

**Course Lab Report**

**Course Name: Assembly Language Programming Experiment**

**Experiment name: Experiment 2 Program execution time and code length optimization**

**Experiment time: 20 18 - 4 - 9, 14:00-17 : 30 Experimental location: No. 90 experimental platform, Room 804, South 1st Floor**

**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 10 , 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: 2018.4.10

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](#_Toc515177486)  [Experimental purpose and requirements](#_Toc515177486)  [1](#_Toc515177486)

[2](#_Toc515177487)  [Experimental content](#_Toc515177487)  [1](#_Toc515177487)

[3](#_Toc515177488)  [Experimental process](#_Toc515177488)  [2](#_Toc515177488)

[3.1](#_Toc515177489)  [Task 1](#_Toc515177489)  [2](#_Toc515177489)

[3.1.1](#_Toc515177490)  [Experimental Step](#_Toc515177490)  [2](#_Toc515177490)

[3.1.2](#_Toc515177491)  [Flowchart](#_Toc515177491)  [2](#_Toc515177491)

[3.1.3](#_Toc515177492)  [Source program](#_Toc515177492)  [3](#_Toc515177492)

[3.1.4](#_Toc515177493)  [Experimental records and analysis](#_Toc515177493)  [7](#_Toc515177493)

[3.2](#_Toc515177494)  [Task 2](#_Toc515177494)  [9](#_Toc515177494)

[3.2.1](#_Toc515177495)  [Experimental steps](#_Toc515177495)  [9](#_Toc515177495)

[3.2.2](#_Toc515177496)  [Flowchart](#_Toc515177496)  [9](#_Toc515177496)

[3.2.3](#_Toc515177497)  [Source program](#_Toc515177497)  [10](#_Toc515177497)

[3.2.4](#_Toc515177498)  [Experimental records and analysis](#_Toc515177498)  [11](#_Toc515177498)

[4](#_Toc515177499)  [Summary and Experience](#_Toc515177499)  [13](#_Toc515177499)

[Reference](#_Toc515177500)  [15](#_Toc515177500)

# Experimental purpose and requirements

1. Learn how programs are timed and how the operating environment affects program execution.
2. Familiar with the characteristics of assembly language instructions, and master the basic methods of code optimization.

# Experimental content

Task 1. Observe the impact of multiple loops on CPU power consumption

Application scenario introduction: Based on the background of Experiment 1 and Task 4, as long as a customer buys a product in the online store, the boss needs to regain the average profit margin of all products. Now suppose that at 0:00 on Double Eleven, there are m pieces of "Bag" products in the SHOP1 online store, and m customers place orders for this product almost at the same time. Please simulate the process of processing the above information in the background and observe the execution time.

The background processing process of the above scenario can be understood as m requests queued up on the same computer to use the program of Experiment 1 Task 4. In order to observe how much time it took between the first customer entering the purchase and the mth customer completing the purchase, we let the adjusted code of function 3 of experiment 1 task 4 execute m times repeatedly, and by calculating the m times of loop execution time difference between before and after execution to feel its impact. Functions other than function three are not included in this m cycle (but can be kept unchanged).

Description of the adjusted function three:

(1) Find the "Bag" product in SHOP1, and judge whether the sold quantity is greater than or equal to the total number of purchases. If so, return to function 1 (1), otherwise, add 1 to the sold quantity.

(2) Refresh the average profit margin of all commodities. 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.

Please modify the program of Experiment 1 and Task 4 according to the above assumptions, and set the m and n values as large as possible (for example, greater than 1000, the specific value will be changed according to the experimental effect, and gradually increase to a more obvious level, such as the time interval of seconds), to get a more pronounced effect.

Tips: (1) Add m cycles of initialization work before entering the adjusted function 3, and add m cycles of conditional judgment and transfer statements after the adjusted function 3 ends.

(2) The subroutine that displays the current time in "seconds and hundredths" is provided in the software download of the school's assembly teaching website. If you call this subroutine before m cycles, and call this subroutine after m cycles are executed, you can observe and feel the time difference before and after executing the cycle on the screen (the time difference needs to be calculated manually, of course, You can also use another timer program on the website, which can calculate the difference for you). Note that since the CPU will be scheduled by time-sharing in the virtual machine environment, the time difference will vary depending on the computer's operating environment and status.

Task 2. Optimize the assembly source program in task 1

Optimization work includes optimization of code length and optimization of execution efficiency. The focus of this optimization is the optimization of execution efficiency. Please reduce the execution time of the program by more than 10% as much as possible by optimizing the program in the loop body for m times. The more you reduce, the higher the rating!

Tips for optimization methods: First, improve performance by selecting instructions with faster execution speeds, for example, converting multiplication and division instructions into shift instructions, addition instructions, etc.; secondly, reducing one instruction in the inner loop body is equivalent to reducing m \*The execution time of n instructions needs to be carefully considered; third, try to use 32-bit register addressing, which can have more opportunities to improve the efficiency of instruction execution.

# experiment procedure

## task 1

### Experimental procedure

1. Prepare the computer experiment environment, rename the copy of experiment 1 task 4 to t1-, and modify the program on this basis .

2. Add m cycles of initialization work before entering the adjusted function 3. Create a new memory variable M as a loop counter, and store a sufficiently large value into M when entering function three from function two. Call the timing program TIMER to start timing before entering the loop .

3. After the adjusted function 3 ends, add the conditional judgment and transfer statement of m loops. Take out the current counter value from M and decrement it, if it is not 0, continue to the next cycle, otherwise call the timer TIMER again to end timing .

4. Modify the functional three- cycle body . First find the BAG in Store 1 , and judge whether the sold quantity is less than the purchased quantity , if not , the sold quantity will increase automatically , if yes , it will prompt and return to the starting point of the program directly.

5. For each product in store 1, after finding the target product in store 2, calculate the profit of the products in the two stores respectively . It should be noted that the profit offset address of the corresponding product in store 1 needs to be saved separately as a memory variable, so that when calculating the total profit, it is convenient to go from the recorded position to the profit in store 1 and then store the total profit in the corresponding position in store 1.

6. Assemble and link the file t1- .

7. Repeatedly run and TD single-step debugging until the program can exit the loop normally and give the corresponding correct profit level .

### flow chart

Figure 3.1.1 is the program flow chart of Task 1.

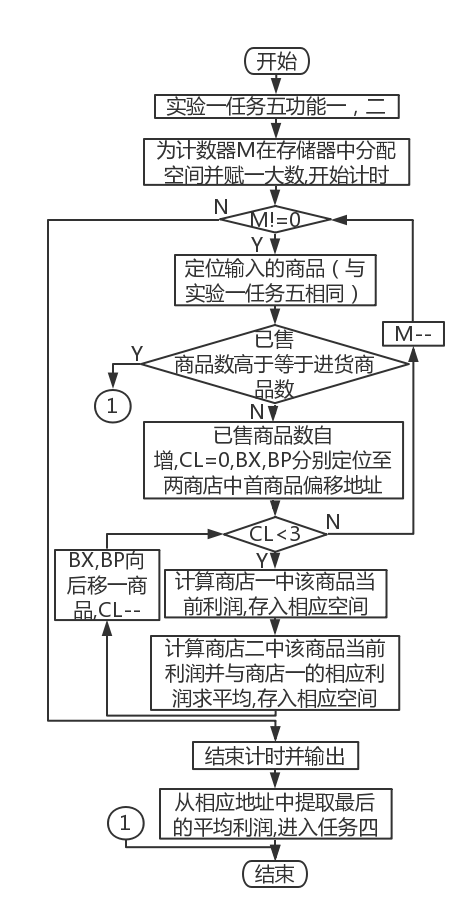


Figure 3.1.1 Program flow chart

### source program

.386

┇ ; This line was previously defined for the relevant data of Experiment 1

GA3 DB'BAG',7 DUP(0) ;Product name

DW 1,3,19000,12,? ; Profit rate has not been calculated

GAN DB N-3 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 'PEN', 7 DUP(0) ; product name

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

GB2 DB 'BOOK',6 DUP(0) ;Product name

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

GB3 DB'BAG',7 DUP(0) ;Product name

DW 20,35,42,35,? ; Profit rate not yet calculated

GBN DB N-3 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:',0AH,0DH,'$'

TIP\_2 DB'Enter PW:',0AH,0DH,'$'

TIP\_3 DB'Input the commodity you want to look for:',0AH,0DH,'$'

TIP\_4 DB'Input the commodity you want to look for again',0AH,0DH,'$'

TIP\_5 DB'Login as visitor',0AH,0DH,'$'

TIP\_6 DB'Login as manager',0AH,0DH,'$'

TIP\_7 DB'ID incorrect',0AH,0DH,'$'

TIP\_8 DB'password incorrect',0AH,0DH,'$'

TIP\_9 DB 0AH,0DH,'$' ;line feed, carriage return

TIP\_10 DB'item utterly sold out',0AH,0DH,'$'

PR1 DW? ; profit address in store one

M DW? ; Cycle times of step 3

┇ ; This line was previously defined for the relevant data of Experiment 1

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

INC DI ; find the real string length

CMP BL,[DI]

JESTEP1

MOV M,10 000 ;Initialize the loop counter each time from function 2 to function 3

MOV AX,0

CALL TIMER

BRK17:

MOV CL,3 ;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:

MOV CX,DS:[BP+16]

INC CX

MOV DS:[BP+16],CX; the sold quantity plus one

MOV CL,3 ; CL is responsible for the traversal of store one

LEA BP,GA1

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

MOV DX,0

LPC\_A: ; Obtain the current product string length in store 1 and store it in AX(AL)

CMPBYTE PTR DS:[BP+ DX],0 ; // Incorrect use of index register , it should be one of SI and DI

JEBRK18

INC SI

JMPLPC\_A

BRK18:

MOV AX,SI

MOV CH,3 ; CH is responsible for the traversal of store two

LEA BX,GB1

LPC\_B:

CALL CMPST ; Compare whether the two product strings are the same, AH should be 0

ORAH, AH

JNZBRK20

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

DECCH

JNZLPC\_B

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

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

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

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

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

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

SUB AX, SI

MOV DI, 100

IMUL DI

IDIV SI

MOV DS:[BP+18],AX;The profit in store 1 is saved in word form

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

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

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

IMUL AX,DS:[BX+16] ;Sold quantity

SUB AX, SI

IMUL DI

IDIV SI

MOV DS:[BX+18],AX;The profit in store 2 is saved in word form

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

CMP AX,0

JLBRK11

MOV DX,0 ; AX is positive

JMP BRK12

BRK11:

MOV DX,0FFFFH ;AX is negative

BRK12:

MOV DI,2

IDIV DI ; average profit

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

DEC CL

ORCL,CL

JZBRK5

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

JMPLPC

BRK5:

LEA BX,M

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

┇ ; After this line is the relevant code of Experiment 1, Task 5, Function 4

TIMER PROC

PUSH DX

PUSH CX

PUSH BX

MOV BX, AX

MOV AH, 2CH

INT 21H ;CH=hour(0-23),CL=minute(0-59),DH=second(0-59),DL=centisecond(0-100)

MOV AL, DH

MOV AH, 0

IMUL AX, AX, 1000

MOV DH, 0

IMUL DX,DX,10

ADD AX, DX

CMP BX, 0

JNZ\_T1

MOV CS:\_TS, AX

\_T0: POP BX

POP CX

POP DX

RET

\_T1: SUB AX, CS:\_TS

JNC\_T2

ADD AX, 60000

\_T2: MOV CX, 0

MOV BX, 10

\_T3: MOV DX, 0

DIV BX

PUSH DX

INC CX

CMP AX, 0

JNZ\_T3

MOV BX, 0

\_T4: POP AX

ADD AL, '0'

MOV CS:\_TMSG[BX], AL

INC BX

LOOP\_T4

PUSH DS

MOV CS:\_TMSG[BX+0], 0AH

MOV CS:\_TMSG[BX+1], 0DH

MOV CS:\_TMSG[BX+2], '$'

LEA DX, \_TS+2

PUSH CS

POP DS

MOV AH, 9

INT 21H

POP DS

JMP\_T0

\_TS DW?

DB 'Time elapsed in ms is'

\_TMSG DB 12 DUP(0)

TIMER ENDP

CMPST PROC ; compare string function

MOV SI,0

LPE:

MOV DH,DS:[BX+SI]

CMP DH,DS:[BP+SI]

JNEBRK19

INCSI

CMPSI,AX

JNELPE

INC SI

CMPBYTE PTR DS:[BX+SI],0; judge whether the traversal of BX is also completed

JNEBRK19

MOVAH,1 ; same string

RET

BRK19:

MOV AH,0 ; strings are different

RET

CMPST ENDP

┇ ; end of code segment after this line

### 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. Rename the copy of task5 in experiment 1 to t1- , modify function 3 according to the idea in the experiment steps , and set the M value to 1000 initially.

3. An error is reported during the compilation connection, as shown in Figure 3.1.2. After careful inspection , the above errors have been ruled out, and the modification method is shown in the source code.

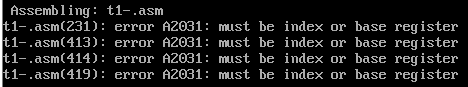


Figure 3.1.2 Assembly error

4. Re-assemble and connect , no exception , as shown in Figure 3.1.3 .

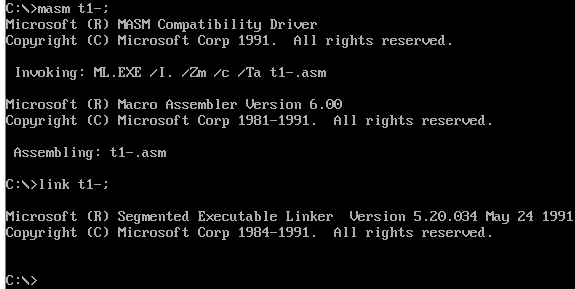


Figure 3.1.3 Assembly connection is successful

4. Execute t1 - and find that the total profit level is correct, but the execution time is too short , as shown in Figure 3.1.4, which is not convenient for task 2 observation, so consider enlarging M to 5000.

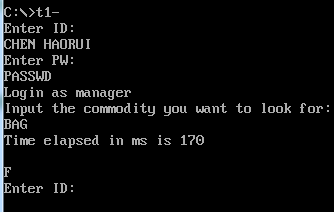


Figure 3.1.4 Total profit rating and execution time when M is 1000

5. Change M to 5000, reassemble , connect, and execute, and find that the execution time is still too small, as shown in Figure 3.1.5 . Scale M up to 10000.



Figure 3.1.5 Total profit rating and execution time when M is 5000

6. Change M to 10000, the calculation shows that the total profit rating is correct, and the execution time is available for task two .

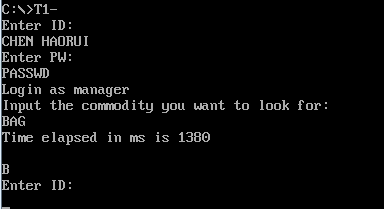


Figure 3.1.6 Total profit rating and execution time when M is 10000

## task 2

### Experimental procedure

1. Prepare the experimental environment.

2. Rename the copy of t1- to t2 - , based on which the cycle to find the item in store one comes up with the order loop in such a way as to keep the item in the memory location .

to reduce the indirect operation of immediate data through registers by creating new memory variables and exchanging space for time .

all 16-bit registers in the loop to 32-bit registers.

5. Assemble , link file t2 .

6. Repeatedly run and TD single-step debugging until the program can exit the cycle normally and give the corresponding correct profit level .

### flow chart

Figure 3.2.1 is the program flow chart of Task 2.

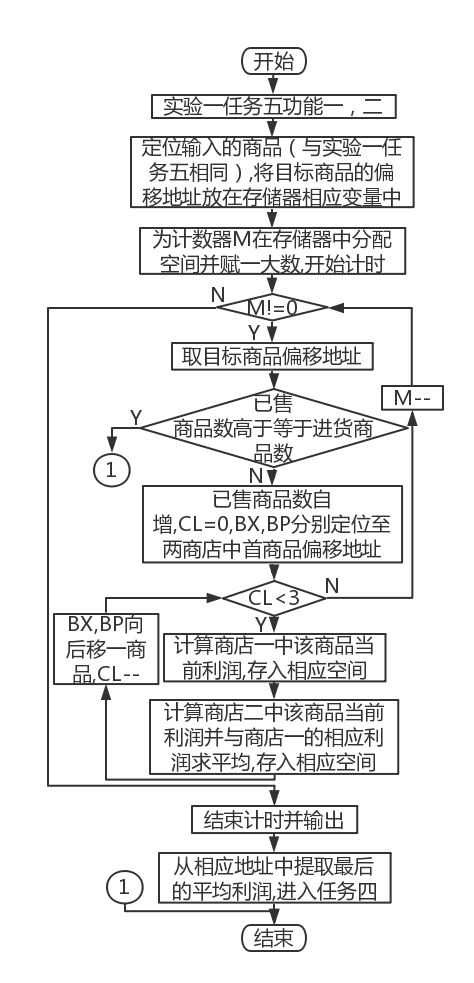


Figure 3.2.1 Program flow chart

### source program

.386

┇ ; Before this line is the relevant data segment of task 1

AUTH DB? ; Determine the login method

HKU DW100

┇ ; end of data segment after this line

┇ ; Before this line is the relevant code of task 1

SUB AX, SI

IMUL HKU

IDIV SI

┇ ; After this line is task-related code

┇ ; Before this line is the relevant code of task 1

SUB AX, SI

IMUL HKU

IDIV SI

┇ ; After this line is task-related code

┇ ; Before this line is the relevant code of task 1

SUB AX, SI

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

CWD

SAR AX,1 ; average profit

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

┇ ; After this line is task-related code

┇ ; Before this line is the relevant code of task 1

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

JNEBRK19

INCAH ; same string

BRK19:

RET

┇ ; end of code segment after this line

### 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. Take the part of searching for goods in the ordering cycle out of the cycle , store the offset address of the target product in the memory, and directly extract its address in the ordering cycle for comparison .

3. There is no abnormality in the compilation and connection process.

4. Enter BAG when running, and find that the prompt "commodity is sold out" pops up almost immediately , as shown in Figure 3.2.2 . However, the number of orders in the program is 10,000, and the sum of the number of orders sold in store 1 and 25 is far less than the number of purchases of 19,000, so a logic error occurs in the judgment . Since this place is only modified on the basis of t1 , it is necessary to check the access process of the offset address.

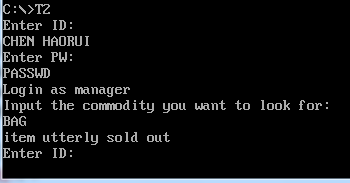
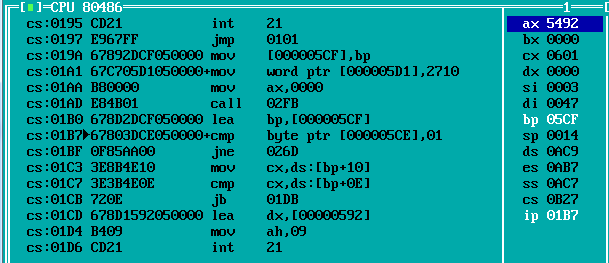


Figure 3.2.2 Program execution error

After executing LEA BP, PR1 , it is found that what is assigned to BP is not the offset address of the target commodity in PR1 , but the offset address of PR1 , as shown in Figure 3.2.3 . So I realized that the use of LEA was wrong , and it should be changed to MOV BP, PR1 here . During the single-step execution of this experiment , because it was too difficult to find the code to be checked bit by bit, I started to study the right-click menu of the CS segment window , and found that after selecting s e arch , there are obvious features near the statement to be checked. When searching for a sentence , you can directly locate the corresponding sentence, and you can locate the target sentence by searching from top to bottom , thereby improving the inspection efficiency .



There is a problem with the assignment to BP

6. Recompile , connect and execute. You can see that the program has completed the loop normally and given the correct result, and the running time has been greatly reduced, as shown in Figure 3.2.4 .

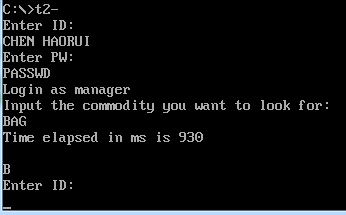


Figure 3.2.4 Significant reduction in running time

7. Create a new register variable HKU and store 100, replace MOV DX, 100 and IMUL DX with IMUL HKU , replace the branch structure that determines the value of DX based on the symbol of the previous AX for calculating the average profit directly with CWD , and replace MOV DI, 2 and IDIV DI is directly replaced by SAR AX,1 . Recompile , connect, and run. The result is shown in Figure 3.2.5 . It is found that the cycle time is reduced again.

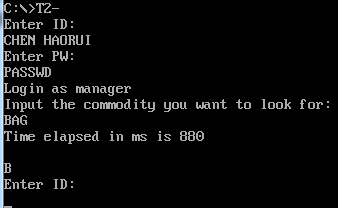


Figure 3.2.5 Runtime reduction

8. Delete irrelevant statements, such as saving the product profit of store 2 ; during the single-step debugging of the CMPST function , it can be seen that AH is always 0 in the loop , so INC AH only when the strings are judged to be equal, and delete the two assignment statements to AH . Recompile , link , run. The result is shown in Figure 3.2.6 . It is found that the cycle time is reduced again ; all the registers used for indirect addressing, base addressing and base addressing plus indexing addressing are changed from 16 bits to 32 bits , and the program is executed again. The running time has not changed.

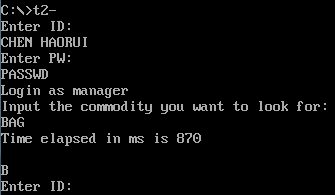


Figure 3.2.6 Runtime reduction

# Summary and experience

In this experiment, I once again realized the importance of carefully reviewing questions. I have basically completed Task 1 before the experimental class , and at the same time, I have found an idea for Task 2 , which needs to be verified. However, I learned that my understanding of Task 1 was wrong after class , so I had to reinvent the wheel , which seriously delayed the experiment . progress .

a deeper understanding of the significant increase in CPU usage as the loop size increases. When writing the loop body in C language , the execution time of the program is seriously disturbed by the environment , and the time fluctuation displayed during the execution observation is relatively severe , so it is difficult to have a deeper understanding of the impact of the loop on the execution time. The loop written in assembly language is less affected by the environment, and its execution time remains relatively stable after multiple executions of the unified loop, thus achieving the effect of controlling irrelevant variables. To a certain extent, this task simulates the response method of the website to multiple requests arriving at the same time in a dense time period. Although the data processing capacity of enterprise-level servers is much greater than that of laptops , the density of instantaneous requests they face is also much greater than that of Task assumptions . Therefore , even when programming in a high-level language, more attention should be paid to optimizing the algorithm, so as to simplify the code length of the original assembly file and further shorten the code execution time.

Task 2 made me understand that reducing time complexity is always one of the main directions of algorithm research . In this task , just changing a double cycle into a single cycle can significantly shorten the program execution time , and other things such as reducing the data transmission between registers and storage, replacing multiplication and division with shifting , and even switching to 32-bit register addressing The resulting optimizations combined may not match that either . However, small variables can also bring about significant changes after being enlarged, so we can't ignore the details of the code, and starting from the instructions can also play a role in optimizing the code.

Thinking questions :

task 1

1. First of all , the execution time before optimization should be made available for program detection by increasing the number of cycles , and at the same time, the shortening of time due to optimization is obvious enough . Secondly, the program should be optimized , and the effect of time optimization should be explained by comparing the length of execution time before and after optimization . The two time detection programs given represent two ways of thinking: one is to give the time before and after the execution of the program , and manually calculate the time difference ; the other is to count the time when the program starts to execute , and to end the timing after the execution of the program is completed, and directly give the execution time .

2. It can be seen from task 1 that as the number of cycles increases, the running time of the program increases significantly, indicating that the CPU usage increases .

3. Run t2- under DOS box and WIN7 32-bit respectively, the results are shown in Figure 4.1 and Figure 4.2 respectively . It can be seen that the program execution speed under Win7 32- bit is significantly faster than that under DOS box.



Figure 4.1 Running t2- under D OSbox

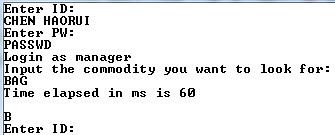


Figure 4.2 Running t2- under WIN7 32- bit

task 2

1. Reduce the number of loop nesting layers ; use more efficient statements; improve the program structure; reduce the data interaction between registers and memory ; use 32-bit register addressing.

2. Instructions in the loop body; a section of instructions that can be replaced by fewer instructions ; arithmetic operation instructions that can be replaced by shift operations ; data interaction instructions.

4. The optimization brought about by the reduction of time complexity is the most obvious, followed by the replacement of arithmetic operations with shift operations ; and other instructions.

# references

[ 1 ] Wang Yuanzhen et al . 80x86 Assembly Language Programming. Version ( 1st Edition )

[ 2 ] Xu Xiangyang . 80x86 Assembly Language Programming Hands-on Guide . Version ( 1st Edition )