

日期：2017.4.27

## 目录

1	实验目的与要求.....	1
2	实验内容.....	1
3	实验过程.....	2
3.1	任务 1 .....	2
3.1.1	源代码 .....	2
3.1.2	实验步骤 .....	3
3.1.3	实验记录与分析 .....	3
3.2	任务 2 .....	5
3.2.1	程序设计思想 .....	5
3.2.2	流程图 .....	5
3.2.3	源程序录入 .....	6
3.2.4	实验步骤 .....	7
3.2.5	实验记录与分析 .....	8
3.3	任务 3 .....	9
3.3.1	源代码 .....	9
3.3.2	实验步骤。 .....	10
3.3.3	实验记录与分析 .....	10
3.4	任务四 .....	11
3.4.1	程序设计思想 .....	11
3.4.2	流程图 .....	12
3.4.3	源程序录入 .....	13
3.4.4	实验步骤 .....	21
3.4.5	实验记录与分析 .....	22
3.5	任务五 .....	24
3.5.1	实验步骤 .....	24
3.5.2	实验记录与分析 .....	25
4	总结与体会.....	28
	参考文献 .....	29

# 汇编语言程序设计实验报告

---

## 1 实验目的与要求

本次实验的主要目的与要求有下面 5 点，所有的任务都会围绕这 5 点进行，希望大家事后检查自己是否达到这些目的与要求。

- (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) 提升对计算机的理解和分析能力

## 2 实验内容

**任务 1：**用三种方式获取中断类型码 19H、21H 对应的中断处理程序的入口地址。

要求：首先进入虚拟机状态，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 查看出口参数即可）。
- (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，键盘上的小写字母都变成了大写字母。

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。

要求：

- (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。

# 汇编语言程序设计实验报告

## 任务 4: 数据加密与反跟踪

在实验一的学生成绩查询程序的基础上，增加查询前输入密码的功能，密码不对则程序退出，只有密码正确之后才能完成后续的功能。密码采用密文的方式存放在数据段中。各科成绩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（建议只采用一到两种反跟踪方法，重点是深入理解和运用好所选择的反跟踪方法）。成绩表中要有编程者自己的名字（姓的全拼+名字的拼音首字母，但大小写可以随意组合）和各科成绩（姓名和成绩都密文存放）。成绩表中只需要定义三个学生的信息即可。

## 任务 5: 跟踪与数据解密

解密同组同学的加密程序，获取该同学的各科成绩。

## 3 实验过程

### 3.1 任务 1

#### 3.1.1 源代码

DOS 系统功能调用	直接读取内存单元
<pre>.386 STACK  SEGMENT USE16 STACK     DB 200 DUP(0) STACK  ENDS  CODE   SEGMENT USE16     ASSUME CS:CODE,DS:DATA,SS:STACK,ES:DATA START:  MOV AX,DATA         MOV DS,AX          MOV AL,21H         MOV AH,35H         INT 21H      ;查看 ES:[BX]          MOV AH,4CH         INT 21H CODE   ENDS         END START</pre>	<pre>.386 STACK  SEGMENT USE16 STACK     DB 200 DUP(0) STACK  ENDS  CODE   SEGMENT USE16     ASSUME CS:CODE,DS:DATA,SS:STACK,ES:DATA START:  MOV AX,0         MOV DS,AX         MOV AX,DS:[0084H]         MOV AH,4CH         INT 21H          MOV AH,4CH         INT 21H CODE   ENDS         END START</pre>

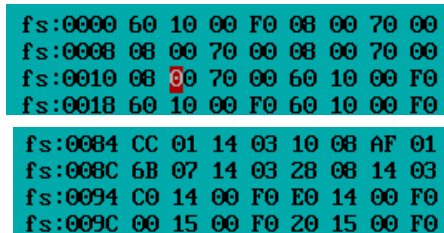
# 汇编语言程序设计实验报告

## 3.1.2 实验步骤

1. 准备上机实验环境，使用以下版本：
  - a) DosBox 虚拟环境: 0.74
  - b) Masm 编译器 6.08
  - c) TD
  - d) Vscode 编辑器
2. 直接运行调试工具 TD.EXE，观察中断矢量表的信息
  - a) 在 DosBox 中打开 td
  - b) 在数据区使用鼠标右击，点击 GoTo
  - c) 输入 00: 00，或其他中断矢量的地址
  - d) 记录观察到的中断矢量表信息
3. 编写程序，用 DOS 系统功能调用方式获取、观察对应功能调用相应的出口参数与（1）中结果是否相同。
  - a) 使用 td，在 td 中直接编写程序 int nH 调用 n 号功能
  - b) 使用 ALT + F7 进入 int 中断程序，观察 CS: IP 的值，与（1）比较
4. 编写程序，直接读取相应内存单元，观察结果数据与（1）（2）中是否相同
  - a) 编写程序，将 00: 00 处（或者相应的 N 号调用所在内存单元）读入 ax, bx
  - b) 在 td 中执行程序，观察 ax, bx 的值
  - c) 记录，并且与（1）（2）对比
5. 使用 TD 将中断矢量表中常用的中随意改成其他的值会发生什么现象？
  - a) 使用 TD 修改中断矢量表中的值
  - b) 使用中断调用调用被修改的中断矢量
  - c) 观察记录现象

## 3.1.3 实验记录与分析

1. 准备环境
2. 直接运行调试工具 TD.EXE，观察中断矢量表的信息



fs:0000	60 10 00 F0 08 00 70 00
fs:0008	08 00 70 00 08 00 70 00
fs:0010	08 00 70 00 60 10 00 F0
fs:0018	60 10 00 F0 60 10 00 F0
fs:0084	CC 01 14 03 10 08 AF 01
fs:008C	6B 07 14 03 28 08 14 03
fs:0094	C0 14 00 F0 E0 14 00 F0
fs:009C	00 15 00 F0 20 15 00 F0

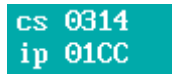
图 1-1 直接观察矢量表

3. 编写程序，用 DOS 系统功能调用方式获取、观察对应功能调用相应的出口参数与（1）中结果是否相同。

# 汇编语言程序设计实验报告

---

观察 int 21H



A screenshot of assembly code showing the values of CS and IP registers. The text is displayed in a green box with white text.

cs	0314
ip	01CC

图 1-2 进入前观察 CS:IP

4. 编写程序，直接读取相应内存单元，观察结果数据与（1）（2）中是否相同  
这时候遇到了一个问题，观察到的中断矢量并不是前两次得到的，在询问老师之后，明白了这是由于 TD 保护了 21 号调用的结果，所以只要更换中断号即可。
5. 使用 TD 将中断矢量表中常用的中随意改成其他的值会发生什么现象？
  - a) 使用 TD 修改中断矢量表中的值  
Td 中会保护中断矢量，自己的修改并不会生效

## 3.2 任务 2

### 3.2.1 程序设计思想

为了实现这一功能，我们首先要对计算机是如何进行键盘读写有所了解。

首先我们明白 CPU 是通过端口来与外设进行联系的。键盘的输入随时都可以到达，当键盘按下需要处理时，cpu 如何及时得到通知并处理？很明显需要使用外部中断。

当键盘的输入达到相关端口时，芯片向 CPU 发出中断类型码为 9 的可屏蔽中断信息，CPU 检测到信息后，若  $IF = 1$  相应中断，引发中断过程，转去执行 int 9 的中断例程。

Int 9 的中断例程：读出相关端口的扫描码；如果是字符键，直接送入 BIOS 键盘缓存区，如果是控制键，转变为状态字节后写入内存中状态字节的单元；进行一些对键盘系统的控制。

BIOS 键盘缓存区是系统启动后，BIOS 存放 int 9 中断例程所接收的键盘输入的内存区，可以存储 15 个键盘输入（每个一个字）：高位扫描码，低位字符码。

还有一个中断调用是 int 16H，功能编号为 0 的功能是从键盘缓存区读取一个键盘输入，并将其从缓冲区删除。结果中 ah = 扫描码，al = ASCII 码。调用如下：检测键盘缓存区是否有数据；没有继续检测；读取缓冲区第一个字单元的键盘输入，送入 ah,al 中；将已读入的键盘输入从缓存区删除。

可见完整的键盘输入历程是 int 9H 和 int 16H 相互配合的程序。

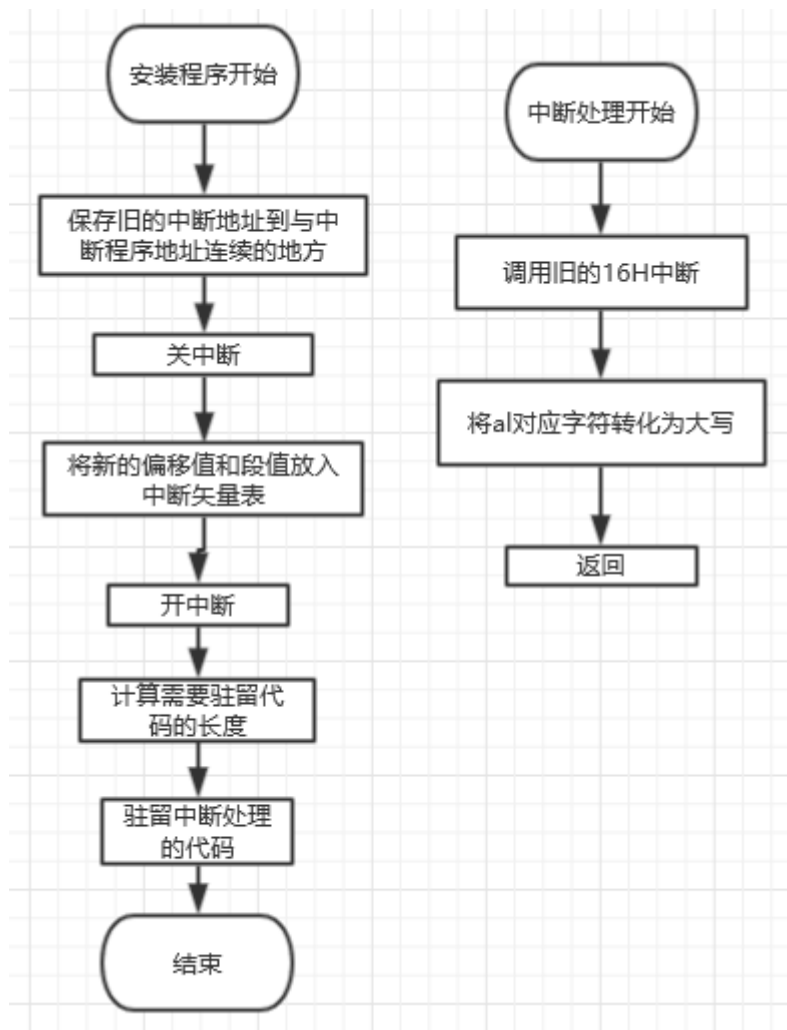
程序设计思想也由上述储备知识而来：

由于 9 号调用需要管理键盘这个外设的端口，修改 16H 号中断更加方便。程序分为下面几步进行：

首先，将旧的中断矢量表中 16H 的信息保存，然后，写新的 16H 中断程序，实现这些功能，调用旧的 16H 程序，得到结果的 AL 变成相应大写字母，然后返回。最后将上述程序装入内存中。

### 3.2.2 流程图

# 汇编语言程序设计实验报告



## 3.2.3 源程序录入

```
.386
code segment use16
    assume cs:code, ss:stack
old_int dw ?,?

NEW_INT:
    cmp ah, 00h
    jz OLD
    cmp ah, 10h
    jz OLD
    jmp dword ptr old_int
    jmp ee
OLD:
    pushf
    call dword ptr old_int
    cmp al, 'z'
    jb ee
    cmp al, 'a'
```



## 汇编语言程序设计实验报告

---

```
        ja ee
        sub al, 20h
ee:
        iret

start:
        xor ax, ax
        mov ds, ax
        mov ax, ds:[16H * 4]
        mov old_int, ax
        mov ax, ds:[16H * 4 + 2]
        mov old_int + 2, ax

        cli
        mov word ptr ds:[16H * 4], offset NEW_INT
        mov ds:[16H * 4 + 2], cs
        sti

        mov dx, offset start + 15
        shr dx, 4
        add dx, 10h

        mov al, 0
        mov ah, 31h
        int 21h
code ends
stack segment stack use16
        db 200 dup(0)
stack ends
        end start
```

### 3.2.4 实验步骤

1. 准备上机环境，与任务一相同
2. 使用两种调用原中断的方法调用：
  - a) 使用 `call` 调用
  - b) 使用 `jmp` 调用，这时需要先对 `CS,IP` 进行入栈操作
3. 为避免错误的程序接管键盘中断时导致键盘操作失灵，需要先用其他方法进行调试改程序：
  - a) 为了调试改程序，首先不要将中断处理程序装入中断矢量表中，可以先在其他程序中调用此函数
  - b) 若成功，执行装入内存的程序
4. 执行中断驻留程序后程序能否正常返回 `DOS`? `DOS` 是否正常工作?
5. 测试自己修改的中断矢量对新开的 `DOSBOX` 有无影响

## 汇编语言程序设计实验报告

---

- a) 新打开一个 DosBox，测试输入小写是否会被转化成大写
6. 实现卸载自己的中断处理程序，将其复原
  - a) 使用 TD 调试，调用 19H 时候，记录入口地址
  - b) 编写程序，得到入口地址对应的内存单元前一个单元所存内容
  - c) 上一个得到的内容就是原中断处理程序的入口地址，将其重新送回中断矢量表中即可。

### 3.2.5 实验记录与分析

1. 准备上机环境，与任务一相同
2. 使用两种调用原中断的方法调用：

CALL 和 JMP 两种方式进入原中断程序，采用 CALL 进入原中断程序后，IRET 返回到 CALL 指令的下一条指令。而 JMP 进入原中断程序和新中断程序 IRET 返回的地方是一样的
3. 为避免错误的程序接管键盘中断时导致键盘操作失灵，需要先用其他方法进行调试改程序：
  - a) 为了调试改程序，首先不要将中断处理程序装入中断矢量表中，可以先在其他程序中调用此函数
  - b) 若成功，再执行装入内存的程序
4. 执行中断驻留程序后程序能否正常返回 DOS?DOS 是否正常工作?

可以正常返回，DOSBox 也可以正常工作
5. 测试自己修改的中断矢量对新开的 DOSBOX 有无影响
  - a) 新打开一个 DosBox，测试输入小写是否会被转化成大写  
发现并没有影响
6. 实现卸载自己的中断处理程序，将其复原
  - a) 使用 TD 调试，调用 19H 时候，记录入口地址
  - b) 得到入口地址对应的内存单元前一个单元所存内容
  - c) 上一个得到的内容就是原中断处理程序的入口地址，将其重新送回中断矢量表中即可。

# 汇编语言程序设计实验报告

---

## 3.3 任务 3

### 3.3.1 源代码

```
.386
STACK SEGMENT USE16 STACK
    DB 200 DUP(0)
STACK ENDS

DATA SEGMENT USE16
BUF DB 0
DATA ENDS

CODE SEGMENT USE16
    ASSUME CS:CODE,DS:DATA,SS:STACK
START:
    MOV AX,DATA
    MOV DS,AX
IO      MACRO A,B
    MOV DL,A
    MOV AH,B
    INT 21H
ENDM

LP1: IO  0,1
    CMP AL,'Q'
    JE OVER
    CMP AL,'q'
    JE OVER
    SUB AL,30H
    OUT 70H,AL
    IN AL,71H

    MOV AH,AL
    MOV CL,4
    SHR AH,CL
    AND AL,00001111B

    ADD AX,3030H
    MOV CX,AX
    IO ':',2
    IO CH,2
    IO CL,2
    IO 0AH,2
    IO 0DH,2
    JMP LP1
OVER:  MOV AH,4CH
```

# 汇编语言程序设计实验报告

---

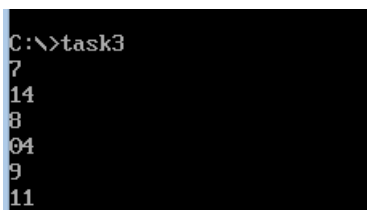
```
INT 21H
CODE ENDS
END START
```

## 3.3.2 实验步骤。

1. 准备上机实验环境。
2. 对原先输入待读取的 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
3. 编译程序，修改错误后重新编译，直至不再报错。
4. 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。

## 3.3.3 实验记录与分析

1. CMOS 内部信息以 BCD 码的形式保存，所以需要将 BCD 码先转换成二进制串，再将二进制串以十六进制字符输出。
2. 编译并运行程序，程序能正常运行并且能正确地输出时间信息。为了使输出显示更加明显，在程序中添加输出换行和回车符的语句。
3. 但是运行结果却没有输出换行和回车。在 TD 中查看，发现输出换行回车的语句缺失。原因可能是执行 ml 时没有重新 masm 源文件，而只是将原来的 obj 文件生成 exe 文件。但是在其他人的电脑上直接执行 ml 是可以将 masm 和 link 相继执行的。可能是在不同电脑配置下运行有差异。
4. 重新 masm 和 link 后，输入和输出以及格式都正确，结果如图 3-4 所示。（7 号代表日期，8 号代表月份，9 号代表年份，输出为 16 进制字符，所以结果分别为 20，4，17。与预期一致。）



```
C:\>task3
7
14
8
04
9
11
```

图 3-1 测试

## 3.4 任务四

### 3.4.1 程序设计思想

本次任务实现的实验的主要流程是这样的：首先进入程序，要求输入密码，若密码正确，则可以继续输入姓名查询成绩，若输入密码错误则程序退出。为了防止别人使用反汇编等手段破解我们的密码，我们需要加入反追踪程序。我们是在实验一的基础上进行的本次实验，在实验一中已经实现的部分不需要重复实现。所以本任务的主要内容可以看成两个部分：① 数据加密；② 反跟踪

首先看数据加密部分，采用不公开的算法加密，这种加密算法实现简单，而破解起来也需要一定时间和毅力，足以完成本实验的任务。程序采用三重加密方式加密密钥，第一步，进行栅栏加密，即：我们将输入分为  $n$  组，然后进行组合，例如：输入 `xjsdhr`，可以分为两组：`xjs`;`dhr`，然后进行组合，变成 `xdjhsr`。第一重加密结束后第二重加密开始，使用凯撒加密，按照这样的公式来进行  $(x - 'c') \bmod 26 + 'a'$ ，即将每个字母映射到一个其他的字母。这样可以使得原密码面目全非。最后一重加密选择进行异或操作，相邻两个字母进行异或，得到结果。得到的结果与我数据段中定义的密文进行比对，比对相同，则输入密码正确。在数据段中定义的密文采用链表的方式进行存储，可以避免密码全部在一个地方造成容易被猜到。

其次便实现反跟踪部分，使用检查修改的中断矢量表、检查堆栈、这两种方法来对抗动态反跟踪调试，使用间接转移对抗静态反跟踪调试，还加入很多无用代码混淆视线。

寄存器使用：

使用：`cl/al` 进行接受输入字符的数目，`bp` 堆栈指示器

第一步加密：`dx` 为使用间接转移所用到的寄存器，`di / si` 为当前所便利到的字符（因为分了两组，`ah` 作为转移两个内存地址中值所用到的中间寄存器

第二步加密：`ax`，`dx` 执行计算相关，`si` 为遍历指示器，`cl` 为计数器

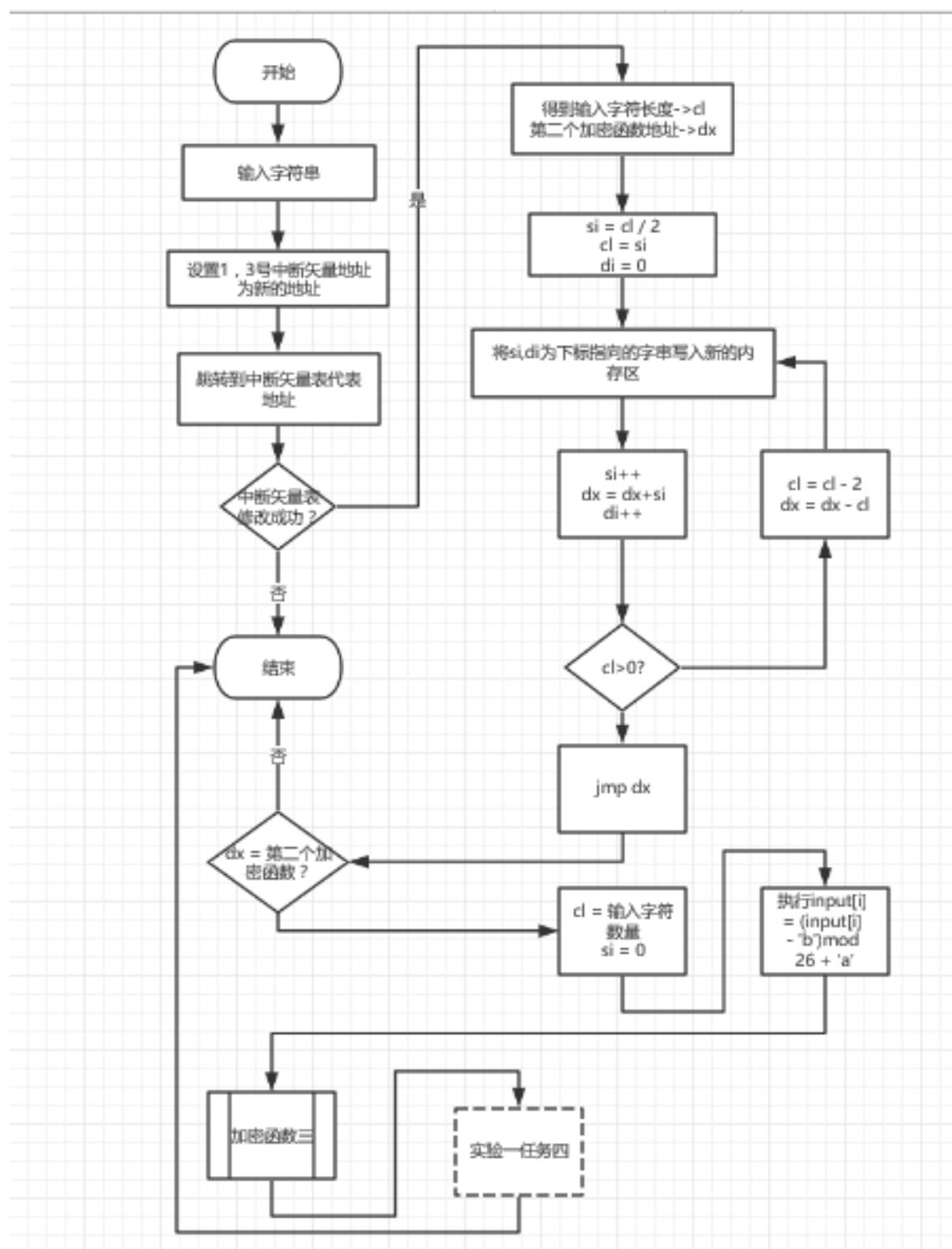
第三步加密：`al, dl` 作为计算单元，`si` 为遍历指示器，`cl` 为计数器

内存链表设置：（部分）

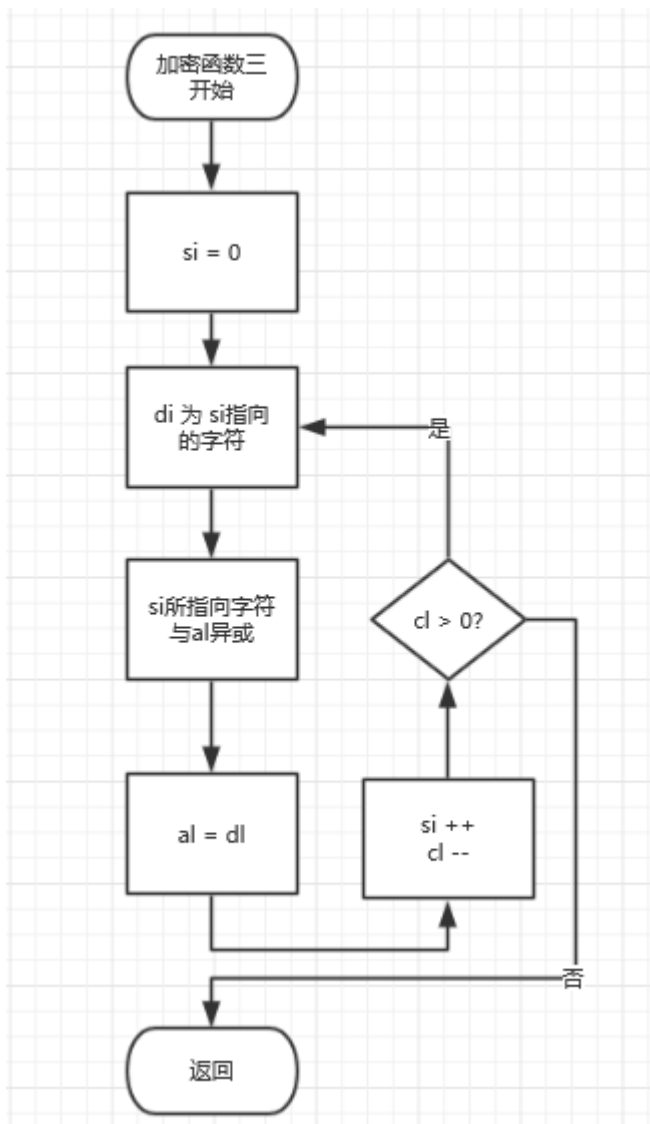
```
pwd1    db 'j' xor 0  pwd2    db 'j' xor 'q'
         dw pwd2         dw pwd3
```

# 汇编语言程序设计实验报告

## 3.4.2 流程图



## 汇编语言程序设计实验报告



### 3.4.3 源程序录入

```
;.386
STACK SEGMENT ;USE16 STACK
    DB 300 DUP(0)
STACK ENDS

DATA SEGMENT; USE16

space_0 db 'You want to slove my program? No possible$'

pwd1    db 'j' xor 0
        dw pwd2

space_1 db 'Oh, fuck hacker fuck you$'

N       EQU 3
```

# 汇编语言程序设计实验报告

---

```
POIN    DW    0

pwd2    db 'j' xor 'q'
        dw pwd3

BUF      DB   ('l' - 30h) * 2, ('i' - 30h) * 2, ('p' - 30h) * 2, ('e' - 30h) * 2, ('i' - 30h) * 2
        DB   ('h' - 30h) * 2, ('a' - 30h) * 2, ('o' - 30h) * 2, 2   DUP((0 - 30h) * 2)
        DB   10 xor 'B', 10 xor 'B', 10 xor 'B', ?
        DB   ('l' - 30h) * 2, ('i' - 30h) * 2, ('s' - 30h) * 2, ('i' - 30h) * 2
        DB   6   DUP((0 - 30h) * 2)
        DB   10 xor 'B', 10 xor 'B', 10 xor 'B', ?
        DB   ('x' - 30h) * 2, ('i' - 30h) * 2, ('n' - 30h) * 2, ('j' - 30h) * 2, ('i' - 30h) * 2
        DB   ('e' - 30h) * 2, 4   DUP((0 - 30h) * 2)
        DB   100 xor 'B', 100 xor 'B', 100 xor 'B', ?

pwd3     db 'n' xor 'q'
        dw pwd4

IN_NAME DB   11
        DB   0
        DB   11 DUP(0)

pwd4     db 'q' xor 'n'
        dw pwd5

CRLF     DB   0DH, 0AH, '$'

pwd5     db 'f' xor 'q'
        dw pwd6

MSG1     DB   0AH, 0DH, 'Please Input Your Name :', '$'
MSG2     DB   0AH, 0DH, 'Not Find This Student!:$'

pwd6     db 'z' xor 'f'
        dw pwd7

MSGA     DB   0AH, 0DH, 'A!$'
MSGB     DB   0AH, 0DH, 'B!$'
MSGC     DB   0AH, 0DH, 'C!$'
MSGD     DB   0AH, 0DH, 'D!$'

pwd7     db 'Hope you have a good time! ^_^ $'
        dw 0

;input key
in_pwd   db 10
        db ?
        db 10 dup(0)
change   db 10 dup(0)
```

---



# 汇编语言程序设计实验报告

---

```
MSG3      db 'Please give me the key to prove you are owner', 0ah, 0dh
          db 'If you input the WRONG Key, exit' , 0ah, 0dh
          db '$'
```

;地址表，用于间接转移反跟踪

```
P1      dw first_encry
E1      dw DIE
P2      dw second_encry
P3      dw third_encry
E2      dw 10 dup(DIE)
PA      dw SET_AVERAGE_GRADE
E3      dw 10 dup(DIE)
P4      dw new_int
```

;中断矢量反追踪

```
old_int_1 dw 0, 0
```

```
old_int_3 dw 0, 0
```

DATA ENDS

CODE SEGMENT ;USE16

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

START:

```
    MOV AX, DATA
    MOV DS, AX
```

```
    lea dx, MSG3
```

```
    mov ah, 9
```

```
    int 21h
```

```
    xor ax, ax
```

```
    mov es, ax
```

```
    mov ax, es:[1 * 4]
```

```
    mov old_int_1, ax
```

```
    mov ax, es:[1 * 4 + 2]
```

```
    mov old_int_1 + 2, ax
```

```
    mov ax, es:[3 * 4]
```

```
    mov old_int_3, ax
```

```
    mov ax, es:[3 * 4 + 2]
```

```
    mov old_int_3 + 2, ax
```

```
    cli
```

```
    mov ax, offset new_int
```

```
    mov es:[1 * 4], ax
```

```
    mov es:[1 * 4 + 2], cs
```

```
    mov es:[3 * 4], ax
```

# 汇编语言程序设计实验报告

---

```
mov es:[3 * 4 + 2], cs
sti
```

```
lea dx, in_pwd ;input key
mov ah, 10
int 21h
```

judge\_key:

```
lea bx, in_pwd
push bx
mov al, [bx + 1]
shr al, 1
mov ah, 0
mov si, ax
mov di, 0
```

```
mov bx, es:[1 * 4]
inc bx
jmp bx
;jmp x
```

second\_encry:

```
mov al, [change + si]
sub al, 63h
jns su
add al, 26
su:
mov ah, 0
mov bl, 26
div bl
add ah, 61h
mov [change + si], ah
```

```
inc si
dec cl
jnz second_encry
```

```
pop bx
mov cl, [bx + 1]
mov si, 0
mov dx, 0
mov al, 0
jmp third_encry
```

new\_int: iret

;x:

```
pop bx
mov dx, [P2]
```

first\_encry:

---

# 汇编语言程序设计实验报告

---

```
mov ah, [bx + di + 2]
shl di, 1
mov [change + di], ah
mov ah, [bx + si + 2]
mov [change + di + 1], ah
xor ah, ah
shr di, 1

cli
push P1
inc di
add dx, di
inc si
sub dx, ax
pop cx
mov bp, sp
mov cx, [bp - 2]
sti

dec al
jz ooo
jmp cx

ooo:
mov cl, [bx + 1]
mov si, 0
push bx
jmp dx
db 'Your will die, goto die!$'

third_encry:
mov dl, [change + si]
nop
cmp ax, bx
xor [change + si], al
mov al, dl
cmp si, cx
inc si
dec cl
jnz third_encry

judge_key_end:
mov si, 0
mov di, 0
mov al, [bx + 1]
mov bx, OFFSET key_right
push bx
lea bx, pwd1
oneByOne:
```

---

# 汇编语言程序设计实验报告

---

```
    mov cl, [bx]
    cmp cl, [change + si]
    jnz key_error
    mov bx, [bx + 1]
    inc si
    dec al
    jnz oneByOne
pop ax
mov bp, sp
mov ax, [bp - 2]
jmp ax
```

key\_right:

INPUT:

```
    MOV DX, OFFSET MSG1
    MOV AH, 9
    INT 21H ;功能一一小题
```

```
    LEA DX, IN_NAME
    MOV AH, 10
    INT 21H ;功能一二小题
```

```
    CALL SET_AVERAGE_GRADE
    MOV BL, IN_NAME + 1
    MOV BH, IN_NAME + 2
```

```
    CMP BL, 0
    JE INPUT
    CMP BH, 'q'
    JE DIE ;功能一 三小题
    MOV BH, 0
```

FIND:

```
    MOV CX, N
    MOV DI, 0
```

FIND\_S:

```
    MOV SI, 0
    PUSH CX
    MOV CX, BX
    CALL EQUAL
    POP CX
    CMP SI, BX
    JE SUCCESS_FIND
```

CONTINUE\_FIND:

```
    CMP CX, 1
```

# 汇编语言程序设计实验报告

---

```
JE NOT_FIND
ADD DI, 14
LOOP FIND_S
```

key\_error:

DIE:

```
cli                                ;还原中断矢量
mov ax, old_int_1
mov es:[1*4], ax
mov ax, old_int_1 + 2
mov es:[1*4+2], ax
mov ax, old_int_3
mov es:[3*4], ax
mov ax, old_int_3 + 2
mov es:[3*4+2], ax
sti
```

```
MOV AH, 4CH
INT 21H
```

NOT\_FIND:

```
MOV DX, OFFSET MSG2
MOV AH, 9
INT 21H
JMP INPUT
```

SUCCESS\_FIND:

```
ADD SI, DI
CMP [BUF + SI], 0a0h
JNE CONTINUE_FIND
SUB SI, DI
MOV WORD PTR [POIN], OFFSET BUF + 10
ADD WORD PTR [POIN], DI
CALL G_ABCD
JMP INPUT
```

;使用寄存器 AX,SI

;需要传入参数 SI = 0

EQUAL:

```
MOV AL, [IN_NAME + SI + 2]
sub al, 30h
shl al, 1
ADD SI, DI
CMP AL, [BUF + SI]
JNE NOT_EQUAL
SUB SI, DI
INC SI
```

# 汇编语言程序设计实验报告

---

```
    LOOP EQUAL
NOT_EQUAL:
    RET
```

```
G_ABCD:
    PUSH AX
    PUSH DX
    PUSH SI
    MOV DX, OFFSET CRLF
    MOV AH, 9
    INT 21H
    MOV SI, [POIN]
    ADD SI, 3
    MOV AX, [SI]
    xor al, 'B'
    MOV AH, 0
    SUB AL, 90
    JS  G_BCD
    MOV DL, 'A'
    JMP SCREEN
```

```
G_BCD:
    MOV AX, [SI]
    MOV AH, 0
    SUB AL, 80
    JS  G_CD
    MOV DL, 'B'
    JMP SCREEN
```

```
G_CD:
    MOV AX, [SI]
    MOV AH, 0
    SUB AL, 70
    JS  G_D
    MOV DL, 'C'
    JMP SCREEN
```

```
G_D:
    MOV DL, 'D'
    JMP SCREEN
```

```
SCREEN:
    MOV AH, 2
    INT 21H
    POP SI
    POP DX
    POP AX
    RET
```

```
SET_AVERAGE_GRADE:
```

---

# 汇编语言程序设计实验报告

---

```
PUSH SI
PUSH AX
PUSH BX
PUSH CX
PUSH DX

MOV SI, 10
MOV CX, N
MATH:
MOV AX, 0
MOV BX, 0
MOV DX, 0
MOV AL, [BUF + SI]
xor al, 'B'
MOV AH, 2
MUL AH ;AX IS CHINESE * 2 <= 200 16 IS ENOUGH
MOV BL, [BUF + SI + 1] ;MATH GRADE
xor bl, 'B'
MOV BH, 0
ADD BX, AX
MOV AL, [BUF + SI + 2] ;ENGLISH
xor al, 'B'
MOV AH, 0
MOV DL, 2
DIV DL
ADD BX, AX
MOV AX, 2
MUL BX
MOV BL, 7
DIV BL
xor al, 'B'
MOV [BUF + SI + 3], AL
ADD SI, 14
LOOP MATH

POP DX
POP CX
POP BX
POP AX
POP SI
RET

CODE ENDS
END START
```

## 3.4.4 实验步骤

1. 准备上机环境

# 汇编语言程序设计实验报告

- a) 操作系统: Dosbox : 0.74
- b) 汇编器: masm: 6.11
- c) 调试器: td 5.0
- d) 编辑器: vscode
2. 编写加密函数
  - a) 按设计思想中规则编写 栅栏加密函数
  - b) 按设计思想中规则编写 凯撒加密函数
  - c) 按设计思想中规则编写 异或加密函数
  - d) 按设计思想中规则编写 链表比对函数
  - e) 将编写的函数综合起来
3. 编写反跟踪代码
  - a) 编写检测中断反跟踪程序
  - b) 在第一个加密函数中使用堆栈检查反跟踪
    - i. 将第一个加密函数地址入栈
    - ii. 将此入栈信息出战
    - iii. 跳转到[esp + 4]对应的地址
  - c) 在第一个加密函数向第二个加密函数跳转过程加入间接转移指令实现跳转
4. 思考如何避免其他人直接跳过密码检测查询学生信息

## 3.4.5 实验记录与分析

1. 准备上机环境
2. 编写加密函数
  - a) 编写栅栏加密函数

首先获得输入字符串，将他们按中间分为两组，进行组合

```
first_encrypt:
    mov ah, [bx + di + 2]
    shl di, 1
    mov [change + di], ah
    mov ah, [bx + si + 2]
    mov [change + di + 1], ah
    xor ah, ah
    shr di, 1

    cli
    push P1
    inc di
    add dx, di
    inc si
    sub dx, ax
    pop cx
    mov bp, sp
    mov cx, [bp - 2]
    sti

    dec al
    jz 000
    jmp cx
```

图 4-1 栅栏加密函数

- b) 编写凯撒加密的过程



# 汇编语言程序设计实验报告

做加密操作  $(x - 'c') \bmod 26 + 'a'$

```
second_encry:
    mov al, [change + si]
    sub al, 63h
    jns su
    add al, 26
su:
    mov ah, 0
    mov bl, 26
    div bl
    add ah, 61h
    mov [change + si], ah

    inc si
    dec cl
    jnz second_encry
```

图 4-2 凯撒加密

## c) 编写异或加密

首先编写异或操作

```
third_encry:
    mov dl, [change + si]
    nop
    cmp ax, bx
    xor [change + si], al
    mov al, dl
    cmp si, cx
    inc si
    dec cl
    jnz third_encry
```

图 4-3 异或加密

然后编写链表比对:

```
judge_key_end:
    mov si, 0
    mov di, 0
    mov al, [bx + 1]
    mov bx, OFFSET key_right
    push bx
    lea bx, pwd1
oneByOne:
    mov cl, [bx]
    cmp cl, [change + si]
    jnz key_error
    mov bx, [bx + 1]
    inc si
    dec al
    jnz oneByOne
pop ax
mov bp, sp
mov ax, [bp - 2]
jmp ax
```

图 4-4 与链表进行比对

## d) 综合加密程序

综合的过程其实是隐藏自己程序的过程，不让对方看到程序的跳转，使用间接转移的方法来做。

# 汇编语言程序设计实验报告

```
lea bx, in_pwd
push bx
mov al, [bx + 1]
shr al, 1
mov ah, 0
mov si, ax
mov di, 0

mov bx, es:[1 * 4]
inc bx
jmp bx

dec al
jz ooo
jmp cx

ooo:
mov cl, [bx + 1]
mov si, 0
push bx
jmp dx
db 'Your will die, goto die!$'
```

图 4-5 将三个加密关联起来

## 2. 编写反跟踪代码

### a) 编写中断反跟踪程序

可以将中断的值修改为某个值，然后可以在执行过程中使用这个值。这里选择修改为地址，然后跳转此地址，这样可以防止单步调试。

```
mov bx, es:[1 * 4]
inc bx
jmp bx
```

图 4-6 中断检查

### b) 编写堆栈检查反跟踪程序

#### i. 将第一个加密函数地址入栈

```
push P1
P1 dw first_encry
```

#### ii. 将此入栈信息出栈

```
pop cx
```

#### iii. 跳转到[sp - 2]对应的地址

```
mov bp, sp
mov cx, [bp - 2]
sti

dec al
jz ooo
jmp cx
```

### c) 在第一个加密函数向第二个加密函数跳转过程加入间接转移指令实现跳转

## 3. 思考如何避免其他人直接跳过密码检测查询学生信息

这里源于我第一次提交 exe 给队友之后的思考，他说直接跳过输入密码就可以看到成绩了。

对于如何防止这样的事情，可以将加密成绩的方式和输入进行关联，这样跳过输入看到的的成绩也是错的。

## 3.5 任务五

### 3.5.1 实验步骤

1. 准备上机环境
2. 观察密码是否是明文存储

# 汇编语言程序设计实验报告

- a) 使用 TD 反汇编 EXE 文件，观察数据段
- b) 寻找类似密码的编码
- c) 暴力尝试
3. 如何综合利用静态反汇编和动态反汇编的信息破解程序？
4. 思考一下，如何用 C 语言（不嵌入汇编语言）实现反跟踪？是否能发现汇编语言的特殊之处？
4. 如何实现对密码的暴力破解？
5. 如何观察到程序存在反跟踪的代码？如何应对反跟踪程序？
  - a) 四种反跟踪的技巧
    - i. 对于修改中断矢量表，由于我们知道中断矢量表的位置，所以只需看程序有咩有访问那一部分内存的操作
    - ii. 对于间接转移，可以观察跳转指令和 call 指令对应的操作数
    - iii. 计时程序可以观察功能调用
    - iv. 堆栈检查观察有没有访问栈外元素
6. 若存在修改中断矢量表的代码时，一般会先关掉中断（也即执行 CLI 指令）。如果不想因为关中断指令的出现让跟踪者容易判断出后续存在反跟踪代码，应如何设计修改中断矢量表的代码，达到不用关中断的目的？
7. 对队友程序进行跟踪

## 3.5.2 实验记录与分析

1. 准备上机环境
2. 观察密码是否是明文存储
  - a) 使用 TD 反汇编 EXE 文件，观察数据段

```
ds:0068 21 20 54 68 65 20 70 61 ! The pa  
ds:0070 73 73 77 6F 72 64 20 69 ssword i  
ds:0078 73 20 66 61 6C 73 65 21 s false!  
ds:0080 70 6C 61 75 73 65 20 61 plause a
```

图 5-1 数据段

观察后发现数据区都是字符串或者是 00(空)，没有发现适合作为密码的数据。基本排除密码明文存储的可能。

- b) 寻找类似密码的编码  
没有类似密码的编码。
- c) 暴力尝试
3. 如何综合利用静态反汇编和动态反汇编的信息破解程序？  
在思考与感悟中有具体阐述。
4. 思考一下，如何用 C 语言（不嵌入汇编语言）实现反跟踪？
5. 如何实现对密码的暴力破解？

加入密码是 16 位，那么存在 40 多个可输入字符，这个可能的组合十分庞大，暴

# 汇编语言程序设计实验报告

力破解绝对要建立在已经观察了一部分代码的规律之后再去做才可以。所以观察一部分容易看书的部分密码，然后按照这种规则进行暴力尝试，可以在某些情况下这样做十分有用。

## 6. 如何观察到程序存在反跟踪的代码？如何应对反跟踪程序？

四种反跟踪的技巧

- 对于修改中断矢量表，由于我们知道中断矢量表的位置，所以只需看程序有访问那一部分内存的操作
  - 对于间接转移，可以观察跳转指令和 `call` 指令对应的操作数
  - 计时程序可以观察功能调用
  - 堆栈检查观察有没有访问栈外元素
7. 若存在修改中断矢量表的代码时，一般会先关掉中断（也即执行 `CLI` 指令）。如果不想因为关中断指令的出现让跟踪者容易判断出后续存在反跟踪代码，应如何设计修改中断矢量表的代码，达到不用关中断的目的？
8. 对队友的程序进行跟踪

我选择在 TD 中进行静态观察代码与动态调试相结合的方式破解队友的密码。

首先观察到队友使用了中断矢量表的方法

```
:001D FA      cli
:001E B83902    mov     ax,0239
:0021 6726A304000000 mov     es:[00000004],ax
:002B 67268C0D060000+mov     es:[00000006],cs
:0030 FB      sti
```

图 5-2 跟踪\_1

于是选择直接跳过这部分程序：

```
s:0039 B80000    mov     cx,0000
s:003C 67268B1D040000+mov     bx,es:[00000004]
s:0044 FFE3      jmp     bx
```

图 5-3 跟踪\_2

观察 `bx` 的值之后直接跳转到这条指令指示的地方。

```
cs:0236 E9B0FE    jmp     00E9
cs:0239 E93FFE    jmp     007B ↑
cs:023C 2F      das
```

图 5-4 成功跳过中断检测

接下来单步执行，寻找他加载自己输入数据的语句，然后寻找比对关系：

```
mov     ah,0A      movzx    si,byte ptr [bx+01]      movzx    si,byte ptr [bx+si+03]
int     21         sub     si,0030      sub     si,0057
lea     bx,[0128]   mov     al,[bx+si-01]      mov     ax,[bx+si]
mov     al,[bx+011] cmp     al,0D      cmp     ax,2134
add     al,30      jne     0056      jne     0056
add     bx,0002     movzx    si,byte ptr [bx+02]      movzx    si,byte ptr [bx+si-03]
cmp     [bx],al     sub     si,0050      sub     si,0050
jne     0056        mov     ax,[bx+si+02]      movzx    ax,byte ptr [bx+si+02]
movzx   si,byte ptr [bx+01] cmp     ax,5F4B      sub     ax,0050
sub     si,0030     jne     0056      mov     bx,014A
mov     al,[bx+si-01] movzx    si,byte ptr [bx+si-01]      cmp     si,ax
cmp     al,0D      sub     si,0040      jmp     004A
jne     0056        movzx    si,byte ptr [bx+si+03]      lea     dx,[0104]
```

图 5-5 加密过程

这里可以看到队友进行的加密过程，已经没有其他的反跟踪代码，我们可以解读下去。

分析第一次比较：

# 汇编语言程序设计实验报告

```
lea    bx,[0120]  
mov     al,[bx+01]  
add     al,30  
add     bx,0002  
cmp     [bx],al  
jne     0056
```

图 5-6 第一次比较

第一次观察后发现是第一个输入的字符要等于有多少个字符，假设输入  $n$  个字符，那么第一个就应该输入  $n$

```
movzx   si,byte ptr [bx+01]  
sub     si,0030  
mov     al,[bx+si-01]  
cmp     al,0D
```

图 5-7 第二次比较

第二次比较是对第二个字符减去三十的值所对应的值所在的输入的位置为回车，也就是输入的最后一个字符，所以这里应该是  $N+1$ ，有点绕

<pre>movzx   si,byte ptr [bx+02] sub     si,0050 mov     ax,[bx+si+02] cmp     ax,5F4B jne     0056</pre>	<pre>movzx   si,byte ptr [bx+si-01] sub     si,0040 movzx   si,byte ptr [bx+si+03] sub     si,0057 mov     ax,[bx+si] cmp     ax,2134 jne     0056</pre>	<pre>movzx   si,byte ptr [bx+si-03] sub     si,0050 movzx   ax,byte ptr [bx+si+02] sub     ax,0050 mov     bx,014A</pre>
---	--	--

图 5-8 第三四五次比较

这几次比较单独拿出来得不到什么有用的消息，若列在一起看，则他们表达了输入的剩余几个字符的相互之间的关系，如同解数独一般才可以解开密码，最终使用九牛二虎之力方能解开。

## 4 总结与体会

本次实验任务很多，有五个，各有各的特色，以下单纯的自己的体会，若有不对的地方还请指正。

首先是任务一，观察中断矢量表，这个可以看作是对接下来的四个任务的一个铺垫，里面可以得到的信息是很多的：首先你可以检查中断矢量地址，然后进行第二个修改中断矢量表中信息的程序就有了思路，第四个反跟踪实验也就明白了中断矢量修改之后在 TD 中会被改回来的事实。得到了反跟踪的一种思想。

然后任务二修改中断矢量，修改中断矢量的流程也比较简单，得到旧的地址（作为恢复使用），将新的地址程序送进去，然后将程序驻留在内存中即可。但是这个实验还让你明白了键盘输入，系统输入输出 IO 的作用。我感觉这里面最难的地方是自己很难调试，有没有效果只能在 DosBox 的命令行下观察，无法在 TD 中观察，而如果中断矢量修改错误了会让自己无法在 dosbox 里面输入任何东西，这时候必须重启，这样带来了一定的困难。

接着的任务三，观察 CMOS 里的信息，这个实验主要是熟悉系统 io，了解 CMOS 信息，比较简单。

接着任务四实现加密与反跟踪，加密好说，自己随便设计就好，关键是反跟踪，这里我的体会主要是自己没有做过跟踪的工作，做反跟踪会有一些不知所措，自己对跟踪的了解只停留在看文档中，并没有了解真正的跟踪。自己的反跟踪停留在老师给的样例的水平。

接着最后一个自己去 gzon 个程序，这是最激动人心的部分，我的队友的程序难点在于解开数独，不在反跟踪，自己使用 F4 直接跳过了他的反跟踪部分。

这里自己需要有几个问题进行思考，汇编语言的特点？反跟踪过程中我们可以在机器级别一条一条执行指令，而 C 语言或者其他高级语言并不能这样做。将机器的大部分状态暴露在程序员面前，这是我认为汇编语言最大的特点。

对于静态反汇编和动态反汇编，这两种其实需要配合使用，只用静态自己很容易出错，只用动态又很容易失去全局观并且很多反跟踪针对的就是动态跟踪。

# 汇 编 语 言 程 序 设 计 实 验 报 告

---

## 参考文献

- [1] 王爽. 汇编语言. 第三版. 清华大学出版社, 2003 01- 310
  - [2] 曹忠升 80X86 汇编语言程序设计 华中科技大学出版社
-