

**Course Lab Report**

**Course Name: Assembly Language Programming Experiment**

**Experiment Name: Experiment 4 Interruption and Anti-Tracking**

**Experiment time: 2017-04-28 , 14 :00-17 : 30 Experiment location: No. 90 test bench , Room 804, South 1st Floor**

**20 17-05-07 , 14 : 30-18 : 00 \_ \_**

**Instructor: Li Haibo**

**Professional class: school handover 201601 class**

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

**Student in the same group: Jing Changjin Report date : May 8 , 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: 2018.5.8

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](#_Toc515198487)  [Experimental purpose and requirements](#_Toc515198487)  [2](#_Toc515198487)

[2](#_Toc515198488)  [Experimental content](#_Toc515198488)  [2](#_Toc515198488)

[3](#_Toc515198489)  [Experimental process](#_Toc515198489)  [3](#_Toc515198489)

[3.1](#_Toc515198490)  [Task 1](#_Toc515198490)  [3](#_Toc515198490)

[3.1.1](#_Toc515198491)  [Experimental Step](#_Toc515198491)  [3](#_Toc515198491)

[3.1.2](#_Toc515198492)  [Experimental records and analysis](#_Toc515198492)  [3](#_Toc515198492)

[3.2](#_Toc515198493)  [Task 2](#_Toc515198493)  [4](#_Toc515198493)

[3.2.1](#_Toc515198494)  [Design idea and storage unit allocation](#_Toc515198494)  [4](#_Toc515198494)

[3.2.2](#_Toc515198495)  [Flowchart](#_Toc515198495)  [4](#_Toc515198495)

[3.2.3](#_Toc515198496)  [Source program](#_Toc515198496)  [5](#_Toc515198496)

[3.2.4](#_Toc515198497)  [Experimental steps](#_Toc515198497)  [6](#_Toc515198497)

[3.2.5](#_Toc515198498)  [Experimental record and analysis](#_Toc515198498)  [6](#_Toc515198498)

[3.3](#_Toc515198499)  [Task 3](#_Toc515198499)  [8](#_Toc515198499)

[3.3.1](#_Toc515198500)  [Design idea and storage unit allocation](#_Toc515198500)  [8](#_Toc515198500)

[3.3.2](#_Toc515198501)  [Flowchart](#_Toc515198501)  [8](#_Toc515198501)

[3.3.3](#_Toc515198502)  [Source program](#_Toc515198502)  [9](#_Toc515198502)

[3.3.4](#_Toc515198503)  [Experimental steps](#_Toc515198503)  [10](#_Toc515198503)

[3.3.5](#_Toc515198504)  [Experimental records and analysis](#_Toc515198504)  [10](#_Toc515198504)

[3.4](#_Toc515198505)  [Task 4](#_Toc515198505)  [10](#_Toc515198505)

[3.4.1](#_Toc515198506)  [Design idea and storage unit allocation](#_Toc515198506)  [10](#_Toc515198506)

[3.4.2](#_Toc515198507)  [Flowchart](#_Toc515198507)  [10](#_Toc515198507)

[3.4.3](#_Toc515198508)  [Source program](#_Toc515198508)  [11](#_Toc515198508)

[3.4.4](#_Toc515198509)  [Experimental steps](#_Toc515198509)  [17](#_Toc515198509)

[3.4.5](#_Toc515198510)  [Experimental records and analysis](#_Toc515198510)  [18](#_Toc515198510)

[3.5](#_Toc515198511)  [Task 1](#_Toc515198511)  [18](#_Toc515198511)

[3.5.1](#_Toc515198512)  [Experimental steps](#_Toc515198512)  [18](#_Toc515198512)

[3.5.2](#_Toc515198513)  [Experimental records and analysis](#_Toc515198513)  [19](#_Toc515198513)

[4](#_Toc515198514)  [Summary and experience](#_Toc515198514)  [23](#_Toc515198514)

[Reference](#_Toc515198515)  [27](#_Toc515198515)

# Experimental purpose and requirements

The main purpose and requirements of this experiment are the following 5 points. All tasks will be carried out around these 5 points. I hope you can check whether you have met these goals and requirements afterwards.

(1) Grasp the concept of interrupt vector table;

(2) Familiar with I/O access, BIOS function call method;

(3) Master the compilation and debugging method of interrupt processing program in real mode;

(4) Familiar with tracking and anti-tracking technologies;

(5) Improve the understanding and analysis ability of computer systems.

# Experimental content

Task 1: Obtain the entry address of the interrupt handler corresponding to the interrupt type code 1H and 10H in three ways.

Requirements: first enter the virtual machine state, and then

(1) Run the debugging tool (TD.EXE) directly, and observe the information in the interrupt vector table.

(2) Write a program, use the DOS system function call method to obtain it, and observe whether the corresponding exit parameters of the function call are the same as the results seen in "(1)" (just use TD to view the exit parameters).

(3) Write a program, directly read the corresponding memory unit, and observe whether the read data is the same as the result seen in "(1)" (just use TD to watch the execution result of the program).

Task 2: Write an interrupt service program that takes over the keyboard interrupt and resides in memory. It is required that after the program returns to the DOS operating system, all lowercase letters on the keyboard will be changed to uppercase letters.

Require:

(1) Execute the program under the DOS virtual machine or DOS window, and the interrupt service program resides in the memory.

(2) When typing lowercase letters under the DOS command line, the screen will display uppercase letters, which will not change when typing uppercase letters. Execute TD and enter the command "mov AX,0 " in the code area to see if all of them are capitalized.

(3) Optional: Write an uninstall program for the interrupt service program to restore the keyboard interrupt service program to its original state (only need to restore the information of the interrupt vector table, and the previously resident program does not need to exit the memory).

Task 3: Read the information of the specified unit in CMOS and display it on the screen in hexadecimal format.

Require:

the address number of a CMOS internal unit to be read in the data segment . Then use the IN/OUT command to read the information of the specified unit in the CMOS .

(2) Display the read information on the screen in hexadecimal form. If it is time information, you can manually judge whether it is consistent with the time displayed by the operating system.

Task 4: Data encryption and anti-tracking

On the basis of the online store product information management program in Experiment 3 Task 1 , the limit on the maximum number of errors when entering the user name and password is added , that is, when the number of input errors reaches three times, it will directly enter the follow-up function in the unlogged state . The boss's password is stored in the data segment in ciphertext , and the purchase prices of various commodities are also stored in the data segment in ciphertext. The encryption method is optional.

One or more methods of timing, interrupt vector table inspection, stack inspection, indirect addressing, etc. can be used to anti-trace (two anti-trace methods are recommended, and the focus is on in-depth understanding and use of the selected anti-trace method) .

In order to simplify the workload of entry and processing, it is only necessary to define the information of three commodities.

Tip: In order to make the password, purchase price, etc. defined in the data segment of the source program become ciphertext after assembly (that is, the plaintext cannot be seen in the final delivered execution program), numerical operators can be used (see textbook P48 ) to transform the initial value of the variable. For example, if you want to turn the purchase price of 50 into ciphertext, the encryption algorithm is XOR operation with the character "W " in the boss password, then it can be written as:

DB 50 XOR ' W '

Task 5: Tracking and Data Decryption

Decrypt the encryption program of the same group of students to obtain the purchase price of each commodity.

# experiment procedure

## task 1

### Experimental procedure

1. Prepare the experimental environment.

2. In the code window of TD, goto 0:0 in the ds segment , and observe the interrupt vector table.

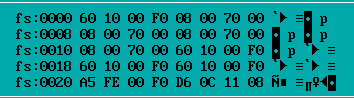
Input the instruction in TD , and observe the values of the two entry addresses by sending the content of a word starting from [1H \*4] and [10H \*4] to ax and bx respectively.

4. Enter the command in TD , call 1H and 10H through the No. 35 call , and observe the value of BX after the command is executed .

### Experiment Recording and Analysis

1. Experimental environmental conditions: P3 1GHz, 256Mmemory; DOSBox0.72 under WINDOWS XP; TD.EXE 5.0.

2. Before this experiment, I know that I can write the operation of specifying the C drive of the virtual machine into DOSBox 0.74 Options .b at . After opening the virtual machine , the target folder will be automatically set as the C drive of the virtual machine . Goto 0:0 in the window displaying the DS segment, and find that the cursor jumps to the first address of fs , indicating that the first address of fs is physical address 0, as shown in Figure 3.1.1 and Figure 3.1.2 . Here it is shown that the 1H entry address is 0080H , and the 10H entry address is 1300H .



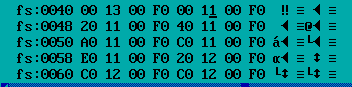


Figure 3.1.1 & Figure 3.1.2 Direct View Interrupt Vector Table

3. Directly send the entry addresses of 1H and 10H to EAX and EBX respectively , and observe the values of the two registers as shown in Figure 3.1.3 . It is found that the entry address of 1H is different from the result in the previous step , it is 0B1AH, and the entry address of 10H is still 1300H .



Figure 3.1.3 Directly observe the terminal vector table

4. Obtain the entry addresses of 1H and 10H by calling No. 35 , the results are shown in Figures 3.1.4 and 3.1,5, the entry address of 01H is 0B1AH , and the entry address of 10H is 1300H , which is the same as the previous step. After consulting the teacher, I learned that td will modify some values of the interrupt vector table , resulting in different results directly observed with TD .



Figure 3.1.4 3 Call No. 5 to obtain the entry address of 1 H



Figure 3.1. 5 3 5 call to obtain 1 0H entry address

## task 2

### Design Thought and Storage Unit Allocation

To modify the interrupt handler , first write a new interrupt handler, then store the IP and CS of the old program into OLD\_INT , and get the IP and CS of the new program into the corresponding position of the old program.

1. Storage unit allocation

OLD\_INT : Double word variable , the first two bytes store the old IP, and the last two bytes store the old CS.

2. Register allocation

AX: Responsible for interrupt program parameters .

DX: Generate the number of resident sections .

### flow chart

Figure 3.2.1 is a flowchart of task 2 memory-resident takeover and new interrupt handler .

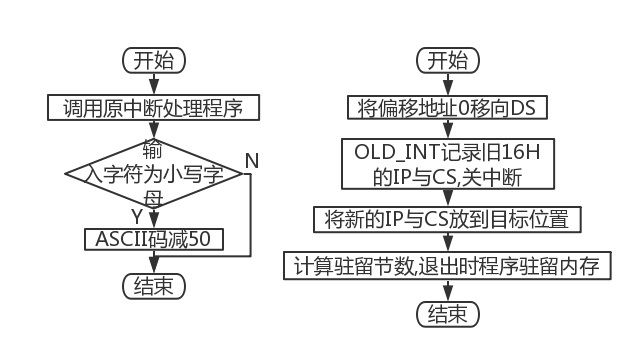


Figure 3.2.1 Program flow chart of memory-resident takeover and new interrupt handler

### source program

T1-1.asm

.386

CODE SEGMENT USE16

ASSUME CS:CODE, SS:STACK

OLD\_INT DW ?,?

NEW16H: CMP AH,0 ; determine whether to call the first type of keyboard input

JE QUIT

CMP AH,10H ; determine whether to call the second keyboard input

JE QUIT

JMP DWORD PTR OLD\_INT ; continue the original interrupt handler function

QUIT: PUSHF

CALL DWORD PTR OLD\_INT

CMP AL,61H

JBBACK

CMP AL,7AH

JA BACK

SUB AL,20H

BACK: IRET

START: XOR AX,AX

MOV DS,AX ;0→DS

MOV AX,DS:[16H\*4]

MOV CS:OLD\_INT,AX

MOV AX,DS:[16H\*4+2] ;Save the old IP,CS

MOV CS:OLD\_INT+2,AX

CLI

MOV WORD PTR DS:[16H\*4],OFFSET NEW16H

MOV DS:[16H\*4+2],CS ;When writing new CS,IP need to turn off interrupt

STI

MOV DX, OFFSET START+15 ; Calculate the number of resident blocks

SHR DX,4

ADD DX,10H

MOV AL,0

MOV AH,31H

INT 21H

CODE ENDS

STACK SEGMENT STACK

DB 200 DUP(0)

STACK ENDS

END START

T1-2. asm

.386

CODE SEGMENT USE16

ASSUME CS:CODE,SS:STACK

;Initial IP:11E0H;Initial CS:0F000H

START: XOR AX,AX

MOV FS,AX ;0->DS

MOV BX,FS:[16H\*4]

MOV DS,FS:[16H\*4+2]

CLI ; Disable interrupts when modifying the interrupt vector table

MOV AX,DS:[BX-4]

MOV WORD PTR FS:[16H\*4],AX ;Put the IP back to the corresponding position of the interrupt vector table

MOV AX,DS:[BX-2]

MOV WORD PTR FS:[16H\*4+2],AX;CS is returned to the corresponding position of the interrupt vector table

STI

;MOV DX, OFFSET START+15 ; Calculate the number of resident blocks

;SHR DX,4

;ADD DX,10H

MOV AH,31H ; When exiting (DX) section main memory unit resides

INT 21H

CODE ENDS

STACK SEGMENT STACK USE16

DB200DUP(0)

STACK ENDS

END START

### Experimental procedure

1. Refer to the memory resident program in the book , and write a new 16 H interrupt handler .

2. Compile and link the program until the compilation no longer reports errors .

3. Execute the program, and if there is an error , debug it step by step until the correct result is produced .

### Experiment Recording and Analysis

1. Prepare the experimental environment.

2. Refer to the memory resident program in the book , and write a new 16 H interrupt handler . There is no problem during compilation and running , but after running and exiting normally, it is found that the cursor is still blinking in the console but the statement cannot be entered normally . After canceling the residency, try to input the character string directly in the program, but it is still unable to input . Eliminate errors in saving the old entry address and quoting the new entry address, and learned that there is a problem in the writing of the new interrupt handler. After consulting the students, I found that I need to judge the keyboard input method first , and then start to judge the symbols in AL if I can't come up .



Figure 3.2.2 Unable to input normally

3. There is no abnormality in the connection process.

4. There is no abnormality during the assembly and connection process, as shown in Figure 3.2.3 .

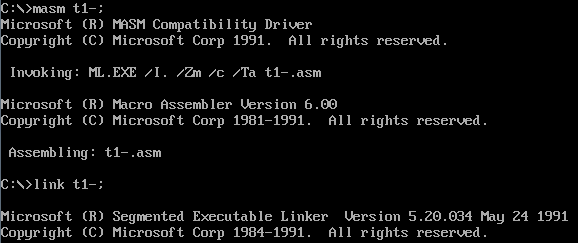


Figure 3.2.3 Assembly connection is normal

5. When CAPS LOCK is off, enter English characters and find that they are displayed as uppercase letters, which proves that the ASCII code has been successfully modified, as shown in Figure 3.2.4 .



Figure 3.2.4 Letters displayed as uppercase

6. Observing the TD shows that the IP and CS of the original interrupt handler are 11E0 H and 0F000 H respectively , as shown in Figure 3.2.5 and Figure 3.2.6 . Since the interrupt handler resides in the memory after exiting t1-1, the old IP and CS can be put back into the interrupt vector table and reside there, which is the idea of unloading the program .



Figure 3.2.5 Original IP

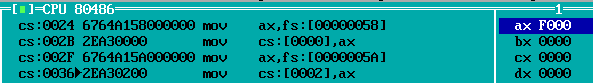


Figure 3.2.6 Original CS

7. Compile the uninstallation program according to this idea , and an error is reported during the compilation process , as shown in Figure 3.2.7 . This reminds me once again that when the highest digit of the hexadecimal system is a letter, 0 should be added in front to facilitate machine recognition.

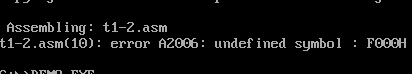


Figure 3.2.7 Data writing error

8. Compile the connection again, and everything is normal during the process . After running t1-1 and then running t1-2, it is found that the lowercase function is restored , but at the same time, CAPS LOCK fails.

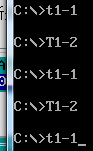


Figure 3.2.8 Only lowercase recovery

9. After thinking , I thought that the uninstaller only needs to send the initial IP and CS back to the interrupt vector table to complete the task, and there is no need to reside in memory. After trying to delete the resident memory statement, compile and connect, and find that the CAPS LOCK function returns to normal, as shown in Figure 3.2.9, but the reason for the failure of CAPS LOCKS after resident in memory needs to be further explored.



Figure 3.2.9 Uppercase Recovery

## task 3

### Design Thought and Storage Unit Allocation

To modify the interrupt handler , first write a new interrupt handler, then store the IP and CS of the old program into OLD\_INT , and get the IP and CS of the new program into the corresponding position of the old program.

1. Storage unit allocation

TIME : A byte string , storing the date converted into ASCII code .

ADS: byte string , storing the CMOS ports that need to be called , from front to back are year -month-day-hour-minute-second

2. Register allocation

SI : Responsible for the traversal of the TIME string .

B X: responsible for the traversal of the ADS string.

AL, AH: store the character converted into ASCII code and put it into the corresponding position of TIME.

### flow chart

Figure 3.3.1 is the program flow chart of task 3 displaying system time.

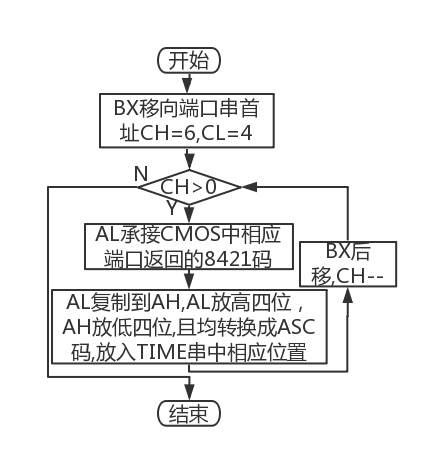


Figure 3.2.1 Program flow chart for displaying system time

### source program

STACK SEGMENT USE16 STACK

DB 20 DUP (0)

STACK ENDS

DATA SEGMENT

TIME DB 0,0,'-',0,0,'-',0,0,' ',0,0,':',0,0,':',0,0,'$' ;YY -MM-DD HH:MM:SS

ADS DB 09H,08H,07H,04H,02H,00H ;CMOS port

DATA ENDS

CODE SEGMENT

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

START:

MOV AX, DATA

MOV DS,AX

LEA SI,TIME ; SI moves to the first address of TIME

LEA BX,ADS ; BX moves to the first address of the port string

MOV CX,0604H ; CH is the number of cycles, CL is the number of right shift

LPA: MOV AL,DS:[BX]

OUT 70H,AL

IN AL,71H ; AL stores the obtained one byte

MOV AH,AL

SHR AL, CL ; AL put high

AND AH,0FH ; AH lower

ADD AX,3030H

MOV WORD PTR DS:[SI],AX

INC BX

ADD SI,3

DEC CH

JNZ LPA

MOV DX,OFFSET TIME ; output time string

MOV AH,09H

INT 21H

MOV AH,4CH

MOV AL,0H

INT 21H

CODE ENDS

END START

### Experimental procedure

1. Prepare the experimental environment.

Define the variable according to the variable design in 3.3.1 , call the ports of the year, month, day, hour, minute, and second in the CMOS through a loop , process it into a hexadecimal ASC II code , and put it into the corresponding string .

3. Repeat the assembly and connection process to solve the assembly problems that may be encountered.

4. Run the program, if there is a problem, use TD to debug step by step until the output is correct.

### 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. Write a CMOS port content extraction program, the goal is to extract the year / month / day / hour / minute / second recorded in the CMOS . First convert the extracted 8421 code to decimal.

3. When compiling and linking, everything works fine .

4. Run the program and find that the program can output the correct result in decimal .



Figure 3.3.2 The decimal output is correct

## task 4

### Design Thought and Storage Unit Allocation

1. Design thinking :

For the user name and password characters , the function encryption method is adopted , and the logical operation pro-encryption method is adopted for the length. Use indirect transfer , timing , check the stack , check the interrupt vector table, add irrelevant code, etc. to achieve anti-trace , and try to mix the function code , anti-trace code and irrelevant code together instead of separate segments .

1. Storage unit allocation :

ADR1~ADR5: All are DW variables , storing the IP and CS of the label in the code segment .

OLDINT1, OLDINT3 : Both are DW type variables , which respectively store the entry addresses of No. 1 and No. 3 interrupts for restoration before the end of the program .

### flow chart

Figure 3.4.1 is the program flow chart of task 4 anti-tracking.

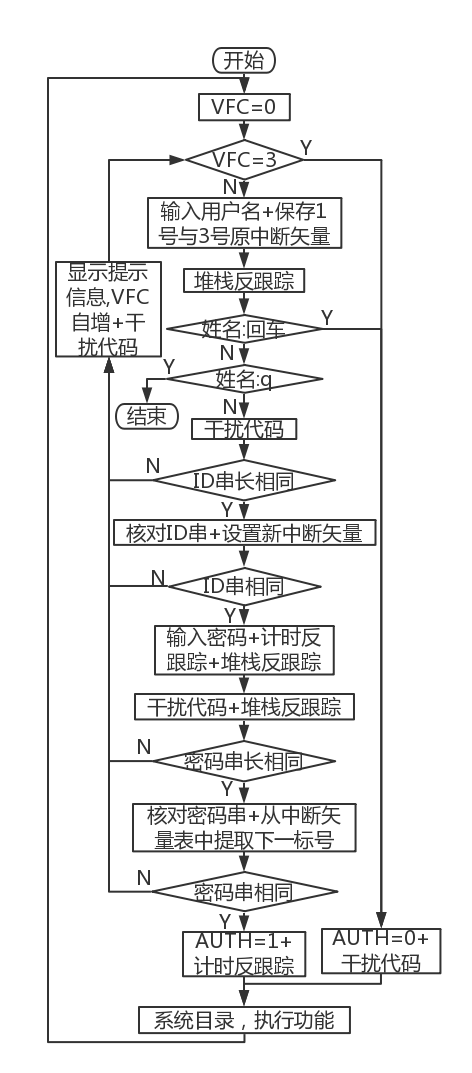


Figure 3.4.1 Anti-tracking flow chart

### source program

┇ ; Before this line is the macro definition related to Experiment 3 Task 1

INTR MACROT ; The parameter is part of the register name

IMUL SI,DS:[B&T&+16] ; Purchase quantity

ORDX,DX

NEGDI

; cli ; Timing anti-trace start

; mov ah,2ch

; int 21h

; push dx ; save fetched seconds and hundredths

IMUL DI,DS:[B&T&+14]

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

SARDI,1

MOV DI,AX

LEA AX,GA2

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

SHR SI,2

NEGDI

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

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

SARDI,1

MOV DI,AX

IMUL DI,DS:[B&T&+10]

ADD SI,9

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

ORDX,SI

NEGDI

ANDDX,DI

NEGDI

IMUL DI,DS:[B&T&+14]

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

SARDI,1

MOV DI,AX

IMUL DI,DS:[B&T&+14]

; mov ah,2ch ; Get second and hundredth second

; int 21h

; \_

; cmp dx,[esp] ; Whether the timing is the same

; jne STEP2

; pop dx

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

SARDI,1

MOV DI,AX

MOVCH,10

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

SUB DI, SI

SHLDX,3

MOV AX,DI

IMUL HKU

IDIV SI

ENDM

┇ ; This line is followed by experiment three task one related macro definition

┇ ; Before this line is the relevant data segment of Experiment 3 Task 1

ADR3 DWEXIT

ADR1 DWBRK10

ADR4 DWLPA\_2

ADR2 DWLPA\_1

ADR5 DWBAR1

OLDINT1 DW 0,0 ; The original interrupt vector of No. 1 interrupt (for anti-tracking of interrupt vector table)

OLDINT3 DW 0,0 ; the original interrupt vector of No. 3 interrupt

┇ ; After this line is the end of experiment 3 task 1 data segment

┇ ; After this line is the beginning of the experiment 3 task 1 code segment

MOV DS,AX

;Take over interrupt for debugging, interrupt vector table anti-trace (indentation 4, lowercase)

;interfering program (indent 16)

;Stack inspection anti-trace (indent 8, lowercase)

;Timing anti-trace (indent 12, lower case)

STEP1:

xor bx,bx

mov fs,bx

mov bl,IN\_NAME

cli

IO TIP\_1,9 ;Display prompt: enter username

shr bl,2

IO IN\_NAME,10 ; input user name

inc bl ;bl=4

push ADR2 ; the address of PASS2 is pushed onto the stack

mov ax, fs:[bx] ; save the original No. 1 and No. 3 interrupt vectors

mov OLDINT1,ax

mov ax,fs:[bx+2]

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

LEA DI,[IN\_NAME+1]; Find the real string length

pop ax

CMP BL,[DI]

JEBAR1

mov bx,[esp-2] ; Get the word on the top of the stack (the address of PASS2), the stack address is small and big, so it is reduced

JMP BRK8

BAR1:

MOV AUTH,0 ; Authentication method: customer

IO TIP\_5,9 ; Display prompt: log in as a guest

JMPSTEP3

BRK8:

INC DI ; the first address of the string body

CMP BYTE PTR [DI],'q' ; compare q

JEEXIT

IO TIP\_9,9 ; Line feed, carriage return

JMP STEP2 ; Verify the correctness of the user name

BRK7:

cli ;timed anti-trace start

mov ah,2ch

int 21h

push dx ; save the acquired seconds and hundredths

IO TIP\_2,9 ;Display prompt: enter password

IO IN\_PWD,10 ;Enter password

IO TIP\_9,9 ; Line feed, carriage return

mov ah,2ch ; Get the second second and hundredth second

int 21h

sti

cli ; stack inspection antitrace

push ADR4 ; the address of PASS2 is pushed onto the stack

JMP BRK9 ; Verify password correctness

STEP2:

MOV DH,ES:[BP+SI]

AND DH,ES:[BP+DI]

sti

MOV CL,IN\_NAME+1; input username string length

INCDX

LEASI,[ECX]

mov OLDINT1+2,ax

MOV CH,CL ; input username string length

mov ax,fs:[3\*4]

XORBYTE PTR ES:[BX+SI],0FAH

mov OLDINT3,ax

OR CL,'x'

mov ax,fs:[3\*4+2]

SHR SI,2

mov OLDINT3+2,ax

CMP CL,BNAME

JNE BAR2

MOVDH,1

LEA DI,[IN\_NAME+2]

LEA SI,[BNAME+1]

jmp bx ; will not transfer to LPA\_1 if tracked

LPA\_1:

XOR BH,BH

MOV BL,[DI]

SUB BL,42

cli ; set a new interrupt vector (written in the LPA\_1 loop body)

mov ax,OFFSET NEWINT

SHL BX,1

mov fs:[1\*4],ax

CMP BL,[SI]

mov fs:[1\*4+2],cs

JNEBAR2

INC SI

mov fs:[3\*4],ax

INC DI

DEC CH

mov fs:[3\*4+2],cs

sti

JNE LPA\_1

JMP BRK7

BAR2:

IO TIP\_7,9 ;Display prompt: user name error

BRK25:

MOV CH, VFC

PUSH DX

INC DX

cli ; stack inspection antitrace

push ADR5 ; the address of ADR5 is pushed onto the stack

CMP AX, 0

INC CH

MOV VFC,CH

MOV BX, 0

POP AX

ADD AL, '0'

pop ax

mov ax,[esp-2] ;Get the word on the top of the stack (address of PASS2) to

sti

MOV ES: [BX], AL

INC BX

CMP CH,3

JNE STEP1

MOV BYTE PTR ES:[BX+0], 0AH

MOV BYTE PTR ES:[BX+1], 0DH

MOV BYTE PTR ES:[BX+2], '$'

POP BX

jmpax

LEA DX,[BX+2]

BRK9:

MOVDL,DH

MOVCH,0

pop bx

MOV CL,IN\_PWD+1 ; Compare the length of the input string with the length of the password

MOV CH,CL

SHR DI,6

MOV DH,0

mov ax,[esp-2] ;Get the word on the top of the stack (address of PASS2) to

sti

XOR CL,'t'

CMP CL, BPASS

JNE BAR3

LEA DI,[IN\_PWD+2]

ADD SI,DX

LEA SI,[BPASS+1]

jmp ax

LPA\_2:

XOR BH,BH

MOV BL,[DI]

SUB BL,75

SHL BX,1

CMP BL,[SI]

JNEBAR3

mov dx,fs:[1\*4] ; Check whether the interrupt vector table is prevented from being modified or restored by the debugging tool

INC SI

INC DI

inc dx

DEC CH

JNE LPA\_2

;JMP BRK10

jmp dx

BAR3:

IO TIP\_8,9 ; Display prompt: wrong password

JMP BRK25

BRK10:

MOV AUTH,1 ;Authentication method: owner

IO TIP\_6,9 ;Display prompt: log in as shop owner

STEP3:

cmp dx,[esp] ; Whether the timing is the same

MOV BX,OFFSET GAN

jz STEP3 ; If the timing is the same, go to STEP3 through the anti-tracking of this timing

MOV CL,'2'

MOV ES:[BX],CX

MOV CL,'6'

MOV SI,1

CMP AUTH,1

JNE BRK6

mov bx,offset ADR3 ; if the timing is different, shift the transfer address away from ADR3(EXIT)

MOV ES:[SI],CX

MOV CL,'9'

MOV ES:[BX+2],CX

MOV CL,'6'

MOV SI,1

MOV ES:[BX+SI+2],CX

pop dx

IO TIP\_13,9 ; Shopkeeper menu

MOV AH,1 ; input option

INT 21H

IO TIP\_9,9

CMP AL, '3'

JESTEP3\_3

┇ ; After this line is the relevant code segment of Experiment 3 Task 1

┇ ; Before this line is the relevant code segment of Experiment 3 Task 1

EXIT:

cli ; restore interrupt vector

mov ax,OLDINT1

mov fs:[1\*4],ax

mov ax,OLDINT1+2

mov fs:[1\*4+2],ax

mov ax,OLDINT3

mov fs:[3\*4],ax

mov ax,OLDINT3+2

mov fs:[3\*4+2],ax

sti

MOV AH,4CH ; exit

INT 21H

NEWINT: iret

TESTINT: jmp ADR1

┇ ; After this line is the relevant code segment of Experiment 3 Task 1

┇ ; Before this line is the experiment three tasks one function four codes

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

cli ; stack inspection antitrace

XOR AH,AH

push BRK29 ; push the address of BRK29 onto the stack

IMUL AX,3

XOR DX,DX

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

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

MOV CL,DH

XCHG AX, ES:[EBP+EAX];[BX+SI]<[BX+DI], exchange

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

ADD AX,3 ;AH>=AL, without exchange, (AH) is directly compared with the last number, which is equivalent to j++

CMP AUTH,CL

JE BRK28

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

pop dx

MOV AX,SI ; equivalent to j=i

mov cx,[esp-2] ;Get the word on the top of the stack (address of PASS2) to

sti

ADDDX,3

jmp cx ; will not transfer to PASS2 if tracked

BRK29:

LEADX,TIP\_12 ; tabulation

┇ ; Before this line is the relevant code segment of the experiment three tasks one work function five

### Experimental procedure

1. Prepare the experimental environment.

2. Based on the t1.asm of Experiment 3, insert relevant codes according to various anti-tracking methods given by the sample program . When adding anti-tracking code and interference code, care should be taken not to interfere with the data transmission of the original program, and the switch interrupt operation cannot be crossed.

3. Repeat the assembly and connection process to solve the assembly problems that may be encountered.

4. Run the program to check whether it works normally . Debugging with TD becomes more difficult due to the added anti-trace code . At this time, when debugging, you should consider checking the anti-tracking program segment by segment and use the elimination method to finally find the problematic segment.

5. Add interference program segments to further strengthen the anti-tracking ability of the program , and the precautions are the same as in 2 .

6. Repeat steps 3 and 4 until the program runs normally and gives correct results.

7. Add a decryption function in the profit calculation of function 3 , and block data when logging in as a tourist in the profit rate display of function 5 , and add interference codes and anti-tracking statements at the same time.

8. Repeat steps 3 and 4 until the program runs normally and gives correct results.

### 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.

Insert relevant codes in the user name, password comparison, and login method check sections according to the various anti-tracking methods given by the sample program . When reading the sample program, I didn't understand PASS3: mov bx,es:[1\*4] and inc bx at the beginning , because here is the statement that should jump to TESTINT , but to the interrupt vector table from the memory What is entered is NEWINT, but it is not clear what statement to execute next after adding one to the IP . Ask the teacher and classmates to understand that the space occupied by the data in the CS segment is different. For example, iret only occupies 1 byte , so after doing inc bx, you can reach the location of TESTINT .

3. Everything is normal when compiling and connecting, but there are many crashes during the running process . After locking the problematic program segment with the elimination method , it is found that the cause of the error is that the new program segment interferes with the data transmission of the original code . Correct the relevant errors and run again.

4. Run the program and find that the program can output the correct result in decimal , as shown in Figure 3.4.2 .

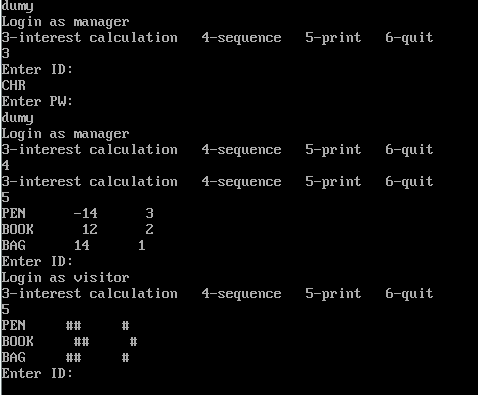


Figure 3.4.2 The program is running normally

## task 1

### Experimental procedure

1. Prepare the experimental environment.

's program in the code window of TD , first observe the interrupt vector table to see if you can see the clue.

3. Execute the program step by step , and assist debugging by changing the IP in combination with the prompts for each address in the disassembly statement .

4. Try to find the processing statement for the password , reverse the password, execute the program normally and enter the system as the owner by entering the correct password .

### 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. After opening the program with TD , first find that the data segment is loaded into ds, and the interrupt vector table is loaded into es, as shown in Figure 3.5.1 .

It is known that loading the interrupt vector table into es involves anti-tracing of the interrupt vector table, so after loading the data segment into ds, modify the IP and jump directly to the function code start EA -0054, as shown in Figure 3.5.2 .

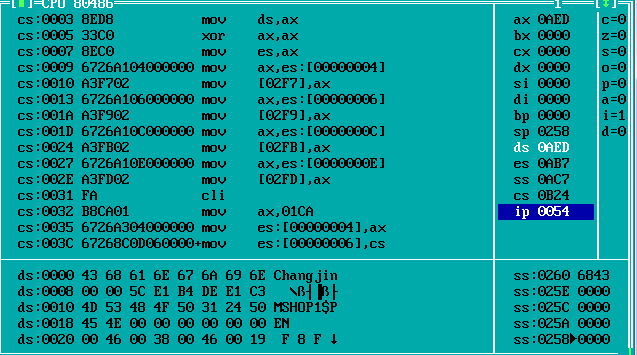


Figure 3.5.1 Modify the IP to 0054, skip the modification of the interrupt vector table

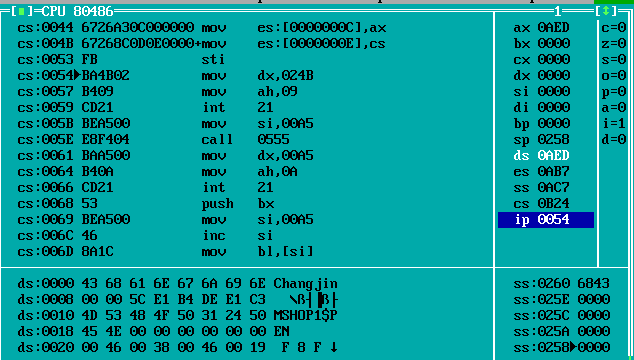


Figure 3.5.2 function code starts

3. Single-step execution shows that cs:0054-cs:005B are prompts for entering passwords. It can also be seen from Figure 3.5.2 that it will jump to cs:0555 for execution . Combined with cs:0061-cs:0066 input password No. 10 call , check the relevant code function with the content of the ds section , judge that the function of this subroutine is to clear the 7 bytes starting from ds:00a6 to save the password to be input , as shown in Figure 3.5.3 .

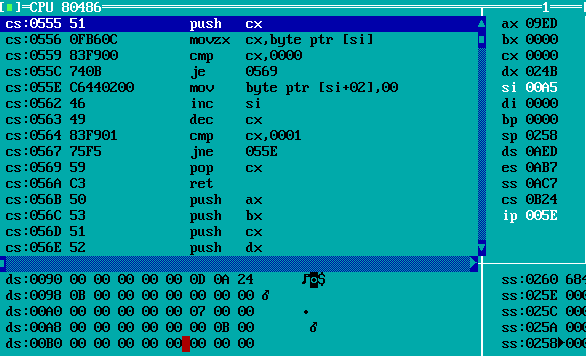


Figure 3.5.3 Interpretation of the subroutine function at cs :0555

4. Conduct code testing by calling No. 10, and the testing password is 308532 . The password storage string length is 7, as shown in Figure 3.5.3 , but because a carriage return character needs to be stored, the effective password length is actually 6. The function of cs:0068-cs:0075 is to delete the carriage return character at the end of the password string , the implementation is shown in Figure 3.5.4, focus on the changes of ds: 00ad ; the function of cs:0076-cs:007b is carriage return and line feed, The implementation is shown in Figure 3.5.4 and Figure 3.5.5.

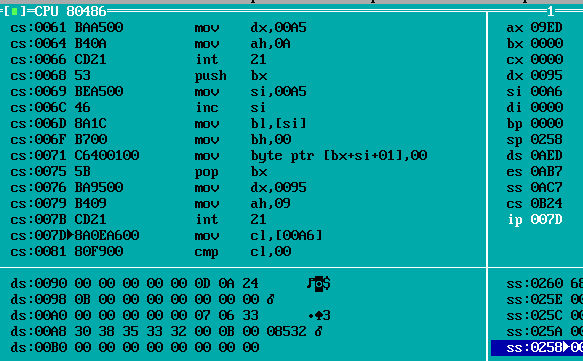


Figure 3.5.4 3 Call No. 5 to obtain the entry address of 1 H

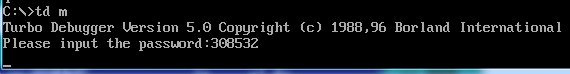


Figure 3.5.5 The cursor moves to the next line

5. cs:007d-cs:0084 judge the length of the input password string , if it is 0 , skip ; As shown in .6 . The number is uniquely determined to be 4 within 0-7 , which is the correct password string length .

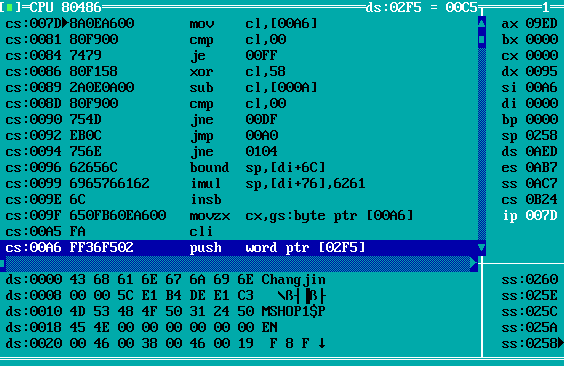


Figure 3.5. 6 Deciphering the password string length

6. Try again with the 4-digit password wasd . The program turns to cs:009 f for execution , and sets cx to 4, as shown in Figure 3.5.7 . cs:00a5 -cs:00b5 is the stack anti-trace , and the stack value should be recorded when cracking , here is 00c5. cs:00a a, cs00ad set si and dl to 0 and 3 respectively, which will work later.

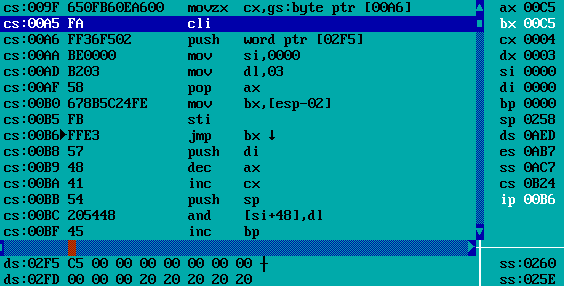


Figure 3.5. 6 crack stack anti-trace

7. cs:00c5 -cs:00d6 is a cycle, the number of times is 4, as shown in Figure 3.5.7 . Observing the ds section , we can know that the password operation starts here . For the four bytes, there are (x-29H)\*4= 0E1H/0B4H/0DEH/0E1H , and the password in the reverse derivation system should be 'test' .

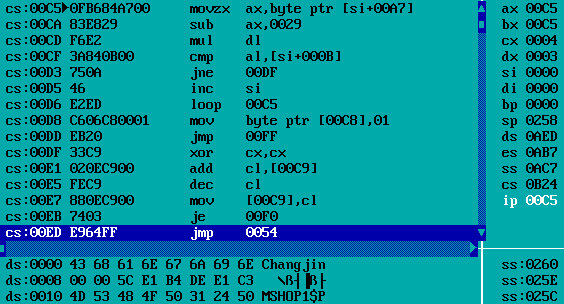


Figure 3.5.7 Crack the password

8. Use test to test the input and log in to the system successfully , as shown in Figure 3.5.8 .

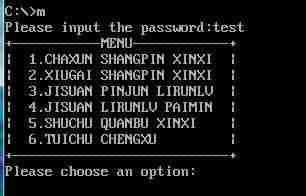


Figure 3.5. 8 Successful login

9. Figure 3.5.9, Figure 3.5.10, and Figure 3.5.11 respectively show the information of three commodities in two stores. Since you are logged in as a store owner, the purchase price has been decrypted.

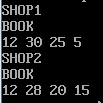
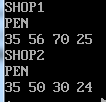
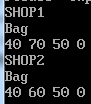
  

Figure 3.5. 9 & Figure 3.5. 10 & Figure 3.5. 11 All information of the three commodities

# Summary and experience

The experiment of Task 1 made me know how the IP and CS of the interrupt vector table are represented in memory , and at the same time I am familiar with using int 35 H The method of viewing the entry address. It should be noted that the value of some interrupt vector tables will not be displayed as the original value in TD , so it should be extracted from the memory instead of observing when using it . Task 2 made me understand the takeover method of the interrupt vector table, and learned the way of memory residence. At the same time, I also understand that the modification of the system function call is not as simple as I imagined . It needs to be judged by the background program provided by the original function call before it can be used. Task 3 gave me a certain understanding of CMOS data addresses , and I am more familiar with the conversion between 8421 codes and ASCII codes .

In task 4, I came into contact with reverse engineering related concepts from the perspective of the defender , and learned about simple data encryption methods and various disassembly methods. I also feel that anti-tracking is an operation of " if you want to deceive your opponent , you must first deceive yourself". The mixture of functional codes , disassembly codes and irrelevant codes has caused some difficulties in my programming . In this case I need to take certain measures to make myself see clearly what I am writing . In addition to full comments, I also use the method of indentation difference to distinguish various codes, which reduces my programming burden to a certain extent . With the anti-tracking code, debugging also becomes difficult. If you don’t pay attention to the IP during single-step execution, you will be abducted by your own anti-tracking . This requires us to understand the anti-tracking method and its own functions. Be aware of it , so that you can jump over the trap during single-step debugging , and you won't be fooled by yourself . At the same time , the elimination method can also be adopted to check the disassembly function one by one to determine the wrong paragraph and gradually narrow the scope of inspection .

In Mission 5 I did reverse engineering for the first time from the attacker 's point of view . My strongest feeling is still to know the basic operations, such as what data segment is installed in the current code. At the beginning of debugging, because I didn't even load the data segment into DS, the console window was not displayed at all when I wanted to enter the code , which made me toss for a long time . Reverse engineering is an extremely patient job , especially when faced with countless jumps. This time I would like to thank my teammates for not setting up too many obstacles on the jump, so that I could successfully crack the program, but after work, the requirements for anti-tracking and reverse engineering of the program have increased a lot. Anti-tracking The link will also be more complicated. For example, the encryption method will definitely not be like writing a simple function or doing a logical operation like it is now . Therefore, it is very important to have a solid foundation of offensive and defensive skills while getting more familiar with the code in school , so that you will not be trapped by some basic logic in practice, and you can use more energy to interpret more complex encryption processes .

Thinking questions ( not involved in the experimental steps ) :

One ,

5. If the CS/IP of the interrupt handler is modified , the wrong program segment will be found according to the wrong entry address , and the proper function will not be obtained during execution , and the system may crash .

two ,

2. Instead of residing the new interrupt handler in the memory, try to input the string of lowercase letters after writing the new entry address to see if the uppercase letters are displayed on the screen . The new interrupt handler cannot be resident in memory until the output is correct .

Specific operation: delete the memory-resident statement and define a string with a length of 12 for input . After loading the new interrupt handler, use lowercase to input strings. After the end, directly uninstall the new interrupt handler and try console input. The result is shown in the figure below, indicating that the interrupt handler is compiled correctly and can be resident in memory.



Figure 4.1 After exiting the program, the input returns to normal

3. You can return to DOS, and DOS works normally . If the memory-resident program is executed repeatedly , the "old" IP and CS saved last time are actually the IP and CS after the interrupt handler was run last time , which actually form a nest, and the uninstall program needs to be executed the same number of times. Put the initial IP and CS back into the interrupt vector table , as shown in Figure 4.2 . But there is also a method to restore to normal with one uninstall no matter how many memory resides are performed : record the IP and CS saved during the first memory reside, and put them directly into the interrupt vector table . The execution result is shown in Figure 4.3 . Specifically for this experiment , the initial IP of CS is 11E0H, and the initial CS is 0F000H.

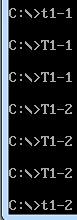
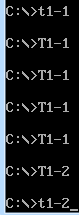
 

Figure 4.2 Nesting of memory-resident programs Figure 4.3 Exit once \_

4. Will not be replaced , as shown in Figure 4.4 .

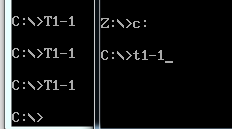


Figure 4.4Modifications in one virtual machine have no effect on another virtual machine

TD to single-step for the second time after performing a memory residency . It is observed that the values of IP and CS saved this time have been changed . The value of IP is the same as that of the double-word variable OLD\_INT defined before the first instruction when writing the program. The situation matches, as shown in Figure 4.5 .

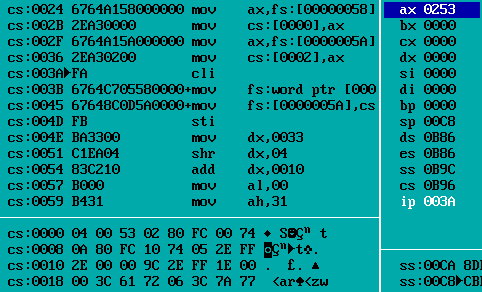


Figure 4.5 The values of I P and CS are changed

6. In view of the initial IP, CS is defined in the first 4 bytes of 16H new CS, so subtract 4 after taking the CS address in the uninstall program to get the old IP, and subtract 2 to get the old CS, and load it into the interrupt vector table The corresponding position of 16H is completed ; it can be judged by the initial IPC and CS observed in TD , and if it is the initial value, it will return without unloading.

three,

1. Give the target port to AL, and use port 70H to indicate that the byte is ready to be accessed; use 71H to hand over the output parameter of the accessed byte to AL.

2. Take the content distance in 00 H (seconds) . If the current system scanning seconds is 59, the compressed BCD code of one byte is 01011001, that is, the high nibble stores the 8421 code of the tens digit , and the bottom nibble stores the 8421 code of the ones digit .

four + five ,

1. Observe the data in the ds segment , and exclude the prompt information with obvious semantics and the data information with a regular structure, and it is easier to find out the password segment saved in clear code from the remaining data , as shown in Figure 4.6 . At this time The password field is 'PASSWD' .

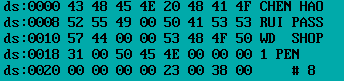


Figure 4.6 Save the password in plain text

2. The first method is to directly use TD to observe in the memory , as shown in Figure 4.7; the second method is to directly open the program with a binary editing tool, and search for the purchase price definition in the text.

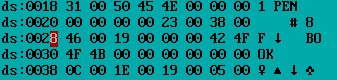


Figure 4.7 Save the purchase price in plain text

3 . Design ideas : First, set the possible upper and lower limits of the password length, traverse all combinations of numbers and characters under the current string length limit and call the file to be cracked to try . If the decryption succeeds or fails , return the corresponding value to decide whether to continue the next attempt or display the correct password .

4. For functional program paragraphs, you can directly use dynamic disassembly and single-step execution to observe the program running process; but for anti-trace statements, parts with more interference codes or transfer through address indirect instructions , you need to use static anti-trace . For stack During single-step debugging such as anti-tracking , the part of the target value will be modified , the original value will be recorded, and the IP will be entered to manually jump ; for indirect transfer through the address , it is necessary to observe the address saved in the data segment to determine whether it can continue after transferring to the new statement Execute normally until the password encryption code is deciphered or the hidden target information is deciphered directly.

5. Familiarize yourself with the general operation steps of anti-trace through examples , and you can easily distinguish it during anti-trace. For specific examples, see 3.5.2 ; for stack anti-trace , modify the target value during single-step debugging such as modifying the interrupt vector table part , record the original value and enter the correct IP at call or jump and jump manually; for indirect transfer through address , you need to observe the address saved in the data segment to determine whether it can continue to execute normally after transferring to the new statement until Decipher the password encryption code or directly decipher the hidden target information; for the timing interruption, you need to manually modify the part of the IP to skip the two timing comparisons .

# references

] Wang Yuanzhen, Cao Zhongsheng , Han Zongfen "80X86 Assembly Language Programming" "Chapter 6 Input/Output and WIN32 Programming"