

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实验**

**实验名称： 中断与反跟踪**

**实验时间： 2018-5-11，18:30-21:50 实验地点： 南一楼804室23号实验台**

**指导教师： 李海波 专业班级：计算机科学与技术201607班**

**学 号： U201614700 姓 名： 王亚宁**

**同组学生： 无 报告日期： 2018年5月11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**

[实验目的与要求 1](#_Toc513400479)

[实验内容 1](#_Toc513400480)

[实验过程 1](#_Toc513400481)

[任务1 1](#_Toc513400482)

[1. 设计思想及存储单元分配 1](#_Toc513400483)

[2. 流程图 1](#_Toc513400484)

[3. 源代码 3](#_Toc513400485)

[4. 实验步骤 4](#_Toc513400486)

[5. 实验记录 4](#_Toc513400487)

[任务2 6](#_Toc513400488)

[1. 设计思想及存储单元分配 6](#_Toc513400489)

[2. 流程图 6](#_Toc513400490)

[3. 源代码 6](#_Toc513400491)

[4. 实验步骤 7](#_Toc513400492)

[5. 实验记录 7](#_Toc513400493)

[任务3 8](#_Toc513400494)

[1. 设计思想及存储单元分配 8](#_Toc513400495)

[2. 流程图 9](#_Toc513400496)

[3. 源代码 9](#_Toc513400497)

[4. 实验步骤 11](#_Toc513400498)

[5. 实验记录 11](#_Toc513400499)

[体会 11](#_Toc513400500)

[参考文献 11](#_Toc513400501)

# 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1H 、10H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，要求在程序返回DOS操作系统后，输入键盘上的小写字母时都变成了大写字母。

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”，看是否都变成了大写。

（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（只需要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。

要求：

（1）在数据段定义一个待读取的CMOS内部单元的地址编号。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与操作系统显示的时间一致。

任务4：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。

任务5：解密同组同学的加密程序，获取各个商品的进货价。

# 实验过程

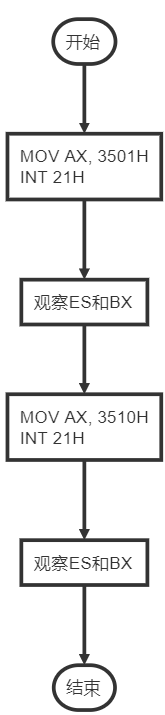
## 任务1

### 1. 设计思想及存储单元分配

使用3中方式读取中断类型码1H，10H对应的中断入口地址，由于题目已经给出了三种方法，在此就不再描述了。

### 2. 流程图

使用系统调用的流程图：



使用第三种方式的流程图：



### 3. 源代码

第二种方法的代码：

.386

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

DATA SEGMENT USE16

DATA ENDS

CODE SEGMENT USE16

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

START:

MOV AX, DATA

MOV DS, AX

; code after this

MOV AX, 3501H

INT 21H

MOV AX, 3510H

INT 21H

;stop here

MOV AH, 4CH

INT 21H

CODE ENDS

END START

第三种方法的代码：

.386

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT USE16

ASSUME CS:CODE, SS:STACK

; 获取中断物理地址的函数，参数AL

; 返回值AX, BX：中断的物理地址

GET\_INT PROC

XOR EBX, EBX

MOV BL, AL

XOR EAX, EAX

SHL BX, 2

PUSH DS

MOV AX, 0

MOV DS, AX

MOV AX, DS:[BX]

MOV BX, DS:[BX + 2]

POP DS

RET

GET\_INT ENDP

START:

; code after this

; 这个还行，直接读取

MOV AL, 1H

CALL GET\_INT

MOV AL, 10H

CALL GET\_INT

;stop here

MOV AH, 4CH

INT 21H

CODE ENDS

END START

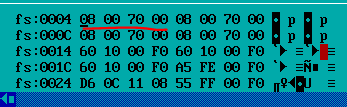
### 4. 实验步骤

（1）第一种方法直接进入td，在数据区寻找00:04H和00:40H即可找到1H中断和10H中断的地址。

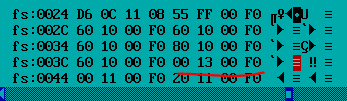
（2）第二种和第三种直接运行程序，查看返回值即可。

### 5. 实验记录

（1）第一种方式查看中断的方法，直接打开td，查看数据区，查看两个中断的地址如图：

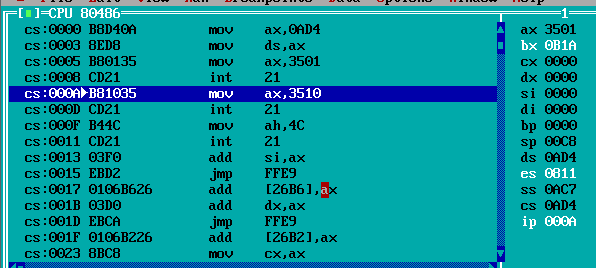


1H中断的处理的地址是0070H:0008H

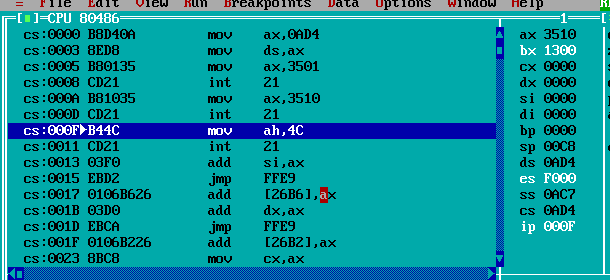


10H的中断处理地址是0F000H:1300H

（2）第二种方式结果的截图：

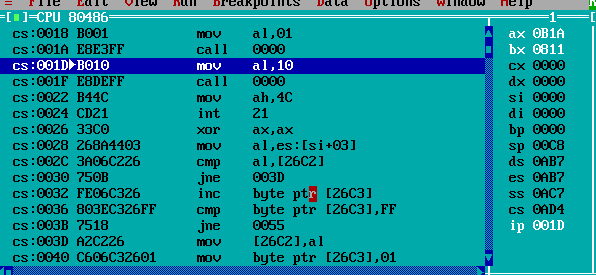


1H中断处理的地址是0811H:0B1AH

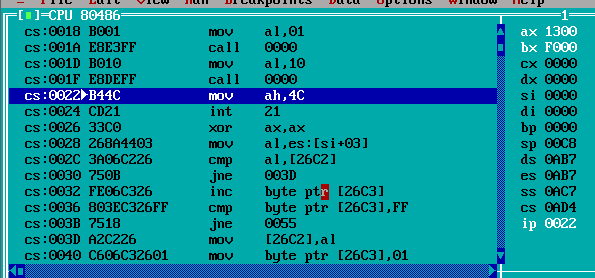


10H的中断处理地址是0F000:1300H

（3）第三种方式的结果：



1H的中断处理地址0811H:0B1AH



10H的中断处理地址是0F000H:1300H

## 任务2

### 1. 设计思想及存储单元分配

实验要求使用中断替换让程序能够将输入的小写字母转换为大写字母，其余不变。

设计思想：参考课本上的替换磁盘读取的那个程序，我的思想就是先保存旧的中断地址，使用自己的写的程序来替换掉旧中断，最后将代码驻留在内存中即可。

### 2. 流程图

### 3. 源代码

写中断的代码：

.386

CODE SEGMENT USE16

ASSUME CS:CODE, SS:STACK

OLD\_INT DW 0, 0

NEW16H PROC

PUSHF

CMP AH, 00H

JE GETKEY

CMP AH, 10H

JE GETKEY

; 调用旧中断

CALL DWORD PTR CS:OLD\_INT

JMP ENDINT

GETKEY:

CALL DWORD PTR OLD\_INT

; 获取了扫描码和ASCII码

; 判断大小写并转换

CMP AL, 'a'

JL ENDINT

CMP AL, 'z'

JG ENDINT

; 是小写字母

SUB AL, 32;变为大写

ENDINT:

IRET

NEW16H ENDP

START:

PUSH CS

POP DS

; code after this

MOV AX, 3516H

INT 21H

MOV OLD\_INT, BX

MOV OLD\_INT + 2, ES

; 替换中断

CLI;关闭中断响应

MOV DX, OFFSET NEW16H

MOV AX, 2516H

INT 21H

STI;开启中断

; 替换完毕，准备驻留内存

MOV DX, OFFSET START + 15

SHR DX, 4

ADD DX, 10H

MOV AL, 0

MOV AH, 31H

INT 21H

CODE ENDS

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

END START

恢复中断的代码：

.386

CODE SEGMENT USE16

ASSUME CS:CODE, SS:STACK

START:

; 替换中断

CLI;关闭中断响应

MOV AX, 0F000H;

MOV DS, AX;

; 直接将旧中断写回

MOV DX, 11E0H

MOV AX, 2516H

INT 21H

STI;开启中断

MOV AH, 4CH

INT 21H

CODE ENDS

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

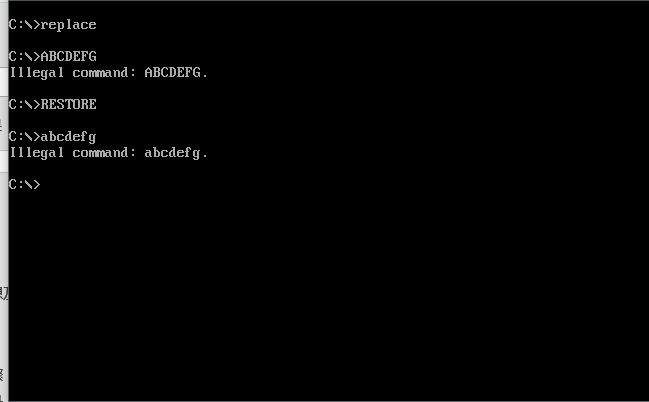
END START

### 4. 实验步骤

将程序编译连接得到可执行文件，直接执行文件即可将中断替换，想要恢复只需要执行恢复文件即可。

### 5. 实验记录

执行的记录如下：



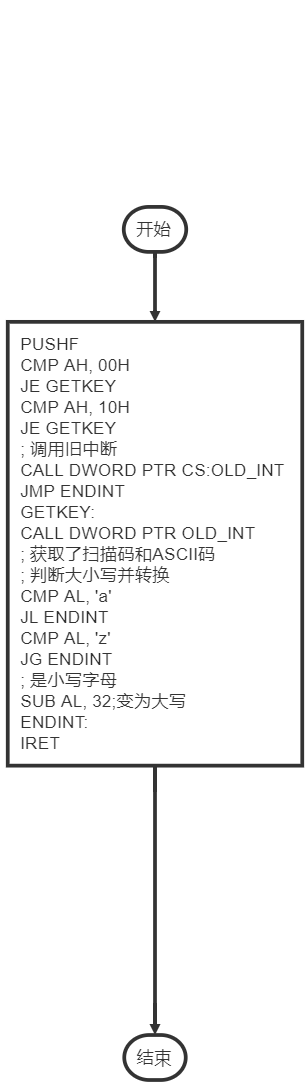
可见执行成功。

## 任务3

### 1. 设计思想及存储单元分配

实验要求从CMOS中读取数据并显示，我选择了最简单的显示时间，主要步骤就是向70H端口写入自己需要的数据号，再从71H中读取即可，其中需要注意的就是要注意只能读取一个byte的数据，包含了两个字符的BCD码，需要将它们分离开处理才能进行输出。

### 2. 流程图



### 3. 源代码

.386

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

DATA SEGMENT USE16

TIME DB '00:00:00$'

DATA ENDS

CODE SEGMENT USE16

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

PRINT\_TIME PROC

PUSH EAX

PUSH EDX

; hours

MOV AL, 4H

OUT 70H, AL

IN AL, 71H

MOV AH, AL

SHR AH, 4

AND AL, 0FH

ADD AH, '0'

ADD AL, '0'

MOV TIME[0], AH

MOV TIME[1], AL

; miuntes

MOV AL, 2

OUT 70H, AL

IN AL, 71H

MOV AH, AL

SHR AH, 4

AND AL, 0FH

ADD AH, '0'

ADD AL, '0'

MOV TIME[3], AH

MOV TIME[4], AL

; seconds

MOV AL, 0

OUT 70H, AL

IN AL, 71H

MOV AH, AL

SHR AH, 4

AND AL, 0FH

ADD AH, '0'

ADD AL, '0'

MOV TIME[6], AH

MOV TIME[7], AL

MOV DX, OFFSET TIME

MOV AH, 9

INT 21H

POP EDX

POP EAX

RET

PRINT\_TIME ENDP

START:

MOV AX, DATA

MOV DS, AX

; code after this

CALL PRINT\_TIME

;stop here

MOV AH, 4CH

INT 21H

CODE ENDS

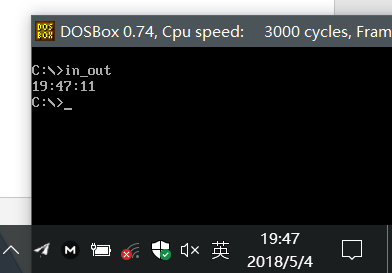
END START

### 4. 实验步骤

将代码编译连接得到可执行文件，执行程序就可以在屏幕上看到系统的时间。

### 5. 实验记录

执行程序如图：



对比可知获取的时间正确。

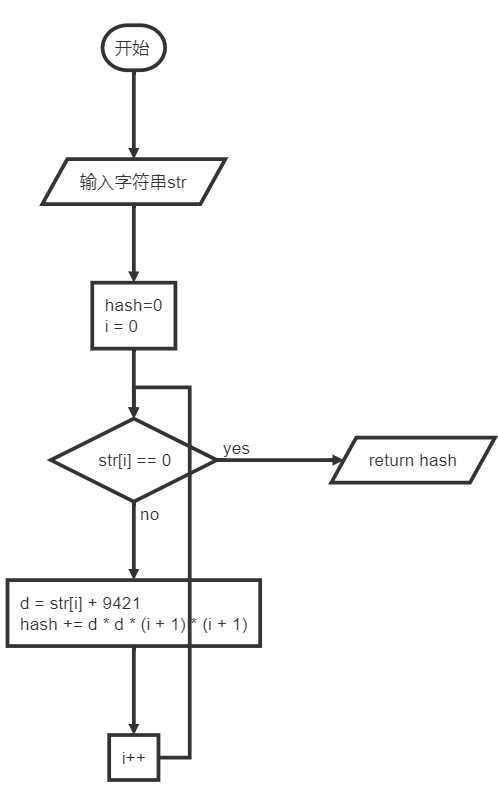
## 任务4

### 1. 设计思想及存储单元分配

本次实验的主要目的是对数据进行加密，其中包括密码和商品的信息，对密码的处理我选择了大多数网站的做法，就是使用加密函数将密码使用MD5或者SHA信息摘要将摘要放在文件中，这样可以减少文件大小也可以降低密码被盗的风险，而数据加密我是用密码的第一个字符对数据进行异或存储，这就意味着即使知道了密码的哈希值不知道密码的具体内容也是不能获取数据的。而且加密函数也不给的话想要破解就只有暴力破解，而暴力破解中及时哈希值相同也无法保证是同一个密码，安全性是可以保证的。

### 2. 流程图

使用c++获取字符串的哈希值的大致思路：



### 3. 源代码

.386

STACK SEGMENT USE16 STACK

DB 400 DUP(0)

STACK ENDS

DATA SEGMENT USE16

NOTICE\_NAME DB 'Please enter user name: $'

NOTICE\_PWD DB 'Please enter user password: $'

NOTICE\_GOODS DB 'Please enter goods name: $'

IN\_GOODS DB 20, ?, 20 DUP(0)

BNAME DD 3964361990; 用户名哈希值

BPASS DD 3970332365; 密码的哈希值

IN\_NAME DB 20, ?, 20 DUP(0)

IN\_PWD DB 20, ?, 20 DUP(0)

G\_CNT = 4

S1 DB 'SHOP1', 5 DUP(0)

BAG1 DB 'BAG', 7 DUP(0)

DW 12, 30, 1000, 5, ? ; 利润率未计算

G\_SIZE = $ - BAG1

GA1 DB 'PEN', 7 DUP(0)

DW 35 XOR 'p', 56 XOR 'p', 70 XOR 'p', 25 XOR 'p', ? ; 利润率未计算

GA2 DB 'BOOK', 6 DUP(0)

DW 12 XOR 'p', 30 XOR 'p', 25 XOR 'p', 5 XOR 'p', ? ; 利润率未计算

GAN DB 'TempValue',0

DW 15 XOR 'p', 20 XOR 'p', 30 XOR 'p', 2 XOR 'p', ?; 其他商品暂时未知

S\_SIZE = $ - S1

S2 DB 'SHOP2', 5 DUP(0) ;网店名称，用0结束

BAG2 DB 'BAG', 7 DUP(0)

DW 12 XOR 'p', 30 XOR 'p', 2000 XOR 'p', 2 XOR 'p', ? ; 利润率未计算

GB1 DB 'PEN', 7 DUP(0) ;商品名称

DW 35 XOR 'p', 50 XOR 'p', 30 XOR 'p', 24 XOR 'p', ? ;利润率还未计算

GB2 DB 'BOOK', 6 DUP(0) ; 商品名称

DW 12 XOR 'p', 28 XOR 'p', 20 XOR 'p', 15 XOR 'p', ? ;利润率还未计算

GBN DB 'TempValue',0

DW 15 XOR 'p', 20 XOR 'p', 30 XOR 'p', 2 XOR 'p', ?

AUTH DB 0; 标志是否通过验证

GG\_INDEX DD 2 DUP(0); 一共两家店,存放查询的物品的地址在内存中

G\_INDEX DD 0

SHIFTLINE DB 13, 10, '$'

S\_CNT = 2

S\_TEMP\_CNT DB 0

G\_TEMP\_CNT DB 0

SG\_INDEX DD 0

S\_INDEX DD 0

G\_COST DD 0

G\_PRO DD 0

TIME DD 0

; 菜单字符串

query DB "1. Query goods.$", 24 DUP(0); len 13

modify DB "2. Modify goods.$", 23 DUP(0); len 14

calcu\_pro DB "3. Calcu average profile rate.$", 9 DUP(0);len 28

sort\_pro DB "4. Sort the profile rate.$", 14 DUP(0); len 23

print\_all DB "5. Print all goods's information.$", 6 DUP(0); len 31

exit DB "6. Exit the program.$", 19 DUP(0); len 18

\_ADD DW 0

NUM DD 0

; 菜单函数中输入选项使用

STRLEN DW 0

; 输入商店名

NOTICE\_SHOP DB 'Please enter shop name: $'

IN\_SHOP DB 10, ?, 10 DUP(0)

; 地址存放，字符串比较使用

ADD1 DW 0

ADD2 DW 0

; 选项一的shop1、shop2

SHOP1\_STR DB 'SHOP1$'

SHOP2\_STR DB 'SHOP2$'

; 获取数字的字符串缓冲区

IN\_NUM DB 20, ?, 20 DUP(0)

; 选项二的地址参数

ADD3 DW 0

ADD4 DW 0

MYBP DW 0

; 输出提示

COST DB 'Buy cost: $'

PRICE DB 'Sale price: $'

BUY\_CNT DB 'Buy count: $'

; 排序函数中使用的地址数组

IDXES DW G\_CNT DUP(0)

MAX DW 0

RANK DW 0

; 输出全部信息函数使用的数据段

PRINT\_SHOP1 DB 'In shop1: $'

PRINT\_SHOP2 DB 'IN shop2: $'

PRINT\_BUY DB 'in price: $'

PRINT\_SELL DB 'sell price: $'

PRINT\_BUY\_CNT DB 'buy count: $'

PRINT\_SELL\_CNT DB 'sell count: $'

PRINT\_PRO DB 'profile: $'

PRINT\_RANK DB 'rank: $'

; print\_one数据区

PRINT\_ONE\_BP DW 0

NAME\_BP DW 0

NAME\_ADD DW 0

; 记录输入密码次数

ERR\_TIME DB 0

DATA ENDS

; 输出字符串的宏

; 参数：字符串偏移地址

; 返回值：无

WRITE MACRO A

PUSH AX

PUSH DX

MOV DX, A

MOV AH, 9

INT 21H

POP DX

POP AX

ENDM

; 读入字符串宏

; 参数：缓冲区偏移地址

; 返回值：无

READ MACRO A

PUSH DX

PUSH AX

MOV DX, A

MOV AH, 10

INT 21H

POP AX

POP DX

ENDM

; 换行宏

; 参数：无

; 返回值：无

CRLF MACRO

WRITE <OFFSET SHIFTLINE>

ENDM

; 输出逗号

; 参数：无

; 返回值：无

COMMA MACRO

PUSH DX

PUSH AX

MOV DL, ','

MOV AH, 2

INT 21H

POP AX

POP DX

ENDM

CODE SEGMENT USE16

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

START:

MOV AX, DATA

MOV DS, AX

; code after this

FUNC1: ; 功能1开始

WRITE <OFFSET NOTICE\_NAME>

READ <OFFSET IN\_NAME>

CRLF

WRITE <OFFSET NOTICE\_PWD>

READ <OFFSET IN\_PWD>

CRLF

FUNC2: ;功能2开始

; 用户名输入长度为0？

MOV AUTH, 0

; 长度为1，检测是否是q

CMP IN\_NAME[1], 1

; 不是1跳check\_name

JG CHECK\_NAME

CMP IN\_NAME[2], 'q'

; 输入为q，退出

JZ QUIT

CHECK\_NAME:

PUSH OFFSET IN\_NAME[2]

CALL GetHash

CMP EAX, BNAME

JE CK\_PASSWD

INC ERR\_TIME

CMP ERR\_TIME, 3

JE FUNC3

JMP FUNC1

CK\_PASSWD:

PUSH OFFSET IN\_PWD[2]

CALL GetHash

MOV AUTH, 1

CMP EAX, BPASS

JZ FUNC3

MOV AUTH, 0

INC ERR\_TIME

CMP ERR\_TIME, 3

JE FUNC3

JMP FUNC1

; 进入功能3

FUNC3:

; 显示菜单

CALL SHOW\_MENU

; 返回值是AL中的选项

; 合法就向下找对应的选项

CMP AL, 1

JE OPT\_1

CMP AL, 2

JE OPT\_2

CMP AL, 3

JE OPT\_3

CMP AL, 4

JE OPT\_4

CMP AL, 5

JE OPT\_5

CMP AL, 6

JE OPT\_6

JMP FUNC3

OPT\_1:

CALL QUERYP

JMP FUNC3

OPT\_2:

WRITE <OFFSET NOTICE\_SHOP>

READ <OFFSET IN\_SHOP>

CRLF

PUSH OFFSET IN\_SHOP[2]

WRITE <OFFSET NOTICE\_GOODS>

READ <OFFSET IN\_GOODS>

CRLF

PUSH OFFSET IN\_GOODS[2]

CALL MODIFYP

JMP FUNC3

OPT\_3:

CALL CALCU\_ALL

JMP FUNC3

OPT\_4:

CALL SORT\_PROP

JMP FUNC3

OPT\_5:

CALL PRINT\_ALLP

JMP FUNC3

OPT\_6:

JMP QUIT

QUIT:

;stop here

MOV AH, 4CH

INT 21H

; GetHash获取传入参数的哈希值

; 参数：字符串偏移地址

; 返回值：EAX字符串哈希值

GetHash PROC

PUSH BP

MOV BP, SP

PUSH EBX

PUSH ECX

PUSH EDX

MOV BX, [BP + 4]

ADD BX, 5

MOV ECX, 6

MOV EAX, 0

LOOP\_HASH:

PUSH EAX

XOR EDX, EDX

MOV DL, [BX]

ADD EDX, 9421

MOV EAX, EDX

MUL EDX

MUL ECX

MUL ECX

MOV EDX, EAX

POP EAX

ADD EAX, EDX

DEC BX

LOOP LOOP\_HASH

POP EDX

POP ECX

POP EBX

POP BP

RET 2; 地址占2个字节

GetHash ENDP

; 函数：获取时间戳，以ms为单位，

; 参数: 无

; 返回值：EAX:

GET\_MS PROC

PUSH EBX

PUSH ECX

PUSH EDX

MOV AH, 2CH

INT 21H

XOR EAX, EAX

XOR EBX, EBX

; 获取小时

MOV BL, CH

ADD EAX, EBX

;; 转为分钟

IMUL EAX, 60

; 获取分钟

MOV BL, CL

ADD EAX, EBX

; 转为s

IMUL EAX, 60

; 获取10ms

MOV BL, DH

ADD EAX, EBX

IMUL EAX, 100

MOV BL, DL

ADD EAX, EBX

; 转为ms

IMUL EAX, 10

POP EDX

POP ECX

POP EBX

RET

GET\_MS ENDP

; 输出函数：输出参数的十进制表示

; 参数：一个字的数值

; 返回值：无

; 注意：

PRINT\_NUM PROC

PUSH BP

MOV BP, SP

PUSH EAX

PUSH EDX

PUSH ECX

PUSH EBX

MOV AX, [BP + 4]

XOR AL, IN\_PWD[2]

MOVSX EAX, AX

MOV EBX, 10

XOR CX, CX

CDQ

CMP EAX, 0

JGE L2

PUSH EAX

PUSH EDX

; 输出负号，正数不输出

MOV DL, '-'

MOV AH, 2

INT 21H

POP EDX

POP EAX

NEG EAX

L2:

CDQ

IDIV EBX

ADD DL, '0'

PUSH DX

; 还原EDX的实际情况

INC CX

CMP EAX, 0

JG L2

; 栈里一共ECX个数字可用

L3:

POP DX

MOV AH, 2

INT 21H

LOOP L3

; 恢复环境

POP EBX

POP ECX

POP EDX

POP EAX

POP BP

RET 2

PRINT\_NUM ENDP

; 输出菜单函数

; 参数：无

; 返回值：输入的菜单选项，存放在AL中

SHOW\_MENU PROC

PUSH BX

PUSH CX

PUSH DX

PUSH DI

WRITE <OFFSET query>

CRLF

CMP AUTH, 1

JNZ STR6

MOV DI, OFFSET modify

MOV CX, 4

SHOW\_MENU\_LOOP1:

WRITE <DI>

CRLF

ADD DI, 40

LOOP SHOW\_MENU\_LOOP1

STR6:

WRITE <OFFSET exit>

CRLF

; 输入选项部分

CALL GET\_NUM

CRLF

;恢复环境

POP DI

POP DX

POP CX

POP BX

RET

SHOW\_MENU ENDP

; 字符串转数字函数

; 参数：字符串地址一个字，字符串长度一个字

; 返回值：转换为的数字，EAX中存放，溢出不管,错误返回-1

FSTRT2 PROC

POP BP

; 弹出参数

POP WORD PTR STRLEN

POP WORD PTR \_ADD

; 保存寄存器

PUSH EBX

PUSH ECX

PUSH EDX

PUSH DI

; 置零

XOR EAX, EAX

XOR EBX, EBX

XOR ECX, ECX

XOR EDX, EDX

; 开始计算

; 首先判断正负号

MOV DI, \_ADD

MOV CX, WORD PTR STRLEN

CMP BYTE PTR [DI], '-'

JNZ FSTRT2\_LOOP1

; 正负号是否占据字符位置

INC DI

DEC CX

FSTRT2\_LOOP1:

MOV BL, [DI]

SUB BL, '0'

CMP BL, 0

JL FSTRT2\_ERR

CMP BL, 9

JG FSTRT2\_ERR

; 拓展BX

MOVZX EBX, BL

IMUL EAX, 10

CDQ

ADD EAX, EBX

INC DI

LOOP FSTRT2\_LOOP1

SUB DI, STRLEN

; 判断正负，负责变号

CMP BYTE PTR [DI], '-'

JNZ FSTRT2\_EXIT

NEG EAX

JMP FSTRT2\_EXIT

FSTRT2\_ERR:

MOV EAX, -1

FSTRT2\_EXIT:

POP DI

POP EDX

POP ECX

POP EBX

PUSH BP

RET

FSTRT2 ENDP

; 查询商品函数

; 参数：无（参数在此函数中输入）

; 返回值：无（直接在此函数中输出）

QUERYP PROC

; 需要使用的寄存器先压栈保存环境

PUSH EAX

PUSH EBX

PUSH ECX

PUSH EDX

PUSH DI

; 输出提示

WRITE <OFFSET NOTICE\_GOODS>

; 输入商品名

READ <OFFSET IN\_GOODS>

CRLF

; 开始在商店一寻找商品

MOV CX, G\_CNT

MOV DI, OFFSET S1[10]

XOR EBX, EBX

MOV BL, IN\_GOODS[1]

; EBX移动到输入名下一个位置

ADD EBX, OFFSET IN\_GOODS[2]

; 给它加个尾巴

MOV BYTE PTR [EBX], '$'

QUERY\_LOOP1:

PUSH DI

PUSH OFFSET IN\_GOODS[2]

CALL CMP\_STR

; 查看输入是否与已有商品名相同

CMP AL, 0

JZ END\_QUERY\_LOOP1

; 转移到下一个物品

ADD DI, G\_SIZE

LOOP QUERY\_LOOP1

JMP NO\_GOODS

END\_QUERY\_LOOP1:

; 存在商品就好办了

; DI此时存放的是商品在第一个商店中的位置

; 输出商品的信息“SHOP1，商品名称，销售价，进货总数，

;已售数量”顺序显示该商品的信息，同时还要将“SHOP2”中该商品的信息也显示出来。

WRITE <OFFSET SHOP1\_STR>

COMMA

WRITE <OFFSET IN\_GOODS[2]>

COMMA

; 得出销售价

PUSH WORD PTR [DI + 12]

CALL PRINT\_NUM

COMMA

; 进货总数

PUSH WORD PTR [DI + 14]

CALL PRINT\_NUM

COMMA

; 已售总量

PUSH WORD PTR [DI + 16]

CALL PRINT\_NUM

CRLF

; 显示商店2中方物品信息

ADD DI, S\_SIZE

WRITE <OFFSET SHOP2\_STR>

COMMA

WRITE <OFFSET IN\_GOODS[2]>

COMMA

; 得出销售价

PUSH WORD PTR [DI + 12]

CALL PRINT\_NUM

COMMA

; 进货总数

PUSH WORD PTR [DI + 14]

CALL PRINT\_NUM

COMMA

; 已售总量

PUSH WORD PTR [DI + 16]

CALL PRINT\_NUM

CRLF

; 没有此商品

NO\_GOODS:

; 函数结束

POP EAX

POP EBX

POP ECX

POP EDX

POP DI

RET

QUERYP ENDP

; 修改商店物品的信息

; 参数：商店名和物品名字符串所在地址

; 返回值，无

MODIFYP PROC

; 首先获取参数

POP MYBP

; ADD3是第一个参数，是商品名吧

POP WORD PTR ADD3

POP WORD PTR ADD4

PUSH EAX

PUSH EBX

PUSH ECX

PUSH DI

; 首先比较商店

MOV DI, OFFSET S1

PUSH DI

PUSH ADD4

CALL CMP\_STR

; 是这个shop进入下环节

CMP AL, 0

JZ FIND\_GOODS

MOV DI, OFFSET S2

PUSH DI

PUSH ADD4

CALL CMP\_STR

CMP AL, 0

; 进入下环节

JZ FIND\_GOODS

; 没进入下环节说明没有此商店

JMP NO\_SHOP

; 寻找商品

FIND\_GOODS:

; 哪个商店？在DI中，不需要关心

; 移动到第一个物品处

ADD DI, 10

; 开始寻找物品名

; 将商品数量输入到CX中

MOV CX, G\_CNT

LOOP\_GOODS:

PUSH DI

PUSH WORD PTR ADD3

; 比较字符串是否相同

CALL CMP\_STR

CMP AL, 0

JZ FOUND\_GOOD

; 移动到下一个物品处

ADD DI, G\_SIZE

LOOP LOOP\_GOODS

JMP NO\_GOODS\_1

FOUND\_GOOD:

; 改变进货价

CHG\_COST:

WRITE <OFFSET COST>

PUSH WORD PTR [DI + 10]

CALL PRINT\_NUM

; 逗号分隔开

COMMA

; 尝试获取数字

READ <OFFSET IN\_NUM>

CRLF

CMP BYTE PTR IN\_NUM[1], 0

JE CHG\_PRICE

PUSH OFFSET IN\_NUM[2]

MOV AL, IN\_NUM[1]

MOV AH, 0

PUSH AX

CALL FSTRT2

CMP AX, -1

JE CHG\_COST

; 将得到的数字的低两字节放入位置

XOR AL, IN\_PWD[2]

MOV [DI + 10], AX

CHG\_PRICE:

;同等逻辑

WRITE <OFFSET PRICE>

PUSH WORD PTR [DI + 12]

CALL PRINT\_NUM

COMMA

; 尝试获取数字

READ <OFFSET IN\_NUM>

CRLF

CMP IN\_NUM[1], 0

JE CHG\_CNT

PUSH OFFSET IN\_NUM[2]

MOV AL, IN\_NUM[1]

MOV AH, 0

PUSH AX

CALL FSTRT2

CMP AX, -1

JE CHG\_PRICE

XOR AL, IN\_PWD[2]

MOV [DI + 12], AX

;; 修改进货数量

CHG\_CNT:

WRITE <OFFSET BUY\_CNT>

PUSH WORD PTR [DI + 14]

CALL PRINT\_NUM

COMMA

; 尝试获取数字

READ <OFFSET IN\_NUM>

CRLF

CMP BYTE PTR IN\_NUM[1], 0

JE NO\_GOODS\_1

PUSH OFFSET IN\_NUM[2]

MOV AL, IN\_NUM[1]

MOV AH, 0

PUSH AX

CALL FSTRT2

CMP AX, -1

JE CHG\_CNT

XOR AL, IN\_PWD[2]

MOV [DI + 14], AX

NO\_GOODS\_1:

NO\_SHOP:

; 结束

POP DI

POP ECX

POP EBX

POP EAX

PUSH MYBP

RET

MODIFYP ENDP

; 函数：计算一个商品的平均利润率

; 参数：第一个商店里的物品的偏移地址

; 返回值：利润率AX

CALCUG\_PRO PROC

POP BP

POP WORD PTR \_ADD

PUSH EBX

PUSH ECX

PUSH EDX

PUSH SI

MOV DWORD PTR G\_PRO, 0

MOV SI, WORD PTR \_ADD

MOV CX, S\_CNT

L1:

PUSH DWORD PTR G\_PRO

XOR EAX, EAX

XOR EBX, EBX

XOR EDX, EDX

MOV AX, [SI + 10]

MOV BX, [SI + 14]

XOR AL, IN\_PWD[2]

XOR BL, IN\_PWD[2]

IMUL AX, BX

; 80x86低位在前

MOVSX EAX, AX; 拓展AX符号位

MOV G\_COST, EAX

MOV AX, [SI + 12]

MOV BX, [SI + 16]

XOR AL, IN\_PWD[2]

XOR BL, IN\_PWD[2]

IMUL AX, BX

MOVSX EAX, AX

MOV G\_PRO, EAX

MOV EAX, G\_PRO

MOV EBX, G\_COST

SUB EAX, EBX

IMUL EAX, 100

CDQ

IDIV EBX; 得出结果

POP DWORD PTR G\_PRO

ADD G\_PRO, EAX

ADD SI, S\_SIZE

LOOP L1

; 恢复现场并返回

MOV EAX, G\_PRO; AX为低位

CDQ

MOV EBX, S\_CNT

IDIV EBX

POP SI

POP EDX

POP ECX

POP EBX

PUSH BP

RET

CALCUG\_PRO ENDP

; 获取所有商品平均利润率函数

; 参数：无

; 返回值：无

CALCU\_ALL PROC

PUSH AX

PUSH DI

MOV CX, G\_CNT

MOV DI, OFFSET S1[10]

CALCU\_ALL\_LOOP1:

PUSH DI

CALL CALCUG\_PRO

MOV [DI + 18], AX

ADD DI, G\_SIZE

LOOP CALCU\_ALL\_LOOP1

POP DI

POP AX

RET

CALCU\_ALL ENDP

; 平均利润率排名函数

; 我选择使用选择排序

; 参数：无

; 返回值：无

SORT\_PROP PROC

PUSH EAX

PUSH EBX

PUSH ECX

PUSH EDX

PUSH DI

PUSH SI

CALL CALCU\_ALL

MOV CX, G\_CNT

MOV BX, OFFSET IDXES

MOV DI, OFFSET S1[10]

; 获取所有的商品偏移地址

GET\_IDX\_LOOP:

MOV [BX], DI

ADD DI, G\_SIZE

; 每次后移两字节

ADD BX, 2

LOOP GET\_IDX\_LOOP

; 获取地址完毕，对地址对应商品排序

; 通过对利润率排名，将排名结果放到shop2对应的利润字段

; RANK存储排名, 排名从1开始吧

MOV RANK, 1

; CX控制外层循环

MOV CX, G\_CNT

; BX指向第一个物品偏移地址的地址

MOV BX, OFFSET IDXES

SORT\_OUT\_LOOP:

PUSH CX

PUSH BX

; AX存放已知最大利润率

MOV AX, -1

; DI获取商品地址

MOV DI, [BX]

; 获取此商品利润率

MOV AX, [DI + 18]

MOV MAX, BX

; 需要从BX下一个物品处开始循环

; CX现在的数值就是剩余的商品数量加1

SORT\_IN\_LOOP:

DEC CX

CMP CX, 0

JE NOT\_BIG

ADD BX, 2

MOV DI, [BX]

MOV DX, [DI + 18]

CMP DX, AX

; 此商品比最大的利润值小到NOT\_BIG

JLE NOT\_BIG

; 否则此商品就是最大的，

MOV AX, DX

; 获取地址在数组中的地址

MOV MAX, BX

NOT\_BIG:

CMP CX, 0

JG SORT\_IN\_LOOP

; 结束，交换

POP BX

; 将最大的与现在的下标所属元素交换

MOV AX, [BX]

MOV DI, MAX

MOV DI, [DI]

MOV [BX], DI

MOV DI, MAX

MOV [DI], AX

MOV DI, [BX]

MOV AX, RANK

INC WORD PTR RANK

ADD DI, S\_SIZE

MOV [DI + 18], AX

ADD BX, 2

POP CX

LOOP SORT\_OUT\_LOOP

POP SI

POP DI

POP EDX

POP ECX

POP EBX

POP EAX

RET

SORT\_PROP ENDP

; 输出全部信息函数

; 参数：无

; 返回值：无

; 将SHOP1和SHOP2中的所有商品信息显示到屏幕上，包括平均利润率和排名（替代了商品原有的利润率字段）。

;具体的显示格式自行定义（可以分网店显示，也可以按照商品排名显示，等等，显示方式可以作为子程序的入口参数）。

PRINT\_ALLP PROC

PUSH EBX

PUSH ECX

PUSH DI

; 输出好办啊

; 商品数量

MOV CX, G\_CNT

; BX指向shop1中商品

; DI指向shop2中商品

MOV BX, OFFSET S1[10]

MOV DI, OFFSET S2[10]

; 开始循环吧

PRINT\_ALL\_LOOP1:

PUSH BX

PUSH DI

CALL PRINT\_ONE

ADD BX, G\_SIZE

ADD DI, G\_SIZE

LOOP PRINT\_ALL\_LOOP1

POP DI

POP ECX

POP EBX

RET

PRINT\_ALLP ENDP

; 输出一个商品

; 参数：商品在两个商店中的偏移

; 返回值：无

PRINT\_ONE PROC

POP PRINT\_ONE\_BP

POP ADD1

POP ADD2

PUSH EAX

PUSH EBX

PUSH DI

MOV BX, ADD2

MOV DI, ADD1

PUSH BX

CALL PRINT\_NAME

CRLF

WRITE <OFFSET PRINT\_SHOP1>

WRITE <OFFSET PRINT\_BUY>

PUSH WORD PTR [BX + 10]

CALL PRINT\_NUM

COMMA

WRITE <OFFSET PRINT\_SELL>

PUSH WORD PTR [BX + 12]

CALL PRINT\_NUM

COMMA

WRITE <OFFSET PRINT\_BUY\_CNT>

PUSH WORD PTR [BX + 14]

CALL PRINT\_NUM

COMMA

WRITE <OFFSET PRINT\_SELL\_CNT>

PUSH WORD PTR [BX + 16]

CALL PRINT\_NUM

CRLF

WRITE <OFFSET PRINT\_SHOP2>

WRITE <OFFSET PRINT\_BUY>

PUSH WORD PTR [DI + 10]

CALL PRINT\_NUM

COMMA

WRITE <OFFSET PRINT\_SELL>

PUSH WORD PTR [DI + 12]

CALL PRINT\_NUM

COMMA

WRITE <OFFSET PRINT\_BUY\_CNT>

PUSH WORD PTR [DI + 14]

CALL PRINT\_NUM

COMMA

WRITE <OFFSET PRINT\_SELL\_CNT>

PUSH WORD PTR [DI + 16]

CALL PRINT\_NUM

CRLF

; 输出利润率

WRITE <OFFSET PRINT\_PRO>

MOV AX, WORD PTR [BX + 18]

XOR AL, IN\_PWD[2]

PUSH AX

CALL PRINT\_NUM

COMMA

; 输出排名

WRITE <OFFSET PRINT\_RANK>

MOV AX, WORD PTR [DI + 18]

XOR AL, IN\_PWD[2]

PUSH AX

CALL PRINT\_NUM

CRLF

POP DI

POP EBX

POP EAX

PUSH PRINT\_ONE\_BP

RET

PRINT\_ONE ENDP

; 比较两个字符串

; 参数：两个字符串的偏移地址

; 返回值AL: 1:不想等，0：相等

; 以第一个字符串结尾为准，调用时注意

CMP\_STR PROC

POP BP

POP WORD PTR ADD1

POP WORD PTR ADD2

PUSH EBX

PUSH ECX

PUSH EDX

PUSH DI

XOR EBX, EBX

XOR EDX, EDX

MOV BX, ADD2

MOV DX, ADD1

MOV AL, 1

CMP\_STR\_LOOP1:

MOV CH, [EBX]

MOV CL, [EDX]

CMP CH, CL

JNZ END\_CMP\_STR

INC EBX

INC EDX

CMP BYTE PTR [EBX], 0

JNZ CMP\_STR\_LOOP1

MOV AL, 0

END\_CMP\_STR:

POP DI

POP EDX

POP ECX

POP EBX

PUSH BP

RET

CMP\_STR ENDP

; 获取一个数字

; 参数无

; 返回值：输入的数字EAX，数字过大将会被截断

; 注意：请求失败返回-1，所以返回-1有可能是输入的-1也肯能是请求失败

GET\_NUM PROC

READ <OFFSET IN\_NUM>

PUSH OFFSET IN\_NUM[2]

MOV AH, 0

MOV AL, IN\_NUM[1]

PUSH AX

; 解析交给函数去做

CALL FSTRT2

RET

GET\_NUM ENDP

; 用来显示程序中以0结尾的字符串，可以用来显示商品名和商店名

; 参数：字单位的地址

; 返回值：无

PRINT\_NAME PROC

POP NAME\_BP

POP NAME\_ADD

PUSH AX

PUSH BX

PUSH DX

MOV BX, NAME\_ADD

NAME\_LOOP:

MOV DL, [BX]

MOV AH, 2

INT 21H

INC BX

CMP BYTE PTR [BX], 0

JNZ NAME\_LOOP

POP DX

POP BX

POP AX

PUSH NAME\_BP

RET

PRINT\_NAME ENDP

CODE ENDS

END START

计算哈希值的c++源码

#include <iostream>

#include <vector>

#include <cstring>

using namespace std;

unsigned int get\_hash(const char \*str) {

unsigned int hash = 0;

int len = strlen(str);

for (int i = 0; i < len; ++i) {

int d = str[i] + 9421;

hash += d \* d \* (i \* i + 2 \* i + 1);

}

return hash;

}

int main() {

unsigned int hash = 0;

char name[] = "yaning";

char passwd[] = "passwd";

cout << get\_hash(name) << "\n";

cout << get\_hash(passwd) << "\n";

return 0;

}

### 4. 实验步骤

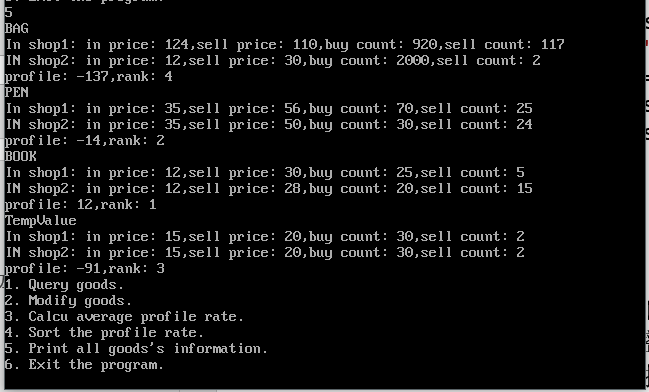
1. 思考如何加密自己的密码，我选择了使用信息摘要算法，这种算法是不可逆的，只能使用暴力枚举才有可能得到密码。

2. 加密自己的数据，使用密码的首字母对数据进行异或操作，这样即使得到和密码哈希值一样的字符串也不可能得到正确的数据。

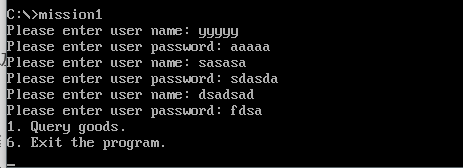
### 5. 实验记录

下面以排序和输出为例：

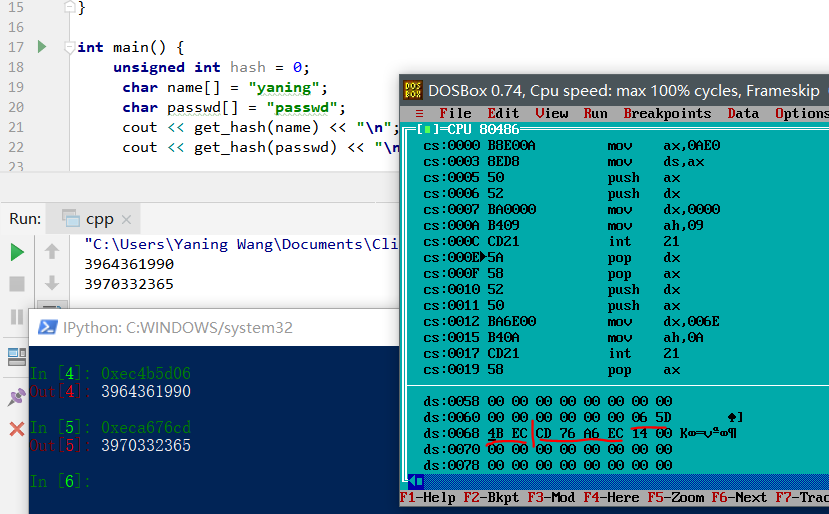
1. 输入正确的用户名和密码时，计算结果如下：



2. 而当你输入错误的用户名和密码时：



3. 在td中查看程序的数据区：



可知此处存放的时我的用户名哈希值和密码的哈希值。

## 任务5

### 1. 设计思想及存储单元分配

略

### 2. 流程图

略

### 3. 源代码

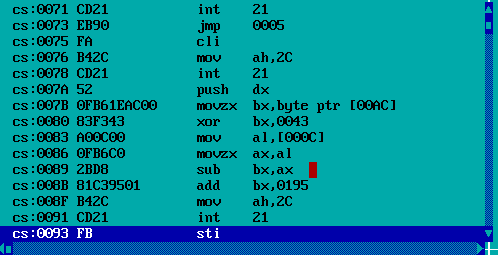
略

### 4. 实验步骤

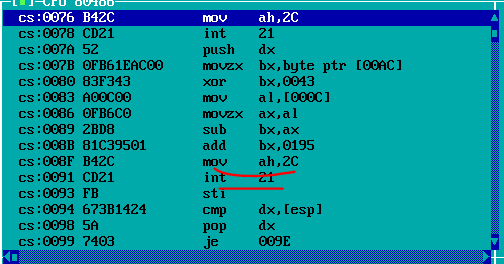
使用td加载可执行文件，单步执行代码，查看代码的执行情况。

### 5. 实验记录

执行过程中我输入错误的用户名和密码会导致程序卡死，这可能是程序的bug，无法继续执行，也有可能是保护数据的一种方式，在执行过程中我发现了其中存在开关中断响应的操作：



在其中对bx进行了一系列操作得到了一个变换后的数值，程序还使用了计时功能，



我认为应该是测试程序的运行时间，如果执行时间过长则直接跳转到程序结束。在下面的获取商品信息的步骤里也使用了密码的数据，没有密码是不能得到商品数据的。

而我的跟踪也在开中断后继续不下去了，可见同学反跟踪还是很难的。

# 体会

本次实验让我理解了dos中的中断的具体物理地址的查找，如何替换已有中断，还有如何恢复中断，也学习了如何从CMOS中读入数据，对计算机的运行有了更深的理解。

在对数据加密的学习中我学会了使用信息摘要对密码或者用户名进行加密的方式，这种方式能够有效防止攻击者通过反汇编获取密码的相关信息。而对同学的程序进行单步执行时也发现很难继续下去，因为其中的很多跳转仅仅是一个比较就会进行，在没有上下文的情况下很难理解这些跳转会跳到哪里去，对跟踪造成很大困难。这次的加密和反加密也让我意识到了代码可读性的重要性，虽然高级语言的代码已经很容易让人读懂，但是如果代码编写的逻辑混乱的话旁人也是很难理解的，重视代码可读性有助于提高自己的团队合作能力。

# 参考文献

[1] 王元珍等.80x86汇编语言程序设计.版本(第1版)

[2] 王晓虹等.汇编语言程序设计教程.版本(第1版)