

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实验**

**实验名称： 实验四 中断与反跟踪**

**实验时间： 2018-4-28，14：00-17：30 实验地点： 南一楼**

**指导教师： 朱虹**

**专业班级： 计算机科学与技术1601班**

**学 号： U201614531 姓 名： 刘本嵩**

**同组学生： 尹宏运 报告日期： 2018年 4 月 28日**

**原创性声明**

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

特此声明！

学生签名：

日期：

成绩评定

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

指导教师签字：

日期：

**目录**

[1 实验目的与要求 2](#_Toc372321810)

[(1) 掌握中断矢量表的概念； 2](#_Toc1166641215)

[(2) 熟悉I/O访问，BIOS功能调用方法； 2](#_Toc1832033884)

[(3) 掌握实方式下中断处理程序的编制与调试方法； 2](#_Toc711244742)

[(4) 熟悉跟踪与反跟踪的技术； 2](#_Toc828040690)

[(5) 提升对计算机系统的理解与分析能力。 2](#_Toc1608036253)

[2 实验内容 3](#_Toc1404305593)

[2.1 任务1：用三种方式获取中断类型码21H对应的中断处理程序的入口地址 3](#_Toc2129865044)

[2.2 任务2：编写一个接管键盘中断的中断服务程序并驻留内存 3](#_Toc391257361)

[2.3 任务3：读取CMOS内指定单元的信息，按照16进制形式显示在屏幕上 3](#_Toc1455872830)

[2.4 任务4：数据加密与反跟踪 4](#_Toc964528818)

[2.5 任务5：跟踪与数据解密 4](#_Toc1227319960)

[3 实验过程 5](#_Toc1197105947)

[3.1 任务1 5](#_Toc1516408315)

[3.1.1 实验步骤 5](#_Toc1086232995)

[3.1.2 程序源代码 5](#_Toc595987315)

[3.1.3 实验记录与分析 6](#_Toc812074241)

[3.2 任务2 8](#_Toc1204465613)

[3.2.1 实验步骤 8](#_Toc1570523159)

[3.2.2 源代码 9](#_Toc936131960)

[3.2.3 实验记录与分析 10](#_Toc2115906326)

[3.3 任务3 12](#_Toc859462146)

[3.3.1 实验步骤 12](#_Toc1616197396)

[3.3.2 程序源代码 12](#_Toc197597666)

[3.3.3 实验记录与分析 12](#_Toc266212991)

[3.4 任务4 14](#_Toc1079602268)

[3.4.1 实验步骤 14](#_Toc406237496)

[3.4.2 程序源代码 14](#_Toc1816838421)

[3.4.3 实验记录与分析 36](#_Toc1814391811)

[3.5 任务4 37](#_Toc247251187)

[3.5.1 实验步骤 37](#_Toc1758315691)

[3.5.2 对方可执行文件 37](#_Toc39229973)

[3.5.3 实验记录与分析 37](#_Toc1413892403)

[4 总结与体会 40](#_Toc1442865927)

[5 参考文献 41](#_Toc750474715)

# 实验目的与要求

1. 掌握中断矢量表的概念；
2. 熟悉I/O访问，BIOS功能调用方法；

(3) 掌握实方式下中断处理程序的编制与调试方法；

(4) 熟悉跟踪与反跟踪的技术；

(5) 提升对计算机系统的理解与分析能力。

# 实验内容

## 任务1：用三种方式获取中断类型码21H对应的中断处理程序的入口地址

要求：首先要进入虚拟机状态，然后

1. 直接运行调试工具（TD.EXE），观察中断矢量表中的信息。

2. 编写程序，用 DOS系统功能调用方式获取，观察功能调用相应的出口参数与“（1）”看到的结果是否相同 （使用TD观看出口参数即可）。

3. 编写程序，直接读取相应内存单元，观察读到的数据与“（1）”看到的结果是否相同 （使用TD观看程序的执行结果即可）。

## 任务2：编写一个接管键盘中断的中断服务程序并驻留内存

要求：

1. 在 DOS虚拟机或DOS窗口下执行程序，中断服务程序驻留内存。

2. 在DOS命令行下键入小写字母，屏幕显示为大写，键入大写时不变。执行TD，在代码区输入指令“mov AX,0”看是否能发生变化。

3. 选作：另外编写一个中断服务程序的卸载程序，将键盘中断服务程序恢复到原来的状态（也就是还原中断矢量表的信息，先前驻留的程序可以不退出内存）。

## 任务3：读取CMOS内指定单元的信息，按照16进制形式显示在屏幕上

1. 先输入待读取的CMOS内部单元的地址编号（可以只处理编号小于10的地址单元）。再使用IN/OUT指令，读取CMOS内的指定单元的信息。

2. 将读取的信息用16进制的形式显示在屏幕上。若是时间信息，可以人工判断一下是否正确。

## 任务4：数据加密与反跟踪

在实验三任务1的**网店商品信息管理程序**的基础上，增加输入用户名和密码时，最大错误次数的限制，即，当输入错误次数达到三次时，直接按照未登录状态进入后续功能。老板的密码采用密文的方式存放在数据段中，各种商品的进货价也以密文方式存放在数据段中。加密方法自选。

可以采用计时、中断矢量表检查、堆栈检查、间接寻址等方式中的一种或多种方式反跟踪（建议采用两种反跟踪方法，重点是深入理解和运用好所选择的反跟踪方法）。

为简化录入和处理的工作量，只需要定义三种商品的信息即可。

**提示：**为了使源程序的数据段中定义的密码、进货价等在汇编之后变成密文（也就是在最后交付出去的执行程序中看不到明文），可以使用数值运算符（参见教材P48）对变量的初始值进行变换。例如，如果想使进货价50变成密文，加密算法是与老板密码中的字符“W”做异或运算，则可写成：

DB 50 XOR ‘W’

## 任务5：跟踪与数据解密

解密同组同学的加密程序，获取各个商品的进货价。

注意：两人一组，每人实现一套自己选择的加密与反跟踪方法，把执行程序交给对方解密（解密时间超过半小时的，说明反跟踪方法基本有效）。如何设计反跟踪程序以及如何跟踪破解，是本次实验报告中重点需要突出的内容。（分组可以按照学号顺序依次构成两人一组。也可以自行调整。如果班上人数是奇数，则三人一组，甲解密乙的，乙解密丙的，丙解密甲的）

# 实验过程

## 任务1

### 实验步骤

1. 直接在虚拟机环境下运行TD，观察中断向量表中21H处中断处理程序的入口地址。

2. 编写程序，在该程序中使用int 21H调用35H的DOS系统功能调用，该系统功能调用的出口参数即为相应中断类型码所对应的入口地址；同时，在该程序中直接读取相应的内存单元，观察读到的数据。

3. 汇编和连接编写的程序，确保所编写的程序没有错误后用TD加载并运行，观察相应的结果。

### 程序源代码

;.386

.model small

;.stack 1024

.data

.code

assume ds:nothing

get\_callback\_from\_ivt\_1:

; arg: interrupt number in al

; return addr in bx

xor ah, ah

add ax, ax

add ax, ax

mov bx, ax

xor ax, ax

mov ds, ax

mov es, ds:[bx+2]

mov bx, ds:[bx]

ret

get\_callback\_from\_ivt\_2:

; arg: interrupt number in al

; return addr in es:bx

mov ah, 35h

int 21h

ret

start:

mov al, 1h

call get\_callback\_from\_ivt\_1

nop

nop

mov al, 1h

call get\_callback\_from\_ivt\_2

nop

nop

mov al, 10h

call get\_callback\_from\_ivt\_1

nop

nop

mov al, 10h

call get\_callback\_from\_ivt\_2

nop

nop

mov ah, 4ch

xor al, al

int 21h

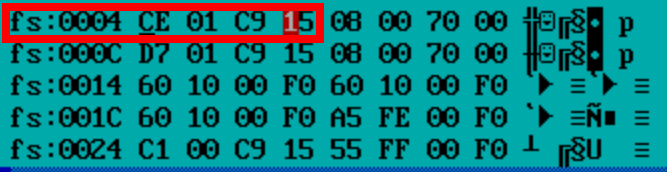
end start

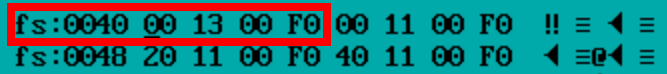
### 实验记录与分析

先汇编生成 target

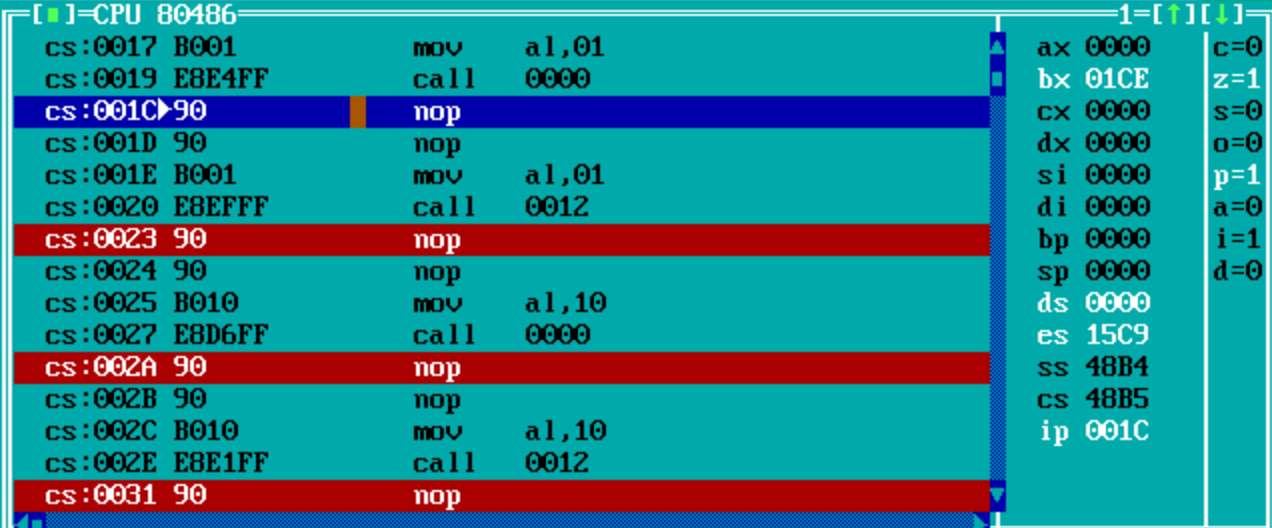


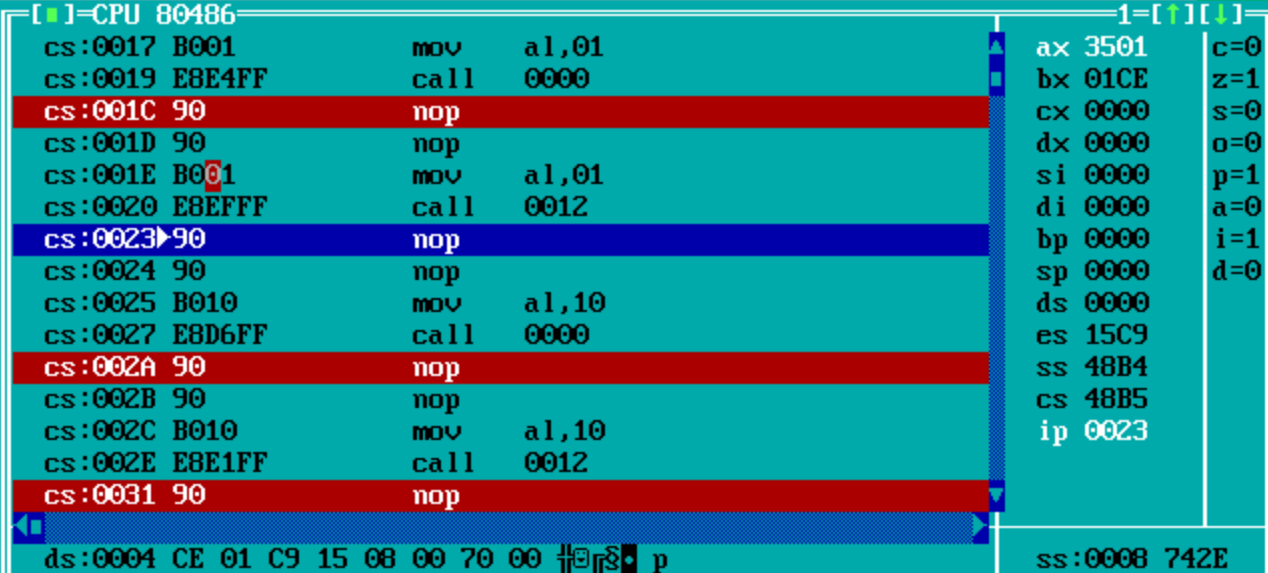
1. 使用 td 查看指定位置的内存,直接确定 ivt 内容。

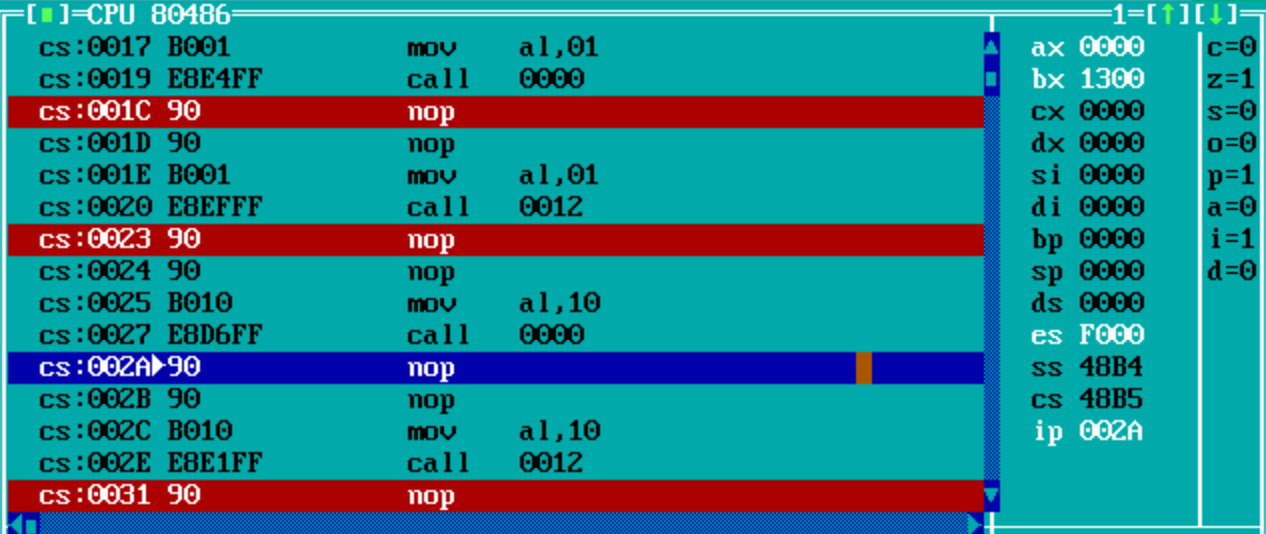


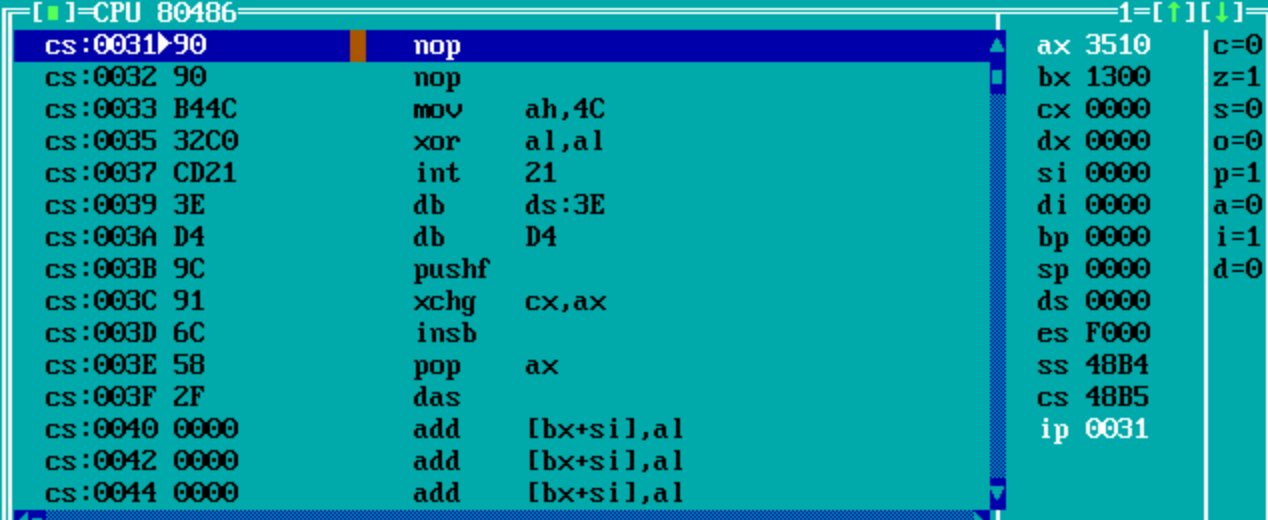


1. 运行程序,在指定的位置打断点,查看分别使用两种方式获得的 ivt 处理程序地址。









显然,3 种方式获得的地址结果相符。

## 任务2

### 实验步骤

1. 根据任务2的要求，编写相关的两个程序，即接管中断并驻留的程序，以及卸载中断的程序。

2. 对于接管中断程序，接管键盘中断并驻留的过程是，首先在程序的前4个字节保存原INT 16H中断的CS和IP，然后关闭中断，用新的CS和IP去替换原INT 16H中断的CS和IP，然后打开中断。最后计算需要驻留在内存中的节数（字节数除以4并向上取整，并且还要加上段前的10H个字节），使用系统31号调用使程序驻留内存。

3. 对于卸载中断程序，中断写在程序的过程是，首先将新INT 16H的中断的IP减去4，取得原INT 16H中断的CS和IP，然后关闭中断，用原INT 16H的CS和IP去还原新的INT 16H的CS和IP，然后打开中断，最后使用系统49H调用释放驻留在内存中的新INT 16H中断即可。

4. 首先各自汇编和连接编写好的接管中断程序和卸载中断程序，确认汇编和连接过程没有错误后，先在命令行中输入“Test”，观察结果。

5. 然后运行接管中断程序，程序返回后再输入“Test”，观察结果。

6. 最后运行卸载中断程序，程序返回后再输入“Test”，观察结果。

### 源代码

.386

.model small; .code code ;;;;;;;use16

code segment use16

real\_int\_addr dw ?,?

fake\_int\_0x16:

cmp ah, 00h

je next

cmp ah, 10h

je next

jmp dword ptr real\_int\_addr

next:

push bp

mov bp, sp

pushf

call dword ptr real\_int\_addr

cmp al, 97

jb quit

cmp al, 122

ja quit

sub al, 32

quit:

pop bp

iret

start:

xor ax, ax

mov ds, ax

mov ax, ds:[16h \* 4]

mov real\_int\_addr, ax

mov ax, ds:[16h \* 4 + 2]

mov real\_int\_addr + 2, ax

cli

mov word ptr ds:[16h \* 4], offset fake\_int\_0x16

mov ds:[16h \* 4 + 2], cs

sti

mov dx, offset start + 15

shr dx, 4

add dx, 10h

mov ax, 3100h

int 21h

end start

code ends

.386

code segment use16

start: xor ax, axmov ds, ax

mov ax, ds:[16h \* 4]

mov cx, ds:[16h \* 4 + 2]

mov es, cx

sub ax, 4

cli

mov si, ax

mov bx, es:[si]

mov word ptr ds:[16h \* 4], bx

add si, 2

mov ax, es:[si]

mov ds:[16h \* 4 + 2], ax

sti

int 49h

mov ax, 4c00h

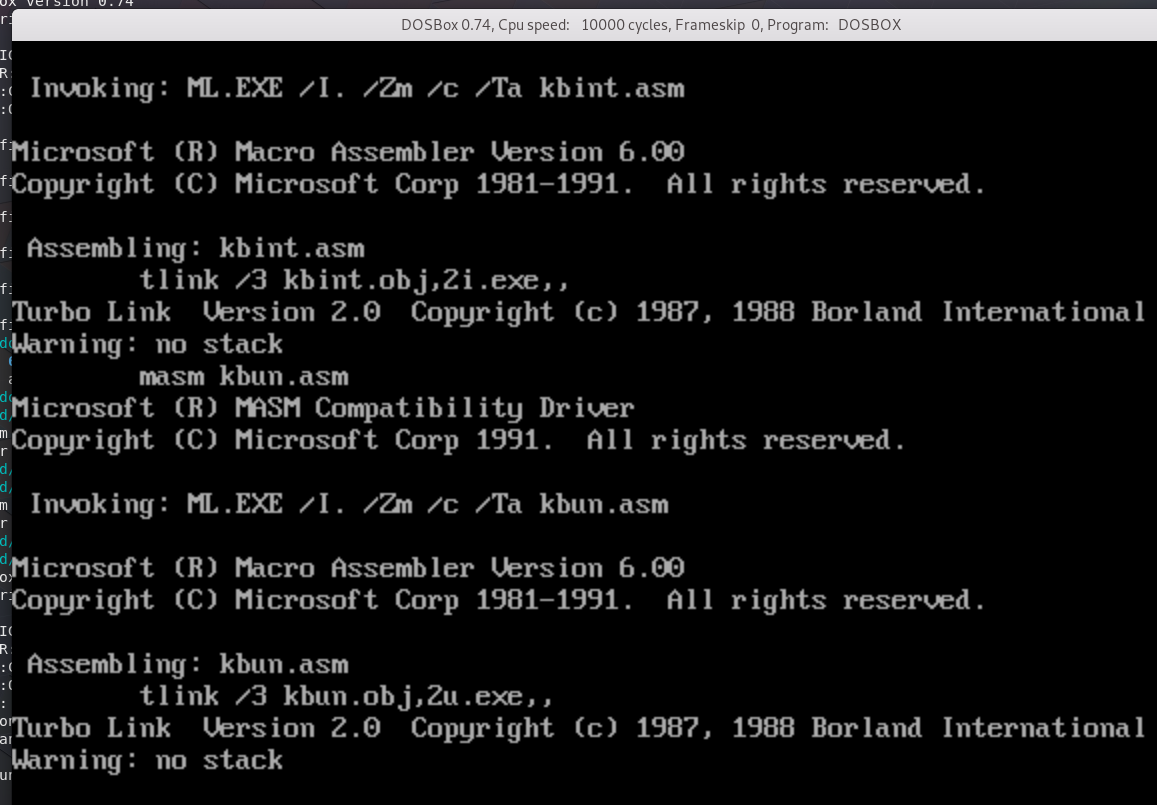
int 21h

code ends

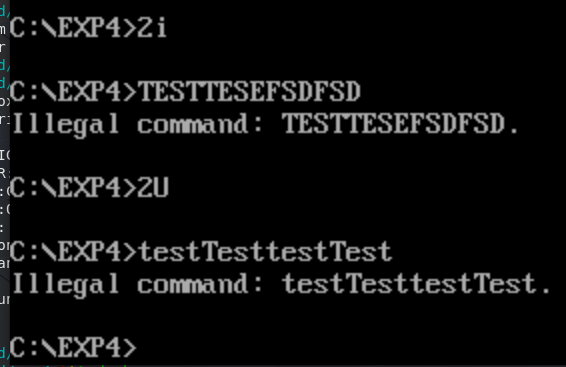
end start

### 实验记录与分析

先汇编生成 target



然后运行查看效果



## 任务3

### 实验步骤

1. 首先根据题目的要求完成任务3的程序，确定汇编连接没有错误后运行。

2. 尝试输入0，2，4，6，7，8，9，观察给出的答案是否正确。

### 程序源代码

.model small

.data

info db 'Port No. ? $'

buffer db 2, 1

number db ?, ?

dos\_newline db 10, '$'

.code

assume ds:\_data

include rlib.inc

start:

mov ax, \_data

mov ds, ax

print\_str\_at info

read\_str\_to buffer

mov al, number

sub al, '0'

out 70h, al

in al, 71h

xor ah, ah

print\_str\_at dos\_newline

call rlib\_print2d\_hex

call rlib\_exit

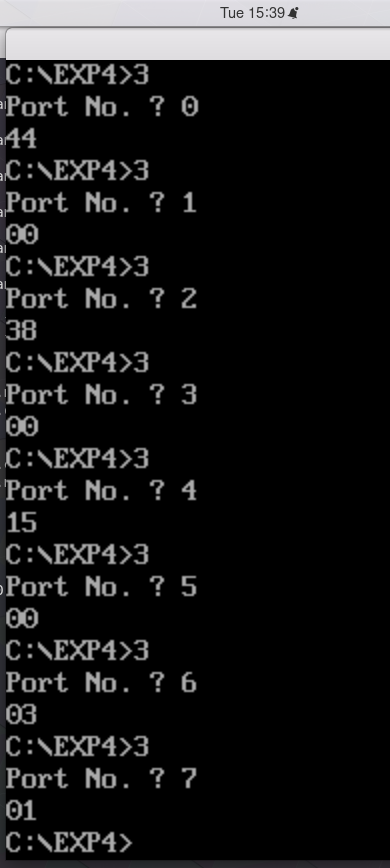
end start

### 实验记录与分析

先汇编生成可执行文件



然后运行并测试,与当前时间进行比较(15:38:44),发现运行正确。同时,这个程序支持以 1a 的形式显示 16 进制字符。



## 任务4

### 实验步骤

1. 修改之前实验的程序，令其登陆之后直接调用查询商品信息功能，以便简化这些无助于混淆的无关代码，为加密代码和花指令留出空间。

2. 实现RSA加密和大数算法(蒙哥马利模乘法)，使用dos能承受的最长(但不少于256位，不超过2048位)密钥长度，存储密文和公钥，然后将用户输入的明文使用公钥加密之后和密文比对(即 将RSA作为缓慢哈希算法使用)。但是相比缓慢哈希算法，RSA的解密程序如果不考虑时间复杂度很容易实现(攻击者当然会使用现代操作系统进行攻击)，有利于负责解密的同学的心情。同时，如果想要做到数据不可被窃取，应当将密码作为对称密钥将所有数据加密。

3. 显然攻击方可以通过反汇编轻易拿到汇编源代码。如果假设攻击方有修改源码的能力，则其只需将没有内联的verify函数的返回值强行修改，或者将软件中存储的密文强行修改。这些可靠的静态调试方法使得计时、反调试等功能无从应用，因此花指令是本次反追踪的重点，但它的破解难度在动态调试工具下也大大降低。要想真正使得数据不可窃取，必须借助密码学知识。

4. 组装start函数，组装所需的object，debug，release。

### 程序源代码

.387

PUBLIC fake\_main\_

EXTRN \_\_STK:BYTE

EXTRN \_\_U4D:BYTE

EXTRN \_\_U4M:BYTE

EXTRN \_\_I8DQ:BYTE

EXTRN \_\_I8DR:BYTE

EXTRN FIDRQQ:BYTE

EXTRN FIWRQQ:BYTE

EXTRN \_\_8087:BYTE

EXTRN \_\_init\_87\_emulator:BYTE

EXTRN \_small\_code\_:BYTE

DGROUP GROUP CONST,CONST2,\_DATA

\_TEXT SEGMENT BYTE PUBLIC USE16 'CODE'

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

r\_hash\_:

push ax

mov ax,0aH

call near ptr \_\_STK

pop ax

push bx

push cx

push si

push di

mov bx,ax

mov di,8bfH

mov si,131H

tmp1:

mov al,byte ptr [bx]

test al,al

je tmp4

xor ah,ah

mov cx,ax

and cl,0fH

xor dx,dx

jcxz tmp3

tmp2:

shl ax,1

rcl dx,1

loop tmp2

tmp3:

add di,ax

adc si,dx

inc bx

jmp tmp1

tmp4:

mov ax,di

mov dx,si

tmp5:

pop di

pop si

pop cx

pop bx

ret

meg\_mod\_:

push ax

mov ax,4

call near ptr \_\_STK

pop ax

push bp

mov bp,sp

xor ax,ax

pop bp

ret 2

naive\_mod\_:

push ax

mov ax,12H

call near ptr \_\_STK

pop ax

push si

push di

push bp

mov bp,sp

sub sp,4

push ax

mov di,dx

mov word ptr -4[bp],bx

mov word ptr -2[bp],cx

xor si,si

cmp word ptr 0aH[bp],0

jne tmp8

cmp word ptr 8[bp],2710H

jae tmp8

mov ax,bx

mov dx,cx

mov bx,word ptr -6[bp]

mov cx,di

call near ptr \_\_U4D

tmp6:

mov ax,si

cwd

cmp dx,word ptr 0aH[bp]

jb tmp7

jne tmp9

cmp ax,word ptr 8[bp]

jae tmp9

tmp7:

mov ax,bx

mov dx,cx

mov bx,word ptr -4[bp]

mov cx,word ptr -2[bp]

call near ptr \_\_U4M

mov bx,word ptr -6[bp]

mov cx,di

call near ptr \_\_U4D

inc si

jmp tmp6

tmp8:

mov ax,word ptr 8[bp]

mov dx,word ptr 0aH[bp]

shr dx,1

rcr ax,1

push dx

push ax

mov ax,word ptr -6[bp]

mov dx,di

call near ptr naive\_mod\_

mov bx,ax

mov cx,dx

call near ptr \_\_U4M

mov bx,word ptr -6[bp]

mov cx,di

call near ptr \_\_U4D

test byte ptr 8[bp],1

je tmp9

mov ax,bx

mov dx,cx

mov bx,word ptr -4[bp]

mov cx,word ptr -2[bp]

call near ptr \_\_U4M

mov bx,word ptr -6[bp]

mov cx,di

call near ptr \_\_U4D

tmp9:

mov ax,bx

mov dx,cx

mov sp,bp

pop bp

pop di

pop si

ret 4

fake\_verify\_:

push ax

mov ax,14H

call near ptr \_\_STK

pop ax

push si

push di

push bp

mov bp,sp

push ax

push dx

push bx

push cx

xor si,si

jmp tmp11

tmp10:

inc si

cmp si,20H

jge tmp14

tmp11:

mov bx,word ptr -2[bp]

add bx,si

mov al,byte ptr [bx]

xor ah,ah

mov cx,ax

xor di,di

test ax,ax

jne tmp12

mov bx,si

shl bx,1

shl bx,1

add bx,word ptr -4[bp]

mov ax,word ptr 2[bx]

or ax,word ptr [bx]

je tmp14

tmp12:

mov ax,word ptr -6[bp]

cwd

push dx

push ax

mov ax,word ptr -8[bp]

cwd

mov bx,cx

mov cx,di

call near ptr naive\_mod\_

mov bx,si

shl bx,1

shl bx,1

add bx,word ptr -4[bp]

cmp dx,word ptr 2[bx]

jne tmp13

cmp ax,word ptr [bx]

je tmp10

tmp13:

xor ax,ax

jmp tmp15

tmp14:

mov ax,1

tmp15:

mov sp,bp

pop bp

pop di

pop si

ret

r\_print\_2d\_:

push ax

mov ax,12H

call near ptr \_\_STK

pop ax

push bx

push cx

push dx

push si

push di

push bp

mov bp,sp

sub sp,4

mov cx,ax

mov ax,ds

mov es,ax

lea di,-4[bp]

mov si,offset DGROUP:tmp75

movsw

movsb

mov bx,0aH

mov ax,cx

cwd

idiv bx

cwd

idiv bx

add dx,30H

mov byte ptr -4[bp],dl

mov ax,cx

cwd

idiv bx

add dx,30H

mov byte ptr -3[bp],dl

lea ax,-4[bp]

call near ptr r\_print\_

tmp16:

mov sp,bp

pop bp

tmp17:

pop di

pop si

tmp18:

pop dx

pop cx

tmp19:

pop bx

ret

r\_print\_long\_:

push ax

mov ax,1cH

call near ptr \_\_STK

pop ax

push bx

push cx

push si

push di

push bp

mov bp,sp

sub sp,0cH

push ax

push dx

mov ax,ds

mov es,ax

lea di,-0cH[bp]

mov si,offset DGROUP:tmp76

movsw

movsw

movsw

movsw

movsw

movsw

lea di,-0cH[bp]

xor ax,ax

xor bx,bx

mov cx,dx

mov dx,word ptr -0eH[bp]

mov si,offset DGROUP:tmp56

call near ptr \_\_I8DQ

mov si,offset DGROUP:tmp57

call near ptr \_\_I8DR

add dx,30H

adc cx,0

adc bx,0

mov byte ptr -0cH[bp],dl

mov ax,word ptr -0eH[bp]

mov dx,word ptr -10H[bp]

mov bx,0ca00H

mov cx,3b9aH

call near ptr \_\_U4D

mov bx,0aH

xor cx,cx

call near ptr \_\_U4D

add bx,30H

mov byte ptr -0bH[bp],bl

mov ax,word ptr -0eH[bp]

mov dx,word ptr -10H[bp]

mov bx,0e100H

mov cx,5f5H

call near ptr \_\_U4D

mov bx,0aH

xor cx,cx

call near ptr \_\_U4D

add bx,30H

mov byte ptr -0aH[bp],bl

mov ax,word ptr -0eH[bp]

mov dx,word ptr -10H[bp]

mov bx,9680H

mov cx,98H

call near ptr \_\_U4D

mov bx,0aH

xor cx,cx

call near ptr \_\_U4D

add bx,30H

mov byte ptr -9[bp],bl

mov ax,word ptr -0eH[bp]

mov dx,word ptr -10H[bp]

mov bx,4240H

mov cx,0fH

call near ptr \_\_U4D

mov bx,0aH

xor cx,cx

call near ptr \_\_U4D

add bx,30H

mov byte ptr -8[bp],bl

mov ax,word ptr -0eH[bp]

mov dx,word ptr -10H[bp]

mov bx,86a0H

mov cx,1

call near ptr \_\_U4D

mov bx,0aH

xor cx,cx

call near ptr \_\_U4D

add bx,30H

mov byte ptr -7[bp],bl

mov ax,word ptr -0eH[bp]

mov dx,word ptr -10H[bp]

mov bx,2710H

xor cx,cx

call near ptr \_\_U4D

mov bx,0aH

xor cx,cx

call near ptr \_\_U4D

add bx,30H

mov byte ptr -6[bp],bl

mov ax,word ptr -0eH[bp]

mov dx,word ptr -10H[bp]

mov bx,3e8H

xor cx,cx

call near ptr \_\_U4D

mov bx,0aH

xor cx,cx

call near ptr \_\_U4D

add bx,30H

mov byte ptr -5[bp],bl

mov ax,word ptr -0eH[bp]

mov dx,word ptr -10H[bp]

mov bx,64H

xor cx,cx

call near ptr \_\_U4D

mov bx,0aH

xor cx,cx

call near ptr \_\_U4D

add bx,30H

mov byte ptr -4[bp],bl

mov ax,word ptr -0eH[bp]

mov dx,word ptr -10H[bp]

mov bx,0aH

xor cx,cx

call near ptr \_\_U4D

mov bx,0aH

xor cx,cx

call near ptr \_\_U4D

add bx,30H

mov byte ptr -3[bp],bl

mov ax,word ptr -0eH[bp]

mov dx,word ptr -10H[bp]

mov bx,0aH

xor cx,cx

call near ptr \_\_U4D

add bx,30H

mov byte ptr -2[bp],bl

tmp20:

cmp byte ptr [di],30H

jne tmp21

inc di

jmp tmp20

tmp21:

mov ax,di

call near ptr r\_print\_

mov sp,bp

pop bp

jmp near ptr tmp5

r\_strlen\_:

push ax

mov ax,4

call near ptr \_\_STK

pop ax

push bx

mov bx,ax

xor ax,ax

tmp22:

cmp byte ptr [bx],0

jne tmp23

jmp near ptr tmp19

tmp23:

inc bx

inc ax

jmp tmp22

r\_streql\_:

push ax

mov ax,6

call near ptr \_\_STK

pop ax

push bx

push si

mov bx,ax

mov si,dx

tmp24:

mov al,byte ptr [bx]

test al,al

je tmp26

cmp al,byte ptr [si]

je tmp25

xor ax,ax

pop si

pop bx

ret

tmp25:

inc bx

inc si

jmp tmp24

tmp26:

cmp byte ptr [si],0

jne tmp27

mov ax,1

pop si

pop bx

ret

tmp27:

xor ah,ah

pop si

pop bx

ret

tmp28:

mov byte ptr -100H[bp+si],24H

lea ax,-100H[bp]

push dx

mov dx,ax

mov ah,9

int 21H

pop dx

jmp near ptr tmp16

r\_noecho\_getchar\_:

mov ax,0cH

call near ptr \_\_STK

push bx

push cx

push dx

push si

push di

mov ah,8

int 21H

tmp29:

jmp near ptr tmp17

r\_getchar\_:

mov ax,0cH

call near ptr \_\_STK

push bx

push cx

push dx

push si

push di

mov ah,1

int 21H

jmp tmp29

r\_exit\_:

mov ax,0cH

call near ptr \_\_STK

push bx

push cx

push dx

push si

push di

mov ah,4cH

xor al,al

int 21H

jmp near ptr tmp17

echo\_:

push ax

mov ax,2

call near ptr \_\_STK

pop ax

call near ptr r\_print\_

mov ax,offset DGROUP:tmp58

r\_print\_:

push ax

mov ax,10eH

call near ptr \_\_STK

pop ax

push bx

push cx

push dx

push si

push di

push bp

mov bp,sp

sub sp,100H

mov dx,ax

call near ptr r\_strlen\_

xor si,si

tmp30:

cmp si,ax

jge tmp28

mov bx,dx

add bx,si

mov bl,byte ptr [bx]

mov byte ptr -100H[bp+si],bl

inc si

jmp tmp30

noecho\_readline\_unsafe\_:

push ax

mov ax,8

call near ptr \_\_STK

pop ax

push bx

push cx

push dx

mov cx,ax

xor dx,dx

tmp31:

call near ptr r\_noecho\_getchar\_

cmp al,0dH

je tmp32

cmp al,0aH

je tmp32

mov bx,cx

add bx,dx

mov byte ptr [bx],al

inc dx

jmp tmp31

tmp32:

mov bx,cx

add bx,dx

mov byte ptr [bx],0

jmp near ptr tmp18

readline\_unsafe\_:

push ax

mov ax,8

call near ptr \_\_STK

pop ax

push bx

push cx

push dx

mov cx,ax

xor dx,dx

tmp33:

call near ptr r\_getchar\_

cmp al,0dH

je tmp32

cmp al,0aH

je tmp32

mov bx,cx

add bx,dx

mov byte ptr [bx],al

inc dx

jmp tmp33

decrypt\_prices\_:

push ax

mov ax,8

call near ptr \_\_STK

pop ax

push bx

push cx

push dx

call near ptr r\_hash\_

mov cx,dx

mov dx,ax

xor ax,ax

tmp34:

mov bx,ax

shl bx,1

shl bx,1

xor word ptr \_input\_price[bx],dx

xor word ptr \_input\_price+2[bx],cx

xor word ptr \_output\_price[bx],dx

xor word ptr \_output\_price+2[bx],cx

inc ax

cmp ax,3

jl tmp34

jmp near ptr tmp18

login\_:

mov ax,2eH

call near ptr \_\_STK

push bx

push cx

push dx

push si

push di

push bp

mov bp,sp

sub sp,20H

mov cx,10H

mov ax,ds

mov es,ax

lea di,-20H[bp]

mov si,offset DGROUP:tmp77

rep movsw

mov ax,offset DGROUP:tmp59

call near ptr r\_print\_

lea ax,-20H[bp]

call near ptr readline\_unsafe\_

mov al,byte ptr -20H[bp]

test al,al

jne tmp35

xor ah,ah

jmp near ptr tmp16

tmp35:

cmp al,71H

jne tmp36

call near ptr r\_exit\_

tmp36:

mov dx,offset \_user

lea ax,-20H[bp]

call near ptr r\_streql\_

test ax,ax

jne tmp38

tmp37:

mov ax,offset DGROUP:tmp60

call near ptr echo\_

xor ax,ax

jmp near ptr tmp16

tmp38:

mov ax,offset DGROUP:tmp61

call near ptr r\_print\_

lea ax,-20H[bp]

call near ptr noecho\_readline\_unsafe\_

mov cx,word ptr \_IllIllIl0OoOO00o

mov bx,word ptr \_llIlIIlIO0O0ooO0

mov dx,offset \_IllIlIlIOoOoOo0o

lea ax,-20H[bp]

call near ptr fake\_verify\_

test ax,ax

je tmp37

mov ax,offset DGROUP:tmp62

call near ptr echo\_

lea ax,-20H[bp]

call near ptr decrypt\_prices\_

mov ax,offset DGROUP:tmp63

call near ptr echo\_

mov ax,1

jmp near ptr tmp16

profit\_mark\_of\_:

push ax

mov ax,1cH

call near ptr \_\_STK

pop ax

push bx

push bp

mov bp,sp

sub sp,16H

mov bx,ax

shl bx,1

shl bx,1

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

mov word ptr -0eH[bp],ax

mov ax,word ptr \_output\_price+2[bx]

mov word ptr -0cH[bp],ax

xor ax,ax

mov word ptr -0aH[bp],ax

mov word ptr -8[bp],ax

fild qword ptr -0eH[bp]

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

mov word ptr -0eH[bp],ax

mov ax,word ptr \_input\_price+2[bx]

mov word ptr -0cH[bp],ax

xor ax,ax

mov word ptr -0aH[bp],ax

mov word ptr -8[bp],ax

fild qword ptr -0eH[bp]

fdivp st(1),st

fadd dword ptr DGROUP:tmp64

fstp dword ptr -6[bp]

fldz

fcomp dword ptr -6[bp]

fstsw word ptr -2[bp]

nop

fwait

mov ax,word ptr -2[bp]

sahf

jbe tmp39

mov al,46H

jmp tmp43

tmp39:

fld dword ptr -6[bp]

fst qword ptr -16H[bp]

fcomp qword ptr DGROUP:tmp65

fstsw word ptr -2[bp]

nop

fwait

mov ax,word ptr -2[bp]

sahf

jbe tmp40

mov al,41H

jmp tmp43

tmp40:

fld qword ptr -16H[bp]

fcomp qword ptr DGROUP:tmp66

fstsw word ptr -2[bp]

nop

fwait

mov ax,word ptr -2[bp]

sahf

jbe tmp41

mov al,42H

jmp tmp43

tmp41:

fld qword ptr -16H[bp]

fcomp qword ptr DGROUP:tmp67

fstsw word ptr -2[bp]

nop

fwait

mov ax,word ptr -2[bp]

sahf

jbe tmp42

mov al,43H

jmp tmp43

tmp42:

mov al,44H

tmp43:

mov sp,bp

pop bp

pop bx

ret

show\_secret\_:

push ax

mov ax,2eH

call near ptr \_\_STK

pop ax

push bx

push cx

push dx

push si

push di

push bp

mov bp,sp

sub sp,20H

mov di,ax

mov si,0ffffH

xor bx,bx

mov ax,offset DGROUP:tmp68

call near ptr r\_print\_

lea ax,-20H[bp]

call near ptr readline\_unsafe\_

mov dx,offset DGROUP:tmp69

lea ax,-20H[bp]

call near ptr r\_streql\_

test ax,ax

je tmp44

call near ptr r\_exit\_

tmp44:

mov dx,offset DGROUP:tmp70

lea ax,-20H[bp]

call near ptr r\_streql\_

test ax,ax

je tmp45

jmp near ptr tmp16

tmp45:

mov cl,5

tmp46:

cmp bx,3

jge tmp48

mov dx,bx

shl dx,cl

add dx,offset \_products

lea ax,-20H[bp]

call near ptr r\_streql\_

test ax,ax

je tmp47

mov si,bx

jmp tmp48

tmp47:

inc bx

jmp tmp46

tmp48:

cmp si,0ffffH

jne tmp49

mov ax,offset DGROUP:tmp71

jmp tmp51

tmp49:

test di,di

je tmp50

mov ax,si

call near ptr profit\_mark\_of\_

mov byte ptr -20H[bp],al

mov byte ptr -1fH[bp],0

mov ax,offset DGROUP:tmp72

call near ptr r\_print\_

shl si,1

shl si,1

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

mov dx,word ptr \_input\_price+2[si]

call near ptr r\_print\_long\_

mov ax,offset DGROUP:tmp73

call near ptr r\_print\_

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

mov dx,word ptr \_output\_price+2[si]

call near ptr r\_print\_long\_

mov ax,offset DGROUP:tmp74

call near ptr r\_print\_

lea ax,-20H[bp]

jmp tmp51

tmp50:

lea ax,-20H[bp]

tmp51:

call near ptr echo\_

jmp near ptr tmp16

fake\_main\_:

mov ax,4

call near ptr \_\_STK

push dx

xor dx,dx

jmp tmp53

tmp52:

inc dx

cmp dx,3

jge tmp55

tmp53:

call near ptr login\_

test ax,ax

je tmp52

tmp54:

mov ax,1

call near ptr show\_secret\_

jmp tmp54

tmp55:

xor ax,ax

call near ptr show\_secret\_

jmp tmp55

\_TEXT ENDS

CONST SEGMENT WORD PUBLIC USE16 'DATA'

tmp56:

DB 0, 0e4H, 0bH, 54H, 2, 0, 0, 0

tmp57:

DB 0aH, 0, 0, 0, 0, 0, 0, 0

tmp58:

DB 0dH, 0aH, 0

tmp59:

DB 59H, 6fH, 75H, 72H, 20H, 6eH, 61H, 6dH

DB 65H, 20H, 70H, 6cH, 65H, 61H, 73H, 65H

DB 3aH, 20H, 0

tmp60:

DB 50H, 65H, 72H, 6dH, 69H, 73H, 73H, 69H

DB 6fH, 6eH, 20H, 64H, 65H, 6eH, 69H, 65H

DB 64H, 2eH, 0

tmp61:

DB 59H, 6fH, 75H, 72H, 20H, 70H, 61H, 73H

DB 73H, 77H, 6fH, 72H, 64H, 20H, 70H, 6cH

DB 65H, 61H, 73H, 65H, 3aH, 20H, 0

tmp62:

DB 50H, 61H, 73H, 73H, 65H, 64H, 2eH, 0

tmp63:

DB 49H, 6eH, 66H, 6fH, 72H, 6dH, 61H, 74H

DB 69H, 6fH, 6eH, 20H, 64H, 65H, 63H, 72H

DB 79H, 70H, 74H, 65H, 64H, 2eH, 0, 0

tmp64:

DB 0, 0, 80H, 0bfH, 0, 0, 0, 0

tmp65:

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

tmp66:

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

tmp67:

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

tmp68:

DB 57H, 68H, 61H, 74H, 20H, 70H, 72H, 6fH

DB 64H, 75H, 63H, 74H, 20H, 3fH, 20H, 0

tmp69:

DB 71H, 0

tmp70:

DB 0

tmp71:

DB 4eH, 6fH, 74H, 20H, 66H, 6fH, 75H, 6eH

DB 64H, 2eH, 0

tmp72:

DB 49H, 6eH, 70H, 75H, 74H, 20H, 70H, 72H

DB 69H, 63H, 65H, 20H, 0

tmp73:

DB 2cH, 20H, 6fH, 75H, 74H, 70H, 75H, 74H

DB 20H, 70H, 72H, 69H, 63H, 65H, 20H, 0

tmp74:

DB 2cH, 20H, 70H, 72H, 6fH, 66H, 69H, 74H

DB 20H, 72H, 61H, 74H, 65H, 20H, 0

CONST ENDS

CONST2 SEGMENT WORD PUBLIC USE16 'DATA'

CONST2 ENDS

\_DATA SEGMENT WORD PUBLIC USE16 'DATA'

\_user:

DB 62H, 65H, 6eH, 73H, 6fH, 6eH, 67H, 20H

DB 6cH, 69H, 75H, 0, 0, 0, 0, 0

DB 0, 0, 0, 0, 0, 0, 0, 0

DB 0, 0, 0, 0, 0, 0, 0, 0

\_IllIlIlIOoOoOo0o:

DB 0e2H, 0aH, 0, 0, 0d2H, 15H, 0, 0

DB 0e6H, 0fH, 0, 0, 0e2H, 0aH, 0, 0

DB 0efH, 15H, 0, 0, 55H, 4, 0, 0

DB 0e6H, 0fH, 0, 0, 0e6H, 0fH, 0, 0

DB 50H, 13H, 0, 0, 5fH, 7, 0, 0

DB 0beH, 4, 0, 0, 0e0H, 15H, 0, 0

DB 92H, 0dH, 0, 0, 98H, 0, 0, 0

DB 45H, 7, 0, 0, 0, 0, 0, 0

DB 07H DUP(0,0,0,0,0,0,0,0)

DB 0, 0, 0, 0, 0, 0, 0, 0

\_IllIllIl0OoOO00o:

DB 0dH, 16H, 19H, 1

\_llIlIIlIO0O0ooO0:

DB 0ffH, 0bbH, 1aH, 0faH

\_products:

DB 62H, 6fH, 6fH, 6bH, 0, 0, 0, 0

DB 0, 0, 0, 0, 0, 0, 0, 0

DB 0, 0, 0, 0, 0, 0, 0, 0

DB 0, 0, 0, 0, 0, 0, 0, 0

DB 70H, 65H, 6eH, 0, 0, 0, 0, 0

DB 0, 0, 0, 0, 0, 0, 0, 0

DB 0, 0, 0, 0, 0, 0, 0, 0

DB 0, 0, 0, 0, 0, 0, 0, 0

DB 73H, 63H, 61H, 6cH, 61H, 0, 0, 0

DB 0, 0, 0, 0, 0, 0, 0, 0

DB 0, 0, 0, 0, 0, 0, 0, 0

DB 0, 0, 0, 0, 0, 0, 0, 0

\_input\_price:

DB 0f3H, 0bdH, 76H, 1, 0fcH, 0e8H, 35H, 1

DB 0b3H, 3aH, 35H, 1

\_output\_price:

DB 0fH, 0b0H, 7, 1, 0a1H, 0c2H, 35H, 1

DB 16H, 0b5H, 35H, 1

tmp75:

DB 0, 0, 0

tmp76:

DB 0, 0, 0, 0, 0, 0, 0, 0

DB 0, 0, 0, 0

tmp77:

DB 0, 0, 0, 0, 0, 0, 0, 0

DB 0, 0, 0, 0, 0, 0, 0, 0

DB 0, 0, 0, 0, 0, 0, 0, 0

DB 0, 0, 0, 0, 0, 0, 0, 0

\_DATA ENDS

END

### 实验记录与分析

由于dosbox时钟周期的设置问题和dos架构的限制(segment等)，没能实现足够强的rsa加密(32层递归就爆栈，循环代替递归没时间实现了)，也没有蒙哥马利模乘法的实现(反汇编还能发现一个空壳子)，因此rsa公钥长度只有32bit，使用在线因式分解工具即可轻松破解，队友只需稍微了解rsa即可体验良好。

考虑到xor的不安全性(已知明文密文可以推断出密钥，明文到密文的映射分布不均匀，具有明显特征)，对正确的密码进行了一个自己随便写的hash(容易被碰撞，不符合加密密码所需的安全性要求，但用作xor的一项正合适)得到uint32\_t，和被加密数字异或拿出正确数字。由于密码本来就会被验证后进入系统，也就没有必要额外对数据设置校验和验证，错误的密码会导致permission denied而不是错误的输出。为了防止由密文直接推断出密钥的hash值的一部分然后暴力破解，明文尽可能多的填满了uint32\_t的位。这一系列操作保证了攻击者必须通过对rsa\_n进行因式分解来开始攻击。

## 任务4

### 实验步骤

拿到exe，先反汇编，观察源代码。一般的，防御者喜欢把用户名和加密过的密码放在一起。显然，.data:0000200附近存在嫌疑。.data:0000020a到.data:00000213很可能是密码。

然后，找到.code中所有对这一段内存的引用。这里使用了debugger(纯静态)，发现1154:010a-1154:0113对应刚才提到的密码嫌疑段。开始动态调试，途中发现有一段看起来和中断表有关的代码，直接跳过防止乱改东西出事。然后在输入密码之后，出现了意外。ob了一下它保存我输入的文本的缓冲区，却发现他紧接着把一个地址丢进了寄存器，这个地址就紧邻着刚才这个缓冲区，不知道什么时候出现的一段看起来是明文的文本出现了。重新看了一眼反汇编工具的数据段，里面确实没有明文。很奇怪，拿出来试一下，成功了。

没有3了。突然结束。

### 对方可执行文件

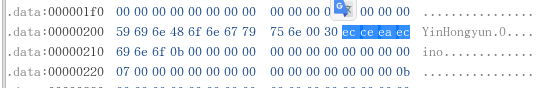
<https://recolic.net/tmp/origin.exe>

<https://cdn.rawgit.com/recolic/80x86-asm-learning/2349aeb2/tmp/origin.exe>

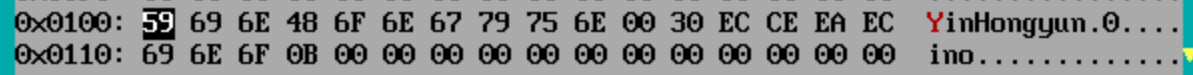
sha256sum: 11f1e517e8ee85844d7f042f8a829d8624f0002e75ecd8bb398ac9886aa37428

### 实验记录与分析

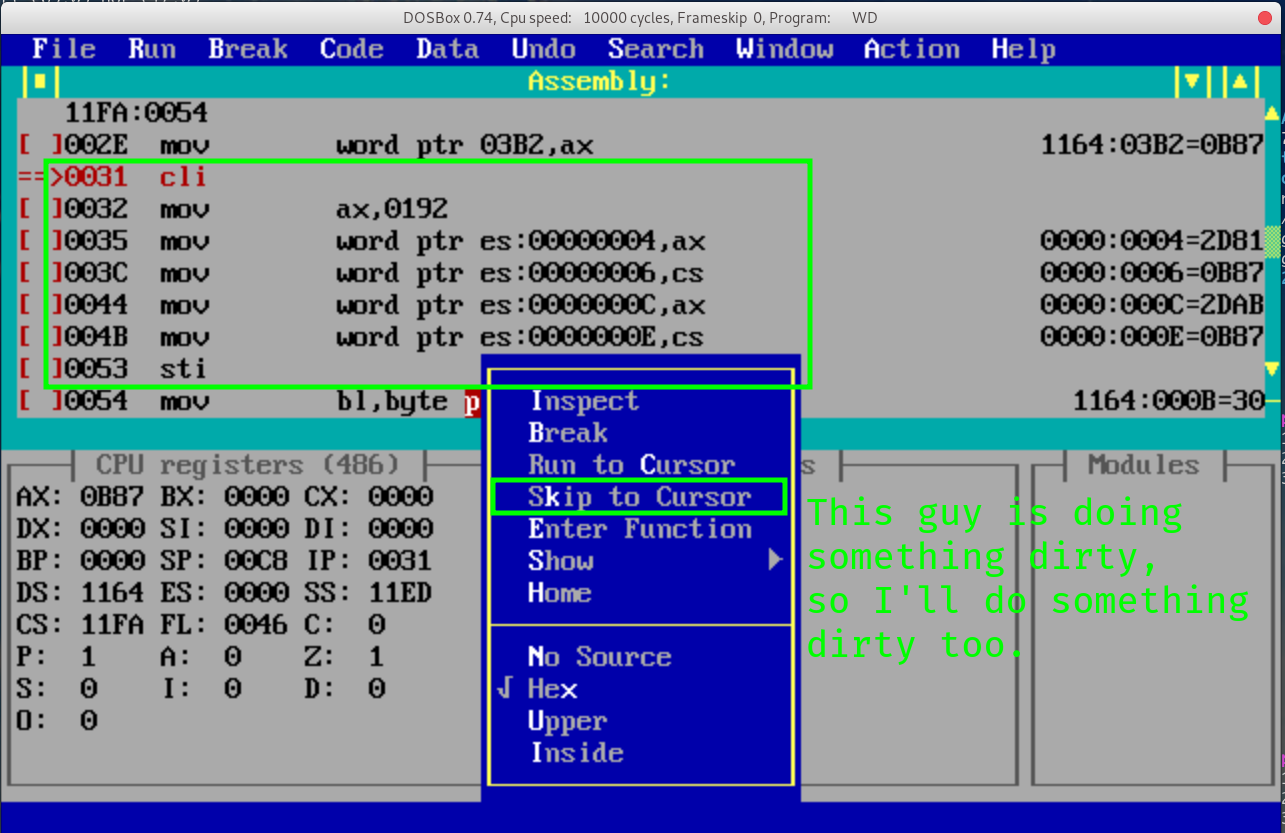
拿到exe，先反汇编，观察源代码。一般的，防御者喜欢把用户名和加密过的密码放在一起。显然，.data:0000200附近存在嫌疑。.data:0000020a到.data:00000213很可能是密码。



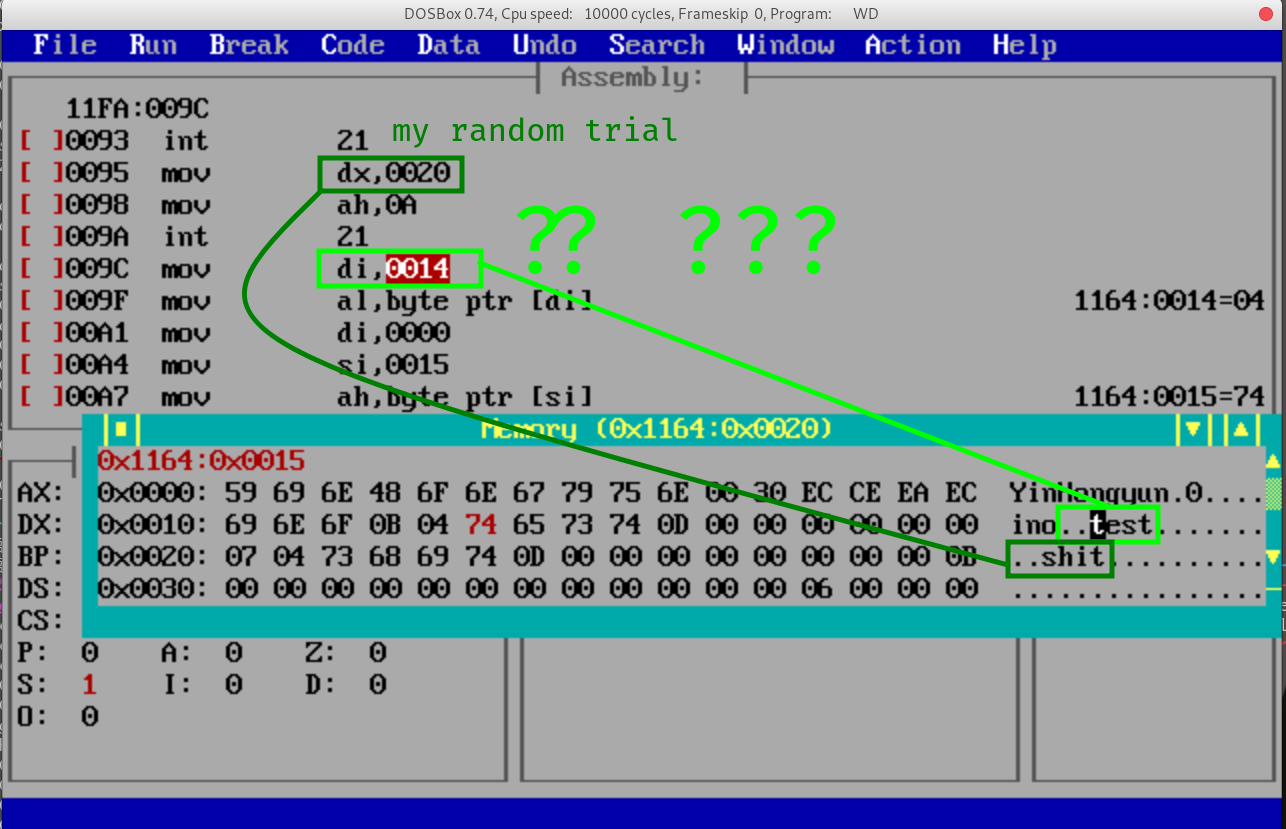
然后，找到.code中所有对这一段内存的引用。这里使用了debugger(纯静态)，发现1154:010a-1154:0113对应刚才提到的密码嫌疑段。



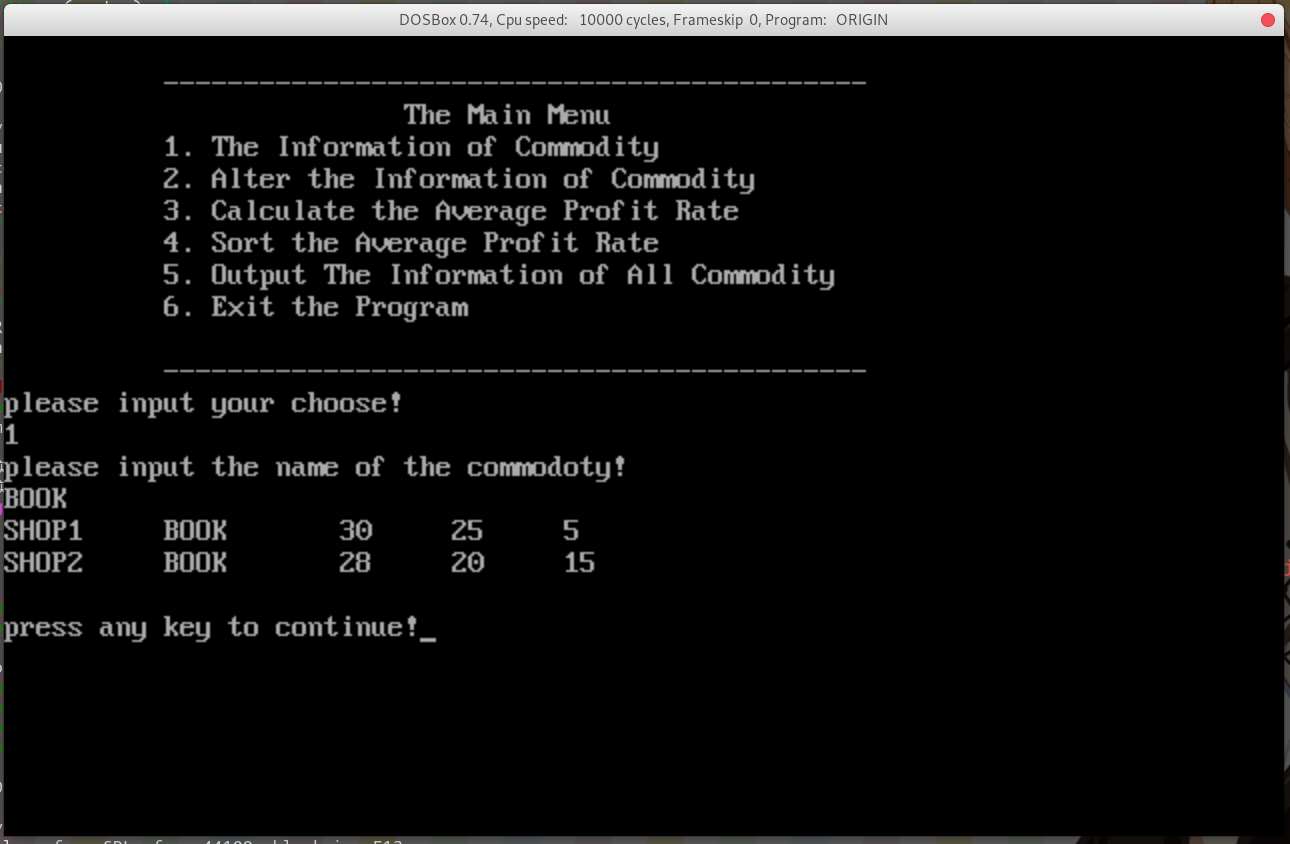
开始动态调试，途中发现有一段看起来和中断表有关的代码，直接跳过防止乱改东西出事。



然后在输入密码之后，出现了意外。ob了一下它保存我输入的文本的缓冲区，却发现他紧接着把一个地址丢进了寄存器，这个地址就紧邻着刚才这个缓冲区，不知道什么时候出现的一段看起来是明文的文本出现了。



重新看了一眼反汇编工具的数据段，里面确实没有明文。很奇怪，拿出来试一下，成功了。



很奇怪，居然没有花指令，大跳乱跳，软件数据预取，软件乱序执行，数据区打乱等基本混淆操作，实验过程心情很不错。

# 总结与体会

在本次实验中,我学会了 ivt 的相关知识和 cmos 的访问接口,并利用 ivt 的知识实现了自定义

中断处理程序。了解了 dos 中中断机制的处理方法,掌握了实现自定义中断处理程序的方法,增加

了利用汇编语言知识解决问题的能力。通过这次实验,我加深了对于这些知识点的了解,能够更加

熟练地在实践中运用已经学到的汇编知识,从底层发现程序的问题所在。能够更加熟练的利用汇编

语言解决问题。同时,我强化练习了 TD 的使用方式,能够更加熟练地使用 TD 反汇编并且调试程

序。

任务4中，实现了简单的加密和hash，实现了一个有明显漏洞的rsa。在任务5中，我追踪和解密了同学的代码，了解了基本的逆向工程方法。而实际上，在任务五中，我们也可以考虑暴力破解密码(虽然较慢)。这些都告诉了我们，没有绝对的安全，我们只能尽力保证，在一定的时间和空间内，我们的程序是相对安全的。在密码学的保证下，应当要求加密算法至少是图灵机难计算的，或者在未来要求加密算法是量子安全的。

# 参考文献

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

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

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

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

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

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

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

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

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

[10] Old Knowledge of x86 Architecture : “8086 Interrupt Mechanism” (https://sw0rdm4n.wordpress.com/category/interrupt-vector-table/)

[11] Wikepedia rsa(algorithm) `https://en.wikipedia.org/wiki/RSA\_(cryptosystem)`

[12] Openssl source code (https://www.openssl.org/)