'Tell me what you want:','0'

in\_name db 15 dup(0)

notFound DB 'NOT FOUND!TRY AGAIN!', 0ah, 0dh, '0'

ui\_query\_failed DB 'NOT FOUND!TRY AGAIN!', '0'

outenter DB 0ah, 0dh, 0

errorInputMsg DB ' Error!', 0ah, 0dh, 0

io\_1 db 'NAME: ',0

io\_2 db 'SALE: ',0

io\_3 db 'IN\_PRICE: ',0

io\_4 db 'OUT\_PRICE: ',0

io\_5 db 'ALL\_ACCOUNT: ',0

io\_6 db 'SALE\_ACCOUNT: ',0

io\_7 db 'COMMOND: ',0

io\_8 db 'good\_sort is: ',0

BUF1 db 10, ?, 10 dup(0)

bufF2T10 db 12 dup(0)

DAT DW 0AH

SIGN DB ?

fmt db '%10s',0

formatout db "%hd", 10, 0 ;

formatin db "%hd",0

putc db ">>",0

.STACK 200

.CODE

gets macro of:req

push ebx

lea ebx, of

push ebx

push dword ptr offset formatin

call scanf

add esp, 8

pop ebx

endm

puts macro of:req

push dword ptr offset of

call printf

add esp, 4

endm

puts\_no macro of:req

push dword ptr of

call printf

add esp, 4

endm

putchar macro

puts putc

endm

puts\_nu macro of:req

push bx

mov bx, word ptr of

push bx

push dword ptr offset formatout

call printf

add esp, 6

pop bx

endm

CRLF macro

puts outenter

endm

scmp macro one:req, anthor:req

push dword ptr 8

push dword ptr one

push dword ptr offset anthor

call strncmp

add esp, 12

endm

; esi good addr

outone proc near

push eax

push ebx

push dx

mov dx, 16

L6:

puts io\_1

lea ebx, [esi].good\_name

puts\_no ebx

CRLF

puts io\_2

puts\_nu [esi].good\_sale

puts io\_3

puts\_nu [esi].good\_in\_price

puts io\_4

puts\_nu [esi].good\_out\_price

puts io\_5

puts\_nu [esi].good\_all\_count

puts io\_6

puts\_nu [esi].good\_sale\_count

puts io\_7

puts\_nu [esi].good\_recommand

puts io\_8

puts\_nu [esi].good\_sort

pop dx

pop ebx

pop eax

ret

outone endp

FUNC1 PROC near

que:

puts checkname

push dword ptr offset in\_name

push dword ptr offset fmt

call scanf

add esp, 8

lea esi, ga1

mov bp, N

query\_loop:

lea ebx, [esi].good\_name

scmp ebx, in\_name

cmp eax, 0

jz query\_find

add esi, sizeof goodStruct

dec bp

jnz query\_loop

query\_not\_find:

puts ui\_query\_failed

jmp que

query\_find:

call outone

ret

FUNC1 ENDP

;di good addr

FUNC2 proc

cha\_que:

puts checkname

push dword ptr offset in\_name

push dword ptr offset fmt

call scanf

add esp, 8

lea esi, ga1

mov bp, N

cha\_query\_loop:

lea ebx, [esi].good\_name

scmp ebx, in\_name

cmp eax, 0

jz cha\_query\_find

add esi, sizeof goodStruct

dec bp

jnz cha\_query\_loop

cha\_query\_not\_find:

puts ui\_query\_failed

jmp cha\_que

cha\_query\_find:

puts io\_2

puts\_nu [esi].good\_sale

putchar

gets [esi].good\_sale

puts io\_3

puts\_nu [esi].good\_in\_price

putchar

gets [esi].good\_in\_price

puts io\_4

puts\_nu [esi].good\_out\_price

putchar

gets [esi].good\_out\_price

puts io\_5

puts\_nu [esi].good\_all\_count

putchar

gets [esi].good\_all\_count

ret

FUNC2 endp

FUNC3 proc

mov bp, N ;bp is iterator

mov esi, offset ga1 ;si is commodity

calc\_pro:

mov ax, [esi].good\_in\_price

mov bx, 1280

xor dx,dx

MUL bx

mov bx, [esi].good\_out\_price

mov cx, [esi].good\_sale

imul bx, cx

div bx

mov cx, ax

mov ax, [esi].good\_sale\_count

sal ax, 6

mov bx, [esi].good\_all\_count

xor dx, dx

div bx

add ax, cx

mov word ptr[esi].good\_recommand, ax

add esi, sizeof goodStruct

dec bp

jnz calc\_pro

ret

FUNC3 endp

;ax good sum, si good addr

FUNC4 proc

push ebx

push cx

push dx

mov ax, N

mov esi, offset ga1 ;si is commodity

jmp L3

L4:

dec ax

add esi, sizeof goodStruct

cmp ax, 1

je exit

L3:

mov dx, [esi].good\_recommand

mov ebx, esi

mov cx, ax

L5:

add ebx, sizeof goodStruct

dec cx

cmp dx, [ebx].good\_recommand

ja L1

jb L2

next:

cmp cx, 1

je L4

jmp L5

L1:

inc [ebx].good\_sort

jmp next

L2:

inc [esi].good\_sort

jmp next

exit:

pop dx

pop cx

pop ebx

ret

FUNC4 endp

;;ax good sum, si good addr

FUNC5 proc

mov ax, N

mov esi, offset ga1 ;si is commodity

L7:

call outone

add esi, sizeof goodStruct

dec ax

jne L7

ret

FUNC5 endp

END

**Re3.cpp:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include "stdio.h"

#include <string.h>

extern "C" long \_cdecl FUNC1();

extern "C" long \_cdecl FUNC2();

extern "C" long \_cdecl FUNC3();

extern "C" long \_cdecl FUNC4();

extern "C" long \_cdecl FUNC5();

int op=1;

int main(int argc, char \*argv[]) {

char password[10];

char username[11];

int vb;

printf("Are you Boos or Vistor?(1/0)\n");

scanf("%d", &vb);

if (vb == 0) {

for(;;){

system("cls"); printf("\n\n");

printf(" Hello Vistor, Welcome to FEISHOP! \n");

printf("-------------------------------------------------\n");

printf(" 1.Query product information\n");

printf(" 6. Exit\n");

printf("-------------------------------------------------\n");

printf(" Tell me what you choose[0~6]:");

scanf("%d", &op);

if (op == 1) {

FUNC1();

getchar(); getchar();

}

if(op==6)

break;

}

return 0;

}

else if (vb == 1) {

L1:

printf("Print username:");

scanf("%s", username);

if (strcmp("LIANGYIFEI", username) == 0) {

L2:

printf("Print password:");

scanf("%s", password);

if (strcmp("991026", password) == 0) {

for (;;) {

system("cls"); printf("\n\n");

printf(" Hello Boss, Welcome to FEISHOP! \n");

printf("-------------------------------------------------\n");

printf(" 1.Query product information\n");

printf(" 2.Modify product information\n");

printf(" 3.Calculate the recommendation \n");

printf(" 4.Recommendation ranking\n");

printf(" 5.All product information\n");

printf(" 6.Exit\n");

printf("-------------------------------------------------\n");

printf(" Tell me what you choose[0~6]:");

scanf("%d", &op);

if (op == 1) {

FUNC1();

getchar(); getchar();

}

if (op == 2) {

FUNC2();

getchar(); getchar();

}

if (op == 3) {

FUNC3();

getchar(); getchar();

}

if (op == 4) {

FUNC4();

getchar(); getchar();

}

if (op == 5) {

FUNC5();

getchar(); getchar();

}

if (op == 6)

break;

}

}

else {

printf("Mismatch,Try again\n");

goto L2;

}

}

else {

printf("Mismatch,Try again\n");

goto L1;

}

}

return 0;

}

#### 3.3.2 实验步骤

1.准备环境

2.汇编，链接，运行

3.测试未登录状态的界面与功能

4.测试登陆状态的界面与功能

#### 3.3.3 实验记录与分析

1.准备环境

首先安装VERSAT进F盘

进入命令行：开始菜单 => 运行 => cmd => 回车；

在c>状态下输入：F: 回车；varset 回车；

输入ml.exe /I. /Zm /c /Ta re3.asm 回车；（生成re3.obj文件）

在visual studio 2017里创建一个空项目，加入主程序和obj文件，如图所示：

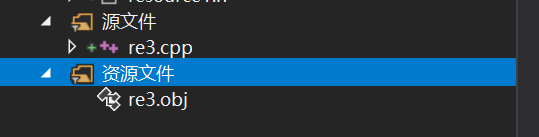
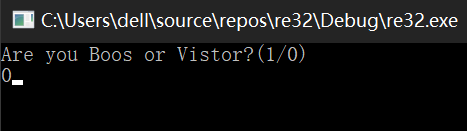
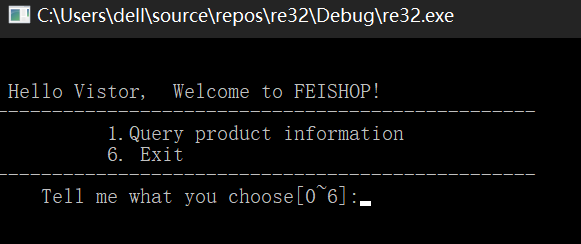


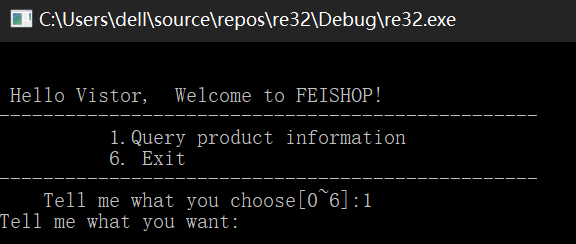
图11.准备

再进行编译和运行。

2.测试未登录状态的界面与功能







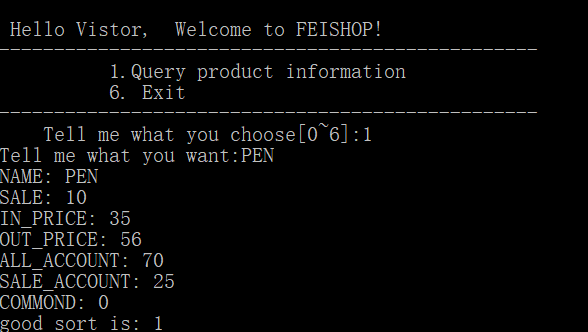


图12.测试

下面进行错误输入测试：

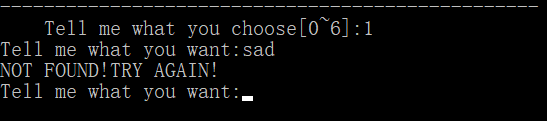
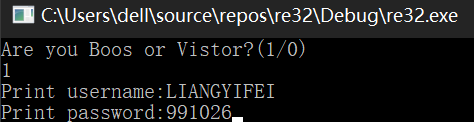
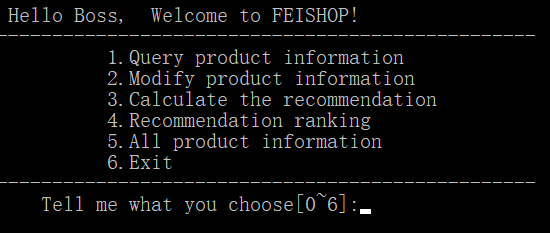
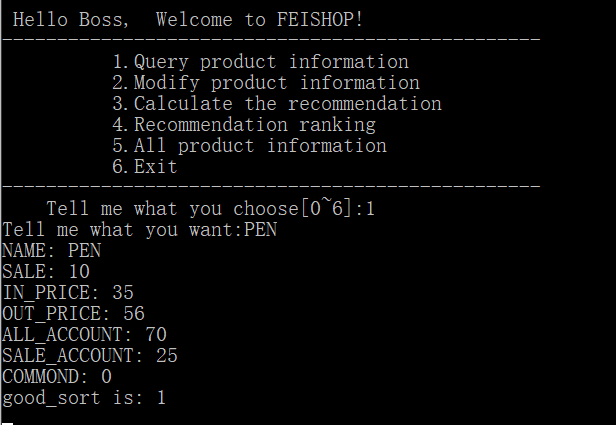


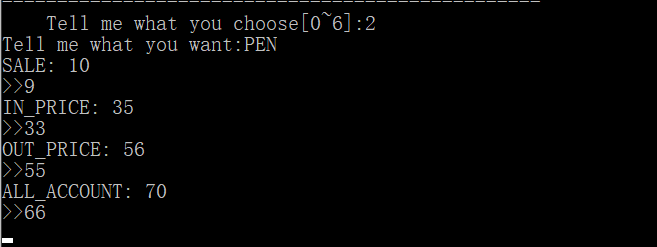
图13.错误输入测试

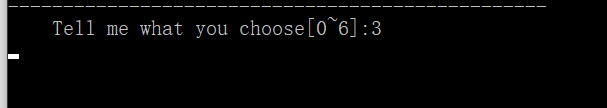
3.登入状态测试：

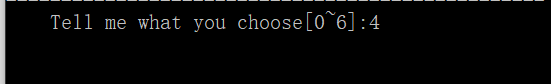












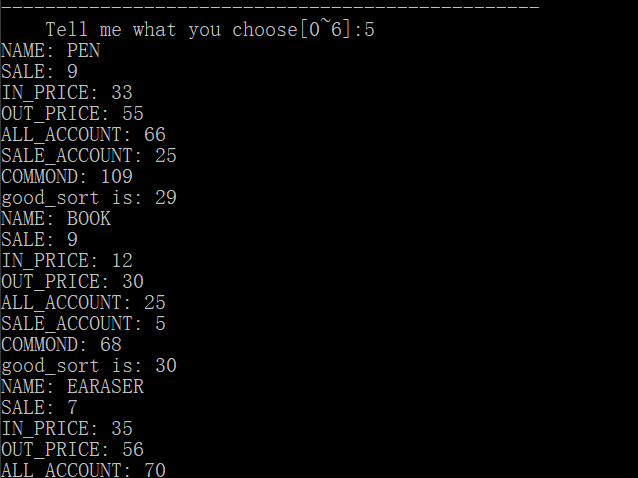


图14.测试

### 3.4 任务2 总结与体会

本次实验最大的问题是：变量定义在哪？一开始我将商品变量以数组形式定义在了C主程序里，导致后面汇编访问变量时过于复杂且错误百出，于是将变量重新定义在了汇编子程序里。在C主程序里调用汇编函数这一问题很顺利的得到了解决，但是如何在汇编程序里调用C函数却令我无比头疼，一开始我打算用C语言实现功能一，这就导致寻找商品的函数：FIND需要用C语言实现，而汇编子程序里的功能二依然需要用到FIND，两者之间却怎么都调用不了。最后我只好放弃，所有功能都用汇编来写，汇编里面只调用C的Printf和Scanf,这样程序才能完美运行。这次实验我觉得设计得很不好，C和汇编的变量传递，函数调用非常容易导致程序逻辑混乱，且如果一个功能用C语言实现，其余功能用汇编实现，又会导致变量之间数据传递出现各种问题，毕竟每一个功能都需要用到商品变量、和用户输入的变量，这要么定义在C语言程序段里，要么定义在汇编程序段里，用两种语言同时调用就是人为增加不便性，导致程序结构很混乱，我认为这样做是不可取的。

# 实验四 中断与反跟踪

**实验名称： 中断与反跟踪**

**实验地点： 南一楼804室**

**指导教师： 张勇**

**专业班级：计算机科学与技术201706班**

**学 号： U2015414556 姓 名： 梁一飞**

**同组学生： 贺子杰、胡晨风 报告日期：2019年4月26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： 纸质版再签名

日期：2017.3.17

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：

**实验目的和要求**

(1) 掌握中断矢量表的概念；

(2） 熟悉I/O访问，BIOS功能调用方法；

(3) 掌握实方式下中断处理程序的编制与调试方法；

(4) 熟悉跟踪与反跟踪的技术；

(5) 提升对计算机系统的理解与分析能力。

## 实验内容

任务1：用三种方式获取中断类型码1H 、13H对应的中断处理程序的入口地址。

**要求：**首先要进入虚拟机状态，然后

（1） 直接运行调试工具（TD.EXE），在其数据区观察中断矢量表中的信息。

（2） 编写程序，用 DOS系统功能调用（具体调用方法见教材示例及附录中的描述）方式获取，观察功能调用相应的出口参数与“（1）”看到的结果是否相同 （使用TD观看出口参数即可）。

（3） 编写程序，直接读取相应内存单元，观察读到的数据与“（1）”看到的结果是否相同 （使用T、D观看程序的执行结果即可）。

任务2：编写一个接管键盘中断的中断服务程序并驻留内存，其主要功能是：在程序驻留并返回到DOS操作系统后，输入键盘上的大写字母时都变成了小写字母。

**要求**：

（1）在 DOS虚拟机下执行程序，中断服务程序驻留内存。

（2）在DOS命令行下键入小写字母时，屏幕显示不变，键入大写时，屏幕显示为小写。执行TD，在代码区输入指令“mov AX,0”，看是否都变成了小写。执行实验三任务1的程序，输入大小写是否正常？

（3）选作：**另外单独**编写一个中断服务程序的卸载程序，将键盘的中断服务程序恢复到原来的状态（只需要还原中断矢量表的信息，先前驻留的程序可以不退出内存）。

任务3：读取CMOS内指定单元的信息，按照16进制形式显示在屏幕上。

**要求**：

1. 在数据段定义一个待读取的CMOS内部单元的地址编号。再使用IN/OUT指令，读取CMOS内的指定单元的信息。

（2） 将读取的信息用16进制的形式显示在屏幕上。若是时间信息，可以人工判断一下是否与操作系统显示的时间一致。

**任务4：数据加密与反跟踪**

在实验三任务1的**网店商品信息管理程序**的基础上，增加输入用户名和密码时，最大错误次数的限制，即，当输入错误次数达到三次时，直接按照未登录状态进入后续功能。老板的密码采用密文的方式存放在数据段中，各种商品的进货价也以密文方式存放在数据段中。加密方法自选（但不应选择复杂的加密算法）。

可以采用计时、中断矢量表检查、堆栈检查、间接寻址等反跟踪方法中的几种方法组合起来进行反跟踪（建议采用两种反跟踪方法，重点是深入理解和运用好所选择的反跟踪方法）。

为简化录入和处理的工作量，只需要定义三种商品的信息即可。

**提示：**为了使源程序的数据段中定义的密码、进货价等在汇编之后变成密文（也就是在最后交付出去的执行程序中看不到明文），可以使用数值运算符（参见教材P48）对变量的初始值进行变换。例如，如果想使进货价50变成密文，加密算法是与老板密码中的字符“W”做异或运算，则可写成：

DB 50 XOR ‘W’

任务5：跟踪与数据解密

解密同组同学的加密程序，获取各个商品的进货价。

注意：两人一组，每人实现一套自己选择的加密与反跟踪方法，把执行程序交给对方解密（解密时间超过半小时的，说明反跟踪方法基本有效）。如何设计反跟踪程序以及如何跟踪破解，是本次实验报告中重点需要突出的内容。

## 实验过程

### 4.1 任务1

#### 4.1.1 实验步骤

方法一：

1. 准备上机实验环境。

2.打开DOSBOX，直接输入TD命令打开程序，在数据段中查找00:0004和00:004CH,查看对应区域存储的值

方法二：

1.编写程序，代码见附件文件

2.利用DOS系统功能的35H号调用，将1H号中断和13H号中断的CS和IP复制到ES和BX中

3.使用DOSBOX进行TD调试，观察ES 和BX 中的值

方法三：

1.根据中断矢量表起始位置和每个中断所占空间相同的规律可知1号中断的地址为00：0004，13号中断的地址为00：004C，编写程序，程序见附件文件

2.运行DOSBOX,观察BX和CX寄存器中的值

#### 4.1.2 源程序

**T11:**

.386

CODE SEGMENT USE16

ASSUME CS:CODE

START:XOR AX,AX

MOV DS,AX

MOV BX,DS:[4]

MOV CX,DS:[6]

MOV AX,3501H

INT 21H

MOV AH,4CH

INT 21H

CODE ENDS

END START

**T12:**

.386

CODE SEGMENT USE16

ASSUME CS:CODE

START:XOR AX,AX

MOV DS,AX

MOV AX,3513H

INT 21H

MOV AH,4CH

INT 21H

CODE ENDS

END START

**T13:**

.386

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT USE16

ASSUME CS:CODE,SS:STACK

START:XOR AX,AX

MOV DS,AX

XOR BX,BX

XOR CX,CX

MOV BX,DS:[4C]

MOV CX,DS:[4E]

MOV AH,4CH

INT 21H

CODE ENDS

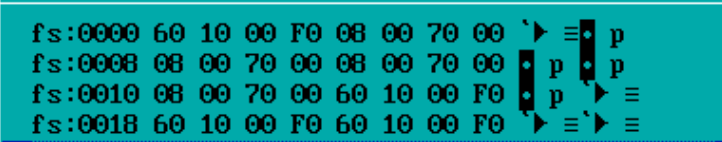
END START

#### 4.1.2 实验记录与分析

方法一：

1H:ip:0008 cs:0070

13H:ip:1140 cs:F000



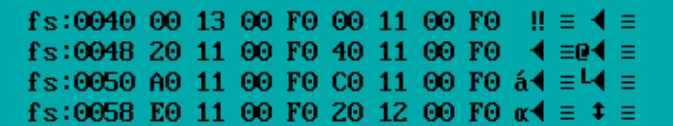
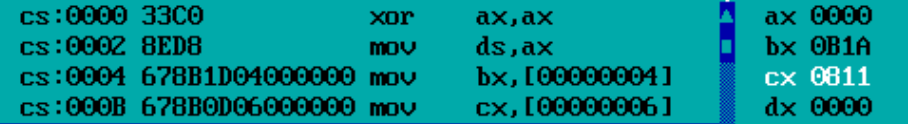


图1.测试

方法二：

01H:



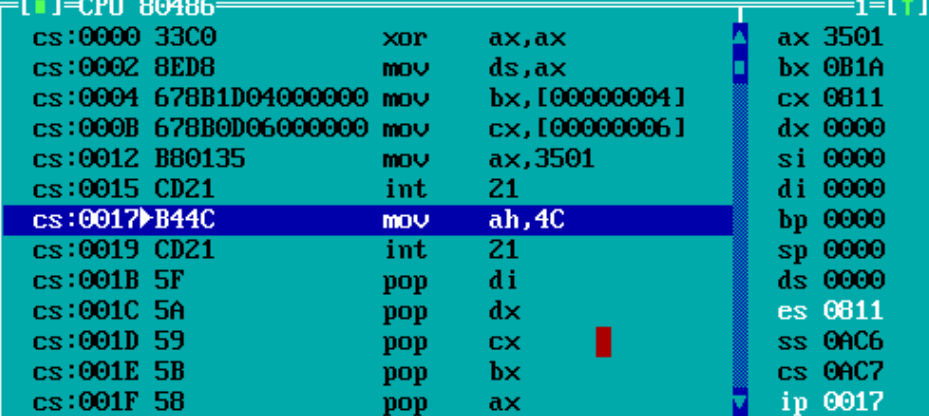


图2.测试

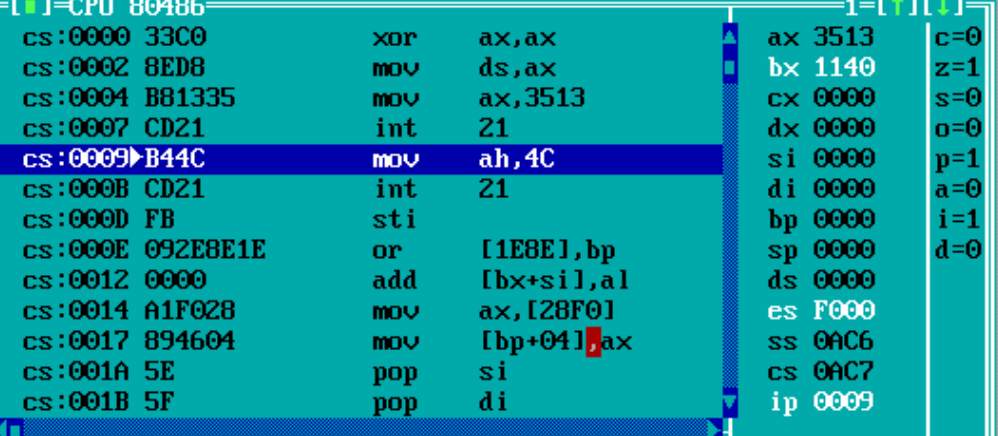
13H: 

图3.测试

由于01H被保护了，所以先用bx和cx获取内容

方法三：

2.13H：

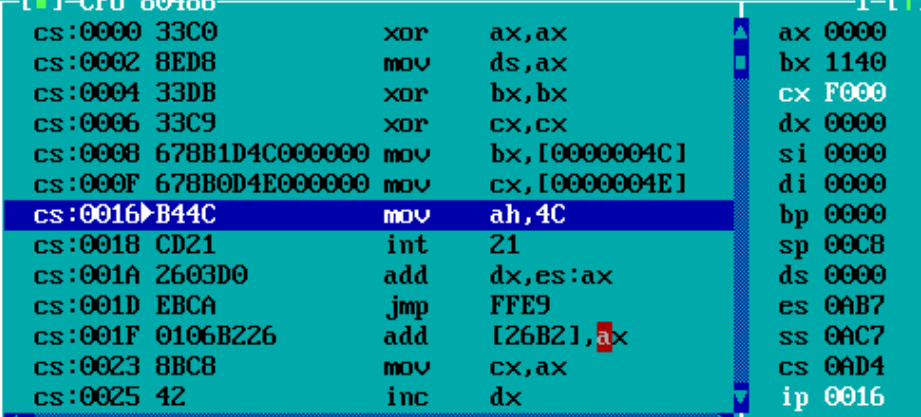


图4.测试

### 4.2 任务2

#### 4.2.1 实验步骤

1.准备实验环境

2.编写程序，程序见附件

3.运行dosbox编译链接运行

#### 4.2.2 源程序

**T21:**

.386

stack segment use16 stack

db 200 dup(0)

stack ends

code segment use16

assume cs:code,ss:stack

old\_int dw ?,?

new16h:cmp ah,0

je quit

cmp ah,10h

je quit

jmp dword ptr old\_int

quit:pushf

call dword ptr old\_int

mov ah,'A'

cmp al,ah

jl exit

mov ah,'Z'

cmp al,ah

jg exit

add al,32

exit:iret

start: xor ax,ax

mov ds,ax

mov ax,ds:[16h\*4];

mov old\_int,ax

mov ax,ds:[16h\*4+2]

mov old\_int+2,ax

;保存旧中断

cli

mov word ptr ds:[16h\*4],offset new16h

;新偏移地址

mov ds:[16h\*4+2],cs

sti

mov dx,offset start+15

shr dx,4

add dx,10h

mov al,0;退出码为0

mov ah,31h

int 21h

code ends

end start

**T22:**

.386

stack segment use16 stack

db 200 dup(0)

stack ends

code segment use16

assume cs:code,ss:stack

start: push bx

xor ax,ax

mov ds,ax

cli

mov bx,11e0h

mov ds:[16h\*4],bx

mov bx,0f000h

mov ds:[16h\*4+2],bx

;还原旧中断

sti

mov dx,offset start+15

shr dx,4

pop bx

add dx,10h

mov al,0;退出码为0

mov ah,31h

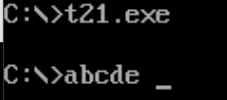
int 21h

code ends

end start

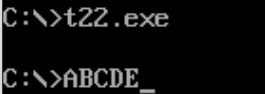
#### 4.2.3 实验记录与分析

1.开启大写键，输入ABCDE



可见全部变成了小写

2.运行还原程序，再输入ABCDE



### 4.3 任务3

#### 4.3.1 实验步骤

1.编写程序

2.编译、链接、运行

#### 4.3.2 源程序

.386

DATA SEGMENT USE16

BUF DB 2

DATA ENDS

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT USE16

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

START:

MOV AX,DATA

MOV DS,AX

LOPA:

MOV AH,1

INT 21H

SUB AL,30H ;transfer to 十进制

OUT 70H,AL ;take out message

IN AL,71H

MOV AH,AL

AND AL,0FH

SHR AH,4

ADD AX,3030H ;transfer to ASCII

XCHG AH,AL

MOV BX,AX

LEA SI,BUF

MOV WORD PTR [SI+1],BX

MOV DL,0AH

MOV AH,2

INT 21H

MOV DL,0DH

MOV AH,2

INT 21H

MOV DL,BL

MOV AH,2

INT 21H

MOV DL,BH

MOV AH,2

INT 21H

MOV DL,0AH

MOV AH,2

INT 21H

MOV DL,0DH

MOV AH,2

INT 21H

JMP LOPA

MOV AH,4CH

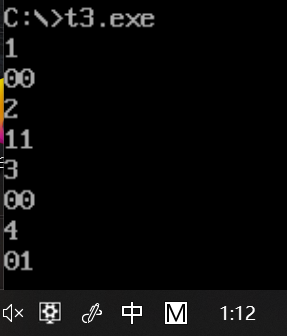
INT 21H

CODE ENDS

END START

#### 4.3.3 实验记录与分析

读取分秒时



还可输入其他功能调用，此处仅演示前四个。

### 4.4 任务4

#### 4.4.1 实验步骤以及源程序

1.加密方法

将密码全部异或了一个‘A’，将进货价与已售数量减去10，并将账号密码的输入上限改为了3.

BPASS DB '9' XOR 'A' *;密码*

DB '9' XOR 'A'

DB '1' XOR 'A'

DB '0' XOR 'A'

DB '2' XOR 'A'

DB '6' XOR 'A'

GA1 DB 'PEN',7 DUP(0) ,10 *;商品名称及折扣*

DW 35-10,56,70-10,25,? ;推荐度还未计算

2.反追踪方法

在程序开头，我修改了中断矢量表，在输入账号密码的时候，加入了时间判断，具体修改如下

.386

DATA SEGMENT USE16

LEAP DW 80 DUP(0)

BNAME DB 'LIANGYIFEI' ;老板姓名

BPASS DB '9' XOR 'A' ;密码

DB '9' XOR 'A'

DB '1' XOR 'A'

DB '0' XOR 'A'

DB '2' XOR 'A'

DB '6' XOR 'A'

SNAME DB 'FEISHOP',0DH,0AH,'$' ;网店名称，用0结束

BUF DB 12 DUP(?)

N EQU 3

POINT DB 'print username and password:',0DH,0AH,'$'

TIP DB 'MISMATCH!',0DH,0AH,'$'

TIP2 DB 'NOT FOUND!TRY AGAIN!',0DH,0AH,'$'

POINT2 DB 'Tell me what you want:',0DH,0AH,'$'

POINT3 DB 'Tell me what you choose:',0DH,0AH,'$'

POINT4 DB 'ERROR!',0DH,0AH,'$'

TAG DB 0DH,0AH,'$'

SIGN DB ?

P DB 3

DAT DW 0AH

TIP3 DB 'Welcome BOSS!According the following tips to choose:',0DH,0AH,'1.Query product information',0DH,0AH,'2.Modify product information',0DH,0AH,'3.Calculate the recommendation',0DH,0AH,'4.Recommendation ranking',0DH,0AH,'5.All product information',0DH,0AH,'6.Exit',0DH,0AH,'$'

TIP4 DB 'Welcome to FEISHOP!According the following tips to choose:',0DH,0AH,'1.Query product information',0DH,0AH,'6.Exit',0DH,0AH,'$'

IN\_NAME DB 11

DB ?

DB 11 DUP(0)

IN\_PWD DB 7

DB ?

DB 7 DUP(0)

IN\_GA DB 11

DB ?

DB 11 DUP(0)

AUTH DB ?

I DW 0

GA1 DB 'PEN',7 DUP(0) ,10 ;商品名称及折扣

DW 35-10,56,70-10,25,? ;推荐度还未计算

GA2 DB 'BOOK', 6 DUP(0) ,9 ;商品名称及折扣

DW 12-10,30,25-10,5,? ;推荐度还未计算

GAN DB N-2 DUP('Temp-Value',8,15-10,0,20,0,30-10,0,2,0,?,?)

NUM DB 10

DB ?

DB 10 DUP(0)

OLDINT1 DW 0,0

OLDINT3 DW 0,0

DATA ENDS

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT USE16

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

START:MOV AX,DATA

MOV DS,AX

XOR AX,AX

MOV ES,AX

MOV AX,ES:[1\*4]

MOV OLDINT1,AX

MOV AX,ES:[1\*4+2]

MOV OLDINT1+2,AX

MOV AX,ES:[3\*4]

MOV OLDINT3,AX

MOV AX,ES:[3\*4+2]

MOV OLDINT3+2,AX

CLI

MOV AX,OFFSET NEWINT

MOV ES:[1\*4],AX

MOV ES:[1\*4+2],CS

MOV ES:[3\*4],AX

MOV ES:[3\*4+2],CS

STI

LEA DX,SNAME

MOV AH,9

INT 21H

MOV SI,OFFSET GA1;计算全部商品的推荐度

CALL CACUL

FUN1: LEA DX,POINT

MOV AH,9

INT 21H

LEA DX,IN\_NAME

MOV AH,10

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

MOV CL,IN\_NAME[1]

CMP CL,0

JE T1

CMP CL,1

JE ISQ

JMP IPW

ISQ: MOV CL,IN\_NAME[2]

CMP CL,'q'

JE QUIT

IPW: LEA DX,IN\_PWD

MOV AH,10

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

cli

MOV AH,2CH

INT 21H

PUSH DX

FUN2\_1: MOV CL,IN\_NAME[1]

CMP CL,10

JNE T2

MOV DI,10

JMP LOPA1

FUN2\_2: MOV CL,IN\_PWD[1]

CMP CL,6

JNE T2

MOV DI,6

JMP LOPA2

LOPA1:MOV CL,BNAME[DI-1]

CMP CL,IN\_NAME[DI+1]

JNE T2

DEC DI

CMP DI,0

JNE LOPA1

JMP FUN2\_2

LOPA2:MOV CL,IN\_PWD[DI+1]

XOR CL,'A'

CMP CL,BPASS[DI-1]

JNE T2

DEC DI

CMP DI,0

JNE LOPA2

MOV AH,2CH

INT 21H

STI

CMP DX,[ESP]

POP DX

JZ OK

JMP QUIT

OK: MOV AUTH,1

FUN3\_1:CMP AUTH,1

JZ PRIN

LEA DX,TIP4

MOV AH,9

INT 21H

JMP FUN3\_2

PRIN:LEA DX,TIP3

MOV AH,9

INT 21H

FUN3\_2:LEA DX,POINT3

MOV AH,9

INT 21H

MOV AH,1

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

CMP AL,'1'

JZ F1C

CMP AL,'2'

JZ F2C

CMP AL,'3'

JZ F3C

CMP AL,'4'

JZ F4C

CMP AL,'5'

JZ F5C

CMP AL,'6'

JZ F6C

WRO: LEA DX,POINT4

MOV AH,9

INT 21H

JMP FUN3\_2

F1C: CALL FUNC1

CALL CLEAR

JMP FUN3\_1

F2C: CMP AUTH,0

JE WRO

CALL FUNC2

CALL CLEAR

JMP FUN3\_1

F3C: CMP AUTH,0

JE WRO

MOV SI,OFFSET GA1

CALL FUNC3

LEA DX,TAG

MOV AH,9

INT 21H

JMP FUN3\_1

F4C: CMP AUTH,0

JE WRO

MOV SI,OFFSET GA1

CALL FUNC4

JMP FUN3\_1

F5C: CMP AUTH,0

JE WRO

MOV SI,OFFSET GA1

CALL FUNC5

LEA DX,TAG

MOV AH,9

INT 21H

JMP FUN3\_1

F6C: CALL CLEAR

JMP FUN1

FUNC1 PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

PUSH SI

PUSH DI

LEA DX,POINT2

MOV AH,9

INT 21H

LEA DX,IN\_GA

MOV AH,10

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

MOV CL,IN\_GA[1]

CMP CL,0

JE F1\_3

MOV DI,OFFSET IN\_GA

MOV SI,OFFSET GA1

CALL FIND

F1\_2: MOV CL,IN\_GA[1]

MOV BX,0

PRINTNAME: ADD CL,-1

JS PRINTVALUE

MOV AH,02H

MOV DL,[SI][BX]

INT 21H

ADD BX,1

JMP PRINTNAME

PRINTVALUE: LEA DX,' '

MOV AH,2

INT 21H

MOV AX,0

MOV AL,[SI+10]

CALL F2T10

CMP AUTH,0

JE T13

MOV AX,WORD PTR[SI+11]

ADD AX,10

CALL F2T10

T13: MOV AX,WORD PTR[SI+13]

CALL F2T10

CMP AUTH,0

JE T17

MOV AX,WORD PTR[SI+15]

ADD AX,10

CALL F2T10

T17: MOV AX,WORD PTR[SI+17]

CALL F2T10

MOV AX,WORD PTR[SI+19]

CMP AX,100

JA ALEVEL

CMP AX,50

JA BLEVEL

CMP AX,10

JA CLEVEL

JMP FLEVEL

ALEVEL: LEA DX,'A'

MOV AH,2

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

JMP F1\_3

BLEVEL: LEA DX,'B'

MOV AH,2

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

JMP F1\_3

CLEVEL: LEA DX,'C'

MOV AH,2

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

JMP F1\_3

FLEVEL: LEA DX,'F'

MOV AH,2

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

F1\_3: POP DI

POP SI

POP DX

POP CX

POP BX

POP AX

RET

FUNC1 ENDP

FUNC2 PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

PUSH SI

PUSH DI

LEA DX,POINT2

MOV AH,9

INT 21H

LEA DX,IN\_GA

MOV AH,10

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

MOV CL,IN\_GA[1]

CMP CL,0

JE F2\_6

MOV DI,OFFSET IN\_GA

MOV SI,OFFSET GA1

CALL FIND

F2\_2: MOV AX,0

MOV AL,[SI+10]

CALL F2T10

LEA DX,'>'

MOV AH,2

INT 21H

LEA DX,NUM

MOV AH,10

INT 21H

MOV DI,OFFSET NUM

ADD DI,2

MOV CL,NUM[1]

CALL F10T2

CMP DI,-1

JE F2\_2

MOV [SI+10],AL

LEA DX,TAG

MOV AH,9

INT 21H

F2\_3: MOV AX,[SI+11]

CALL F2T10

LEA DX,'>'

MOV AH,2

INT 21H

LEA DX,NUM

MOV AH,10

INT 21H

MOV DI,OFFSET NUM

ADD DI,2

MOV CL,NUM[1]

CALL F10T2

CMP DI,-1

JE F2\_3

MOV [SI+11],WORD PTR AX

LEA DX,TAG

MOV AH,9

INT 21H

F2\_4: MOV AX,[SI+13]

CALL F2T10

LEA DX,'>'

MOV AH,2

INT 21H

LEA DX,NUM

MOV AH,10

INT 21H

MOV DI,OFFSET NUM

ADD DI,2

MOV CL,NUM[1]

CALL F10T2

CMP DI,-1

JE F2\_4

MOV [SI+13],WORD PTR AX

LEA DX,TAG

MOV AH,9

INT 21H

F2\_5: MOV AX,[SI+15]

CALL F2T10

LEA DX,'>'

MOV AH,2

INT 21H

LEA DX,NUM

MOV AH,10

INT 21H

MOV DI,OFFSET NUM

ADD DI,2

MOV CL,NUM[1]

CALL F10T2

CMP DI,-1

JE F2\_5

MOV [SI+15],AL

LEA DX,TAG

MOV AH,9

INT 21H

F2\_6: POP DI

POP SI

POP DX

POP CX

POP BX

POP AX

RET

FUNC2 ENDP

T1: MOV AUTH,0

ADD P,3

JMP FUN3\_1

T2: LEA DX,TIP

MOV AH,9

INT 21H

SUB P,1

CMP P,0

JE T1

JMP FUN1

T3: LEA DX,TIP2

MOV AH,9

INT 21H

JMP FUN3\_1

FIND PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

MOV CL,IN\_GA[1]

MOVZX AX,CL

ADD DI,AX

ADD DI,2

MOV AL,0

MOV [DI],AL

SUB SI,21

MOV DI,OFFSET IN\_GA

MOV I,N

LOPA3:ADD I,-1

JS T3

ADD SI,21

MOV CL,9

LOPA4:MOV BL,CL

MOV AL,2[DI+BX]

CMP AL,0[SI][BX]

JNE LOPA3

ADD CL,-1

JS FEND

JMP LOPA4

FEND: POP DX

POP CX

POP BX

POP AX

RET

FIND ENDP

CLEAR PROC

MOV AX,0

MOV BX,0

MOV CX,0

MOV DX,0

MOV SI,0

MOV DI,0

MOV BX,11

MOV SI, OFFSET IN\_GA

CLEAR1: MOV BYTE PTR[SI+BX],0

ADD BX,-1

CMP BX,0

JNE CLEAR1

MOV SI,0

RET

CLEAR ENDP

CACUL PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

ADD SI,-21

MOV I,N

LOPA5:ADD I,-1

JS CAUEND

ADD SI,21

MOV AX,WORD PTR [SI+11]

ADD AX,10

MOV BX,0080H

MUL BX

MOV BX,WORD PTR [SI+13]

DIV BX

MOV BX,000AH

MUL BX

MOV BL,[SI+10]

MOV BH,00H

DIV BX

MOV CX,AX

MOV AX,WORD PTR [SI+17]

MOV BX,0040H

MUL BX

MOV BX,WORD PTR [SI+15]

ADD BX,10

DIV BX

ADD AX,CX

MOV WORD PTR [SI+19],AX

MOV AX,0

MOV BX,0

MOV CX,0

JMP LOPA5

CAUEND: POP DX

POP CX

POP BX

POP AX

RET

CACUL ENDP

NEWINT:IRET

QUIT: CLI

MOV AX,OLDINT1

MOV ES:[1\*4],AX

MOV AX,OLDINT1+2

MOV ES:[1\*4+2],AX

MOV AX,OLDINT3

MOV ES:[3\*4],AX

MOV AX,OLDINT3+2

MOV ES:[3\*4+2],AX

STI

MOV AH,4CH

INT 21H

F10T2 PROC FAR

PUSH DX

PUSH BX

MOV AX,0

MOV SIGN,0

MOV BL,[DI]

CMP BL,'+'

JZ A

CMP BL,'-'

JNE NEXT2

MOV SIGN,1

A:

DEC CX

JZ ERR

NEXT1:

INC DI

MOV BL,[DI]

NEXT2:

CMP BL,'0'

JB ERR

CMP BL,'9'

JA ERR

SUB BL,30H

MOV BH,0

MUL DAT

JO ERR

ADD AX,BX

JC ERR

DEC CX

JNZ NEXT1

CMP SIGN,1

JNE QQ

NEG AX

QQ:

POP BX

POP DX

RET

ERR:

MOV DI,-1

JMP QQ

F10T2 ENDP

F2T10 PROC FAR

PUSH EBX

PUSH SI

LEA SI,BUF

CMP DX,32

JE B

MOVSX EAX,AX

B: OR EAX,EAX

JNS PLUS

NEG EAX

MOV BYTE PTR [SI],'-'

INC SI

PLUS: MOV EBX,10

CALL RDAIX

MOV BYTE PTR [SI],'$'

LEA DX,BUF

MOV AH,9

INT 21H

LEA DX,' '

MOV AH,2

INT 21H

POP SI

POP EBX

RET

F2T10 ENDP

RDAIX PROC FAR

PUSH CX

PUSH EDX

XOR CX,CX

LOP1:XOR EDX,EDX

DIV EBX

PUSH DX

INC CX

OR EAX,EAX

JNZ LOP1

LOP2:POP AX

CMP AL,10

JB L1

ADD AL,7

L1: ADD AL,30H

MOV [SI],AL

INC SI

LOOP LOP2

POP EDX

POP CX

RET

RDAIX ENDP

CODE ENDS

END START

#### 4.4.2 源程序

**T41:**

EXTERN FUNC3 :FAR

EXTERN FUNC4 :FAR

EXTERN FUNC5 :FAR

PUBLIC F2T10

PUBLIC RDAIX

.386

DATA SEGMENT USE16

LEAP DW 80 DUP(0)

BNAME DB 'LIANGYIFEI' ;老板姓名

BPASS DB '9' XOR 'A' ;密码

DB '9' XOR 'A'

DB '1' XOR 'A'

DB '0' XOR 'A'

DB '2' XOR 'A'

DB '6' XOR 'A'

SNAME DB 'FEISHOP',0DH,0AH,'$' ;网店名称，用0结束

BUF DB 12 DUP(?)

N EQU 3

POINT DB 'print username and password:',0DH,0AH,'$'

TIP DB 'MISMATCH!',0DH,0AH,'$'

TIP2 DB 'NOT FOUND!TRY AGAIN!',0DH,0AH,'$'

POINT2 DB 'Tell me what you want:',0DH,0AH,'$'

POINT3 DB 'Tell me what you choose:',0DH,0AH,'$'

POINT4 DB 'ERROR!',0DH,0AH,'$'

TAG DB 0DH,0AH,'$'

SIGN DB ?

P DB 3

DAT DW 0AH

TIP3 DB 'Welcome BOSS!According the following tips to choose:',0DH,0AH,'1.Query product information',0DH,0AH,'2.Modify product information',0DH,0AH,'3.Calculate the recommendation',0DH,0AH,'4.Recommendation ranking',0DH,0AH,'5.All product information',0DH,0AH,'6.Exit',0DH,0AH,'$'

TIP4 DB 'Welcome to FEISHOP!According the following tips to choose:',0DH,0AH,'1.Query product information',0DH,0AH,'6.Exit',0DH,0AH,'$'

IN\_NAME DB 11

DB ?

DB 11 DUP(0)

IN\_PWD DB 7

DB ?

DB 7 DUP(0)

IN\_GA DB 11

DB ?

DB 11 DUP(0)

AUTH DB ?

I DW 0

GA1 DB 'PEN',7 DUP(0) ,10 ;商品名称及折扣

DW 35-10,56,70-10,25,? ;推荐度还未计算

GA2 DB 'BOOK', 6 DUP(0) ,9 ;商品名称及折扣

DW 12-10,30,25-10,5,? ;推荐度还未计算

GAN DB N-2 DUP('Temp-Value',8,15-10,0,20,0,30-10,0,2,0,?,?)

NUM DB 10

DB ?

DB 10 DUP(0)

OLDINT1 DW 0,0

OLDINT3 DW 0,0

DATA ENDS

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT USE16

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

START:MOV AX,DATA

MOV DS,AX

XOR AX,AX

MOV ES,AX

MOV AX,ES:[1\*4]

MOV OLDINT1,AX

MOV AX,ES:[1\*4+2]

MOV OLDINT1+2,AX

MOV AX,ES:[3\*4]

MOV OLDINT3,AX

MOV AX,ES:[3\*4+2]

MOV OLDINT3+2,AX

CLI

MOV AX,OFFSET NEWINT

MOV ES:[1\*4],AX

MOV ES:[1\*4+2],CS

MOV ES:[3\*4],AX

MOV ES:[3\*4+2],CS

STI

LEA DX,SNAME

MOV AH,9

INT 21H

MOV SI,OFFSET GA1;计算全部商品的推荐度

CALL CACUL

FUN1: LEA DX,POINT

MOV AH,9

INT 21H

LEA DX,IN\_NAME

MOV AH,10

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

MOV CL,IN\_NAME[1]

CMP CL,0

JE T1

CMP CL,1

JE ISQ

JMP IPW

ISQ: MOV CL,IN\_NAME[2]

CMP CL,'q'

JE QUIT

IPW: LEA DX,IN\_PWD

MOV AH,10

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

cli

MOV AH,2CH

INT 21H

PUSH DX

FUN2\_1: MOV CL,IN\_NAME[1]

CMP CL,10

JNE T2

MOV DI,10

JMP LOPA1

FUN2\_2: MOV CL,IN\_PWD[1]

CMP CL,6

JNE T2

MOV DI,6

JMP LOPA2

LOPA1:MOV CL,BNAME[DI-1]

CMP CL,IN\_NAME[DI+1]

JNE T2

DEC DI

CMP DI,0

JNE LOPA1

JMP FUN2\_2

LOPA2:MOV CL,IN\_PWD[DI+1]

XOR CL,'A'

CMP CL,BPASS[DI-1]

JNE T2

DEC DI

CMP DI,0

JNE LOPA2

MOV AH,2CH

INT 21H

STI

CMP DX,[ESP]

POP DX

JZ OK

JMP QUIT

OK: MOV AUTH,1

FUN3\_1:CMP AUTH,1

JZ PRIN

LEA DX,TIP4

MOV AH,9

INT 21H

JMP FUN3\_2

PRIN:LEA DX,TIP3

MOV AH,9

INT 21H

FUN3\_2:LEA DX,POINT3

MOV AH,9

INT 21H

MOV AH,1

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

CMP AL,'1'

JZ F1C

CMP AL,'2'

JZ F2C

CMP AL,'3'

JZ F3C

CMP AL,'4'

JZ F4C

CMP AL,'5'

JZ F5C

CMP AL,'6'

JZ F6C

WRO: LEA DX,POINT4

MOV AH,9

INT 21H

JMP FUN3\_2

F1C: CALL FUNC1

CALL CLEAR

JMP FUN3\_1

F2C: CMP AUTH,0

JE WRO

CALL FUNC2

CALL CLEAR

JMP FUN3\_1

F3C: CMP AUTH,0

JE WRO

MOV SI,OFFSET GA1

CALL FUNC3

LEA DX,TAG

MOV AH,9

INT 21H

JMP FUN3\_1

F4C: CMP AUTH,0

JE WRO

MOV SI,OFFSET GA1

CALL FUNC4

JMP FUN3\_1

F5C: CMP AUTH,0

JE WRO

MOV SI,OFFSET GA1

CALL FUNC5

LEA DX,TAG

MOV AH,9

INT 21H

JMP FUN3\_1

F6C: CALL CLEAR

JMP FUN1

FUNC1 PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

PUSH SI

PUSH DI

LEA DX,POINT2

MOV AH,9

INT 21H

LEA DX,IN\_GA

MOV AH,10

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

MOV CL,IN\_GA[1]

CMP CL,0

JE F1\_3

MOV DI,OFFSET IN\_GA

MOV SI,OFFSET GA1

CALL FIND

F1\_2: MOV CL,IN\_GA[1]

MOV BX,0

PRINTNAME: ADD CL,-1

JS PRINTVALUE

MOV AH,02H

MOV DL,[SI][BX]

INT 21H

ADD BX,1

JMP PRINTNAME

PRINTVALUE: LEA DX,' '

MOV AH,2

INT 21H

MOV AX,0

MOV AL,[SI+10]

CALL F2T10

CMP AUTH,0

JE T13

MOV AX,WORD PTR[SI+11]

ADD AX,10

CALL F2T10

T13: MOV AX,WORD PTR[SI+13]

CALL F2T10

CMP AUTH,0

JE T17

MOV AX,WORD PTR[SI+15]

ADD AX,10

CALL F2T10

T17: MOV AX,WORD PTR[SI+17]

CALL F2T10

MOV AX,WORD PTR[SI+19]

CMP AX,100

JA ALEVEL

CMP AX,50

JA BLEVEL

CMP AX,10

JA CLEVEL

JMP FLEVEL

ALEVEL: LEA DX,'A'

MOV AH,2

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

JMP F1\_3

BLEVEL: LEA DX,'B'

MOV AH,2

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

JMP F1\_3

CLEVEL: LEA DX,'C'

MOV AH,2

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

JMP F1\_3

FLEVEL: LEA DX,'F'

MOV AH,2

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

F1\_3: POP DI

POP SI

POP DX

POP CX

POP BX

POP AX

RET

FUNC1 ENDP

FUNC2 PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

PUSH SI

PUSH DI

LEA DX,POINT2

MOV AH,9

INT 21H

LEA DX,IN\_GA

MOV AH,10

INT 21H

LEA DX,TAG

MOV AH,9

INT 21H

MOV CL,IN\_GA[1]

CMP CL,0

JE F2\_6

MOV DI,OFFSET IN\_GA

MOV SI,OFFSET GA1

CALL FIND

F2\_2: MOV AX,0

MOV AL,[SI+10]

CALL F2T10

LEA DX,'>'

MOV AH,2

INT 21H

LEA DX,NUM

MOV AH,10

INT 21H

MOV DI,OFFSET NUM

ADD DI,2

MOV CL,NUM[1]

CALL F10T2

CMP DI,-1

JE F2\_2

MOV [SI+10],AL

LEA DX,TAG

MOV AH,9

INT 21H

F2\_3: MOV AX,[SI+11]

CALL F2T10

LEA DX,'>'

MOV AH,2

INT 21H

LEA DX,NUM

MOV AH,10

INT 21H

MOV DI,OFFSET NUM

ADD DI,2

MOV CL,NUM[1]

CALL F10T2

CMP DI,-1

JE F2\_3

MOV [SI+11],WORD PTR AX

LEA DX,TAG

MOV AH,9

INT 21H

F2\_4: MOV AX,[SI+13]

CALL F2T10

LEA DX,'>'

MOV AH,2

INT 21H

LEA DX,NUM

MOV AH,10

INT 21H

MOV DI,OFFSET NUM

ADD DI,2

MOV CL,NUM[1]

CALL F10T2

CMP DI,-1

JE F2\_4

MOV [SI+13],WORD PTR AX

LEA DX,TAG

MOV AH,9

INT 21H

F2\_5: MOV AX,[SI+15]

CALL F2T10

LEA DX,'>'

MOV AH,2

INT 21H

LEA DX,NUM

MOV AH,10

INT 21H

MOV DI,OFFSET NUM

ADD DI,2

MOV CL,NUM[1]

CALL F10T2

CMP DI,-1

JE F2\_5

MOV [SI+15],AL

LEA DX,TAG

MOV AH,9

INT 21H

F2\_6: POP DI

POP SI

POP DX

POP CX

POP BX

POP AX

RET

FUNC2 ENDP

T1: MOV AUTH,0

ADD P,3

JMP FUN3\_1

T2: LEA DX,TIP

MOV AH,9

INT 21H

SUB P,1

CMP P,0

JE T1

JMP FUN1

T3: LEA DX,TIP2

MOV AH,9

INT 21H

JMP FUN3\_1

FIND PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

MOV CL,IN\_GA[1]

MOVZX AX,CL

ADD DI,AX

ADD DI,2

MOV AL,0

MOV [DI],AL

SUB SI,21

MOV DI,OFFSET IN\_GA

MOV I,N

LOPA3:ADD I,-1

JS T3

ADD SI,21

MOV CL,9

LOPA4:MOV BL,CL

MOV AL,2[DI+BX]

CMP AL,0[SI][BX]

JNE LOPA3

ADD CL,-1

JS FEND

JMP LOPA4

FEND: POP DX

POP CX

POP BX

POP AX

RET

FIND ENDP

CLEAR PROC

MOV AX,0

MOV BX,0

MOV CX,0

MOV DX,0

MOV SI,0

MOV DI,0

MOV BX,11

MOV SI, OFFSET IN\_GA

CLEAR1: MOV BYTE PTR[SI+BX],0

ADD BX,-1

CMP BX,0

JNE CLEAR1

MOV SI,0

RET

CLEAR ENDP

CACUL PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

ADD SI,-21

MOV I,N

LOPA5:ADD I,-1

JS CAUEND

ADD SI,21

MOV AX,WORD PTR [SI+11]

ADD AX,10

MOV BX,0080H

MUL BX

MOV BX,WORD PTR [SI+13]

DIV BX

MOV BX,000AH

MUL BX

MOV BL,[SI+10]

MOV BH,00H

DIV BX

MOV CX,AX

MOV AX,WORD PTR [SI+17]

MOV BX,0040H

MUL BX

MOV BX,WORD PTR [SI+15]

ADD BX,10

DIV BX

ADD AX,CX

MOV WORD PTR [SI+19],AX

MOV AX,0

MOV BX,0

MOV CX,0

JMP LOPA5

CAUEND: POP DX

POP CX

POP BX

POP AX

RET

CACUL ENDP

NEWINT:IRET

QUIT: CLI

MOV AX,OLDINT1

MOV ES:[1\*4],AX

MOV AX,OLDINT1+2

MOV ES:[1\*4+2],AX

MOV AX,OLDINT3

MOV ES:[3\*4],AX

MOV AX,OLDINT3+2

MOV ES:[3\*4+2],AX

STI

MOV AH,4CH

INT 21H

F10T2 PROC FAR

PUSH DX

PUSH BX

MOV AX,0

MOV SIGN,0

MOV BL,[DI]

CMP BL,'+'

JZ A

CMP BL,'-'

JNE NEXT2

MOV SIGN,1

A:

DEC CX

JZ ERR

NEXT1:

INC DI

MOV BL,[DI]

NEXT2:

CMP BL,'0'

JB ERR

CMP BL,'9'

JA ERR

SUB BL,30H

MOV BH,0

MUL DAT

JO ERR

ADD AX,BX

JC ERR

DEC CX

JNZ NEXT1

CMP SIGN,1

JNE QQ

NEG AX

QQ:

POP BX

POP DX

RET

ERR:

MOV DI,-1

JMP QQ

F10T2 ENDP

F2T10 PROC FAR

PUSH EBX

PUSH SI

LEA SI,BUF

CMP DX,32

JE B

MOVSX EAX,AX

B: OR EAX,EAX

JNS PLUS

NEG EAX

MOV BYTE PTR [SI],'-'

INC SI

PLUS: MOV EBX,10

CALL RDAIX

MOV BYTE PTR [SI],'$'

LEA DX,BUF

MOV AH,9

INT 21H

LEA DX,' '

MOV AH,2

INT 21H

POP SI

POP EBX

RET

F2T10 ENDP

RDAIX PROC FAR

PUSH CX

PUSH EDX

XOR CX,CX

LOP1:XOR EDX,EDX

DIV EBX

PUSH DX

INC CX

OR EAX,EAX

JNZ LOP1

LOP2:POP AX

CMP AL,10

JB L1

ADD AL,7

L1: ADD AL,30H

MOV [SI],AL

INC SI

LOOP LOP2

POP EDX

POP CX

RET

RDAIX ENDP

CODE ENDS

END START

**T42:**

PUBLIC FUNC3,FUNC4,FUNC5

EXTERN F2T10:FAR

EXTERN RDAIX:FAR

.386

STACK1 SEGMENT USE16 STACK

DB 200 DUP(0)

STACK1 ENDS

DATA SEGMENT USE16 PARA PUBLIC 'DATA'

BUF DB 12 DUP(?)

N EQU 3

M EQU 6

PAI DW N DUP(0)

PAIZ DW N DUP(0)

DATA ENDS

CODE1 SEGMENT USE16

ASSUME CS:CODE1, DS:DATA, SS:STACK1

FUNC3 PROC FAR

PUSH DI

PUSH EAX

PUSH EBX

PUSH ECX

PUSH EDX

MOV DI,N

TJD:MOVZX EAX,WORD PTR 11[SI]

ADD EAX,10

SAL EAX,7

MOV EDX,0

MOVZX EBX,WORD PTR 13[SI]

DIV EBX

MOV EBX,0000000AH

MUL EBX

MOVZX EBX,BYTE PTR 10[SI]

MOV EDX,0

DIV EBX

MOV ECX,EAX

MOVZX EAX,WORD PTR 17[SI]

SAL EAX,6

MOV EDX,0

MOVZX EBX,WORD PTR 15[SI]

ADD EBX,10

DIV EBX

ADD EAX,ECX

MOV 19[SI],AX

ADD SI,21

DEC DI

CMP DI,0

JNZ TJD

POP EDX

POP ECX

POP EBX

POP EAX

POP DI

RET

FUNC3 ENDP

FUNC4 PROC FAR

PUSH DI

PUSH AX

PUSH EBX

PUSH ECX

MOV DI,0

LOOP4: MOV AX,WORD PTR 19[SI]

MOV PAI[DI],AX

ADD SI,21

ADD DI,2

CMP DI,M

JNZ LOOP4

MOV DI,1

LOOP1: MOV EBX,1

MOV ECX,0

MOV AX,PAI

LOOP2: CMP AX,PAI[EBX\*2]

JNB NEXT2

NEXT1: MOV AX,PAI[EBX\*2]

MOV ECX,EBX

NEXT2: ADD EBX,1

CMP EBX,N

JNZ LOOP2

MOV EBX,ECX

MOV PAIZ[EBX\*2],DI

MOV PAI[EBX\*2],0

INC DI

CMP DI,N+1

JNZ LOOP1

POP ECX

POP EBX

POP AX

POP DI

RET

FUNC4 ENDP

;入口参数

; SI 第一个商品的地址

;出口参数 显示所有商品的信息

FUNC5 PROC FAR

PUSH DI

PUSH DX

PUSH BX

PUSH AX

MOV DI,0

LOOP5:

MOV DX,0AH

MOV AH,2

INT 21H

MOV DX,0DH

MOV AH,2

INT 21H

MOV BX,0

LOOP3:

MOV AL,BYTE PTR [BX+SI]

CMP AL,0

JZ K

MOV DL,AL

MOV AH,2

INT 21H

INC BX

CMP BX,10

JNZ LOOP3

K: MOV DX,' '

MOV AH,2

INT 21H

MOV AL,BYTE PTR 10[SI]

MOV AH,0

MOV DX,16

CALL F2T10

MOV DL,' '

MOV AH,2

INT 21H

MOV AX,WORD PTR 11[SI]

ADD AX,10

MOV DX,16

CALL F2T10

MOV DL,' '

MOV AH,2

INT 21H

MOV AX,WORD PTR 13[SI]

MOV DX,16

CALL F2T10

MOV DL,' '

MOV AH,2

INT 21H

MOV AX,WORD PTR 15[SI]

ADD AX,10

MOV DX,16

CALL F2T10

MOV DL,' '

MOV AH,2

INT 21H

MOV AX,WORD PTR 17[SI]

MOV DX,16

CALL F2T10

MOV DL,' '

MOV AH,2

INT 21H

MOV AX,WORD PTR 19[SI]

MOV DX,16

CALL F2T10

MOV DL,' '

MOV AH,2

INT 21H

MOV AX,PAIZ[DI]

MOV DX,16

CALL F2T10

ADD DI,2

ADD SI,21

CMP DI,M

JNZ LOOP5

POP AX

POP BX

POP DX

POP DI

RET

FUNC5 ENDP

CODE1 ENDS

end

#### 4.4.3 实验记录与分析

1.首先尝试暴力破解

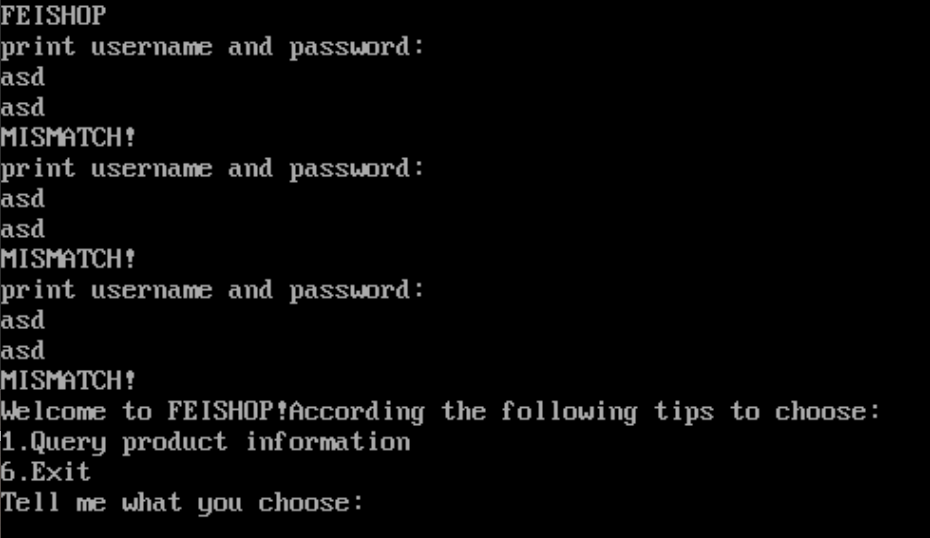


图5.暴力破解

可见，在三次输错密码后，直接进入了游客界面

2.然后进入TD界面，未到输入账号密码，程序就到了一个特殊的地方。

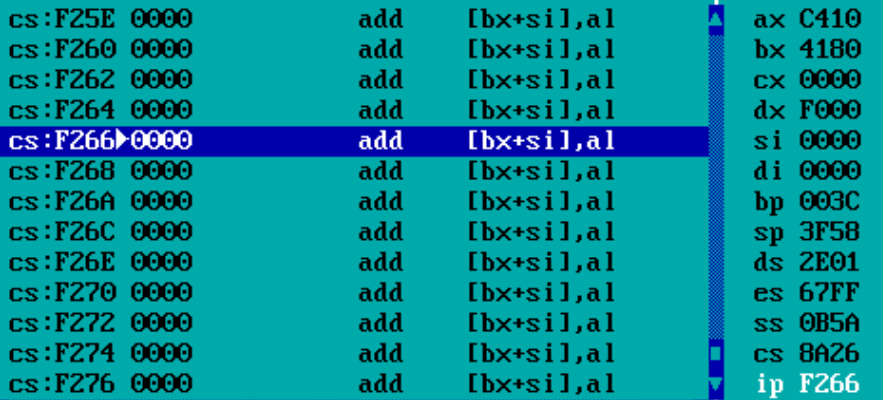


图6.反追踪

### 4.5 任务5

#### 4.5.1 实验步骤

1.对贺子杰的程序尝试暴力破解

2.在TD中找到内存段进行破解

3.采用其他方式破解

#### 4.5.2 实验记录与分析

1.首先对贺子杰的程序尝试暴力破解，可见，输错三次账号密码后，直接进入了游客登陆

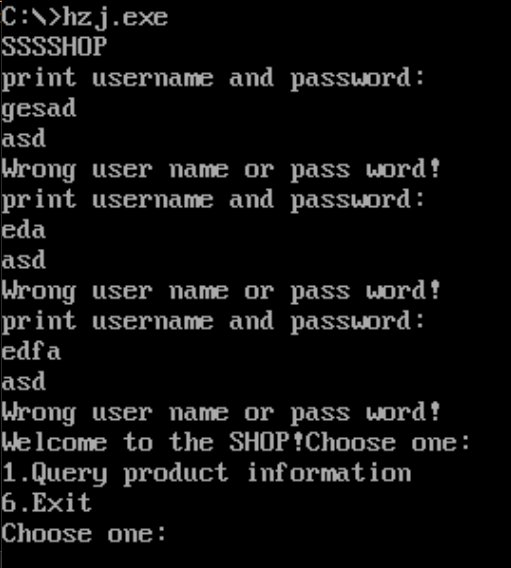


图7.暴力破解

2.寻找内存段



图8.寻找内存段

由于数据进行过加密，获取的数据是错误的

3.单步调试

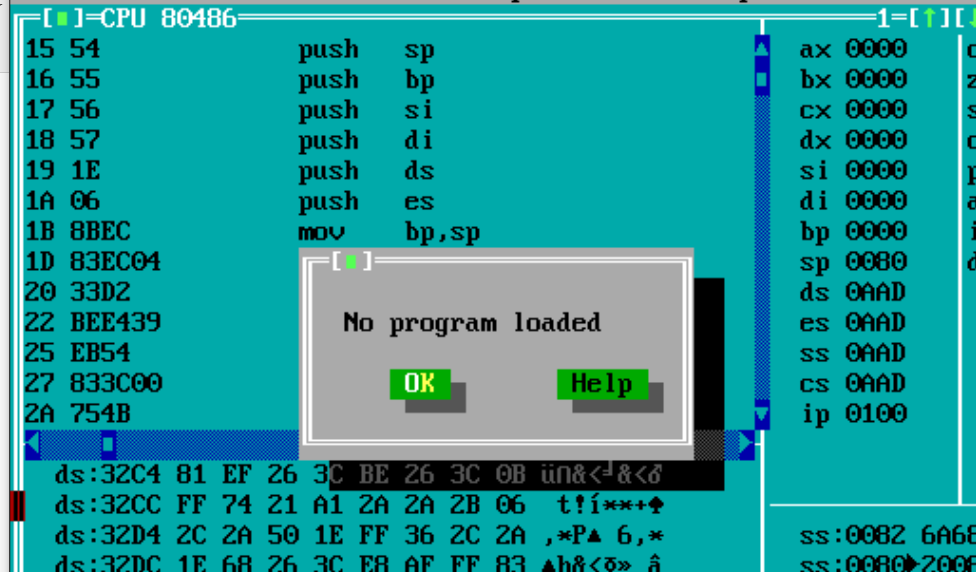


图9.寻找内存段

第一步就无法运行。

4.寻找AUTH变量，将其改为1

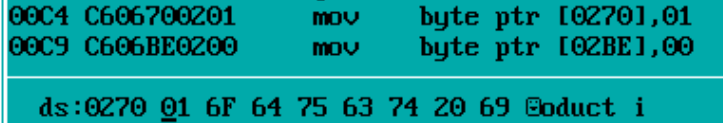


图9.寻找AUTH变量

观察反汇编代码，在密码比对的后一句有一个将1送入某内存地址的语句，推知这一内存地址存放的AUTH变量，则将其改为1，然后运行程序，跳过输入账号密码，可直接进入管理员系统，获取进货价。

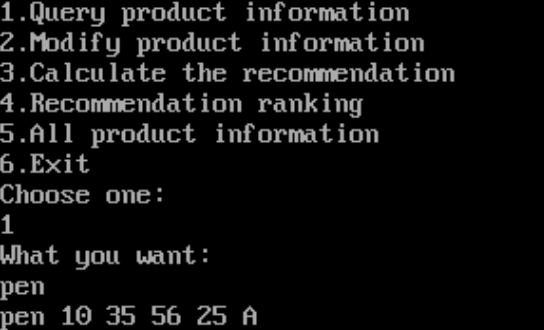


图10.获取进货价

### 4.6 总结与体会

本次实验问题主要出在反跟踪上，在进行反跟踪时，由于中断矢量表被修改，所以不好通过单步调试运行来进行破解。由于本次实验大家的程序相通之处较多，所以可以通过修改AUTH变量的值来达到破解的目的，但是面对一个完全陌生的程序该如何破解呢？那就得费心思去通过反汇编语言寻找程序的内在逻辑，这是一个很耗费时间的辛苦活。通过这次实验也让我知道了程序如何反追踪，短短几行代码就可以让想破解的人耗费大量功夫，在这次实验中，加密这一步我弄得很简单，其实加密也可以复杂一点，比如用调用CMOS获取硬件时间和密码进行相与来进行加密，这样即使别人找到数据段，也无法轻易进行还原。本次实验让我对中断矢量表有了一个更深入的认识。

# 实验五 WIN32编程。

**实验名称： WIN32编程**

**实验地点： 南一楼804室**

**指导教师： 张勇**

**专业班级：计算机科学与技术201706班**

**学 号： U2015414556 姓 名： 梁一飞**

**同组学生： 无 报告日期： 2019年 5 月 10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： 纸质版再签名

日期：2017.3.17

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：

**实验目的和要求**

熟悉WIN32程序的设计和调试方法；

熟悉宏汇编语言中INVOKE、结构变量、简化段定义等功能；

进一步理解机器语言、汇编语言、高级语言之间以及实方式、保护方式之间的一些关系。

## 实验内容

编写一个基于窗口的WIN32程序，实现**网店商品信息管理程序**的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5.1 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。

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）。



图5.2 商品信息显示示意图

**上述任务中，需要解决的问题的提示：**

1. 安装MASM32软件包，观察MASM32软件包目录结构和环境配置。

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。

3. 观察TD32与16位TD的异同。

4. 调试WIN32程序与16位段程序的主要差异是什么？

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，可以参考文献3。

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“？”或其他符号代替）。请通过修改偏移地址来改变观察的区间，记录此现象。

7. 编写和处理简单资源脚本，装入菜单，观察效果。

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。

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。

10. 观察结构变量的推荐度等字段的偏移，体会结构变量优点。

11. 观察简化段的效果。

12. 观察Invoke语句翻译成机器码后的特点，记录参数压栈顺序。单步跟踪到调用系统API函数的位置，观察相关代码的特点。

13. 本次实验需要突出WIN32的框架、调试方法、与16位段程序的区别等几个内容。

**参考资料：**

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

[2] 汇编语言教学网站-》资料下载-》案例-》win32程序、编译和连接

其中的操作说明，给出了几种编译和链接生成执行程序的方法。

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

其中的操作说明，给出了几种编译和链接生成执行程序的方法。

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

## 实验过程

### 5.1 实验步骤

1.安装masm32

2.编写程序

3.打开cmd

4.输入d:

输入cd win32sample

5.为了方便编译，我改写了bat文件，改写如下：

@echo off

set Masm32Dir=d:\Masm32

set include=%Masm32Dir%\Include;%include%

set lib=%Masm32Dir%\lib;%lib%

set path=%Masm32Dir%\Bin;%Masm32Dir%;%PATH%

set Masm32Dir=

echo on

ml /c /coff /Zi demo.asm

link /subsystem:windows /debug /debugtype:cv demo.obj menu.res

6.直接输入begin.bat，即可自动编译程序文件。

7.运行观察结果。

### 5.2 源程序

**菜单：**

#define IDM\_FILE\_EXIT 10001

#define IDM\_HELP\_ABOUT 10101

#define ID\_ACTION\_RECOMMODATION 10011

#define ID\_ACTION\_LIST 10012

MYMENU MENU

BEGIN

POPUP "&File"

BEGIN

MENUITEM "&Exit", IDM\_FILE\_EXIT

END

POPUP "&Action"

BEGIN

MENUITEM "&Recommodation", ID\_ACTION\_RECOMMODATION

MENUITEM "&List", ID\_ACTION\_LIST

END

POPUP "&Help"

BEGIN

MENUITEM "&About", IDM\_HELP\_ABOUT

END

END

**主程序：**

.386

.model flat,stdcall

option casemap:none

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

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

Display proto :DWORD

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

good struct

myname db 10 dup(0)

discount db 0

buyingprice dw 0

sellingprice dw 0

total dw 0

sale dw 0

recommodation dw 0

good ends

.data

I db 0

ClassName db 'TryWinClass',0

AppName db 'My First Window',0

MenuName db 'MyMenu',0

DlgName db 'MyDialog',0

AboutMsg db 'I am student LYF',0

hInstance dd 0

CommandLine dd 0

buf good <>

good <'PEN',10,35,56,70,25,?>

good <'NOTE',9,12,30,25,5,?>

good <'BOOK',8,15,20,30,2,?>

good <'EARASER',9,2,3,80,60,?>

good <'CARD',9,40,50,30,10,?>

good 3 dup(<>)

msg\_myname db '商品名称',0

msg\_discount db '折扣',0

msg\_buyingprice db '进货价',0

msg\_sellingprice db '销售价',0

msg\_total db '进货总数',0

msg\_sale db '已售数量',0

msg\_recommodation db '推荐度',0

discount1 db 2,'10'

buyingprice1 db 2,'35'

sellingprice1 db 2,'56'

total db 2,'70'

sale1 db 2,'25'

discount2 db 1,'9'

buyingprice2 db 2,'12'

sellingprice2 db 2,'30'

tota2 db 2,'25'

sale2 db 1,'5'

discount3 db 1,'8'

buyingprice3 db 2,'15'

sellingprice3 db 2,'20'

tota3 db 2,'30'

sale3 db 1,'2'

discount4 db 1,'9'

buyingprice4 db 1,'2'

sellingprice4 db 1,'3'

tota4 db 2,'80'

sale4 db 2,'60'

discount5 db 1,'9'

buyingprice5 db 2,'40'

sellingprice5 db 2,'50'

tota5 db 2,'30'

sale5 db 2,'10'

re1 db 10 dup(' ')

re2 db 10 dup(' ')

re3 db 10 dup(' ')

re4 db 10 dup(' ')

re5 db 10 dup(' ')

menuItem db 0 ;当前菜单状态, 1=处于list, 0=Clear

.stack

.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

LOCAL ps:PAINTSTRUCT

.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 == ID\_ACTION\_RECOMMODATION

MOV ESI,OFFSET buf+21

call CACUL

MOV EDI,OFFSET buf+21

MOV ESI,OFFSET re1

call F2T10

.ELSEIF wParam == ID\_ACTION\_LIST

mov menuItem, 1

invoke InvalidateRect,hWnd,0,1 ;擦除整个客户区

invoke UpdateWindow, hWnd

.ELSEIF wParam == IDM\_HELP\_ABOUT

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

.ENDIF

.ELSEIF uMsg == WM\_PAINT

invoke BeginPaint,hWnd, addr ps

mov hdc,eax

.IF menuItem == 1

invoke Display,hdc

.ENDIF

invoke EndPaint,hWnd,addr ps

.ELSE

invoke DefWindowProc,hWnd,uMsg,wParam,lParam

ret

.ENDIF

xor eax,eax

ret

WndProc endp

CACUL PROC

PUSH AX

PUSH BX

PUSH CX

PUSH DX

ADD ESI,-21

MOV I,5

LOPA5:ADD I,-1

JS CAUEND

ADD ESI,21

MOV AX,WORD PTR [ESI+11]

ADD AX,10

MOV BX,0080H

MUL BX

MOV BX,WORD PTR [ESI+13]

DIV BX

MOV BX,000AH

MUL BX

MOV BL,[ESI+10]

MOV BH,00H

DIV BX

MOV CX,AX

MOV AX,WORD PTR [ESI+17]

MOV BX,0040H

MUL BX

MOV BX,WORD PTR [ESI+15]

DIV BX

ADD AX,CX

MOV [ESI+19],AX

MOV AX,0

MOV BX,0

MOV CX,0

JMP LOPA5

CAUEND: POP DX

POP CX

POP BX

POP AX

RET

CACUL ENDP

RADIX\_S PROC NEAR STDCALL USES EBX EDX ESI,LpResult,Radix:DWORD,NUM:DWORD

LOCAL COUNT:WORD

MOV COUNT,0

MOV EAX,NUM

MOV ESI,LpResult

mov EBX,Radix

LOP1:MOV EDX,0

DIV EBX

PUSH DX

INC COUNT

OR EAX,EAX

JNZ LOP1

LOP2:POP AX

CMP AL,10

JB L1

ADD AL,7

L1: ADD AL,30H

MOV [ESI],AL

INC ESI

DEC COUNT

JNZ LOP2

MOV AX,SI

RET

RADIX\_S ENDP

F2T10 PROC

PUSH EAX

PUSH EBX

SUB EDI,21

SUB ESI,10

MOV I,5

LOPA3:ADD I,-1

JS FEND

ADD EDI,21

ADD ESI,10

MOV EAX,[EDI+19]

invoke RADIX\_S,ESI,10,EAX

JMP LOPA3

FEND:

POP EBX

POP EAX

RET

F2T10 ENDP

Display proc hdc:HDC

XX equ 10

YY equ 10

XX\_GAP equ 100

YY\_GAP equ 30

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

invoke TextOut,hdc,XX+1\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_discount,4

invoke TextOut,hdc,XX+2\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_buyingprice,6

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

invoke TextOut,hdc,XX+4\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_total,8

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

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

;;

invoke TextOut,hdc,XX+0\*XX\_GAP,YY+1\*YY\_GAP,offset buf[1\*21].myname,3

invoke TextOut,hdc,XX+1\*XX\_GAP,YY+1\*YY\_GAP,offset discount1+1,discount1

invoke TextOut,hdc,XX+2\*XX\_GAP,YY+1\*YY\_GAP,offset buyingprice1+1,buyingprice1

invoke TextOut,hdc,XX+3\*XX\_GAP,YY+1\*YY\_GAP,offset sellingprice1+1,sellingprice1

invoke TextOut,hdc,XX+4\*XX\_GAP,YY+1\*YY\_GAP,offset total+1,total

invoke TextOut,hdc,XX+5\*XX\_GAP,YY+1\*YY\_GAP,offset sale1+1,sale1

invoke TextOut,hdc,XX+6\*XX\_GAP,YY+1\*YY\_GAP,offset re1,3

invoke TextOut,hdc,XX+0\*XX\_GAP,YY+2\*YY\_GAP,offset buf[2\*21].myname,4

invoke TextOut,hdc,XX+1\*XX\_GAP,YY+2\*YY\_GAP,offset discount2+1,discount2

invoke TextOut,hdc,XX+2\*XX\_GAP,YY+2\*YY\_GAP,offset buyingprice2+1,buyingprice2

invoke TextOut,hdc,XX+3\*XX\_GAP,YY+2\*YY\_GAP,offset sellingprice2+1,sellingprice2

invoke TextOut,hdc,XX+4\*XX\_GAP,YY+2\*YY\_GAP,offset tota2+1,tota2

invoke TextOut,hdc,XX+5\*XX\_GAP,YY+2\*YY\_GAP,offset sale2+1,sale2

invoke TextOut,hdc,XX+6\*XX\_GAP,YY+2\*YY\_GAP,offset re2,3

invoke TextOut,hdc,XX+0\*XX\_GAP,YY+3\*YY\_GAP,offset buf[3\*21].myname,4

invoke TextOut,hdc,XX+1\*XX\_GAP,YY+3\*YY\_GAP,offset discount3+1,discount3

invoke TextOut,hdc,XX+2\*XX\_GAP,YY+3\*YY\_GAP,offset buyingprice3+1,buyingprice3

invoke TextOut,hdc,XX+3\*XX\_GAP,YY+3\*YY\_GAP,offset sellingprice3+1,sellingprice3

invoke TextOut,hdc,XX+4\*XX\_GAP,YY+3\*YY\_GAP,offset tota3+1,tota3

invoke TextOut,hdc,XX+5\*XX\_GAP,YY+3\*YY\_GAP,offset sale3+1,sale3

invoke TextOut,hdc,XX+6\*XX\_GAP,YY+3\*YY\_GAP,offset re3,3

invoke TextOut,hdc,XX+0\*XX\_GAP,YY+4\*YY\_GAP,offset buf[4\*21].myname,7

invoke TextOut,hdc,XX+1\*XX\_GAP,YY+4\*YY\_GAP,offset discount4+1,discount4

invoke TextOut,hdc,XX+2\*XX\_GAP,YY+4\*YY\_GAP,offset buyingprice4+1,buyingprice4

invoke TextOut,hdc,XX+3\*XX\_GAP,YY+4\*YY\_GAP,offset sellingprice4+1,sellingprice4

invoke TextOut,hdc,XX+4\*XX\_GAP,YY+4\*YY\_GAP,offset tota4+1,tota4

invoke TextOut,hdc,XX+5\*XX\_GAP,YY+4\*YY\_GAP,offset sale4+1,sale4

invoke TextOut,hdc,XX+6\*XX\_GAP,YY+4\*YY\_GAP,offset re4,3

invoke TextOut,hdc,XX+0\*XX\_GAP,YY+5\*YY\_GAP,offset buf[5\*21].myname,4

invoke TextOut,hdc,XX+1\*XX\_GAP,YY+5\*YY\_GAP,offset discount5+1,discount5

invoke TextOut,hdc,XX+2\*XX\_GAP,YY+5\*YY\_GAP,offset buyingprice5+1,buyingprice5

invoke TextOut,hdc,XX+3\*XX\_GAP,YY+5\*YY\_GAP,offset sellingprice5+1,sellingprice5

invoke TextOut,hdc,XX+4\*XX\_GAP,YY+5\*YY\_GAP,offset tota5+1,tota5

invoke TextOut,hdc,XX+5\*XX\_GAP,YY+5\*YY\_GAP,offset sale5+1,sale5

invoke TextOut,hdc,XX+6\*XX\_GAP,YY+5\*YY\_GAP,offset re5,3

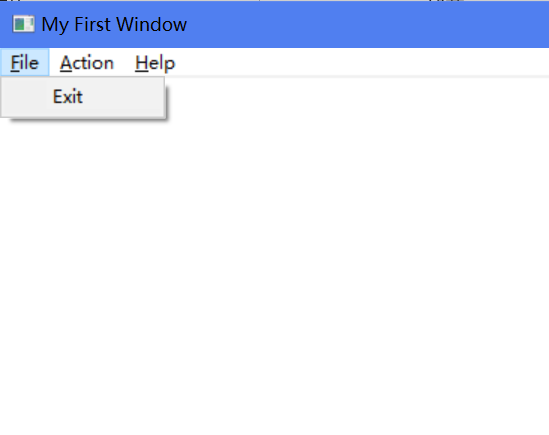
ret

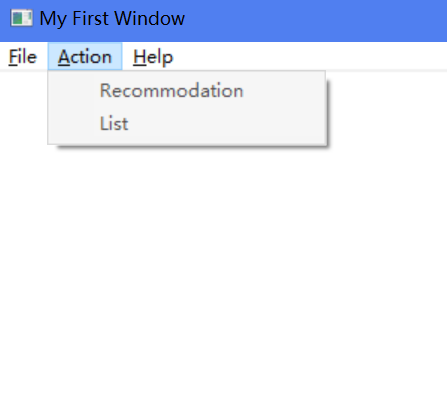
Display endp

end Start

### 5.3 实验记录与分析

菜单界面如图所示：





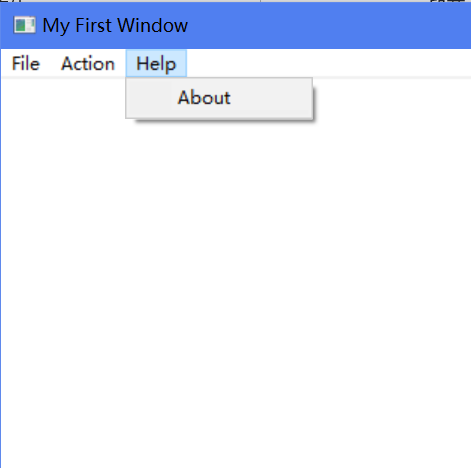


图1.菜单界面图

点击Help-about，观察结果：

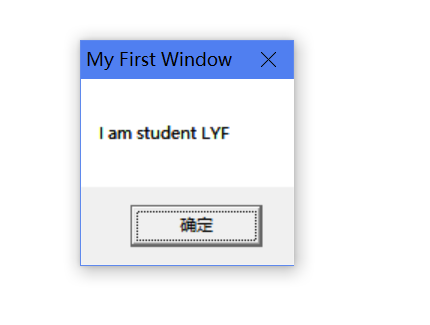


图2.Help-about

点击Action-List，观察结果：

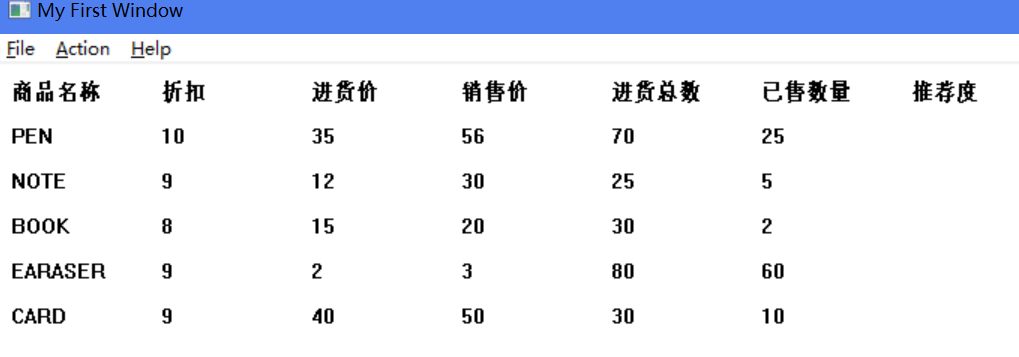


图3.Action-List

点击Action-Recommodation，然后再点击Action-List，观察结果：

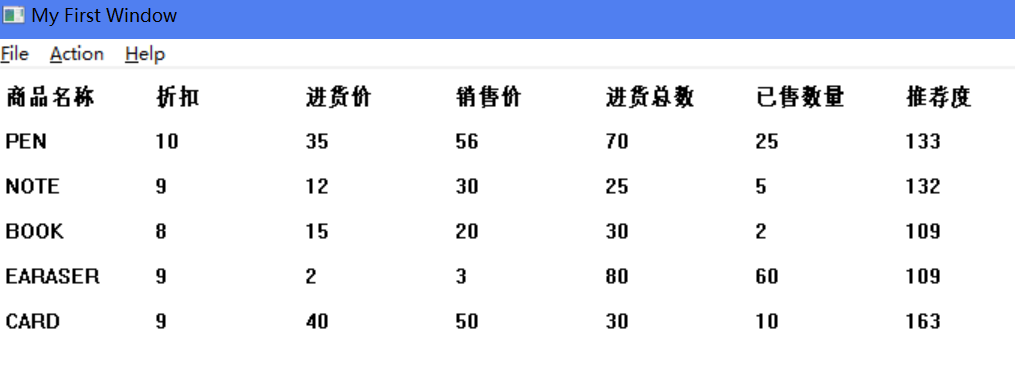


图4.Action-Recommodation

可见，结果是正确的。

### 5.4 具体函数实现

本次实验关键函数主要是两个，1.计算推荐度 2.将二进制转换为十进制

计算推荐度我用的函数是之前实验用到的函数CACUL，实现方法与前面几次实验无大差别，主要就是变量定义变成了结构体，不过循环方法还是与之前一样。

将二进制改为十进制，我参照课本上的RADIX\_S,按照自己的程序进行了修改，计算完后，为了方便输出，额外定义了5个变量储存计算后的推荐度。

### 5.5 总结与体会

本次实验问题主要出现在将二进制改为十进制这一步骤上，一开始我用的是之前实验用到的程序RADIX，可是每次一运行到计算推荐度就会崩溃，经过调试，系统显示是MOV [ESI],AX这一步访问权限受限，但是在此之前我也进行过类似操作，都无问题，故我推测是在循环上出了问题，于是参照课本上的RADIX\_S，将用CX控制循环改成了用一个变量控制循环，问题得到了解决。另一个是通过这次实验，我发现WIN32编程中SI寄存器无法使用，改为ESI后问题得到了解决。