目录

汇 编 语 言 程 序 设 计 实 验 报 告

1 实验目的与要求

(1) 掌握子程序设计的方法与技巧，熟悉子程序的参数传递方法和调用原理。
(2) 掌握宏指令、模块化程序的设计方法。
(3) 掌握较大规模程序的合作开发与调试方法。
(4) 掌握汇编语言程序与 C 语言程序混合编程的方法。
(5) 了解 C 编译器的基本优化方法。
(6) 了解 C 语言编译器的命名方法，主、子程序之间参数传递的机制。

2 实验内容

任务 1 宏与子程序设计

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）。

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。

（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（或网上）的进制转换程序：十进制转二进制的子程序 F10T2 和二进制转十进制的子程序 F2T10。

任务 2：在 C 语言程序中调用汇编语言实现的函数

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

3 实验过程

3.1 任务 1

3.1.1 设计思想及存储单元分配

1) 设计思想

a) 总体模块分配：

1. 主体程序（包括判断输入的姓名和密码是否正确，程序退出，功能菜单的显示）

2. 菜单功能 1，查询商品信息。主要内容为判断输入的商品名称是否正确与输出商品信息。

3. 菜单功能 2，修改商品信息。主要内容为修改输入商品的折扣，进货价等数字。

4. 菜单功能 3，计算推荐度。即按照公式计算推荐度，存到数据段对应的位置。

5. 菜单功能 4，推荐度排名。按照推荐度从高到底给每一个商品一个排名信息。

6. 菜单功能 5，输出商店所有信息。按照排名从高到底输出所有商品的所有信息。

b) 共用子程序 API 说明以及共用变量说明

子程序:（都为 NEAR 型）

1. TRANS10: 功能为将 2 进制转换为 10 进制，入口参数 AX 为想转换的数字，出口参数为将转换后的字符串存入 TEMP 中。

2. OUTNAME: 功能为显示用户输入的字符串，入口参数为 AX

3. CALCREFER: 功能为计算推荐度，该子程序需要用到商品的首地址（即 GA1（第一个商品）的偏移值）并修改数据段中的值，所以需要将 GA1 进行 public。

4. RANKLEVEL：功能为将商品按照推荐度进行排序，同样需要将 GA1 进行 public，因为用到了数据段中对应的值。

5. ALLINFO：功能为将商品按照排序输出所有商品信息，需要将 GA1 进行 public，输出数据段中的信息以及排名信息。

共用变量：

1. GA1: 存储第一个商品的信息，BYTE 类型。

2. OUT1、OUT2、OUT3、OUT4、OUT5、OUT6、OUT7：存储提示信息，用于输出提示信息，BYTE 类型。

3. TEMPADDR: 存储商品的首地址，WORD 类型

4. TEMP: 用于存储十进制转换之后的数字，BYTE 类型

c) 我的模块设计思想

1. 定义宏 PRINTS,SCANS,CRLF，用于 9 号调用输出字符串，10 号调用输入字符串，输出换行。

2. 输出菜单信息，接受用户输入并跳转。判断是否登录，若登录跳过 2-5 项提示信息的

输出，只输出 1 和 6。接收用户输入的数字，从 1 开始顺序逐个判断，直到匹配到 1-6 中的一个数字，跳转到相应的模块。

3. 查询商品信息，并输出信息。

将实验 1 中的寻找货物匹配的部分改写为子程序 FINDITEM。设计思路与实验一相同，双重循环，外层循环每个货物，内层循环一个货物的名字，如果输入匹配完毕且标准下一位不为 0，则匹配成功。

编写 OUTNAME 子程序，输出商品名称。该函数的作用是碰到 0 即终止输出。利用循环，一个一个字符的输出，每次输出时和 0 比较，若不为 0 则输出，碰到 0 则终止子程序。

编写二进制转换为十进制子程序 TRANS10，并定义 TENOUT 宏，调用自尘能够徐 TEANS10,输出十几进制字符串。子程序思想是，用被转换数除以 10，余数入栈，直到被除数到 0，余数一次次出栈，转换成 ASCII 码。

查询到商品信息之后，用 OUTNAME 输出名称，用 TENOUT 依次输出销售价，进货总数，已售数量，推荐度对应的十进制字符串。

4. 修改商品信息。

编写十进制转二进制子程序 BITRANS。程序设计思想是从最高位开始，将当前位乘以 10 加上低位（下一位），循环到加上最后一位。

编写宏 INEDIT，改变原来的输入值，输入十进制字符串，存入数据段对应的位置。

编写子程序 EDITP。逐一调用宏 INEDIT 修改对应的值。

2） 存储单元分配

AX：存入数据。

BX：存入数据，作为循环指示器。

CX：循环计数器。

DI、SI、BP：暂时存入当前地址。

AL、AH、BL、BH、CL、CH、DL、DH：暂时存入数据

3.1.2 流程图

1) 查询及信息输出模块子程序流程图与总流程图

FINDITEM 子程序流程图如下：

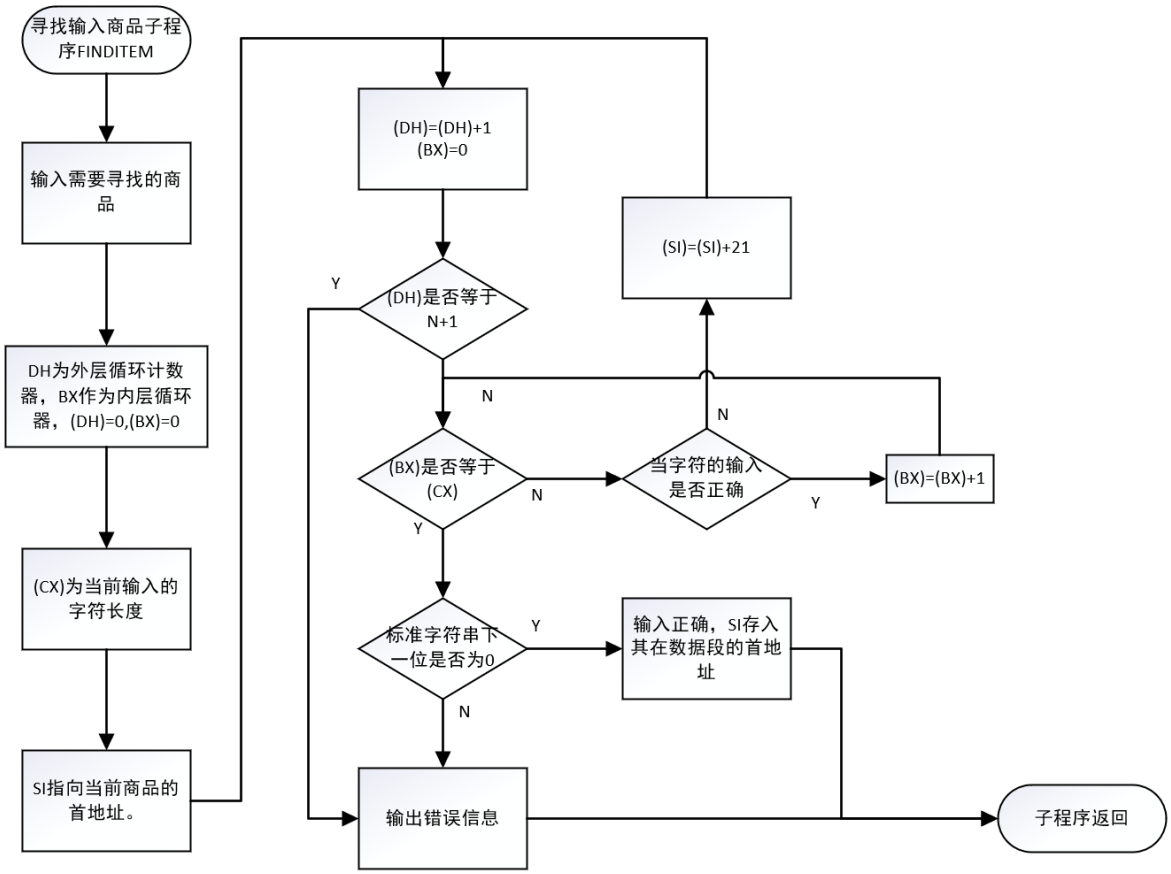


图 3-1 FINDITEM 子程序流程图

OUTNAME 子程序流程图如下：

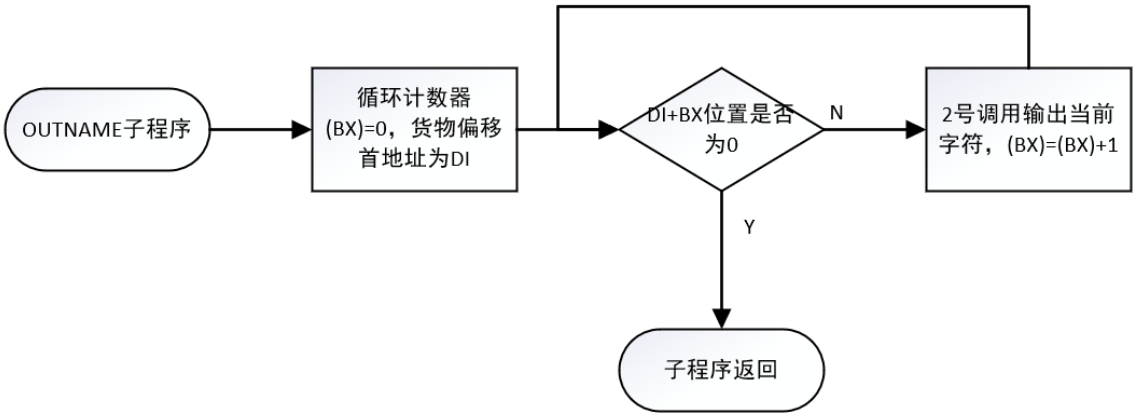


图 3-2 OUTNAME 子程序流程图

二进制转换为十进制子程序 TRANS10 流程图如下：

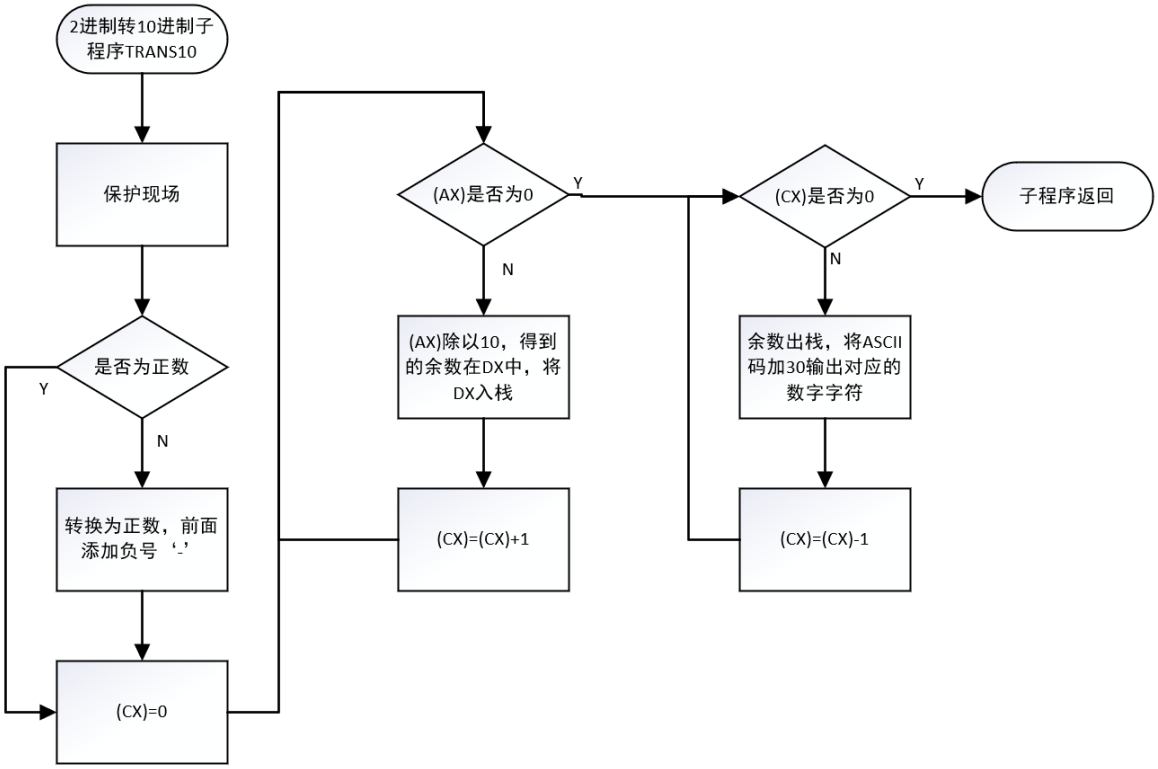


图 3-3 TRANS10 子程序流程图

查询输出模块总流程图如下：

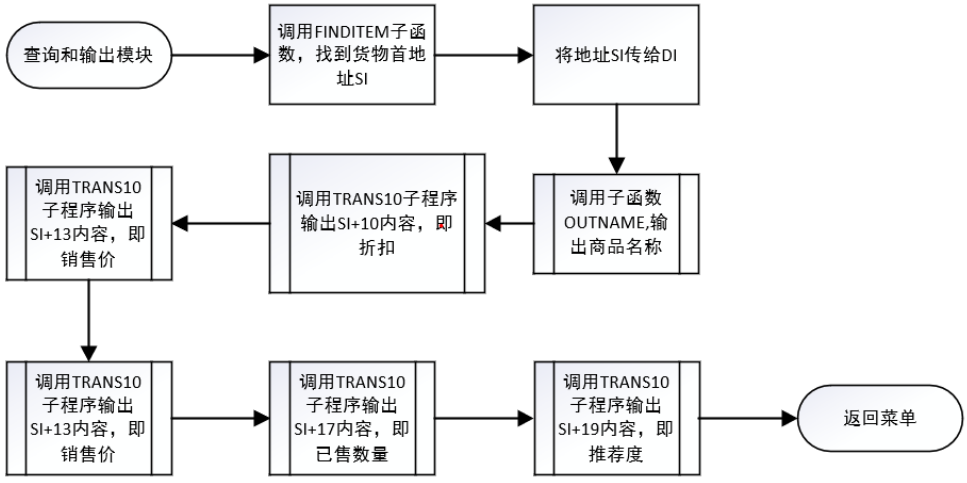


图 3-4 查询输出模块总流程图

2） 修改商品信息模块子程序流程图与总流程图如下：

二进制转换为十进制子程序 BITRANS 流程图如下：

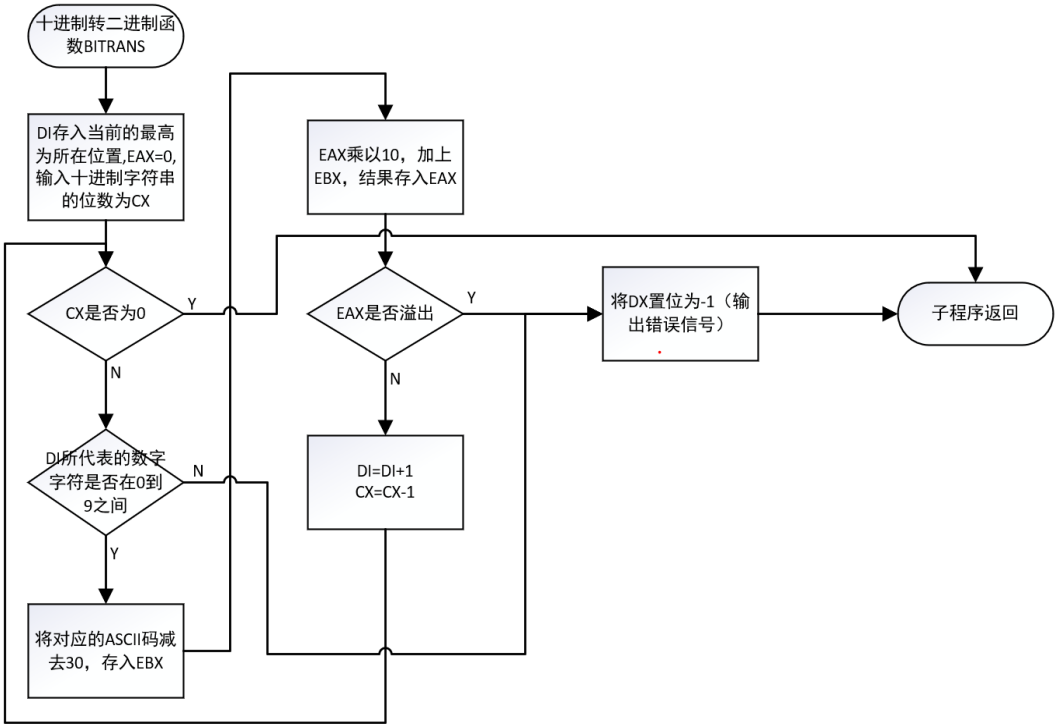


图 3-5 BITRANS 子程序流程图如下

修改商品信息总流程图如下：

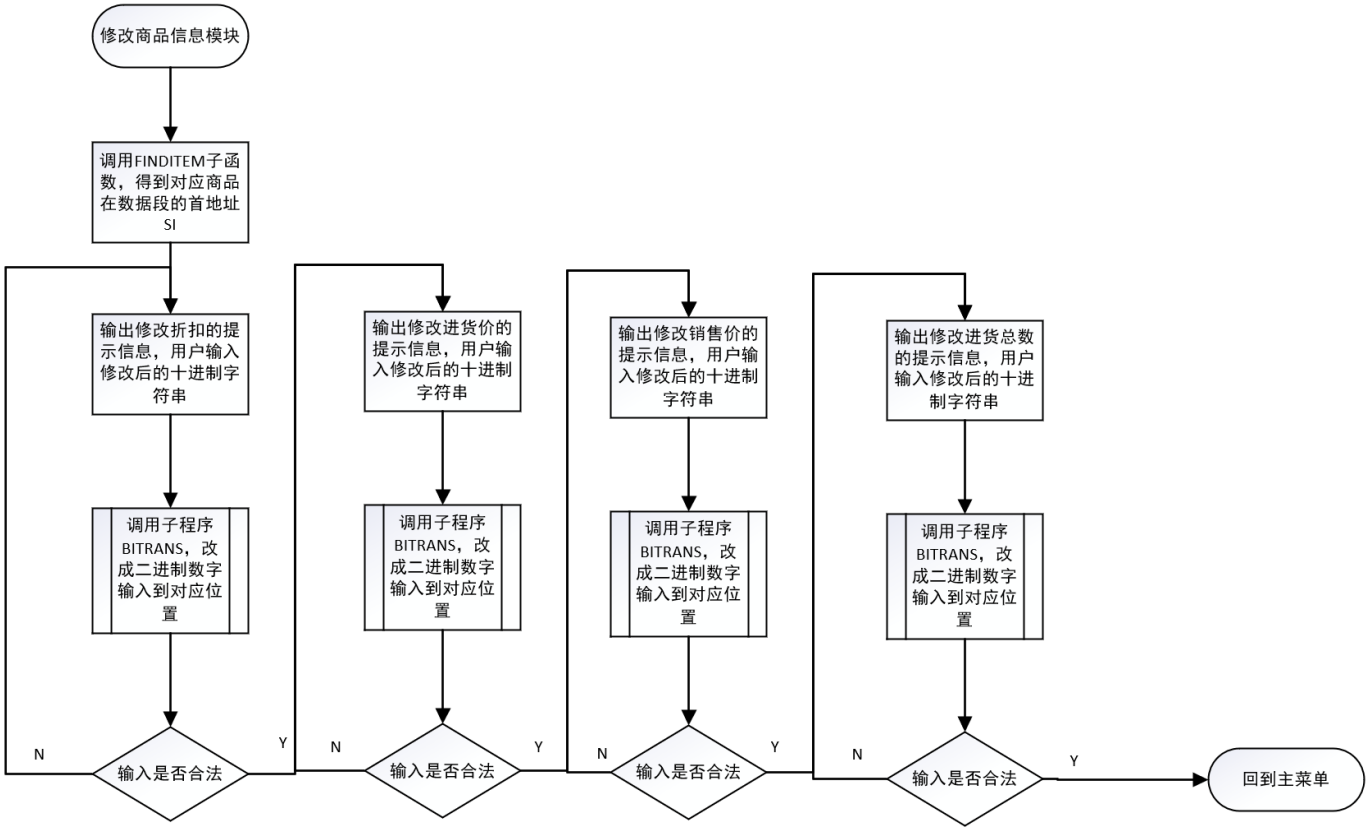


图 3-6 修改商品信息总流程图

3) 程序总流程图如下：

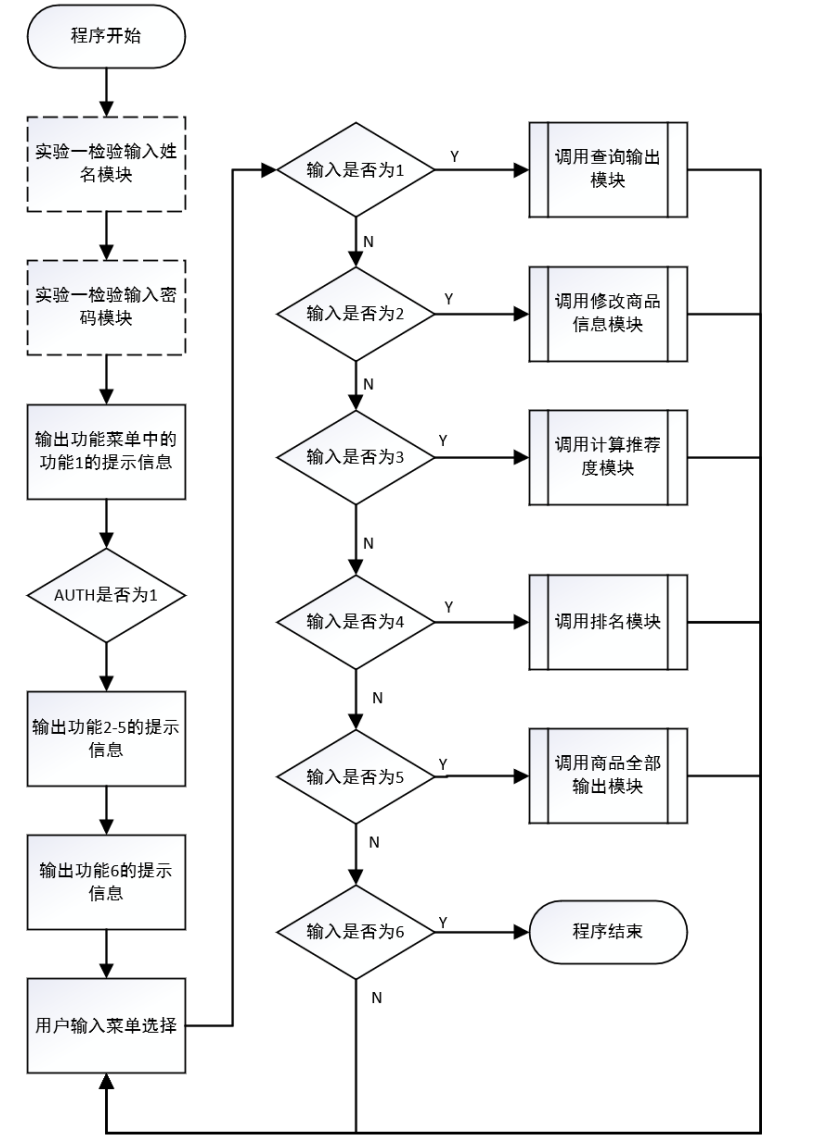


图 3-7 总体流程图

3.1.3 源程序

```
;Lab3-part1
;The main program and function1 and function2

.386

PUBLIC OUTNAME, TRANS10, GA1, TEMP, OUT1, OUT2, OUT3, OUT4, OUT5, OUT6, OUT7, TEMPADDR
EXTERN CALCREFER : NEAR, RANKLEVEL : NEAR, ALLINFO : NEAR

PRINTS MACRO PR ;9 号调用输出字符串
 PUSH AX
 PUSH DX
```

```
 LEA DX,PR
 MOV AH,9H
 INT 21H
 POP DX
 POP AX
ENDM

SCANS MACRO SC ;10 号调用输入字符串
 PUSH AX
 PUSH DX
 LEA DX,SC
 MOV AH,10
 INT 21H
 POP DX
 POP AX
ENDM

CRLF MACRO ;换行
 PUSH DX
 PUSH AX
 MOV DL,0AH
 MOV AH,2H
 INT 21H
 POP AX
 POP DX
ENDM

TENOUT MACRO NUM ;输出十进制数字
 PUSH AX
 MOV AX, WORD PTR [NUM]
 CALL TRANS10
 PRINTS TEMP
 POP AX
ENDM

TENOUT_DIS MACRO NUM ;输出十进制数字
 PUSH AX
 MOVZX AX, BYTE PTR [NUM]
 CALL TRANS10
 PRINTS TEMP
 POP AX
ENDM

 INEDIT MACRO NUM, C1, C2 ;更改原来的值
 SCANS IN_EDIT ;输入改变的值
 CRLF
 LEA DI, IN_EDIT+2
 MOVZX CX, IN_EDIT+1
 CMP CX, 0
 JZ C2 ;输入回车则直接跳过修改
 CALL BITRANS ;10 进制转 2 进制
 CMP DI, -1
 JZ C1
 MOV AX, TEMP_NUM
 MOV WORD PTR [NUM],AX
ENDM
INEDIT_BIT MACRO NUM, C1, C2 ;更改原来的值
```

```
 SCANS IN_EDIT ;输入改变的值
 CRLF
 LEA DI, IN_EDIT+2
 MOVZX CX, IN_EDIT+1
 CMP CX, 0
 JZ C2 ;输入回车则直接跳过修改
 CALL BITRANS ;10 进制转 2 进制
 CMP DI, -1
 JZ C1
 MOV AX, TEMP_NUM
 MOV [NUM], AL
ENDM

STACK SEGMENT USE16 PARA STACK 'STACK'
 DB 500 DUP(0)
STACK ENDS

DATA SEGMENT USE16 PARA PUBLIC 'DATA'
 BNAME DB 'ABC', 7 DUP(0)  ;老板姓名
 BPASS DB 'ABC', 7 DUP(0) ;密码
 N EQU 8 ;商品数量
 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, ? ;进货价，销售价，进货总数，已售数量，推荐度
    GA3 DB 'PENCIL', 4 DUP(0), 9
        DW 50, 30, 25, 5, ?
    GA4 DB 'BAG',7 DUP(0), 10
        DW 50, 80, 60, 20, ?
 GA5 DB 'TEXTBOOK', 2 DUP(0), 8
     DW 40, 80, 50, 20, ?
 GA6 DB 'PAPER', 5 DUP(0), 9
     DW 5, 10, 50, 30, ?
 GA7 DB 'GLUE', 6 DUP(0), 10
        DW 10, 20, 50, 30, ?
 GA8 DB 'CANDY', 5 DUP(0), 9
     DW 2, 4, 60, 40, ?

 AUTH DB 0
 IN_NAME DB 12
          DB ?
          DB 12 DUP(0)

 IN_PWD DB 10
           DB ?
           DB 10 DUP(0)

 IN_ITEM DB 12
          DB ?
               DB 12 DUP(0)

 IN_EDIT DB 10
          DB ?
          DB 10 DUP(0)

 IN_MENU DB 5
          DB ?
```

```
          DB 5 DUP(0)
 TEMP DB 10 DUP(0)

 MENU0 DB 'PLEASE CHOOSE ONE FROM 1-6$'
 MENU1 DB '1=SEARCH THE IMFORMATION OF ITEMS$'
 MENU2 DB '2=EDIT THE IMFORMATION OF ITEM$'
 MENU3 DB '3=CALCULATE THE RECOMENDATION$'
 MENU4 DB '4=RANK THE RECOMENDATION$'
 MENU5 DB '5=OUTPUT THE IMFORMATION OF THE ITEMS$'
 MENU6 DB '6=EXIT$'
 OUT1 DB 'ITEM NAME: $'
 OUT2 DB 'DISCOUNT: $'
 OUT3 DB 'SAILING PRICE: $'
 OUT4 DB 'TOTAL STOCK NUMBERS: $'
 OUT5 DB 'SOLD OUT NUMBERS: $'
 OUT6 DB 'RECOMENDATION LEVEL: $'
 OUT7 DB 'INPUT PRICE: $'

 PUTNAME DB 'PLEASE INPUT THE NAME(INPUT ENTER TO LOOKUP, INPUT Q TO QUIT)$'
 NOTITEM DB 'DO NOT FIND THE ITEM$'
 PUTPASSWORD DB 'PLEASE INPUT THE PASSWORD$'
 ITEMNAME DB 'PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)$'
 LOGFAIL DB 'YOUR USER NAME OR PASSWORD IS WRONG!$'
 LOGINREMIND DB 'IDENTIFICATION GOT!$'
 NOTE DB '->$'
 TEMPADDR DW ?
 TEMP_NUM DW ?
DATA ENDS

CODE SEGMENT USE16 PARA PUBLIC 'CODE'
 ASSUME CS:CODE,DS:DATA,SS:STACK

;寻找输入的商品，将其在数据段的偏移地址存入 TEMPADDR
FINDITEM PROC USES CX SI DX BX
 PRINTS ITEMNAME ;提示输入商品
 CRLF
 SCANS IN_ITEM ;输入商品
 CRLF
 MOV CH, 0
 MOV CL, IN_ITEM+1
 CMP CL, 0
 JZ MENU ;输入回车返回菜单
 MOV SI, OFFSET GA1
 MOV DH, 0
 SUB SI, 21
 MOV BX, 0
CMPA:
 ADD SI, 21
 INC DH;外层循环加一，比较下一个商品
 CMP DH, N+1
 JZ FINDFAIL ;循环完了所有商品以后没找到，输出错误信息
CMPB:
 MOV DL, [SI+BX] ;存入当前要比较的字符
 CMP BYTE PTR [IN_ITEM+BX+2], DL ;比较输入是否正确
 JNZ CMPA ;若不同，比较下一个商品
 INC BX ;内层循环加一，比较下一个字符
 CMP BX, CX ;比较是否全部比完
```

```
 JNZ CMPB ;若没有全部比较完，比较下一个字符
 CMP BYTE PTR [SI+BX], 0 ;判断比较完后下一位是否为 0
 JNZ CMPA ;若不是 0 则为子串，继续比较下一位
 MOV TEMPADDR, SI
 RET
FINDITEM ENDP

;显示商品信息输出名字,入口参数为 DI 存入首地址
OUTNAME PROC USES BX SI DX AX
 MOV BX, 0 ;循环计数器
 MOV SI, TEMPADDR
LOPS:
 MOV DL, [SI+BX]
 CMP DL, 0
 JZ LOPE
 MOV AH, 2H
 INT 21H
 INC BX
 JMP LOPS
LOPE:
 RET
OUTNAME ENDP

;二进制转换为 10 进制
;入口参数 AX 为想传递的数字
;出口参数为将转换后的字符串存入 TEMP 中
TRANS10 PROC USES BX CX DX SI
 MOV BX, 10 ;除数
 MOV CX, 0 ;计数器
 LEA SI, TEMP
 OR AX, AX ;正数即转换
 JNS LOPX
 NEG AX ;负数转换为正数
 MOV BYTE PTR [SI], '-' ;存入负号
 INC SI
LOPX:
 XOR DX, DX ;清零
 DIV BX
 PUSH DX
 INC CX
 OR AX, AX ;AX 为 0 的时候跳出循环
 JNZ LOPX
LOPY:
 POP AX ;余数出栈
 ADD AL, 30H
 MOV [SI], AL
 INC SI
 LOOP LOPY
 MOV BYTE PTR [SI], '$' ;结束符
 RET
TRANS10 ENDP

;10 进制转 2 进制
;入口：DI 表示首地址,CX 表示串长
;出口：AX 存放结果，DI 为-1 表示错误
BITRANS PROC USES EBX EAX
 MOV EAX, 0
```

```
 MOV BL, [DI]
 CMP BL, '0'
 JZ NOTR ;若输入第一个为0，则输入错误
 DEC DI
GO1:
 INC DI
 MOV BL, [DI]
GO2:
 CMP BL, '0'
 JB NOTR ;小于0输出错误
 CMP BL, '9'
 JA NOTR ;大于0输出错误
 SUB BL, 30H
 MOVZX EBX,BL
 IMUL EAX, 10
 JO NOTR ;溢出则输出错误
 ADD EAX, EBX
 JO NOTR
 JS NOTR
 JC NOTR ;溢出，变成负数，进位都输出错误
 DEC CX
 JNZ GO1
 CMP EAX, 7FFFH ;看转换结果是否超过16位
 JA NOTR ;超过16位则输出错误
RE:
 MOV TEMP_NUM, AX
 RET
NOTR:
 MOV DI, -1
 JMP RE
BITRANS ENDP


;修改信息，入口参数SI，当前位置
EDITP PROC USES SI
 CALL FINDITEM
 MOV SI, TEMPADDR
L1:
 PRINTS OUT2
 TENOUT_DIS SI+10
 PRINTS NOTE
 INEDIT_BIT SI+10, L1, L2
L2:
 PRINTS OUT7
 TENOUT SI+11
 PRINTS NOTE
 INEDIT SI+11, L2, L3
L3:
 PRINTS OUT3
 TENOUT SI+13
 PRINTS NOTE
 INEDIT SI+13, L3, L4
L4:
 PRINTS OUT4
 TENOUT SI+15
 PRINTS NOTE
 INEDIT SI+15, L4, L5
```

```
L5:
 PRINTS OUT5
 TENOUT SI+17
 PRINTS NOTE
 INEDIT SI+17, L5, L6
L6:
 RET
EDITP ENDP

START:
 MOV AX, DATA
 MOV DS, AX
 JMP INNAME

FINDFAIL: ;没找到货物输出错误信息并重新寻找
 PRINTS NOTITEM
 CRLF
 JMP FINDIT

INNAME: ;输入姓名
 CRLF
 PRINTS PUTNAME ;提示输入姓名
 CRLF
 SCANS IN_NAME ;输入姓名
 CRLF
 MOV CL, IN_NAME+1
 MOV CH, 0
 CMP CL, 0
 JZ MENU ;若回车直接进入功能三
 CMP CL, 1
 JZ IFEXIT
 MOV BX, 0 ;循环计数器
 JMP COMPA

IFEXIT: ;判断是否输入 q 退出
 LEA BP, OFFSET IN_NAME
 ADD BP, 2
 CMP DS:BYTE PTR [BP], 'q'
 JZ EXIT
 JMP COMPA

COMPA: ;比较输入姓名是否正确
 MOV DL, [BNAME+BX]
 CMP BYTE PTR [IN_NAME+BX+2], DL
 JNZ LFAIL ;不相同则输出错误提示信息
 INC BX
 CMP BX, CX
 JNZ COMPA
 CMP BYTE PTR [BNAME+BX], 0
 JNZ LFAIL ;若是字串输出错误信息
 JMP PASSWORD

LFAIL: ;登陆未成功输出错误信息
 PRINTS LOGFAIL
 CRLF
 JMP INNAME
PASSWORD: ;进入输入密码模块
```

```
 PRINTS PUTPASSWORD ;提示输入密码
 CRLF
 SCANS IN_PWD ;输入密码
 CRLF
 MOV CL, IN_PWD+1
 MOV CH, 0 ;CX 存密码位数
 MOV BX, 0
 JMP COMPB

COMPB:
 MOV DL, [BPASS+BX]
 CMP BYTE PTR [IN_PWD+BX+2], DL
 JNZ LFAIL ;若当前位比较不相符，就输出错误信息
 INC BX
 CMP BX, CX
 JNZ COMPB ;继续比较下一位
 CMP BYTE PTR [BPASS+BX], 0
 JNZ LFAIL ;若是子串则提示错误信息
 MOV DL, 1
 MOV AUTH, DL ;登陆通过给 AUTH 赋值为 1
 PRINTS LOGINREMIND
 CRLF
 JMP MENU

MENU:
 CRLF
 PRINTS MENU0
 CRLF
 PRINTS MENU1
 CRLF
 CMP AUTH, 0
 JZ MENU06 ;未登录状态只输出 1 和 6
 PRINTS MENU2
 CRLF
 PRINTS MENU3
 CRLF
 PRINTS MENU4
 CRLF
 PRINTS MENU5
 CRLF
MENU06:
 PRINTS MENU6
 CRLF

CHMENU:
 SCANS IN_MENU ;
 MOV CL, IN_MENU+1
 MOV CH, 0
 CMP CL, 0
 JZ MENU ;输入回车回到功能 3-1
 CMP CL, 1
 JZ COMPC

; 菜单跳转程序
COMPC:
 LEA BP,IN_MENU+2
 CMP DS:BYTE PTR [BP], '1'
```

```
 JZ FINDIT
 CMP DS:BYTE PTR [BP], '2'
 JZ EDITIT
 CMP DS:BYTE PTR [BP], '3'
 JZ CALCULATEREFER
 CMP DS:BYTE PTR [BP], '4'
 JZ RANKINGLEVEL
 CMP DS:BYTE PTR [BP], '5'
 JZ SHOWALLINFO
 CMP DS:BYTE PTR [BP], '6'
 JZ EXIT
FINDIT:
 CALL FINDITEM ;找到商品，偏移地址为SI
 PRINTS OUT1
 ; MOV DI,SI
 CALL OUTNAME ;输出商品名称，入口参数为首地址在SI中
 MOV SI, TEMPADDR
 CRLF
 PRINTS OUT2
 TENOUT_DIS SI+10
 CRLF
 PRINTS OUT3
 TENOUT SI+13
 CRLF
 PRINTS OUT4
 TENOUT SI+15
 CRLF
 PRINTS OUT5
 TENOUT SI+17
 CRLF
 PRINTS OUT6
 TENOUT SI+19
 CRLF
 JMP MENU

EDITIT:
 CALL EDITP
 JMP MENU

CALCULATEREFER:
 CALL CALCREFER
 JMP MENU

RANKINGLEVEL:
 CALL RANKLEVEL
 JMP MENU

SHOWALLINFO:
 CALL ALLINFO
 JMP MENU

EXIT:
 MOV AX,4C00H  ; CS:05DFH, CS:0731H
 INT 21H

CODE ENDS
 END START
```

3.1.4 实验步骤

1. 使用编辑程序 EDIT.EXE 录入源程序.

2. 将我所负责的模块进行编译，若有报错，可查阅手册解决报错问题。

3. 将我的模块的.obj 文件与同组同学的.obj 文件连接，若有报错解决报错问题。

4. 程序测试：

1) 自己的模块测试（先将自己写的模块写成一个可正常执行自己模块功能的程序，由于我只负责菜单功能 1 和功能 2，因为推荐度还没计算，故都为 0，且菜单输入功能 3-5 都进入功能 2）

a) 登录状态下查看菜单，是否正常显示 1-6 的提示信息；非登录状态下查看菜单，是否只显示 1 和 6 的提示信息。

b) 菜单功能 1，查询商店中存在的商品，看能否正常输出商品信息；查询商店中不存在的商品，看能否提示错误。

c) 菜单功能 2，输入合法的更改信息（合理的数值），再次调用功能 1，看值是否改变；测试能否输入回车跳过一些值的更改，调用功能 1，看输入回车的地方值是否没变；输入非法更改信息，例如字母，看程序是否会要求重新输入。

2) 测试两人合并后的程序是否正确

a) 运用 td 观察程序是否按照预想顺序进行连接。（包括数据段、代码段）

b) 测试合并之后的程序，主要测试对方编写的功能 3、4、5。具体操作为，先分别调用 3 和 5，然后再调用 5，看输出的推荐度是否正确、排名是否正确。

5. 调用 td，观察 FAR 类型子程序的 RET 机器码与 NEAR 类型子程序的 RET 机器码，比较两者有何不同。

6. 用 td 观察将一个 NEAR 类型的子程序强行使用 FAR 调用（CALL FAR PTR 子程序名），会出现什么情况，反之，对于一个 FAR 类型的子程序强行用 NEAR 调用会怎么样。

7. 将 EXTERN 语句放到.386 之前与.386 之后，编译后用 td 观察，比较两者的不同。

3.1.5 实验记录与分析

1. 汇编源程序和连接同组同学程序的过程没有出现异常。

2. 程序功能测试

1) 自己的模块测试

由于我负责的是主程序和菜单功能 1 和菜单功能 2 所以主要测试菜单显示、菜单功能 1 和菜单功能 2.

a) 菜单显示测试

首先，未登录的情况下显示菜单只有功能 1 和功能 6 的提示，截图如下：

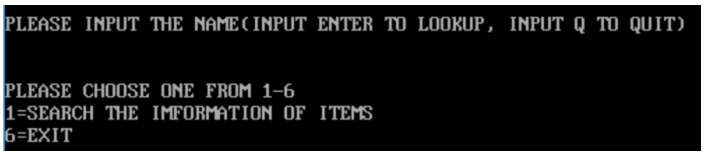


图 3-8 未登录情况下菜单显示

登录状况下，菜单显示功能 1-6 的提示，截图如下：

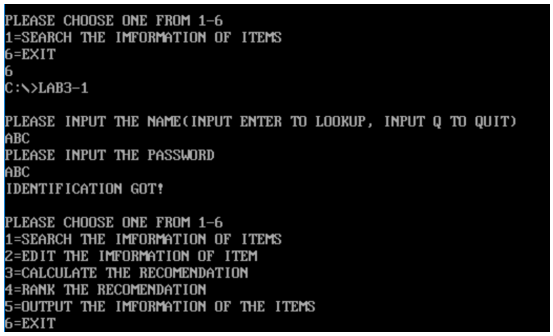


图 3-9 登录情况下菜单显示

b) 菜单功能 1 测试

输入商店中存在的商品 PEN，得到结果如下：

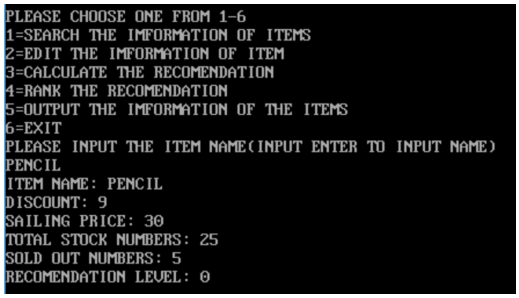


图 3-10 功能 1 商品“PENCIL”的测试样例

输入商店中存在的商品 BOOK，得到结果如下：

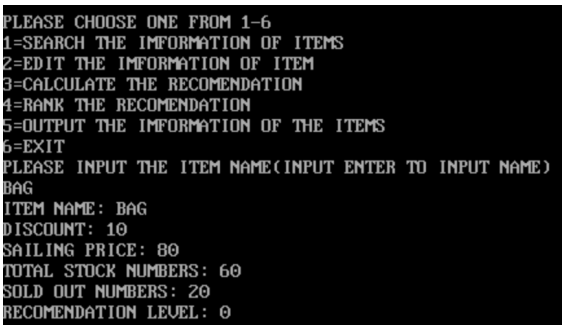


图 3-11 功能 1 商品“BAG”的测试样例

输入商店中不存在的商品 TISSUE，输出错误提示，截图如下：

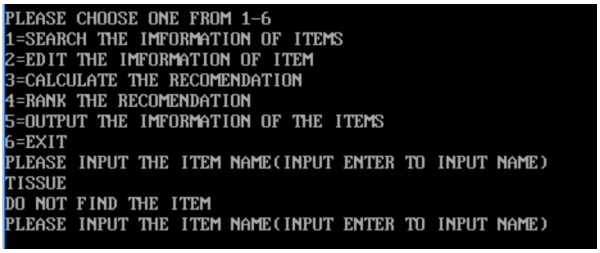


图 3-12 功能 1 输入商店中不存在的商品测试截图

c) 菜单功能 2 测试

i. 输入合法的更改数值（合理的数字），看是否能正常更改并显示更改之后的值。

以 PENCIL 为例，更改每一项的数值，截图如下：

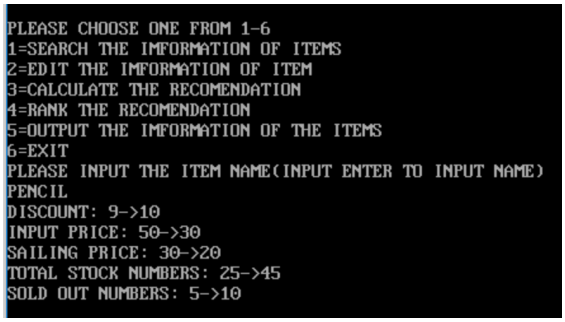


图 3-13 输入合法的更改数值截图

再次调用功能 1，显示更改之后的数值，如下图所示：

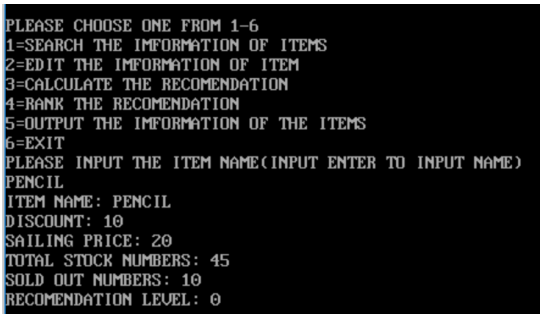


图 3-14 更改之后的数值显示

可以发现，数据段中对应的数值被正常更改，测试结果正确。

ii. 输入空格跳过一些数值的更改，看能否得到正确的结果。

这里以 BOOK 为例，值更改第一项的数值，其余 3 项跳过，截图如下：

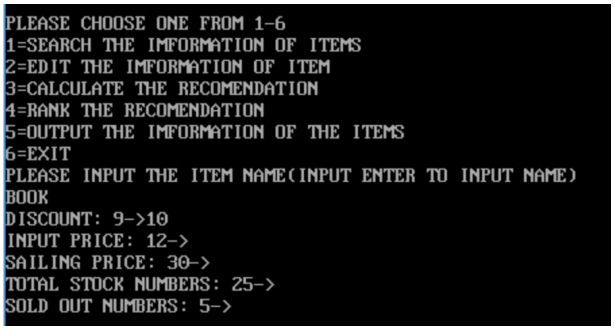


图 3-15 使用空格跳过一些数值的更改

再次调用菜单功能 1，发现第一项数值被正确更改，其余数值没变，截图如下：

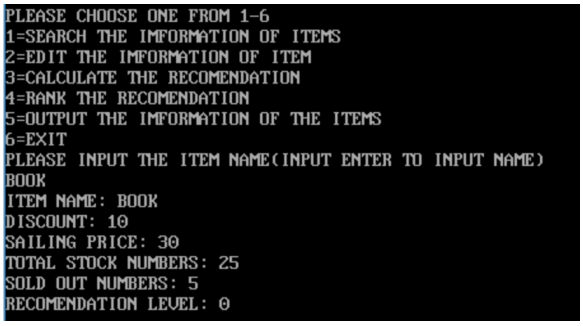
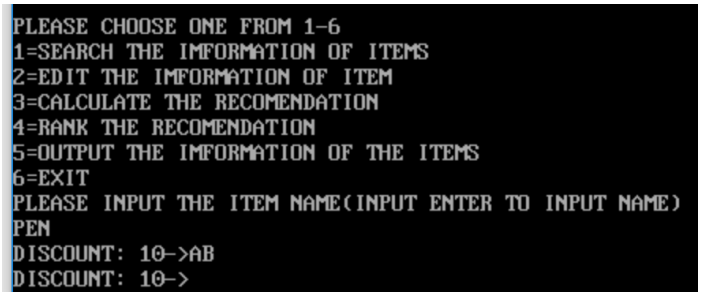


图 3-16 使用空格跳过一些数值的更改

这也就说明使用空格能够正确跳过不需要更改的项，而保留原来的值，被更改的项输出的是新的值。测试结果证明程序功能正确。

iii. 输入非法更改信息，例如字母，看程序是否会要求重新输入

这里以商品 PEN 为例，在第一项修改值内输入大写字母 AB，会要求重新输入第一项的值，截图如下所示：



直到输入正确信息才能修改下一项，截图如下所示：

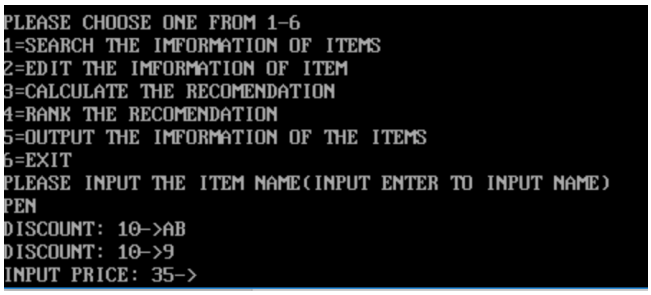


图 3-18 输入合法正确数值截图 1

输入小写字母 ab，也会被要求重新修改，截图如下所示：

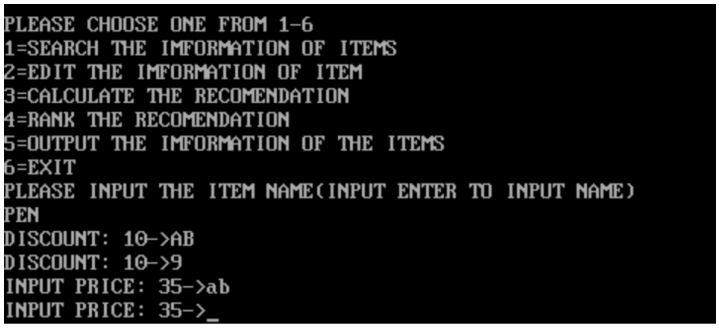


图 3-19 输入非法修改信息截图 2

输入正确信息就能跳转到下一项修改，截图如下所示：

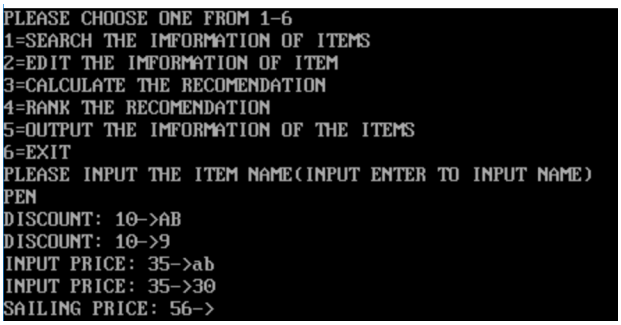


图 3-20 输入合法正确数值截图 2

下面几项的修改全部跳过，调用菜单 1 号功能，输出修改后商品信息，发现商品被修改后的信息全是合法的，截图如下：

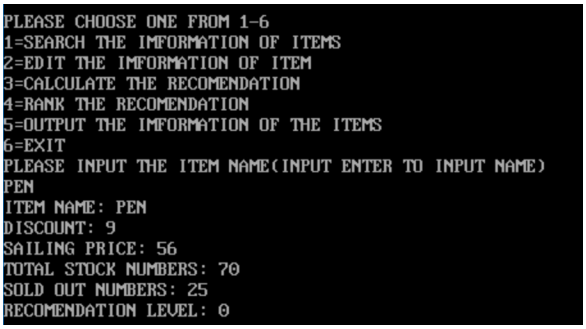


图 3-21 在上述情况下数值修改后的截图

综上，我们可以看出，遇到输入值不合法时（为字母），程序会自动要求重新输入，且更

改过后的值不受非法输入的影响，只变为合法输入的更改值。证明程序功能正确。

2) 两人合并后的程序测试

a) 测试两个程序是否正常连接

i. 用 td 查看代码段是否正常连接

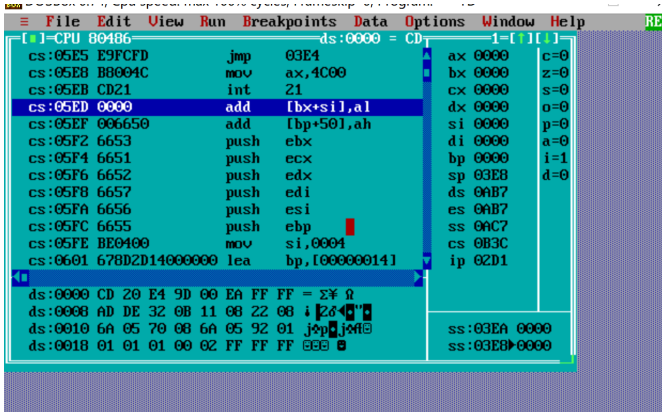


图 3-22 代码段连接截图

在偏移值 05ED 之前为第一个文件代码段最后部分，偏移值 05F2 之后为第二个文件计算推荐度子程序部分。可以看出，程序正常连接，中间插入了两条指令，即偏移值为 05ED 以及偏移值 05EF 的语句，作为连接。

ii. 查看数据段是否正常连接

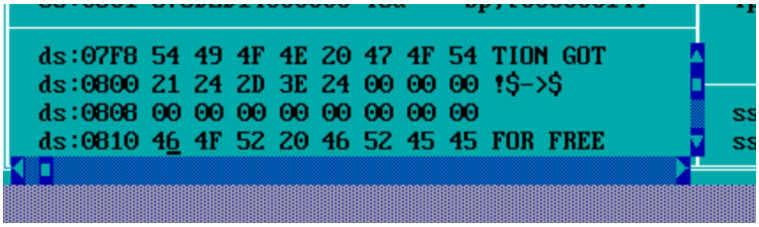


图 3-23 数据段连接截图

在偏移值 0809 之前，为第一个文件的数据段内容（最后一条指令为声明变量 EMP_NUM DW ?）；在偏移值 0810 之后为为第二个文件数据段内容（第一条指令 BUF1 DB 'FOR FREE', 0AH, 0DH, '$'）。可以看出，程序正常连接，中间插入了 7 个 byte 的 0 作为缓冲。

b) 测试对方的模块能否正常运行

调用功能 3，计算推荐度，如下图：

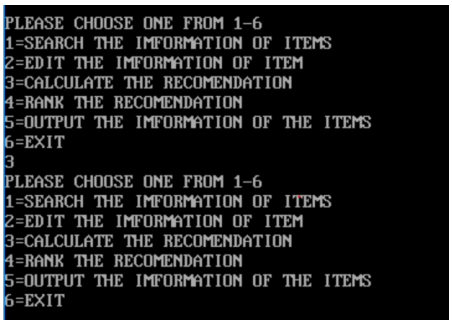


图 3-24 调用功能 3 示意图

首先我们调用功能 4，对推荐度进行排序，如下图：

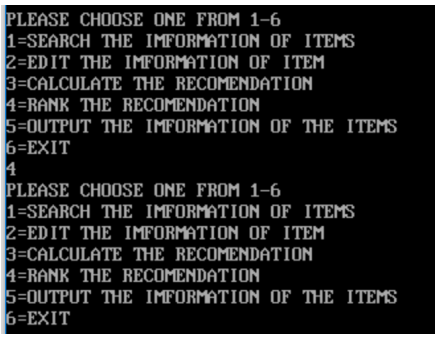


图 3-25 调用功能 4 示意图

此时我们已经得到了推荐度和商品排名信息，此时我们调用功能 5 将商品信息按排名。由下图可以看出，推荐度已经被正常计算，且是由排名从高到低输出的。截图如下：

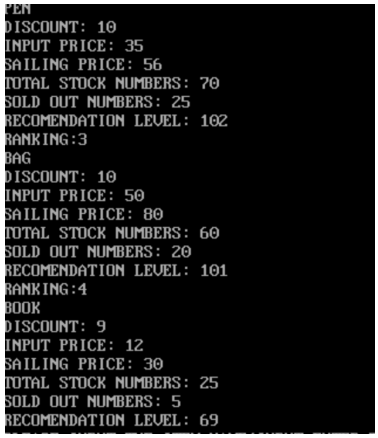


图 3-26 调用功能 5 输出所有商品信息（带排名）截图

综上所述，功能 3、功能 4、功能 5 能被正确调用，证明两人连接的程序是准确的。

3. 调用 td 观察 FAR 类型子程序的 RET 机器码与 NEAR 类型子程序的机器码，比较两者的不同。

我们将 CALCREFER 函数设置为 FAR 类型，RANKLEVEL 设置为 NEAR 类型，截图如下：

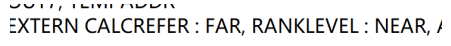


图 3-27 near 类型和 far 类型设定截图

下面给出 td 中 FAR 类型返回的截图，即子程序 CALCREFER 的返回在 td 中的截图：

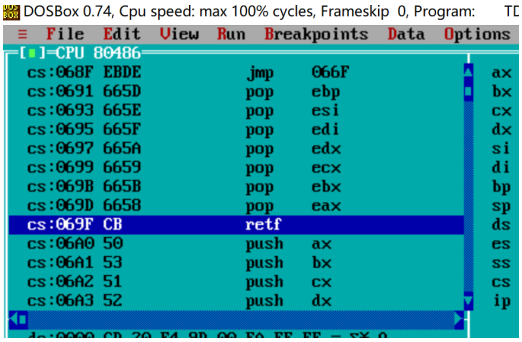


图 3-28 FAR 类型返回截图

下面给出 td 中 NEAR 类型返回的截图，即子程序 RANKLEVEL 的返回在 td 中的截图：

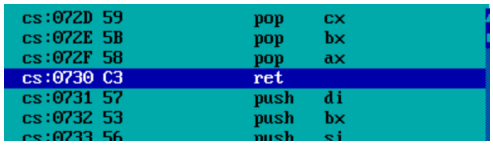


图 3-29 NEAR 类型返回截图

可以看出，NEAR 类型的返回的机器码为 C3，FAR 类型返回的机器码为 CB，而且 NEAR 类型返回在 td 中显示的语句为“ret”，FAR 类型返回在 td 中显示为“retf”。

4. 用 td 观察若将 NEAR 子程序强行使用 FAR 调用，或者一个 FAR 类型子程序强行用 NEAR 调用，会有什么结果。

子程序 CALCREFER 为 FAR 类型，子程序 RANKLEVEL 类型为 NEAR 类型。

1) 下面我们用 NEAR 类型强行调用 CALEFER 函数，截图如下：

```
CALCULATEREFER:
        CALL NEAR PTR CALCREFER
        JMP MENU
```

图 3-30 用 NEAR 强行调用 FAR 子程序截图

调用语句在 td 中如下图，发现机器码异常，截图如下所示：

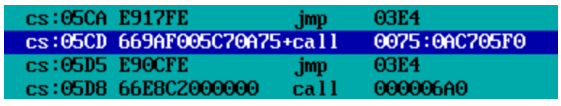


图 3-31 td 中对应调用语句的截图

此时我们运行这个程序，调用功能 3，发现程序无法正常运行，截图如下：

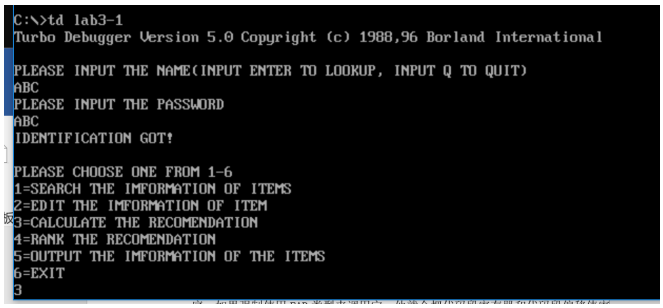


图 3-32 NEAR 强行 FAR 调用 FAR 类型子程序测试

2) 用 FAR 类型强行调用 RANKLEVEL 函数，截图如下：

```
RANKINGLEVEL:
        CALL FAR PTR RANKLEVEL
        JMP MENU
```

图 3-33 FAR 强行调用 NEAR 类型子程序截图

在 td 中对应调用语句的截图如下，该机器码正常：

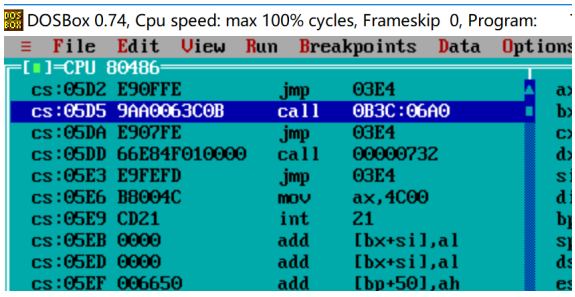


图 3-34 对应调用语句在 td 中截图

执行功能 4，发现能正常运行，截图如下：

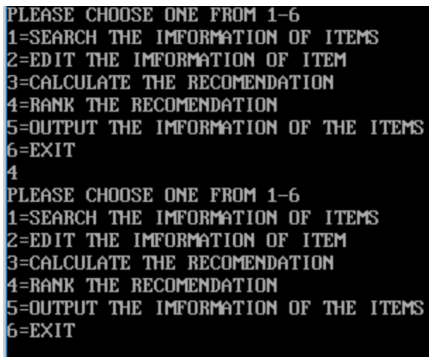


图 3-35 FAR 强行调用 NEAR 类型子程运行序截图

3) 总结

1. 用 NEAR 强行调用 FAR 类型子程序时，其调用语句机器码会出现异常，程序无法正常运行，而用 FAR 强行调用 NEAR 类型子程序时，机器码未发现异常，程序能正常运行。

2. 对这一现象进行解释，原因是 NEAR 类型机子程序发生调用的时候主程序堆栈中只压入 ip 的值，FAR 类型子程序发生调用时，会将 cs 和 ip 全部压入栈。所以当用 FAR 调用 NEAR 的时候不会出错，因为 ip 已经正常被压入栈，而 NEAR 强行调用 FAR 的时候，由于 cs 没有

被正常压入栈，所以会出现错误。

5. 用 td 观察将 EXTERN 语句放到.386 之前与.386 之后，有什么不同。

EXTERN 语句放在.386 之后，下图展示在 td 中，调用另一个文件子程序的语句，其中，光标所指的位置为调用函数 CALCULATEREFER 的语句：

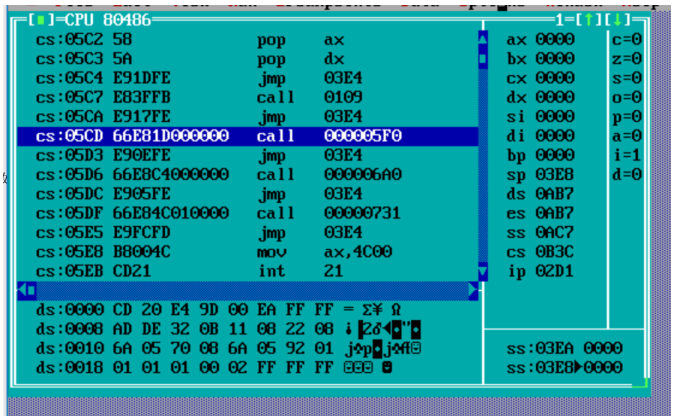


图 3-36 EXTERN 语句放在.386 语句之后调用子程序的截图

将 EXTERN 语句放在.386 之前，下图展示在 td 中，调用另一个文件子程序的语句，其中，光鸟所指的位置为调用函数 CALCULATEREFER 的语句：

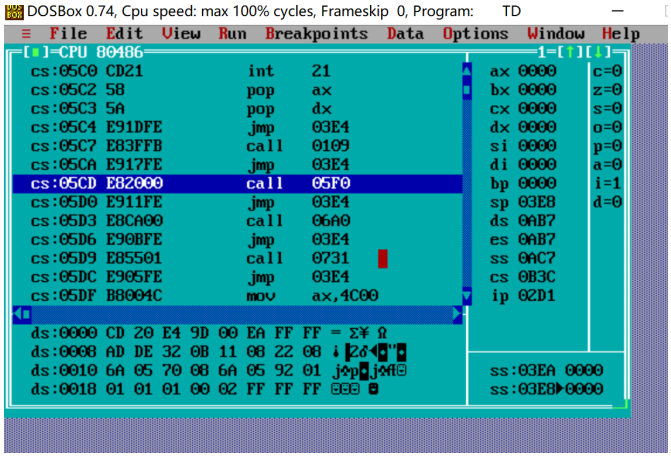


图 3-37 EXTERN 语句放在.386 语句之前调用子程序的截图

经过测试我们发现无论 EXTERN 语句放在.386 之前或者是.386 之后，程序都可以正常运行。

观察上述两幅截图，我们可以得出结论：

1. 将 EXTERN 语句放在.386 之前和.386 语句和之后，不影响程序语句的偏移地址。

2. 将 EXTERN 语句放在.386 之后时机器码为 12 位（16 进制），而放在.386 之前机器码为 6 位（16 进制）。

3. 将 EXTERN 语句放在.386 之后时 td 显示的 CALL 的数值是 8 位的（16 进制），而放在.386 之前的时候 CALL 的数值是 4 位（16 进制）的，且 8 位 CALL 的数值为 4 位数值高位补 4 个 0（16 进制）。

3.2 任务 2

3.2.1 设计思想及存储单元分配

1) 设计思想

该实验要求用 C 语言编写任务 1 的主控程序，以及其中一个菜单功能，然后调用汇编语言实现其它功能。

我的做法是用 C 语言编写包括商店数据存储、识别用户名以及密码、显示菜单、跳转菜单、程序退出功能在内的主控程序，然后用 C 语言编写 search 函数（即菜单功能 1），剩下的 4 个功能通过调用汇编代码实现。

实现思想主要如下：

1.数据存储：用一个结构体 shop，包括以下几个成员：商品名 item[10]（char 类型）、折扣 discount（char 类型）、进价 input（short 类型）、售价 sell（short 类型）、进货总数 initem（short 类型）、已售数量 outitem（short 类型）、推荐度 rec（short 类型）。然后我声明一个结构体数组用来存储我所有的商品信息。

2. 用户名判断：利用 if-else 结构，利用 strcmp 函数比较字符串。首先 gets 用户输入的 username，给一个 while（1）的无限循环，如果 strcmp 输入和真实用户名相等，则跳出循环，否则再如果判断到了回车，直接跳到显示菜单部分，否则再如果判断到了‘q’，直接跳到程序结束部分，即 return 0，其他情况则为用户名输错，提示错误信息，继续循环。

3. 密码判断：利用 if-else 结构，strcmp 函数比较字符串。接在用户名判断后面，还是先给一个 while（1）的无限循环。首先用 scanf 读入用书输入的 password。如果用 strcmp 判断得到输入的 password 与真实密码相同，则将 auth 置 1，跳出无限循环，其他情况则提示错误信息，继续循环。

4. 显示菜单：首先输出菜单信息 1，然后用 if 语句判断 auth 是否是 1，若为真，则输出信息 2-5，然后输出信息 6。

5. 跳转菜单：接收用户输入的跳转信息（数字 1-6），利用 if-else 语句，判断用户将要执行哪个菜单功能。如果输入为 1，则跳转到 search 函数，然后回到显示菜单；再如果输入为 2，调用汇编语言的 EDITP 函数，然后回到主菜单。依次类推，用 if-else if 语句一直从 1 判断到 6，如果用户输入的数字不是 1-6，即添加一个 else 语句输出错误信息。

6. search 函数：首先利用 if 语句和 strcmp 函数判断输入是否为回车，若是则返回。再利用 for 循环和 strcmp 函数，遍历商店里的商品信息，判断其货物名是否与商店里面已经有的货物相等，如果相等则输出所有货物信息，并跳出循环。如果遍历完了所有货物都没有找到，则输出所有错误信息。

7. 功能 2-5 函数:利用 extern 语句调用汇编程序中 EDITP,CALCREFER,RANKLEVEL,ALLINFO 函数（任务 1 中的汇编子程序）。

2） 存储单元分配

a） C 语言部分:

商店中的商品信息：利用一个结构体存储，结构体的成员包括商品名 item[10]（char 类型）、折扣 discount（char 类型）、进价 input（short 类型）、售价 sell（short 类型）、进货总数 initem（short 类型）、已售数量 outitem（short 类型）、推荐度 rec（short 类型）。然后利用结构体数组存储多个商品信息。

b） 汇编部分:

基本与任务 1 相同。

AX：存入数据。

BX：存入数据，循环计数。

CX：存储数据，循环计数。

DI、SI、BP：存入地址。

AL、AH、BL、BH、CL、CH、DL、DH：暂时存入数据。

3.2.2 流程图

下图展示 C 语言函数 search 的流程图：

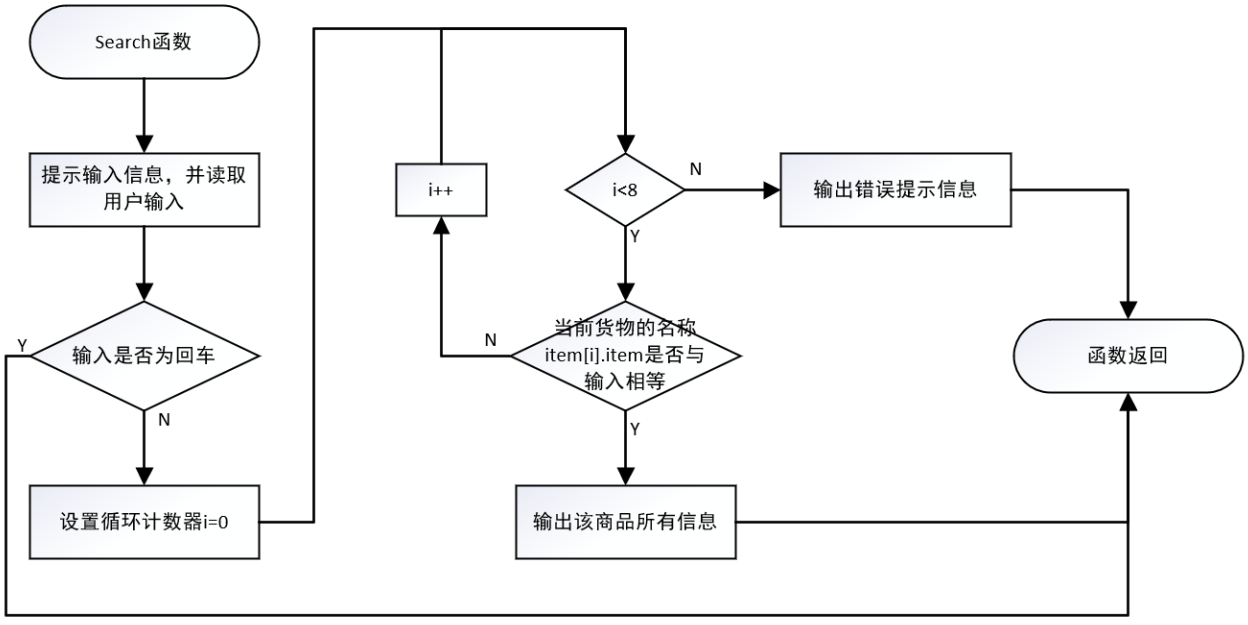


图 3-38 C 语言函数 search 流程图

下面显示 C 语言主控模块流程图：

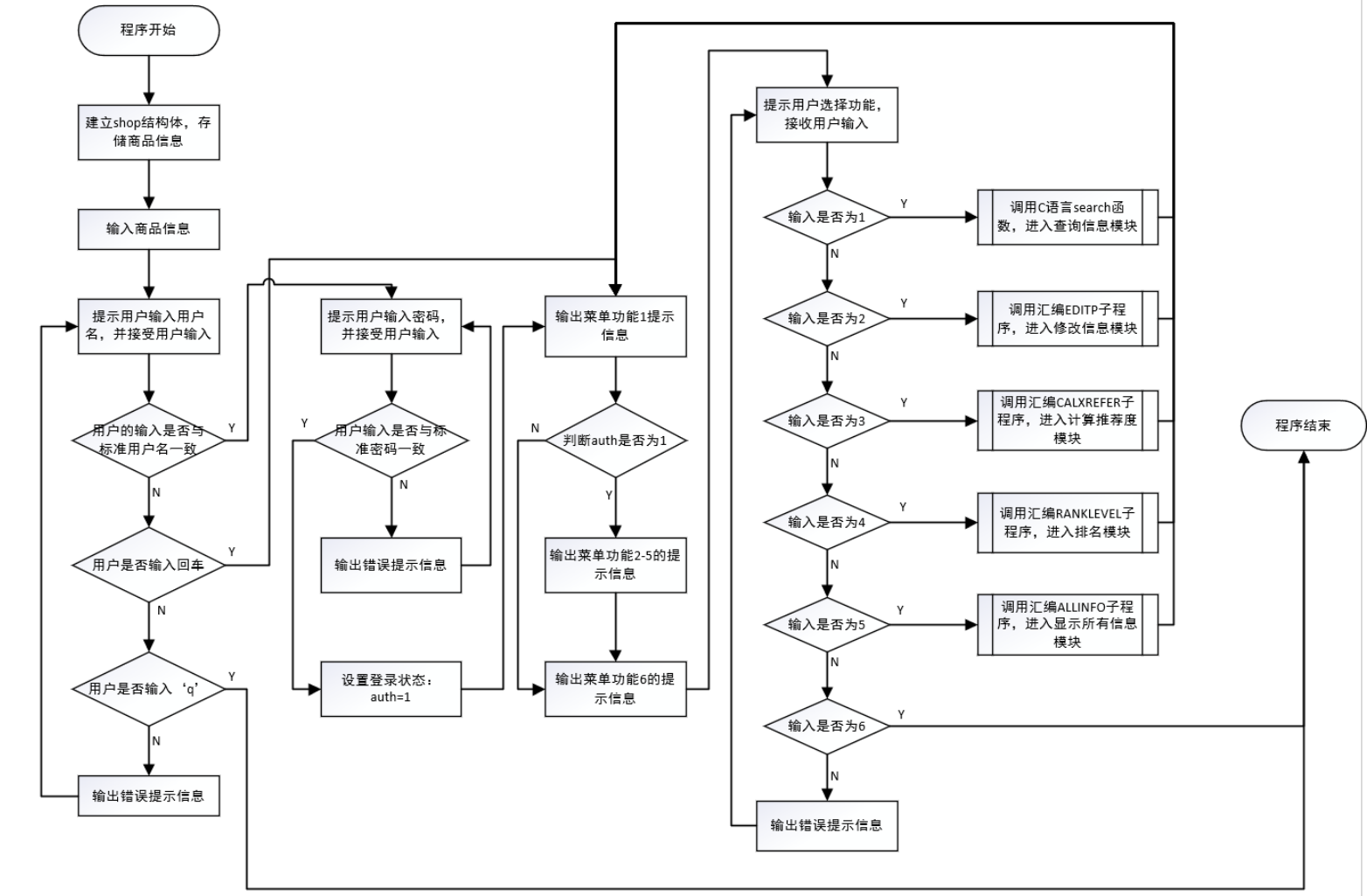


图 3-39 C 语言主控模块流程图

功能 2-5 的流程图与任务 1 基本一致。

3.2.3 源程序

```
#include <stdio.h>
#include <string.h>

extern void EDITIT(void);
extern void CALCULATEREFER(void);
extern void RANKINGLEVEL(void);

struct shop
{
    char item[10]; //商品名
    unsigned char discount; //折扣
    short input;  //进价
```

```
    short sell; //售价
    short initem; //进货总数
    short outitem; //已售数量
    short rec; //推荐度
};


void search(struct shop *item)
{
    char a[10];
    int i=0;
    printf("PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)\n");
    scanf("%s",&a);
    if(strcmp(a,"")==0)
    {
        return;
    }
    for(i=0;i<8;i++)
    {
        if(strcmp(a,item[i].item)==0)
        {
            printf("\n");
            printf("ITEM NAME: %s\n",item[i].item);
            printf("DISCOUNT: %d\n",item[i].discount);
            printf("INPUT PRICE: %d\n",item[i].input);
            printf("SAILING PRICE: %d\n",item[i].sell);
            printf("TOTAL STOCK NUMBERS: %d\n",item[i].initem);
            printf("SOLD OUT NUMBERS: %d\n",item[i].outitem);
            printf("RECOMENDATION LEVEL: %d\n",item[i].rec);
            break;
        }
    }
    if(i==8)
    {
        printf("\nDO NOT FIND THE ITEM!\n");
    }
}

void OUT(struct shop *item,int recomend[10])
{
    int i=0;
    for(i=0;i<8;i++)
    {
        printf("ITEM NAME: %s\n",item[i].item);
        printf("RANK: %d\n",recomend[i]);
        printf("DISCOUNT: %d\n",item[i].discount);
        printf("INPUT PRICE: %d\n",item[i].input);
        printf("SAILING PRICE: %d\n",item[i].sell);
        printf("TOTAL STOCK NUMBERS: %d\n",item[i].initem);
        printf("SOLD OUT NUMBERS: %d\n",item[i].outitem);
        printf("RECOMEDATION LEVEL: %d\n",item[i].rec);
        printf("\n");
    }
}

void fm1(struct shop *item)
{
```

```
    char a[10];
    char dis;
    int i;
    short einput,esell,einitem,eoutitem;
    printf("PLEASE INPUT HE ITEM NAME(INPUT ENTER TO INPUT NAME)\n");
    scanf("%s",&a);
    getchar();
    for(i=0;i<8;i++)
    {
        if(strcmp(a,item[i].item)==0)
        {
            printf("DISCOUNT: %d->",item[i].discount);
            scanf("%c",&dis);
            item[i].discount=dis;
            printf("INPUT PRICE: %d->",item[i].input);
            scanf("%d",&einput);
            item[i].input=einput;
            printf("SAILING PRICE: %d->",item[i].sell);
            scanf("%d",&esell);
            item[i].sell=esell;
            printf("TOTAL STOCK NUMBERS: %d->",item[i].initem);
            scanf("%d",&einitem);
            item[i].initem=einitem;
            printf("SOLD OUT NUMBERS: %d->",item[i].outitem);
            scanf("%d",&eoutitem);
            item[i].outitem=eoutitem;
            break;
        }
    }
    if(i==8)
    {
        printf("DO NOT FIND THE ITEM!\n");
    }
}

void fm2(struct shop *item)
{
    int i=0;
    for(i=0;i<8;i++)
    {
        short a,b,c,d,f;
        a=item[i].input*128*10/(item[i].discount*item[i].sell);
        b=item[i].outitem*64/item[i].initem;
        c=a+b;
        item[i].rec=c;
    }
}

void fm3(struct shop *item, int *recomend)
{
    short i,j;
    int a=1;
    struct shop temp;
    for(i=0;i<7;i++)
    {
        for(j=i+1;j<8;j++)
        {
```

```
                if(item[j].rec>item[i].rec)
                {
                    temp=item[i];
                    item[i]=item[j];
                    item[j]=temp;
                }
            }
        }
        recomend[0] = a;
        for(i=0;i<7;i++)
        {
            if(item[i].rec!=item[i+1].rec)
                a++;
            recomend[i+1]=a;
        }
    }

    int main()
    {
        struct shop item[8]={
            {"PEN",10,35,56,70,25,0},
            {"BOOK",9,35,56,70,25,0},
            {"PENCIL",9,50,30,25,5,0},
            {"BAG",9,50,80,60,20,0},
            {"TEXTBOOK",9,40,80,50,20,0},
            {"PAPER",9,5,10,50,30,0},
            {"GLUE",10,10,20,50,30,0},
            {"CANDY",9,2,4,60,40,0},
        };
        int num=0;
        int auth=0;
        char username[10],password[10];
        int recomend[10];

        while(1)
        {
            printf("PLEASE INPUT THE NAME(INPUT ENTER TO LOOKUP, INPUT q TO QUIT)\n");
            gets(username);
            if(strcmp(username,"ABC")==0)
            {
                break;
            }
            else if(strcmp(username,"")==0)
            {
                goto MENU;
            }
            else if(strcmp(username,"q")==0)
            {
                goto EXIT;
            }
            else
            {
                printf("WRONG NAME!\n");
            }
        }

        while(1)
```

```
{
     printf("PLEASE INPUT THE PASSWORD\n");
     scanf("%s",password);
     if(strcmp(password,"ABC")==0)
     {
          auth=1; //转换登录状态
          break;
     }
     else
     {
          printf("WRONG PASSWORD!\n");
     }
}

MENU:
     while(1)
     {
          printf("\nPLEASE CHOOSE ONE FROM 1-6\n");
          printf("1=SEARCH THE IMFORMATION OF ITEMS\n");
          if(auth==1)
          {
               printf("2=EDIT THE IMFORMATION OF ITEM\n");
               printf("3=CALCULATE THE RECOMENDATION\n");
               printf("4=RANK THE RECOMENDATION\n");
               printf("5=OUTPUT THE IMFORMATION OF THE ITEM\n");

          }
          printf("6=EXIT\n\n");
          printf("INPUT THE NUMBER\n");
          scanf("%d",&num);
          if(num==1)
          {
               search(item);
               goto MENU;
          }
          else if(num==2)
          {
               EDITIT();
               goto MENU;
          }
          else if(num==3)
          {
               CALCULATEREFER();
               goto MENU;
          }
          else if(num==4)
          {
               RANKINGLEVEL();
               goto MENU;
          }
          else if(num==5)
          {
               OUT(item,recomend);
               goto MENU;
          }
          else if(num==6)
          {
```

```
                goto EXIT;
            }
            else
            {
                printf("INPUT THE WRONG NUMBER!");
            }
        }

    EXIT:
        return 0;
}
```

3.2.4 实验步骤

1. 准备上机环境。下载并安装 Masm60 和 BorlandC 3.1。在 BC31 的 C 语言开发环境下与汇编语言程序实现混合编程。利用自己的编辑器(VS Code)写好 C 语言源程序，并改动任务 1 汇编程序使得其能够与编写的 C 语言正常编译。

2. 利用 bcc 编译写好的 C 语言源程序和改动后的汇编程序。若产生报错，可查阅相关手册查看错误原因，直到能够正常生成 obj 文件和 exe 文件。

3. 测试所得到的可执行文件功能是否正确。

a) 对于用户名输入的判断以及密码输入的测试：分别输入正确的用户名和错误的用户名，看输入正确的情况下，程序是否会继续到输入密码环节；若输入错误，看程序是否会提示输出错误。输入密码时输入正确密码看是否能正常显示菜单，输入错误的时候看能否提示错误。

b) 菜单显示测试：分别在登录和未登录的情况下执行到显示菜单的环节，看登录状况下是否显示 1-6 的所有提示信息，未登录状态下是否只显示 1 和 6 的提示信息。

c) 功能 1 测试：输入正确的和错误的商品信息，看输入正确商品时是否能正常显示信息，输入错误商品时是否能提示报错。

d) 功能为 2 测试：输入商品，更改其中的数值，然后再调用功能 1，看是否能正确更改信息。

e) 功能 3、4、5 测试：调用功能 3 和功能 4 分别计算推荐度和排名，然后调用功能 5，输出全部信息（包括排名），验证输出的推荐度和排名是否正确。

4. 对于生成的混合编程形成的程序，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关关系，具体做以下实验：

a) 看 C 语言语句编译之后形成的汇编语句的特点。(包括 C 语言的函数变成汇编有哪些特点，C 语言的数据存储变成汇编有哪些特点等等)

b) 观察 C 语言程序调用汇编语言时的特点。

5. 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，观察是否破坏程序正确性的目的，具体步骤如下：

在多行表示计算的式子中插入汇编语句，改变寄存器的值，看对程序运行结果有什么影响。查看 C 语言编译之后的汇编代码，解释其原因。

3.2.5 实验记录与分析

1. 利用 bcc 编译程序 assembly.c 和 support.asm 的时候没有产生报错，正常生成了 obj 文件和

exe 文件。

2. 程序功能测试

（1）输入姓名与输入密码测试

首先输入正确的姓名，发现会跳转到输入密码环节，截图如下：

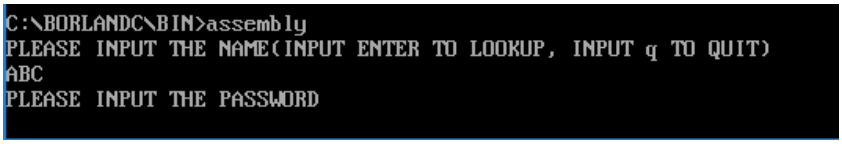


图 3-40 输入姓名测试 1

输入错误的姓名，发现会提示错误并重新输入，截图如下：

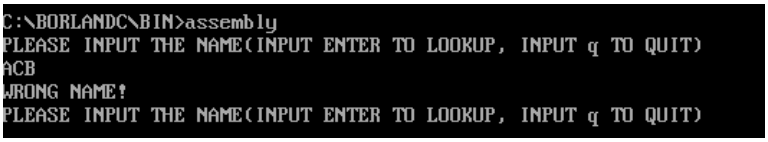


图 3-41 输入姓名测试 1

输入正确的密码跳转到菜单，截图如下：

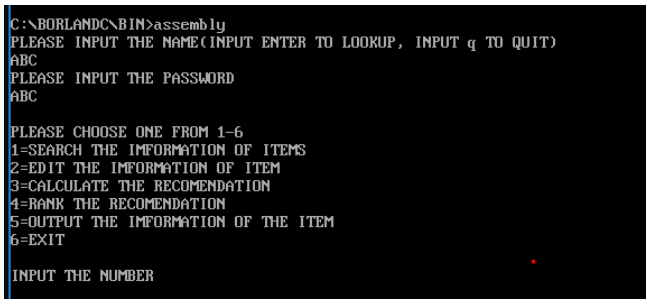


图 3-42 输入密码测试 1

输入错误的密码会产生错误提示，并要求重新输入，截图如下：

图 3-43 输入密码测试 2

（2）功能 1 测试

输入商店里面存在的商品时，正常显示信息（推荐度没计算时为 0），截图如下：

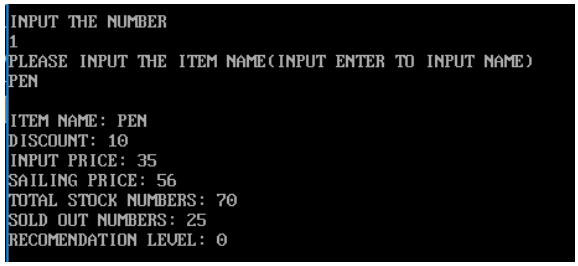


图 3-44 功能 1 测试 1

输入商店里面不存在的商品时，提示错误信息，截图如下：

图 3-45 功能 1 测试 2

（3）功能 2 测试

更改商店中商品的值，如下图所示：

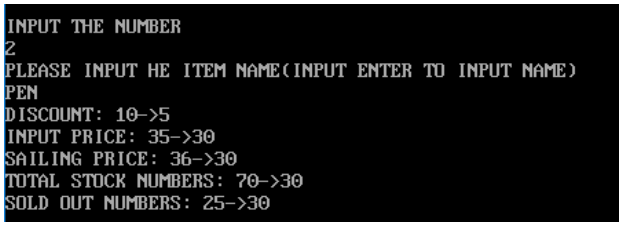


图 3-46 功能 2 测试截图 1

此时我们调用功能 1，看商品信息是否被正确更改。结果证明商品信息得到了正确的修改，截图如下：

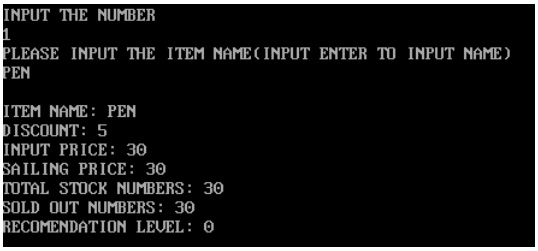


图 3-47 功能 2 测试截图 2

（4）功能 3、4、5 测试

首先调用功能 3，对推荐度进行计算，截图如下所示：

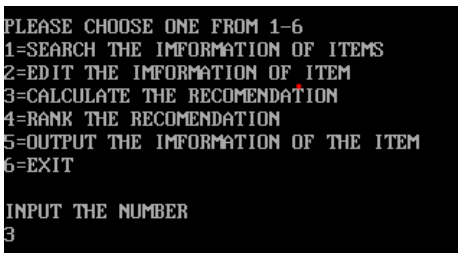


图 3-48 调用功能 3

再调用功能 4，对推荐度进行排序，截图如下所示：

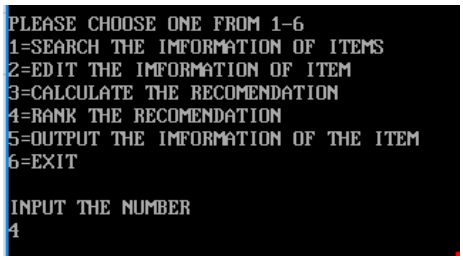


图 3-49 调用功能 4

此时，我们已经计算出了商店所有商品的推荐度，并从高到进行了排序，然后，调用功能 5，输出所有信息，可以看出，推荐度已经被正确计算，且按照从高到低的顺序排名，截图如下所示：

图 3-50 调用功能 5 输出带排名的信息截图

我们可以得出，推荐度的计算与排名的计算都是正确的，输出功能也是正确的，即功能 3、4、5 得以验证。

3. 观察 C 语言产生的汇编代码的特点与 C 内嵌汇编时候的连接特点

我们利用 BCC -S 指令编译 assembly.c support.asm 文件（即目标 C 文件与调用用的汇编文件），生成一个 ASM 文件，名称为（assembly.asm）,这个文件里面记录着将 assembly.c support.asm 编译之后形成的汇编语言文件，从中我们可以清晰地看到每条 C 语言语句编译成汇编语言之后的语句特点。

（1）C 语言翻译成汇编之后汇编语句的特点

a) 首先我们看 C 语言函数 search 编译成汇编之后的语句，结果如下：

```
;
;       void search(struct shop *item)
;
        assume  cs:_TEXT
_search proc    near
        push    bp
        mov     bp,sp
        sub     sp,10
        push    si
        push    di
        mov     di,word ptr [bp+4]
.
```

图 3-51 C 语言函数 search 编译成汇编之后的结果

可以看出，汇编语言将 C 语言函数变成了一个子程序，并且在命名前面你加上了下划线，在子程序开始时，执行保护现场操作，和汇编语言子程序结构一致。

b) 然后我们看 C 语言输出语句变成汇编语言之后的语句，结果如下：

```
;
;               printf("PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)\n");
;
        mov     ax,offset DGROUP:s@
        push    ax
        call    near ptr _printf
        pop     cx
```

图 3-52 C 输出语句变成汇编之后的结果

可见，汇编语言先把要输出的内容存在数据段中，即图中所示的 s@段，然后汇编语言调用了一个函数_printf 用于输出这个语句，在调用_printf 之前将 ax 压栈，调用后将 cx 出栈。

c) 接下来我们看下 C 语言存储商店信息（一个结构体数组）变成汇编语言之后的语句，结果如下：

```
;
;       {
;               struct shop item[8]={
;
;
;                       {"PEN",10,35,56,70,25,0},
;                       {"BOOK",9,35,56,70,25,0},
;                       {"PENCIL",9,50,30,25,5,0},
;                       {"BAG",9,50,80,60,20,0},
;                       {"TEXTBOOK",9,40,80,50,20,0}
;                       {"PAPER",9,5,10,50,30,0},
;                       {"GLUE",10,10,20,50,30,0},
;                       {"CANDY",9,2,4,60,40,0},
;               };
;
        lea     ax,word ptr [bp-190]
        push    ss
        push    ax
        mov     ax,offset DGROUP:d@w+0
        push    ds
        push    ax
        mov     cx,168
        call    near ptr N_SCOPY@
```

图 3-53 C 语言存储结构体变成汇编语句

可以看出，汇编将所有商品的信息数值存在数据段中，首地址为 DGROUP:d@w+0，然后调用 N_SCOPY@存入了一个变量中。

（2）C 语言调用汇编子程序的特点

```
;
;                       {
;                               CALCULATEREFER();
;
        call    near ptr _CALCULATEREFER
```

图 3-54 C 语言调用汇编子程序

可以看出，C 语言调用汇编子程序时，汇编语言直接翻译为用 NAER 类型调用该函数，其中函数名之前加上了下划线。

在 td 中查看调用的部分如下：

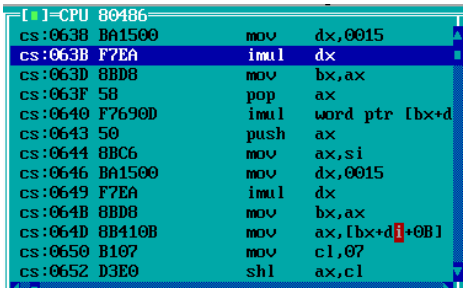


图 3-55 在 td 中查看 CALCULATER 函数对应部分

可以看出，对应的汇编语句与我们单独编译 asm 文件在 td 中查看到的机器表示基本一致，所以我们得出，对应的汇编子程序在 C 语言编译成汇编时并没有改变其结构。

4. 在 C 语言中不合理地嵌入汇编语句，观察执行结果

我们在多行计算式子中插入汇编语句 MOV BP,10，如下截图所示：

```
d=d*e/10; //a=10,b=20,c=10,d=20,e=5
a=a*128;
asm mov bp,10
a=a/d;
b=b*64;
b=b/c;
sum=a+b;
printf("RECOMENDATION LEVEL: %d\n",sum);
```

图 3-56 在多行计算式中不合理地嵌入汇编语句

编译执行之后我们可以发现，运行结果如下：

图 3-57 嵌入不合理汇编语句之后运行结果

这说明计算结果为 30，而通过我们手动计算，这个数值应该等于 256，所以 MOV AX,0 这一语句破坏了原本的 C 语言程序，我们利用 bcc -S 指令，调生成将 C 语言转变成汇编语言之后的代码文件，对应部分截图如下：

```
;
;               a=a*128;
;
        mov     ax,si
        mov     cl,7
        shl     ax,cl
        mov     si,ax
;
;       //      asm mov si,0
;       //      asm mov di,0
;       //      asm mov ax,0
;       //      asm mov cx,0
;               asm mov bp,10
;
        mov      bp,10
;
;               a=a/d;
;
        mov     ax,si
        cwd
        idiv    word ptr [bp-4]
        mov     si,ax
;
;               b=b*64;
;
        mov     ax,di
        mov     cl,6
        shl     ax,cl
        mov     di,ax
```

图 3-58 对应生成的汇编语句

注：其中用分号注释掉的是在原来的 C 语言文件中的语句，后面所跟随的汇编代码是机器翻译 C 语言文件成汇编文件的结果。

我们可以看出，机器把 asm mov bp,10 翻译成了对应的语句 mov ax,10。而我们注意到机器在翻译 a=a/d 的时候，用到了 idiv word ptr [bp-4]指令，而我们在之前已经认为的将 bp 的值变为了 0，故结果错误。

去掉 asm mov bp,10 便能得到正确的结果，截图如下：

图 3-59 去掉干扰汇编语句之后得到正确结果

总结：我们在 C 语言中内嵌汇编语言的时候（尤其是遇到上述多行计算式的情况），如果没考虑 C 语言翻译成汇编之后机器的调用情况，而嵌入修改寄存器的语句，则很有可能导致程序运行错误。

4 总结与体会

任务一是我们第一次尝试模块化设计，通过这次实验，我们对于程序的模块化有了深刻的了解，对于子程序的编写实践性的理解，对于子程序的有关概念，例如 NEAR 类型子程序和 FAR 类型的子程序在机器中的表达方式与特点，也进行了实践与理解。同时，这次实验也是第一次尝试两人分工合作，两人在合作的时候最重要的就是统一寄存器的分配、函数的使用、变量的制定，这些协议应该在编写测试程序之前就详细制定好，这次实验很好地积累了我们对于模块化设计以及多人合作连接程序的经验与能力。

在实验过程中遇到的最大的困难就是我们两人在编写程序最初没有约定好一些协议，而是只大概约定好了一些函数和功能的分配，这导致我们在后来连接的过程中花了很大的力气对于一些分配不合理的寄存器，可能会产生冲突的变量，以及子程序要保护的寄存器等等方面做了大幅度的修改，花费了一定的时间。所以，在做多人合作的时候，一定先要把整体框架想清楚，在这个基础上，详细地制定每个变量，特殊寄存器的使用，子程序的 API，子程序的调用关系，这些都是很重要的。

任务二让我们了解到不通的编程语言是可以协助解决一个问题的，我们可以利用各个语言的长处与处理特点更好的更高效的解决一个问题。这个任务让我们实现了 C 语言与汇编语言的混合编程，通过查看 C 语言变成汇编语言之后的源代码，我们了解到了 C 语言编译成汇编语言之后的特点，也了解了 C 语言调用汇编语言的特点。同时，我也做了实验发现在有的情况下，不能够在 C 语言代码中随意嵌入汇编语言改变寄存器的值，这样会导致程序运行错误（尤其是在几行计算公式对应的 C 语句的中间），我们需要考虑临近的语句翻译成机器码之后的特点，再考虑怎样嵌入汇编语句。

在做任务二的过程中遇到的最大困难就是 C 语言调用汇编子程序的时候会出现一些错误，会导致程序运行错误，需要结合错误现象慢慢调试。

参考文献

[1] 王元珍 曹忠升 韩宗芬 编著.80X86 汇编语言程序设计.武汉:华中科技大学出版社,2005 年 4 月.128 页-187 页

[2] URL: https://blog.csdn.net/qq_43079376/article/details/85166040