

**Course Lab Report**

**Course Name: Assembly Language Programming Experiment**

**Experiment Name: Experiment 3 Modular Programming**

**Experiment time: 20 18 - 4 - 16, 14:00-17:30 Experiment location: Experiment platform No. 90, Room 804, South 1st Floor**

**20 18-4-23 , 14 : 00-17:30**

**Instructor: Li Haibo**

**Professional class: school handover 201601 class**

**Student ID: U2016 12696 Name: Chen Haorui**

**Student in the same group: Jing Changjin Date of report: April 24, 2017**

**statement of originality**

I solemnly declare that the content of this report is independently completed by me, and the references to viewpoints, methods, data and literature have been pointed out in the text. Except for the content cited in the text, this report does not contain any other individual or collective published works or achievements, and there is no plagiarism or plagiarism.

Hereby declare!

Student signature:

Date: 2018.4.24

performance evaluation

|  |  |  |
| --- | --- | --- |
| Experiment completion quality score (70 points) (experimental steps are clear, detailed and in-depth, experimental records are true and complete, etc.) | Report writing quality score (30 points) (report specification, complete, smooth, detailed, etc.) | Total score (100 points) |
|  |  |  |

Instructor's signature:

date:

**Table of contents**

[1](#_Toc515195831)  [Experimental purpose and requirements](#_Toc515195831)  [1](#_Toc515195831)

[2](#_Toc515195832)  [Experimental content](#_Toc515195832)  [1](#_Toc515195832)

[3](#_Toc515195833)  [Experimental process](#_Toc515195833)  [3](#_Toc515195833)

[3.1](#_Toc515195834)  [Task 1](#_Toc515195834)  [3](#_Toc515195834)

[3.1.1](#_Toc515195835)  [Design idea and storage unit allocation](#_Toc515195835)  [3](#_Toc515195835)

[3.1.2](#_Toc515195836)  [Experimental Step](#_Toc515195836)  [4](#_Toc515195836)

[3.1.3](#_Toc515195837)  [Flowchart](#_Toc515195837)  [4](#_Toc515195837)

[3.1.4](#_Toc515195838)  [Source program](#_Toc515195838)  [5](#_Toc515195838)

[3.1.5](#_Toc515195839)  [Experimental records and analysis](#_Toc515195839)  [12](#_Toc515195839)

[3.2](#_Toc515195840)  [Task 2](#_Toc515195840)  [16](#_Toc515195840)

[3.2.1](#_Toc515195841)  [Design ideas and variable description](#_Toc515195841)  [16](#_Toc515195841)

[3.2.2](#_Toc515195842)  [Source program](#_Toc515195842)  [16](#_Toc515195842)

[3.2.3](#_Toc515195843)  [Experimental steps](#_Toc515195843)  [23](#_Toc515195843)

[3.2.4](#_Toc515195844)  [Experimental records and analysis](#_Toc515195844)  [23](#_Toc515195844)

[4](#_Toc515195845)  [Summary and experience](#_Toc515195845)  [25](#_Toc515195845)

[Reference](#_Toc515195846)  [27](#_Toc515195846)

# Experimental purpose and requirements

The main purpose and requirements of this experiment are the following 6 points. All tasks will be carried out around these 6 points. I hope you can check whether you have met these goals and requirements afterwards.

1. Master the methods and techniques of subroutine design, and be familiar with the parameter transfer method and calling principle of subroutines;
2. Master the design method of macro instruction and modular program;
3. Master the cooperative development and debugging methods of large-scale programs;
4. Master the method of mixed programming of assembly language program and C language program;
5. Familiar with the basic optimization method of C compiler;
6. Understand the naming method of the C language compiler and the mechanism of parameter passing between the main program and the subroutine.

# Experimental content

Task 1 Macro and subroutine design (try to complete it in the first 4 class hours, and the experimental report before going to the computer should complete the content of this task except the experimental record and analysis, summary and experience)

Further modify and enhance the function of the online store commodity information management program of the experiment one task four, mainly adjust the function three.

1. Description of the adjusted function 3

(1) First, a function menu is displayed (the format is self-defined. If it is not logged in, only menus "1" and "6" are displayed):

1=query product information, 2=modify product information, 3=calculate average profit margin,

4=Calculate profit margin ranking, 5=Output all product information, 6=Exit the program.

Enter a number from 1-6 to enter the corresponding function.

(2) Query product information

Prompt the user to enter the product name to be queried. If the product cannot be found in the first online shop, it will be prompted to enter the product name again. If only press Enter, it will return to function three (1).

After finding the product, display the information of the product in the order of: "SHOP1, product name, sales price, total purchases, sold quantity", and also display the information of the product in "SHOP2". Return to function three (1) after displaying.

(3) Modify product information

Prompt the user to enter the product name to modify the information (specify the online store name first). [If the next processing steps are written as a subroutine, then the name of the online store and the product name (or its offset address) are the entry parameters of the subroutine, whether it is found, whether it is a carriage return or whether the modification is successful is the exit parameter]. If the product cannot be found in the designated online store, it will be prompted to enter the online store name and product name again. If only press Enter, it will return to function three (1).

After finding the product, display the original values one by one in the order of: purchase price, sales price, and total purchases, and then enter the new value (if the input is wrong, display and modify the information again. If you return directly car, the information will not be modified).

Such as: purchase price: 25》24 //The symbol "》" is only used as a separator, and other separators can also be selected

Sales price: 46》5A6 //An illegal value has been entered, and the next line will display and enter again

Sales price: 46》56

The total number of purchases: 30》//When you press Enter directly, this information will not be modified

After processing all three items of information, return to function three ( 1).

(4) Calculate the average profit margin

First calculate the profit rate PR1 of the first product in SHOP1, then find the product in the SHOP2 online store, and calculate its profit rate PR2. Finally, calculate the average profit rate APR=(PR1+PR2)/2 of the commodity, and save it in the profit rate field of SHOP1. Repeat the above steps to calculate the average profit rate of each commodity in turn. Back to function three (1).

(5) Calculate profit margin ranking

Each product in SHOP2 is ranked according to the size of the average profit rate, and the ranking information is stored in the field of profit rate of the product in SHOP2. Back to function three (1).

(6) Output all product information

Display all product information in SHOP1 and SHOP2 on the screen, including the average profit rate and ranking (replacing the original profit rate field of the product). The specific display format can be defined by yourself (it can be displayed by online store, or it can be displayed according to the product ranking, etc., and the display method can be used as the entry parameter of the subroutine). Back to function three (1).

2. Other requirements

(1) A team of two people, one person is in charge of the main program including menu display, program exit, and the functions (1) and (2) in the menu; the other person is in charge of the functions (3), (4) and (5). Each assembles its own module, designs a test method, and passes the test; then hand over its own module to the other party, each integrates the other party's program into its own program, connects to generate a program, and then performs overall debugging.

Note that at the beginning of each module, the name of the author and the names of the classmates in the same group are indicated. When integrating them together, pay attention to delete the extra codes added during your own testing. If there are duplicate modules (for example: two people will use the base conversion subroutine, each module may have the same base conversion program). Duplicate parts need to be removed.

Suggested grouping method: two people in a group according to the order of the student number. If the number of students in the class is odd, the last three people will be in a group (two of them have the same division of labor, and the third person only needs to choose one of the students' modules and their own module) integrated).

(2) The basic requirement of ranking is to calculate the ranking according to the average profit rate from high to low, and you can also consider ranking according to specified fields (such as sold quantity, etc.). The ranks are the same when the average profit rate is the same, and the ranking of the next adjacent average profit rate should be the next value of "sum" of all the top-ranked commodity categories.

(3) Define the No. 9 and No. 10 DOS system function calls as macro instructions and call them. Functions (1)-(5) should be implemented as subroutines as much as possible. Need to refer to the base conversion program in the book: the subroutine F10T2 for converting decimal to binary and the subroutine F2T10 for converting binary to decimal.

Task 2: Call a function implemented in assembly language in a C language program

(It is mainly completed in the second 4 in-class hours. The experimental report before going to the computer should complete the content of the task except the experimental record and analysis, summary and experience)

For the transformation of the program of task 1, the main control program and a certain function with more input and output (such as one of the functions (1), (2) and (5)) are implemented in C language, and other functions are implemented in independent Implemented in the form of assembly language subroutines; call assembly language subroutines in C language programs.

**Requirements and tips:**

(1) In different C language development environments, the mixed programming with assembly language programs may be implemented in different ways. Please choose the C language development environment you are familiar with and find relevant information to complete this experiment (BC31 is recommended, its function and operation method are relatively simple).

(2) In the experiment report, give your development environment and its implementation method in more detail.

(3) Observe the naming rules of various symbols in the C language compiler (referring to the naming rules that can be recognized inside the compiler, for example, whether to add an underscore "\_" in front of the symbol name, etc.), parameter transfer between the main program and the subroutine The mechanism, the method of reclaiming the stack space after passing parameters through the stack (to design a C function with multiple parameters that need to be passed).

(4) For the execution program formed by mixed programming, use a debugging tool to observe the relationship between the program code formed by C language and the program code formed by assembly language, including the value of segment and offset, and assembly instructions to access C variables How is it translated, etc.

(5) Please try to unreasonably embed assembly language instruction statements in the C language source program to destroy the correctness of the C language program. For example, add an assembly instruction statement that modifies the AX register (or other registers such as DS) in the middle of several consecutive C language statements, and the content of AX should not be modified here, so that it can be observed that the C language program is damaged The effect of correctness (this experiment shows that in the C language program, if the assembly instruction statement is randomly embedded without considering the machine code that the upper and lower statements are translated into, there may be a risk of error).

(6) Observe the impact of the optimization strategy of the C compiler on the code. Through actual observation and analysis, the efficiency of recording the assembly language program in this experiment will be better than that of the C language program (at least one observation is given).

(7) By debugging a mixed programming program, experience the difference from the debugging process of a program written in pure assembly language.

(8) Through this experiment, I hope everyone understands that different programming languages can work together to solve a problem, and can use the characteristics of different languages to better solve problems; use the knowledge of assembly language to better understand advanced programming languages. The internal processing principles and strategies of the language provide support for writing better C language programs and making good use of C compilers.

# experiment procedure

## task 1

### Design Thought and Storage Unit Allocation

function 3 is the same as experiment 2, function 3 (2) , and will not be repeated here.

In function 4, first initialize the serial number of the same element of the temporary array according to the number of valid products , find the same product in store 2 for each product in store 1 , and store the profit EA in store 1 into the temporary array Corresponding position with the product in store 2 . Sort the elements in the array in descending order according to the size of the elements corresponding to the EAs of the temporary array elements , and then change the serial numbers to the same if there are two elements pointing to the same value. For each product in store 2 , find an element with the same EA in the temporary array according to the EA stored in its corresponding position , put the serial number of the element into the position where the EA was stored before , and return.

In function five , based on each element in store one, first find the same product in store two, and output the name of the product verbatim . Call the base conversion subroutine MRADIX to convert the average profit stored in the merchandise of Store 1 into ASCII codes of decimal numbers and output them , and finally convert their rankings to the corresponding ASCII codes and output them .

1. Storage unit allocation

TIP\_12 : The string with a small length and ending with $ is used to make intervals when outputting different types of data in function 5 .

M : stores the number of valid products in the store , which can be modified according to specific situations .

TAD : Used in function 4 , a 90-byte temporary array , 3 bytes of which are used as an element, the first byte of each element stores the serial number of the current element ( may not be initialized) , and the last two words The section stores the 16-bit offset address.

HKU : store the number 100, used in function 3 for multiplication.

2. Register allocation: 3 functions combined use all registers. See the comment section before each subroutine in the source program for details .

### Experimental procedure

1. Prepare the computer experiment environment, rename the copy of task 2 of experiment 2 to t1-, and modify the program on this basis .

2. Write function 3 - function 5 according to the idea of 3.1.1 .

3. Repeat the compilation-modification process until the compilation no longer reports errors.

4. Enter td single-step debugging, check the logic errors in the program and modify them.

5. Extract function 3 to function 5 in t1- to form a new file, adapt to the directory provided by the same group of students and the remaining two functions , modify the section name , and add function and variable attributes.

6. Repeat the compilation-modification process until the compilation no longer reports an error.

7. Enter td single-step debugging, check the logic errors in the program and modify them.

### flow chart

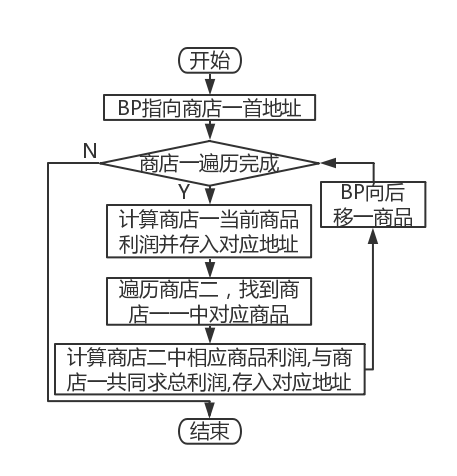
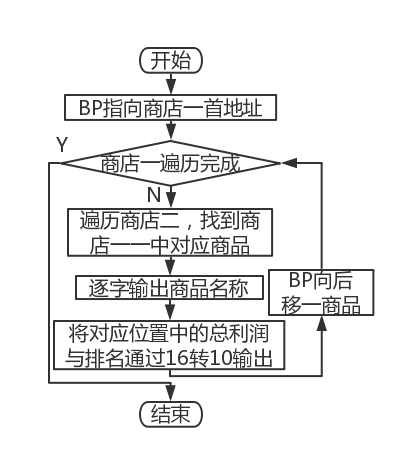
 

Figure 3.1.1 Task 3 flowchart Figure 3.1.2 Task 5 flowchart

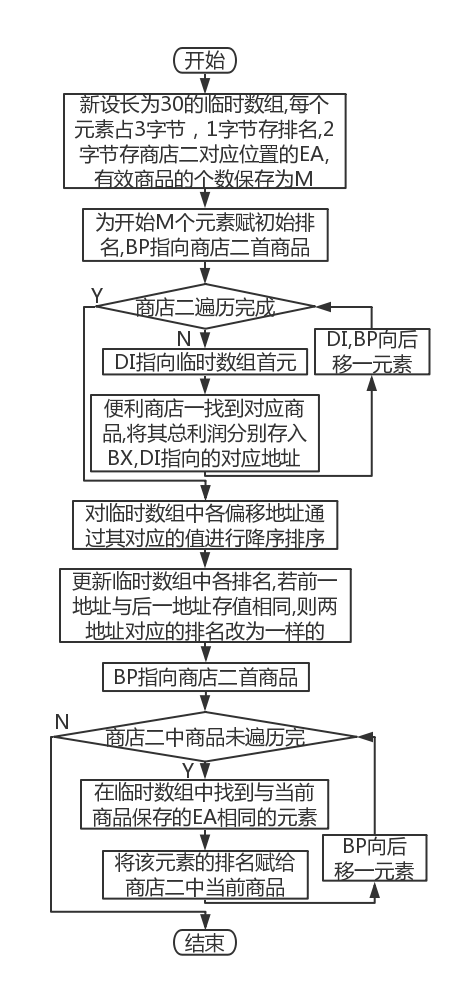


Figure 3.1.3 Task 4 flow chart

### source program

T1-.ASM:

.386

INTR MACROT ; The parameter is part of the register name

MOV SI,DS:[B&T&+10] ; Purchase price

IMUL SI,DS:[B&T&+14] ; Purchase quantity

MOV AX,DS:[B&T&+12] ; Sales price

IMUL AX,DS:[B&T&+16]; Sold Quantity

SUB AX, SI

IMUL HKU

IDIV SI

ENDM

FND MACROCOUT, PIN, A, B, N; The parameter is the return label when it succeeds

MOV SI,0

LP&A&: ; Get the current product string length in store 1 and save it in AX(AL)

CMPBYTE PTR DS:[BP+SI],0

JEBRK&N&

INC SI

JMPLP&A&

BRK&N&:

MOV AX,SI

MOV CH,M ; CH is responsible for the traversal of store two

LEA BX, PIN

LP&B&:

CALL CMPST ; Compare whether the two commodity strings are the same, if not, AH is 0 (default value)

ORAH, AH

JNZCOUT

ADDBX,20 ; BX moves to the next item in store 2, continue to compare

DECCH

JNZLP&B&

JMPSTEP1 ; The products in store 2 have been traversed, but still not found

ENDM

;----------------------------

STACK SEGMENT STACK USE16

┇ ; After this line, it is defined for the relevant data of Experiment 2

┇ ; This line was previously defined for the relevant data of Experiment 2

AUTH DB? ; Determine the login method

TAD DB90DUP(0); Temporary array storing ranking and offset addresses

TNM DB3DUP(0); Temporary array for storing decimal profits

HKU DW100

┇ ; This line was previously defined for the relevant data of Experiment 2

┇ ; Before this line is the relevant code of Experiment 1, Task 5, Function 2

STEP3:

LEA DX,TIP\_13 ;menu

MOV AH,9

INT 21H

MOV AH,1 ; input option

INT 21H

LEA DX, TIP\_9

MOV AH,9

INT 21H

CMP AL, '3'

JESTEP3\_3

CMP AL,'4'

JESTEP3\_4

CMPAL,'5'

JESTEP3\_5

JMPSTEP4

MOV CL,M ;Outer loop counter

LEA BP,GA1

LPB:

LEA BX, IN\_GOOD

INC BX

MOV DX,[BX]

MOV DH,0

LPBA: ; Compare the current string in store 1 with the input string

LEA BX, IN\_GOOD

INC BX

MOV AX,[BX]

MOV AH,0

SUB AX,DX

MOV SI,AX

MOV BX,DS:[BP+SI]

MOV BH,0

MOV DI,BX

LEA BX, IN\_GOOD

ADD BX,2

MOV AX,SI

XLAT

CMP DI, AX

JNEBRK1 ; Exit the loop directly when the current character is different

DEC DX

JNE LPBA ;Continue to the next cycle when SI is not 0

BRK1:

CMP DX,0 ;If SI is not 0, it means that the traversal of the input string has not reached the end, that is, the strings are different, and the next string should be compared; if it is 0, it will jump out of the outer loop

JNEBRK13

INC SI

MOV BX,DS:[BP+SI]

CMP BX,0

JEBRK2

BRK13:

ADD BP,20 ; BP moves to the next item in store one

DEC CL

JNELPB ; The product has not been searched

BRK2:

CMP CL,0 ; CL is not 0, indicating that the product has been found

JNEBRK3

LEA DX, TIP\_4 ; Display prompt: re-enter the product name to be checked

MOV AH,9

INT 21H

JMP STEP3

BRK3:

CMP AUTH,1

JNE BRK6

MOV CX,DS:[BP+16]; Sold quantity (for comparison)

CMP CX, DS:[BP+14]; Purchase quantity

JBBRK15 ; The sold quantity is lower than the purchased quantity

LEA DX, TIP\_10 ; Display prompt: the sold quantity is higher than the purchased quantity

MOV AH,9

INT 21H

JMP STEP1

BRK15:

INC WORD PTR DS:[BP+16] ;Sold quantity plus one

STEP3\_3:

CALL PART3

JMP STEP3

STEP3\_4:

CALL PART4

JMP STEP3

STEP3\_5:

CALL PART5

JMP STEP3

┇ ; After this line is experiment 1 task 5 function 3 related codes

┇ ; Before this line is the relevant code of Experiment 2 C MPST

; Subroutine name: PART3

;Function: Calculate the total profit margin of each commodity in the two stores

; Entry parameters: none

;exit parameters: none

;Registers used:

;CL——outer loop counter

;BP——pointer to the item in store 1

PART3 PROCUSES AX CX SI BP

MOV CL,M ; CL is responsible for the traversal of store one

LEA BP,GA1

LPC: ;The previous values of all registers can be invalidated

FND BRK20, GB1, N, O, 21 ; cyclically compare the string pointed to by BP and BX, if they are the same, go to BRK20

BRK20: ; Update the average profit rate of each product (CL, BX, BP are occupied)

INTR P ; Calculate the profit margin of the commodity pointed to by BP

MOV DS:[BP+18],AX ;Save BP profit

INTR X ; Calculate the profit margin of the commodity pointed to by BX

ADD AX,DS:[BP+18] ; Add two profits

CWD

SAR AX,1 ; average profit

MOV DS:[BP+18],AX ;The average profit is stored in the profit field of SHOP1

DEC CL

OR CL,CL

JZ BAR5

ADD BP,20 ; BP moves to the next item in store one

JMP LPC

BAR5:

RET

PART3 ENDP

; Subroutine name: PART4

;Function: sort the products in the two stores according to the total profit margin, and put them in the corresponding EA of the corresponding products in the second store

; Entry parameters: None

;exit parameters: none

;Registers used:

;BX—pointer to the item in store 1

;BP——pointer to the product in store 2

;CL——outer loop counter

;CH——inner loop counter

;DI——pointer to temporary array/do index addressing register

;SI——do index addressing register

PART4 PROCUSES AX BX CX DX BP SI DI

MOV BX,OFFSETTAD

MOV CL,M ; CL is responsible for the traversal of store two in LPD

XOR CH,CH

LPF: ; Initialize the temporary array number

INC CH

MOV [BX],CH

ADD BX,3 ; BX moves to the next structure

CMP CH, CL ; Initialize the label according to the number of valid commodities

JNE LPF

LEA BP,GB1

LEA DI,[TAD+1]

LPP: ;Find the corresponding product in store 1, store the total profit address in SHOP2 and the temporary array

FND BRK4,GA1,L,M,17 ; cyclically compare the string pointed to by BP and BX, if they are the same, go to BRK4

BRK4:

ADD BX,18

MOV DS:[BP+18],BX; Store the offset address of the total profit of the commodity in store 1 into store 2

MOV [DI],BX ; Store the offset address of the total profit of the commodity in store 1 into a temporary array

ADD DI,3 ; DI moves to the next element in the temporary array

DEC CL

OR CL,CL

JZ BRK5

ADD BP,20

JMP LPP

BRK5: ; Sort the elements in the temporary array

LEA BX,[TAD+1] ;Get the first address of the array

MOV CL,M ; Control the number of loops

XOR CH,CH

XOR SI, SI ; clear SI

MOV DI,3

MOV AL,M ; Offset upper limit

XOR AH,AH

IMUL AX,3

L1: MOV BP, [BX+SI] ; Use base indexing to fetch operands, L1 is the outer loop, (SI) is the loop variable, ; equivalent to i

L2: CMP BP, [BX+DI] ; L2 is the inner loop, (DI) is the loop variable, equivalent to j

JGE L3

XCHG BP, [BX+DI] ;[BX+SI]<[BX+DI], exchange

MOV [BX+SI], BP ; Send the new value of BP back to [BX+SI]

L3: ADD DI,3 ;AH>=AL, no need to exchange, (AH) is directly compared with the last number, which is equivalent to j++

CMP DI,AX ; Determine whether the inner loop is over

JB L2

ADD SI,3 ; Add 3 to the outer variable SI

MOV DI,SI ; equivalent to j=i

ADDDI,3

LOOP L1 ; exchange between two memory data through registers

LEABX, TAD ;Modify the ranking value when the profit is equal

MOV SI,0

LPG:

MOVAX,[BX+SI]

MOV CX,[BX+SI+3]

MOVDX,[EAX]

CMP DX,[ECX] ; Compare the relationship between the two profit margins before and after

JNEBRK11

MOV CL,[BX+SI]

MOV[BX+SI+3],CL; the serial number becomes the same

BRK11:

ADD SI,3

MOV DL,M

XOR DH,DH

DEC DX

IMUL DX,3

CMP SI,DX

JBLPG

LEA BP, GB1 ; Compare the address stored in the corresponding position in store 2 with the temporary array, if they are the same, assign the corresponding serial number in the temporary array to the corresponding product in store 2

XOR SI,SI ; SI is set to 0

XOR CH,CH ; outer loop counter

LPH:

XOR DI,DI

XOR CL,CL ; inner loop counter

LPH\_1:

MOVAX,[BX+DI+1]

CMP AX,DS:[BP+SI+18]

JNEBRK12

MOV AL,[BX+DI]

XOR AH,AH

MOV DS:[BP+SI+18],AX

JMP BRK14

BRK12:

ADD DI,3 ;DI points to the next address in the temporary array

INC CL

MOV AL,M

CMP CL,AL

JNELPH\_1 ;The current product ranking has not been generated

MOVWORD PTR DS:[BP+SI+18],0;Safety statement, if no address identical to the target product is found in the temporary array, set its rank to 0

BRK14:

ADD SI,20 ; SI points to the next item in store 2

INC CH

MOV AL,M

CMP CH,AL

JNE LPH ;Store 2 has not been traversed

RET

PART4 ENDP

; Subroutine name: PART5

;Function: output the name of each commodity, average profit and ranking

; Entry parameters: None

;exit parameters: none

;Registers used:

;BX——pointer to the item in store 2

;BP——pointer to the item in store 1

;CL——outer loop counter

;CH——inner loop counter

;DI——pointer to temporary array/do index addressing register

;SI——do index addressing register

;DL——for output characters

;AX - average profit

PART5 PROCUSES AX BX CX DX SI

MOV CL,M ; CL is responsible for the traversal of store one

LEA BP,GA1

LPI: ;The previous values of all registers can be invalidated

FND BRK16,GB1,J,K,18 ; cyclically compare the string pointed to by BP and BX, if they are the same, go to BRK16

BRK16:

XOR SI,SI

MOV CH,AL

LPI\_A: ;Loop output commodity string

MOV DL,[BX+SI]

MOV AH,2

INT 21H

INC SI

DEC CH ; The product string has not been output

JNZ LPI\_A

LEADX,TIP\_12 ; tabulation

MOV AH,9

INT 21H

MOV AX,DS:[BP+18]; get average profit

MOV DX,AX

AND DX,8000H

CMP DX,0

PUSH BX

MOV BX,10 ; target base

JZBRK22

DEC AX ; return source code from complement (stripping)

NOT AX

PUSH AX

LEA SI, TNM ; SI points to a temporary array

MOV DL,'-' ; output negative sign

MOV AH,2

INT 21H

POPAX

BRK22:

CALL MRADIX ; Convert profit to decimal and output ASCII characters

LEADX,TIP\_12 ; tabulation

MOV AH,9

INT 21H

POP BX

MOV DL,[BX+18] ;Extract ranking

ADD DL,30H

MOV AH,2

INT 21H

LEADX, TIP\_9 ; Newline

MOV AH,9

INT 21H

DEC CL

JZBAR6 ; Commodities have not been traversed

ADD BP,20

JMP LPI

BAR6:

RET

PART5 ENDP

; Subroutine name: MRADIX

;Function: Convert 16-bit unsigned binary number in AX to P-ary number (16-bit segment)

; Entry parameters:

;AX——store the 16-bit unsigned binary number to be converted

;BX——store the base to be converted

;exit parameters: none

;Registers used:

;CX——The counter when P-ary numbers are pushed into the stack and popped out of the stack

;DX——store the high digit or remainder of the dividend when doing division

MRADIX PROCUSES CX DX

XOR CX,CX

LPQ:

XOR DX,DX

DIV BX

PUSH DX

INC CX ; Save digits

OR AX,AX

JNZ LPQ

LPR:

POP AX

CMP AL,10

JB BRK23

ADD AL,7

BRK23:

ADD AL,30H

MOV DL,AL ; output the current character directly

MOV AH,2

INT 21H

LOOP LPR

RET

MRADIX ENDP

;----------------------------

CODE ENDS

END START

W an1 .asm

NAME WAN1

EXTERN CRLF:BYTE,GA1:BYTE,GB1:BYTE

PUBLIC PART3, PART4, PART5, SET

.386

WRITE MACROOUT\_BUF; macro definition instead of call No. 9

LEA DX, OUT\_BUF

MOV AH,9

INT 21H

ENDM

┇ ; This line is followed by t1-related macro definitions

┇ ; This line is preceded by t1-related macro definitions

;----------------------------

DATA1 SEGMENTUSE16PARA PUBLIC 'D1'

TIP\_12 DB'$'

M DB3 ; Number of valid commodities

TAD DB90DUP(0); Temporary array storing ranking and offset addresses

HKU DW100

TNM DB3DUP(0); Temporary array for storing decimal profits

DATA1 ENDS

STACK SEGMENTUSE16PARASTACK'STACK'

DB 200 DUP(0)

STACK ENDS

;------------------------------

CODE SEGMENT USE16PARA PUBLIC 'CODE'

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

; Subroutine name: SET

;Function: Load data segment DATA1 into ES

; Entry parameters: none

;exit parameters: none

;Registers used:

;AX——as a transfer register

SET PROC

MOVAX, DATA1

MOVES, AX

RET

SET ENDP

┇ ; This line is followed by t1-subroutines

### Experiment Recording and Analysis

1. Experimental environmental conditions: Intel® Core™ i5-3230M CPU 2.60GHz, 2.86G memory; DOSBox0.74 under WINDOWS 7; notepad++ 7.55; MASM.EXE 6.0;LINK.EXE 5.2;TD.EXE 5.0.

2. According to the idea of 3.1.1 , write the program, assemble and connect , and there is no problem in the process, as shown in Figure 3.1.4.

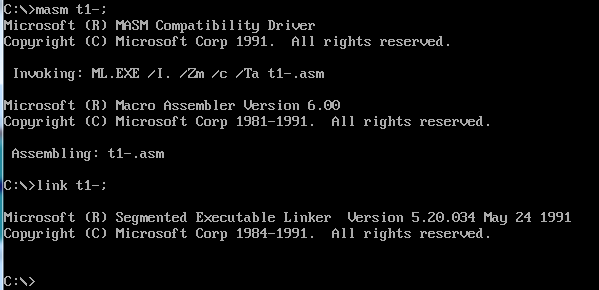


Figure 3.1.4 The compilation and connection process is normal

functions 3, 4, and 5 successively to check, and the running results are shown in Figure 3.1.5 . The first column is the product name, the second column is the total profit (percentage sign is not output) , and the third column is the rank. The total profit and ranking are consistent with the results of manual calculation of the profit of each commodity . It can be seen that the three functions work normally when compiled and run separately . It should be noted that it is best to run in the order of function 3 - function 4 - function 5 , otherwise the uninitialized data in the memory will get wrong results .

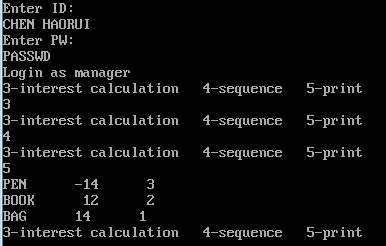


Figure 3.1.5 Results when running alone

4. Extract the relevant parts of function 3 in t1-.asm and write it into the file wan1.asm, add the paragraph name and the description of PUBLIC and EXTERN at the same time, and call the functions of function 3 -function 5 in menu.asm , to supplement the PUBLIC and EXTERN instructions added by wan1.asm . During compilation and connection , no abnormal conditions occurred, as shown in Figures 3.1.6 and 3.1.7.



Figure 3.1.6 Assembly menu and wan1

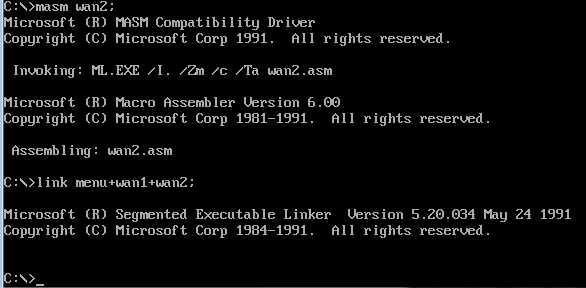


Figure 3.1.7 Assembly menu and wan1

5. Functions 1 and 2 have been tested by another classmate , so directly test function 3 - function 5. When testing functions 3 and 4 , the program is found to be stuck , as shown in Figure 3.1.8 and 3.1.9 ; when testing function 5 , it is found that the output is garbled , as shown in Figure 3.1.10 . After td single-step debugging inspection, it is found that the data segment in w an1.asm has not been loaded into DS , causing the program to freeze /crash . Use the main program in the menu to call the subroutine in wan1 to load the data segment in wan1 into DS, and modify the related PUBLIC and EXTERN statements at the same time .

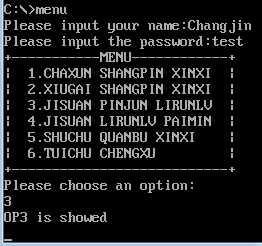
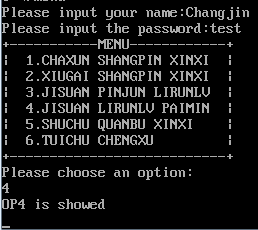
 

Figure 3.1. 8 function 3 stuck Figure 3.1.9 Function 4 stuck



Figure 3.1. 10 functions and 5 scrolling screens

6. Re-compile and connect , and execute function five again, and find the error shown in Figure 3.1.11 , and guess that the order of profit in function four is wrong. After inspection, it is found that the sorting object is the address where the profit is saved in store 1 instead of the saved profit when sorting, that is, the address should be fetched twice ; secondly , when comparing, CL should be assigned M-1 instead of M.

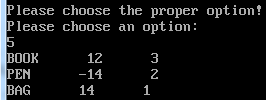


Figure 3.1. 11 Connection error reported

7. After compiling and connecting again, the output result is shown in Figure 3.1.12, which is the same as the corresponding relationship in Figure 3.1.5 , which proves that the result is correct.

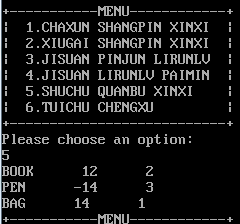


Figure 3.1.12 The result is correct

## task 2

### Design thinking and variable description

the menu interface and functions 1 , 2, and 5 in C language , and add extern and public descriptions to related variables and functions ; refer to the example to add an underscore prefix to the variables and functions used in the assembly language part and put them in each function Add the corresponding data segment load statement .

Define the data structure of the commodity in C language , and rewrite each commodity variable defined in assembly language with the structure in C language .

1. storage unit allocation

Write the product as a structure, and all the products in the two stores form two structure arrays with a length of 30 and each with 3 elements initialized :

typedef structure GOOD {

char name[10]; // name

short ipr, opr, inm, onm , //Purchase price , sales price, purchase quantity, sold quantity

int itr ; // ( where profit is stored )

} GOOD

GOOD shop1[30] = { { "PEN" ,35,56,70,25 },{ "BOOK" ,12,30,25,5 },{ "BAG" ,25,35,20,12 } };

GOOD shop2[30] = { { "BOOK" ,12,28,20,15 },{ "PEN" ,35,50,30,24 },{ "BAG" ,20,35,42,35 } };

char t ad[ 150]: Temporary array storing serial number and 32-bit address

char id[10] = "CHR" , pw[6] = "test" , cbuf[10] : store name, password and input string respectively

char jhj[] = " purchase price " , xsj[] = " sales price " , jhzs[] = " total purchase price " : as a parameter, it is convenient to call in the function

2. Register allocation

the comments for each function in the assembly language section for details .

### source program

F or vs. asm

INTR MACROT ; The parameter is part of the register name

MOV SI,DS:[ED&T&+10] ; Purchase price

IMUL SI,DS:[ED&T&+14] ; Purchase quantity

MOV AX,DS:[ED&T&+12] ;Sales price

IMUL AX,DS:[ED&T&+16] ; Sold quantity

SUB AX, SI

MOV BX, 100

PUSH EDX

IMUL BX

IDIV SI

POP EDX

ENDM

FND MACROCOUT, PIN, A, B, N; The parameter is the return label when it succeeds

MOV ESI,0

LP&A&: ; Get the current product string length in store 1 and save it in AX(AL)

CMPBYTE PTR DS:[EDX+ESI],0

JEBRK&N&

INC SI

JMPLP&A&

BRK&N&:

MOV AX,SI

MOV CH,M ; CH is responsible for the traversal of store two

MOV EDI,PIN

LP&B&:

CALL CMPST ; Compare whether the two commodity strings are the same, if not, AH is 0 (default value)

ORAH, AH

JNZCOUT

ADDDI, 24 ; DI moves to the next product in store 2, continue to compare

DECCH

JNZLP&B&

ENDM

.386

.model flat, c

.code

public ITRST,QUEUE

; Subroutine name: ITRST

;Function: Calculate the total profit margin of each commodity in the two stores

; Entry parameters: none

;exit parameters: none

;Registers used:

;CL——outer loop counter

;EDX—pointer to the item in store 1

ITRST PROC M: BYTE, shop1: DWORD, shop2: DWORD

MOV CL,M ; CL is responsible for the traversal of store one

MOV EDX, shop1

LPC: ;The previous values of all registers can be invalidated

FND BRK20,shop2,N,O,21 ; cyclically compare the string pointed to by DX and DI, if they are the same, go to BRK20

BRK20: ; Update the average profit rate of each product (CL, BX, BP are occupied)

INTRX ; Calculate the profit margin of the commodity pointed to by DX

MOV DS:[EDX+20],AX; save DX profit

INTRI ; Calculate the profit margin of the commodity pointed to by DI

ADD AX,DS:[EDX+20]; Add the two profits

SAR AX,1 ; average profit

MOV DS:[EDX+20],AX; The average profit is stored in the profit field of SHOP1

DEC CL

ORCL,CL

JZBAR5

ADD EDX,24 ; BP moves to the next item in store one

JMPLPC

BAR5:

RET

ITRST ENDP

; Subroutine name: QUEUE

;Function: sort the products in the two stores according to the total profit margin, and put them in the corresponding EA of the corresponding products in the second store

; Entry parameters: None

;exit parameters: none

;Registers used:

;BX——

;BP——pointer to the product in store 2

;CL——outer loop counter

;CH——inner loop counter

;DI - pointer to temporary array

;BP——pointer to the item in store 1

;CL——outer loop counter

;CH——inner loop counter

QUEUE PROCM: BYTE, shop1: DWORD, shop2: DWORD, TAD: DWORD

MOVE BX,TAD

MOV CL,M ; CL is responsible for the traversal of store two in LPD

XOR CH,CH

LPF: ; Initialize the temporary array number

INC CH

MOV[EBX],CH

ADD EBX,5 ; BX moves to the next structure

CMP CH, CL ; Initialize the label according to the number of valid commodities

JNELPF

MOV EDX,shop2

MOV EBX,TAD

INC EBX

LPP: ;Find the corresponding product in store 1, store the total profit address in SHOP2 and the temporary array

PUSH EBX

FNDBRK4, shop1, L, M, 17 ; cyclically compare the string pointed to by BP and BX, if they are the same, go to BRK4

BRK4:

POPEBX

ADD EDI,20

MOVDS:[EDX+18],EDI; Store the offset address of the total profit of the commodity in store 1 into store 2

MOV[EBX],EDI ; Store the offset address of the total profit of the commodity in store 1 into a temporary array

ADD EBX,5 ; DI moves to the next element in the temporary array

DEC CL

ORCL,CL

JZBRK5

ADDEDX,24

JMPLPP

BRK5: ; Sort the elements in the temporary array

MOV EBX,TAD

INC EBX ; Get the first address of the array

MOV CL,M ; Control the number of loops

XOR CH,CH

XOR ESI, ESI ; clear SI

MOV EDI,5

MOV AL,M ; Offset upper limit

XOR AH,AH

IMUL AX,5

L1: MOV EDX, [EBX+ESI] ; Use base indexing to fetch operands, L1 is the outer loop, (SI) is the loop variable, ; equivalent to i

L2: CMP EDX, [EBX+EDI] ; L2 is the inner loop, (DI) is the loop variable, equivalent to j

JGE L3

XCHG EDX, [EBX+EDI] ;[BX+SI]<[BX+DI], exchange

MOV [EBX+ESI], EDX ; send the new value of BP back to [BX+SI]

L3: ADD EDI,5 ;AH>=AL, no need to exchange, (AH) is directly compared with the last number, which is equivalent to j++

CMP EDI, EAX ; Determine whether the inner loop is over

JB L2

ADD ESI,5 ; the outer variable SI plus 5

MOV EDI,ESI ; equivalent to j=i

ADDEDI,5

LOOP L1 ; exchange between two memory data through registers

MOVEBX,TAD ;Modify the ranking value when the profit is equal

XOR ESI,ESI

LPG:

MOVEAX,[EBX+ESI+1]

MOV ECX,[EBX+ESI+6]

MOVDX,[EAX]

CMP DX,[ECX] ; Compare the relationship between the two profit margins before and after

JNEBRK11

MOV CL,[EBX+ESI]

MOV[EBX+ESI+5], CL; the serial number becomes the same

BRK11:

ADD ESI,5

MOV DL,M

XOR DH,DH

DEC DX

IMUL DX,5

CMP SI,DX

JBLPG

MOV EDX,shop2 ; Compare the address stored in the store 2 with the corresponding location in the temporary array, if they are the same, assign the corresponding serial number in the temporary array to the corresponding item in the store 2

XOR SI,SI ; SI is set to 0

XOR CH,CH ; outer loop counter

LPH:

XOR DI,DI

XOR CL,CL ; inner loop counter

LPH\_1:

MOVAX,[EBX+EDI+1]

CMP AX,DS:[EDX+ESI+18] ; Compare two addresses

JNEBRK12

MOV AL,[EBX+EDI]

XOR AH,AH

MOV DS:[EDX+ESI+20],AX ; Put the obtained ranking into the corresponding position of store 2

JMP BRK14

BRK12:

ADD DI,5 ; DI points to the next address in the temporary array

INC CL

MOV AL,M

CMP CL,AL

JNELPH\_1 ;The current product ranking has not been generated

MOVWORD PTR DS:[EDX+ESI+20],0;Safety statement, in case no address identical to the target product is found in the temporary array, set its rank to 0

BRK14:

ADD SI,24 ; SI points to the next item in store 2

INC CH

MOV AL,M

CMP CH,AL

JNE LPH ;Store 2 has not been traversed

RET

QUEUE ENDP

; Subroutine name: COMST

; Function: compare strings

; Entry parameters: none

;exit parameters: none

;Registers used:

;AX——as a transfer register

CMPST PROC ; compare string function

MOV ESI,0

LPE:

MOV BH,DS:[EDI+ESI]

CMP BH,DS:[EDX+ESI]

JNEBRK19

INCSI

CMPSI,AX

JNELPE

INC ESI

CMPBYTE PTR DS:[EDI+ESI],0; judge whether BX is also traversed

JNEBRK19

INCAH ; same string

BRK19:

RET

CMPST ENDP

end

Main . c

#include <stdio.h>

#include <string.h>

#pragma warning ( disable :4996)

typedef structure GOOD {

char name[10];

short ipr, opr, inm, onm;

int itr;

} GOOD ;

char jhj[] = " Purchase Price " , xsj[] = " Sales Price " , jhzs[] = " Total Purchase Price " , flag;

;

extern void ITRST( char , GOOD \*, GOOD \*);

extern void QUEUE( char , GOOD \*, GOOD \*, char \*);

short alt( GOOD \* p , char \* buf );

short alt( GOOD \* p , char \* buf ) {

char cbuf[4];

unsigned i;

do {

short n = 0;

flag = 1;

printf( "%s:%hd >>" , buf , p ->ipr);

scanf( "%s" , cbuf); getchar();

if (!strlen(cbuf)) {

if (!strcmp( buf , jhj)) return p- >ipr;

if (!strcmp( buf , xsj)) return p- >opr;

if (!strcmp( buf , jhzs)) return p- >inm;

}

for (i = 0; i < strlen(cbuf); i++) {

if (cbuf[i]< '0' || cbuf[i]> '9' ) {

flag = 0; break ;

}

n = n \* 10 + cbuf[i] - '0' ;

}

if (flag == 0) continue ;

return n;

} while (1);

}

void main() {

char id[10] = "CHR" , pw[6] = "test" , op, cbuf[10], \*buf2 = ":" , \*buf3 = "access failed\n" , tad[150];

GOOD \*p = NULL ;

GOOD shop1[30] = { { "PEN" ,35,56,70,25 },{ "BOOK" ,12,30,25,5 },{ "BAG" ,25,35,20,12 } };

GOOD shop2[30] = { { "BOOK" ,12,28,20,15 },{ "PEN" ,35,50,30,24 },{ "BAG" ,20,35,42,35 } };

char n = 3, auth, count, i, j;

do {

do {

printf( "input your name:" );

scanf( "%s" , cbuf); getchar();

if (!strlen(cbuf)) {

auth = 0; break ;

}

if (strcmp(id, cbuf)) continue ;

printf( "enter password:" );

scanf( "%s" , cbuf); getchar();

if (!strcmp(pw, cbuf)) {

auth = 1; break ;

}

} while (1);

count = 0;

do {

if (count) printf( " Please re-enter options \n" );

printf( " Select operation :\n1. Query product information " );

if (auth) printf( "\t2. Modify product information \t3. Calculate average profit rate \n4. Calculate profit rate ranking \t5. Output all product information " );

printf( "\t6.Exit \ n" );

op = getchar(); getchar();

} while (op < '1' || op> '6' );

if (op == '6' ) break ;

switch (op) {

case '1' :

flag = 1;

do {

printf( "input name of good:" );

scanf( "%s" , cbuf); getchar();

if (!strlen(cbuf)) {

flag = 0; break ;

}

for (i = 0;i < 3;i++)

if (!strcmp(shop1[i].name, cbuf)) break ;

if (i == 3) {

printf( " Please re-enter \n" ); continue ;

}

break ;

} while (1);

if (flag == 0) break ;

printf( "shop1\ nCommodity name : %s\ tSales price : %hd\tTotal purchase : %hd\tTotal sales :%hd\n" , shop1[i].name, shop1[i].opr , shop1[i].inm, shop1[i].onm);

for (i = 0;i < 3;i++)

if (!strcmp(shop2[i].name, cbuf)) break ;

printf( "shop2\ nCommodity name : %s\ tSales price : %hd\tTotal purchase : %hd\tTotal sales :%hd\n" , shop2[i].name, shop2[i].opr , shop2[i].inm, shop2[i].onm);

break ;

case '2' :

flag = 1;

do {

printf( " Enter the store name :" );

scanf( "%s" , cbuf); getchar();

if (cbuf[4] == '1' ) p = shop1;

if (cbuf[4] == '2' ) p = shop2;

printf( " Enter product name " );

scanf( "%s" , cbuf); getchar();

if (!strlen(cbuf)) {

flag = 0; break ;

}

for (i = 0;i < 3;i++)

if (!strcmp(shop1[i].name, cbuf)) break ;

if (i == 3) {

printf( " Please re-enter \n" ); continue ;

}

break ;

} while (1);

if (flag == 0) break ;

p->ipr = alt(p, jhj);

p->opr = alt(p, xsj);

p->inm = alt(p, jhzs);

break ;

case '3' :

ITRST(n,&(shop1[0]),&(shop2[0]));

break ;

case '4' :

QUEUE(n, &(shop1[0]), &(shop2[0]),&(tad[0]));

break ;

case '5' :

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

if (strcmp(shop1[i].name, shop2[j].name)) continue ;

printf( "%s\t%hd\t%d\n" , shop1[i].name, shop1[i].itr, shop2[j].itr);

}

}

break ;

}

} while (1);

}

### Experimental procedure

1. Prepare the experimental environment.

2. Create a new project in VS2015, and rewrite the menu and functions in C language .

Extract functions 3 and 4 in task 1 , organize them into forvs.asm , add an underline before variable names and function names , and add EXTERN and PUBLIC declarations to related variables and functions .

4. Keep trying to generate a solution and modify it according to the error until it is successfully generated.

5. Step through the program until each function returns the correct result .

### Experiment Recording and Analysis

1. Experimental environmental conditions: Intel® Core™ i5-3230M CPU @ 2.60 GHz, 2.86G memory; WINDOWS 7 ; Notepad++7.5.5.0 ; VS2015.

an empty project aasm in VS2015 , follow the C and assembly mixed programming tutorial in VS environment to set environment variables, and implement the main material list and function 1, 2, and 5 in main.c according to the C language writing method taught in the class and import them Project , compile , and find an error according to Figure 3.2.1 . After consulting with classmates, I found that VS is picky about functions , so add #pragma warning ( disable :4996) The error is canceled .

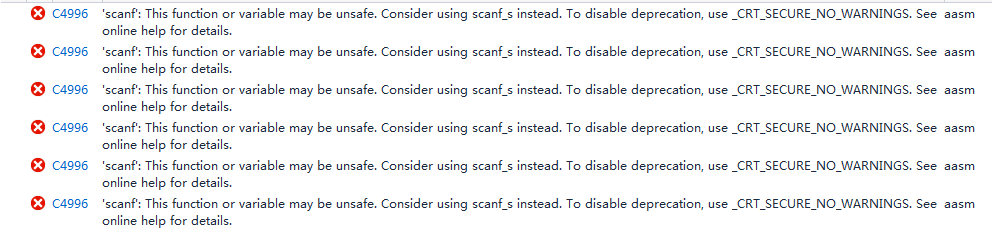


Figure 3.2.1 Compilation error

3. Compilation in the VS environment to participate in mixed programming does not need to add an underscore before the variable name and function name . Delete the original data segment in task 1 wan 1, transfer the original defined variable to the variable definition of C language , pass it into the assembly language subroutine as a parameter , and add the storage model definition according to the Win32 programming requirement .

the address in the WIN32 environment is 32 bits , all addressing related registers are changed from "XX" to "EXX" . Since some addresses need to be saved in memory when performing function 4 , the length of the temporary array tad It should be 150 (from WORD storage address to DWORD storage address). At the same time , because the parameters in the function definition of C language adopt the method of passing addresses, the variables in the assembly language store the addresses, and all LEA statements should be changed to MOV statements . The assembly source file is imported into the project aasm, and the compilation/assembly is successful , as shown in Figure 3.2.2 .

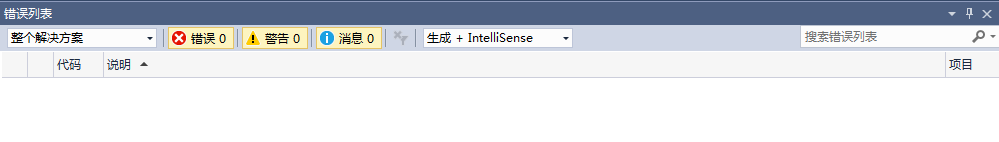


Figure 3.2. 2 Compile/Assemble Successfully

5. When running function 3, it is found that the function cannot return. After consulting my classmates, I learned how to view registers and memory under VS. Single-step execution of function 3, it is found that the value of the variable changes before and after the execution of MOV EBP, shop1, resulting in an operation failure later, as shown in Figure 3.2.3 and Figure 3.2.4 . After consulting the teacher, I found that the local variable actually uses EBP as a pointer. If the value of EBP is changed, the address and value of the corresponding variable should change , so return to forvs.asm and change all EBP to other registers. Since the requirements for registers are much looser when addressing under 32 bits , it is only necessary to consider whether the registers are occupied. After the modification, re-observe the execution and found that the correct value was stored in the correct address .

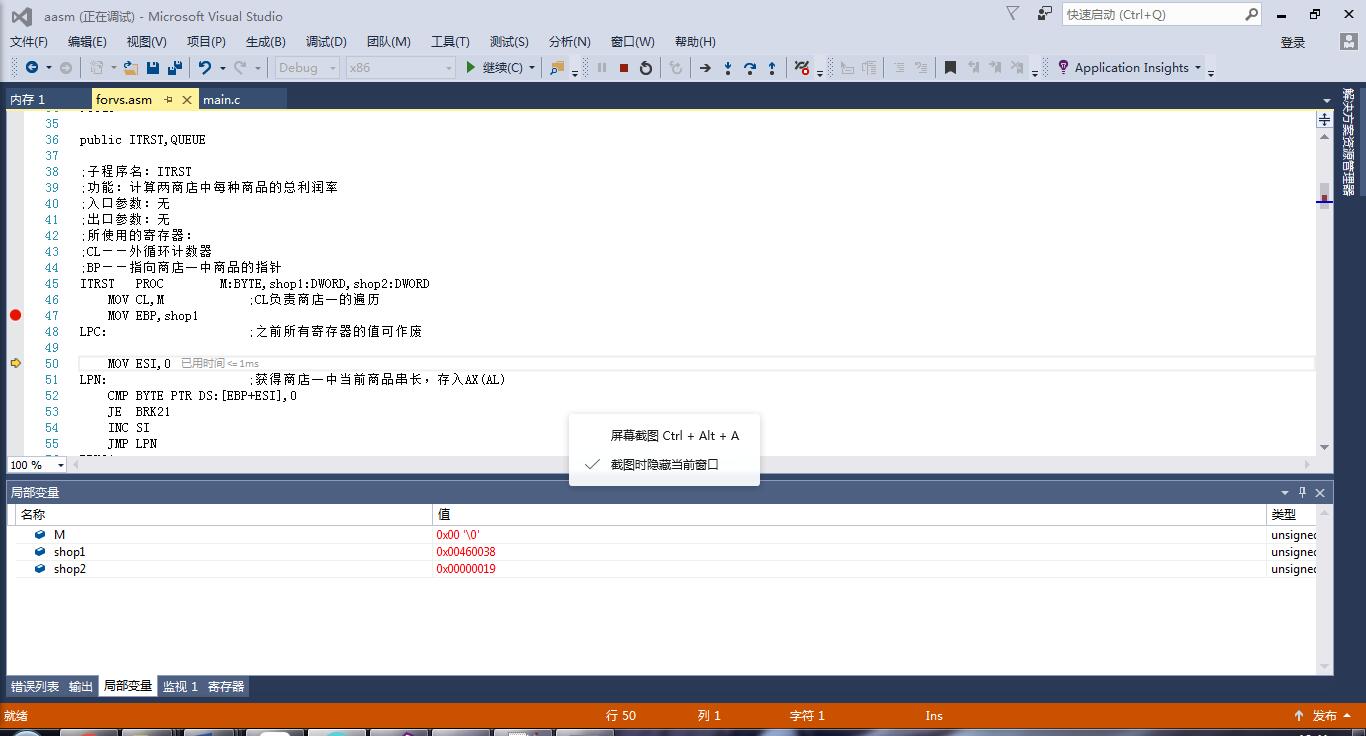
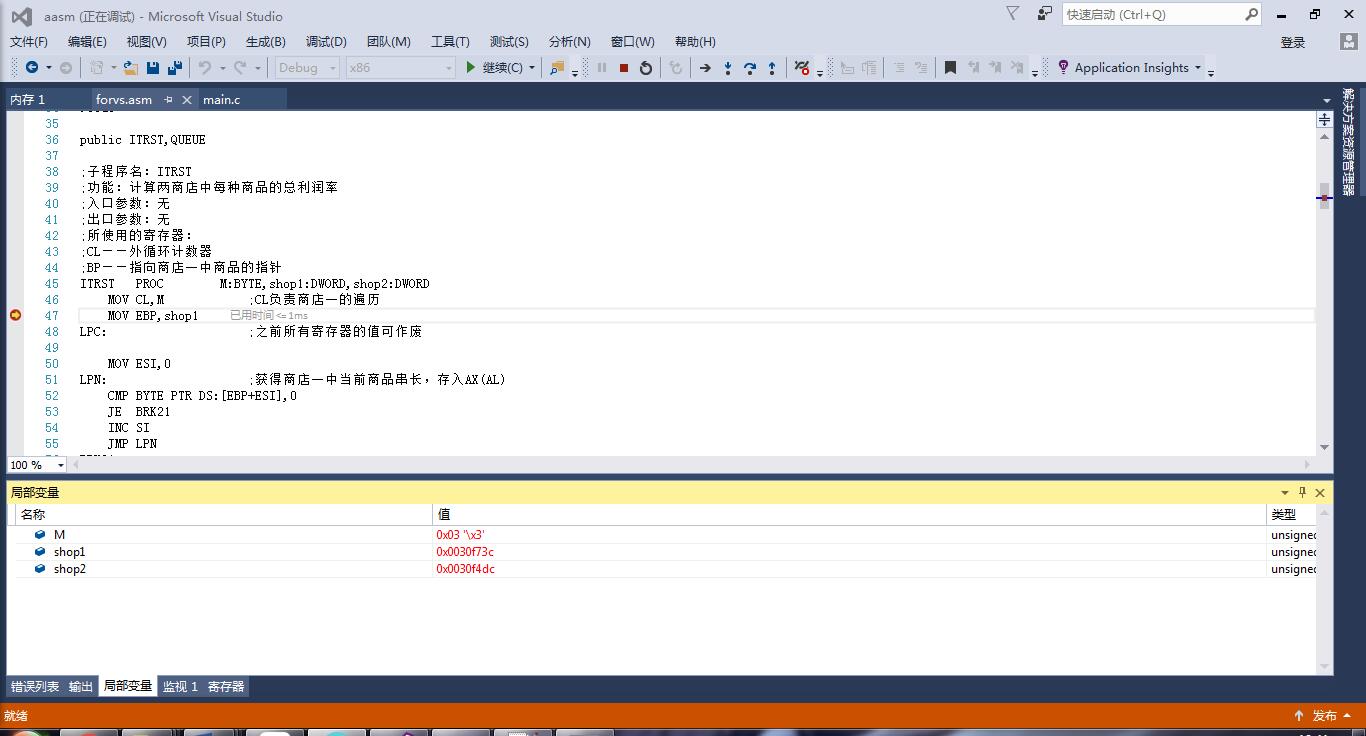


Figure 3.2. 3 & Figure 3.2. 4 Comparison of variable values before and after executing MOV EBP, shop1

6. On the premise of ensuring the normal operation of function four, enter function five, and find that the output result is wrong , as shown in Figure 3.2.5 .

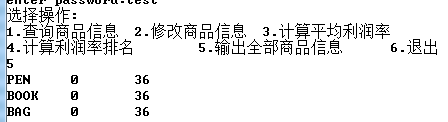


Figure 3.2. Representation of 6 G OOD types in memory

7. Observe the representation of the GOOD structure in memory, and it can be observed that there is a two-byte cc between the last int type and the previous short type , as shown in Figure 3.2.6 . After consulting my classmates, I learned that the range of int is the next four bytes, and two ccs are used for byte alignment . Modify the storage position of profit and ranking in the structure ( 18 is changed to 20 ) , make it fall within the range of int , and run again.



Figure 3.2. Representation of 6 G OOD types in memory

8. When executing function five, it is found that the results except the profit of pen are completely correct, and the binary representation of 65522 is just the complement of the correct value -14, as shown in Figure 3.2.7. Recall that when calculating the profit, it is stored in the memory in the form of two bytes ( short ) , but here the int type is still used for output , that is, the decimal number 65522 is formed together with the upper 16 bits of 0 . Change the output method of profit to short and run again .

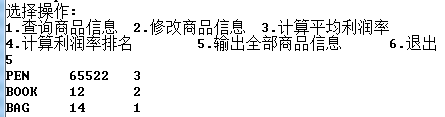


Figure 3.2. 7 PEN profit error

9. Function five outputs the correct result , as shown in Figure 3.2.8.

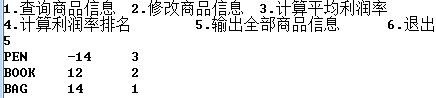


Figure 3.2. 8 output is correct

# Summary and experience

Through the experiment of Task 1, I basically understood the operation of multi-file assembly under DOS , subroutines , parameter passing and macro definition related operations and knowledge , and at the same time began to consciously use the data interaction between memory and registers through arrays . At the same time, this is the first programming assignment I have completed in collaboration with others . I have realized the importance of coordinating the interface between members in advance and agreeing on public variables . This time we are doing the above operation after numbering our own part , and found that it takes a lot of time to compare the variables of our program with the other party's program one by one .

The first big problem I faced in the experiment of Task 2 was the selection of programming environment. Before this experiment, I rarely used VS due to its large size and slow speed. This time it can be said that it is the first time I have seriously considered using VS programming . However, I was unfamiliar with the environment , and the inexplicable error that only appeared under VS when writing the C language part made me switch to B orland for a while, and the complaints of many students against B orland C made me entangled in which compiler to choose . Finally, after consulting my classmates, I finally used the pragma statement to solve the strange error report, and started to modify the assembly language part. In this task, I am familiar with the basic operation of VS2015 , which I have never used before . I saw the data alignment in the IDE for the first time , and learned that EBP is occupied due to parameter passing when assembly and C mixed programming , but I must admit that More practice is needed for the use of models and environment variables to increase proficiency .

The lesson I have learned is that I should have a basic understanding of common errors under common IDEs and prepare solutions so as not to be in a hurry when doing tasks. It is difficult to build a computer by yourself without moving the IDE .

Thinking questions:

one,

1. Use F7 to enter single-step debugging and enter the subroutine ; you can also use the cursor to select the statement of the subroutine , and F4 to complete it in one step.

2. In the near type function, ret remains unchanged ; in the far type function, ret becomes retf. When the F ar type function is called , it first pushes into the current segment CS and then pushes into IP.

3. The module call diagram, subroutine function description, and input/output description can significantly increase the readability of the code, not only for others to understand their own code, but also for themselves to know the status of each register when debugging .

7. The macro instruction will be expanded into the corresponding position of the code segment during disassembly , while the relative position of the subroutine to the main program remains unchanged.

8. You can add a border to the menu to make it more eye-catching, and you can also tabulate the various options in the menu to make them more neatly arranged.

11. Program segments that are frequently used but inconvenient to be written as subroutines can be written as macros to simplify the code .

two ,

in VS2015 do not need to be underlined ; the main and subroutines pass parameters through the stack , and EBP is needed .

It is found that the position of the data segment is different during each debugging. For example, the same variable shop1 has various values in multiple executions. If the variable is defined as shop1:DWORD in assembly , MOV EDX,shop1 becomes 00D4279E mov edx,dword ptr [shop1] after disassembly .

7. When multi-language mixed programming, there are more angles than single-language observation. For example, when debugging C language this time , you can observe the memory or disassemble the statement to check the logical correctness.

# references

[1] Xu Xiangyang, "80X86 Assembly Language Programming Hands-on Guide", "Chapter 11 Connection of Assembly Language Program and C Program ".