

**Course Lab Report**

**Course Name: Assembly Language Programming Experiment**

**Experiment name: Experiment 1 Programming Basics**

**Experiment time: 20 18 - 3 - 26, 14:00-17:30 Experiment location: No. 90 experimental platform, Room 804, South 1st Floor**

**2018-4-2 , 14:00-17:30**

**Instructor: Li Haibo**

**Professional class: school handover 201601 class**

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

**Students in the same group: None Date of report: April 3, 2018**

**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: 2018.4.3

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](#_Toc515139489)  [Experimental purpose and requirements](#_Toc515139489)  [1](#_Toc515139489)

[2](#_Toc515139490)  [Experimental content](#_Toc515139490)  [1](#_Toc515139490)

[3](#_Toc515139492)  [Experimental process](#_Toc515139492)  [3](#_Toc515139492)

[3.1](#_Toc515139493)  [Task 1](#_Toc515139493)  [3](#_Toc515139493)

[3.1.1](#_Toc515139494)  [Experimental Step](#_Toc515139494)  [3](#_Toc515139494)

[3.1.2](#_Toc515139495)  [Experimental record and analysis](#_Toc515139495)  [4](#_Toc515139495)

[3.2](#_Toc515139496)  [Task 2](#_Toc515139496)  [5](#_Toc515139496)

[3.2.1](#_Toc515139497)  [Experimental Step](#_Toc515139497)  [5](#_Toc515139497)

[3.2.2](#_Toc515139498)  [Source program](#_Toc515139498)  [6](#_Toc515139498)

[3.2.3](#_Toc515139499)  [Experimental records and analysis](#_Toc515139499)  [7](#_Toc515139499)

[3.3](#_Toc515139500)  [Task 3](#_Toc515139500)  [9](#_Toc515139500)

[3.3.1](#_Toc515139501)  [Experimental steps](#_Toc515139501)  [9](#_Toc515139501)

[3.3.2](#_Toc515139502)  [Source program](#_Toc515139502)  [10](#_Toc515139502)

[3.3.3](#_Toc515139503)  [Experimental record and analysis](#_Toc515139503)  [11](#_Toc515139503)

[3.4](#_Toc515139504)  [Task 4](#_Toc515139504)  [14](#_Toc515139504)

[3.4.1](#_Toc515139505)  [Experimental steps](#_Toc515139505)  [14](#_Toc515139505)

[3.4.2](#_Toc515139506)  [Source program](#_Toc515139506)  [14](#_Toc515139506)

[3.4.3](#_Toc515139507)  [Experimental records and analysis](#_Toc515139507)  [15](#_Toc515139507)

[3.5](#_Toc515139508)  [Task 5](#_Toc515139508)  [17](#_Toc515139508)

[3.5.1](#_Toc515139509)  [Design ideas and storage unit allocation](#_Toc515139509)  [17](#_Toc515139509)

[3.5.2](#_Toc515139510)  [Flowchart](#_Toc515139510)  [18](#_Toc515139510)

[3.5.3](#_Toc515139511)  [Source program](#_Toc515139511)  [20](#_Toc515139511)

[3.5.4](#_Toc515139512)  [Experimental steps](#_Toc515139512)  [27](#_Toc515139512)

[3.5.5](#_Toc515139513)  [Experimental records and analysis](#_Toc515139513)  [27](#_Toc515139513)

[4](#_Toc515139514)  [Summary and experience](#_Toc515139514)  [30](#_Toc515139514)

[Reference](#_Toc515139515)  [34](#_Toc515139515)

# 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 use of assembly source program editing tools, assembler, linker, and debugging tool TD;
2. Understand the representations of numbers, symbols, addressing modes, etc. in computers;
3. Understand the relationship between instruction execution and flag bit changes;
4. Familiar with commonly used DOS function calls;
5. Familiar with the structure and control methods of branch and loop programs, master the debugging methods of branch and loop programs;
6. Deepen the understanding of transfer instructions and some commonly used assembly instructions.

# Experimental content

Task 1: Question 1.14 of P31 in the "80X86 Assembly Language Programming" textbook.

Requirements: (1) Directly input commands in TD to complete the functions of sum and difference of two numbers. The summed/differenced result is placed in (AH).

(2) Please indicate in advance the contents of the flag bits SF, OF, CF, and ZF after executing the command (AH).

(3) Record the results after execution on the computer, and compare with the corresponding content in (2).

(4) In the difference operation, if A and B are regarded as signed numbers, and A>B, what are the characteristics of the flag? If A and B are regarded as unsigned numbers, and A>B, what are the characteristics of the flag?

Task 2. Question 2.3 of P45 in the "80X86 Assembly Language Programming" textbook.

the numbers of (BX), (BP), (SI), and (DI) before "MOV CX" and "INT 21H" respectively .10”

(2) Record the content of 40 bytes from the beginning of the data segment from program execution to exit, and indicate whether the program running result is consistent with the assumption.

(3) Add a program before the label LOPA to realize a new function: first display the prompt message "Press any key to begin! " , and then continue to execute the program at LOPA after pressing a key.

Task 3. Rewriting of question 2.4 of P45 in the "80X86 Assembly Language Programming" textbook.

Requirements: (1) The implemented functions remain unchanged, and the registers in the addressing mode used when accessing variables in the data segment are changed to 32-bit registers.

(2) The data access in the memory unit adopts the indexed addressing method.

(3) Record the content of the 40 bytes from the beginning of the data segment from the execution of the program to the exit, and check whether the running result of the program is consistent with the assumption.

(4) Observe and record the storage form of the machine instruction code in the memory in the TD code window, and compare it with the disassembly statement provided in TD and the source program statement written by yourself, and compare it with task 2. (Record one similar statement, focus on understanding the corresponding relationship between machine code and assembly statement, especially pay attention to the form of operand addressing mode).

(5) Observe that when the continuously stored binary strings are disassembled into assembly language statements, the disassembly starts from different byte positions. What is the result? Understand the importance of IP/EIP indicating where an instruction starts.

Task 4. Memory unit access.

Using four different memory addressing methods, store the last four digits of your student number in the storage area starting with XUEHAO in turn, requiring the student number to be stored in characters.

Requirements: Give the complete program in the report; give a screenshot of the running effect; (no need to draw a flow chart); in the program comment, clearly indicate what addressing mode is used when accessing the storage unit.

Task 5. Design and implement a program for online store commodity information query.

1. Experimental background

There is a boss who opened two online stores SHOP1 and SHOP2; each online store sells n kinds of goods, and the types of goods sold in different online stores are the same, but the quantity and sales price can be different. The information of each commodity includes: commodity name (10 bytes, fill in 0 if the name is insufficient), purchase price (word type), sales price (word type), total purchase amount (word type), sold quantity (word type) , profit margin (%) [= (sales price \* sold quantity - purchase price \* total purchase) \* 100/(purchase price \* total purchase), word type]. When the boss manages the online store information, he needs to enter his name (10 bytes, fill in the missing part with 0) and password (6 bytes, fill in the missing part with 0), after logging in, he can view all the information of the product; customers (no need to log in) You can view the information of each product in all online stores except the purchase price and profit margin.

For example:

BNAME DB 'ZHANG SAN',0 ;The name of the boss (must be the pinyin of your own name)

BPASS DB 'test',0,0; password

N EQU 30

S1 DB 'SHOP1',0 ; shop name, end with 0

GA1 DB 'PEN', 7 DUP(0) ; product name

DW 35, 56, 70, 25, ? ; Profit rate not yet calculated

GA2 DB 'BOOK', 6 DUP(0) ; product name

DW 12, 30, 25, 5, ? ; Profit rate not yet calculated

GAN DB N-2 DUP('Temp-Value', 15, 0, 20, 0, 30, 0, 2, 0, ?, ?) ; Except for 2 products that have been specifically defined, other product information is temporarily assumed for the same.

S2 DB 'SHOP2',0 ;The name of the online store, ending with 0

GB1 DB 'BOOK', 6 DUP(0) ; product name

DW 12, 28, 20, 15, ? ; Profit rate not yet calculated

GB2 DB 'PEN', 7 DUP(0) ; Trade name

DW 35, 50, 30, 24, ? ; Profit rate not yet calculated

...

2. Function 1: Prompt and enter the name and password of the login user

(1) Use the No. 9 DOS system function call to prompt the user to input their name and password.

(2) Use the No. 10 DOS system function call, and input the name and password respectively. The input name character string is placed in the storage area with in\_name as the first address, the password is placed in the storage area with in\_pwd as the first address, and the processing of function 2 is entered.

(3) If you just enter the carriage return when entering the name, send 0 to the AUTH byte variable, skip function two, and enter function three; if you only enter the character q when entering the name, the program exits.

3. Function 2: Login information authentication

( 1) Use a loop program structure to compare whether the name is correct. If incorrect, skip to (3).

(2) If it is correct, then compare whether the passwords are the same, if not, skip to (3).

(3) If the name or password is wrong, it will prompt that the login fails, and return to the position of "function one (1)", prompt and re-enter the name and password.

(4) If the name and password are correct, send 1 to the AUTH variable and enter function three.

Tip: When comparing strings, when the length of the input string is used as the number of cycles, if the cycle is terminated because the number of cycles is reduced to 0, it is necessary to judge whether the next character of the string defined in the online store is the terminator 0 , if so, it can be determined to be found (this is done to avoid misjudgment that the input string is only a subset of the string defined in the data segment).

4. Function 3: Calculate the profit rate of the specified commodity.

(1) 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 one (1).

(2) Determine the login status, if it is already logged in, go to (3). Otherwise, go to (4).

(3) First calculate the profit rate PR1 of the product in the first online store, and then find the product in the second online store, and calculate its profit rate PR2. Finally, calculate the average profit rate of the product APR=(PR1+PR2)/2. Enter function four.

(4) If it is not logged in, only the name of the product will be displayed on the next line, and then return to function 1 (1).

It is required to avoid overflow as much as possible.

Tip: Use the loop program structure and pay attention to the flexible use of addressing modes. Only the integer part of the result is kept.

5. Function 4: judge the average profit rate calculated by function 3, and display the judgment result.

(1) Grade display method: if the average profit rate is greater than or equal to 90%, it will display "A"; if it is greater than or equal to 50%, it will display "B"; if it is greater than or equal to 20%, it will display "C"; if it is greater than or equal to 0%, it will display "D" ; Less than 0%, display "F".

Prompt: use the branch program structure, and use the No. 2 DOS system function call to display the result (note that "%" should not appear in the calculation formula and instruction statement).

(2) Use the transfer command to return to "Function 1 (1)" (prompt and enter name and password).

# experiment procedure

## task 1

### Experimental procedure

1. Prepare the experimental environment.

2. Under the current cursor in the TD code window, input the instruction statement corresponding to the two 8-digit values corresponding to the expression MOV A L, ( the first operand); ADD AL, (the second operand); observe The relationship between the content displayed in the code area and the characters you input; then make sure that CS:IP points to the position of the first instruction you input, execute it three times in a single step, observe the changes in the register content, and record the result of the flag register.

(1) After ADD is expected to be executed (AH ) =8 D00 H SF=1, OF=1, CF=0, ZF=0

(2) After ADD is expected to be executed (AH ) = 7A00 H SF=0, OF=1, CF=1, ZF=0

(3) After ADD is expected to be executed (AH ) =0 800 H SF=0, OF=0, CF=1, ZF=0

Repeat the above process to calculate the remaining expressions and compare the results.

3. Input MOV AH,10H; MOV AL,-5H; SUB AH,AL; observe the characteristics of the flag;

Input MOV AH, 0FFH; MOV AL, -5H; SUB AH, AL; observe the characteristics of the flag.

### 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. Enter the command MOV AH, +0110011 B, AD D AH, +1011010B . The result after executing three instructions is shown in Figure 3.1.1. It can be seen that the calculation result in the high byte of AX ( 8D00 H) and the state of the flag (CF=0, ZF=0, SF=1, OF=1) are consistent with the prior expectation. When I first came into contact with TD , I didn't know how to exit TD and return to the DOS box main interface after executing the statement , so I had to force exit the virtual machine and re-enter. After consulting the students, I learned that I can add the end operation statement MOV AH, 4CH and INT 21H at the end of the program segment , or I can directly return to the DOS interface with INT 21H .

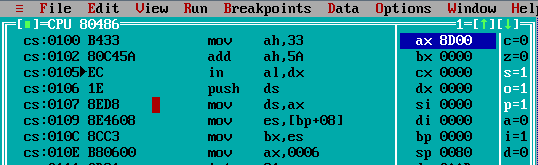


Figure 3.1.1 The state after the first calculation statement is executed

3. Enter the command MOV AH, -0101001 B, AD D AH, -1011101B . The result after executing three instructions is shown in Figure 3.1.2. The calculation result in the high byte of AX ( 7A00 H) and the state of the flag (CF=1, ZF=0, SF=0, OF=1) are consistent with the prior expectation. After consulting my classmates , I understood that when I want to return to the first statement of the execution program, I can modify the address stored in ip to be the location of the first command , and by the way, I understand the significance of ip to ensure the orderly operation of the program.

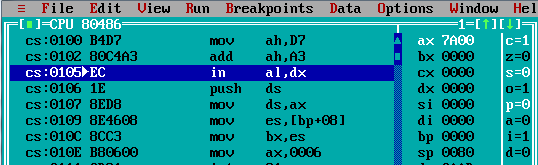


Figure 3.1.2 The state after the second calculation statement is executed

4. Enter the command MOV AH, +1100101 B, AD D AH, -1011101B . The result after executing three instructions is shown in Figure 3.1.3. The calculation result in the high byte of AX ( 0800 H) and the state of the flag (CF=1, ZF=0, SF=0, OF=0) are consistent with the prior expectation.

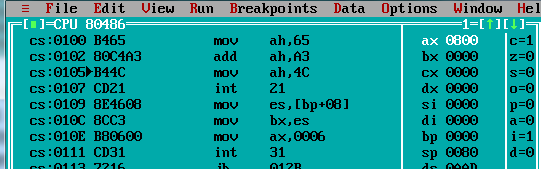


Figure 3.1.3 The state after the third formula executes the statement

5. Input MOV AH, 10H; MOV AL, -5H; SUB AH, AL , and find that the high eight bits of a x become the difference between ah and al while the low eight bits remain unchanged, sf=0, because the highest bit of ax is still after the operation 0.

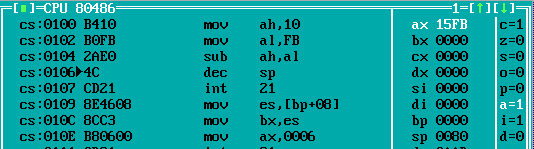


Figure 3.1.4 The state after the first calculation statement is executed

6. Input MOV AH, 0FFH; MOV AL, -5H; SUB AH, AL, and find that the upper eight bits of a x become the algebraic sum of ah and al, while the lower eight bits remain unchanged, sf=0, because ax is the highest after the operation Bit changes from 1 to 0.

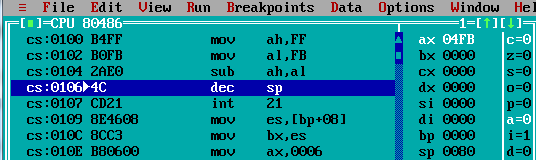


Figure 3.1.5 The state after the second calculation statement is executed

## task 2

### Experimental procedure

1. Prepare the experimental environment.

2. Use notepad ++ to input the source program, and save the file as task2.ASM . Source files were assembled using MASM 6.0. That is MASM task2 ; observe the prompt information, if there is an error, use the editing program to correct the error, save it and reassemble it until no more error is reported.

task2 .OBJ file generated by the assembly into an executable file. That is, LINK task2 ;

If the connection reports an error, modify the source program according to the error message. Then reassemble and connect until no more errors are reported and the task2.EXE file is generated.

4. Open TD.exe , open task2.exe from TD .

5 . Set breakpoints at the two target statements , step through the program to the breakpoints , and observe the target values . That is, enter task2 at the command line prompt and press Enter to observe the execution phenomenon.

6. Use the goto statement, input ds:0, and view the content of the first 40 bytes of the data segment , which is expected to be :

00 01 02 03 04 05 06 07 08 09

00 01 02 03 04 05 06 07 08 09

01 02 03 04 05 06 07 08 09 0A

04 05 06 07 08 09 0A 0B 0C 0D

7. Add storage space for the target string, and add a statement before the LOPA label :

LEA DX,STR

MOV AH,9

INT 21H

MOV AH,1

INT 21H

Compile , connect, run, and observe the experimental results.

### source program

.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)

MSTR DB 'Press any key to begin!$' ; allocate space for the string

DATA ENDS

CODE SEGMENT USE16

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

START:MOV AX,DATA

MOV DS,AX

MOV SI,OFFSET BUF1

MOV DI,OFFSET BUF2

MOV BX,OFFSET BUF3

MOV BP,OFFSET BUF4

MOV CX,10

LEA DX, MSTR ; output string

MOV AH,9

INT 21H

MOV AH,1 ;Enter any character

INT 21H

LOPA: MOV AL,[SI]

MOV [DI],AL

INCAL

MOV [BX],AL

ADD AL,3

MOV DS:[BP],AL

INC SI

INC DI

INC BP

INC BX

DEC CX

JNZ LOPA

MOV AH,4CH

INT 21H

CODE ENDS

END START

### 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. Compile and link the source program to generate corresponding task2.obj and task2.exe.

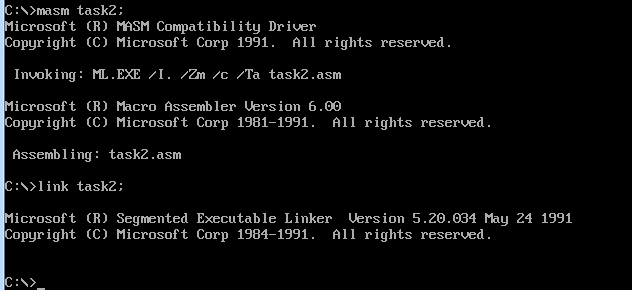


Figure 3.2.1 Assembly, connection

3. Step through the program with TD and set breakpoints at MOV CX, 10 and INT 21H .

4. F7 single-step execution to MOV CX,10 , the result is shown in Figure 3.2.2 . It can be seen that SI, DI, BX, and BP are the head pointers of four sections of space with a length of 10 , and their addresses differ by 10.

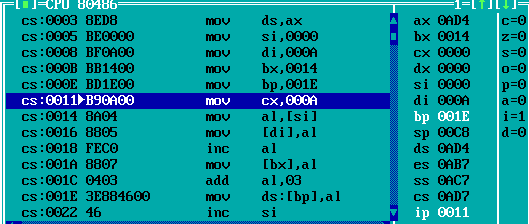


Figure 3.2.2 Execute to the result of MOV CX,10

5. Single-step execution to INT 21H , the result is shown in Figure 3.2.3 . It can be seen that 10 has been added to SI, DI, BX, and BP , and their addresses are still different by 10. After consulting with classmates , I found that I can use the F5 key to expand the window , which is convenient for screenshots.

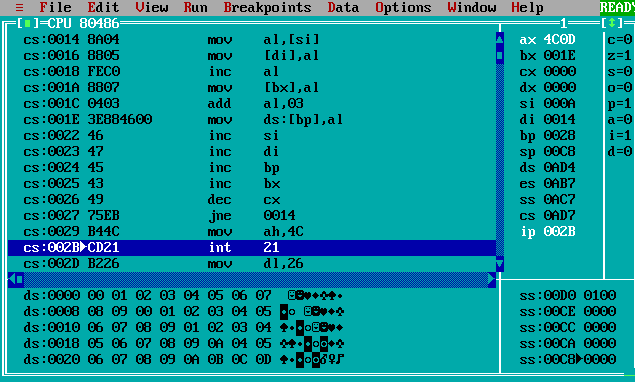


Figure 3.2.3 Execution to the result at INT 21H

6. Use goto to view the contents of the first 40 bytes of the data segment after ds:0, as shown in Figure 3.2.3 . Each element in the BUF1 section is directly copied into the BUF2 section, after adding 1 , it is copied into the BUF3 section , and after adding 3 , it is copied into the BUF4 section , so the relationship between the elements in the same position in each section of BUF1, BUF2, BUF3, and BUF4 : BUF1=BUF2 =BUF3-1=BUF4-4.

7. Modify BUF2 and define a new string in the data section : STR DB 'Press any key to begin!$' . Forgetting to add '$' after the string in the first definition results in garbled output . Add the statement before the loop :

LEA DX,STR

MOV AH,9

INT 21H

MOV AH,1

INT 21H

Save task2.asm, compile, connect, execute , and observe the output.



Figure 3.2.4 Operation status

## task 3

### Experimental procedure

1. Prepare the experimental environment.

2. On the basis of the second question, delete USE16 in the definition of the data segment , and change all register addressing related registers from 16 bits to 32 bits, and save the file name as task3-32.ASM . Source files were assembled using MASM 6.0. That is, MASM task3-32.asm ; observe the prompt information. If there is an error, use the editing program to correct the error, save it and re-compile it until no more error is reported.

task3-32 .OBJ file generated by the assembly into an executable file. Namely LINK task3-32 ; if the connection reports an error, modify the source program according to the error message. Then reassemble and connect until no more errors are reported and the task3-32.EXE file is generated.

4. Create a new task3-bz.asm , delete all register self-increment statements based on the code of task3-32.asm , and use a counter as the basis to design an offset address with an unused 32-bit register . Modify all MOV statements to use indexed addressing .

5. Repeat process 2-3 for task3 -bz.asm to generate corresponding task3-bz.obj and task3-bz.exe files .

6. Use TD.EXE to observe the execution of task3-bz . That is, TD task3-bz ; press Enter. Set INT 21H for single-step execution , and use the goto statement to observe the contents of the first 40 bytes of the data segment . Its content is assumed to be exactly the same as in task two .

7. Observe and record the storage form of the machine instruction code in the memory, and compare it with the disassembly statement provided in TD and the source program statement written by yourself, and also compare it with task 2. Screenshot its machine instructions separately.

8. Observe the result of disassembling from different byte positions when the continuously stored binary strings are disassembled into assembly language statements.

### source program

task3-32.sam

.386

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

STACK SEGMENT STACK USE16

DB200 DUP(0)

STACK ENDS

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

DATA SEGMENT

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,OFFSET BUF1

MOV EDI,OFFSET BUF2

MOV EBX,OFFSET BUF3

MOV EBP,OFFSET BUF4

MOV CX,10

LOPA: MOV EAX,[ESI]

MOV [EDI],AL

INCAL

MOV [EBX],AL

ADD AL,3

MOV DS:[EBP],AL

INC ESI

INC EBP

INC EBX

INC EDI

DEC CX

JNZ LOPA

MOV AH,4CH ; exit

INT 21H

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

CODE ENDS

END START

task3-bz.asm

.386

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

STACK SEGMENT STACK USE16

DB200 DUP(0)

STACK ENDS

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

DATA SEGMENT

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,OFFSET BUF1

MOV EDI,OFFSET BUF2

MOV EBX,OFFSET BUF3

MOV EBP,OFFSET BUF4

MOV ECX,10

LOPA: MOV EDX,10

SUB EDX,ECX ;//Modify the unified offset address of the storage space

MOV AL,[ESI] +EDX ; // design error , the correct statement should be MOV AL,[ESI+EDX]

MOV [ED I]+EDX,AL ; // design error , the correct statement should be MOV AL,[EDI+EDX]

INCAL

MOV [EBX] +EDX,AL ; // design error , the correct statement should be MOV AL,[EBX+EDX]

ADD AL,3

MOV DS:[EBP+EDX],AL ; //design error , the correct statement should be MOV MOV DS:[EBP+EDX],AL

DEC CX ; Counter decrements

JNZ LOPA

MOV AH,4CH ; exit

INT 21H

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

CODE ENDS

END START

### 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. Create a new task3-32.asm. Based on task2.asm , delete USE16 when defining the data segment , and change all register addressing related registers from 16 bits to 32 bits.

3. Compile and link task3-32 .asm , and find that the corresponding .obj and .exe files are generated .



Figure 3.3.1 Compile and link task3-32.asm



Figure 3.3.2 Corresponding .obj and .exe files

4. Compile task3-bz.asm and find an error. According to the prompt, carefully check the corresponding line, and found the cause of the error. See the description after "//" in the source program.

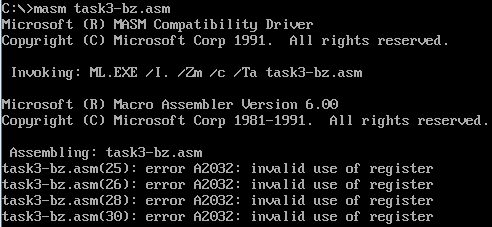


Figure 3.3.3 Assembly task3-bz.asm error

5. After the modification , compile and connect task3-bz .asm , and find that the corresponding .obj and .exe files are generated .

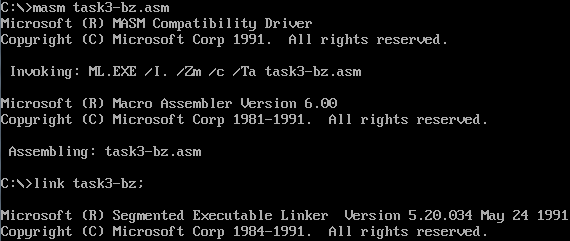
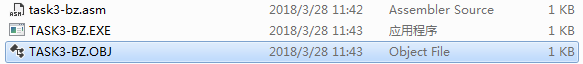
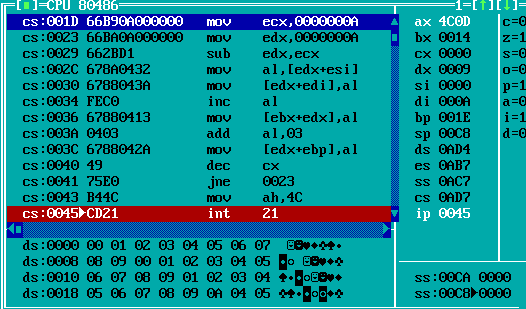


Figure 3.3.4 Compile and link task3 - bz.asm



The .obj and .exe files corresponding to Figure 3.3.5

6. Execute step by step with TD until exiting the program, use goto statement to access the content of the first 40 bytes of the data segment , the results are shown in Figure 3.3.6 and 3.3.7 . It can be found that its content is exactly the same as in task two.



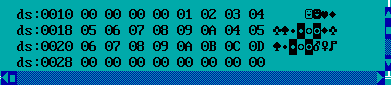


Figure 3.3.6 & Figure 3.3.7 The content of the first 40 bytes in d s

7. In Figure 3.3.6, the first column from the left is the name of the code segment cs, the second column is the address , and the third column is the machine code of the operation instruction .

8. Restart the debugging , and directly modify the address of the instruction to be called at ip to 0043 when the cycle reaches the jnx statement . After confirming, continue to operate, and find that the program should continue to execute at the return address of 0023 , but it jumps out of the loop. Observe ds again, which is consistent with the result obtained after executing a loop . It can be seen that ip prompts the position of the next scheduled execution statement, which is of great significance to ensure that the program is executed according to the flow specified by the designer .

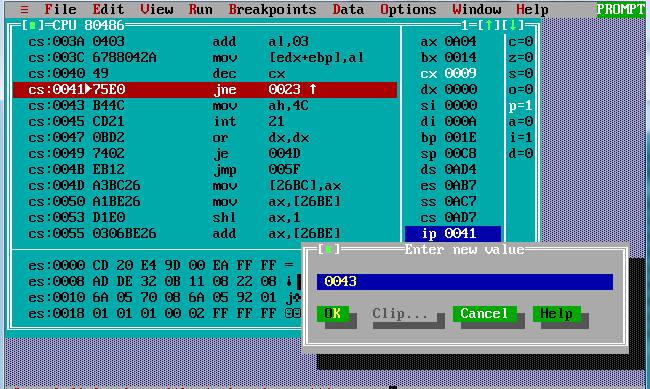
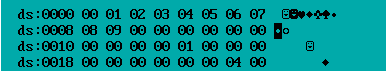


Figure 3.3.8 Modify the address of the next statement



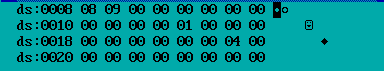


Figure 3.3.9 & 3.3.10 Changes in the content of the first 40 bytes in the data segment at the end of the program

## task 4

### Experimental procedure

1. Prepare the experimental environment.

2. Use notepad++ to input the source program, and save the file as task4.ASM .

3. Use MASM 6.0 to assemble the source files. That is MASM task4 ; observe the prompt information, if there is an error, use the editing program to correct the error, and recompile after the market until no error is reported.

task4 .OBJ file generated by the assembly into an executable file. Namely LINK task4 ;

If the connection reports an error, modify the source program according to the error message. Then reassemble and connect until no more errors are reported and the CUBE.EXE file is generated.

5. Use TD to debug the program step by step and observe the execution phenomenon. Modify for logic errors until correct execution results are obtained for various inputs.

### source program

Task4.asm

.386

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

STACK SEGMENT STACK USE16

DB 20 DUP (0)

STACK ENDS

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

DATA SEGMENT USE16

XUEHAO DB 4 DUP(0)

DATA ENDS

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

CODE SEGMENT USE16

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

START: MOV AX,DATA

MOV DS,AX

MOV BX,OFFSET XUEHAO

MOV CL,'2'

MOV [BX],CX ; BX uses register indirect addressing, CX uses register addressing

MOV CL,'6'

MOV AL,1

MOV DS:[AX],CX; // When register indirect addressing uses 16 registers, only one of BX, BP, SI, and DI can be used. Here, change it to SI together with the previous line

MOV CL,'9'

MOV [BX+2],CX ; indexed addressing

MOV CL,'6'

MOV SI,1

MOV [BX+SI+2],CX; base address plus index addressing

MOV AH,4CH ; exit

INT 21H

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

CODE ENDS

END START

### 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. Create a new task 4 .asm, allocate space for XUEHAO , and insert data into the space at the beginning of XUEHAO according to the four addressing modes.

3. During the first compilation, an error is found as shown in Figure 3.4.1.



Figure 3.4.1 Error reported in compilation

According to the prompt, carefully check the corresponding line, and found the cause of the error. See the description after "//" in the source program .

4. Compile and connect to generate corresponding .obj and .exe files .

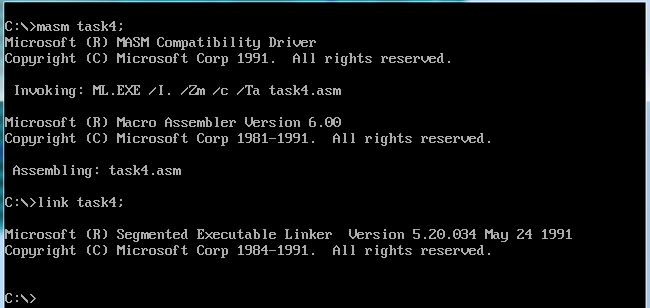


Figure 3.4.2 Assembly and connection process

5. Execute the program step by step with TD, and debug step by step until the first character is inserted .

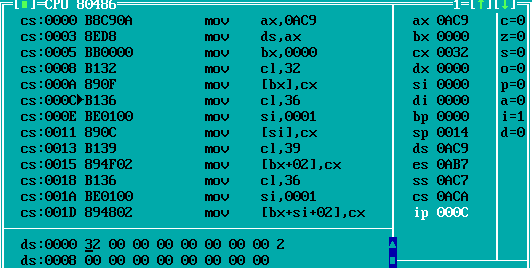


Figure 3.4.3 Put in the first character

6. Step through to insert the second character .

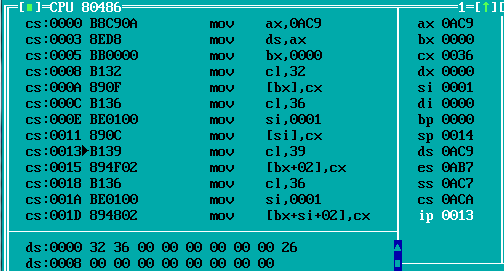


Figure 3.4.4 Put in the second character

7. Step through to the third character .

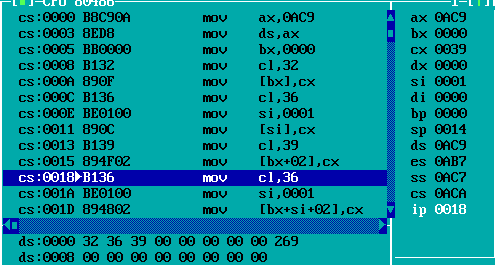


Figure 3.4.5 Put in the third character

8. Single-step debugging until the fourth character is inserted . All four characters are successfully inserted , which proves that the program runs correctly.

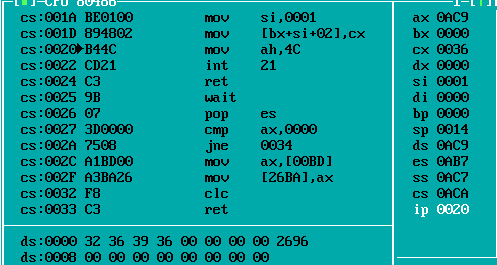


Figure 3.4.6 Put the fourth character

## task 5

### Design Thought and Storage Unit Allocation

The whole program is a loop body that contains four modules and only provides one way to exit . Module 1 ( entering the login user’s name and password ) is closely related to module 2 ( login information authentication ) , and module 3 ( calculating the specified commodity Profit rate ) contains two double loops , and module 4 ( judgment of profit rate ) is a branch structure .

1. Storage unit allocation

BNAME : The given correct username , 12 bytes long .

B PASS : The given correct password , 8 bytes long .

S1: store name , 6 bytes long .

GA1: Information of item 1 (PEN) in store 1 , including a 10-byte name character string and 5 character type numbers, with a total length of 20 bytes . The storage content type and length of all subsequent variables marking commodity types are the same as GA1.

GA2: Information of item 2 ( BOOK ) in store 1 .

GAN: Occupancy variables for the remaining 28 goods in Store 1. The content and length of the storage space occupied by each goods are the same as those of GA1 . Use the DUP statement to complete the allocation of 28 spaces .

S2: Store 2 name , 6 bytes long .

G B 1: information of item 1 ( BOOK ) in store 2 .

G B 2: information of item 2 (PEN) in store 2 .

G B N: Occupancy variables for the remaining 28 goods in store 2 , the content and length are the same as GAN.

TIP\_1 : Tip 1 : Enter username

TIP\_2 : Prompt message 2: Enter password

TIP\_3 : Tip 3: Enter the product name to be checked

TIP\_4 : Tip 4: Re- enter the product name to be checked

TIP\_5 : Tip 5: Log in as a tourist

TIP\_6 : Tip Six: Log in as the store owner

TIP\_7 : Prompt message seven: Username error

TIP\_8 : Prompt message eight: Password error

IN\_NAME: Store the input username , the first byte stores the string space length, the second byte stores the actual string length, and the next 12 bytes store the input username string . A total of 14 bytes .

IN\_PWD : Store the input password , the space allocation scheme is the same as IN\_NAME, the difference is that the password string is stored in 8 bytes . A total of 10 bytes .

IN\_GOOD : store the input product name , the space allocation scheme is the same as IN\_NAME, the difference is that 10 bytes are used to store the product string . A total of 12 bytes .

AUTH : 1 byte, storing the login system method

PR1: 1 word , the address for storing profit one. According to this address, the profit one can be retrieved by register indirect addressing .

2. Register allocation

All 16-bit data registers are used except SP and all 16-bit pointer index registers are used. The three indicator index registers are mainly used for addressing, and the data registers are used for data processing transfer and undertaking system calls .

### flow chart

Figure 3.2.1 is the program flow chart of task 2 to find the cube value of a number.

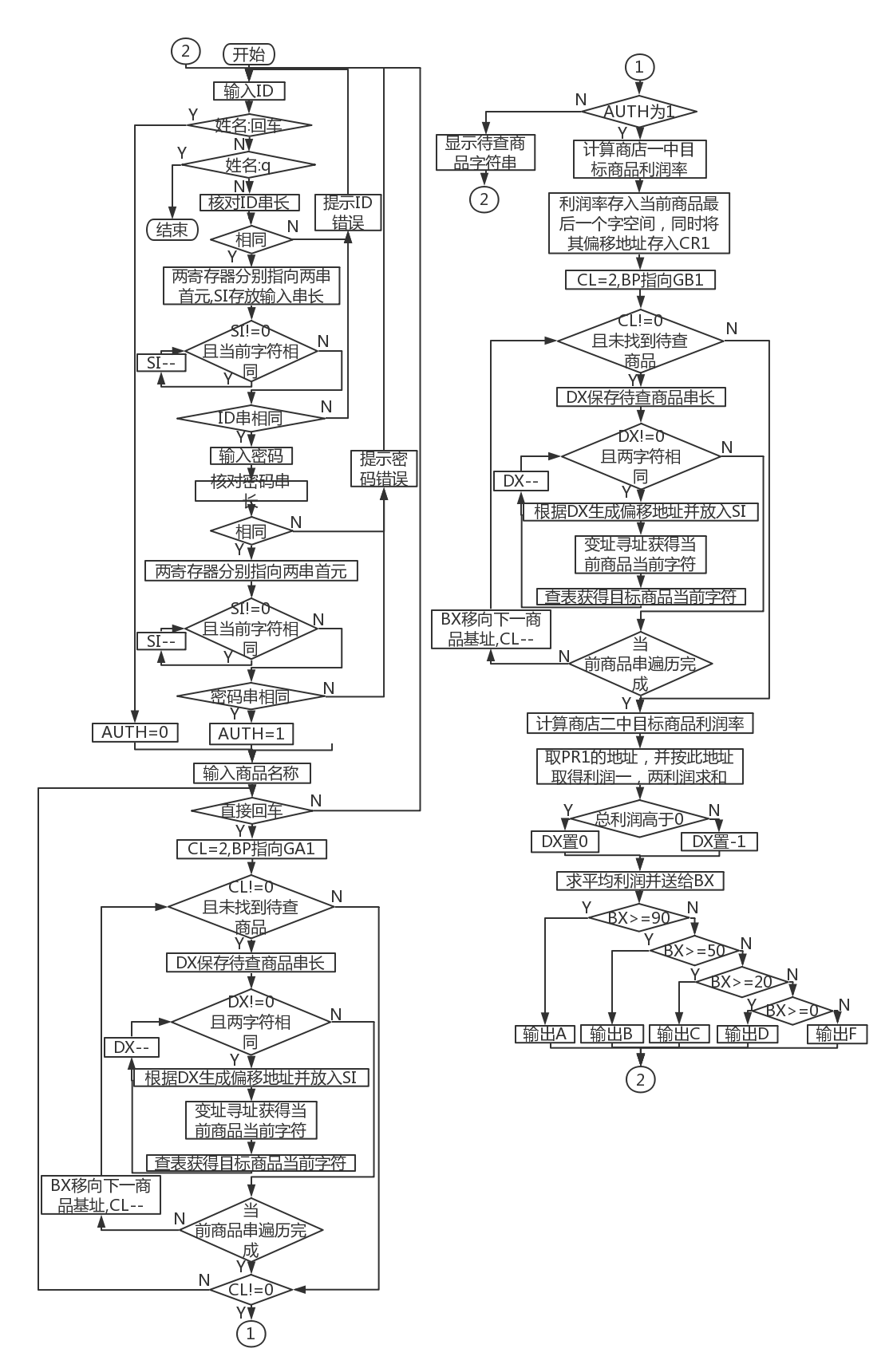


Figure 3.5.1 Overall program flow chart

### source program

.386

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

STACK SEGMENT STACK USE16

DB 20 DUP (0)

STACK ENDS

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

DATA SEGMENT

BNAME DB 'CHEN HAORUI',0 ;Boss name (must be the pinyin of your own name)

BPASS DB 'PASSWD',0,0 ;password

N EQU 30

S1 DB 'SHOP1',0 ; shop name, end with 0

GA1 DB 'PEN', 7 DUP(0) ; product name

DW 35,56,70,25,? ; Profit rate not yet calculated

GA2 DB 'BOOK', 6 DUP(0) ;Product name

DW 12,30,25,5,? ; Profit rate not yet calculated

GAN DB N-2 DUP( 'Temp-Value',15,0,20,0,30,0,2,0,?,?) ; Except for 2 product information that have been specifically defined, other product information is temporarily assumed for the same.

S2 DB 'SHOP2',0 ;The name of the online store, ending with 0

GB1 DB 'BOOK',6 DUP(0) ;Product name

DW 12,28,20,15,? ; Profit rate not yet calculated

GB2 DB 'PEN', 7 DUP(0) ; product name

DW 35,50,30,24,? ; Profit rate not yet calculated

GBN DB N-2 DUP( 'Temp-Value',15,0,20,0,30,0,2,0,?,?) ; Except for 2 products whose information has been specifically defined, other product information is temporarily assumed for the same.

TIP\_1 DB'Enter ID:$'

TIP\_2 DB'Enter PW:$'

TIP\_3 DB'Input the commodity you want to look for:$'

TIP\_4 DB'Input the commodity you want to look for again$'

TIP\_5 DB'Login as visitor$'

TIP\_6 DB'Login as manager$'

TIP\_7 DB'ID incorrect$'

TIP\_8 DB'password incorrect$'

IN\_NAME DB 12

DB?

DB12DUP(0); store the entered user name

IN\_PWD DB 8

DB?

DB8DUP(0); Store the entered password

IN\_GOOD DB 10

DB?

DB10DUP(0); store the entered commodity name

AUTH DB? ; Determine the login method

PR1 DW? ; profit address in store one

DATA ENDS

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

CODE SEGMENT USE16

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

START:

MOV AX, DATA

MOV DS,AX

STEP1:

LEA DX,TIP\_1 ;Display prompt: enter user name

MOV AH,9

INT 21H

MOV DL,0AH ;line feed

MOV AH,2

INT 21H

MOV DL,0DH ;Enter

MOV AH,2

INT 21H

LEA DX,IN\_NAME ;Enter username

MOV AH,10

INT 21H

MOV BL,0 ;The real string length is 0 when the carriage return character is input alone

LEA DI, IN\_NAME

INC DI ; find the real string length

CMP BL,[DI]

JEBAR1

JMP BRK8

BAR1:

MOV AUTH,0 ; Authentication method: customer

LEA DX, TIP\_5 ; Display prompt: log in as a guest

MOV AH,9

INT 21H

MOV DL,0AH ;line feed

MOV AH,2

INT 21H

MOV DL,0DH ;Enter

MOV AH,2

INT 21H

JMPSTEP3

BRK8:

MOVBL,'q' ; compare q

INC DI ; the first address of the string body

CMP BL,[DI]

JEEXIT

MOV DL,0AH ;line feed

MOV AH,2

INT 21H

MOV DL,0DH ;Enter

MOV AH,2

INT 21H

JMP STEP2 ; Verify the correctness of the user name

BRK7:

LEA DX, TIP\_2 ; Display prompt: enter password

MOV AH,9

INT 21H

MOV DL,0AH ;line feed

MOV AH,2

INT 21H

MOV DL,0DH ;Enter

MOV AH,2

INT 21H

LEA DX,IN\_PWD ;Enter password

MOV AH,10

INT 21H

MOV DL,0AH ;line feed

MOV AH,2

INT 21H

MOV DL,0DH ;Enter

MOV AH,2

INT 21H

JMP BRK9 ; Verify password correctness

STEP2:

MOV CL,11 ;Correct user name string length

MOV CH,[IN\_NAME+1]; input username string length

CMP CH,CL

JNE BAR2

LPA\_1:

LEA DI, IN\_NAME

LEA DI,[DI+2]

LEA DX,BNAME

MOV BX,[DX] ; // When register indirect addressing uses 16 registers, only one of BX, BP, SI, DI can be used , here it is changed to SI

CMP BX,[DI]

JNEBAR2

INC DX

INC DI

DEC CL

JNE LPA\_1

JMP BRK7

BAR2:

LEA DX, TIP\_7 ; Display prompt: wrong user name

MOV AH,9

INT 21H

MOV DL,0AH ;line feed

MOV AH,2

INT 21H

MOV DL,0DH ;Enter

MOV AH,2

INT 21H

JMPSTEP1

BRK9:

MOV CL,6 ;Correct password string length

MOV CH,[IN\_PWD+1]; input password string length

CMP CH,CL

JNE BAR3

LPA\_2:

LEA DI, IN\_PWD

LEA DI,[DI+2]

LEA DX, BPASS

MOV BX,[DX] ; // When register indirect addressing uses 16 registers, only one of BX, BP, SI, DI can be used , here it is changed to SI

CMP BX,[DI]

JNEBAR3

INC DX

INC DI

DEC CL

JNE LPA\_2

JMP BRK10

BAR3:

LEA DX, TIP\_8 ; Display prompt: wrong password

MOV AH,9

INT 21H

MOV DL,0AH ;line feed

MOV AH,2

INT 21H

MOV DL,0DH ;Enter

MOV AH,2

INT 21H

JMPSTEP1

BRK10:

MOV AUTH,1 ;Authentication method: owner

LEA DX, TIP\_6 ;Display prompt: log in as shop owner

MOV AH,9

INT 21H

MOV DL,0AH ;line feed

MOV AH,2

INT 21H

MOV DL,0DH ;Enter

MOV AH,2

INT 21H

STEP3:

LEA DX, TIP\_3 ; Display prompt: enter the product name to be checked

MOV AH,9

INT 21H

MOV DL,0AH ;line feed

MOV AH,2

INT 21H

MOV DL,0DH ;Enter

MOV AH,2

INT 21H

LEA DX,IN\_GOOD ;Input the product name to be checked

MOV AH,10

INT 21H

MOV BL,0 ;The real string length is 0 when the carriage return character is input alone

LEA DI, IN\_GOOD

INC DI ; find the real string length

CMP BL,[DI]

JESTEP1

MOV CL,2 ;Outer loop counter

LEA BP,GA1

LPB:

LEA BX, IN\_GOOD

INC BX

MOV DX,[BX]

MOV DH,0

LPBA:

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

MOV DL,0AH ;line feed

MOV AH,2

INT 21H

MOV DL,0DH ;Enter

MOV AH,2

INT 21H

JMP STEP3

BRK3:

CMP AUTH,1

JNE BRK6

MOV BX,DS:[BP+10]; Purchase price

;BP is currently at the head of the target product name string, and the corresponding offset address should be added when fetching data

IMUL BX,DS:[BP+14] ; Purchase quantity

MOV AX,DS:[BP+12];Sales price

IMUL AX,DS:[BP+16]; Sold Quantity

SUB AX,BX

MOV CX, 100

IMUL CX

IDIV BX

ADD BP,18

MOV DS:[BP],AX ;The profit in store 1 is saved in word form

LEA DI, PR1

MOV [DI],BP ; save the address of profit one

; Find the location of the item in store 2

MOV CL,2 ;Outer loop counter

LEA BP,GB1

LPC:

LEA BX, IN\_GOOD

INC BX

MOV DX,[BX]

MOV DH,0

LPCA:

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

JNEBRK4 ; Exit the loop directly when the current character is different

DEC DX

JNE LPCA ; Continue to the next cycle when CH is not 0

BRK4:

CMP DX,0 ; If CH is not 0, it means that the traversal of the 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

JNEBRK14

INC SI

MOV BX,DS:[BP+SI]

CMP BX,0

JEBRK5

BRK14:

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

DEC CL

JNELPC ; The product has not been searched

BRK5:

MOV BX,DS:[BP+10]; Purchase price

;DX is currently at the end of the target product name string, and the corresponding offset address should be added when fetching data

IMUL BX,DS:[BP+14] ; Purchase quantity

MOV AX,DS:[BP+12];Sales price

IMUL AX,DS:[BP+16]; Sold Quantity

SUB AX,BX

MOV CX, 100

IMUL CX

IDIV BX

MOV DS:[BP+18],AX; Profit in store 2 is saved in word form

MOV DI,PR1 ; Take profit one address from PR1

MOV BX,[DI]

ADD AX,BX

CMP AX,0

JLBRK11

MOV DX,0 ; AX is positive

JMP BRK12

BRK11:

MOV DX,0FFFFH ;AX is negative

BRK12:

MOV CX,2

IDIV CX ; average profit

MOV BX,AX

JMP STEP4

BRK6:

MOV DL,0AH ;line feed

MOV AH,2

INT 21H

MOV DL,0DH ;Enter

MOV AH,2

INT 21H

LEA BP, IN\_GOOD

INC BP

MOV BX,DS:[BP] ;IN\_GOOD effective string length

MOV BH,0

LPD:

LEA BP, IN\_GOOD

INC BP ; IN\_GOOD effective string length

MOV AX,DS:[BP]

MOV AH,0

SUB AX,BX

MOV SI,AX

INC BP

MOV DL,DS:[BP+SI]

MOV AH,2

INT 21H

DEC BX

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

MOV DL,0AH ;line feed

MOV AH,2

INT 21H

MOV DL,0DH ;Enter

MOV AH,2

INT 21H

JMP STEP1

STEP4: ; EBX has been occupied, store the total profit margin

MOV DL,0AH ;line feed

MOV AH,2

INT 21H

MOV DL,0DH ;Enter

MOV AH,2

INT 21H

CMP BX,90

JGE SW1

CMP BX,50

JGE SW2

CMP BX,20

JGE SW3

CMP BX,0

JGE SW4

JMP SW5

SW1:MOV DL,'A'

JMP BAR4

SW2:MOV DL,'B'

JMP BAR4

SW3:MOV DL,'C'

JMP BAR4

SW4:MOV DL,'D'

JMP BAR4

SW5:MOV DL,'F'

BAR4:

MOV AH,2

INT 21H

MOV DL,0AH ;line feed

MOV AH,2

INT 21H

MOV DL,0DH ;Enter

MOV AH,2

INT 21H

JMP STEP1

EXIT:

MOV AH,4CH ; exit

INT 21H

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

CODE ENDS

END START

### Experimental procedure

1. Prepare the experimental environment.

2. Use notepad++ to input the source program, and save the file as task5.ASM .

3. Use MASM 6.0 to assemble the source files. That is MASM task5 ; observe the prompt information, if there is an error, use the editing program to correct the error, save it and re-compile it until no more error is reported.

task5 .OBJ file generated by the assembly into an executable file. Namely LINK task5 ;

If the connection reports an error, modify the source program according to the error message. Then reassemble and connect until no more errors are reported and the CUBE.EXE file is generated.

5. Use TD to debug the program step by step and observe the execution phenomenon. Modify for logic errors until correct execution results are obtained for various inputs.

### 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. When assembling the source program, the assembler reported 3 errors, as shown in Figure 3.2.2:

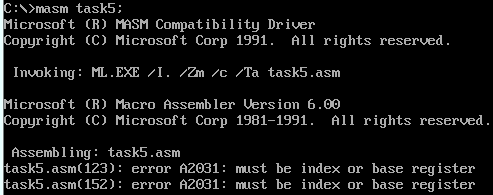


Figure 3.5.2 Compiler reports an error

According to the prompt, carefully check the corresponding line, and found the cause of the error. See the description after "//" in the source program .

3. There is no abnormality in the connection process.

4. Use TD to step through task5. When debugging function 1, I thought at first that the ASCII code of the carriage return is stored in the first byte of the string when directly entering the carriage return , and the actual string length is 1. However, I found that the actual string length is still 0 after directly entering the carriage return , as shown in Figure 3.2 .3 shown . Therefore, I changed the judgment of whether the first character is a carriage return to the judgment of whether the actual string length is 0, so as to ensure that the corresponding result is given when entering the carriage return , that is, logging in as a tourist . But comparing the first character with the carriage return here gives the same correct result.

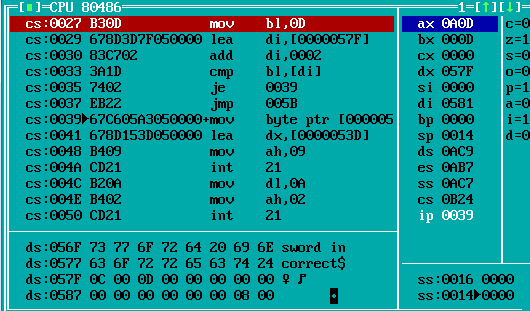


Figure 3.5.3 Directly enter the carriage return character memory in the user name string

5. Trial run the program . It was found that the format error as shown in Figure 3.5.4 appeared after directly entering a space ( login as a tourist ) , that is, the string of " input user name " was swallowed, and the string of " input product name " was directly connected to the " login as a tourist" " Afterwards . After checking the code , I found that I did not strictly wrap the line and enter after each output message, which led to the above error. So I focused on checking the calls related to the output string, and agreed to add a carriage return and line feed after it . In class today , I saw the technique provided by the teacher to output the carriage return and line feed into a string together . In programs with a large number of string output, the amount of code can be reduced to a certain extent , and I plan to apply it in future experiments.

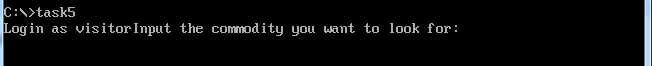


Figure 3.5.4 Directly enter the carriage return character memory in the user name string

6. When the single-step debugging function is three, input information according to the method shown in Figure 3.5.4 . Since the previous part has been single-step debugging to ensure correctness, so in this process, the corresponding product should be found in store one and the correct profit should be obtained. However, obvious errors in the data were found , as shown in Figure 3.5.5. After reading the book, I found that the default segment register is SS when BP is used as the base address in indexed addressing , that is, the data is not fetched from the corresponding offset address in DS , resulting in an error. After adding the DS: mark in front, fetching data returns to normal.

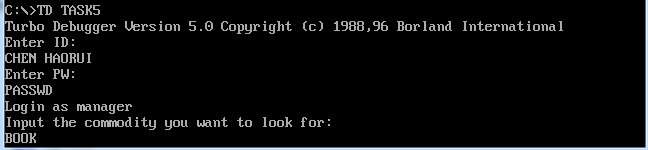


Figure 3.5.5 Input information

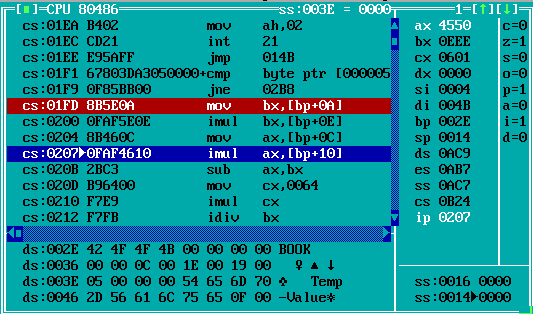


Figure 3.5.6 The data transfer to BX and AX is obviously wrong

7. Single-step debugging function 3 , find that the product is PEN. The previous codes are all guaranteed to be correct. When taking the average of the two profits , the program jumps to the position shown in Figure 3.5.6 after dividing by 2 . Considering that the sum of the two profits of PEN is a negative number, combined with the use of DX when performing signed word division on negative numbers , it is guessed that the reason for the program error is that DX is not 0FFFFH. Therefore, DX should be modified to 0 (the total profit is positive ) or 0FFFFH (the total profit is negative) according to the total profit before the division . After modification, the program operation returns to normal and correct results can be obtained.

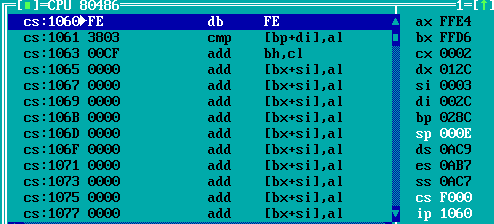


Figure 3.5.7 The program jumps to the wrong location

8. Single-step debugging function 4 , when the target product is PEN , the correct total profit should be negative, but the program jumps at the wrong position, resulting in the output result being A. After checking according to the book, it is found that JNB is an unsigned number jump instruction, but the comparison here is a signed number, so an error occurs . After changing all JAE in this part to JGE, the output will be correct , as shown in Figure 3.5.9 .

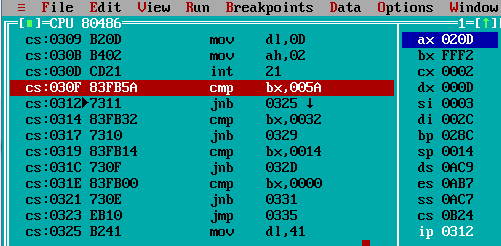


Figure 3.5.8 Jump timing error

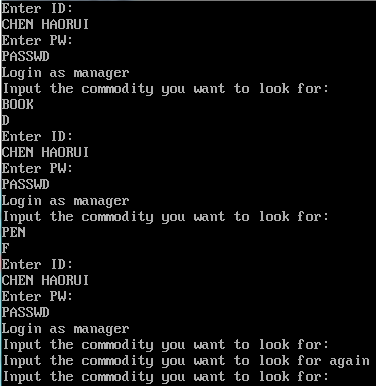


Figure 3.5.9 output is correct

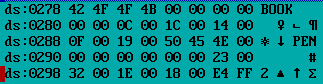


Figure 3.5.10 The total profit is saved in the words starting with ds:028A H and ds:029EH respectively

# Summary and experience

Through the experiment of task 1, I basically understand the single-step debugging of TD and the content of each part of the interface . I know that in TD, I can directly input assembly statements in the code segment, and the method of viewing the byte content of the ds segment and the input of real-time data can be used. , ten, hexadecimal and other forms , and at the same time , I am more familiar with the changing conditions of each flag . My experience is that when writing commands directly in TD , you can add an exit program statement after the code segment to conveniently exit TD and return to the DOS box main interface when the single-step debugging is completed.

Through the experiment of task 2, I am familiar with the whole process of compiling, linking and single-step debugging of the assembly source program and the simple way of running the command of the assembly source program. I am also more familiar with the use of input and output commands . Since compilation error information will be displayed according to the number of lines during assembly , Notepad is not a convenient code editing interface . You can choose a tool such as notepad++ with line number reference to edit the code to locate errors more conveniently when assembling errors.

In task 3 , I understood that the storage form of data in the data segment is only related to its defined type , and does not change with the bit length of the segment space . By observing the one-to-one correspondence between assembly code and machine instructions, I learned how it is stored. At the same time, I also realized the importance of IP 's instruction to instructions to keep the logic correct during program operation .

During task 4, I can say that I have encountered the same situation many times : " After listening to the class, I thought I would do it , but I still can't do it" and " I don't know how to be wrong , and I don't know how to be right when I am right". . Completing this task made me more familiar with the four addressing modes, the corresponding restrictions (especially under 16 bits), and some basic syntax . By debugging and observing the values of each register, I have a deeper ability to identify whether a certain register is taking an address or a value in a statement , when to take an address, and when to take a value.

Task 5 is a difficulty as the teacher said . This is the first assembly language program I 've tried to write . There are many factors that need to be considered when programming , the first point is the jump between program segments . Simple traversal is okay to say, it took me a lot of effort to rely on the exit of the judgment condition in the loop structure and the jump position of the double loop . In addition, difficulties were encountered in the use of registers . When using the double loop to search for the target product in store 2 , I have used all the 16-bit registers I have learned except SP and IP , but at this time I still need a register to save the position of profit 1 in the memory, so I started the first Consider once the problem of transferring data from register to memory . However, although the problem is solved by temporarily saving data in registers this time , frequent data interaction will increase the amount of code and also increase the time for program execution . The balance needs to be found in the future.

The order of the segments defined by the source program corresponds to the order of the segments transferred to the memory. However, the data segment is not assigned to the actual first address of the data segment at the beginning, and it needs to execute our relevant assignment statement before it becomes effective. In addition, the various addressing methods in the memory are indeed not as intuitive as the source program, but we can see that the offset value of the variable has been well calculated, and we can obtain the corresponding position information of the variable in the source program and the memory. The offset address of the variable is convenient for us to enter the address in the data display area to view the corresponding variable content.

Thinking questions :

task 1

1. Enter a new command directly in the selected line of code, and press F7 to execute it step by step .

2. Enter the offset address of the first statement in CS in ip , and press Enter.

Add H after the number , indicating that it is a hexadecimal number

4. As shown in Figure 4.1 , AH is D8, and AL is still 0.



Figure 4.1 Values of AL and AH

6. Check the value of each register in the register window ; right-click and select Registers 32-bit to switch YES/NO; directly select the target register and enter the new value.

8. When the highest bit of the calculation result is 1 , SF is 1; when the result of addition is negative/the result of subtraction is the same as the subtrahend, OF is 1; when addition is carried from the highest bit /subtraction is borrowed from the highest bit, CF is 1

Task 2 & 3

1. Open the file with the O pen option in the File menu at the top of the TD ; the largest window is the code area

2. Set a breakpoint : select the statement, F2 or Toggle in R un in the top menu ; single-step run to the breakpoint : F7 or F8; run directly to the breakpoint: R un To Cursor in R un in the top menu

3. Select the statement , F4

7. Right click and select goto, enter ss:0

8. Assembly : masm (program name); connection : link (program name);

9. When encountering an error report that has never been seen before, Baidu/ ask the teacher/ask the classmate , and form experience after seeing a lot

11. The data is saved in DS in hexadecimal form and in bytes . The first definition is placed in the low position, and the later definition is placed in the high position .

12. Expressed as machine code , the content is a combination of instructions and data stored in registers

task 5

One ,

2. Continuously output the following characters until it encounters '$' , as shown in Figure 4.2 .

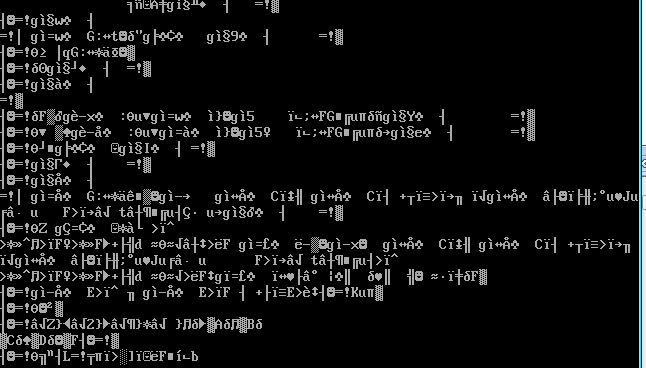


Figure 4.2 output garbled characters

3. The picture above shows the case where the base address of the "Enter ID" string in task5 is not assigned to DX . Figure 4.4 shows that the initial value of DX is 0 when debugging is started . That is, the string will not be output from the expected position, but the string will be output from the position of DX before the 9th call , until a '$' is encountered , as shown in Figure 4.3.

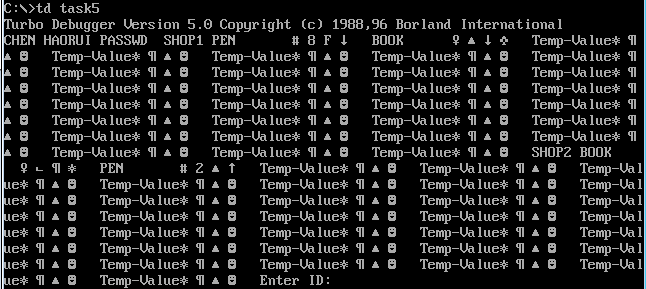


Figure 4.3 Output garbled characters

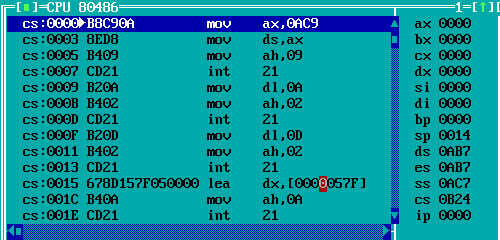


Figure 4.4 Value of D X

4. Stop inputting characters, and also stop going to the next step , and press Enter again when the length of the character string is less than the given length .

5. First compare the length of the given name string and the input string , if they are different, report an error and return directly.

converge to the target value during the loop , an infinite loop will occur.

# references

[1] Xu Xiangyang, "80X86 Assembly Language Programming Hands-on Guide"