

**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： 2018-4-16，14：00-17：30 实验地点： 南一楼**

**指导教师： 朱虹**

**专业班级： 计算机科学与技术1601班**

**学 号： U201614531 姓 名： 刘本嵩**

**同组学生： 罗皓云 报告日期： 2018年 4 月 25日**

**原创性声明**

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

特此声明！

学生签名：

日期：

成绩评定

|  |  |  |
| --- | --- | --- |
| 实验完成质量得分（70分）（实验步骤清晰详细深入，实验记录真实完整等） | 报告撰写质量得分（30分）（报告规范、完整、通顺、详实等） | 总成绩（100分） |
|  |  |  |

指导教师签字：

日期：

**目录**

[1 实验目的与要求 1](#_Toc305152398)

[2 实验内容 2](#_Toc396755018)

[2.1 任务1：宏与子程序设计 2](#_Toc1085432658)

[2.2 任务2：在C语言程序中调用汇编语言实现的函数 3](#_Toc2045395577)

[3 实验过程 4](#_Toc870307574)

[3.1 任务1 4](#_Toc1967803428)

[3.1.1 实验步骤 4](#_Toc562106098)

[3.1.2 模块源代码 4](#_Toc824599809)

[3.1.3 实验记录与分析 20](#_Toc352415609)

[3.2 任务2 22](#_Toc1417857236)

[3.2.1 实验步骤 22](#_Toc773904411)

[3.2.2 源代码 23](#_Toc1182736163)

[3.2.3 实验记录与分析 32](#_Toc1112176312)

[4 总结与体会 37](#_Toc1886361760)

[5 参考文献 38](#_Toc1977292092)

# 实验目的与要求

1. 掌握子程序设计的方法与技巧，熟悉子程序的参数传递方法和调用原理；

2. 掌握宏指令、模块化程序的设计方法;

3. 掌握较大规模程序的合作开发与调试方法；

4. 掌握汇编语言程序与C语言程序混合编程的方法；

5. 熟悉C编译器的基本优化方法;

6. 了解C语言编译器的命名方法，主、子程序之间参数传递的机制。

# 实验内容

## 任务1：宏与子程序设计

进一步修改与增强实验一任务四的网店商品信息管理程序的功能，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=计算平均利润率，

4=计算利润率排名，5=输出全部商品信息，6=程序退出。

输入1-6的数字进入对应的功能。

（2）查询商品信息

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

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

（3）修改商品信息

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

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

如：进货价：25》24 //符号“》”仅作为分隔符，也可以选择其他分隔符号

销售价：46》5A6 //输入了非法数值，下一行重新显示和输入

销售价：46》56

进货总数：30》 //直接回车时，对这项信息不做修改

当对三项信息都处理完毕后，回到功能三（1）。

（4）计算平均利润率

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

（5）计算利润率排名

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

（6）输出全部商品信息

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

## 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。

# 实验过程

## 任务1

### 实验步骤

1. 根据任务的要求完成相应程序模块的编写，并声明公共符号以供另一个模块使用。

2. 测试自己编写的模块的正确性，经过编译，并保证没有错误。

3. 将自己编写的模块同队友编写的模块同进行编译、连接，并保证可以正确运行。

### 模块源代码

name module2

public calc\_profit\_rate, calc\_rank, dump\_303

.model small

.stack 256

.data

user\_name db 'bensongliu',0

user\_pswd db 'test',0,0

n equ 30

s1 db 'shop1',0

ga1 db 'pen$', 6 dup(0)

dw 35,56,70,25,?

ga2 db 'book$', 5 dup(0)

dw 12,30,25,5,?

gan db n-2 dup( 'tempvalue$',15,0,20,0,30,0,2,0,?,?)

s2 db 'shop2',0 ; 网店名称，用0结束

gb1 db 'book$', 5 dup(0) ; 商品名称

dw 12,28,20,15,?

gb2 db 'pen$', 6 dup(0) ; 商品名称

dw 35,50,30,24,?

gbn db n-2 dup( 'tempvalue$',15,0,20,0,30,0,2,0,?,?)

name\_buffer db 12, ?, 12 dup(0)

pswd\_buffer db 8, ?, 8 dup(0)

product\_buffer db 12, ?, 12 dup('$')

shop\_buffer db 0

buffer1 dw 8,?,8 dup(0)

in\_cho db 2, ?, 0, 0

pr1 dw 0,0

pr2 dw 0,0

pr\_sum dw 0,0

logged db 0

login\_ok db ?

mark db ?

must\_return db 0

buffer3 db 0ah,0dh,'$'

buffer4 db 0h

menu\_prompt db 'choice ~ ',10

db '1, query product info',10

db '2, edit product',10

db '3, calc profit',10

db '4, calc rank',10

db '5, print calc-ed info',10

db '6, exit',10

db ' ? ','$'

str1 db 12 dup(?)

str2 db 'User: $'

str3 db 'Password: $'

str4 db 'Exception occurred.$'

str5 db 'Permission denied.$'

str6 db 'Product ? $'

str7 db 'Shop (1/2) ? $'

str8 db 'Buy price $'

str9 db 'Sell price $'

str10 db 'On stock $'

str11 db '>> $'

str12 db 'commodity information:','$'

str13 db 'shop1','$'

str14 db 'shop2','$'

str15 db 'Buy price ','$'

str16 db 'Sell price ','$'

str17 db 'On stock ','$'

str18 db 'Sold ','$'

str19 db 'Profit ','$'

str20 db 'Rank ','$'

.code

rlib\_macro\_print macro buffer

lea dx,buffer

mov ah,9

int 21h

endm

rlib\_macro\_readstr macro buffer

lea dx,buffer

mov ah,10

int 21h

endm

; Written by Bensong Liu <root@recolic.net>

; Partner := Haoyun Luo

calc\_profit\_rate:

call calc\_profit\_rate\_impl

lea dx, str17

rlib\_macro\_print buffer3

jmp tmp112

; Written by Bensong Liu <root@recolic.net>

; Partner := Haoyun Luo

calc\_rank:

call calc\_rank\_impl

lea dx, str18

rlib\_macro\_print buffer3

jmp tmp112

tmp111:

call tmp167

push cx

mov cx,[di+12]

rlib\_macro\_print buffer3

rlib\_macro\_print str16

mov cx, ax

mov dx,16

pop cx

call tmp167

rlib\_macro\_print buffer3

rlib\_macro\_print str17

mov ax, di

mov dx,16

mov ax,[ax+14]

call tmp167

rlib\_macro\_print buffer3

rlib\_macro\_print str18

xor dx, dx

or dx, di

mov ax,[dx+16]

mov dx,16

call tmp167

call locate\_product

jmp tmp129

; Written by Bensong Liu <root@recolic.net>

; Partner := Haoyun Luo

calc\_profit\_rate\_impl proc

push ax

push bx

push cx

push dx

push di

push si

lea di, str20

tmp136:

mov ax,[ga1+12]

mov bx,[ga1+16]

imul bx

xor cx, cx

mov cx,ax

inc ax

mov ax,[ga1+10]

mov bx,[ga1+14]

imul bx

xor bx, bx

mov bx,ax

mov ax,cx

mov cx,bx

sub ax,bx

cwde

mov ebx,100

imul eax,ebx

mov bx,cx

idiv bx

mov word ptr[pr1],ax

call locate\_product

jmp tmp137

tmp137:

mov ax,[si+12]

mov bx,[si+16]

imul bx

xor cx, cx

mov cx,ax

mov ax,di

mov ax,[si+10]

mov bx,[si+14]

imul bx

xor bx, bx

mov bx,ax

mov ax,cx

mov cx,bx

sub ax,bx

cwde

mov ebx,100

imul eax,ebx

mov bx,cx

idiv bx

mov word ptr[pr2],ax

mov bx,offset pr1

mov cx,word ptr[bx]

add ax,cx

sar ax,1

mov word ptr[di+18],ax

add di,20

mov bx,offset s2

cmp di,bx

jge term\_307

jmp tmp136

term\_307:

pop ax

pop bx

pop dx

pop cx

pop bx

pop ax

ret

calc\_profit\_rate\_impl endp

; Written by Bensong Liu <root@recolic.net>

; Partner := Haoyun Luo

calc\_rank\_impl proc

push ax

push bx

push cx

push dx

push di

push si

lea bx, s2

lea di, ga1

tmp140:

mov dx,1

mov cx,[di+18]

mov si,offset ga1

tmp141:

add si,20

mov ax,[si+18]

cmp si,bx

jl tmp125

jmp tmp142

tmp125:

cmp cx,ax

jge tmp141

inc dx

jmp tmp141

tmp142:

mov [si+18],dx

call locate\_product

add di,20

cmp di,bx

jl tmp140

term\_308:

pop si

pop di

pop dx

pop cx

pop bx

pop ax

ret

calc\_rank\_impl endp

; Written by Bensong Liu <root@recolic.net>

; Partner := Haoyun Luo

dump\_303 proc

rlib\_macro\_print str12

rlib\_macro\_print buffer3

rlib\_macro\_print str13

rlib\_macro\_print buffer3

rlib\_macro\_print product\_buffer+2

rlib\_macro\_print buffer3

rlib\_macro\_print str15

mov dx,16

mov ax,[di+10]

jmp tmp111

rlib\_macro\_print product\_buffer+2

rlib\_macro\_print buffer3

cmp eax,7fffh

jc tmp176

dec cx

jne tmp174

jnz tmp172

cmp dx,16

ja tmp176

tmp129:

rlib\_macro\_print buffer3

rlib\_macro\_print buffer3

rlib\_macro\_print str14

rlib\_macro\_print buffer3

rlib\_macro\_print product\_buffer+2

rlib\_macro\_print buffer3

rlib\_macro\_print str15

mov ax,[si+10]

mov dx,16

call tmp167

rlib\_macro\_print buffer3

rlib\_macro\_print str16

mov ax,[si+12]

mov dx,16

call tmp167

rlib\_macro\_print buffer3

rlib\_macro\_print str17

mov ax,[si+14]

mov dx,16

call tmp167

rlib\_macro\_print buffer3

rlib\_macro\_print str18

mov ax,[si+16]

mov dx,16

call tmp167

rlib\_macro\_print buffer3

rlib\_macro\_print buffer3

ret

dump\_303 endp

; Written by Bensong Liu <root@recolic.net>

; Partner := Haoyun Luo

find\_304 proc

push ax

push bx

push cx

push dx

push si

tmp10:

rlib\_macro\_print buffer3

rlib\_macro\_print str6

rlib\_macro\_readstr product\_buffer

rlib\_macro\_print buffer3

cmp product\_buffer[1],0

je tmp147

jmp tmp9

tmp147:

mov [mark],-1

jmp term\_306

; search the product

tmp9: mov di,offset s1

add di,6

mov bl,[product\_buffer+1]

mov bh,0

mov si,offset product\_buffer

cmp bx,10

je tmp149

mov dl,[di+bx]

cmp dl,0

je tmp7

tmp149:

mov ch,[si+bx+1]

mov cl,[di+bx-1]

cmp cl,0

je tmp7

cmp ch,cl

jne tmp7

jmp tmp8

tmp8:

cmp bx,1

je tmp6

dec bx

jmp tmp149

tmp7:

and ch, ch

xor bx, bx

mov bl,[product\_buffer+1]

mov cl,[logged]

add di,20

dec cx

lea cx, s2

cmp cx,di

je tmp10

tmp154:

jmp tmp149

tmp6:

mov [mark],1

jmp term\_306

term\_306:

pop si

pop dx

pop cx

pop bx

pop ax

ret

find\_304 endp

; Written by Bensong Liu <root@recolic.net>

; Partner := Haoyun Luo

locate\_product proc

push ax

push bx

push cx

push dx

push di

push si

mov bx,10

mov cx, 'q'

lea si, gb1

inc bx

looper302:

mov cl,[di+bx-1]

mov ch,[si+bx-1]

dec bx

sub ch,cl

jnz tmp161

tmp160:

cmp bx,1

je tmp162

jmp looper302

tmp161:

add si,20

mov bx,11

jmp looper302

tmp162:

pop ax

pop di

pop dx

pop cx

pop bx

pop ax

ret

locate\_product endp

start:

mov ax,data

mov ds,ax

lea dx,str2

mov ah,9

int 21h

lea dx,offset name\_buffer

mov ah,10

int 21h

lea dx,buffer3

mov ah,9

int 21h

cmp name\_buffer[1],0

je tmp14

mov bh,name\_buffer+2

cmp bh,'q'

je tmp17

lea dx,str3

mov ah,9

int 21h

lea dx,offset pswd\_buffer

mov ah,10

int 21h

jmp tmp13

mov ax, 4c00h

int 21h

tmp18:mov ax, 4c00h

int 21h

tmp17: mov bh,name\_buffer+1

cmp bh,1

je tmp18

tmp16: mov ax, 4c00h

int 21h

tmp14:

mov buffer4,0

jmp tmp112

tmp13:

mov di,offset name\_buffer

mov bx,di

inc bx

mov si,10

mov bx,[bx]

mov bh,0

cmp si,bx

jne tmp11

tmp24:

mov ch,1[di][bx]

mov cl,[user\_name+bx-1]

cmp ch,cl

jne tmp11

dec si

jne tmp24

jmp tmp12

tmp12: mov di,offset pswd\_buffer

mov bx,di

inc bx

mov si,4

mov bx,[bx]

mov bh,0

cmp si,bx

jne tmp11

tmp23:

mov ch,1[di][bx]

mov cl,[user\_pswd+bx-1]

cmp ch,cl

jne tmp11

dec si

jne tmp23

mov buffer4,1h

jmp tmp112

tmp11: lea dx,buffer3

mov ah,9

int 21h

lea dx,str5

mov ah,9

int 21h

lea dx,buffer3

mov ah,9

int 21h

jmp start

tmp112:

cmp buffer4,0

jz tmp113

jmp tmp114

tmp113:

rlib\_macro\_print buffer3

rlib\_macro\_print menu\_prompt

rlib\_macro\_print buffer3

rlib\_macro\_print menu1

rlib\_macro\_print buffer3

rlib\_macro\_print menu6

rlib\_macro\_print buffer3

rlib\_macro\_print menu7

rlib\_macro\_readstr in\_cho

rlib\_macro\_print buffer3

lea di,in\_cho+2

mov al,[di]

cmp al,'1'

jz query\_products

cmp al,'6'

jz tmp129

jmp start

tmp114:rlib\_macro\_print buffer3

rlib\_macro\_print menu\_prompt

rlib\_macro\_print buffer3

rlib\_macro\_readstr in\_cho

lea di,in\_cho+2

mov al,[di]

cmp al,'1'

jz query\_products

cmp al,'2'

jz edit\_input

cmp al,'3'

jz calc\_profit\_rate

cmp al,'4'

jz calc\_rank

cmp al,'5'

jz tmp122

cmp al,'6'

jz tmp129

rlib\_macro\_print str4

rlib\_macro\_print buffer3

jmp tmp114

tmp122:rlib\_macro\_print str12

rlib\_macro\_print buffer3

rlib\_macro\_print str13

rlib\_macro\_print buffer3

mov cx,3

lea di,s1+6

jmp tmp123

tmp123:rlib\_macro\_print buffer3

rlib\_macro\_print [di]

rlib\_macro\_print buffer3

rlib\_macro\_print str15

mov ax,[di+10]

mov dx,16

call tmp167

rlib\_macro\_print buffer3

rlib\_macro\_print str16

mov ax,[di+12]

mov dx,16

call tmp167

rlib\_macro\_print buffer3

rlib\_macro\_print str17

mov ax,[di+14]

mov dx,16

call tmp167

rlib\_macro\_print buffer3

rlib\_macro\_print str18

mov ax,[di+16]

mov dx,16

call tmp167

rlib\_macro\_print buffer3

cmp logged,0

jz tmp124

jmp tmp125

tmp124:rlib\_macro\_print str19

jmp tmp126

tmp125:rlib\_macro\_print str20

jmp tmp126

tmp126:mov ax,[di+18]

mov dx,16

call tmp167

rlib\_macro\_print buffer3

tmp127:add di,20

dec cx

jnz tmp123

cmp logged,1

jz tmp112

tmp128:mov logged,1

rlib\_macro\_print str14

rlib\_macro\_print buffer3

mov cx,3

lea di,gb1

jmp tmp123

tmp129: mov ah,4ch

int 21h

tmp163 proc

push cx

push edx

xor cx,cx

tmp164: xor edx,edx

div ebx

push dx

inc cx

or eax,eax

jnz tmp164

tmp165: pop ax

cmp al,10

jb tmp166

add al,7

tmp166: add al,30h

mov [si],al

inc si

loop tmp165

pop edx

pop cx

ret

tmp163 endp

tmp167 proc far

push ebx

push si

lea si,str1

cmp dx,32

je tmp168

movsx eax,ax

tmp168: or eax,eax

jns tmp169

neg eax

mov byte ptr[si],'-'

inc si

tmp169: mov ebx,10

call tmp163

mov byte ptr[si],'$'

lea dx,str1

mov ah,9

int 21h

pop si

pop ebx

ret

tmp167 endp

tmp170 proc far

push ebx

mov eax,0

mov login\_ok,0

mov bl,[si]

cmp bl,'+'

je tmp171

cmp bl,'-'

jne tmp173

mov login\_ok,1

tmp171:dec cx

jz tmp176

tmp172:inc si

mov bl,[si]

tmp173:cmp bl,'0'

jb tmp176

cmp bl,'9'

ja tmp176

sub bl,30h

movzx ebx,bl

imul eax,10

jo tmp176

add eax,ebx

jo tmp176

js tmp176

jc tmp176

dec cx

jnz tmp172

cmp dx,16

jne tmp174

cmp eax,7fffh

ja tmp176

tmp174:cmp login\_ok,1

jne tmp175

neg eax

tmp175:pop ebx

ret

tmp176:mov si,-1

jmp tmp175

tmp170 endp

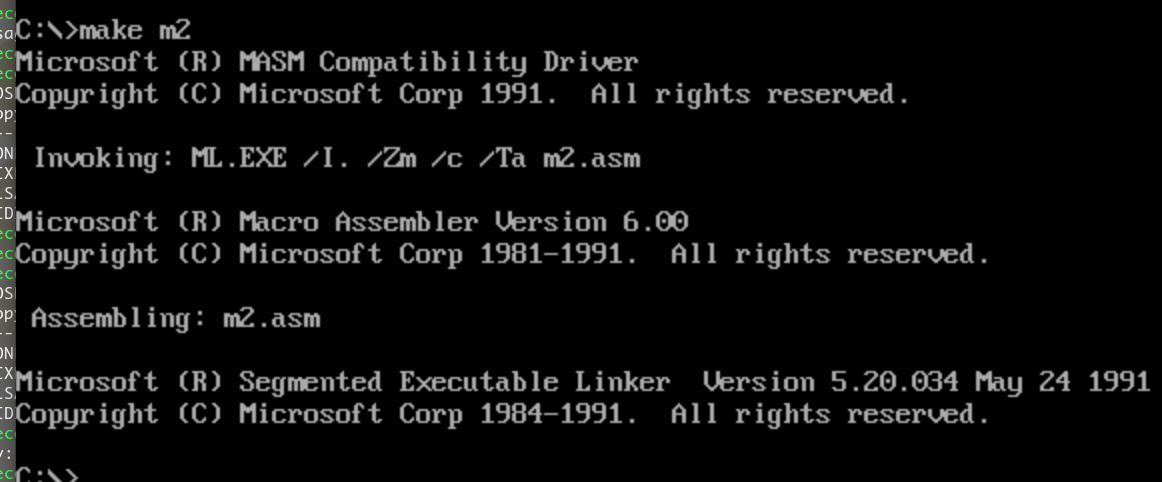
end start

end

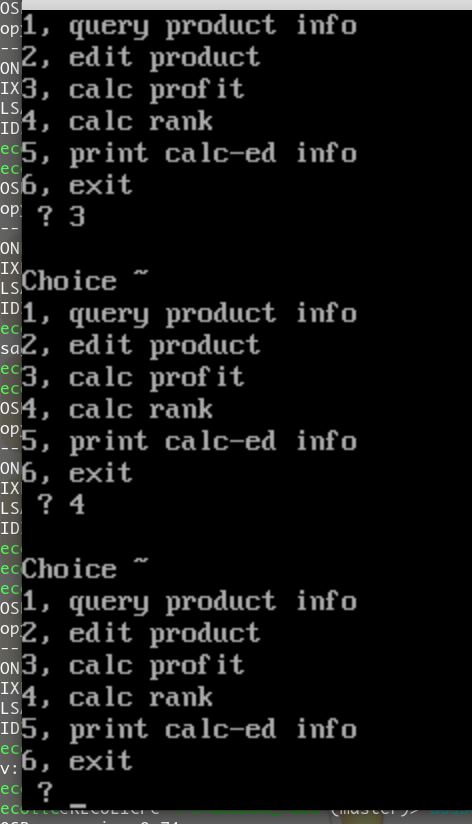
### 实验记录与分析

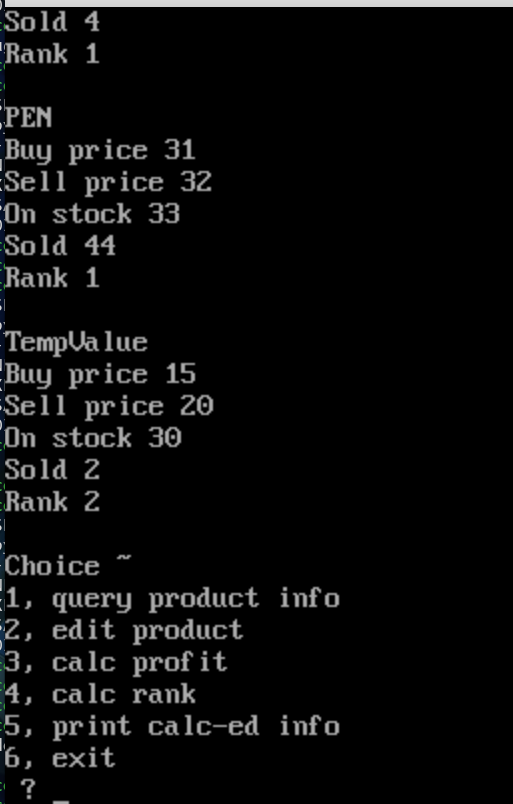
1. 首先完成模块二的8086asm的编写。

2. 如下图所示，汇编链接运行。

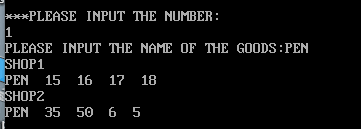
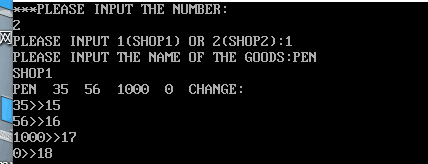
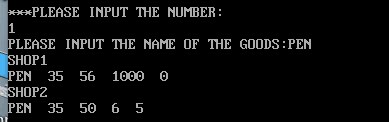
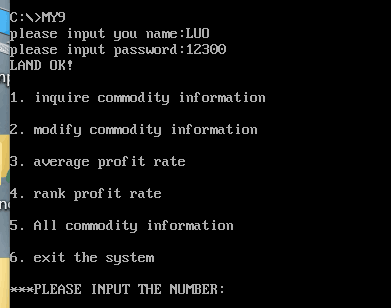


1. 分别测试3个功能。按照要求，3/4功能没有输出，功能5的输出取决于预设的数值，其他功能由模块1完成，不进行测试。





1. 模块1的测试如下，使用模块1提供的主控程序。



## 任务2

### 实验步骤

1. 根据任务2的要求编写C语言程序.
2. 修改模块化的汇编语言程序的符号，使其适合与C编译器生成的汇编代码链接。修改汇编程序中的调用，学习watcomC的标准库接口写法，使其能够调用libc。为了符合wlink的接口规定以便调用其提供的glibc，这里按照由当年的watcom官方文档提供的指导进行符号定义和汇编重写。例如，使用80387提供的浮点数运算指令重写profit计算部分，使用wcc指定的传参方式和返回方式，使用C语言内存布局。其中，由汇编实现不包含IO的利润率计算、排名计算等CPU密集型功能，并与C语言实现的IO程序进行链接。
3. make

### 源代码

~/t/d/exp32 ❯❯❯ make report

echo ';;;;;;;;;;;;main.c' && cat main.c && echo ';;;;;;;;;;;;;;;afx.h' && cat afx.h && echo ';;;;;;;;;;;;;impl.asm' && cat impl.asm && echo ';;;;;;;;;;;;;;;;makefile' && cat makefile

;;;;;;;;;;;;main.c

/\*\*

\* This source is extremely unsafe. You must not use it under any consequence

\* except hust homework.

\*/

#include "afx.h"

char user[32] = "bensong liu";

char pswd[32] = "test";

char shopA[16] = "SHOP1";

char shopB[16] = "SHOP2";

char products\_name[PRODUCT\_NUM][32] = {"pen", "book", "scala"};

int input\_price[PRODUCT\_NUM] = {35, 22, 61};

int output\_price[PRODUCT\_NUM] = {11, 28, 1000};

float profit\_table[PRODUCT\_NUM] = {0,0,0};

int rank\_table[PRODUCT\_NUM] = {-1,-1,-1};

bool logged\_in = false;

void echo(const char \*txt) {

printf("%s\r\n", txt);

}

void readline\_unsafe(char \*out) {

char c;

int pos = 0;

while(true) {

c = getchar();

if(c == '\r' || c == '\n')

break;

out[pos] = c;

++pos;

}

out[pos] = '\0';

}

bool login() {

char buffer[32] = "";

int cter;

again:

printf("Your name please: ");

readline\_unsafe(buffer);

if(buffer[0] == '\0')

goto again;

if(buffer[0] == 'q')

exit(0);

if(!streql(buffer, user)) {

echo("Permission denied.");

goto again;

}

printf("Your password please: ");

readline\_unsafe(buffer);

if(!streql(buffer, pswd)) {

echo("Permission denied.");

goto again;

}

echo("Passed.");

return true;

}

void func1\_ls() {

int cter;

for(cter = 0; cter < PRODUCT\_NUM; ++cter) {

printf("%s > InputPrice %d OutputPrice %d.\r\n", products\_name[cter], input\_price[cter], output\_price[cter]);

}

}

void func2\_edit() {

char name[32];

int input, output;

int pos;

echo("Give `$name $inputPrice $outputPrice` please.");

scanf("%s %d %d", name, &input, &output);

pos = search\_product(name);

if(pos == -1) {

echo("Not found error.");

return;

}

input\_price[pos] = input;

output\_price[pos] = output;

echo("Done.");

}

void func5\_print\_profit\_and\_rank() {

int cter;

for(cter = 0; cter < PRODUCT\_NUM; ++cter) {

printf("%s: profit mark %c rank %d\r\n", products\_name[cter], profit\_mark\_of(cter), rank\_table[cter]);

}

}

void func6\_exit() {

exit(0);

}

int main() {

char option[16];

login(); /\*Always true\*/

while(true) {

echo("");

echo("func1\_ls");

echo("func2\_edit");

echo("func3\_calc\_profit");

echo("func4\_calc\_rank (depend: func3)");

echo("func5\_print\_profit\_and\_rank (depend: func4)");

echo("func6\_exit");

printf(" 1-6 ? ");

readline\_unsafe(option);

switch(option[0]) {

case '1':

func1\_ls();

break;

case '2':

func2\_edit();

break;

case '3':

func3\_fill\_profit\_table();

break;

case '4':

func4\_fill\_rank\_table();

break;

case '5':

func5\_print\_profit\_and\_rank();

break;

case '6':

func6\_exit();

break;

}

}

};;;;;;;;;;;;;;;afx.h

#ifndef AFX\_H

#define AFX\_H

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

/\* C99 only \*/

typedef int bool;

#define true 1

#define false 0

#define streql !strcmp

#define PRODUCT\_NUM 3

void func1\_ls();

void func2\_edit();

void func3\_fill\_profit\_table();

void func4\_fill\_rank\_table();

void func5\_print\_profit\_and\_rank();

void func6\_exit();

char profit\_mark\_of(int index);

int search\_product(const char \*name);

#endif

;;;;;;;;;;;;;impl.asm

.387

PUBLIC func3\_fill\_profit\_table\_

PUBLIC func4\_fill\_rank\_table\_

PUBLIC profit\_mark\_of\_

PUBLIC search\_product\_

EXTRN \_\_STK:BYTE

EXTRN \_output\_price:BYTE

EXTRN \_input\_price:BYTE

EXTRN FIDRQQ:BYTE

EXTRN \_profit\_table:BYTE

EXTRN FIWRQQ:BYTE

EXTRN \_rank\_table:BYTE

EXTRN \_products\_name:BYTE

EXTRN strcmp\_:BYTE

EXTRN \_\_8087:BYTE

EXTRN \_\_init\_87\_emulator:BYTE

EXTRN \_small\_code\_:BYTE

DGROUP GROUP CONST,CONST2,\_DATA

\_TEXT SEGMENT BYTE PUBLIC USE16 'CODE'

ASSUME CS:\_TEXT, DS:DGROUP, SS:DGROUP

func3\_fill\_profit\_table\_:

mov ax,0cH

call near ptr \_\_STK

push bx

push dx

push si

push bp

mov bp,sp

sub sp,2

xor bx,bx

tmp1:

mov si,bx

shl si,1

mov ax,word ptr \_output\_price[si]

cwd

idiv word ptr \_input\_price[si]

dec ax

mov word ptr -2[bp],ax

fild word ptr -2[bp]

mov si,bx

shl si,1

shl si,1

fstp dword ptr \_profit\_table[si]

inc bx

nop

fwait

cmp bx,3

jl tmp1

mov sp,bp

pop bp

pop si

pop dx

pop bx

ret

func4\_fill\_rank\_table\_:

mov ax,0eH

call near ptr \_\_STK

push bx

push cx

push dx

push si

push bp

mov bp,sp

sub sp,2

xor cx,cx

jmp tmp3

tmp2:

inc cx

cmp cx,3

jge tmp6

tmp3:

mov bx,cx

shl bx,1

mov word ptr \_rank\_table[bx],1

xor dx,dx

jmp tmp5

tmp4:

inc dx

cmp dx,3

jge tmp2

tmp5:

mov bx,cx

shl bx,1

shl bx,1

mov si,dx

shl si,1

shl si,1

fld dword ptr \_profit\_table[bx]

fcomp dword ptr \_profit\_table[si]

fstsw word ptr -2[bp]

nop

fwait

mov ax,word ptr -2[bp]

sahf

jae tmp4

mov bx,cx

shl bx,1

inc word ptr \_rank\_table[bx]

jmp tmp4

tmp6:

mov sp,bp

pop bp

tmp7:

pop si

pop dx

pop cx

pop bx

ret

profit\_mark\_of\_:

push ax

mov ax,18H

call near ptr \_\_STK

pop ax

push bx

push dx

push bp

mov bp,sp

sub sp,10H

mov bx,ax

shl bx,1

mov ax,word ptr \_output\_price[bx]

cwd

idiv word ptr \_input\_price[bx]

dec ax

mov word ptr -4[bp],ax

fild word ptr -4[bp]

fstp dword ptr -8[bp]

fldz

fcomp dword ptr -8[bp]

fstsw word ptr -2[bp]

nop

fwait

mov ax,word ptr -2[bp]

sahf

jbe tmp8

mov al,46H

jmp tmp12

tmp8:

fld dword ptr -8[bp]

fst qword ptr -10H[bp]

fcomp qword ptr DGROUP:tmp16

fstsw word ptr -2[bp]

nop

fwait

mov ax,word ptr -2[bp]

sahf

jbe tmp9

mov al,41H

jmp tmp12

tmp9:

fld qword ptr -10H[bp]

fcomp qword ptr DGROUP:tmp17

fstsw word ptr -2[bp]

nop

fwait

mov ax,word ptr -2[bp]

sahf

jbe tmp10

mov al,42H

jmp tmp12

tmp10:

fld qword ptr -10H[bp]

fcomp qword ptr DGROUP:tmp18

fstsw word ptr -2[bp]

nop

fwait

mov ax,word ptr -2[bp]

sahf

jbe tmp11

mov al,43H

jmp tmp12

tmp11:

mov al,44H

tmp12:

mov sp,bp

pop bp

pop dx

pop bx

ret

search\_product\_:

push ax

mov ax,0aH

call near ptr \_\_STK

pop ax

push bx

push cx

push dx

push si

mov si,ax

xor bx,bx

mov cl,5

jmp tmp14

tmp13:

inc bx

cmp bx,3

jge tmp15

tmp14:

mov dx,bx

shl dx,cl

add dx,offset \_products\_name

mov ax,si

call near ptr strcmp\_

test ax,ax

jne tmp13

mov ax,bx

jmp near ptr tmp7

tmp15:

mov ax,0ffffH

jmp near ptr tmp7

\_TEXT ENDS

CONST SEGMENT WORD PUBLIC USE16 'DATA'

tmp16:

DB 0cdH, 0ccH, 0ccH, 0ccH, 0ccH, 0ccH, 0ecH, 3fH

tmp17:

DB 0, 0, 0, 0, 0, 0, 0e0H, 3fH

tmp18:

DB 9aH, 99H, 99H, 99H, 99H, 99H, 0c9H, 3fH

CONST ENDS

CONST2 SEGMENT WORD PUBLIC USE16 'DATA'

CONST2 ENDS

\_DATA SEGMENT WORD PUBLIC USE16 'DATA'

\_DATA ENDS

END

;;;;;;;;;;;;;;;;makefile

fake: main impl-asm

wlink name main.exe file main.obj,impl.obj

main:

wcc main.c -i=..\watcom\h

impl-asm:

wasm impl.asm

clean:

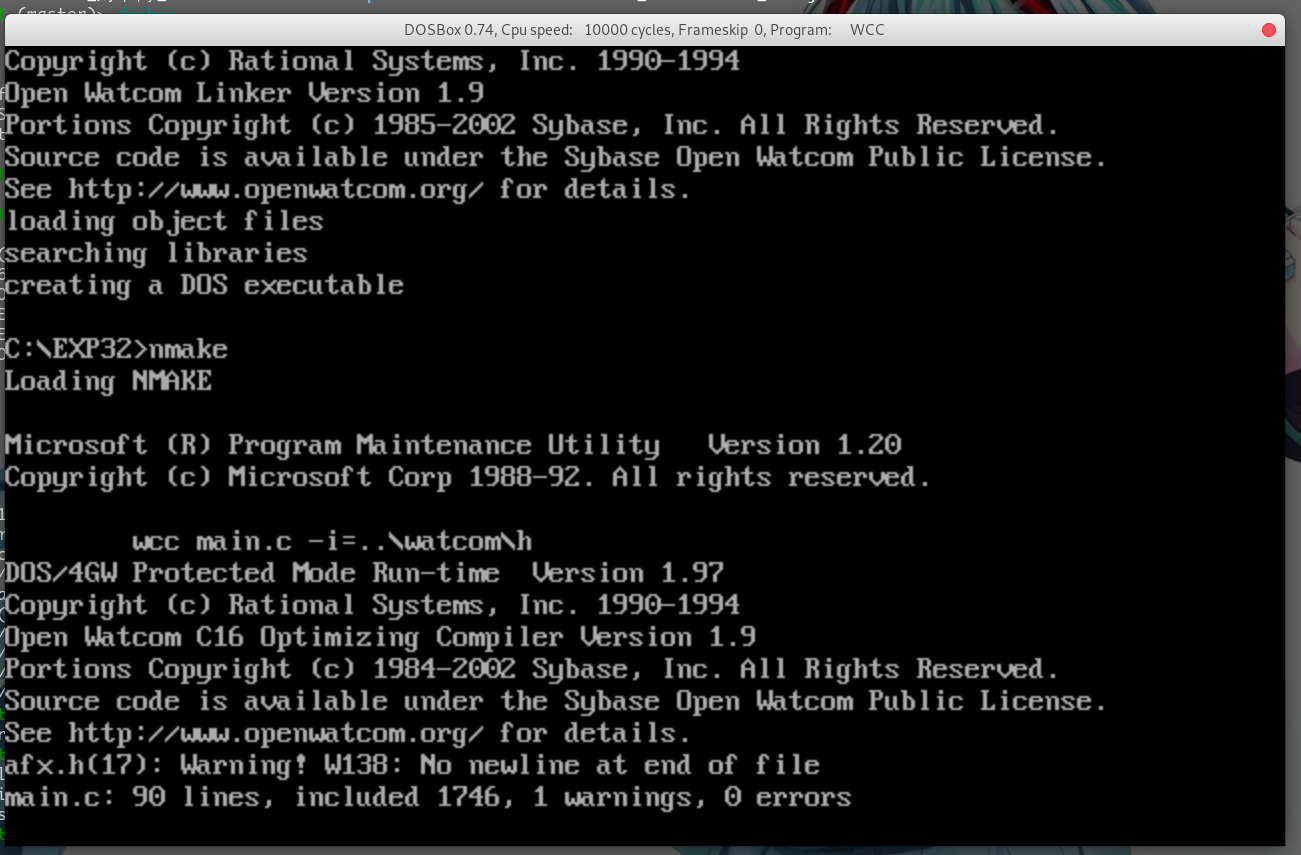
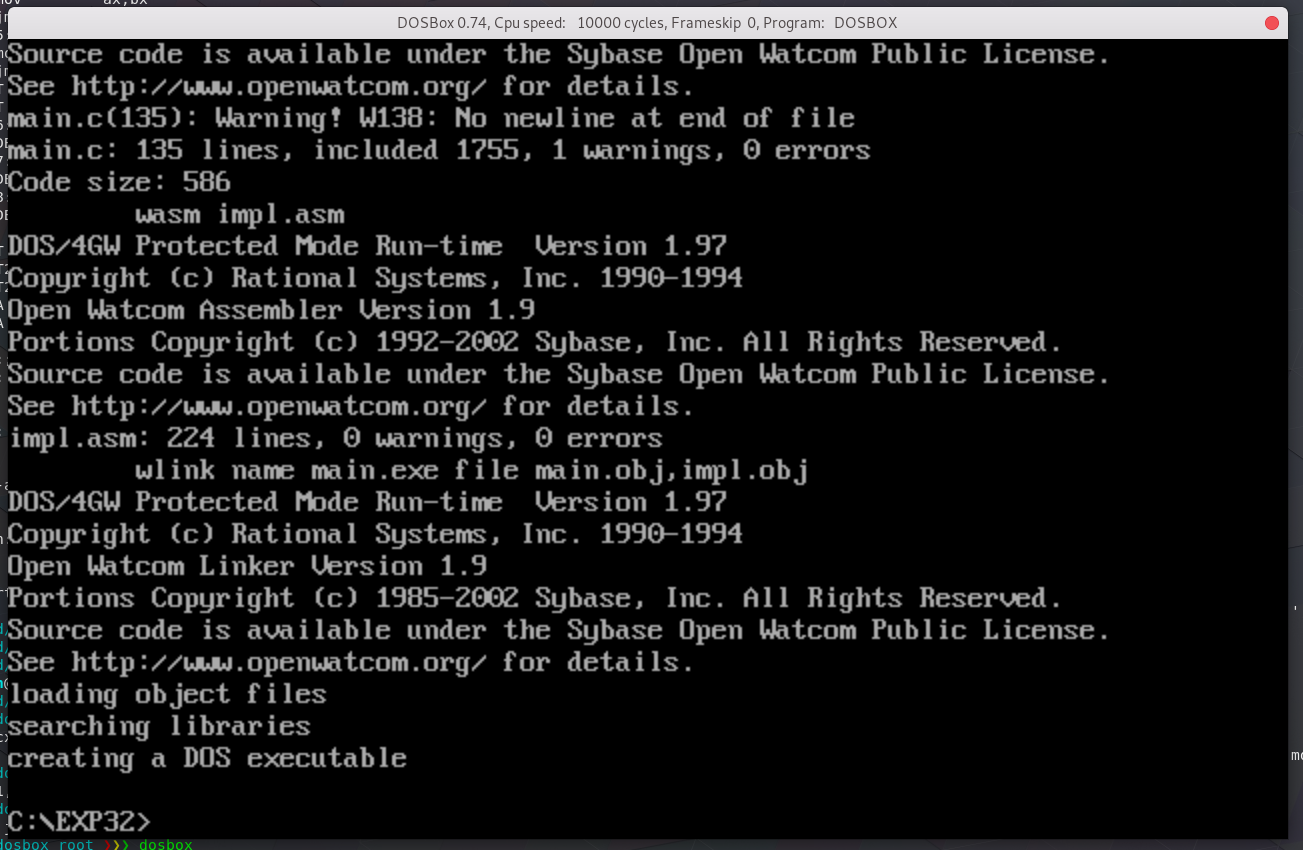
del main.obj main.exe impl.obj main.err

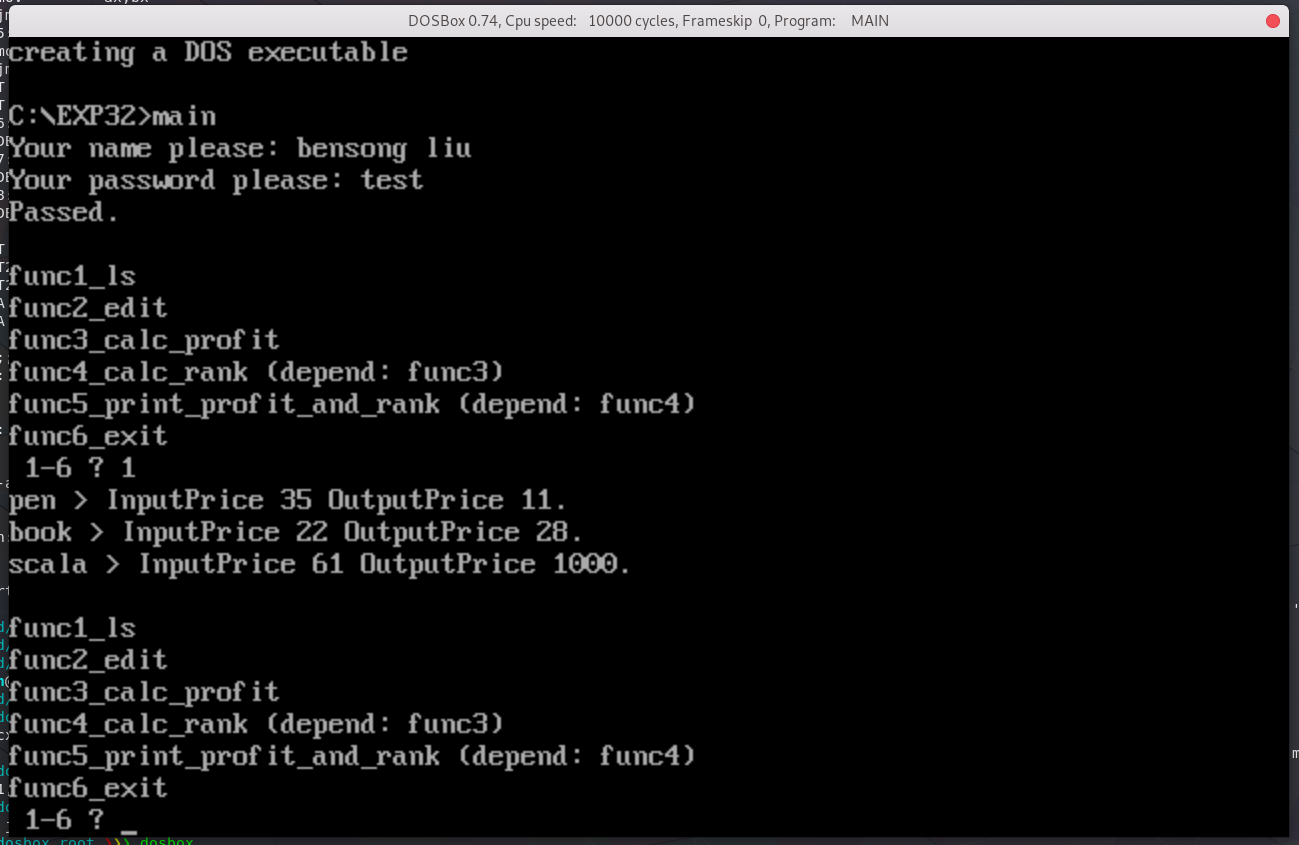
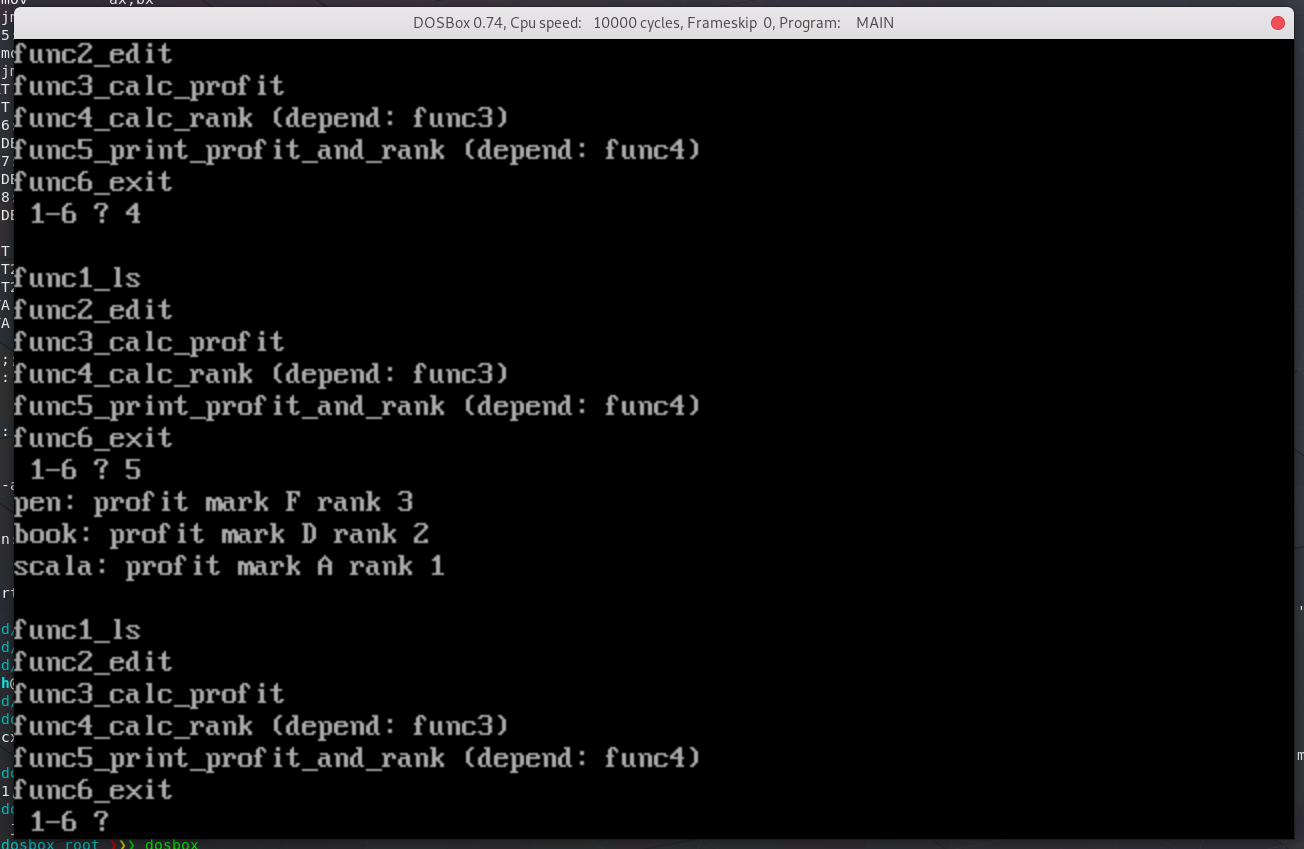
report:

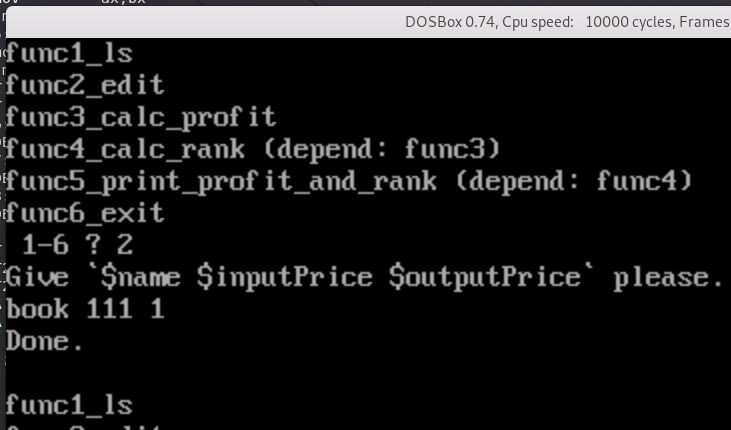
echo ';;;;;;;;;;;;main.c' && cat main.c && echo ';;;;;;;;;;;;;;;afx.h' && cat afx.h && echo ';;;;;;;;;;;;;impl.asm' && cat impl.asm && echo ';;;;;;;;;;;;;;;;makefile' && cat makefile

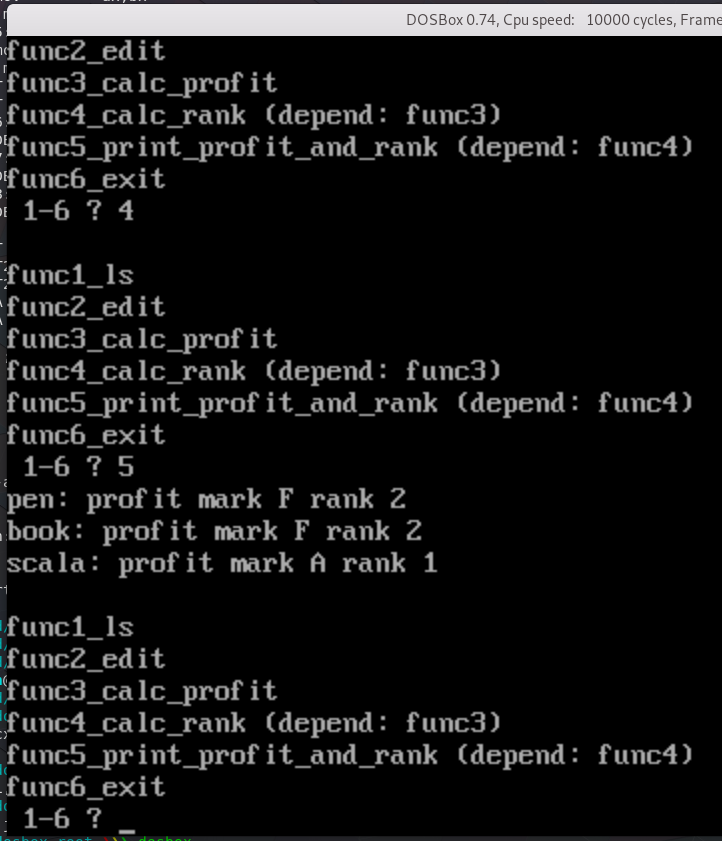
### 实验记录与分析

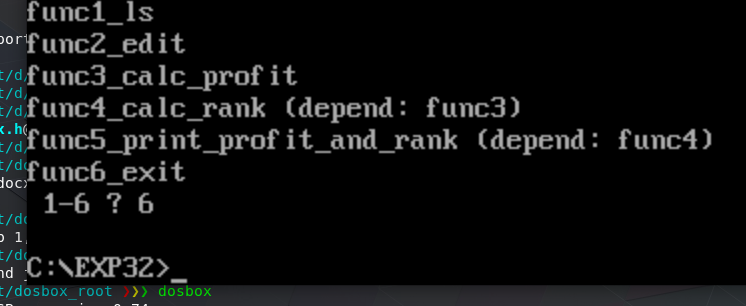
如图，在配置好OpenWatcom工具链之后，直接运行make或nmake进行编译链接。



然后，运行编译好的可执行文件，经过简单测试，功能正常。







# 总结与体会

在本次实验中，我主要掌握了模块化汇编的方法，了解到了在汇编语言中如何协同工作.任务一主要需要两人一组编写两个汇编模块并让两个模块协同工作。在这个实验中，我使用了宏来简化汇编程序，较少常用系统调用对程序员时间的开销。我在合作中实现345功能，是lib的提供者，但也在静态链接的过程中了解了public，extrn等符号的用法，也明白了如何修改段说明使得程序再连接时能够按照所预想的方式进行连接，以实现模块间互相调用，协同工作的结果。这也充分说明了20世纪90年代C语言编译器处理链接的过程中的代码翻译的方法，有利于加深对基础高级编程语言的了解。

在任务2中，OpenWatcom工具链相比Turbo系列显然具有优势。例如，wlink会自动将需要静态链接的libc到目标程序，wasm可以完全兼容masm的语法，但只有使用wasm生成的obj才能被wlink进行链接。需要特别注意的是，C语言是大小写敏感的语言，因此dos和C的习惯风格会发生冲突从而无法链接，必须注意符号对应的问题，我在此令asm服从C。同时，C语言倾向于使用struct表示product，但这会引入内存对齐的问题，并给汇编代码的移植带来问题。我在此使用尽最大可能减少汇编程序对全局变量的访问的问题来减少移植过程中的出错概率，并重新实现了所要求的asm调用接口。

# 参考文献

[1] Intel(R) 64 and IA-32 Architectures Software Developer’s Manual(https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf)

[2] (out-of-date) INTEL 80386 PROGRAMMER'S REFERENCE MANUAL 1986(https://css.csail.mit.edu/6.858/2015/readings/i386.pdf)

[3] (out-of-date) Open Watcom Toolset (http://www.openwatcom.org/)

[4] (out-of-date) Microsoft ASM Language for MS-DOS (https://en.wikipedia.org/wiki/Microsoft\_Macro\_Assembler)

[5] 80386 instruction set indexed by MIT.edu (https://pdos.csail.mit.edu/6.828/2017/readings/i386/c17.htm)

[6] (out-of-date) MASM directives (http://stanislavs.org/helppc/directives.html)

[7] (out-of-date) DOS Interrupts Reference by SCU.edu.au (http://spike.scu.edu.au/~barry/interrupts.html)

[8] DOSBox: DOS Simulator for modern computer(not a VM) (https://www.dosbox.com/wiki/)

[9] x86 arch introduction by wikibooks.org (https://en.wikibooks.org/wiki/X86\_Assembly/X86\_Architecture)