

目录

1	实验目的与要求.....	2
2	实验内容.....	2
3	实验过程.....	3
3.1	任务 1	3
3.1.1	设计思想及存储单元分配	3
3.1.2	源程序	4
3.1.3	实验步骤	4
3.1.4	实验记录与分析	5
3.2	任务 2	9
3.2.1	设计思想及存储单元分配	9
3.2.2	流程图	10
3.2.3	源程序	12
3.2.4	实验步骤	13
3.2.5	实验记录与分析	13
3.3	任务 3	16
3.3.1	设计思想及存储单元分配	16
3.3.2	流程图	17
3.3.3	源程序	17
3.3.4	实验步骤	19
3.3.5	实验记录与分析	19
3.4	任务 4	23
3.4.1	设计思想及存储单元分配	23
3.4.2	流程图	24
3.4.3	源程序	26
3.4.4	实验步骤	33
3.4.5	实验记录与分析	34
3.5	任务 5	37
3.5.1	设计思想及存储单元分配	37
3.5.2	流程图	37
3.5.3	实验步骤	38
3.5.4	实验记录与分析	38
4	总结与体会.....	42

汇 编 语 言 程 序 设 计 实 验 报 告

参考文献	43
------------	----

1 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。

2 实验内容

任务 1：用三种方式获取中断类型码 1H、13H 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)，在其数据区观察中断矢量表中的信息。
- (2) 编写程序，用 DOS 系统功能调用（具体调用方法见教材示例及附录中的描述）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，其主要功能是：在程序驻留并返回到 DOS 操作系统后，输入键盘上的大写字母时都变成了小写字母。

- (1) 在 DOS 虚拟机下执行程序，中断服务程序驻留内存。
- (2) 在 DOS 命令行下键入小写字母时，屏幕显示不变，键入大写时，屏幕显示为小写。执行 TD，在代码区输入指令“mov AX,0”，看是否都变成了小写。执行实验三任务 1 的程序，输入大小写是否正常？
- (3) 选作：另外单独编写一个中断服务程序的卸载程序，将自己驻留的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。

- (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：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（但不

汇编语言程序设计实验报告

应选择复杂的加密算法)。

可以采用计时、中断矢量表检查、堆栈检查、间接寻址等反跟踪方法中的几种方法组合起来进行反跟踪(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: 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。

3 实验过程

3.1 任务 1

3.1.1 设计思想及存储单元分配

1) 设计思想

1. 直接用 TD 查看中断矢量表,找到 1H 和 13H 的中断入口地址。我们先将 DS 置 0。由于每一个中断表项占有 4 个 byte,所以 1H 偏移地址为 4H,即跳转到 0000:0004 查看,对应的 CS 偏移地址为 4H,IP 的偏移地址为 6H。13H 的偏移地址为 4CH,即跳转到 0000:004C 查看,对应的 CS 偏移地址为 4CH,IP 偏移地址为 4EH。读出对应的 CS 和 IP。

2. 调用 DOS35 号功能,编写程序时将 1H 送入 AL,调用 INT21,然后将 13H 送入 AL,调用 INT21。用 TD 单步调试查看,其中 BX 和 ES 分别保存了对应的中断入口地址信息(BX 保存 IP,ES 保存 CS)。执行到对应语句的时候,查看 BX 和 ES 的值,看是否与 1 中方法的结果一致。

3. 分别用 BX 和 CX 读取相应的内存单元,先用 BX 读取[4H],CX 读取[6H],此为 1H 对应的内存单元,然后用 BX 读取[4CH],CX 读取[4EH],此为 13H 对应的内存单元,在运用 TD 单步调试查看,查看到对应位置时 BX 和 CX 的值是多少,即可以看出对应的 CS 和 IP 是多少(BX 对应 CS,CX 对应 IP)。

2) 存储单元分配

BX: 存储对应的 CS

CX: 存储对应的 IP

AX: 用于临时存储变量

汇编语言程序设计实验报告

3.1.2 源程序

方案 2 源程序:

```
.386
CODE SEGMENT USE16
    ASSUME CS:CODE
START:
    MOV AX, 0
    MOV DS, AX
    MOV AH, 35H
    MOV AL, 1
    INT 21H
    MOV AL, 13H
    INT 21H
CODE ENDS
END START
```

方案 3 源程序:

```
.386
CODE SEGMENT USE16
    ASSUME CS:CODE
START:
    MOV AX, 0
    MOV DS, AX
    MOV BX, WORD PTR DS:[4H] ;读取 1H 对应内存单元
    MOV CX, WORD PTR DS:[6H]
    MOV BX, WORD PTR DS:[4CH] ;读取 13H 对应内存单元
    MOV CX, WORD PTR DS:[4EH]
    MOV AH, 4CH
    INT 21H
CODE ENDS
```

3.1.3 实验步骤

1. 打开 dosbox 进入 td, 将 DS 置 0, 使用 goto 语句到 0004H, 查看 0004H 中的值, 为 1H 中断入口地址的 IP, 查看 0006H 中的值, 为 1H 中断入口地址的 CS。使用 goto 语句到 004CH, 查看 004CH 中的值为 13H 中断入口地址 IP, 004EH 中的值为 13H 中断入口地址的 CS。记下上述各值。

2. 录入方法二的程序, 编译连接形成可执行文件, 若有报错则处理报错。在 TD 中运行, 观看将 1H 送入 AL, 且执行了 INT21 的值之后, BX 和 ES 的值, BX 存放 IP, ES 存放 CS。记下这两个值。观看将 13H 送入 AL, 且执行了 INT21 的值之后, BX 和 ES 的值, BX 存放 IP, ES 存放 CS。记下这两个值, 比较其与一的观察是否相同。

3. 录入程序方法三的程序, 编译连接形成可执行文件, 若有报错则处理报错。打开 TD 进行测试, 查看执行到将[4H]和[6H]移动到 BX 和 CX 之后, BX 和 CX 的值(分别对应 13H 中断入口地址 IP 和 CS)。然后查看执行到将[4CH]和[4EH]移动到 BX 和 CX 之后, BX 和 CX 的值(分别对应 13H 中断入口地址 IP 和 CS), 比较其与一的观察是否相同。

4. 用 td 对整个中断矢量表进行观察, 看出其特点。

5. 用 td 把中断矢量表中的矢量值随意改成其他值, 观察出现的状况。

汇编语言程序设计实验报告

3.1.4 实验记录与分析

1. 在 td 中看 1H 和 13H 的中断入口地址。

1) 1H 的中断入口地址:

找到 0000:0004H, 截图如下:

```
s:0004 0008 0070 0008 0070
s:000C 0008 0070 0008 0070
s:0014 1060 F000 1060 F000
s:001C 1060 F000 FE45 F000
```

图 3-1 1H 的中断入口地址

可以看出, 1H 的中断入口地址为 CS:0070, IP:0008

2) 13H 的中断入口地址:

找到 0000:004CH, 截图如下:

```
s:004C 1140 F000 11A0 F000
s:0054 11C0 F000 11E0 F000
s:005C 1220 F000 12C0 F000
s:0064 12C0 F000 1240 F000
```

图 3-2 13H 中断入口地址

可以看出, 13H 的中断入口地址为 CS: F000, IP: 1140

2. DOS35 号功能调用查看中断入口地址

1) 用 td 单步执行编译好的程序, 到执行完 mov AL,01 和 INT 21H 之后, 截图如下:

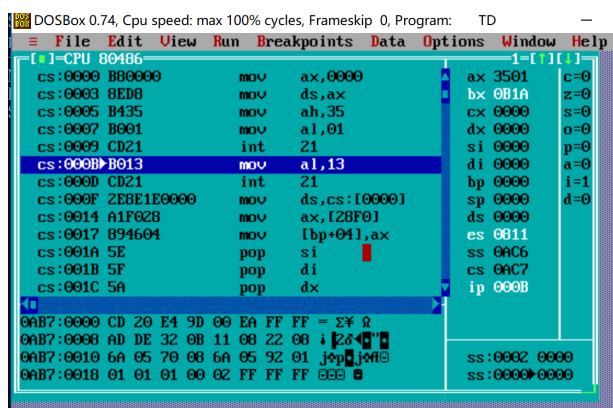


图 3-3 DOS35 号功能查看 1H 的中断入口地址

可以看出 BX 为 0B1A, ES 为 0811, 即 IP 为 0B1A, ES 为 0811。

2) 到执行完 MOV AL,13H 和 INT 21H 之后, 截图如下:

汇编语言程序设计实验报告

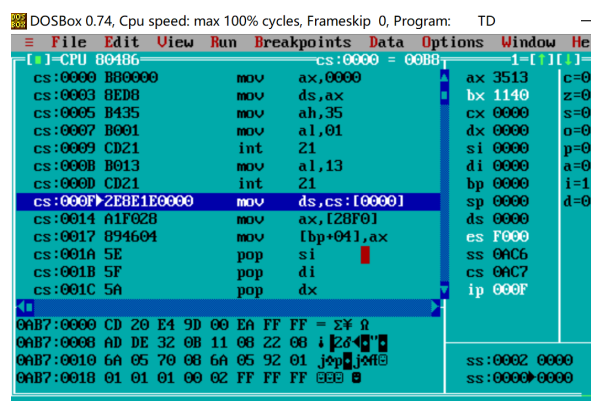


图 3-4 DOS35 号功能查看 13H 的中断入口地址

可以看出 BX 为 1140, ES 为 F000, 即 IP 为 1140, CS 为 F000。

由结果我们可以看出, 13H 的中断入口地址与在 TD 下看到的值相等, 但是 1H 的中断入口地址与在 TD 下面的不同, 我们给出以下解释: 首先, 1H 是 TD 要用的中断指令, 所以看上去和用 35H 调用所得到的入口地址会不一样。然后, TD 显示给用户的其实是一个被保护的副本, 而其真实值是随着被调用关系而改变的, 但是用 35H 调用并不能看到真实值, 只能看到 TD 给你的副本, 这也就解释了为什么 1H 在 DOS35H 和 TD 下直接看会有偏差。

3. 直接读取内存单元查看中断入口地址

用 td 单步执行编译好的程序, 将 DS:[4H]和 DS:[6H]分别移动进 BX 和 CX 之后, 如下截图:

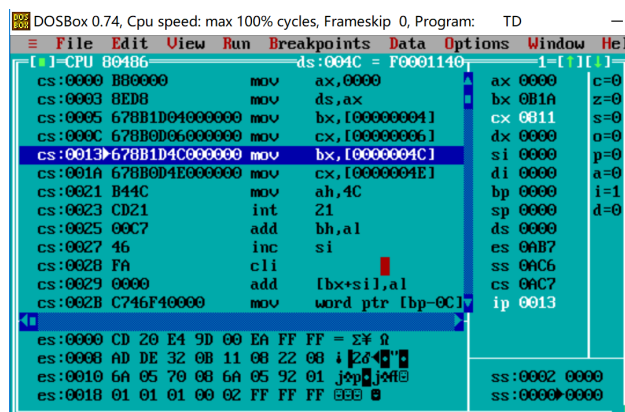


图 3-5 直接读取内存单元查看 1H 的中断入口地址

可以看出 BX 为 0B1A, CX 为 0B11, 即对应的 IP 为 0B1A, CS 为 0811。

将 DS:[4CH]和 DS:[4EH]分别移动进 BX 和 CX 之后, 如下截图:

汇编语言程序设计实验报告

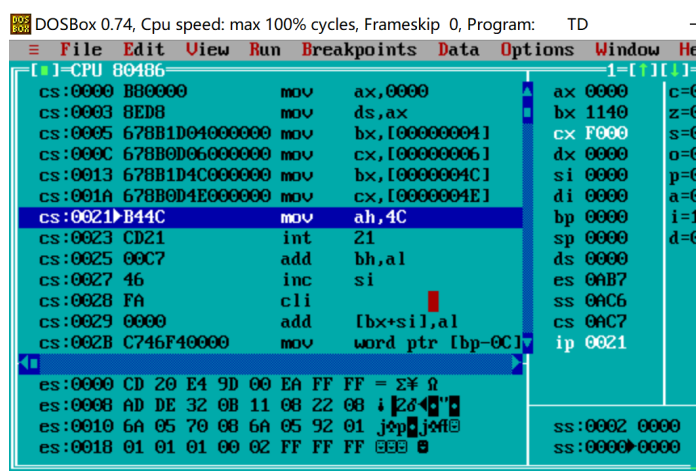


图 3-6 直接读取内存单元查看 13H 的中断入口地址

可以看出 BX 为 1140, CX 为 F000, 即对应的 IP 为 1140, CS 为 F000。

可以看出, 直接读取内存单元查看时, 13H 的结果与直接调用 TD 一致, 而 1H 的结果与直接调用 TD 不一致, 但是与 DOS35H 功能调用的结果一样。所以我们同样可以用上述原因解释这一现象。1H 是 TD 自己要用的中断程序, 而用 DOS35H 调用和直接查看内存时, 我们看到的实际是 TD 提供给我们看的一个假的、没有被改变的副本值, 而不是真实值。而用 TD 直接查看 1H 中断入口地址的时候我们看到的是真实值, 这就能解释 1H 入口地址在 3 中方式下查看的差异。

4. 观察整个中断矢量表, 分析其特点

中断矢量表的部分截图如下所示:

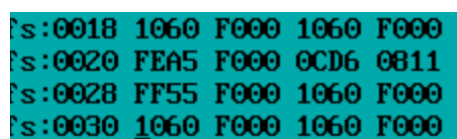


图 3-7 中断矢量表截图 1

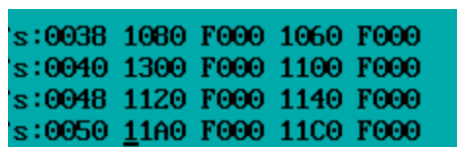


图 3-8 中断矢量表截图 2

可以看出, 在中断矢量表中, 有许多中断入口地址是相同的, 比如说 8H, CH, DH, EH 的中断入口地址是相同的, 均为 F000:1060。我们寻找中断程序地址 F000:1060 所对应的代码, 发现截图如下:

汇编语言程序设计实验报告



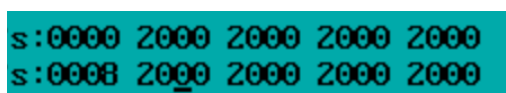
F000:1060	FE	db	FE
F000:1061	3B03	cmp	[bp+di],al
F000:1063	00CF	add	bh,cl

图 3-9 寻找 F000:1060 对应的代码

这是在保护模式下看到的代码，DOS（DOS 中的 TD）实际工作是在实际环境下但是展示给我们看的是保护模式下的代码，所以目前并不能得到更多有用的信息。所以我们唯一能确定的是，在中断矢量表中，有很多的入口地址指向的位置都是相同的。

5. 改变中断矢量表中的值，看对程序会有什么影响。

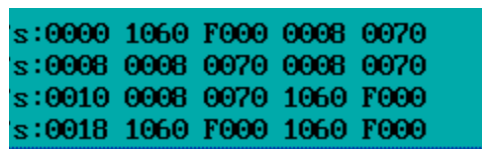
首先，我们改变中断矢量表中前 8 个字的值，全部变成 2000，如下图所示



s:0000	2000	2000	2000	2000
s:0008	2000	2000	2000	2000

图 3-10 修改中断矢量表中前 8 个字的值

我们再随便打开一个程序，例如打开 1-2.exe，这时候再查看中断矢量表，如下图所示：

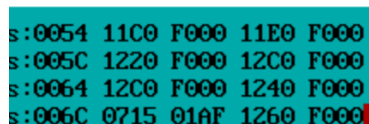


s:0000	1060	F000	0008	0070
s:0008	0008	0070	0008	0070
s:0010	0008	0070	1060	F000
s:0018	1060	F000	1060	F000

图 3-10 打开程序后中断矢量表的值

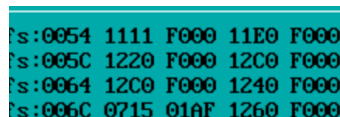
我们发现，中断矢量表恢复了原来的值。所以我们得到结论，对于中断矢量表中的值，DOS 系统对于其是有一定的保护措施，修改的时候修改的只是系统提的副本，而遇到其他情况时，例如键盘输入导致中断，或者像上述运行了一个新的程序的时候，中断矢量表会马上恢复备份，成原来的值，证明我们的改动并没有真正改变中断矢量表的值。

而在有些情况下，修改中断矢量表中的有一些地址是会引起错误的，例如修改 21H 对应的中断入口地址，将[0054]对应由 11C0 改成 1111，如下图所示：



s:0054	11C0	F000	11E0	F000
s:005C	1220	F000	12C0	F000
s:0064	12C0	F000	1240	F000
s:006C	0715	01AF	1260	F000

图 3-11 修改 21H 对应的中断入口地址（修改前）



s:0054	1111	F000	11E0	F000
s:005C	1220	F000	12C0	F000
s:0064	12C0	F000	1240	F000
s:006C	0715	01AF	1260	F000

图 3-12 修改 21H 对应的中断入口地址（修改后）

就在修改完成的同时，我们发现 DOSBox Status Window 中出现如下报错：

汇编语言程序设计实验报告

```
BIOS INT14: Unhandled call AH=4F DX= c73
BIOS INT14: Unhandled call AH=4F DX=1e18
BIOS INT14: Unhandled call AH=4F DX=3c18
BIOS INT14: Unhandled call AH=4F DX=3020
BIOS INT14: Unhandled call AH=4F DX=3020
BIOS INT14: Unhandled call AH=4F DX=100c
BIOS INT14: Unhandled call AH=4F DX=100c
BIOS INT14: Unhandled call AH=4F DX=219f
BIOS INT14: Unhandled call AH=4F DX=399f
BIOS INT14: Unhandled call AH=4F DX=239f
BIOS INT14: Unhandled call AH=4F DX= 100
BIOS INT14: Unhandled call AH=4F DX= 600
BIOS INT14: Unhandled call AH=4F DX= f00
BIOS INT14: Unhandled call AH=4F DX=290e
BIOS INT14: Unhandled call AH=4F DX=460e
BIOS INT14: Unhandled call AH=4F DX=670e
BIOS INT14: Unhandled call AH=4F DX=8c0e
BIOS INT14: Unhandled call AH=4F DX=5e00
BIOS INT14: Unhandled call AH=4F DX=bc00
BIOS INT14: Unhandled call AH=4F DX=239f
BIOS INT14: Unhandled call AH=4F DX=1e18
BIOS INT14: Unhandled call AH=4F DX=229f
BIOS INT14: Unhandled call AH=4F DX=3f9f
```

图 3-13 修改 21H 对应的中断入口地址的报错

而且会发现，td 会死机，即再也不能修改任何中断矢量表里面的值。

我们可以得出结论，21H 为系统功能调用，如果擅自修改其值，可能会导致程序或者 td 运行崩溃。所以说，虽然说有些中断矢量表里面的值是被保护的，可以修改的，但是对于一些系统的关键调用，最好不要修改其值，会导致程序（td）运行崩溃。

3.2 任务 2

3.2.1 设计思想及存储单元分配

1) 设计思想

1. 该实验要对 INT 16 处的对应的中断程序进行接管，首先我们用 PINSTRUCT 保存旧指令的 IP 和 CS。

2. 我们把中断矢量表 0000:0058H 开始的四个字节修改为新中断程序的 IP 和 CS。

3. 设计新的中断程序。由于 00H 和 10H 表示读入字符，所以对 AH 进行判断，如果是 00H 或者 10H，证明系统开始接受了读入字符的信息，于是开始执行将大写字母转换为小写字母的功能；如果不是 00H 或者 10H，则直接调用旧的中断程序。

4. 大小写转换的规则为，比较输入的字符（存在 AL 中），如果其 ASCII 码在 41H 到 5AH 之间，则将其 ASCII 码加上 20H，得到对应的大写 ASCII 码，如果不是，则直接而退出新的中断程序。

5. 中断程序的卸载。在 td 中读取旧指令的 IP 是 01E0H，CS 是 0F100H，将其分别存入 AX 和 BX，然后恢复原来的 CS 和 IP，将其分别存入 DS:[58H] 和 DS:[5AH]，恢复原来的地址。

汇编语言程序设计实验报告

2) 存储单元分配

AL: 接收输入的字符

AX、DX: 用于暂时保存数据和地址

3.2.2 流程图

中断子程序（把大写转换为小写）流程图如下所示：

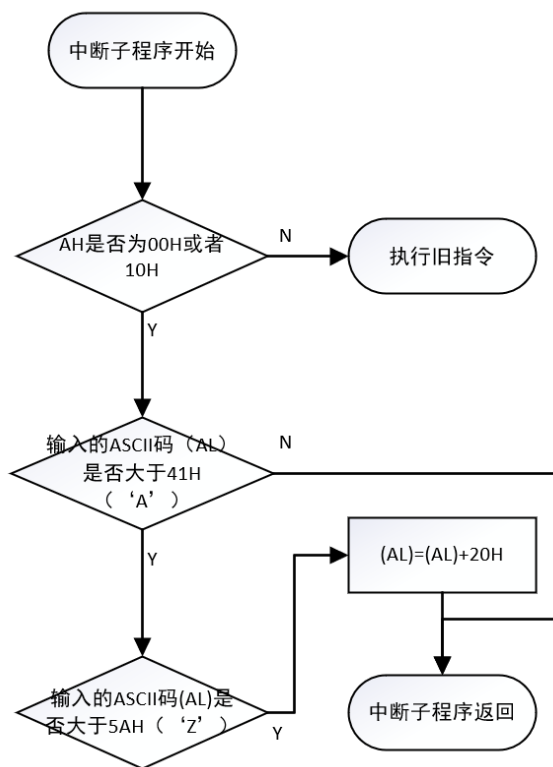


图 3-14 中断子程序流程图

对中断程序接管总流程图如下图所示：

汇编语言程序设计实验报告

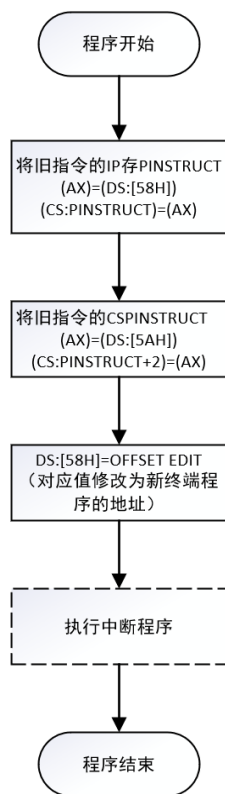


图 3-15 中断程序接管流程图

中断程序卸载流程图如下所示：

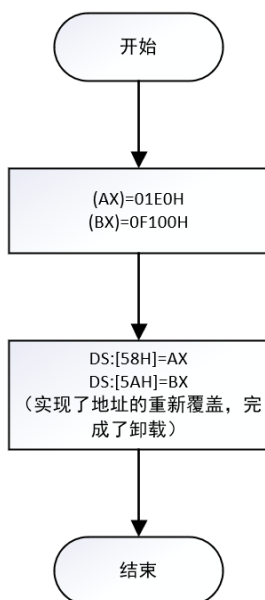


图 3-16 中断程序卸载流程图

汇编语言程序设计实验报告

3.2.3 源程序

中断接管程序代码:

```
.386
CODE SEGMENT USE16
    ASSUME CS:CODE, SS:STACK

    PINSTRUCT DW ?, ? ; 储存旧指令的偏移地址和段值

EDIT:
    CMP AH, 00H
    JZ LOP1
    CMP AH, 10H
    JZ LOP1
    JMP DWORD PTR CS:PINSTRUCT
LOP1:
    PUSHF
    CALL DWORD PTR CS:PINSTRUCT
    CMP AL, 41H ; 判断是否大于 A
    JAE LOP2
    JMP EXIT
LOP2:
    CMP AL, 5AH ; 判断是否小于 Z
    JBE TRANS
    JMP EXIT
TRANS:
    ADD AL, 20H ; 将大写变为小写
EXIT:
    IRET

START:
    XOR AX, AX
    MOV DS, AX
    MOV AX, DS:[58H]
    MOV CS:PINSTRUCT, AX ; 保存旧指令的 IP
    MOV AX, DS:[5AH]
    MOV CS:PINSTRUCT+2, AX ; 保存旧指令的 CS
    CLI
    MOV WORD PTR DS:[58H], OFFSET EDIT
    MOV DS:[5AH], CS
    STI
    MOV DX, OFFSET START+15
    SHR DX, 4
    ADD DX, 10H
    MOV AL, 0 ; DOS31 号功能调用
    MOV AH, 31H
    INT 21H
CODE ENDS
STACK SEGMENT USE16 STACK
    DB 200 DUP(0)
STACK ENDS
END START
```

中断卸载程序代码:

```
.386
CODE SEGMENT USE16
```

汇编语言程序设计实验报告

```
ASSUME CS:CODE, SS:STACK
START:
    XOR AX, AX
    MOV DS, AX
    CLI
    MOV AX, 01E0H ;原来的地址
    MOV BX, 0F100H
    MOV WORD PTR DS:[58H], AX ;恢复原地址
    MOV WORD PTR DS:[5AH], BX
    STI
    MOV AH, 4CH
    INT 21H
CODE ENDS
STACK SEGMENT USE16 STACK
    DB 200 DUP(0)
STACK ENDS
END START
```

3.2.4 实验步骤

1. 录入中断接管程序和中断卸载程序，用 masm 进行编译连接，若有报错，查阅手册解决报错，生成可执行文件。
2. 调用中断接管程序并测试：往屏幕中输入同时含有大写和小写的混合字符串，看大写字符是否能被转换为小写字符输出在屏幕上。
3. 调用中断卸载程序并测试：在步骤 2 的基础上，调用中断卸载程序，往屏幕中输入同时含有大写和小写的混合字符串，看是否能正常输出大写字符，若能正常输出，则卸载成功。
4. 在步骤 2 的基础上，同时打开另一个虚拟 DOS 窗口，观察键盘大小写是否被替代。
5. 调用中断接管程序的基础上，在 td 中输入 mov AX,0，看大写是否能正常变为小写。
6. 使用其它方法确定自己的中断处理程序已经被调用。调用中断接管程序，用 TD 进行调试，查看中断 16H 的入口地址，在 CS 段查看该地址所对应的代码，看是否为我们所编写的中断程序。

3.2.5 实验记录与分析

1. 编译程序以及连接过程没有错误出现。
2. 调用中断接管程序并测试

首先调用中断管理程序，如下图所示：

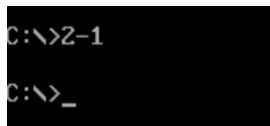
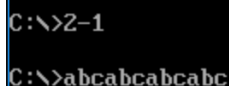


图 3-17 调用中断管理程序示意图

向屏幕中输入含有大小写的混合字符 ABCabcABCabc，输出到屏幕上结果如下：

汇编语言程序设计实验报告

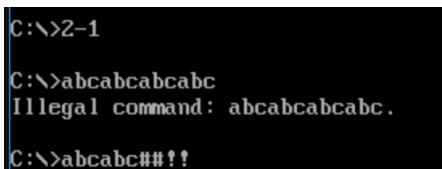


```
C:\>2-1
C:\>abcabcabcabc
```

图 3-18 输入大小写混合字符的运行截图

可以看出，混合字符中大写字母全部变为了小写字母，运行正确。

向屏幕中输入一些带有特殊字符的混合字符，看是否会干扰程序执行，例如输入 ABCabc##!!，输出到屏幕上如下：



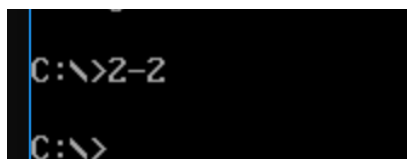
```
C:\>2-1
C:\>abcabcabcabc
Illegal command: abcabcabcabc.
C:\>abcabc##!!
```

图 3-19 输入带有特殊字符的混合字符的运行截图

可以看出，大写字符还是正常变为了小写字母，特殊字符并没有干扰判断。

3.调用卸载程序并测试

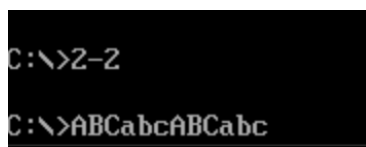
在上一问的实验基础上，运行卸载程序，如下图所示：



```
C:\>2-2
C:\>
```

图 3-20 调用卸载程序示意图

将含有大小写的混合字符 ABCabcABCabc，输出到屏幕上结果如下：




```
C:\>2-2
C:\>ABCabcABCabc
```

图 3-21 卸载后输入大小写混合字符的运行截图

可以发现大写字符不会再变为小写字母，运行正确。

将一些带有特殊字符的混合字符输入，看是否会干扰程序执行，例如输入 ABCabc##!!，输出到屏幕上如下：



```
C:\>2-2
C:\>ABCabcABCabc
Illegal command: ABCabcABCabc.
C:\>ABCabc##!!
```

图 3-22 卸载后输入带有特殊字符的混合字符的运行截图

发现输出正常，大写字符也不会变为小写字母，运行正确。

汇编语言程序设计实验报告

4. 在步骤二的条件下，打开另一个 DOS 窗口，观察大写是否会被小写替代。

同时打开两个 DOS 窗口，先在第一个窗口调用程序 2-1（即含有中断接管程序），写入 ABCabc，然后另一个不调用，也写入 ABCabc，观察得到截图如下：

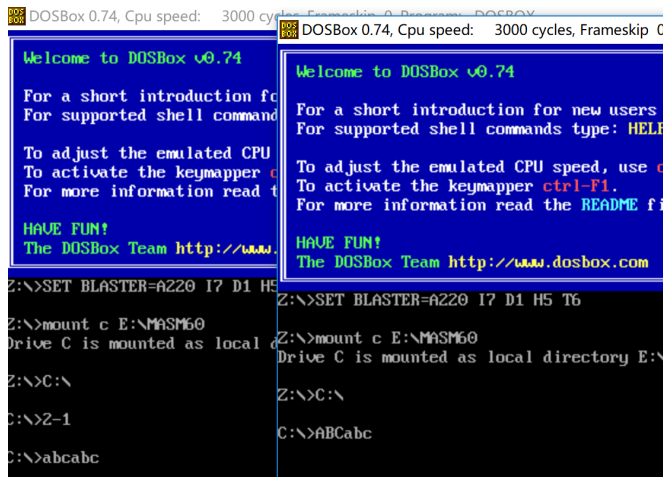


图 3-23 打开另一个 DOS 窗口观察是否会变大写为小写

结果我们发现在另一个 DOS 窗口中，大写不会变为小写。这说明我们编写的新中断程序只能在当前虚拟机下执行，这说明我们修改中断的时候，并没有深入到直接能够修改从键盘输入的大小写字符，而是只在虚拟机的层面上进行了改变，这实际上是一个映射关系，就是在键盘上输入了一个字符，只在虚拟机上进行映射，若调用了修改中断程序，则将大写字符变成了一个小写字符，并没有深入到键盘输入这一个层面。

5. 在调用了中断接管程序的基础上，在 td 中输入 mov AX,0。看大写是否能正常变为小写。

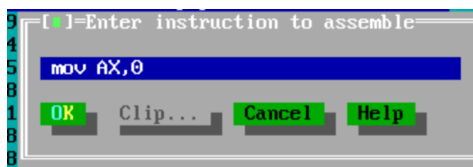


图 3-24 判断调用接管程序之后 td 中的大写是否会变为小写

由实验结果我们可以看出，调用了 2-1（大写变小写的接管程序）之后，td 中的大写并没有变为小写，这说明 16H 的中断地址并没有被成功替换。我的分析是 td 在执行的时候，有一套自己的中断矢量表，这个矢量表不受到外部程序修改的影响。td 在真实执行的时候，用的就是这样一套矢量表，所以在调用 16H 的中断地址的时候，并没有使用我们的修改地址，而是使用的它自己的原来的地址，所以并没有将大写变为小写。

6. 用其他方式确定中断程序被调用

调用中断接管程序，找到 16H 对应的中断入口地址为 01A2:0058，截图如下：

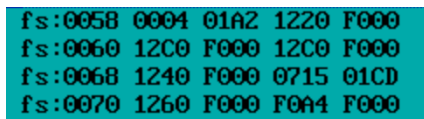
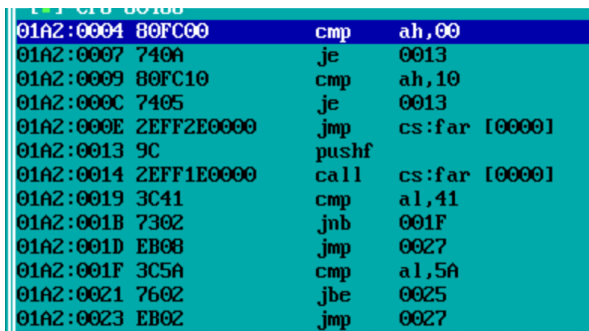


图 3-25 调用中断接管程序后 16H 的中断入口

汇编语言程序设计实验报告

找到 CS 段中 01A2:0058 所对应的代码，如下图所示：



01A2:0004	80FC00	cmp	ah,00
01A2:0007	740A	je	0013
01A2:0009	80FC10	cmp	ah,10
01A2:000C	7405	je	0013
01A2:000E	2EFF2E0000	jmp	cs:far [0000]
01A2:0013	9C	pushf	
01A2:0014	2EFF1E0000	call	cs:far [0000]
01A2:0019	3C41	cmp	al,41
01A2:001B	7302	jnb	001F
01A2:001D	EB08	jmp	0027
01A2:001F	3C5A	cmp	al,5A
01A2:0021	7602	jbe	0025
01A2:0023	EB02	jmp	0027

图 3-26 调用中断接管程序后 16H 的中断入口地址的代码

可以看出，这时候的程序是我们所编写的中断程序，实现大写转换为小写（由开始几句代码可以看出，开始几句为判断 AH 是否为 00 或者 10，即判断是否读入键盘输入）。所以我们可以看出，我们所编写的中断程序被调用了。

3.3 任务 3

3.3.1 设计思想及存储单元分配

1) 设计思想

1. 在数据段定义一个待读取的 CMOS 内部单元的地址编号，名称为 BIA。
2. 用 AL 读入需要操作的 CMOS 地址编号，即 BIA，调用 OUT, 用 70 号端口准备访问该地址。
3. 调用 IN, 用 71 号端口读入 CMOS 对应地址的数据，存入 AL。
4. 将高四位移送到 AH，AL 中保留低四位。具体做法为：先将 AL MOV 进入 AH，AH 右移动四位，即 AH 存入的是对应数据高四位；将 AL 与 0FH 相与，这样 AL 中存入的就是低四位。
5. 将 AL 和 AH 都加上 30H，得到对应 ASCII 码（也可以将 AX 加上 3030H），交换 AH 和 AL，将高位放在前面显示。
6. 调用 9 号功能将十进制信息输出。然后将十进制字符串转为 16 进制数，输出对应的 16 进制数。

2) 存储单元分配

寄存器：

AL：用于接收访问地址编号，用于存储得到的数据的低四位。

AH：用于存储得到数据的高四位。

BX, DX：用于临时存储变量。

存储器：

BIA：用于接收 CMOS 内部单元的地址编号。

NUM：用于接收对应地址编号储存的数据。

汇编语言程序设计实验报告

3.3.2 流程图

下图为任务三设计的流程图：

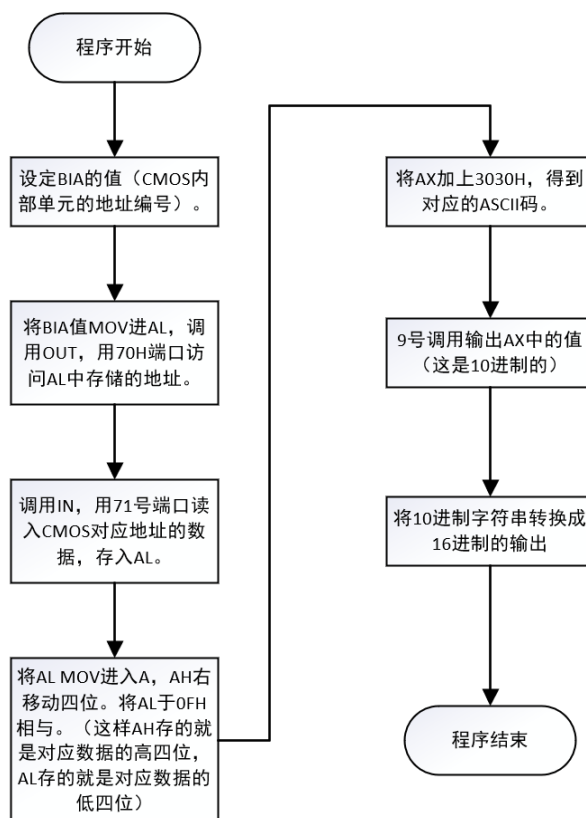


图 3-27 任务三设计流程图

3.3.3 源程序

```
.386
DATA SEGMENT USE16
    INF1 DB 'IN DECIMAL: $'
    INF2 DB 'IN HEXADECIMAL: $'
    INF3 DB 'HOUR SHOW$'
    INF4 DB 'MINUTE SHOW$'
    INF5 DB 'DATE SHOW$'
    INF6 DB 'MONTH SHOW$'
    INF7 DB 'ZERO SHOW$'
    BIA DB 04H ;储存当前调用的地址
    NUM DB ? ;储存当前地址的数据
DATA ENDS

STACK SEGMENT USE16 STACK
    DB 200 DUP(0)
STACK ENDS

CODE SEGMENT USE16
    ASSUME CS:CODE, DS:DATA, SS:STACK
```

汇编语言程序设计实验报告

START:

```
MOV AX, DATA
MOV DS, AX
MOV AL, BIA ;读取操作地址
OUT 70H, AL ;设定访问信息地址
JMP $+2 ;延时
IN AL, 71H
MOV AH, AL
AND AL, 0FH ;AL 中存入得到数据的低四位
SHR AH, 4 ;得到数据的高四位存入 AH 第四位中
ADD AX, 3030H ;转换为 ASCII 码
XCHG AH, AL ;将高位放在前面显示
MOV WORD PTR NUM, AX ;得到数据结果放入 NUM 中
;提示输出的信息种类
LEA DX, INF7
MOV AH, 9H
INT 21H
MOV DL, 0AH
MOV AH, 2H
INT 21H
;输出十进制时间
LEA DX, INF1
MOV AH, 9H
INT 21H
MOV BX, OFFSET NUM
ADD BX, 2
MOV AX, '$'
MOV [BX], AX
LEA DX, NUM
MOV AH, 9H
INT 21H
;输出十六进制时间
MOV DL, 0AH
MOV AH, 2H
INT 21H
LEA DX, INF2
MOV AH, 9H
INT 21H
CALL SHOW16
MOV AH, 4CH
INT 21H
```

;将 10 进制数转换为 16 进制输出，入口参数为 BX 存放待转换数

TRANS16 PROC

MOV CX, 4 ;16 进制

LOP1:

```
ROL BX, 4 ;从最高四位开始处理，将其通过循环移位移动到最低 4 位
MOV AL, BL
AND AL, 0FH ;将 AL 高四位位置为 0，取低四位
ADD AL, 30H
CMP AL, '9'
JBE PRINT ;如果为 0-9，直接输出
ADD AL, 07H ;否则转换为 A-F，再输出
```

PRINT:

```
MOV DL, AL
MOV AH, 2H
INT 21H ;输出 al 中的字符
```

汇编语言程序设计实验报告

```
DEC CX
JNZ LOP1
RET
TRANS16 ENDP

;将读取到的数字字符串转换成数字, NUM 中存入要转换的数字, 存入 BX
TRANS1 PROC
    MOV BX, 0
    MOV SI, OFFSET NUM
LOP2:
    MOV AH, 0
    MOV AL, BYTE PTR[SI]
    CMP AX, '$'
    JZ EXIT
    SUB AX, '0'
    XCHG BX, AX
    MOV CX, 10
    MUL CX
    ADD BX, AX
    INC SI
    JMP LOP2
EXIT: RET
TRANS1 ENDP

SHOW16 PROC
    CALL TRANS1
    CALL TRANS16
    RET
SHOW16 ENDP

CODE ENDS
END START
```

3.3.4 实验步骤

1. 录入源程序, 用 dosbox 进行编译和连接, 如发现报错则返回修改报错, 可查阅手册解决相关问题, 生成可执行文件。
2. 测试程序。分别设置 BIA 的值 (待读取的 CMOS 内部单元的地址编号) 为:
 - a) 04H, 读取本机时间的时, 与本机时间对照, 看是否一致。
 - b) 02H, 读取本机时间的分, 与本机时间对照, 看是否一致。
 - c) 调用 07H 查看 Date-Day (几号); 调用 08H 查看 Date-Month (几月); 并且验证。
 - d) 调用 12H 号地址 (输出恒为 0), 查看输出是否为 0。
3. 打开 TD, 在 TD 下用 IN/OUT 指令获取 CMOS 数据。

3.3.5 实验记录与分析

1. 编译程序以及连接过程没有错误出现, 正常生成可执行文件。

汇编语言程序设计实验报告

2. 测试程序

1) 设置 BIA 的值为 04H, 查看本机的时 (HOUR) 的信息, 截图如下:

```
BIA DB 04H ;储存当前调用的地址
NUM DB ? ;储存当前地址的数据
DATA ENDS
```

图 3-28 调用 CMOS 中地址 04H 截图

编译链接程序, 调用程序, 截图如下:

```
C:\>3
HOUR SHOW
IN DECIMAL: 19
IN HEXADECIMAL: 0013
```

图 3-29 显示小时时间截图

可以看出, CMOS 调用的结果为现在的小时信息为 19 点 (十进制) (0013 十六进制)。

下面我们查看本机真实时间, 发现是 19 点, 截图如下:



图 3-30 本机真实时间

可以看出, 用 CMOS 调用的时 (HOUR) 的信息与我们本机的时间相同, 对照一致。

2) 设置 BIA 的值为 02H, 查看本机的分 (MINUTE) 的信息, 截图如下:

```
BIA DB 02H ;储存当前调用的地址
NUM DB ? ;储存当前地址的数据
DATA ENDS
```

图 3-31 调用 CMOS 中地址 02H 截图

编译链接程序, 调用程序, 截图如下:

```
C:\>3
MINUTE SHOW
IN DECIMAL: 23
IN HEXADECIMAL: 0017
```

图 3-32 显示分钟时间截图

可以看出, CMOS 调用的结果为现在的分钟时间信息为 23 分 (十进制) (0017 十六进制)。

下面我们查看本机真实时间, 发现是 23 分, 截图如下:

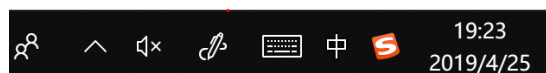


图 3-33 本机真实时间

可以看出, 用 CMOS 调用的分 (MINUTE) 的信息与我们本机的时间相同, 对照一致。

3) 设置 BIA 的值为 07H, 查看本机的日期 (DATE) 的信息, 截图如下:

汇编语言程序设计实验报告

```
BIA DB 07H ;储存当前调用的地址
NUM DB ? ;储存当前地址的数据
DATA ENDS
```

图 3-34 调用 CMOS 中地址 07H 截图

编译链接程序，调用程序，截图如下：

```
C:\>3
DATE SHOW
IN DECIMAL: 25
IN HEXADECIMAL: 0019
```

图 3-34 显示日期时间截图

可以看出，CMOS 调用的结果为现在的日期信息为 25 号（十进制）（0019 十六进制）。

下面我们查看本机真实日期，发现是 25 号，截图如下：

19:35
2019/4/25

图 3-35 本机真实日期

可以看出，用 CMOS 调用的日期（DATE）的信息与我们本机的相同，对照一致。

4) 设置 BIA 的值为 08H，查看本机的月份（MONTH）的信息，截图如下：

```
BIA DB 08H ;储存当前调用的地址
NUM DB ? ;储存当前地址的数据
DATA ENDS
```

图 3-36 调用 CMOS 中地址 08H 截图

编译链接程序，调用程序，截图如下：

```
C:\>3
MONTH SHOW
IN DECIMAL: 04
IN HEXADECIMAL: 0004
C:\>_
```

图 3-37 显示月份截图

可以看出，CMOS 调用的结果为现在的月份信息为 4 月（十进制）（0004 十六进制）。

下面我们查看本机真实月份信息，发现是 4 月，截图如下：

19:38
2019/4/25

图 3-38 本机真实月份

可以看出，用 CMOS 调用的月份（MONTH）的信息与我们本机的相同，对照一致。

5) 设置 BIA 的值为 12H（输出恒为 0），截图如下：

汇编语言程序设计实验报告

```
BIA DB 12H ;储存当前调用的地址
NUM DB ? ;储存当前地址的数据
DATA ENDS
```

图 3-39 调用 CMOS 中地址 12H 截图

编译链接程序，调用程序，截图如下：

```
C:\>3
ZERO SHOW
IN DECIMAL: 00
IN HEXADECIMAL: 0000
```

图 3-40 显示 0 截图

可以看出，CMOS 调用的结果与预想一致，输出为 0。

3. 打开 TD，在 TD 下用 IN/OUT 指令获取 CMOS 数据。

1) 获取分钟信息（MINUTE）

我们在 TD 中输入如下指令：MOV AL,2; OUT 70H,AL; IN AL,7H，如下图所示：

```
IN AL,71H
OUT 70H,AL
MOV AL,2
```

图 3-41 输入获取分钟信息的指令

发现分钟信息已经被送入了 AL，图中本机分钟信息为 51 分，截图如下：

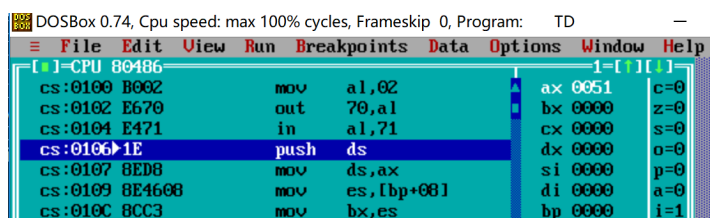


图 3-42 AL 中存入分钟信息截图

与本机分钟时间对照，发现本机分钟时间（MINUTE）为 51 分，截图如下：



图 3-43 本机 MINUTE 时间

所以我们通过在 td 下输入指令从 CMOS 中获取了正确的 MINUTE 信息。

2) 获取日期信息（DATE）

我们在 TD 中输入如下指令：MOV AL,7H; OUT 70H,AL; IN AL,7H，如下图所示：

汇编语言程序设计实验报告

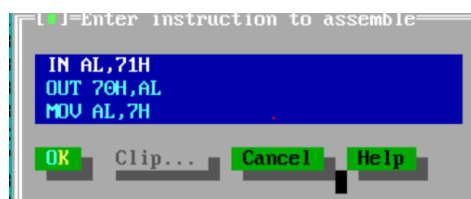


图 3-44 输入获取日期信息的指令

发现日期信息已经被送入了 AL，图中本机日期信息为 25 日，截图如下：

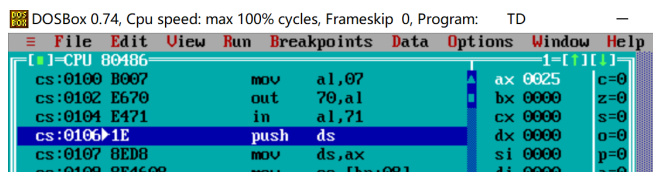


图 3-45 AL 中存入日期信息截图

与本机日期对照，发现本机日期（DATE）为 25 日，截图如下：



图 3-46 本机日期时间

所以我们通过在 td 下输入指令从 CMOS 中获取了正确的 DATE 信息。

3.4 任务 4

3.4.1 设计思想及存储单元分配

1) 设计思想

1. 登录密码加密思想：原本密码为 ABC，我们利用函数 $(X-29H)*3$ 对密码进行编码，即 $('A' -29H)*3$, $('B' -29H)*3$, $('C' -29H)*3$ ，达到加密密码的效果。对于判断用户输入密码模块的部分，我们只需要把用户输入的数值减去 29H 再乘以 3，再与数据段中的密文部分进行比较判断即可。

2. 进货价加密思想：将进货价与 'D' 进行异或操作，即 $X \text{ XOR } 'D'$ ，在数据段中进行加密，在代码段实际运算的时候，只需要再对于数据段中的数值再对 'D' 进行异或即可。

3. 输入三次密码不正确则跳过登录环节设计思想：数据段中定义一个变量 PASSTIME，其值为 3，每当输入密码错误的时候，PASSTIME 的值减 1，当 PASSTIME 的值为 0 的时候，自动跳过入查询货物环节，且将 AUTH 置 0。

4. 中断矢量表反跟踪设计思想：用 OLDINT1 和 OLDINT3 存储 1 号中断和 3 号中断的原中断矢量。然后修改他们的入口地址指向 NEWINT，NEWINT 对应的是一条 IRET 指令。然后判断对应的地址是否改成了新设置的入口地址，如果不是，代表有调试工具，则程序结束。

5. 通过计时的方法地址动态调试跟踪设计思想：将几条指令进行计时：即判断用户输入姓名的时候是否为回车，以及判断用户输入姓名的时候是否为 q 这几条汇编语句进行计时。（在指令开始之前获取一次时间，指令结束之后再次获取一下时间）。若时间差值为 0 或 55ms（常用计时程序精度为 55ms）则未被跟踪，否则退出程序。

汇编语言程序设计实验报告

6. 检查堆栈抵制动态调试跟踪：先把 PASSWORD 的地址设定为到栈顶以上几个字存储区的内容，然后到特定位置的时候进行出栈操作，存入 bx 寄存器，并且执行指令 jmp bx。看是否能正常跳转到 PASSWORD 的位置。（若有调试工具出现，则栈顶值被修改，不能正常跳转到 PASSWORD）

2) 存储单元分配

寄存器：

AX：暂存一些地址

AL、AH、BL、BH：暂存数据

存储器：

OLDINT1：储存 1 号中断的原中断矢量

OLDINT3：储存 3 号中断的原中断矢量

CURRENT：用于存储当前还能输入密码的次数

3.4.2 流程图

下面先给出三种反跟踪方法的流程图：

检查中断矢量反跟踪法流程图如下：

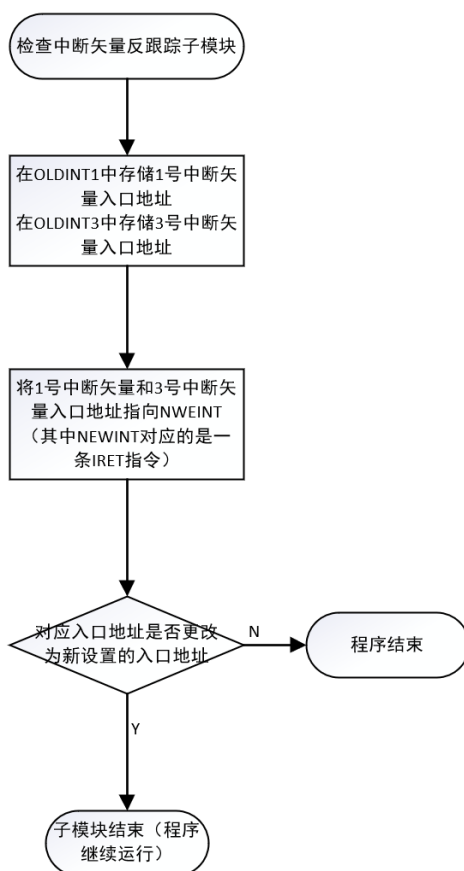


图 3-47 检查中断矢量表反跟踪法流程图

汇编语言程序设计实验报告

计时反跟踪法流程图如下：

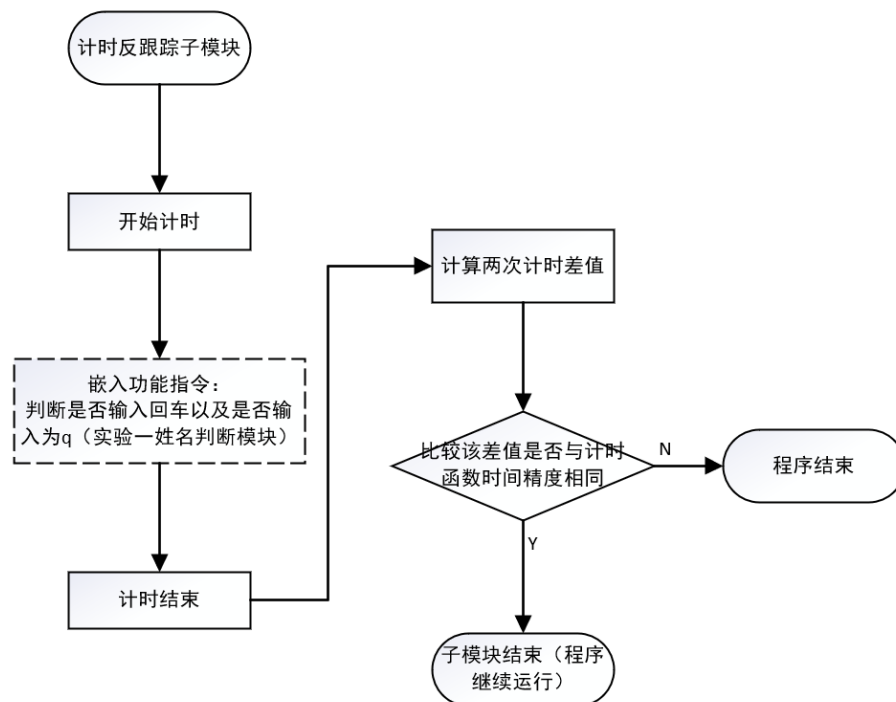


图 3-48 计时反跟踪法流程图

检查堆栈法反跟踪流程图如下：

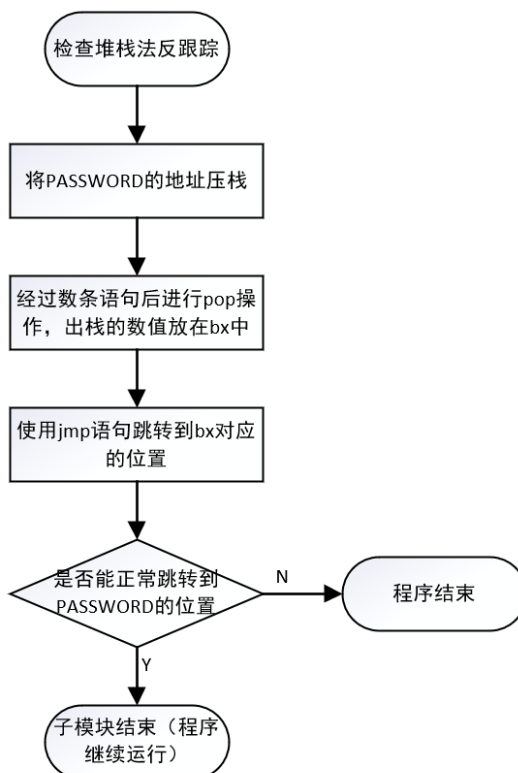


图 3-49 检查堆栈反跟踪法流程图

汇编语言程序设计实验报告

程序总流程图如下：

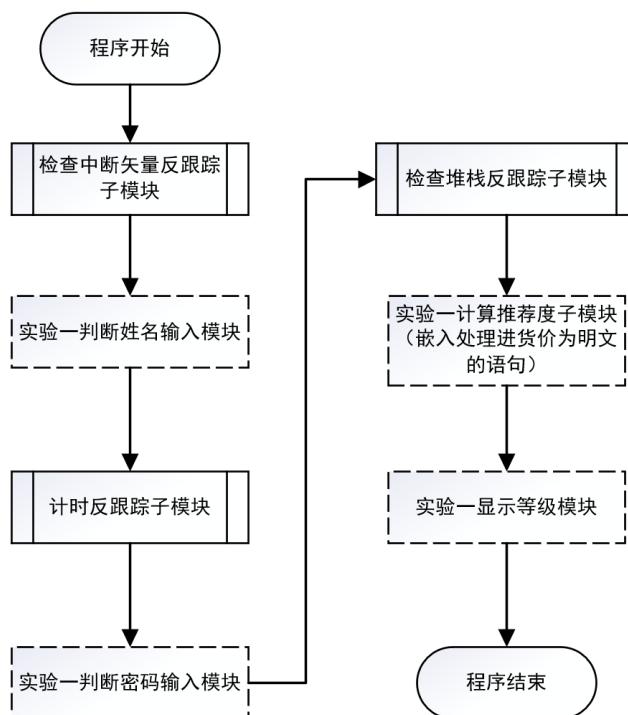


图 3-50 任务 4 总流程图

3.4.3 源程序

以下程序中，加密部分和反跟踪部分都用小写字母予以区分：

.386

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

```
DATA SEGMENT USE16
    BNAME DB 'WL DING', 0, 0, 0 ;老板姓名
    BPASS DB ('A' - 29H) * 3, ('B' - 29H) * 3, ('C' - 29H) * 3 ;密码
    N EQU 30 ;商品数量
    GA1 DB 'PEN', 7 DUP(0), 10 ;商品名称及折扣
        DW 35 XOR 'D', 56, 70, 25, ? ;推荐度还未计算
    GA2 DB 'BOOK', 6 DUP(0), 9 ;商品名称及折扣
        DW 12 XOR 'D', 30, 25, 5, ? ;推荐度还未计算
    GA3 DB 'PENCIL', 4 DUP(0), 9
        DW 20 XOR 'D', 30, 25, 5, ? ;设置比较大的进货量，防溢出测试
    E1 DW OVER
    P2 DW PASSWORD
    CURRENT DB 0
    OLDINT1 DW 0, 0 ;1号中断的原中断矢量（用于中断矢量表反跟踪）
    OLDINT3 DW 0, 0 ;3号中断的原中断矢量
    PASSTIME DB 3 ;设定最大输入密码次数

    AUTH DB 0
    TEMP1 DW ?
```

汇编语言程序设计实验报告

```
IN_NAME DB 12
         DB ?
         DB 12 DUP(0)
IN_PWD DB 10
        DB ?
        DB 10 DUP(0)
IN_ITEM DB 12
        DB ?
        DB 12 DUP(0)
TWO TIME DB 'YOU HAVE ONLY TWO TIME TO INPUT PASSWORD$'
ONE TIME DB 'YOU HAVE ONE TIME TO INPUT PASSWORD$'
PUTNAME DB 'PLEASE INPUT THE NAME(INPUT ENTER TO LOOKUP, INPUT Q TO QUIT)$'
NOTITEM DB 'DO NOT FIND THE ITEM$'
PUTPASSWORD DB 'PLEASE INPUT THE PASSWORD$'
ITEMNAME DB 'PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)$'
LOGFAIL1 DB 'YOUR USER NAME IS WRONG!$'
LOGFAIL2 DB 'YOUR PASSWORD IS WRONG!$'
LOGINREMINDB DB 'IDENTIFICATION GOT!$'
DATA ENDS

CODE SEGMENT USE16
    ASSUME CS:CODE, DS:DATA
START:
    MOV AX, DATA
    MOV DS, AX ;设置数据段寄存器的值

    xor ax, ax ;接管调试用中断, 中断矢量表反跟踪
    mov es, ax
    mov ax, es:[1*4] ;保存原 1 号和 3 号中断矢量
    mov OLDINT1, ax
    mov ax, es:[1*4+2]
    mov OLDINT1+2, ax
    mov ax, es:[3*4]
    mov OLDINT3, ax
    mov ax, es:[3*4+2]
    mov OLDINT3+2, ax
    cli ;设置新的中断矢量
    mov ax, OFFSET NEWINT
    mov es:[1*4], ax
    mov es:[1*4+2], cs
    mov es:[3*4], ax
    mov es:[3*4+2], cs
    sti

    JMP INNAME

INNAME:
    MOV AUTH, 0
    LEA DX, PUTNAME ;提示用户输入名字
    MOV AH, 9H
    INT 21H
    MOV DL, 0AH
    MOV AH, 2H
    INT 21H

    LEA DX, IN_NAME
```

汇编语言程序设计实验报告

```
MOV AH, 10
INT 21H
MOV DL, 0AH
MOV AH, 2H
INT 21H
LEA BX, OFFSET IN_NAME

cli ;计时反跟踪
mov ah, 2ch
int 21h
push dx ;保存获取的秒和百分秒

CMP BYTE PTR [BX+2], 0DH ;如果是回车，则直接查询
JE LOOKUP
CMP BYTE PTR [BX+2], 'Q' ;如果是 q 就退出
JE EXIT

mov ah, 2ch ;获取第二次秒与百分秒
int 21h
sti
cmp dx, [esp] ;计时是否相同
pop dx
jz OK1 ;如果计时相同，通过这次反跟踪
jmp OVER ;如果计时不同，则跳转到 OVER

OK1:
cli ;堆栈检查反跟踪
push p2 ;PASSWORD 的地址压栈

MOV DI, OFFSET IN_NAME ;指向输入的地址
ADD DI, 2
MOV SI, OFFSET BNAME ;指向原来匹配名字的地址
SUB SI, 1
SUB DI, 1
MOV CX, 0

LOPA:
ADD DI, 1
ADD SI, 1
ADD CX, 1
CMP CX, 8H
JE PASSWORD
MOV BL, BYTE PTR [SI] ;在 BL 中存入真实的姓名
CMP BL, BYTE PTR [DI] ;将输入值与真实姓名作比较
JE LOPA ;如果为真，继续执行下一个比较
JNE INPUTFAIL1 ;如不为真，则提示失败

pop ax
mov bx, [esp-2] ;把栈顶的地址（PASSWORD 的地址）取到
sti
jmp bx ;如果被跟踪，则不会转移到 PASSWORD
db 'i do not know!'

INPUTFAIL1:
MOV AUTH, 0
MOV DL, 0AH
```

汇编语言程序设计实验报告

```
MOV AH, 2H
INT 21H
    LEA DX, LOGFAIL1
MOV AH, 9
INT 21H
MOV DL, 0AH
MOV AH, 2H
INT 21H
JMP INNAME
```

PASSWORD:

```
LEA DX, PUTPASSWORD ;提示用户输入密码
MOV AH, 9H
INT 21H
MOV DL, 0AH
MOV AH, 2H
INT 21H
```

```
LEA DX, OFFSET IN_PWD
MOV AH, 10
INT 21H
MOV DL, 0AH
MOV AH, 2H
INT 21H
LEA BX, OFFSET IN_PWD
```

```
MOV DI, OFFSET IN_PWD ;指向输入的地址
ADD DI, 2
MOV SI, OFFSET BPASS ;指向原来匹配密码的地址
SUB SI, 1
SUB DI, 1
MOV CX, 0
```

LOPB:

```
ADD DI, 1
ADD SI, 1
ADD CX, 1
    CMP CX, 4H
JE SUCCESSFUL ;如果密码匹配成功，则输出成功信息
MOV BL, BYTE PTR [DI] ;存入用户输入
```

```
sub bl, 29h ;按照解密规则生成真实密码
mov CURRENT, bl
add bl, CURRENT
add bl, CURRENT
```

```
CMP BL, BYTE PTR [SI] ;输入密码和真是密码比较
JE LOPB ;若为真比较下一位
JNE INPUTFAIL2;若不为真则提示失败
```

INPUTFAIL2:

```
MOV AUTH, 0
MOV DL, 0AH
MOV AH, 2H
INT 21H
LEA DX, LOGFAIL2
MOV AH, 9
```

汇编语言程序设计实验报告

```
INT 21H
MOV DL, 0AH
MOV AH, 2H
INT 21H

dec PASSTIME      ;判断输入密码或者用户名错误次数是否达到三次
cmp PASSTIME, 0   ;若输入三次不正确则退出程序
JE LOOKUP
cmp PASSTIME, 1
je one
cmp PASSTIME, 2
je two
one:
    lea dx, offset ONETIME ;若只输错一次或者两次，则提示剩下输入次数
    mov ah, 9h
    int 21h
    mov dl, 0ah
    mov ah, 2h
    int 21h
    jmp PASSWORD
two:
    lea dx, offset TWOTIME
    mov ah, 9h
    int 21h
    mov dl, 0ah
    mov ah, 2h
    int 21h
    jmp PASSWORD

SUCCESSFUL:
MOV AUTH, 1
MOV DL, 0AH
MOV AH, 2H
INT 21H
LEA DX, LOGINREMINDE ;若成功则输出成功的提示信息
MOV AH, 9
INT 21H
MOV DL, 0AH
MOV AH, 2H
INT 21H
JMP LOOKUP

IFPASS:
mov bx, es:[1*4] ;检查中断矢量表是否被调试工具阻止修改或者恢复
Inc bx
jmp bx ;如果正常修改，这里跳转到 TESTINT（即正常进入 LOOKUP），否则不能正常跳转
db 'now, you see.'

LOOKUP:
MOV DX, OFFSET ITEMNAME ;提示输入要查找的货物
MOV AH, 9H
INT 21H
MOV DL, 0AH
MOV AH, 2H
INT 21H
LEA DX, IN_ITEM ;用户输入要查找的货物
MOV AH, 10
```

汇编语言程序设计实验报告

```
INT 21H
MOV BL, IN_ITEM+1
MOV BH, 0
MOV BYTE PTR IN_ITEM+2[BX], '$' ;在输入串尾部补上'$'
MOV DL, 0AH
MOV AH, 2H
INT 21H
```

```
MOV SI, OFFSET IN_ITEM
CMP BYTE PTR [SI+2], 0DH ;判断是否为回车，若是则回到输入用户名
JE INNAME
    MOV CX, N ;设定循环的总数
MOV DX, OFFSET GA1
```

NEXT:

```
MOV DI, OFFSET IN_ITEM
MOV BL, BYTE PTR [DI+1]
ADD BX, 1
ADD DI, 2
MOV SI, DX
SUB SI, 1
SUB DI, 1
```

LOPC:

```
ADD DI, 1
ADD SI, 1
SUB BL, 1
CMP BL, 0
JE FINDSUC
MOV AL, BYTE PTR [SI]
CMP AL, BYTE PTR [DI] ;逐个判断货物名称是否相符
JE LOPC
DEC CX
ADD DX, 21
CMP CX, 0
JNE NEXT ;若货物没有循环完，判断下一个货物
JE FINDFAIL ;若循环完了还没有找到，则查找失败
```

FINDFAIL:

```
LEA DX, NOTITEM ;如果没有找到，输出没有找到的错误消息
MOV AH, 9H
INT 21H
MOV DL, 0AH
MOV AH, 2H
INT 21H
JMP LOOKUP ;回到 LOOKUP 重新输入寻找
```

FINDSUC:

```
CMP AUTH, 1 ;判断登录状态，若是若已经登陆则显示该商品名称，若不是则计算推荐度
JE PRINT
MOV SI, DX
MOV AL, [SI+10] ;折扣
    MOV AH, 0
    MOV BX, [SI+13] ;销售价
```

```
xor bx, 'D' ;解密得到真实数值
```


汇编语言程序设计实验报告

```
MUL BX ;销售价乘以折扣
MOV BX, 10
MOV DX, 0
DIV BX ;销售价乘以折扣除以 10, 实际售价
MOV CX, AX
MOV AX, [SI]+11 ;进货价
MOV BX, 128
MUL BX ;进货价乘以 128
MOV DX, 0
DIV CX ;进货价乘以 128 除以销售价
MOV TEMP1, AX
MOV AX, [SI]+17 ;已售数量
MOV BX, 64
MUL BX ;已售数量乘以 64
MOV BX, [SI]+15 ;进货数量
MOV DX, 0
DIV BX ;已售数量乘以 64 除以进货数量
ADD AX, TEMP1 ;两部分推荐度相加, 存入 AX
CMP AX, 100
JGE LEVELA ;大于 100 输出 A
CMP AX, 50
JGE LEVELB ;大于 50 输出 B
CMP AX, 10
JGE LEVELC ;大于 10 输出 C
JMP LEVELF ;其他输出 F
```

LEVELA:

```
MOV DL, 41H
MOV AH, 2
INT 21H
MOV DL, 0AH
MOV AH, 2
INT 21H
JMP INNAME
```

LEVELB:

```
MOV DL, 42H
MOV AH, 2
INT 21H
MOV DL, 0AH
MOV AH, 2
INT 21H
JMP INNAME
```

LEVELC:

```
MOV DL, 43H
MOV AH, 2
INT 21H
MOV DL, 0AH
MOV AH, 2
INT 21H
JMP INNAME
```

LEVELF:

```
MOV DL, 46H
MOV AH, 2
```

汇编语言程序设计实验报告

```
INT 21H
MOV DL, 0AH
MOV AH, 2
INT 21H
JMP INNAME

PRINT:
    LEA DX, IN_ITEM+2
    MOV AH, 9
    INT 21H
    MOV DL, 0AH
    MOV AH, 2
    INT 21H
    JMP INNAME

NEWINT:
    iret
TESTINT:
    jmp LOOKUP

OVER:
    cli                                ;还原中断矢量
        mov ax, OLDINT1
        mov es:[1*4], ax
        mov ax, OLDINT1+2
        mov es:[1*4+2], ax
        mov ax, OLDINT3
        mov es:[3*4], ax
        mov ax, OLDINT3+2
        mov es:[3*4+2], ax
        sti

EXIT:
    MOV AH, 4CH
    INT 21H

CODE ENDS

END START
```

3.4.4 实验步骤

1. 录入源程序，并对源程序进行编译和连接，生成可执行文件。若有报错，可查阅手册解决报错。
2. 测试输入错误的密码三次，看是否能跳过登入环节，直接进入查找环节。
3. 查看数据段中密码和进货价，看是否被加密成密文，观察其特点。
4. 用调试工具去执行自己的程序，观察自己的反跟踪程序代码是否在调试工具中出现，单步执行程序，看是否能达到预计的反跟踪的效果。
5. 总结分析下运用的三种反跟踪方法（计时法，检查堆栈法，检查中断矢量表法）是如何达到反跟踪的效果的。

汇编语言程序设计实验报告

3.4.5 实验记录与分析

1. 在编译和连接的过程中没有出现报错。
2. 测试连续输错密码三次，看能否跳过登入环节直接进入查找环节

连续测试三次输入错误密码截图如下：

```
C:\>4
PLEASE INPUT THE NAME(INPUT ENTER TO LOOKUP, INPUT Q TO QUIT)
WL DING
PLEASE INPUT THE PASSWORD
ABD
YOUR PASSWORD IS WRONG!
YOU HAVE ONLY TWO TIME TO INPUT PASSWORD
PLEASE INPUT THE PASSWORD
ABD
YOUR PASSWORD IS WRONG!
YOU HAVE ONE TIME TO INPUT PASSWORD
PLEASE INPUT THE PASSWORD
ABD
YOUR PASSWORD IS WRONG!
PLEASE INPUT THE ITEM NAME(INPUT ENTER TO INPUT NAME)
```

图 3-51 连续输入三次错误密码测试

由上述截图可以看出，输入了三次错误的密码 ABD，而每次输入错误的之后，除了会告诉错误提示，还会告诉还剩下几次输入密码的机会，如果剩余机会为 0，则直接跳过登入环节到查找货物环节，所以该功能运行正常。

3. 查看数据段中密码和进货价，看是否被加密成密文，观察其特点。

运用 td 运行该程序，找到密码对应的位置，截图如下：

```
ds:01D0 57 4C 20 44 49 4E 47 00 WL DING
ds:01D8 00 00 48 4B 4E 50 45 4E HKNPEN
ds:01E0 00 00 00 00 00 00 00 0A
ds:01E8 67 00 38 00 46 00 19 00 g 8 F ↓
```

图 3-52 密码暗文在数据段中的截图

可以看到，密码被加密成了暗文，为 48H、4EH、50H，恰好对应了明文密码 ABC 以及加密规则 (X-29H)*3。

找到商品 PEN 的信息在数据段中对应的位置，截图如下：

```
ds:01D8 00 00 48 4B 4E 50 45 4E HKNPEN
ds:01E0 00 00 00 00 00 00 00 0A
ds:01E8 67 00 38 00 46 00 19 00 g 8 F ↓
ds:01F0 00 00 42 4F 4F 4B 00 00 BOOK
```

图 3-53 PEN 在数据段中的暗文

可以看出，PEN 的进货价在数据段中以暗文形式存储，暗文为 67H。其余数值均用明文存储，折扣、商品售价、商品数目、已售数目用明文表示，分别为：0AH、38H、46H、19H。

找到商品 BOOK 的信息在数据段中对应的位置，截图如下：

```
ds:01F0 00 00 42 4F 4F 4B 00 00 BOOK
ds:01F8 00 00 00 00 09 48 00 1E oH ▲
ds:0200 00 19 00 05 00 00 00 50 ↓ ♠ P
ds:0208 45 4E 43 49 4C 00 00 00 ENCIL
```

图 3-54 BOOK 在数据段中的暗文

汇编语言程序设计实验报告

可以看出, BOOK 的进货价在数据段中以暗文形式存储, 暗文为 48H。其余数值均用明文存储, 折扣、商品售价、商品数目、已售数目用明文表示, 分别为: 09H、1EH、19H、05H。

找到商品 PENCIL 的信息在数据段中对应的位置, 截图如下:

```
ds:0208 45 4E 43 49 4C 00 00 00 ENCIL
ds:0210 00 09 50 00 1E 00 19 00 oP ▲ ↓
ds:0218 05 00 00 00 F1 02 FC 00 ♠ ±0^n
ds:0220 00 00 00 00 00 00 00 00
```

图 3-55 PENCIL 在数据段中的暗文

可以看出, PENCIL 的进货价在数据段中以暗文形式存储, 暗文为 50H。其余数值均用明文存储, 折扣、商品售价、商品数目、已售数目用明文表示, 分别为: 09H、1EH、19H、05H。

由上述现象可以看出, PEN、BOOK 和 PENCIL 的进货价分别为 35、12、20, 它们都根据加密规则 $X \oplus 'D'$ 分别被加密成 67H、48H、50H, 在数据段中显示, 故加密成功。

4. 用调试工具去执行自己的程序, 观察自己的反跟踪程序代码是否在调试工具中出现, 单步执行程序, 看是否能达到预计的反跟踪的效果。

检查中断矢量表法反跟踪代码在 td 环境下显示如下:

```
[*] CPU 80486 es:0000 = 00220811
cs:0009 6726A104000000 mov ax,es:[00000004]
cs:0010 A35100 mov [0051],ax
cs:0013 6726A106000000 mov ax,es:[00000006]
cs:001A A35300 mov [0053],ax
cs:001D 6726A10C000000 mov ax,es:[0000000C]
cs:0024 A35500 mov [0055],ax
cs:0027 6726A10E000000 mov ax,es:[0000000E]
cs:002E A35700 mov [0057],ax
cs:0031 FA cli
cs:0032 B8ED02 mov ax,02ED
cs:0035 6726A304000000 mov es:[00000004],ax
cs:003C 67268C0D060000+mov es:[00000006],cs
cs:0044 6726A30C000000 mov es:[0000000C],ax
```

图 3-56 检查中断矢量表法反跟踪代码在 td 下的显示 1

```
2 8A1C mov bl,[si]
4 3A1D cmp bl,[di]
```

图 3-57 检查中断矢量表法反跟踪代码在 td 下的显示 2

可以看出, 图 1 表示 1 号中断和 3 号中断的原地址已经备份存储到内存中, 并且已经修改了 1 号中断和 3 号中断的入口地址为预设值, 图 2 位检查入口地址是否真的改为预设值, 以确定没有 td 的存在, 可以看出, 反跟踪代码装载成功。

计时法反跟踪代码在 td 环境下显示如下:

汇编语言程序设计实验报告

```

I-CPU 80486
cs:0078 FA      cli
cs:0079 B42C     mov     ah,2C
cs:007B CD21     int      21
cs:007D 52       push    dx
cs:007E 807F020D cmp     byte ptr [bx+02]
cs:0082 0F844C01 je      01D2
cs:0086 807F0251 cmp     byte ptr [bx+02]
cs:008A 0F848D02 je      031B
cs:008E B42C     mov     ah,2C
cs:0090 CD21     int      21
cs:0092 FB       sti
cs:0093 673B1424 cmp     dx,[esp]
cs:0097 5A       pop     dx

```

图 3-58 计时法反跟踪代码在 td 下的显示

可以看出，这里有两次获取本机时间的操作，并且将时间差与标准计时函数的运行时间进行比较，若相同则能确定没有 td 的存在；否则有 td 的存在，反跟踪代码装载成功。

检查堆栈法反跟踪代码在 td 环境下显示如下：

```

cs:009D FA      cli
cs:009E FF364E00 push    word ptr [004E]
cs:00A2 FF364E00 push    word ptr [004E]

```

图 3-59 检查堆栈法反跟踪代码在 td 下的显示 1

```

cs:00CA 58       pop     ax
cs:00CB 678B5C24FE mov    bx,[esp-02]
cs:00D0 FB       sti
cs:00D1 FFE3     jmp     bx

```

图 3-60 检查堆栈法反跟踪代码在 td 下的显示 2

图一可以看出将地址 004E（验证密码模块的首地址）压入栈中，如果没有 td 存在，栈顶的值不会改变，若有 td 存在则会改变。之后出栈将地址放入 bx，然后再 jmp 到 bx 中。若没有 td 存在，则能正常执行，否则不能正常进行模块跳转，反跟踪代码装载成功。

检验单步执行时是否能达到预定的反跟踪效果：

从程序开始的时候开始单步执行，首先遇到的是检查中断矢量反跟踪，发现运行到比较修改的地址是否等于预先设置的修改地址时，程序按照预想退出；重新用 td 执行，跳过检查中断矢量反跟踪，遇到的是计时反跟踪，发现遇到比较时间的语句前后程序按照预想退出；再重启 td，设置断点跳过前面的检查中断矢量反跟踪以及计时反跟踪语句，最后到了检查对战法反跟踪语句，单步执行，发现到最后 jmp bx 的时候，无法正常跳转到输入密码模块。故设计的反跟踪都是有效的。

5. 分析总结程序中运用的三种反跟踪方法是如何达到反跟踪效果的

计时法反跟踪：在运用 td 调试（人工单步调试）的时候，指令间的间隔一般都会比较大，容易达到秒级。而常用的计时程序的精度是 55ms，通过计时执行指令的时间差，和 55ms 进行比较，若时间差不为 55ms，则可以判断存在跟踪。

汇编语言程序设计实验报告

检查堆栈法反跟踪：动态调试（td）的过程中，会产生单步中断，该中断响应会使用被调试程序的堆栈，修改栈顶以上几个字存储区的内容。所以我们可以预先在程序中执行压栈出栈操作，即可设定栈顶以上几个字的内容，然后我们再获取栈顶以上几个字的内容，判断是否为先前压栈时的内容，即可判断是否被调试（有 td 运行）。

检查中断矢量表法反跟踪：在程序中修改中断矢量表中 1 号和 3 号中断服务程序的入口地址。如果是在 td 环境下，调试工具会阻止你修改中断矢量表，然后我们只要判断中断矢量表对应的位置是否改成了我们设置的地址，若不是，则证明有调试工具存在（阻止你修改中断矢量表）。

3.5 任务 5

3.5.1 设计思想及存储单元分配

1) 设计思想

本次实验运用 td 进行单步调试，跟踪破解同组同学的加密程序。

浏览 td 中的代码，若遇到可能是反跟踪的代码段，但发现代码段中有有用的功能代码，则在其尾部设置断点，运行过去；若遇到可能是反跟踪的代码段，但该段中无有用的功能代码，则可以直接修改代码段的 IP，跳过该段的执行。

在计算推荐度模块的时候，查看 td 中的代码，找出加密方式，然后查看数据段的数据，结合加密方式，破解进货价的数值。

2) 存储单元分配

由于该实验是破解同组同学的加密程序，故无存储单元的分配信息。

3.5.2 流程图

破解方法的流程图如下所示：

汇编语言程序设计实验报告

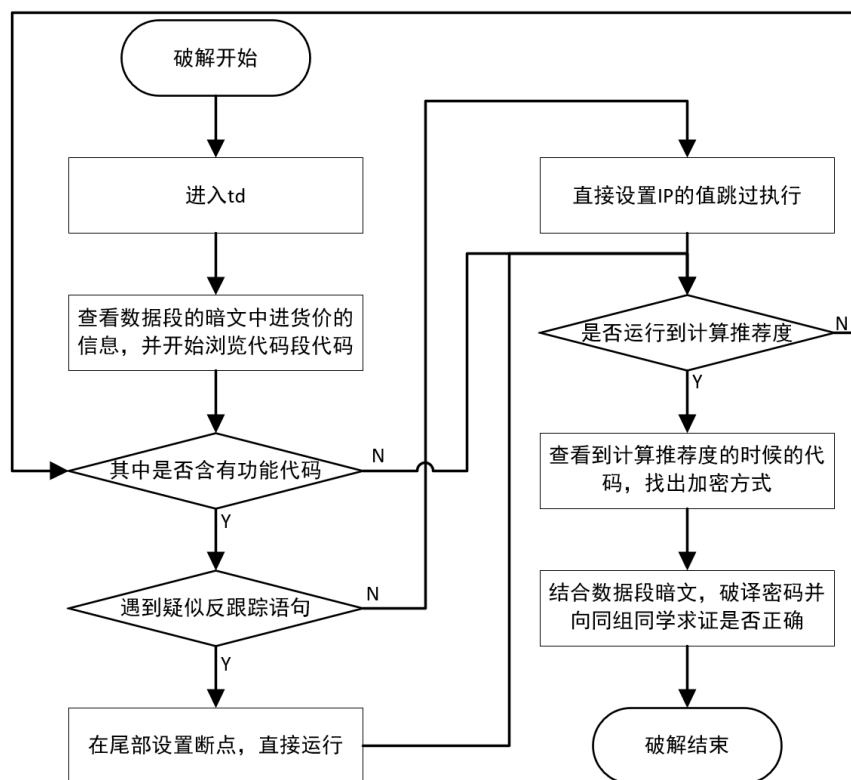


图 3-61 破解方法总流程图

3.5.3 实验步骤

1. 用 td 与运行同组同学的加密程序，开始解密。
2. 利用设计思想里面的方法，跳过疑似的反跟踪段，即对于疑似反跟踪段，若其中含有有用的功能代码，则在尾部设置断点，将程序运行过去；若没有功能代码，则直接修改 IP 运行过去。
3. 在计算推荐度模块中查看 td 中的代码，找出加密方式，结合数据段中的密文，破解进货价的数值。
4. 向同组同学验证解密出的进货价数值是否正确。
5. 描述如何对密码实现快速的暴力破解。
6. 描述总结如何观察到程序中存在反跟踪代码，如何应对反跟踪程序（解密程序）。

3.5.4 实验记录与分析

1. 用 td 运行同组同学的加密程序，实现解密，获取其商店中进货价的数值

- 1) 查看数段的内容，查找进货价的暗文。

下图是 PEN 存在数据段中的暗文：

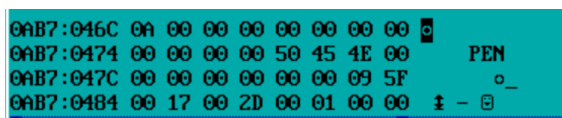


图 3-62 PEN 在数据段中的暗文

汇编语言程序设计实验报告

可以看出，PEN 的进货价在数据段中以暗文形式存储，暗文为 5FH。其余数值均用明文存储，折扣、商品售价、商品数目、已售数目用明文表示，分别为：09H、17H、2DH、01H。

下图是商品 BAG 在数据段中的暗文：

```
0AB7:048C 00 42 41 47 00 00 00 00 BAG
0AB7:0494 00 00 00 08 5F 00 22 00 "
0AB7:049C 36 00 08 00 00 00 43 4F 6 6 CO
0AB7:04A4 4F 4B 49 45 00 00 00 00 OKIE
```

图 3-63 BAG 在数据段中的暗文

可以看出，BAG 的进货价在数据段中以暗文形式存储，暗文为 5FH。其余数值均用明文存储，折扣、商品售价、商品数目、已售数目用明文表示，分别为：08H、22H、36H、08H。

下图是商品 COOKIE 在数据段中的暗文：

```
0AB7:049C 36 00 08 00 00 00 43 4F 6 6 CO
0AB7:04A4 4F 4B 49 45 00 00 00 00 OKIE
0AB7:04AC 02 5E 00 2A 00 06 00 03 03 * + ♥
0AB7:04B4 00 00 00 0A 00 00 00 00 00
```

图 3-64 COOKIE 在数据段中的暗文

可以看出，PEN 的进货价在数据段中以暗文形式存储，暗文为 5EH。其余数值均用明文存储，折扣、商品售价、商品数目、已售数目用明文表示，分别为：02H、2AH、06H、03H。

2) 在代码段中单步执行代码，按照实验步骤中的方法跳过反跟踪代码

下图为遇到的疑似检查中断矢量反跟踪的代码，截图如下：

```
B20:010E 6726A104000000 mov ax,es:[00000004]
B20:0115 39067F03 cmp [037F],ax
B20:0119 7529 jne 0144
B20:011B 6726A106000000 mov ax,es:[00000006]
B20:0122 39068103 cmp [0381],ax
B20:0126 751C jne 0144
B20:0128 6726A10C000000 mov ax,es:[0000000C]
B20:012F 39068303 cmp [0383],ax
B20:0133 750F jne 0144
B20:0135 6726A10E000000 mov ax,es:[0000000E]
B20:013C 39068503 cmp [0385],ax
B20:0140 7502 jne 0144
```

图 3-65 疑似检查中断矢量反跟踪的代码

通过检查发现这一段中没有有用的功能代码，我们只需要改变 IP 为 0140 跳过这一段即可。

接着向下单步执行，遇到疑似计时反跟踪法代码，因为有两处获取当前时间的指令并且有比较的语句，截图如下：

```
[J]-CPU 80486
cs:0606 FA cli
cs:0607 B42C mov ah,2C
cs:0609 CD21 int 21
cs:060B 52 push dx
cs:060C 803E610100 cmp byte ptr [0161],
cs:0611 7497 je 05AA
cs:0613 B42C mov ah,2C
cs:0615 CD21 int 21
cs:0617 FB sti
cs:0618 673B1424 cmp dx,[esp]
cs:061C 5A pop dx
cs:061D 0F85BF02 jne 06E0
cs:0621 FA cli
```

图 3-66 疑似计时反跟踪的代码

汇编语言程序设计实验报告

但是我们发现这一段代码中有两条指令，为 0611 和 0613 对应的指令，是判断登录姓名时的功能语句，所以这一段不能修改 IP 直接跳过，我们在 0621 处设置断点，将这一段运行过去，截图如下：

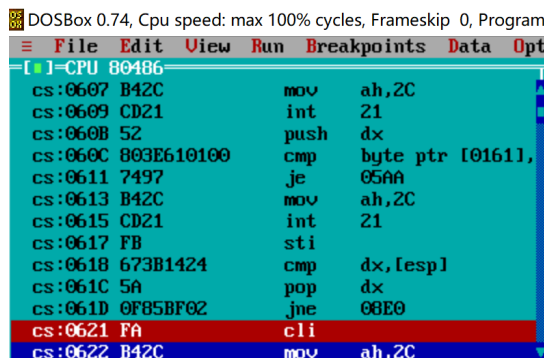


图 3-67 设置断点将计时反跟踪直接运行过去

3) 找到推荐度计算模块，根据其中的线索推出加密方式

在计算推荐度的时候，程序设计者一定会将暗文翻译成明文进行计算，通过这一点，我们可以在计算推荐度模块寻找加密线索。

在单步执行到计算推荐度模块的时候，寻找到的解密线索如下：

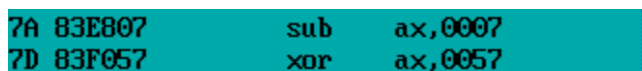


图 3-68 计算推荐度模块时的解密线索

由这个解密线索可以看出，在把暗文还原成明文的时候，首先将 ax 减去 7H，然后再将 ax 与 0057H，即字母 ‘W’ 作异或运算，所以我们可以推出对于暗文进价 X 的解密算法为： $(X-7H) \text{ XOR } 0057H$ 。根据这个算法我们分别推得 PEN、BAG 和 COOKIE 的进货价分别为：1h、1h 和 2h。

4) 找同组同学验证这一结论

数据段中有关截图如下：

```
COMMOD1 DB 'PEN', 7 DUP(0), 9; 商品名称 0 商品折扣 10
          DW (1+7) XOR 'W', 23, 45, 1, ?; 商品进价 11 商品售价 13 商品数目 15 已售数目 17 推荐度 19
          DB 'BAG', 7 DUP(0), 8
          DW (1+7) XOR 'W', 34, 54, 8, ?
          DB 'COOKIE', 4 DUP(0), 2
          DW (2+7) XOR 'W', 42, 6, 3, ?
```

图 3-69 对于解密结果进行验证

可以发现，进货价分别为 1h、1h 和 2h，即解密结果正确，而由代码看出加密方程为：

$(X+7H) \text{ XOR } 0057H$ ，故解密方程为 $(X-7H) \text{ XOR } 0057H$ ，故解密方程也推导正确

2. 描述如何对密码实现快速的暴力破解

若密码以明文的形式存入在待破解程序中，则可以在数据段中直接查看疑似为密码的数据，然后实际验证；也可以用文件编辑工具打开待破解文件查看内容，找到疑似内容，再实际验证。

若密码以暗文的形式存入在带哦你姐程序中，首先我们在数据段中找到密码对应的暗文，然后在判断密码的模块中找到加密的方式，然后结合暗文和加密方式，一个个试验，得到明文数据，

汇编语言程序设计实验报告

进行求解验证。（这里需编写脚本进行测试，减少人工试验时间）

3. 描述总结如何观察到程序中存在反跟踪代码，如何应对反跟踪程序（解密程序）。

a) 对于三种常用的反汇编方法，给出具体观察的描述：

1. 通过计时的方法反跟踪：先浏览 td 中的指令，若发现有多次获取时间指令出现，还对其获取的时间进行了比较，可以初步判定这里有计时反跟踪，我们可以设置断点跳过这一段。

2. 通过检查堆栈的方法反跟踪：根据其特点，若程序中压栈了一个地址，并且后来将这个地址出栈并且跳转到这里，可以初步判断这里有个检查堆栈的反跟踪，其目的是为了看这个地址改变了没有，判断这里是否有反跟踪的情况。

3. 通过检查中断矢量来反跟踪：根据这种反跟踪的特点，若程序中出现了大量且秘籍的对于中断矢量的备份保存以及重新设置的语句，我们可以预判这里有一个检查中断矢量反跟踪，我们可以接着向后面浏览，如果有比较中断矢量入口地址的语句出现，我们基本上就可以判定这里有一个检查中断矢量的反跟踪。

b) 应对反跟踪的方法：

1. 查看疑似反跟踪代码段里面是否有与程序功能有关的功能代码，若有，则在这一段的段尾设置断点，按 F9 直接运行过去。

2. 若意思反跟踪代码段里面没有与程序功能相关的代码，则我们可以直接设置新的 IP 的值，将这一段直接跳过不执行。

4 总结与体会

通过这次实验，我对于程序的中断、驻留、跟踪以及反跟踪等知识有了更深入的了解，也对于 dosbox 虚拟机有了更深入的了解，比如 dosbox 虚拟机在实在模式下运行但是展现的是保护模式的代码，在 dosbox 虚拟机对程序进行接管并不能影响计算机系统和其他虚拟机，调用的底层命令也只能在虚拟机内部使用等等。

任务一主要是运用 td 查看中断矢量表，先开始一直没弄懂为什么在 td 中直接查看和编程查看的数值是不等的，后来经过询问老师以及和同学的讨论得知，td 对于一些中断矢量入口是有保护的，所以我们看到的是被 td 保护的中断矢量入口值，这也就解释了为什么编程查看和 td 直接查看的不同。

任务二让我们对 dosbox 虚拟机有了更深入了解，我们发现我们编写的程序只在当前虚拟机环境下有效，但在电脑实际系统、其它环境甚至 td 环境下都没有效果，所以我们可以得出结论，我们编写的接管程序实际上只是在 dosbox 虚拟机内做了一个映射，这个映射只能影响当前虚拟机，而对于其它环境无效。

任务三主要是让我们了解 CMOS 内部存储单元的作用。

任务四和任务五让我们对于数据段信息加密有了基本的认识，也对于跟踪与反跟踪有了更加深刻的理解。我们学会了编写检查中断矢量表法、计时法、检查堆栈法等方法来进行反跟踪，防止他人破获自己的密文；同时也学会了预估反跟踪代码，结合设置断点和跳转 IP 的方式，来对一个有反跟踪代码的程序进行解密。解密过程中最重要的是要细致寻找解密信息与反跟踪代码，正确设置断点，需要十分有耐心。

汇 编 语 言 程 序 设 计 实 验 报 告

参考文献

- [1] 王元珍 曹忠升 韩宗芬 编著. 80X86 汇编语言程序设计. 武汉: 华中科技大学出版社, 2005 年 4 月. 214 页-220 页
- [2] URL: https://blog.csdn.net/qq_28877125/article/details/72783433