

华中科技大学

课程实验报告

课程名称： 汇编语言程序设计实验

实验时段： 2019 年 3 月-2019 年 5 月

指导教师： 张勇

专业班级： 计算机 201706 班

学 号： U201714761

姓 名： 胡澳

实验报告成绩：

实验 序号	一	二	三	四	五	总评
成绩						

备注：总评是实验报告的最终成绩，是五次成绩的平均值，它将与平时实验过程中的表现一起构成本实验课程的最终成绩。

指导教师签字：

日期：

计算机科学与技术学院

华中科技大学

课程实验报告

课程名称： 汇编语言程序设计实验

实验名称： 实验一 编程基础

实验时间： 2018-3-19, 14: 00-17: 30

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： U201714761

姓 名： 胡澳

同组学生： 无

报告日期： 2018 年 3 月 20 日

原创性声明

本人郑重声明：本报告的内容由本人独立完成，有关观点、方法、数据和文献等的引用已经在文中指出。除文中已经注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品或成果，不存在剽窃、抄袭行为。

特此声明！

学生签名：

日期：2018.3.20

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：

汇编语言程序设计实验报告

目 录

1 实验目的与要求	2
2 实验内容	2
3 实验过程	5
3.1 任务 1~3	5
3.1.1 任务 3 源代码.....	5
3.1.2 关键的实验记录与分析.....	6
3.2 任务 4.....	9
3.2.1 设计思想及存储单元分配.....	9
3.2.2 流程图.....	10
3.2.3 源程序.....	10
3.2.4 实验步骤.....	18
3.2.5 实验记录与分析.....	18
4 总结和体会	21

汇编语言程序设计实验报告

1 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。

2 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）。

汇编语言程序设计实验报告

(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 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 $[=(\text{进货价}/\text{实际销售价格}+\text{已售数量}/(2*\text{进货数量}))*128, \text{字类型}]$ 。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

汇编语言程序设计实验报告

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）。

汇编语言程序设计实验报告

3 实验过程

3.1 任务 1~3

3.1.1 任务 3 源代码

```
.386
STACK SEGMENT USE16 STACK
    DB 200 DUP(0)
STACK ENDS

DATA SEGMENT USE16
BUF1    DB 0,1,2,3,4,5,6,7,8,9
BUF2    DB 10 DUP(0)
BUF3    DB 10 DUP(0)
BUF4    DB 10 DUP(0)
DATA ENDS

CODE SEGMENT USE16
    ASSUME CS:CODE, DS:DATA, SS:STACK
START:
    MOV AX,DATA
    MOV DS,AX
    MOV ESI,0
    MOV EDI,0
    MOV EBX,0
    MOV EBP,0
    MOV CX,10
LOPA:
    MOV AL, BUF1[ESI]
    MOV BUF2[EDI],AL
    INC AL
    MOV BUF3[EBX], AL
    ADD AL,3
    MOV BUF4[EBP],AL
    INC ESI
    INC EDI
    INC EBP
    INC EBX
    DEC CX
    JNZ LOPA
    MOV AH,4CH
    INT 21H
CODE ENDS
```

汇编语言程序设计实验报告

END START

3.1.2 关键的实验记录与分析

1. 实验一

使用 TD 求题目中三组数据的和，实验记录截图如图 1-1、1-2、1-3 所示。

执行指令后，(AH)、SF、OF、CF、ZF 的值见表 1-1。

表 1-1 执行指令后(AH)、SF、OF、CF、ZF 的值

算式	(AH)	SF	OF	CF	ZF
0110011B + 1011010B	8DH	1	1	0	0
-0101001B + (-101101B)	7AH	0	1	1	0
1100101B + (-101101B)	08H	0	0	1	0

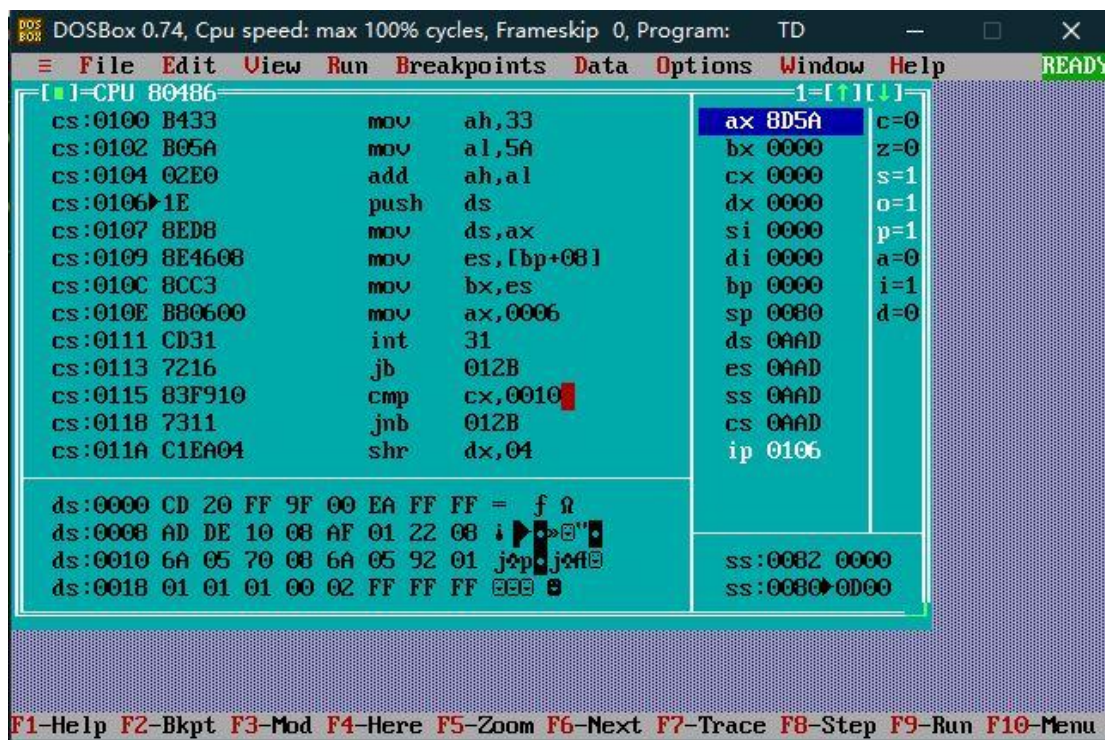


图 1-1 0110011B + 1011010B 计算结果截图

汇编语言程序设计实验报告

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TD

File Edit View Run Breakpoints Data Options Window Help

CPU 80486

cs:0100 B4D7	mov	ah,D7	ax 7AA3	c=1
cs:0102 B0A3	mov	al,A3	bx 0000	z=0
cs:0104 02E0	add	ah,al	cx 0000	s=0
cs:0106 1E	push	ds	dx 0000	o=1
cs:0107 8ED8	mov	ds,ax	si 0000	p=0
cs:0109 8E4608	mov	es,[bp+08]	di 0000	a=0
cs:010C 8CC3	mov	bx,es	bp 0000	i=1
cs:010E B80600	mov	ax,0006	sp 0080	d=0
cs:0111 CD31	int	31	ds 0AAD	
cs:0113 7216	jb	012B	es 0AAD	
cs:0115 83F910	cmp	cx,0010	ss 0AAD	
cs:0118 7311	jnb	012B	cs 0AAD	
cs:011A C1EA04	shr	dx,04	ip 0106	

ds:0000 CD 20 FF 9F 00 EA FF FF = f 0
ds:0008 AD DE 10 08 AF 01 22 08 i 0 0 0 0
ds:0010 6A 05 70 08 6A 05 92 01 j 0 p 0 j 0 f 0
ds:0018 01 01 01 00 02 FF FF FF 0 0 0 0

ss:0082 0000
ss:0080 0D00

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

图 1-2 -0101001B + (-1011101B)计算结果截图

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TD

File Edit View Run Breakpoints Data Options Window Help

CPU 80486

cs:0100 B465	mov	ah,65	ax 08A3	c=1
cs:0102 B0A3	mov	al,A3	bx 0000	z=0
cs:0104 02E0	add	ah,al	cx 0000	s=0
cs:0106 1E	push	ds	dx 0000	o=0
cs:0107 8ED8	mov	ds,ax	si 0000	p=0
cs:0109 8E4608	mov	es,[bp+08]	di 0000	a=0
cs:010C 8CC3	mov	bx,es	bp 0000	i=1
cs:010E B80600	mov	ax,0006	sp 0080	d=0
cs:0111 CD31	int	31	ds 0AAD	
cs:0113 7216	jb	012B	es 0AAD	
cs:0115 83F910	cmp	cx,0010	ss 0AAD	
cs:0118 7311	jnb	012B	cs 0AAD	
cs:011A C1EA04	shr	dx,04	ip 0106	

ds:0000 CD 20 FF 9F 00 EA FF FF = f 0
ds:0008 AD DE 10 08 AF 01 22 08 i 0 0 0 0
ds:0010 6A 05 70 08 6A 05 92 01 j 0 p 0 j 0 f 0
ds:0018 01 01 01 00 02 FF FF FF 0 0 0 0

ss:0082 0000
ss:0080 0D00

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

图 1-3 1100101B + (-1011101B)计算结果截图

2. 任务二

使用文本编辑器将题 2-3 的源代码输入到 problem2.asm 中，在 DosBox 中使用 masm 编译、使用 link 链接得到 problem2.exe 的可执行文件。使用 td 打开该文件，首先单步执行，

汇编语言程序设计实验报告

观察执行指令后相应寄存器以及标志位的变化。当执行到“MOV CX, 10”之前时，td 界面的截图如图 1-4 所示。在指令“INT 21H”处添加断点，将程序执行到断点出，此时 td 界面的截图如图 1-5 所示，此时内存显示区使用 goto 功能，使该区域显示从 DS:0000 开始的内存空间的内容，该内存空间的内容的截图如图 1-6 所示。当程序执行到这两条指令前的时候，(BX)、(BP)、(SI)、(DI)的值见表 1-2。

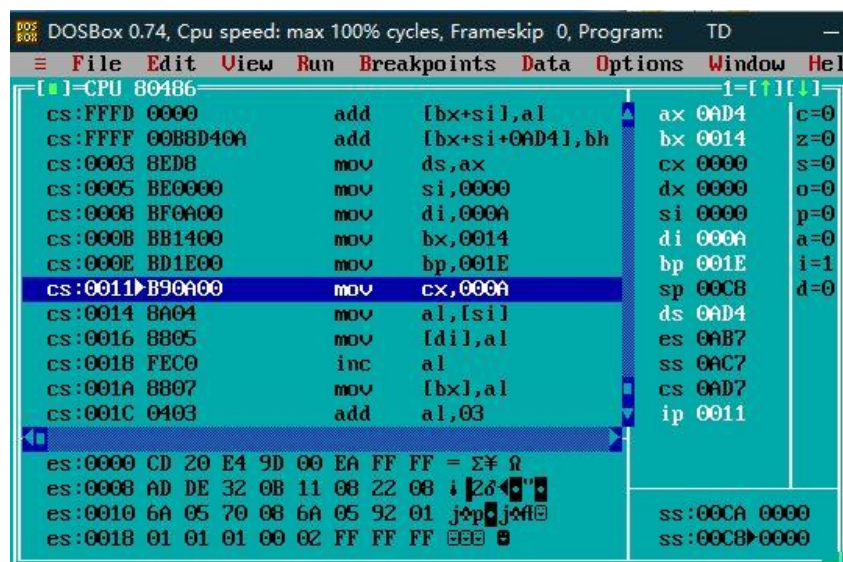


图 1-4 程序执行到“MOV CX, 10”前的截图

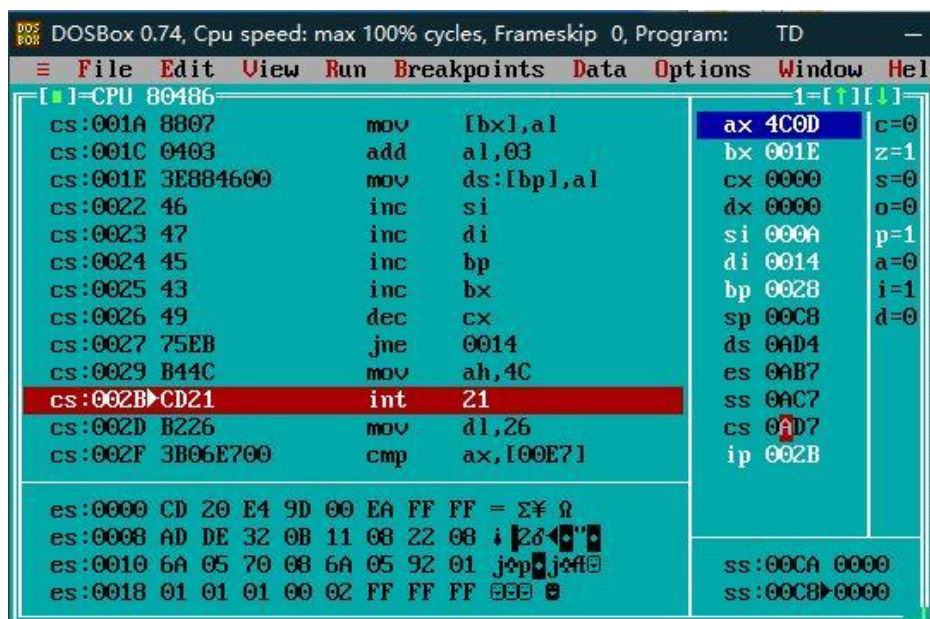


图 1-5 程序执行到“INT 21H”前的截图

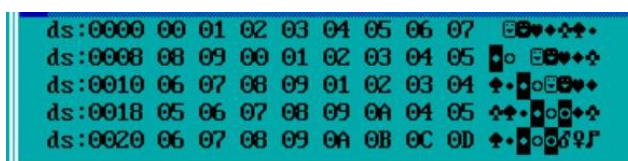


图 1-6 程序执行到“INT 21H”前 DS 段内存的内容

汇编语言程序设计实验报告

表 1-2 程序执行到相应指令前(BX)、(BP)、(SI)、(DI)的值

指令	BX	BP	SI	DI
MOV CX, 10	14H	1EH	0	0AH
INT 21H	1EH	28H	0AH	14H

3. 任务三

修改后的源程序见 3.1.1，将该源程序编译链接后使用 td 打开调试，在程序结束前加上断点，并将程序执行到该断点的位置，然后将内存显示区域调整为显示 DS 段的内存内容，此时内存信息的截图如图 1-7 所示。

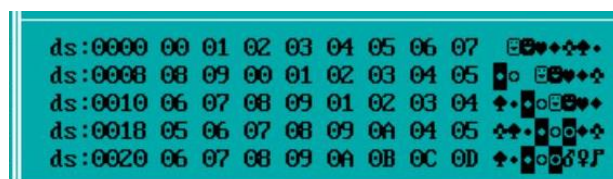


图 1-7 修改后的程序运行到结束前 DS 段的内存内容

将该内存信息与修改前 DS 段的内存信息、即图 1-6 的内容相互比较，发现两者内容完全相等，由此可推断，修改前后的程序功能未发生变化。

从不同位置反汇编二进制串会产生不同的汇编指令，由此可知，IP/EIP 寄存器指明下一条待执行指令的偏移地址对于程序的执行是至关重要的，否则，计算机将无法正确的解析可执行文件，从而无法正确执行程序。

3.2 任务 4

3.2.1 设计思想及存储单元分配

程序的设计思路如下：调用 DOS 系统功能，在控制台与用户进行简单的交互。首先程序使用 9 号功能输出提示信息，并使用 10 号功能读取用户输入，然后根据输入判断用户是老板还是顾客，并将该信息存储到相应的内存单元中。判断规则为：若用户输入了用户名和密码，则判断输入是否正确，若正确，则为老板，否则，输出登陆错误；若用户未输入数据，则认为用户为顾客。然后程序提示用户输入待查看的商品的名称，程序读取到商品名称后查找该商品，若找到该商品，则根据用户的类型给定相应的输出，若未找到该商品，则输出未找到该商品。

存储单元分配方案如下：所有的数据均存储在数据段中。老板名字以及密码均定义为 10 个字节的串，并在该串的后面存储一个数字，用于存储名字和密码的长度。商店名称定义为以 '\$' 结束的串。将用户类型定义为字节类型，1 表示为老板，0 表示为顾客。商品信息首先为一个 10 个字节的串，用于保存商品名称，接下来一个字节用于存储商品名称的长度，接下来一个字节为商品的折扣率，最后有 5 个字类型的数据，分别存储商品的进货价、销售价、进货数量、已售数量和推荐度。另外，还设有两个内存空间用于存储用户输入的用户名和密码。

汇编语言程序设计实验报告

3.2.2 流程图

程序流程图如图 1-8 所示。

3.2.3 源程序

程序的源代码如下：

```
.386
data segment use16
    bname DB 'huao', 6 dup(0), 4 ;boss name
    bpass DB 'huao', 6 dup(0), 4 ;password
    N EQU 30 ;number
    shopname DB 'huao$' ;shop name

    auth DB ?

    ga1 DB 'pen$', 6 dup(0)
        DB 3
        DB 10
        DW 35, 56, 70, 25, ?
    ga2 DB 'book$', 5 dup(0)
        DB 4
        DB 9
        DW 12, 30, 25, 5, ?

    welcomeMsg DB 'welcome to ', '$'
    welcomeShopMsg DB ' shop$'

    loginErrorMsg DB 'Login Error', 0ah, 0dh, '$'

    outenter DB 0ah, 0dh, '$'
    info DB 'input your name and password', 0ah, 0dh, '$'
    in_name DB 10, ?, 10 dup(0), '$'
    in_pass DB 6, ?, 6 dup(0), '$'

    checkname DB 'input commodity you want to check:', '$'
    goodname DB 10, ?, 10 dup(0)
    notFound DB 'good not found', 0ah, 0dh, '$'
data ends

stack segment use16 stack
    DB 200 dup(0)
stack ends
```

汇编语言程序设计实验报告

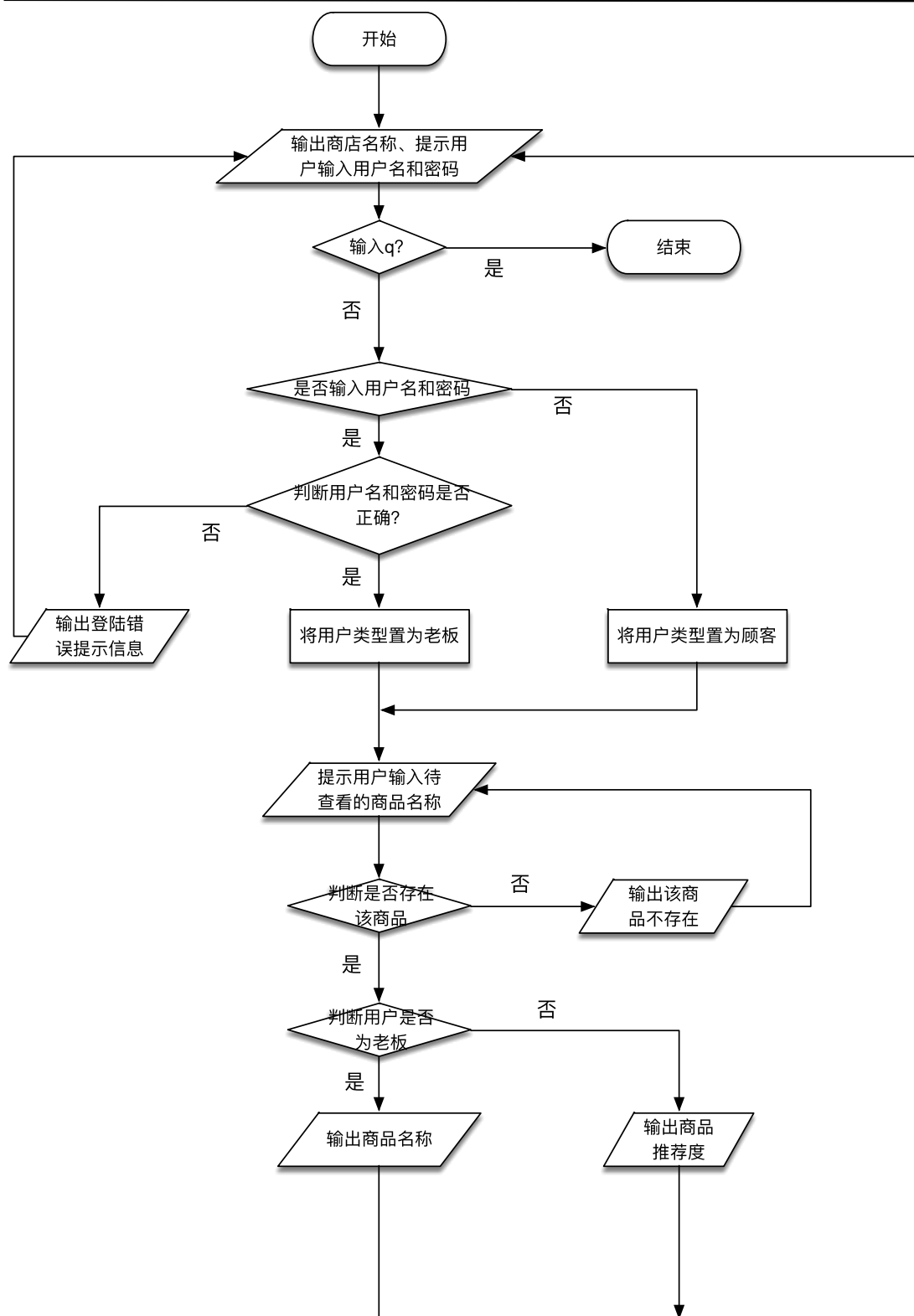


图 1-8 任务 4 流程图

```
code segment use16
    assume DS:data, CS:code, SS:stack
start:
    mov ax, data
    mov ds, ax
```

汇编语言程序设计实验报告

```
lea dx, welcomeMsg[0]
mov ah, 9
int 21h
```

```
lea dx, shopname[0]
mov ah, 9
int 21h
```

```
lea dx, welcomeShopMsg[0]
mov ah, 9
int 21h
```

```
lea dx, outenter[0]
mov ah, 9
int 21h
```

```
lea dx, info[0]
mov ah, 9
int 21h
```

```
lea dx, in_name[0]
mov ah, 10
int 21h
```

```
lea dx, outenter[0]
mov ah, 9
int 21h
```

```
mov al, in_name[1]
cbw
mov bx, ax
mov ax, 0
mov in_name[bx + 2], al
```

```
mov al, in_name[1]
cmp al, 0
jz user
cmp al, 1
jz checkQuit
jmp next3
```

```
checkQuit:
mov al, in_name[2]
```

汇编语言程序设计实验报告

```
    cmp al, 71h
    jz quit

next3:
    lea dx, in_pass[0]
    mov ah, 10
    int 21h

    mov al, in_pass[1]
    cbw
    mov bx, ax
    mov ax, '$'
    mov in_pass[bx + 2], al

    lea dx, outenter[0]
    mov ah, 9
    int 21h

    lea bx, in_name[2]
    mov al, in_name[1]
    cmp al, bname[10]
    jnz errorLogin
    call compareName
    cmp ch, 1
    jnz errorLogin

    lea bx, in_pass[2]
    mov al, in_pass[1]
    cmp al, bpass[10]
    jnz errorLogin
    call comparePass
    cmp ch, 1
    jnz errorLogin

    mov al, 1
    mov auth[0], al
    jmp recommandGood

errorLogin:
    lea dx, loginErrorMsg[0]
    mov ah, 9
    int 21h
    jmp start
```

汇编语言程序设计实验报告

user:

```
mov al, 0
mov auth[0], al
jmp recommandGood
```

recommandGood:

```
lea dx, checkname[0]
mov ah, 9
int 21h
```

```
lea dx, goodname[0]
mov ah, 10
int 21h
```

```
lea dx, outenter[0]
mov ah, 9
int 21h
```

```
mov cl, goodname[1]
cmp cl, 0
jz start
```

```
call findgood
cmp ch, -1
jz printNotFound
```

```
mov cl, auth[0]
cmp cl, 1
jz authOK
jmp authNotOK
```

authOK:

```
mov dx, bx
mov ah, 9
int 21h
```

```
lea dx, outenter[0]
mov ah, 9
int 21h
```

```
jmp start
```

authNotOK:

```
mov ax, [bx + 12]
```


汇编语言程序设计实验报告

```
mov cx, 1280
mul cx
push dx
push ax
mov al, byte ptr [bx + 14]
mov cl, [bx + 11]
mul cl
mov cx, ax
pop ax
pop dx
div cx
push ax
```

```
mov ax, [bx + 18]
mov cx, 64
mul cx
mov cx, [bx + 16]
div cx
mov dx, ax
pop ax
add ax, dx
```

```
jmp func4
```

```
printNotFound:
    lea dx, notFound[0]
    mov ah, 9
    int 21h
    jmp recommandGood
```

```
func4:
    cmp ax, 100
    jns printA
    cmp ax, 50
    jns printB
    cmp ax, 10
    jns printC
    jmp printF
```

```
printA:
    mov dl, 41H
    mov ah, 2
    int 21h
    jmp returnStart
```

汇编语言程序设计实验报告

printB:

```
    mov dl, 42H
    mov ah, 2
    int 21h
    jmp returnStart
```

printC:

```
    mov dl, 43H
    mov ah, 2
    int 21h
    jmp returnStart
```

printF:

```
    mov dl, 46H
    mov ah, 2
    int 21h
    jmp returnStart
```

returnStart:

```
    lea dx, outenter[0]
    mov ah, 9
    int 21h
    jmp start
```

Quit:

```
    mov ah, 4Ch
    int 21h
```

findgood proc

```
    mov bl, goodname[1]
    mov bh, 0
    mov si, bx
```

```
    lea bx, ga1[0]
    mov al, [bx + 10]
    mov ah, 0
    cmp ax, si
    jnz next1
```

```
    call compare
    cmp ch, 1
    jz foundgood
```

汇编语言程序设计实验报告

```
next1:
    lea bx, ga2[0]
    mov al, [bx + 10]
    mov ah, 0
    cmp ax, si
    jnz next2

    call compare
    cmp ch, 1
    jz foundgood

next2:
    mov ch, -1
    ret

foundgood:
    mov ch, bh
    ret
foundgood endp

compare proc
comp:
    cmp al, 0
    jz found
    dec al
    mov ah, 0
    mov di, ax
    mov dl, goodname[2 + di]
    sub dl, [bx + di]
    jz comp

    mov ch, 0
    ret
found:
    mov ch, 1
    ret
compare endp

compareName proc
compName:
    cmp al, 0
    jz trueName
    dec al
    mov ah, 0
```

汇编语言程序设计实验报告

```
    mov di, ax
    mov dl, bname[di]
    sub dl, [bx + di]
    jz compName

    mov ch, 0
    ret
trueName:
    mov ch, 1
    ret
compareName endp

comparePass proc
compPass:
    cmp al, 0
    jz truePass
    dec al
    mov ah, 0
    mov di, ax
    mov dl, bpass[di]
    sub dl, [bx + di]
    jz compPass

    mov ch, 0
    ret
truePass:
    mov ch, 1
    ret
comparePass endp

code ends
end start
```

3.2.4 实验步骤

使用 visual studio code 进行源代码的编辑, 将文件命名为 problem4.asm。在 dosbox 虚拟机中使用 masm 对其进行编译, 然后使用 link 对其进行链接, 得到可执行文件 problem4.exe。然后可以使用 td 对其进行调试, 或者直接输入文件名执行该文件。

3.2.5 实验记录与分析

1. 读取用户名和密码的操作

该操作有 3 种可能情况, 分别为用户名和密码正确(如图 1-9 所示)、未输入用户名和密码(如图 1-10 所示)、输入 q 退出程序(如图 1-11 所示)。

汇 编 语 言 程 序 设 计 实 验 报 告

```
C:\>PROBLEM4.EXE
welcome to huao shop
input your name and password
huao
huao
input commodity you want to check: _
```

图 1-9 用户名和密码输入正确

```
C:\>PROBLEM4.EXE
welcome to huao shop
input your name and password

input commodity you want to check:
```

图 1-10 未输入用户名和密码

```
C:\>PROBLEM4.EXE
welcome to huao shop
input your name and password
q

C:\>
```

图 1-11 输入 q, 退出程序

2. 输入待查询的商品名称

输入商品名称有 3 种情况。若不存在该商品，则程序输出“good not found”，并提示用户重新输入待查询的商品名称(如图 1-12 所示)；若存在该商品，则根据用户的类型执行不同的操作，若用户为老板，则程序输出该商品的名称(如图 1-13 所示)，否则，程序输出该商品的推荐度(如图 1-14 所示)，执行完上述操作后，程序重新提醒用户输入用户名和密码。

```
input commodity you want to check:pencil
good not found
input commodity you want to check:
```

图 1-12 未找到待查询商品信息

汇 编 语 言 程 序 设 计 实 验 报 告

```
input commodity you want to check:pen
pen
welcome to huao shop
input your name and password
_
```

图 1-13 老板查询商品

```
input commodity you want to check:pen
A
welcome to huao shop
input your name and password
_
```

图 1-14 顾客查询商品

汇编语言程序设计实验报告

4 总结和体会

本次实验是汇编语言的第一次实验。在本次实验中，我了解了汇编语言程序的生成和执行环境，对汇编语言的使用有了初步的认识。

通过本次实验，我学会了使用虚拟机在 DOS 操作系统下使用 `masm`、`link` 来编译和链接汇编语言源代码，得到相应的可执行文件，并使用 `td` 对生成的可执行文件进行调试。在使用 `td` 调试程序的过程中，通过观察程序执行过程中各寄存器以及标志位的变化，并将其与源代码进行对照，我对汇编语言的指令有了更直观的认识，特别是对于标志位的变化，通过直观的观察程序的执行过程，使我更加明确各标志位的含义。

另外，通过使用 `td` 的反汇编功能从可执行文件的不同位置开始反汇编，让我对计算机执行程序的过程有了更加深刻的理解。可执行文件在计算机中表现为一个二进制串，只有使用正确的方式对其进行解读才能够正确的执行该程序，本次实验让我认识到 `IP/EIP` 寄存器在程序执行过程中的重要意义，也让我对计算机程序的本质有了更深刻的了解。

任务 4 是一个规模较大的汇编程序，在程序中存在大量的分支程序，因此需要执行很多判断和分支跳转指令，在对这些分支进行设计和编写的过程中，我对汇编程序的执行过程的认识更加的深刻了，另外，本程序中有大量的数据存储在内存中，由于很多指令不能够对多个内存同时进行操作，而寄存器的数量有限，我在编写程序的过程中加强了如何有效的使用少量的寄存器完成相应的操作的能力，同时，在对内存的操作中，我更加深刻的认识到了各种寻址方式的使用方法。由于只有少数几个寄存器可以用于变址寻址和基址加变址寻址，在本次实验中，我使用过错误的寄存器用于寻址操作，导致了一些错误，通过修正这些错误，让我对这些寻址方式中寄存器的使用有了更加深刻的印象。

华中科技大学

课程实验报告

课程名称： 汇编语言程序设计实验

实验名称： 实验二 程序优化

实验时间： 2018-4-2, 14: 00-17: 30

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： U201714761

姓 名： 胡澳

同组学生： 无

报告日期： 2018 年 4 月 2 日

原创性声明

本人郑重声明：本报告的内容由本人独立完成，有关观点、方法、数据和文献等的引用已经在文中指出。除文中已经注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品或成果，不存在剽窃、抄袭行为。

特此声明！

学生签名：

日期：2018.4.2

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：

汇编语言程序设计实验报告

目 录

1 实验目的与要求	24
2 实验内容	24
3 实验过程	26
3.1 任务 1.....	26
3.1.1 设计思路.....	26
3.1.2 流程图.....	26
3.1.3 源代码.....	27
3.1.4 实验步骤.....	30
3.1.5 实验记录和分析.....	31
3.2 任务 2.....	32
3.2.1 设计思路.....	32
3.2.2 流程图.....	32
3.2.3 源代码.....	32
3.2.4 实验步骤.....	34
3.2.5 实验记录和分析.....	34
4 总结和体会	36

汇编语言程序设计实验报告

1 实验目的与要求

- (1) 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。
- (2) 熟悉汇编语言指令的特点，掌握代码优化的基本方法。

2 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。

汇编语言程序设计实验报告

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。

3 实验过程

3.1 任务 1

3.1.1 设计思路

在实验一任务 4 的基础上，将商品数量增加至 N(本实验中取 1000)，并将最后一个商品设置为 bag，将其数量设置为 numOfBag(本实验中取 1000)，其余商品全部设置成相同的商品。另外，将商品定义为结构体类型，便于重复定义相同的商品；将计算所有商品的推荐度定义为子程序，以便于在循环结构中调用。在读取到用户输入商品后，判断该商品是否为 bag，若为 bag，则执行 numOfBag 次循环，模拟相应次数的购买，否则，直接按照实验一任务 4 执行。另外，在读取到用户输入商品后开始计时，当程序执行完相应的操作后，终止计时并输出时间。

3.1.2 流程图

程序流程图如图 2-1 所示。

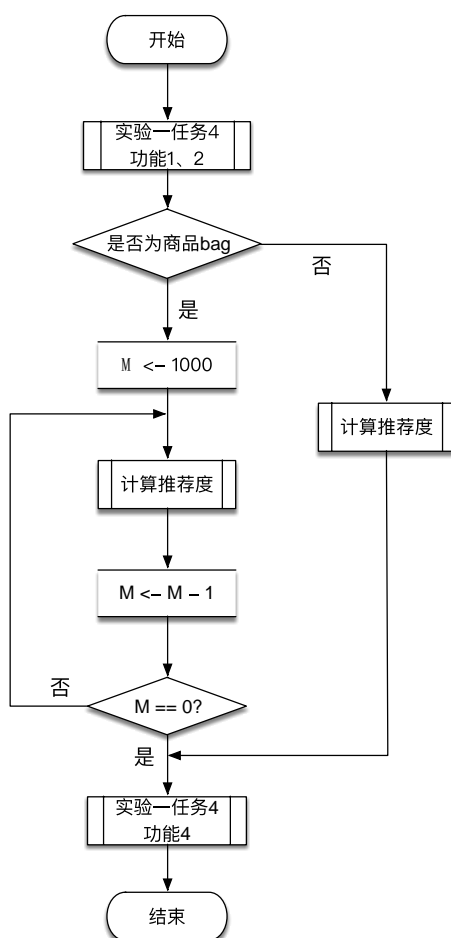


图 2-1 程序流程图

汇编语言程序设计实验报告

3.1.3 源代码

由于本任务基于实验一任务 4，因此本节中仅展示与实验一任务 4 源程序中不同的部分。如下为在实验一基础上增加和修改的部分代码，未改动代码使用“.....”表示。

```
goodStruct struct
    good_name DB 10 dup(0)
    good_len DB 0
    good_sale DB 10
    good_in_price DW ?
    good_out_price DW ?
    good_all_count DW ?
    good_sale_count DW ?
    good_recommamd DW ?
goodStruct ends
..... ; 此处为实验一任务 4 中的代码
ga1 goodStruct <'pen$', 3, 10, 35, 56, 70, 25, ?> ; 将商品使用结构体重新定义
ga2 goodStruct <'book$', 4, 9, 12, 30, 25, 5, ?>

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

bag goodStruct <'bag$', 3, 10, 20, 50, numOfBag, 0, ?>

goodLen EQU $ - bag
..... ; 此处为实验一任务 4 中的代码
mov ax, 0 ; 添加计时和循环部分
call TIMER

call findgood
cmp ch, -1
jz printNotFound

mov ax, [bx].good_all_count

mov cl, auth[0]
cmp cl, 1
jz authOK
jmp authNotOK

authOK:
    mov dx, bx
    mov ah, 9
    int 21h

    lea dx, outenter[0]
```

汇编语言程序设计实验报告

```
mov ah, 9
int 21h

mov ax, 1
call TIMER

lea dx, outenter[0]
mov ah, 9
int 21h

jmp start

authNotOK:
    call calIndex
    cmp ax, numOfBag
    jnz func4
buygood:
    mov cx, [bx].good_sale_count
    add cx, 1
    mov [bx].good_sale_count, cx
    call calIndex
    sub ax, 1
    cmp ax, 0
    jnz buygood

    jmp func4

func4:
    mov ax, 1
    call TIMER

    mov dl, [bx + 20]
    mov ah, 2
    int 21H
    jmp returnStart

returnStart:
    lea dx, outenter[0]
    mov ah, 9
    int 21h

    jmp start

printNotFound:
```

汇编语言程序设计实验报告

```
mov ax, 1
call TIMER
lea dx, notFound[0]
mov ah, 9
int 21h
jmp recommandGood
..... ; 此处为实验一任务 4 中的代码
calIndex proc ; 增加计算推荐度的函数
    pusha

    lea bx, ga1[0]
    sub bx, goodLen

    mov di, N
    inc di

calLoop:
    dec di
    add bx, goodLen
    cmp di, 0
    jz calEnd

    mov ax, [bx].good_in_price
    mov cx, 1280
    mul cx
    push dx
    push ax
    mov al, byte ptr [bx].good_out_price
    mov cl, [bx].good_sale
    mul cl
    mov cx, ax
    pop ax
    pop dx
    div cx
    push ax

    mov ax, [bx].good_sale_count
    mov cx, 64
    mul cx
    mov cx, [bx].good_all_count
    div cx
    mov dx, ax
    pop ax
    add ax, dx
```

汇编语言程序设计实验报告

```
    cmp ax, 100
    jns recommA
    cmp ax, 50
    jns recommB
    cmp ax, 10
    jns recommC
    jmp recommF

recommA:
    push ax
    mov ax, 'A'
    mov [bx + 20], al
    pop ax
    jmp calLoop
recommB:
    push ax
    mov ax, 'B'
    mov [bx + 20], al
    pop ax
    jmp calLoop
recommC:
    push ax
    mov ax, 'C'
    mov [bx + 20], al
    pop ax
    jmp calLoop
recommF:
    push ax
    mov ax, 'F'
    mov [bx + 20], al
    pop ax
    jmp calLoop

calEnd:
    popa
    ret
calIndex endp
.....
```

3.1.4 实验步骤

使用 visual studio code 编辑代码,对实验一任务 4 的代码进行相应的修改,并使用 dosbox 虚拟机中对其进行编译、链接、执行和调试。

汇 编 语 言 程 序 设 计 实 验 报 告

3.1.5 实验记录和分析

1. 当用户输入除 bag 外的其他商品或登录后老板查看各商品时,程序与实验一中相同,此时执行时间极短,难以被记录,因此输出执行时间为 0。其测试如图 2-2 和图 2-3 所示。

```
C:\>EXPR2_1.EXE
welcome to huao shop
input your name and password

input commodity you want to check:book
Time elapsed in ms is 0
B
welcome to huao shop
input your name and password
```

图 2-2 顾客查看除 bag 外的其他商品

```
C:\>EXPR2_1.EXE
welcome to huao shop
input your name and password
huao
huao
input commodity you want to check:bag
bag
Time elapsed in ms is 0

welcome to huao shop
input your name and password
_
```

图 2-3 老板查看商品信息

2. 顾客查看商品 bag 时,程序将模拟抢购情况,此时,程序根据商品种类数量以及 bag 商品的数量的设置情况的不同,将执行次数较多的循环,因此会消耗较长的时间。如图 2-4 所示。

```
C:\>EXPR2_1.EXE
welcome to huao shop
input your name and password

input commodity you want to check:bag
Time elapsed in ms is 10660
A
welcome to huao shop
input your name and password
```

图 2-4 顾客查看商品 bag

3.2 任务 2

3.2.1 设计思路

由于不同的汇编指令的实际执行效率不同、寄存器的读写效率远高于内存的读写效率等影响因素，使用效率较高的指令、高效的使用寄存器、减少内存读写等操作均可以减少程序的运行时间，提高运行效率。

本实验中，使用 inc、dec 等指令替换 add 和 sub 指令，尽可能使用寄存器保存参数而不使用内存保存参数来优化程序运行效率。

3.2.2 流程图

本程序流程图与任务 1 相同，如图 2-1 所示。

3.2.3 源代码

本程序是对任务 1 中程序的修改和优化，因此本节中仅展示与任务 1 源程序中不同的部分。如下为在任务 1 基础上增加和修改的部分代码，未改动代码使用“.....”表示。

本优化中主要针对计算推荐度的函数进行。

```
..... ; 此处为任务 2 中的代码
calIndex proc ; 对计算推荐度的函数进行修改
    pusha

    lea bx, ga1[0]
    sub bx, goodLen

    mov di, N+1
```

汇编语言程序设计实验报告

```
calLoop:
    add bx, goodLen
    dec di
    jz calEnd

    mov ax, [bx].good_in_price
    mov cx, 1280
    mul cx
    mov bp, dx
    mov si, ax
    mov al, byte ptr [bx].good_out_price
    mov cl, [bx].good_sale
    mul cl
    mov cx, ax
    mov dx, bp
    mov ax, si
    div cx

    mov bp, ax
    mov ax, [bx].good_sale_count
    mov cx, 64
    mul cx
    mov cx, [bx].good_all_count
    div cx
    add ax, bp

    cmp ax, 100
    jns recommA
    cmp ax, 50
    jns recommB
    cmp ax, 10
    jns recommC
    jmp recommF

recommA:
    mov bp, 'A'
    mov [bx + 20], bp
    jmp calLoop
recommB:
    mov bp, 'B'
    mov [bx + 20], bp
    jmp calLoop
recommC:
```

汇编语言程序设计实验报告

```
mov bp, 'C'
mov [bx + 20], bp
jmp calLoop
recommF:
mov bp, 'F'
mov [bx + 20], bp
jmp calLoop

calEnd:
popa
ret
calIndex endp
..... ; 此处为任务 2 中的代码
```

3.2.4 实验步骤

使用 visual studio code 编辑代码，对任务 1 的代码进行相应的修改，并使用 dosbox 虚拟机中对其进行编译、链接、执行和调试。

3.2.5 实验记录和分析

分别运行任务 1 中未优化的程序和任务 2 中优化后的程序，获得两者中作为顾客查看商品 bag 所需的执行时间，如图 2-5 和图 2-6 所示。

未优化的程序完成上述过程的时间为 10660ms，优化后的程序完成相同过程的时间为 9060ms，因此优化率为 $\frac{(10660-9060)}{10660} = 15\%$ 。由此可见，通过使用效率更高的汇编指令、尽可能使用寄存器存储数据而不是内存可以有效的提高汇编程序的运行效率。

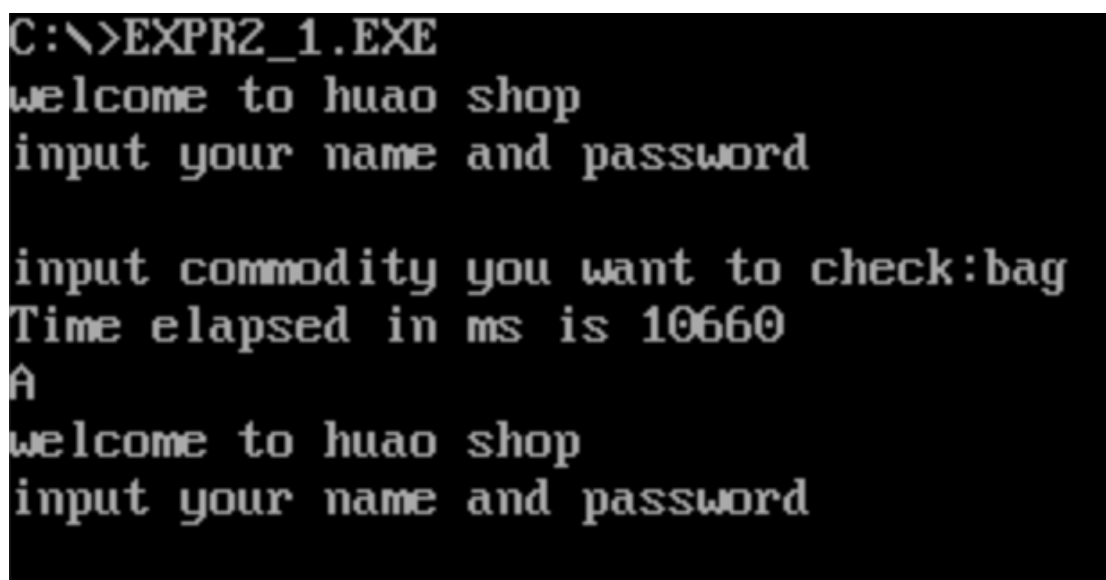
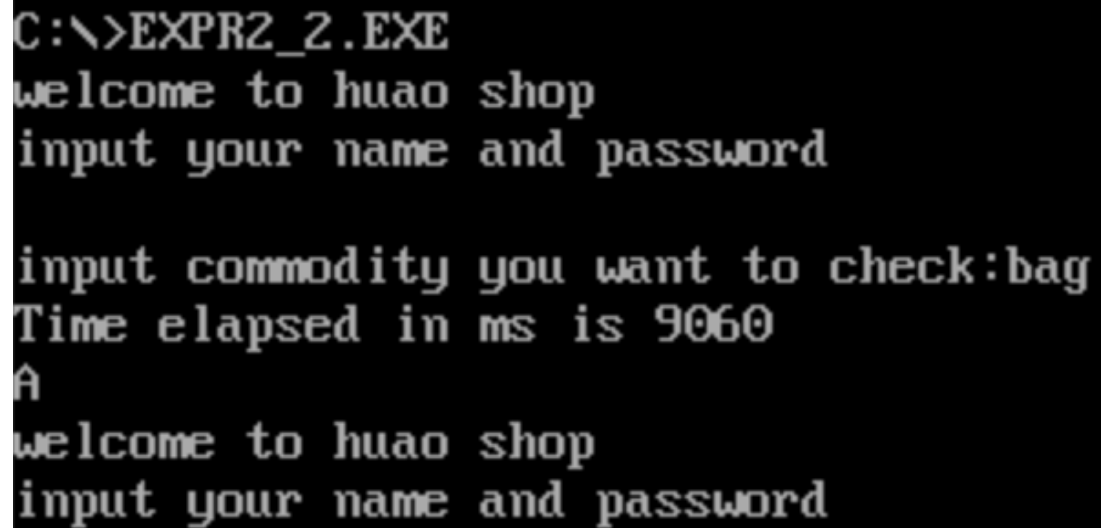


图 2-5 未优化程序运行时间



```
C:\>EXPR2_2.EXE
welcome to huao shop
input your name and password

input commodity you want to check:bag
Time elapsed in ms is 9060
A
welcome to huao shop
input your name and password
```

图 2-6 优化后程序运行时间

汇编语言程序设计实验报告

4 总结和体会

本次实验中，我对汇编程序进行了一定的优化，并得到了较为明显的优化率。通过本次实验，我认识到了不同的汇编指令之间存在的效率的区别，例如乘除法与移位指令之间有时可以实现相同的功能，但是移位指令的效率要远高于乘除法指令。另外，本次实验让我认识到计算机不同的存储层次的读写效率的明显差距，cpu 在处理寄存器中的数据效率要远高于处理内存中数据的效率，由此可见，在编写汇编程序时，合理使用寄存器和内存来存储数据可以有效的提高程序的运行效率。

华中科技大学

课程实验报告

课程名称： 汇编语言程序设计实验

实验名称： 实验三 模块化程序设计

实验时间： 2018-4-9, 14: 00-17: 30 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： U201714761 姓 名： 胡澳

同组学生： 西月栋 报告日期： 2018 年 4 月 16 日

原创性声明

本人郑重声明：本报告的内容由本人独立完成，有关观点、方法、数据和文献等的引用已经在文中指出。除文中已经注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品或成果，不存在剽窃、抄袭行为。

特此声明！

学生签名：

日期：2018.4.16

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：

汇编语言程序设计实验报告

目 录

1 实验目的与要求	39
2 实验内容	39
3 实验过程	42
3.1 任务 1.....	42
3.1.1 设计思路.....	42
3.1.2 流程图.....	42
3.1.3 源代码.....	43
3.1.4 实验步骤.....	54
3.1.5 实验记录和分析.....	54
3.2 任务 2.....	57
3.2.1 设计思路.....	57
3.2.2 流程图.....	57
3.2.3 源代码.....	57
3.2.4 实验步骤.....	63
3.2.5 实验记录和分析.....	63
4 总结和体会	66

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)。（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

汇编语言程序设计实验报告

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）。

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；

汇 编 语 言 程 序 设 计 实 验 报 告

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。

3 实验过程

3.1 任务 1

3.1.1 设计思路

本次实验中，我主要完成了主程序和功能(1)和(2)。主程序中主要实现了对各函数的调度功能，其中，我完成了用于显示菜单的函数 ShowMenu 以及退出程序的函数 ExitProgram，另外，我还设计了函数 InputString 和 ShowString 将 9 号和 10 号中断的功能进行封装，用于对字符串的读取和输出。

主函数中首先调用相应的函数完成欢迎信息的输出以及用户登录的功能，并确定用户类型，然后根据用户类型输出相应的目录，并提示用户选择相应的功能，并将程序跳转到执行相应功能的代码部分，若用户输入非法的字符，则程序将提示用户输入错误并要求其重新输入。

对于功能 1，首先提示用户输入待查找的商品名称，然后将该商品名与已有的商品名逐个比较，直到找到该商品。若找到该商品，则调用 outone 函数(该函数由西月栋设计，用于输出一个商品的全部信息)输出该商品的全部信息，若未找到，则输出错误提示信息，并要求用户重新输入。

对于功能 2，首先与功能 1 相同，找到待修改的商品名称。若找到该商品，则调用 change_good 函数(该函数由西月栋设计，用于修改商品信息)，若未找到，则输出错误提示信息，并要求用户重新输入。

3.1.2 流程图

程序流程图如图 3-1 所示。

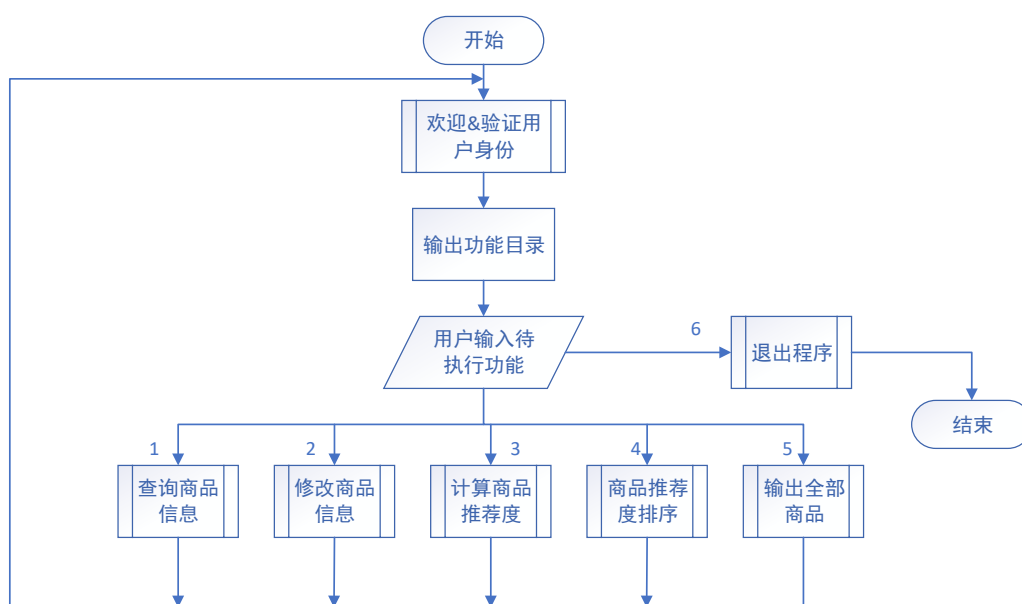


图 3-1 程序流程图

汇编语言程序设计实验报告

3.1.3 源代码

本次实验中，我完成的代码在 main.asm 文件中，其中的源代码如下：

```
extrn outone:far, sort:far, outall:far, change_good:far
public io_1, io_2, io_3, io_4, io_5, io_6, io_7, io_8, BUF1, bufF2T10, DAT, SIGN
.386
```

```
goodStruct struct
    good_nameDB 10 dup(0)
    good_len DB 0
    good_sale DB 10
    good_in_price DW ?
    good_out_price DW ?
    good_all_count DW ?
    good_sale_count DW ?
    good_recommamd DW ?
    good_sort DW 1
goodStruct ends
```

```
data segment use16
    bname DB 'huao', 6 dup(0), 4 ;boss name
    bpassDB 'huao', 6 dup(0), 4 ;password
    N EQU 10 ;number
    shopname DB 'huao$' ;shop name
```

```
auth DB ?
numOfBag EQU 1000
```

```
ga1 goodStruct <'pen$', 3, 10, 35, 56, 70, 25, ?>
ga2 goodStruct <'book$', 4, 9, 12, 30, 25, 5, ?>
goodStruct N-3 dup(<'egg$', 3, 7, 35, 56, 70, 25, ?>)
bag goodStruct <'bag$', 3, 10, 20, 50, numOfBag, 0, ?>
goodLen EQU $ - bag
welcomeMsg DB 'welcome to ', '$'
welcomeShopMsg DB 'shop$'
loginErrorMsg DB 'Login Error', 0ah, 0dh, '$'
outenter DB 0ah, 0dh, '$'
info DB 'input your name and password', 0ah, 0dh, '$'
in_name DB 10, ?, 10 dup(0), '$'
in_pass DB 6, ?, 6 dup(0), '$'
checkname DB 'input commodity you want to check:', '$'
goodname DB 10, ?, 10 dup(0)
notFound DB 'good not found', 0ah, 0dh, '$'
menu1 DB '1. Query product information', 0ah, 0dh, '$'
menu2 DB '2. Edit product information', 0ah, 0dh, '$'
```

汇编语言程序设计实验报告

```
menu3      DB  '3. Calculate recommendation', 0ah, 0dh, '$'
menu4      DB  '4. Calculate recommendation ranking', 0ah, 0dh, '$'
menu5      DB  '5. Output all product information', 0ah, 0dh, '$'
menu6      DB  '6. exit the program', 0ah, 0dh, '$'
```

```
errorMsg DB 'Input Error. Input again.', 0ah, 0dh, '$'
```

```
io_1 db 'good_name is: $'
io_2 db 'good_sale is: $'
io_3 db 'good_in_price is: $'
io_4 db 'good_out_price is: $'
io_5 db 'good_all_count is: $'
io_6 db 'good_sale_count is: $'
io_7 db 'good_recommand is: $'
io_8 db 'good_sort is: $'
```

```
BUF1 db 10, ?, 10 dup(0)
buff2T10 db 12 dup(0)
DAT DW 0AH
SIGN DB ?
data ends
```

```
stack segment use16 stack
    DB 200 dup(0)
stack ends
```

```
code segment use16
WelcomeFunc PROTO near stdcall
InputString PROTO near stdcall stringInputAddr:WORD
ShowMenu PROTO near stdcall userType:BYTE
ShowString PROTO near stdcall stringAddr:WORD
ExitProgram PROTO near stdcall
quaryGood proto near stdcall
```

```
start:
    mov ax, data
    mov ds, ax
    invoke WelcomeFunc
    mov bx, offset auth
```

```
chooseFunc:
    invoke ShowMenu, ds:[bx]
    mov ah, 1
    int 21h
```

汇编语言程序设计实验报告

```
mov bx, offset outenter
```

```
invoke ShowString, bx
```

```
;此时 al 中为功能数
```

```
cmp al, '1'
```

```
jz func1
```

```
cmp al, '2'
```

```
jz func2
```

```
cmp al, '3'
```

```
jz func3
```

```
cmp al, '4'
```

```
jz func4
```

```
cmp al, '5'
```

```
jz func5
```

```
cmp al, '6'
```

```
jz func6
```

```
mov bx, offset errorInputMsg
```

```
invoke ShowString, bx
```

```
jmp chooseFunc
```

```
func1:
```

```
invoke quarryGood
```

```
jmp start
```

```
func2:
```

```
push di
```

```
mov bx, offset checkname
```

```
invoke ShowString, bx
```

```
mov bx, offset goodname
```

```
invoke InputString, bx
```

```
mov bx, offset outenter
```

```
invoke ShowString, bx
```

```
mov si, offset goodname
```

```
mov cl, [si + 1]
```

```
cmp cl, 0
```

```
jz start
```

```
call findgood
```

```
cmp ch, -1
```

```
jz printNotFound2
```

```
mov di, bx
```

```
call change_good
```

```
pop di
```

汇编语言程序设计实验报告

```
    jmp start

printNotFound2:
    mov dx, offset notFound
    invoke ShowString, dx
    jmp func2

func3:
    call calIndex
    jmp start

func4:
    mov ax, N
    mov si, offset ga1
    call sort
    jmp start

func5:
    mov ax, N
    mov si, offset ga1
    call outall
    jmp start

func6:
    invoke ExitProgram
quaryGood proc near stdcall
quaryGoodBegin:
    mov bx, offset checkname
    invoke ShowString, bx

    mov bx, offset goodname
    invoke InputString, bx

    mov bx, offset outenter
    invoke ShowString, bx

    mov si, offset goodname
    mov cl, [si + 1]
    cmp cl, 0
    jz EndJmpStart

    call findgood
    cmp ch, -1
    jz printNotFound

    mov si, offset auth
    mov cl, [si]
```


汇编语言程序设计实验报告

```
cmp cl, 1
jz authOK
jmp authNotOK
```

```
authOK:
    push si
    mov si, bx
    call outone
    pop si
    jmp EndJmpStart
```

```
authNotOK:
    push si
    mov si, bx
    call outone
    pop si
    jmp EndJmpStart
```

```
printNotFound:
    mov dx, offset notFound
    invoke ShowString, dx
    jmp quarryGoodBegin
EndJmpStart:
    ret
quarryGood endp
```

WelcomeFunc PROC near stdcall uses ax bx dx

```
welcomeStart:
    mov ax, offset welcomeMsg
    invoke ShowString, ax
    mov ax, offset shopname
    invoke ShowString, ax
    mov ax, offset welcomeShopMsg
    invoke ShowString, ax
    mov ax, offset outenter
    invoke ShowString, ax
    mov ax, offset info
    invoke ShowString, ax
    mov ax, offset in_name
    invoke InputString, ax
    mov ax, offset outenter
    invoke ShowString, ax
;下面将读入的回车符号置为 0
    mov si, offset in_name
```

汇编语言程序设计实验报告

```
mov al, ds:[si + 1]
mov ah, 0
mov bx, ax
mov ax, 0
mov ds:[si + bx + 2], al
```

```
mov al, ds:[si + 1]
cmp al, 0
jz welcomeUser
cmp al, 1
jz checkQuit
jmp next3
```

```
checkQuit:
mov al, ds:[si + 2]
cmp al, 71h
invoke ExitProgram
```

```
next3:
mov dx, offset in_pass
invoke InputString, dx
```

```
mov si, offset in_pass
mov al, ds:[si + 1]
mov ah, 0
mov bx, ax
mov ax, 0
mov ds:[si + bx + 2], al
```

```
mov dx, offset outenter
invoke ShowString, dx
```

```
lea bx, in_name[2]
mov si, offset in_name
mov al, ds:[si + 1]
mov si, offset bname
cmp al, ds:[si + 10]
jnz errorLogin
call compareName
cmp ch, 1
jnz errorLogin
```

```
lea bx, in_pass[2]
mov si, offset in_pass
```

汇编语言程序设计实验报告

```
mov al, ds:[si + 1]
    mov si, offset bpass
cmp al, ds:[si + 10]
jnz errorLogin
call comparePass
cmp ch, 1
jnz errorLogin
```

```
mov al, 1
    mov si, offset auth
mov [si], al
ret
```

```
errorLogin:
lea dx, loginErrorMsg[0]
mov ah, 9
int 21h
jmp welcomeStart
```

```
welcomeUser:
    mov al, 0
    mov si, offset auth
mov [si], al
ret
```

WelcomeFunc ENDP

InputString PROC near stdcall uses dx ax, stringInputAddr:WORD

```
mov dx, stringInputAddr
mov ah, 10
int 21h
ret
```

InputString ENDP

ShowMenu PROC near stdcall uses ax, userType:BYTE

```
mov ax, offset menu1
invoke ShowString, ax
cmp userType, 1
jnz user
mov ax, offset menu2
invoke ShowString, ax
mov ax, offset menu3
invoke ShowString, ax
mov ax, offset menu4
```

汇编语言程序设计实验报告

```
    invoke ShowString, ax
    mov ax, offset menu5
    invoke ShowString, ax
user:
    mov ax, offset menu6
    invoke ShowString, ax
    ret
ShowMenu ENDP

ShowString PROC near stdcall uses dx ax, stringAddr:WORD
    mov dx, stringAddr
    mov ah, 9
    int 21h
    ret
ShowString ENDP

ExitProgram PROC near stdcall
    mov ax, 4C00h
    int 21h
ExitProgram ENDP

compareName proc
    push si
compName:
    cmp al, 0
    jz trueName
    dec al
    mov ah, 0
    mov di, ax
        mov si, offset bname
        add si, di
    mov dl, [si]
    sub dl, [bx + di]
    jz compName

    mov ch, 0
    pop si
    ret
trueName:
    mov ch, 1
    pop si
    ret
compareName endp
```

汇编语言程序设计实验报告

```
comparePass proc
    push si
compPass:
    cmp al, 0
    jz truePass
    dec al
    mov ah, 0
    mov di, ax
        mov si, offset bpass
        add si, di
    mov dl, [si]
    sub dl, [bx + di]
    jz compPass

    mov ch, 0
    pop si
    ret
truePass:
    mov ch, 1
    pop si
    ret
comparePass endp

findgood proc
    push di
        mov di, offset goodname
    mov bl, [di + 1]
    mov bh, 0
    mov si, bx

        mov di, offset gal
    lea bx, [di]
    sub bx, goodLen

    mov dx, N
    inc dx

goodLoop:
    dec dx
    add bx, goodLen
    cmp dx, 0
    jz next2
    mov al, [bx + 10]
    mov ah, 0
```

汇编语言程序设计实验报告

```
    cmp ax, si
    jnz goodLoop
    call compare
    cmp ch, 1
    jz foundgood
    jmp goodLoop

next2:
    mov ch, -1
    pop di
    ret

foundgood:
    mov ch, bh
    pop di
    ret
findgood endp

compare proc
    push si
comp:
    cmp al, 0
    jz found
    dec al
    mov ah, 0
    mov di, ax
    mov si, offset goodname
    add si, di
    mov dl, [si + 2]
    sub dl, [bx + di]
    jz comp

    mov ch, 0
    pop si
    ret
found:
    mov ch, 1
    pop si
    ret
compare endp

calIndex proc
    pusha
```

汇编语言程序设计实验报告

```
    mov si, offset gal
    lea bx, [si]
    sub bx, goodLen

    mov di, N
    inc di

calLoop:
    dec di
    add bx, goodLen
    cmp di, 0
    jz calEnd

    mov ax, [bx].good_in_price
    mov cx, 1280
    mul cx
    push dx
    push ax
    mov al, byte ptr [bx].good_out_price
    mov cl, [bx].good_sale
    mul cl
    mov cx, ax
    pop ax
    pop dx
    div cx
    push ax

    mov ax, [bx].good_sale_count
    mov cx, 64
    mul cx
    mov cx, [bx].good_all_count
    div cx
    mov dx, ax
    pop ax
    add ax, dx

    cmp ax, 100
    jns recommA
    cmp ax, 50
    jns recommB
    cmp ax, 10
    jns recommC
    jmp recommF
```

汇编语言程序设计实验报告

```
recommA:
    push ax
    mov ax, 'A'
    mov [bx + 20], al
    pop ax
    jmp calLoop
recommB:
    push ax
    mov ax, 'B'
    mov [bx + 20], al
    pop ax
    jmp calLoop
recommC:
    push ax
    mov ax, 'C'
    mov [bx + 20], al
    pop ax
    jmp calLoop
recommF:
    push ax
    mov ax, 'F'
    mov [bx + 20], al
    pop ax
    jmp calLoop

calEnd:
    popa
    ret
calIndex endp
code ends
    end start
```

3.1.4 实验步骤

使用 visual studio code 编写代码，并在 dosbox 中完成相应的编译、链接和调试过程。本实验中包含 3 个源文件，分别为 main.asm, sort.asm 以及 F10T2W.asm。将三个文件使用 masm 分别编译，得到三个.obj 的文件，然后将这三个 obj 文件使用工具 link 链接起来，得到可执行文件，使用 td 工具打开该文件即可对其进行调试过程。

3.1.5 实验记录和分析

1. 对菜单功能的测试

当用户输入正确的用户名和密码时，程序将给出功能菜单，显示 6 个功能，当用户为输入用户名和密码时，程序将给出 2 个功能的功能菜单，其测试如图 3-2 和图 3-3 所示。


```
C:\>MAIN.EXE
welcome to huao shop
input your name and password
huao
huao
1. Query product information
2. Edit product information
3. Calculate recommendation
4. Calculate recommendation ranking
5. Output all product information
6. exit the program
```

图 3-2 老板查看的功能菜单

```
C:\>MAIN.EXE
welcome to huao shop
input your name and password

1. Query product information
6. exit the program
```

图 3-3 用户查看的功能菜单

2. 查询商品信息功能测试

当用户选择功能 1 时，程序执行查询商品信息的功能，当用户输入一个存在的商品名称时，程序将输出该商品的全部信息，若该商品不存在，则程序提醒用户该商品不存在，并要求其重新输入。其测试截图如图 3-4 和图 3-5 所示。

```
1
input commodity you want to check:pen
good_name is: pen
good_sale is: 10
good_in_price is: 35
good_out_price is: 56
good_all_count is: 70
good_sale_count is: 25
good_recommand is: 0
good_sort is: 1
welcome to huao shop
input your name and password
```

图 3-4 当商品存在时，查询商品信息

汇编语言程序设计实验报告

```
1
input commodity you want to check:pencil
good not found
input commodity you want to check:_
```

图 3-5 当商品不存在时，查询商品信息

3. 修改商品信息功能测试

当用户选择功能 2 时，程序将执行修改商品信息的功能，当该商品存在时，程序将逐个的输出该商品的信息并要求用户输入该信息修改后的值，若该商品不存在，则程序将输出该商品不存在并提示用户重新输入商品名称，其测试截图如图 3-6 和 3-7 所示。

```
2
input commodity you want to check:pen
good_sale is: 10 20
good_in_price is: 35 30
good_out_price is: 56 40
good_all_count is: 70 50
welcome to huao shop
input your name and password
```

图 3-6 商品存在时，修改商品信息

```
2
input commodity you want to check:pencil
good not found
input commodity you want to check:
```

图 3-7 商品不存在时，修改商品信息

4. 退出程序功能测试

当用户选择功能 6 时，程序将退出，其测试截图如图 3-8 所示。

```
welcome to huao shop
input your name and password

1. Query product information
6. exit the program
6

C:\>_
```

图 3-8 退出程序功能测试截图

汇编语言程序设计实验报告

3.2 任务 2

3.2.1 设计思路

此任务中，使用 C 语言改写任务一中的汇编主程序，并将其中查找商品的功能使用 C 语言函数实现，另外，由于本次任务在 windows 环境下实现，因此使用 windows api 函数以及 C 语言库函数替代原本在任务一中使用的 dos 中断功能。

3.2.2 流程图

此任务中程序的流程与任务一中的程序相同，仅仅对其中的实现方法进行了相应的修改，因此程序流程图与任务一相同，见图 3-1。

3.2.3 源代码

此任务中，对原任务一中的汇编函数修改较少，其中修改部分如下。

1. 添加对待使用 C 语言函数及 windows api 的声明

```
printf proto cdecl :dword
scanf proto cdecl :dword, :dword
strlen proto cdecl :dword
ExitProcess proto stdcall :dword
putchar proto cdecl :byte
```

2. 对输入输出函数的修改

此任务中使用 C 语言 printf、putchar 和 scanf 函数实现输入输出功能，以替代 dos 下的 9 号和 10 号中断功能。因此对函数 InputString、宏 gets、宏 puts、宏 CRLF 以及函数 showString 做如下修改。

```
ShowString PROC stdcall stringAddr:DWORD
    invoke printf, stringAddr
    ret
ShowString ENDP
```

```
InputString PROC stdcall uses edx eax, stringInputAddr:DWORD
    mov eax, stringInputAddr
    lea edx, scanfArgStr
    add eax, 2
    invoke scanf, edx, eax
    mov dl, 10
    mov eax, stringInputAddr
    mov [eax], dl
    add eax, 2
    invoke strlen, eax
    mov edx, stringInputAddr
    mov [edx + 1], al
    mov [edx + 12], 0
```

汇编语言程序设计实验报告

```
ret
InputString ENDP
puts macro offset:req
push eax
mov eax, offset offset
invoke printf, eax
pop eax
endm

;input change cx si,cx is length, si is addr
gets macro offset:req
lea esi, offset
invoke InputString, esi
mov cl, [esi + 1]
movsx cx, cl
add esi, 2
endm

CRLF macro
invoke putchar, 0ah
invoke putchar, 0dh
endm
```

3. 对退出程序函数的修改

程序退出函数使用 windows api 完成而不是 dos 中断，更改后的函数如下

```
ExitProgram PROC
invoke ExitProcess, 0
ret
ExitProgram ENDP
```

本次任务中添加了 C 语言程序，其源代码如下

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define N 10          //商品数量
#define numOfBag 1000

typedef struct goodStruct {
    char good_name[10];
    char good_len;
    char good_sale;
    short good_in_price;
    short good_out_price;
    short good_all_count;
    short good_sale_count;
```

汇编语言程序设计实验报告

```
short good_recommand;
short good_sort;
} goodStruct;

char bname[11] = "huao";
char bpass[11] = "huao";
char shopname[] = "huao";
char auth;
goodStruct good[N];
int goodLen = sizeof(goodStruct);
char welcomeMsg[] = "welcome to ";
char welcomeShopMag[] = " shop";
char loginErrorMsg[] = "Login Error\n";
char outenter[] = "\n";
char info[] = "input your name and password\n";
char in_name[13];
char in_pass[9];
char checkname[] = "input commodity you want to check:";
char goodname[12];
char notFound[] = "good not found\n";
char menu1[] = "1. Query product information\n";
char menu2[] = "2. Edit product information\n";
char menu3[] = "3. Calculate recommendation\n";
char menu4[] = "4. Calculate recommendation ranking\n";
char menu5[] = "5. Output all product information\n";
char menu6[] = "6. exit the program\n";

char errorInputMsg[] = "Input Error. Input again.\n";

char bufF2T10[12];
short DAT = 10;
char SIGN;

extern "C" void _cdecl WelcomeFunc(char *welcomeMsg, char *shopname, char *welcomeShopMsg, char
*outenter, char *info, char *in_name, char *in_pass, char *bname, char *bpass, char *auth, char *loginErrorMsg);

extern "C" void _cdecl ShowMenu(char auth, char *menu1, char *menu2, char *menu3, char *menu4, char
*menu5, char *menu6);;

extern "C" void _cdecl callIndex(void *goodAddr, int goodLen, short goodNum);;

extern "C" void _cdecl change_good(char *goodAddr, char *SIGN, short *DAT);

extern "C" void _cdecl outone(char *goodAddr);
```

汇编语言程序设计实验报告

```
extern "C" void _cdecl outall(short goodSum, void *goodAddr);

extern "C" void _cdecl sort(short goodSum, void *goodAddr);

extern "C" void _cdecl ExitProgram(void);

void init(void);
char *findGood(char *goodName);

int main(void)
{
    init();
    char temp;
    int funcNum;
    char *goodAddr;
    while (1)
    {
        WelcomeFunc(welcomeMsg, shopname, welcomeShopMag, outenter, info, in_name, in_pass, bname,
bpass, &auth, loginErrorMsg);
        ShowMenu(auth, menu1, menu2, menu3, menu4, menu5, menu6);

        printf("Please choose a function:");
        scanf("%d", &funcNum);
        switch (funcNum)
        {
            case 1:
                while (1)
                {
                    printf(checkname);
                    scanf("%s", goodname);
                    goodAddr = findGood(goodname);
                    if (goodAddr == NULL)
                    {
                        printf(notFound);
                    }
                    else
                    {
                        break;
                    }
                }
                outone(goodAddr);
                break;
            case 2:
```

汇编语言程序设计实验报告

```
        while (1)
        {
            printf(checkname);
            scanf("%s", goodname);
            goodAddr = findGood(goodname);
            if (goodAddr == NULL)
                printf(notFound);
            else
                break;
        }
        change_good(goodAddr, &SIGN, &DAT);
        break;
case 3:
    calIndex(good, goodLen, N);
    break;
case 4:
    sort(N, good);
    break;
case 5:
    outall(N, good);
    break;
case 6:
    ExitProgram();
    break;
default:
    printf("input error, please input again\n");
    break;
    }
}
printf("\n");
scanf("%c", &temp);
getchar();
strlen(bname);
return 0;
}

char *findGood(char *goodName)
{
    void *goodAddr = good;
    for (int i = 0; i < N; i++)
    {
        if (strcmp(goodName, good[i].good_name) == 0)
        {
            return (char *)goodAddr + i * goodLen;
        }
    }
}
```

汇编语言程序设计实验报告

```
    }  
}  
return NULL;  
}  
  
void init(void) {  
    bname[10] = 4;  
    bpass[10] = 4;  
    strcpy(good[0].good_name, "pen");  
    good[0].good_len = 3;  
    good[0].good_sale = 10;  
    good[0].good_in_price = 35;  
    good[0].good_out_price = 56;  
    good[0].good_all_count = 70;  
    good[0].good_sale_count = 25;  
    good[0].good_sort = 1;  
    strcpy(good[1].good_name, "book");  
    good[1].good_len = 4;  
    good[1].good_sale = 9;  
    good[1].good_in_price = 12;  
    good[1].good_out_price = 30;  
    good[1].good_all_count = 25;  
    good[1].good_sale_count = 5;  
    good[1].good_sort = 1;  
    for (int i = 2; i < N - 1; i++)  
    {  
        strcpy(good[i].good_name, "egg");  
        good[i].good_len = 3;  
        good[i].good_sale = 7;  
        good[i].good_in_price = 35;  
        good[i].good_out_price = 56;  
        good[i].good_all_count = 70;  
        good[i].good_sale_count = 25;  
        good[i].good_sort = 1;  
    }  
    strcpy(good[N - 1].good_name, "bag");  
    good[N - 1].good_len = 3;  
    good[N - 1].good_sale = 10;  
    good[N - 1].good_in_price = 20;  
    good[N - 1].good_out_price = 50;  
    good[N - 1].good_all_count = numOfBag;  
    good[N - 1].good_sale_count = 0;  
    good[N - 1].good_sort = 1;  
}
```


汇编语言程序设计实验报告

3.2.4 实验步骤

本次实验在 windows 环境下完成，使用 visual studio 完成了代码的编写、C 语言与汇编语言的链接以及代码的运行和调试过程。本次任务中一共包含四个文件，分别为 main.cpp, func1.asm, sort.asm 以及 F10T2.asm，其中 main.cpp 为 C 语言源程序，func1.asm 包含了任务一中 main.asm 中包含的函数，sort.asm 和 F10T2.asm 与任务一中的文件类似，仅修改了部分与系统相关的内容，具体的修改部分见 3.2.3 节。

首先，在 windows 的控制台中使用 ml /I . /Zm /c /Ta [文件名]对汇编源文件进行编译，得到.obj 的文件，然后将得到的三个.obj 文件添加到 visual studio 的项目中，使用 visual studio 生成解决方案，即可得到相应的可执行文件。

另外，在 visual studio 中使用调试工具可直接对该可执行文件进行汇编指令级的调试。

3.2.5 实验记录和分析

本次实验中主要实现了对六个功能的模块化设计，对六个功能的测试结果如下。

如图 3-9 和 3-10 为对查询商品功能的测试截图。其中图 3-9 为商品存在时的情况，图 3-10 为商品不存在的情况。

```
Please choose a function:1
input commodity you want to check:pen
good_name is: pen
good_sale is: 10
good_in_price is: 35
good_out_price is: 56
good_all_count is: 70
good_sale_count is: 25
good_recommand is: 0
good_sort is: 1
welcome to huao shop
input your name and password
```

图 3-9 商品存在时，查询商品功能测试

```
Please choose a function:1
input commodity you want to check:pencil
good not found
input commodity you want to check:
```

图 3-10 商品不存在时，查询商品功能测试

如图 3-11 为修改商品信息功能的测试截图。

汇编语言程序设计实验报告

```
Please choose a function:2
input commodity you want to check:pen
good_sale is: 10 20
good_in_price is: 35 30
good_out_price is: 56 40
good_all_count is: 70 50
welcome to huao shop
input your name and password
```

图 3-11 修改商品信息功能测试

执行功能 3、4 后，程序没有明确输出，但是已对内存中相应的数据进行了修改，当程序执行功能 3 和 4 后，再次调用功能 1，可以得知，程序对商品的推荐度做了计算并根据推荐度给出了各商品的排名，再次调用功能 1 后的程序截图如图 3-12 所示。

```
Please choose a function:1
input commodity you want to check:pen
good_name is: pen
good_sale is: 20
good_in_price is: 30
good_out_price is: 40
good_all_count is: 50
good_sale_count is: 25
good_recommand is: 66
good_sort is: 1
welcome to huao shop
input your name and password
```

图 3-12 调用功能 3、4 后调用功能 1 的输出截图


如图 3-13 为输出全部商品信息的测试截图。

```
good_all_count is: 70
good_sale_count is: 25
good_recommand is: 65
good_sort is: 4
good_name is: egg
good_sale is: 7
good_in_price is: 35
good_out_price is: 56
good_all_count is: 70
good_sale_count is: 25
good_recommand is: 65
good_sort is: 4
good_name is: bag
good_sale is: 10
good_in_price is: 20
good_out_price is: 50
good_all_count is: 1000
good_sale_count is: 0
good_recommand is: 66
good_sort is: 1
welcome to huao shop
input your name and password
```

图 3-13 输出全部商品信息功能测试

汇 编 语 言 程 序 设 计 实 验 报 告

如图 3-14 为退出程序功能测试。选择功能 6 后，程序直接退出，并返回到控制台。



```
6. exit the program
Please choose a function:6
PS C:\Users\胡澳\Desktop\expr3_1\expr3_1\Debug>
```

图 3.14 退出程序功能测试

4 总结和体会

在本次实验的任务一中，我和小组成员西月栋各自完成了源代码的编写，在编写源代码以及各自编译的过程中，由于前几次的实验基础，我们可以很好的完成，但是在将文件链接的时候，我们遇到了很多问题。例如在多个文件中均定义了数据段，在链接后访问数据段元素时导致访问位置与预期不相符以及调用另一个文件中的函数时出现的问题，在参阅课本并且做出相应的尝试后，我们最终解决了这些问题，并完成了任务一的要求。

通过完成任务一，我对汇编语言的链接有了更多的认识，同时也对汇编语言编译过程中对变量、函数等的处理有了更深刻的了解。

在任务二中，我同样遇到了很多的链接时出现的错误，在经过对代码的仔细分析和修正后，最终将这些问题一一解决。在这一过程中，我直观的认识到了不同的函数调用方式对于出现执行方式的影响，同时也对 C 语言的编译过程以及编译结果有了新的认识，在对链接错误的修改过程中，我发现即使 C 语言中包含了 `stdio.h` 中所有函数的声明，但是只有其中被使用过的函数的定义会被编译并出现在 `.obj` 文件中，对于未被使用的函数，则编译器并不会处理它的定义部分。

通过本次实验，我对 C 语言和汇编语言的编译、链接过程有了更深入的认识，也给我在编写代码的过程中避免出错提供了大量的经验；另外，本次实验也让我对模块化程序设计有了更加正确的认识，让我了解到了很多在分模块编写程序时应该注意的问题。

华中科技大学

课程实验报告

课程名称： 汇编语言程序设计实验

实验名称： 实验四 中断与反跟踪

实验时间： 2018-4-23, 14: 00-17: 30

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： U201714761

姓 名： 胡澳

同组学生： 西月栋

报告日期： 2018 年 4 月 23 日

原创性声明

本人郑重声明：本报告的内容由本人独立完成，有关观点、方法、数据和文献等的引用已经在文中指出。除文中已经注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品或成果，不存在剽窃、抄袭行为。

特此声明！

学生签名：

日期：2018.4.23

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：

汇编语言程序设计实验报告

目 录

1 实验目的与要求	69
2 实验内容	69
3 实验过程	71
3.1 任务 1.....	71
3.1.1 设计思路.....	71
3.1.2 源代码.....	71
3.1.3 实验步骤.....	72
3.1.4 实验记录与分析.....	72
3.2 任务 2.....	75
3.2.1 设计思路.....	75
3.2.2 源代码.....	75
3.2.3 实验步骤.....	76
3.2.4 实验记录与分析.....	77
3.3 任务 3.....	77
3.3.1 设计思路.....	77
3.3.2 源代码.....	78
3.3.3 实验步骤.....	79
3.3.4 实验记录与分析.....	79
3.4 任务 4.....	80
3.4.1 设计思路.....	80
3.4.2 源代码.....	80
3.4.3 实验步骤.....	82
3.4.4 实验记录与分析.....	82
3.5 任务 5.....	84
3.5.1 设计思路.....	84
3.5.2 实验步骤.....	84
3.5.3 实验记录与分析.....	84
4 总结和体会	86

汇编语言程序设计实验报告

1 实验目的与要求

- (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。

2 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 TD 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

汇编语言程序设计实验报告

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。

3 实验过程

3.1 任务 1

3.1.1 设计思路

使用三种方法获取中断类型码 1H 和 13H 对应的中断处理程序的入口地址，三种方法的思路如下。

(1) 直接在 TD 调试工具中查看内存内容。中断矢量表存储在物理地址 00000~003FFH 的位置，每个中断的入口地址占 4 个字节，因此，中断 1H 的入口地址存储在 0000:[0004](即 00004)处，中断 13H 的入口地址存储在 0000:[004CH](即 0004C)处。在 TD 中，首先执行代码 “mov ax, 0”和 “mov ds, ax”，将 ds 寄存器赋值为 0，然后查看内存 ds:[0004]和 ds:[004CH]处的内容，即为相应的中断处理程序的入口地址。

(2) 使用 dos 功能调用获取中断处理程序的入口地址。dos 功能调用中的 35H 号功能为获取中断处理程序的入口地址，编写程序调用 35H 号功能即可在相应的寄存器中获取到待获得的地址。

(3) 编写程序获取内存空间的内容。编写程序将中断矢量表中相应位置的内容复制到寄存器中，即可观察到中断矢量表中的内容，从而得知相应中断的入口地址。

3.1.2 源代码

方法 2 和方法 3 编写了相应的程序，其程序代码如下。

```
; method 2
.386
code segment use16
start:
    mov ah, 35h
    mov al, 1h
    int 21h
    mov ah, 35h
    mov al, 13h
    int 21h
code ends
stack segment use16 stack
    DB 200 dup(0)
stack ends
end start

; method 3
.386
stack segment use16 stack
    db 200 dup(0)
```

汇编语言程序设计实验报告

```
stack ends
code segment use16
start:
    mov ax, 0
    mov ds, ax

    mov es, ds:[6]
    mov bx, ds:[4]

    mov es, ds:[4EH]
    mov bx, ds:[4CH]

code ends
end start
```

3.1.3 实验步骤

1. 方法 1 步骤

在 dosbox 中打开 td 工具，首先在代码区输入两行代码 “mov ax, 0”和 “mov ds, ax”，依次执行这两行代码，然后将内存区中查看的内存修改为 ds:[0004]和 ds:[004CH]，依次查看这两个内存地址中的内容。

2. 方法 2 步骤

在 dosbox 中，使用 masm 和 link 工具对汇编源文件 task1_2.asm 进行编译链接，得到可执行文件 task1_2.exe，使用 td 对该文件进行调试，单步执行到相应的位置后观察 es 和 bx 寄存器中的内容，即为相应的中断服务程序的段地址和偏移地址。

3. 方法 3 步骤

在 dosbox 中，使用 masm 和 link 工具对汇编源文件 task1_3.asm 进行编译链接，得到可执行文件 task1_3.exe，使用 td 对该文件进行调试，单步执行到相应的位置后观察由相应内存的内容赋值的寄存器的值，即为相应的中断服务程序的段地址和偏移地址。

3.1.4 实验记录与分析

1. 方法 1 中查看到的内存内容如图 3-1 和图 3-2 所示。

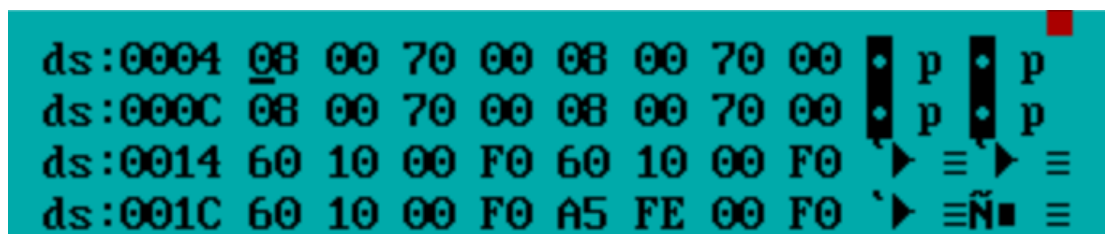


图 3-1 内存 0000:[0004]处的内容

汇编语言程序设计实验报告

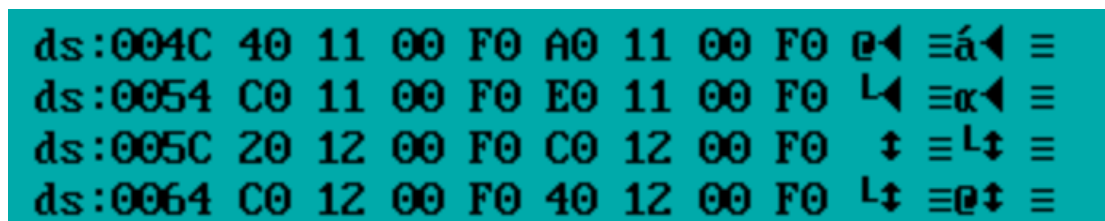


图 3-2 内存 0000:[004CH]处的内容

由图可知，1H 中断的中断服务程序的段地址为 0070H，偏移地址为 0008H；13H 中断的中断服务程序的段地址为 0F000H，偏移地址为 1140H。

2. 方法 2 对 1H 和 13H 分别调用 35H 号功能后的结果如图 3-3 和图 3-4 所示。

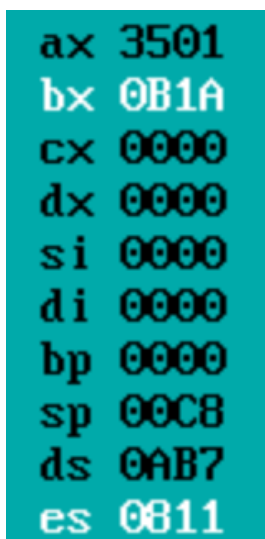


图 3-3 对 1H 调用 35H 号功能得到其段地址 es 和偏移地址 bx

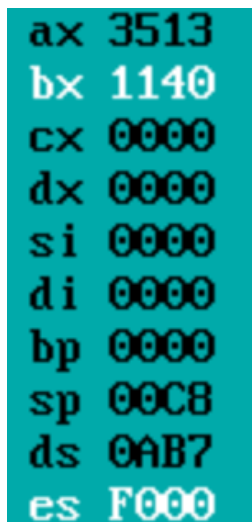


图 3-4 对 13H 调用 35H 号功能得到其段地址 es 和偏移地址 bx

由图可知，1H 中断的中断服务程序的段地址为 0811H，偏移地址为 0B1AH；13H 中断的中断服务程序的段地址为 0F000H，偏移地址为 1140H。

3. 方法 3 将方法 1 中直接查看的内存内容赋值给相应的寄存器(本实验中使用 es 和 bx)，对 1H 和 13H 的操作结果如图 3-4 和图 3-6 所示。

```
ax 0000
bx 0B1A
cx 0000
dx 0000
si 0000
di 0000
bp 0000
sp 00C8
ds 0000
es 0811
```

图 3-5 将 0000:[0004]开始的两个字分别赋值给 bx 和 es

```
ax 0000
bx 1140
cx 0000
dx 0000
si 0000
di 0000
bp 0000
sp 00C8
ds 0000
es F000
```

图 3-6 将 0000:[004CH]开始的两个字分别赋值给 bx 和 es

由图可知，1H 中断的中断服务程序的段地址为 0811H，偏移地址为 0B1AH；13H 中断的中断服务程序的段地址为 0F000H，偏移地址为 1140H。

使用上述三种方法获得的中断服务程序的入口地址中，13H 的入口地址三者相同，均为 0F000H:[1140H]，但是 1H 的入口地址方法 2 和方法 3 得到的结果相同，与方法 1 的结果不同，根据老师的指导可知，该不同是由于 dos 操作系统对内存中物理地址开始的位置的几个字节的保护措施造成的，使得在 td 工具中无法直接查看到这些内存空间的真实内容。因此，无法用方法 1 获取 1H 中断的入口地址，但是可以使用方法 2 和方法 3 间接获取该中断的地址，1H 的入口地址为 0811H:[0B1AH]。

汇编语言程序设计实验报告

3.2 任务 2

3.2.1 设计思路

1. 接管 16H 中断服务程序的设计思路

首先保存原有 16H 中断服务程序的入口地址，然后设计一个新的中断服务程序。在新的中断服务程序中，若输入功能号不为 0 或者 10H，则程序直接跳转到原来的中断服务程序中，若输入功能号为 0 或者 10H，则首先将标志寄存器压栈，并调用原来的 16 号中断服务程序，然后将其读取到的字符的 ascii 码(存储在 al 中)与 41H 和 5BH 进行比较，若在 41H 和 5BH 之间(包括 41H 和 5BH)，则将 al 增加 20H，再使用 iret 返回，否则，直接使用 iret 返回。使用新的中断服务程序的段地址和偏移地址取代中断矢量表中 16H 中断对应的内存空间，并将该部分代码驻留在内存中，即可接管 16H 中断服务程序。

2. 恢复原有 16H 中断服务程序的设计思路

在为接管 16H 中断服务程序的 dos 系统上，使用 td 查看 16H 中断服务程序的入口地址，即内存 0000:[0058H]处的内容，将该位置的一个双字的内容记录下来，将该内容直接赋值给内存 0000:[0058H]处，即可恢复原有的 16H 中断服务程序。

3.2.2 源代码

```
; task2_1.asm 接管 16H 中断服务程序
.386
code segment use16
    assume cs:code, ss:stack
oldINT16H dw ?,?
newINT16H:
    cmp ah, 0
    jz trans
    cmp ah, 10h
    jz trans
    jmp dword ptr oldINT16H
trans:
    pushf
    call dword ptr oldINT16H
    cmp al, 'A'
    jb nottrans
    cmp al, 'Z'
    ja nottrans
    add al, 20h
nottrans:
    iret

start:
```

汇编语言程序设计实验报告

```
mov ax, 0
mov es, ax
mov ax, es:[16H * 4]
mov oldINT16H, ax
mov ax, es:[16H * 4 + 2]
mov oldINT16H + 2, ax

CLI

mov word ptr es:[16H * 4], offset newINT16H
mov es:[16H * 4 + 2], cs
STI

mov dx, offset start + 15
shr dx, 4
add dx, 10h
mov ah, 31h
int 21h
code ends
stack segment use16 stack
    db 200 dup(0)
stack ends
end start

; task2_2.asm 恢复原有 16H 中断服务程序
.386
code segment use16
start:
    mov ax, 0
    mov es, ax
    mov dword ptr es:[16H * 4], 0F00011E0H
    mov ah, 4Ch
    int 21h
code ends
end start
```

3.2.3 实验步骤

1. 接管 16H 中断服务程序

在 dosbox 中, 使用 masm 和 link 工具对汇编源程序 task2_1.asm 进行编译和链接, 得到可执行文件 task2_1.exe, 直行 task2_2.exe, 即可实现对 16H 中断服务程序的接管工作。

2. 恢复原有 16H 中断服务程序

在 dosbox 中, 使用 masm 和 link 工具对汇编源程序 task2_2.asm 进行编译和链接, 得到可执行文件 task2_2.exe, 执行 task2_2.exe, 即可实现对 16H 中断服务程序的恢复工作。

汇编语言程序设计实验报告

3.2.4 实验记录与分析

1. 执行 task2_1.exe 接管 16H 中断服务程序后，分别在 td 中以及实验三任务 1 的程序中进行测试。在 td 的代码区中输入 “mov AX, 0”，其结果如图 3-7 所示。运行实验三任务 1 的程序，并输入 “HUAO”，其结果如图 3-8 所示。

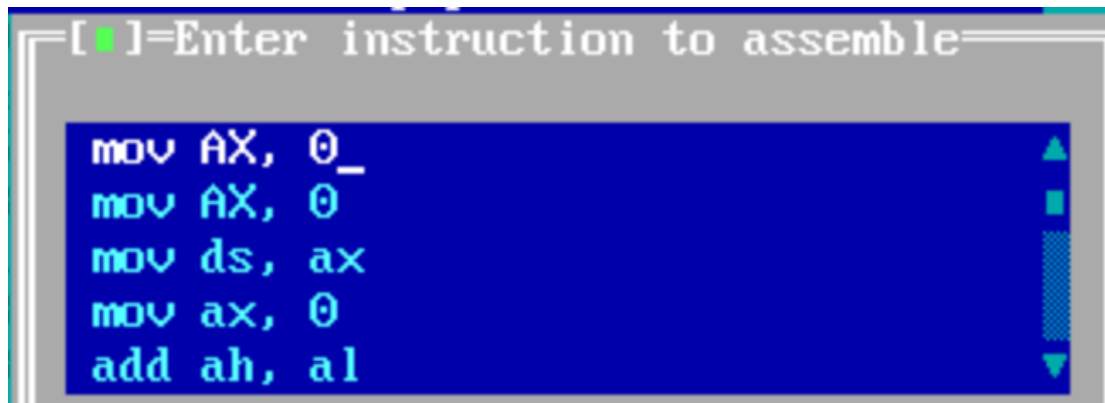


图 3-7 接管 16H 中断服务程序后，在 td 中输入 “mov AX, 0”

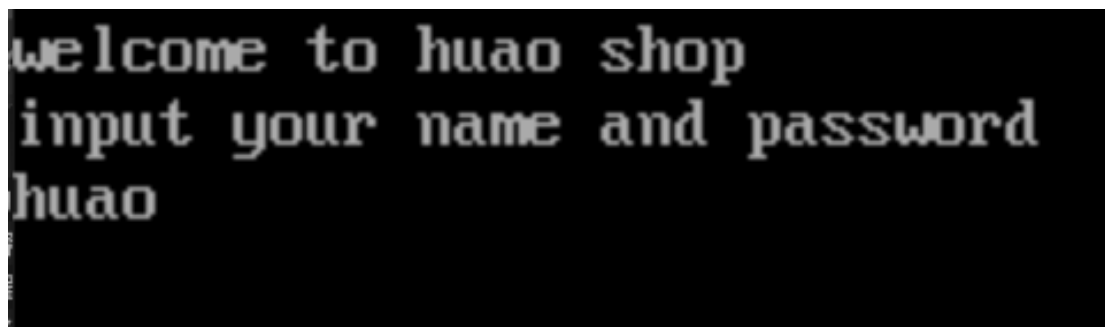


图 3-8 接管 16H 中断服务程序后，在实验三任务 1 中输入 “HUAO”

由图可知，接管 16H 中断服务程序后，对在 td 中的输入没有产生影响，但是对自己编写的程序中的输入会产生影响。

2. 执行 task2_2.exe 恢复原来的 16H 中断服务程序后，再次执行实验三任务 1 的程序，输入 “HUAO”，其结果如图 3-9 所示。

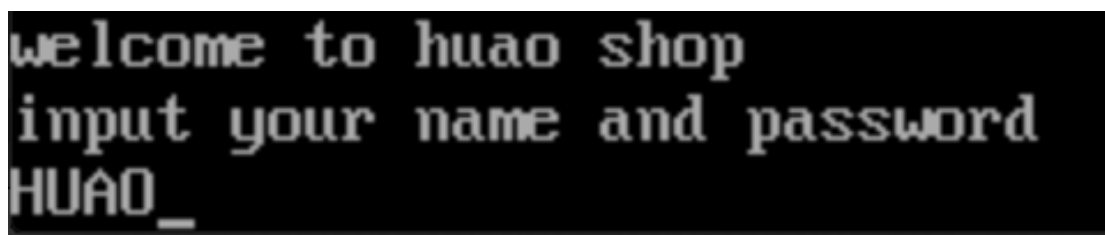


图 3-9 恢复原有 16H 中断服务程序后，在实验三任务 1 中输入 “HUAO”

由图可知，程序 task2_2.exe 成功恢复了原有的 16H 中断服务程序。

3.3 任务 3

3.3.1 设计思路

在数据段中定义一个字节变量，存储待读取的 CMOS 数据地址，使用 out 指令将该地

汇编语言程序设计实验报告

址输出到 70H 的外设端口，然后使用 in 指令从 71H 端口读取数据，即为从 CMOS 中读取相应端口的值。

3.3.2 源代码

从 CMOS 中读取信息的源代码如下。

.386

putnum MACRO

local large, endputnum

mov ah, 2

cmp dl, 9

ja large

add dl, '0'

jmp endputnum

large:

add dl, 'A'

sub dl, 10

endputnum:

int 21h

ENDM

data segment use16

CMOSAddr db 2 ; 修改此处的值以指定待读取的 CMOS 单元地址

data ends

stack segment use16 stack

db 200 dup(0)

stack ends

code segment use16

assume ds:data, cs:code, ss:stack

start:

mov ax, data

mov ds, ax

mov al, 0

mov al, CMOSAddr

out 70h, al

in al, 71h

mov bl, al

mov dl, bl

shr dl, 4

putnum

汇编语言程序设计实验报告

```
mov dl, bl
shl dl, 4
shr dl, 4
putnum

mov ax, 4C00H
int 21h
code ends
end start
```

3.3.3 实验步骤

在 dosbox 中使用 masm 和 link 对 task3.asm 进行编译和链接,得到可执行文件 task3.exe, 执行该文件将输出指定的 CMOS 单元的信息。修改源文件数据段中的 CMOS 地址值, 重复上述操作, 即可从 CMOS 中读取到不同位置的数据。

3.3.4 实验记录与分析

依次将数据段中 CMOSaddr 的值改为 2H, 4H, 6H, 7H, 8H, 9H, 执行相应的可执行文件, 输出结果如图所示。下列测试的完成日期为 2019 年 4 月 27 日星期六。

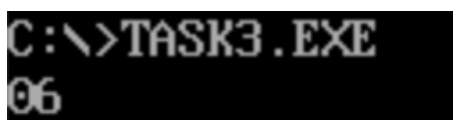


图 3-10 CMOSaddr = 2H(minute), 当前时间为 14:06

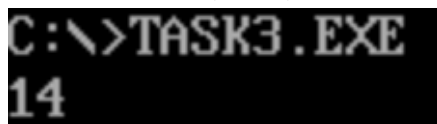


图 3-11 CMOSaddr = 4H(hour), 当前时间为 14:08

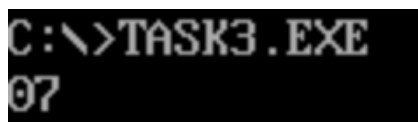


图 3-12 CMOSaddr = 6H(week)



图 3-13 CMOSaddr = 7H(day)

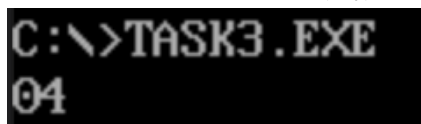


图 3-14 CMOSaddr = 8H(month)

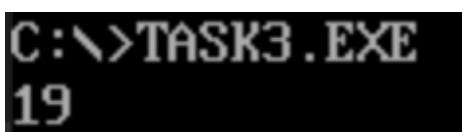


图 3-15 CMOSaddr = 9H(year)

汇编语言程序设计实验报告

由上述运行结果可知，程序从 CMOS 中读取到的时间信息与程序执行时的时间相同，说明程序正确的从 CMOS 中读取到了待读取的数据。

3.4 任务 4

3.4.1 设计思路

本任务中主要包含以下几部分内容，其设计思路如下。

(1) 限制用户登录错误的次数。在用户登录的过程中使用一个寄存器用于记录错误登录的次数，当该次数超过 3 次时，程序不再跳转到登录界面，而是直接视为未登录状态继续执行。

(2) 对用户的密码进行加密存储。本次任务中，我选择存储用户密码的格雷码来代替用户的密码，当用户输入密码后，将其与其右移一位后结果进行异或运算得到其格雷码，将其与存储的用户密码的密文进行比较，若二者相同，则密码正确，否则，密码错误。

(3) 对商品的进货价进行加密存储。本次任务中，我选择存储商品的进货价的两倍加 5 来代替商品的进货价，同时，在商品信息中添加一些随机的无关数据，以隐藏进货价存储的真实位置。

(4) 给程序添加反跟踪功能。本次任务中，我使用了计数和堆栈检查两种方法在进行反跟踪。对于计时方法，当程序执行部分指令时耗时明显大于程序直接运行时所需时间，则可以认为该程序处于调试状态，当程序计时结果如此时，程序直接跳转到相应位置终止程序；对于堆栈检查方法，若程序处于调试状态，程序会使用堆栈空间，在此过程中会对堆栈顶以外的数据进行修改，在堆栈检查方法中，首先将一个数据压入堆栈中，然后将其弹出，然后执行一段指令后，查看堆栈段中栈顶上相应空间内存存储的内容与先前压栈的数据是否相同，若相同，则认为该部分代码未被调试，否则，说明该部分代码被调试执行，此时，程序直接跳转到相应的位置终止程序。

3.4.2 源代码

本次任务在实验三任务 1 的基础上完成，因此此处仅展示在实验三任务 1 的基础上添加或修改的代码部分。下面代码中省略号部分代表与实验三任务 1 中相同的代码。

```
.....
bname DB 'huao', 6 dup(0), 4      ;boss name
; bpass DB 'huao', 6 dup(0), 4      ;password
bpass DB 0dch, 0cfh, 0d1h, 58h, 0dch, 45h, 77h, 0ach, 0FFH, 0, 4      ;使用秘文存储用户密码
.....

mov bx, offset countLoginError
mov al, [bx]
cmp al, 1
ja welcomeUser    ;限制错误登陆次数
inc al
```

汇编语言程序设计实验报告

```
mov [bx], al
jmp welcomeStart
.....
; 使用计时方法反跟踪
cli

mov ah, 2ch
int 21h ;计时 1
push dx

mov ecx, [bx]
mov edx, ecx
shr edx, 1
xor ecx, edx
push edx
push ecx

mov ah, 2ch
int 21h ;计时 2
sti

mov bx, offset pass1
mov ax, dx
pop ecx
pop edx
cmp ax, [esp]
pop ax
jz ok1
mov bx, offset E1
ok1:
;mov bx, [bx]
cmp word ptr cs:[bx], 0BBEH
jz ok2
jmp E1
ok2:
jmp bx
db 'this code is useless'

pass1:
mov si, offset bpass
mov edx, [si]
cmp ecx, edx
jz truePass
.....
; 使用堆栈检测进行反跟踪
```

汇编语言程序设计实验报告

```
cli      ;堆栈检查反跟踪
mov ax, offset pass2
push ax
pop ax

mov ax, [si + 20H]
sub ax, 5
shr ax, 1

mov bx, [esp - 2]
sti
mov cx, offset pass2
cmp cx, bx
jnz error
jmp bx
pass2:
call F2T10
CRLF
.....
```

3.4.3 实验步骤

将修改后的代码使用 `masm` 分别编译, 然后使用 `link` 将三个文件链接起来生成可执行文件。直接执行该可执行文件观察程序执行情况, 然后使用 `td` 对该可执行程序进行调试, 观察程序执行情况, 并于非调试情况下的执行情况进行比较。

3.4.4 实验记录与分析

(1) 对使用计时方法进行反跟踪的代码部分进行调试, 观察到两次调用 `2CH` 系统功能时返回的时间 `dx` 发生改变, 此时, 程序跳转到退出功能处。调试过程中, 两次计时的截图如图 3-16 和图 3-17 所示。



图 3-16 第一处获取系统时间结果



图 3-17 第二处获取系统时间结果

由于在调试过程中，程序中的指令单步执行，因此消耗的时间要远大于程序直接执行时所需的时间，由于两处计时之间包含的指令数较少，可以认为两处的时间应该几乎相同，但是在如图所示的过程中，这些指令的执行消耗大量的时间，因此可以认为程序处于调试状态，此时，将程序直接跳转到退出程序的代码处。

(2) 对使用堆栈检测方法反跟踪的代码部分进行调试，观察堆栈顶外一个字数据的变化，在调试的过程中，可以看到在调试完部分代码后，重新读取该位置数据得到的结果与写入该位置的值不相同，此时，程序将跳转到退出程序的代码处，程序终止运行。将 ax 的值压入堆栈后弹出，相应的寄存器的值如图 3-18 所示，重新读取该位置的值并写入待 bx 中，此时相应的寄存器的值如图 3-19 所示。

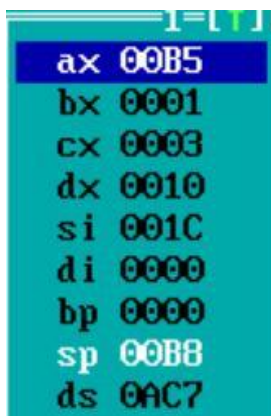


图 3-18 将 ax 先压栈后出栈时的寄存器的值



图 3-19 将栈顶外的一个字的内容读取到 bx 中

由图可知，在调试过程中，调试工具会使用程序的堆栈段的空间，使得栈顶外的内存空

汇编语言程序设计实验报告

间的内容发生改变。由于程序本身没有修改这些内存空间的值，因此根据该特点可以得知程序是否处于调试状态，若检测到为调试状态，则程序直接跳转到结束程序的代码处，从而实现反跟踪的目的。

3.5 任务 5

3.5.1 设计思路

在得到同组同学的可执行文件后，使用 `td` 调试工具对其进行单步调试，首先通过添加断点的方式判断各部分的功能，然后对与输入密码和输出商品信息相关的部分进行单步调试，从而分析其中的反跟踪方法以及加密方法。

3.5.2 实验步骤

首先，在 `dosbox` 中直接运行该可执行文件，尝试输入相关的信息，了解程序的大致执行步骤，然后使用 `td` 调试该程序，尝试修改其中的部分代码，绕过一些用于反跟踪的代码以及密码验证部分，从而直接使用已登录状态的功能。

3.5.3 实验记录与分析

(1) 在单步调试的过程中，执行程序开始的部分代码后，发现键盘输入失效，无法继续调试，分析此部分代码可知，其接管了系统的部分中断服务程序，导致在 `td` 中无法继续实现调试过程，为继续观测程序执行的情况，在程序接管系统中断服务前添加跳转指令，直接跳转到该部分代码的后面，使得键盘在 `td` 状态下可以正常使用，从而实现对程序的继续调试。此部分代码的截图如图 3-20 所示。

cs:0000	B8C70A	mov	ax,0AC7
cs:0003	8ED8	mov	ds,ax
cs:0005	33C0	xor	ax,ax
cs:0007	8EC0	mov	es,ax
cs:0009	26A10400	mov	ax,es:[0004]
cs:000D	A30400	mov	[0004],ax
cs:0010	26A10600	mov	ax,es:[0006]
cs:0014	A30600	mov	[0006],ax
cs:0017	26A10C00	mov	ax,es:[000C]
cs:001B	A30800	mov	[0008],ax
cs:001E	26A10E00	mov	ax,es:[000E]
cs:0022	A30A00	mov	[000A],ax
cs:0025	FA	cli	
cs:0026	B84800	mov	ax,0048
cs:0029	26A30400	mov	es:[0004],ax

图 3-20 接管中断服务程序的部分代码

将该部分代码上面的两行代码替换成如下代码，然后继续单步执行，即可跳过该部分代码，并继续调试后面的程序。

汇编语言程序设计实验报告

`mov bx, 48H` ; 查看后面的代码可知,跳出修改中断矢量表代码的部分可以直接跳转到 CS:[48H] 处。

`jmp bx`

通过执行上面两行代码,程序未能成功修改中断矢量表,因此可以继续使用 `td` 进行调试。

(2) 继续调试找到判断用户密码是否正确的部分,其中包含修改用户是否登录的标志位的代码,如图 3-21 所示。

```
cs:0264 7505      jne     026B
cs:0266 BB0000     mov     bx,0000
cs:0269 EB03      jmp     026E
cs:026B BBFFFF     mov     bx,FFFF
```

图 3-21 修改用户是否登录标志位的代码部分

图中通过对 `bx` 赋值并在后面的代码中将其存储到数据段中的相应位置,从而实现对用户是否处于登录状态进行判断。

为绕过登录状态判断,将其中“`mov bx,FFFF`”一行修改为“`mov bx,0000`”,然后继续执行程序,即可在用户未登录的状态下仍然被认定为已经登录,从而使用已登录状态的各种功能,此时,查看某商品的信息,程序将会输出其进货价,如图 3-22 所示。

```
input commodity you want to check:pen
good_name is: pen
good_sale is: 10
good_in_price is: 231
good_out_price is: 56
good_all_count is: 70
good_sale_count is: 25
good_recommand is: 65
good_sort is: 1
```

图 3-22 伪装为登录状态查看商品信息

从上图可知,此方法可以伪装成以登录状态使用相应的功能,但是仍然未能获取到真实的进货价。

通过与同组同学交流可知,其进货价使用正确的密码进行相应的加密,并使用用户输入的密码进行解密,因此即使绕过登录判断,由于用于存储用户输入密码的内存空间中存放的内容并非正确的密码,因此无法对进货价进行正确的解密,从而无法获取到真实的进货价。

4 总结和体会

在任务 1 和任务 2 中，我通过不同的方式获取了 dos 下中断服务程序的入口地址，并编写了相应的程序对其进行接管，通过完成这两个任务，我对 dos 下的中断矢量表、中断服务程序的概念有了更加深入的认识，也让我了解到了汇编语言调用 dos 系统功能的实现方式和本质。在任务 3 中，我完成了汇编语言与外部设备之间的数据通信，并成功地从 CMOS 中读取到了想要的信息。

通过前三个任务，我对 dos 操作系统以及计算机本身的工作原理有了更加深入的认识，同时，我也练习使用了 IN/OUT 等先前未使用过的汇编指令，让我对这些使用较少的指令的功能和使用方法有了直观的认识。

在任务 4 和任务 5 中，我和同组的同学分别完成了数据的加密以及汇编程序的反跟踪，并且对对方的程序尝试跟踪和解密。在此次任务中，我认识到了汇编程序可能存在的很多安全隐患，也学会了一些解决这些问题的方法。在了解反跟踪的过程中，我对汇编程序的执行过程有了更加明确的认识，另外，我也知道了 td 工具调试程序时会执行的一些操作以及对程序产生的影响，使我对程序的调试过程有了新的认识。

在本次实验中，我也遇到了很多的问题。例如在任务 1 中无法直接从 td 工具的内存查看区域中直接读取中断矢量表中的前几个数据，在任务 4 中，我发现在 td 的内存查看区域中同样不能够查看程序的堆栈段中的部分数据。这些问题让我对 dos 操作系统以及 td 调试工具对于敏感内存区域的保护有了一定的了解。

华中科技大学

课程实验报告

课程名称： 汇编语言程序设计实验

实验名称： 实验五 WIN32 编程

实验时间： 2018-5-7, 14: 00-17: 30

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： U201714761

姓 名： 胡澳

同组学生： 无

报告日期： 2018 年 5 月 7 日

原创性声明

本人郑重声明：本报告的内容由本人独立完成，有关观点、方法、数据和文献等的引用已经在文中指出。除文中已经注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品或成果，不存在剽窃、抄袭行为。

特此声明！

学生签名：

日期：2018.5.7

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：

汇编语言程序设计实验报告

目 录

1 实验目的与要求	89
2 实验内容	89
3 实验过程	91
3.1 功能 1.....	91
3.1.1 设计思路.....	91
3.1.2 源代码.....	91
3.1.3 实验步骤.....	91
3.2 功能 2.....	91
3.2.1 设计思路.....	91
3.2.2 源代码.....	92
3.2.3 实验步骤.....	95
3.2.4 实验记录和分析.....	95
4 总结和体会	97

汇编语言程序设计实验报告

1 实验目的与要求

- (1) 熟悉 WIN32 程序的设计和调试方法；
- (2) 熟悉宏汇编语言中 INVOKE、结构变量、简化段定义等功能；
- (3) 进一步理解机器语言、汇编语言、高级语言之间以及实方式、保护方式之间的一些关系。

2 实验内容

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)。

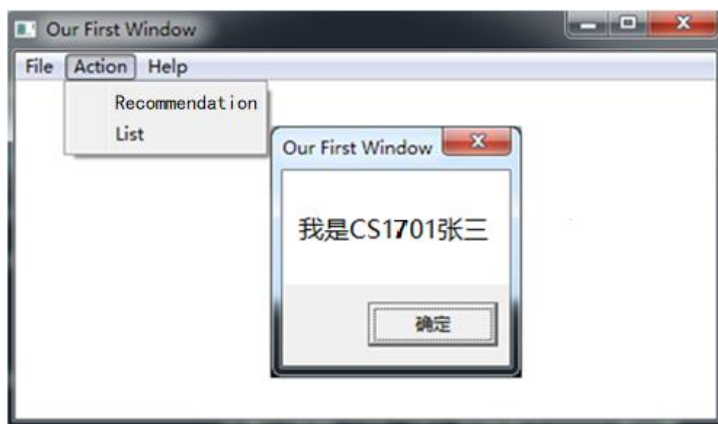


图 5.1 菜单示例

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)。

汇编语言程序设计实验报告



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

图 5.2 商品信息显示示意图

汇 编 语 言 程 序 设 计 实 验 报 告

3 实验过程

3.1 功能 1

3.1.1 设计思路

参考样例程序中资源文件 menu.rc 的格式，设计本程序的菜单资源信息，并为不同的按钮定义相应的常量用于程序中对窗口消息的分类和分发。

3.1.2 源代码

资源文件 menu.rc 的源代码如下。

```
#define IDM_FILE_EXIT 10001
#define IDM_ACTION_RECOMMAND 10101
#define IDM_ACTION_LIST 10102
#define IDM_HELP_ABOUT 10201

MyMenu MENU
BEGIN
    POPUP "&File"
        BEGIN
            MENUITEM "E&xit",IDM_FILE_EXIT
        END

    POPUP "&Action"
        BEGIN
            MENUITEM "R&ecommandation", IDM_ACTION_RECOMMAND
            MENUITEM "L&ist",IDM_ACTION_LIST
        END

    POPUP "&Help"
        BEGIN
            MENUITEM "&About",IDM_HELP_ABOUT
        END
    END
```

3.1.3 实验步骤

使用 rc 程序将资源文件 menu.rc 转化为 menu.RES 文件，以便在后续链接过程中使用。

3.2 功能 2

3.2.1 设计思路

1. Recommendation 功能。对实验三中 CalRecommand 函数进行相应的修改，使其能够

汇编语言程序设计实验报告

在 win32 环境下正常工作。调用该函数即可完成对所有商品推荐度的计算。

2. List 功能。首先对实验三中的 F2T10 和 RADIX 函数进行相应的修改，使其能够在 win32 环境下工作，并将转化后的字符串存储到指定内存单元中、返回字符串长度。然后设计一个 OutputOne 函数输出某个商品的一个信息。对 OutputOne 函数传递不同的参数，即可完成对所有商品信息的输出。

3.2.2 源代码

1. 为完成 Recommendation 功能，对 CalRecommand 函数进行修改，修改后的函数代码如下。

```
CalRecommand proc
    pushad

    mov esi, offset gal
    lea ebx, [esi]
    sub ebx, goodLen

    mov di, N
    inc di

calLoop:
    dec di
    add ebx, goodLen
    cmp di, 0
    jz calEnd

    mov ax, [ebx + 12]
    mov cx, 1280
    mul cx
    push dx
    push ax
    mov al, byte ptr [ebx + 14]
    mov cl, [ebx + 11]
    mul cl
    mov cx, ax
    pop ax
    pop dx
    div cx
    push ax

    mov ax, [ebx + 18]
    mov cx, 64
    mul cx
    mov cx, [ebx + 16]
```

汇编语言程序设计实验报告

```
div cx
mov dx, ax
pop ax
add ax, dx

mov [ebx + 20], ax
jmp calLoop

calEnd:
popad
ret
CalRecommand endp
```

2. 完成 List 功能时，对实验三的 F2T10 和 RADIX 函数进行了修改，并设计了函数 OutputOne，其代码如下。

```
F2T10 proc
push ebx
push edx
push esi
mov edx, 32
lea esi, bufF2T10
movsx eax, ax

or eax, eax
jns plus
neg eax
mov byte ptr [esi], '-'
inc esi
plus:
mov ebx, 10
invoke RADIX
mov byte ptr [esi], '$'

lea edx, bufF2T10
sub esi, edx
mov eax, esi
pop esi
pop edx
pop ebx
ret
F2T10 endp

RADIX proc
push ecx
```

汇编语言程序设计实验报告

```
    push edx
    xor ecx,ecx
LOP1:
    xor edx, edx
    div ebx
    push edx
    inc ecx
    or eax, eax
    jnz LOP1
LOP2:
    pop eax
    cmp al, 10
    jb LO1
    add AL, 7
LO1:
    add al ,30H
    mov [esi], al
    inc esi
    loop LOP2
    pop edx
    pop ecx
    ret
RADIX endp

OutputOne proc countOut:DWORD, countLocate:DWORD, hdc:DWORD
    pushad
    mov ebx, countOut
    xor eax, eax
    cmp ebx, 11
    jz number1
    mov ax, [esi + ebx]
    jmp next1
number1:
    mov al, [esi + ebx]
next1:
    invoke F2T10
    mov ecx, countLocate
    imul ecx, XX_GAP
    add ecx, XX
    invoke TextOut, hdc, ecx, edx, offset bufF2T10, eax
    popad
    ret
OutputOne endp
```


汇编语言程序设计实验报告

3.2.3 实验步骤

使用 ml 对汇编源程序 demo.asm 进行编译得到 demo.obj 文件, 然后使用 link 将 demo.obj 和 menu.res 文件进行链接, 得到可执行文件 demo.exe。编译和链接命令如下。

```
ml /c /coff /F1 /Zi /I C:\masm32\lib /I C:\masm32\include /Ta .demo.asm //编译
link /subsystem:windows /debug /debugtype:cv /LIBPATH:C:\masm32\lib demo.obj menu.res //链接
```

3.2.4 实验记录和分析

1. 显示商品信息功能(List)。点击菜单中的 List 选项, 程序将在窗口中显示所有商品的信息。如图 5-3 所示。

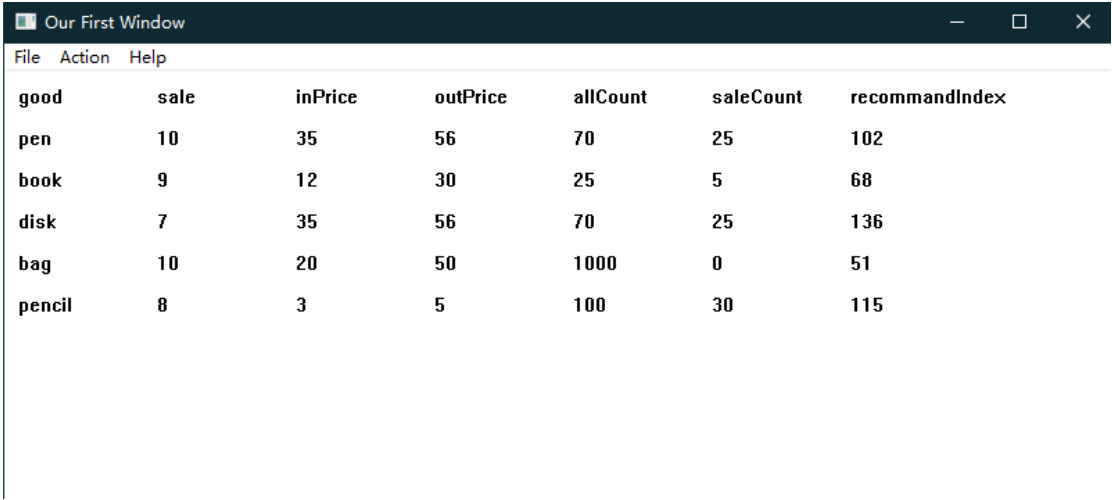


图 5-3 未计算推荐度时显示商品信息

2. 显示个人信息功能(About)。点击菜单中的 About 选项, 程序将弹出一个消息窗口, 窗口中显示个人信息。如图 5-4 所示。

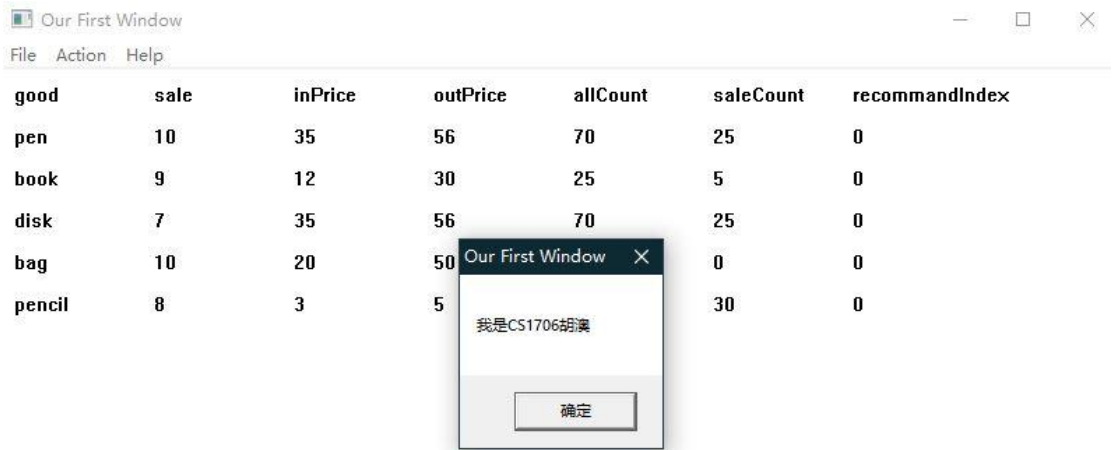


图 5-4 About 功能个人信息弹窗

汇编语言程序设计实验报告

3. 计算推荐度功能(Recommendation)。点击菜单中的 Recommendation 选项，再次点击 List 选项，程序将重新输出全部商品信息，此时输出的推荐度为计算得到的推荐度。如图 5-5 所示。

Our First Window						
File Action Help						
good	sale	inPrice	outPrice	allCount	saleCount	recommandIndex
pen	10	35	56	70	25	102
book	9	12	30	25	5	68
disk	7	35	56	70	25	136
bag	10	20	50	1000	0	51
pencil	8	3	5	100	30	115

图 5-5 计算推荐度后输出商品信息

可使用 td32 对本程序进行调试，并在调试过程中观察程序计算各商品推荐度的过程。首先使用 ml 的/FI 选项生成 lst 文件，再该文件中查看 calRecommand 函数的偏移地址，使用 td32 打开可执行文件 demo.exe，转移到该偏移地址处并增加断点。继续执行程序，在窗口中点击菜单中的 Recommendation 选项，此时程序执行到断点处，单步执行程序即可看到程序执行过程。如图 5-6 为程序计算完第一个商品的推荐度时的调试界面截图，推荐度存储在 eax 中。商品 1 的推荐度为 102，即此时 eax 中存储的 66H。

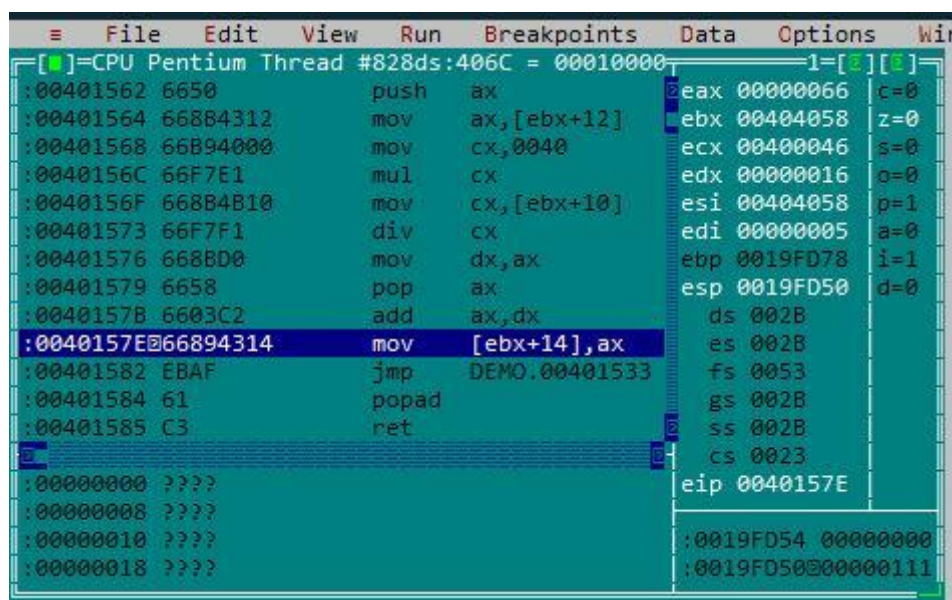


图 5-6 调试计算商品 1 推荐度

汇编语言程序设计实验报告

4 总结和体会

本次实验中，我完成了在 win32 窗口中显示网点商品管理系统商品信息的显示。通过本次实验，我熟悉了 win32 程序的基本结构，并对 win32 中的消息处理等概念有了直观的认识。另外，本次实验中，我对 dos 下的 16 位汇编和 windows 下的 win32 汇编的区别有了更明确的认识。

在本次实验中，我遇到了很多问题，例如使用 windows api 函数时，由于函数对部分寄存器内容的修改，导致程序无法按照预期的方式执行，通过参考相关的资料并对程序做了一些调整后，我解决了该问题。在这个过程中，我认识到了编写函数时保护寄存器内容以及编写函数文档的重要性，这也给我自己编写程序代码提供了很多的借鉴意义。