

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实验**

**专业班级：计算机科学与技术1706班**

**学 号： U201713321**

**姓 名： 袁杰威**

**指导教师： 金良海**

**实验时段： 2020年4月20日~4月25日**

**实验地点： 南一楼804室**

**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：

报告日期：2020.5.10

实验报告成绩评定：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 |
| 实验完成质量（70%），报告撰写质量（30%），每次满分20分。 |  |  |  |  |  |
| 合计（100分） |  | | | | |

备注：实验完成质量从实验目的达成程度，设计方案、实验方法步骤、实验记录与结果分析论述清楚等方面评价；报告撰写质量从撰写规范、完整、通顺、详实等方面评价。

指导教师签字：

                    日期：

**目录**

[课程总体说明 - 2 -](#_Toc35182379)

[0.1 课程目标 - 2 -](#_Toc35182380)

[0.2 成绩构成 - 2 -](#_Toc35182381)

[0.3 实验任务的总体描述 - 2 -](#_Toc35182382)

[1 编程基础 1](#_Toc35182383)

[1.1 实验目的与要求 1](#_Toc35182384)

[1.2 实验内容 1](#_Toc35182385)

[1.3 任务1.1实验过程 1](#_Toc35182386)

[1.3.1 实验方法说明 1](#_Toc35182387)

[1.3.2 实验记录与分析 1](#_Toc35182388)

[1.4 任务1.2的实验过程 2](#_Toc35182389)

[1.4.1 设计思想及存储单元分配 2](#_Toc35182390)

[1.4.2 流程图 2](#_Toc35182391)

[1.4.3 源程序 3](#_Toc35182392)

[1.4.4 实验步骤 4](#_Toc35182393)

[1.4.5 实验记录与分析 4](#_Toc35182394)

[1.5 小结 5](#_Toc35182395)

[1.5.1 主要收获 6](#_Toc35182396)

[1.5.2 主要看法 6](#_Toc35182397)

[2 程序优化 7](#_Toc35182398)

[2.1 实验目的与要求 7](#_Toc35182399)

[2.2 实验内容 7](#_Toc35182400)

[2.3 任务2.1实验过程 7](#_Toc35182401)

[2.3.1 实验方法说明 7](#_Toc35182402)

[2.3.2 实验记录与分析 7](#_Toc35182403)

[2.4 任务2.2实验过程 7](#_Toc35182404)

[2.4.1 实验方法说明 7](#_Toc35182405)

[2.4.2 实验记录与分析 7](#_Toc35182406)

[2.5 小结 7](#_Toc35182407)

[2.5.1 主要收获 7](#_Toc35182408)

[2.5.2 主要看法 7](#_Toc35182409)

[3 模块化程序设计 7](#_Toc35182410)

[3.1 实验目的与要求 7](#_Toc35182411)

[3.2 实验内容 7](#_Toc35182412)

[。。。。。。。。 7](#_Toc35182413)

[4 中断与反跟踪 8](#_Toc35182414)

[4.1 实验目的与要求 8](#_Toc35182415)

[4.2 实验内容 8](#_Toc35182416)

[。。。。。。。。 8](#_Toc35182417)

[5 WIN32程序设计 8](#_Toc35182418)

[5.1 实验目的与要求 8](#_Toc35182419)

[5.2 实验内容 8](#_Toc35182420)

[。。。。。。。。 8](#_Toc35182421)

[参考文献 9](#_Toc35182422)

# 课程总体说明

## 0.1 课程目标

下表是本课程的目标及与支撑的毕业要求指标点之间的关系。请大家关注下表中最后一列“实验中的注意事项”的内容，以便更有针对性的满足课程目标的要求。

|  |  |  |
| --- | --- | --- |
| 课程目标 | 支撑的毕业要求指标点 | 实验中的注意事项 |
| 掌握汇编语言程序设计的全周期、全流程的基本方法与技术，通过程序调试、数据记录和分析，了解影响设计目标和技术方案的多种因素。 | 3.1掌握与计算机复杂工程问题有关的工程设计和软硬件产品开发全周期、全流程的基本设计/开发方法和技术，了解影响设计目标和技术方案的多种因素。 | 不能只写代码完成功能，还要有设计、调试、记录、分析等部分的内容。 |
| 掌握编写、调试汇编语言程序的基本方法与技术，能根据实验任务要求,设计出较充分利用了汇编语言优势的软件功能部件或软件系统。 | 3.2能为计算机复杂工程问题解决方案设计满足特定需求的软/硬件模块。 | 要思考与运用汇编语言的优势编写某些程序。 |
| 熟悉支持汇编语言开发、调试以及软件反汇编的主流工具的功能、特点与局限性及使用方法。 | 5.1了解计算机专业常用的现代仪器、信息技术工具、工程工具和模拟软件的使用原理和方法，并理解其局限性。 | 熟悉实验中使用的工具，把对工具的看法记录在案。 |

## 0.2 成绩构成

实验课程综合成绩由实验过程成绩和实验报告成绩二部分构成。**实验过程成绩**：30%。主要考察各实验完成过程中的情况，希望大家做到预习准备充分，操作认真熟练，在规定的时间内完成实验任务，结果正确，积极发现和提出问题，交流讨论时描述问题准确、清晰。实验报告成绩：70%。主要考核报告体现的实验完成质量(含问题的分析、设计思想与程序、针对问题的实验方法与步骤、实验记录、实验结果分析等方面)和报告格式规范等撰写质量方面的内容。

## 0.3 实验任务的总体描述

本课程安排了8次4学时的课内实验课时，将实现一个具有一定复杂程度的系统。对该系统的相关要求被划分成了**5个主题**：1）搭建原型系统；2）在原型系统基础上探索程序指令级别的优化；3）通过模块化调整与优化原型系统的程序结构；4）通过中断、内存数据和地址操纵、跟踪与反跟踪、加密等措施增强系统安全性；5）程序在不同平台上的移植。

针对这5个主题，对应地布置了5次实验。**实验1（编程基础）**安排8个课内学时熟悉汇编语言程序设计的基本方法、技术与工具，设计实现指定原型系统的主要功能。针对原型系统的搭建，实验报告中要有全周期、全流程的描述。**实验2（程序优化）**安排4个课内学时探索如何通过选择不同的指令及组合关系来优化程序的性能或代码长度。**实验3（模块化程序设计）**安排8个课内学时，利用子程序、模块化程序设计方法、与C语言混合编程等，调整与优化程序结构。**实验4（中断与反跟踪）**安排8个课内学时，通过利用中断机制、内存数据和地址操纵技术、跟踪与反跟踪技巧、加密等措施增强系统安全性。**实验5（WIN32程序设计）**安排4个课内学时，熟悉在不同操作系统平台上移植实现已有系统功能的基本方法。每次实验的侧重面有所不同，但都会涉及到课程目标的三个方面，因此，需要大家在实验过程中以及实验报告中有所注意和体现。

**本次课程所涉及的原型系统是一个网店商品信息管理系统。下面描述该系统的基本需求，后续每次实验都是以这个基本需求为背景而展开的。**

有一个老板在网上开了1个网店SHOP，网店里有n种商品销售。每种商品的信息包括：商品名称（最长名称9个字节，其后加一个数值0表示名称结束），折扣（字节类型，取值0~10；0表示免费赠送，10表示不打折，1~9为折扣率；实际销售价格=销售价\*折扣/10），进货价(字类型)，销售价（字类型），进货总数（字类型），已售数量（字类型），推荐度【=（进货价/实际销售价格+已售数量/（2\*进货数量））\*128，字类型】。老板管理网店信息时需要输入自己的名字（最长名字9个字节，其后加一个数值0表示结束）和密码（最长密码6个字节，其后加一个数值0表示结束），老板登录后可查看商品的全部信息；顾客（无需登录）可以查看网店中每个商品除了进货价以外的信息，可以对指定商品下单预定。

该系统被执行后，首先显示一个菜单界面，菜单界面信息包括：

当前用户名：（老板名称或顾客）

当前浏览商品名称：（没有时空缺）

请输入数字1…9选择功能：

1.登录/重新登录

2.查找指定商品并显示其信息

3.下订单

4.计算商品推荐度

5.排名

6.修改商品信息

7.迁移商店运行环境

8.显示当前代码段首址

9.退出

当用户输入某一个有效数字后，就进入到指定的功能中执行，执行完之后再回到该菜单界面。如果选择的是退出功能，则程序退出。该菜单中每项菜单的具体功能要求详见每次的实验任务描述。

# 编程基础

## 实验目的与要求

本次实验的主要目的与要求有下面6点，所有的任务都会围绕这6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. 掌握汇编源程序编辑工具、汇编程序、连接程序、调试工具TD的使用；
2. 理解数、符号、寻址方式等在计算机内的表现形式；
3. 理解指令执行与标志位改变之间的关系；
4. 熟悉常用的DOS功能调用；
5. 熟悉分支、循环程序的结构及控制方法，掌握分支、循环程序的调试方法；
6. 加深对转移指令及一些常用的汇编指令的理解。

## 实验内容

**任务1.1**：《80X86汇编语言程序设计》教材中 P31的 1.14题。要求：

(1) 直接在TD中输入指令，完成两个数的求和、求差的功能。求和/差后的结果放在(AH)中。

(2) 请事先指出执行指令后(AH)、标志位 SF、OF、CF、ZF的内容。

(3) 记录上机执行后的结果，与（2）中对应的内容比较。

**任务1.2：**《80X86汇编语言程序设计》教材中 P45的 2.3题。要求：

（1）分别记录执行到“MOV CX，10”和“INT 21H”之前的(BX),(BP), (SI), (DI)各是多少。

（2）记录程序执行到退出之前数据段开始40个字节的内容，指出程序运行结果是否与设想的一致。

**任务1.3：**《80X86汇编语言程序设计》教材中 P45的 2.4题的改写。要求：

(1) 实现的功能不变，但对数据段中变量访问时所用到的变址寄存器采用32位寄存器。

(2) 记录程序执行到退出之前数据段开始40个字节的内容，检查程序运行结果是否与设想的一致。

(3)在TD代码窗口中观察并记录机器指令代码在内存中的存放形式，并与TD中提供的反汇编语句及自己编写的源程序语句进行对照，也与任务1.2做对比。（相似语句记录一条即可，重点理解机器码与汇编语句的对应关系，尤其注意操作数寻址方式的编码形式，比如寄存器间接寻址、变址寻址、32位寄存器与16位寄存器编码的不同、段前缀在代码里是如何表示的等）。

（4）观察连续存放的二进制串在反汇编成汇编语言语句时，从不同字节位置开始反汇编，结果怎样？理解 IP/EIP指明指令起始位置的重要性。

## 任务1.1实验过程

### 实验方法说明

1. 准备上机实验环境，对实验用到的软件进行安装、运行，通过试用初步了解软件的基本功能、操作等。

2. 在TD的代码窗口中的当前光标下输入第一个运算式对应的两个8位数值对应的指令语句MOV AH,01001101B；MOV AL,-01110010B；ADD AH,AL；观察代码区显示的内容与自己输入字符之间的关系；然后确定CS:IP指向的是自己输入的第一条指令的位置，单步执行三次，观察寄存器内容的变化，记录标志寄存器的结果。

1. 预计执行ADD之后：。（AH）=0DBH、SF=1、OF=0、CF=0、ZF=0输入MOV AH,10H；MOV AL,-5H；SUB AH,AL；观察标志位特点；输入MOV AH,0FFH；MOV AL,-5H；SUB AH,AL；观察标志位特点。

实验预测如下

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | AH | SF | OF | CF | ZF |
| 1 | 8DH | 1 | 1 | 0 | 0 |
| 2 | 7AH | 0 | 1 | 1 | 0 |
| 3 | 08H | 0 | 0 | 1 | 0 |

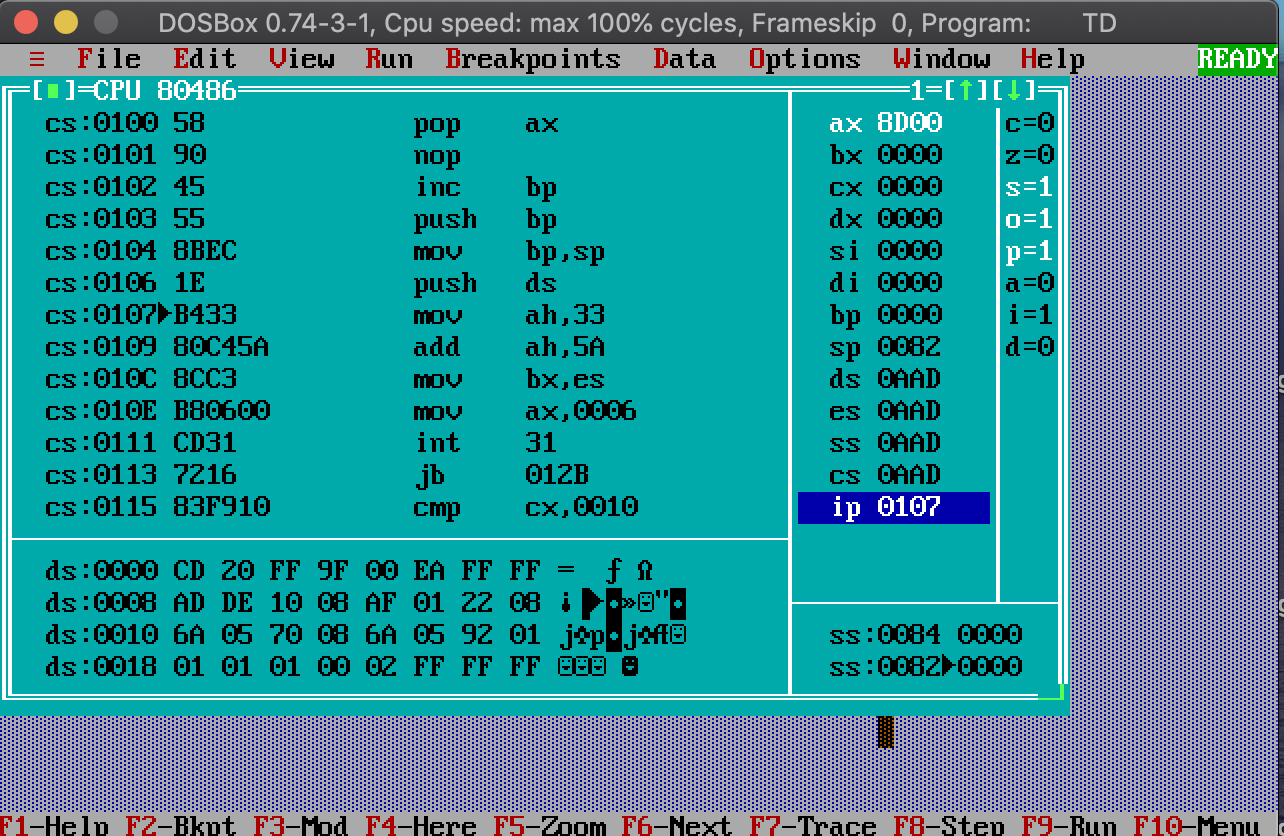
### 实验记录与分析

1. 实验环境条件：intel i5-7360U 2.3GHz，8G内存；MacOS下DOSBox0.74； TD.EXE 5.0。

2. 第（1）小题，x1=+0110011B,x2=+1011010B。

在TD之间输入指令 MOV AH,00110011B; ADD AH,01011010B,F7进行单步调试。

调试结果如图1.1所示。可得ax即AH=8DH,CF=0,ZF=0,SF=1,OF=1。与预测相同。

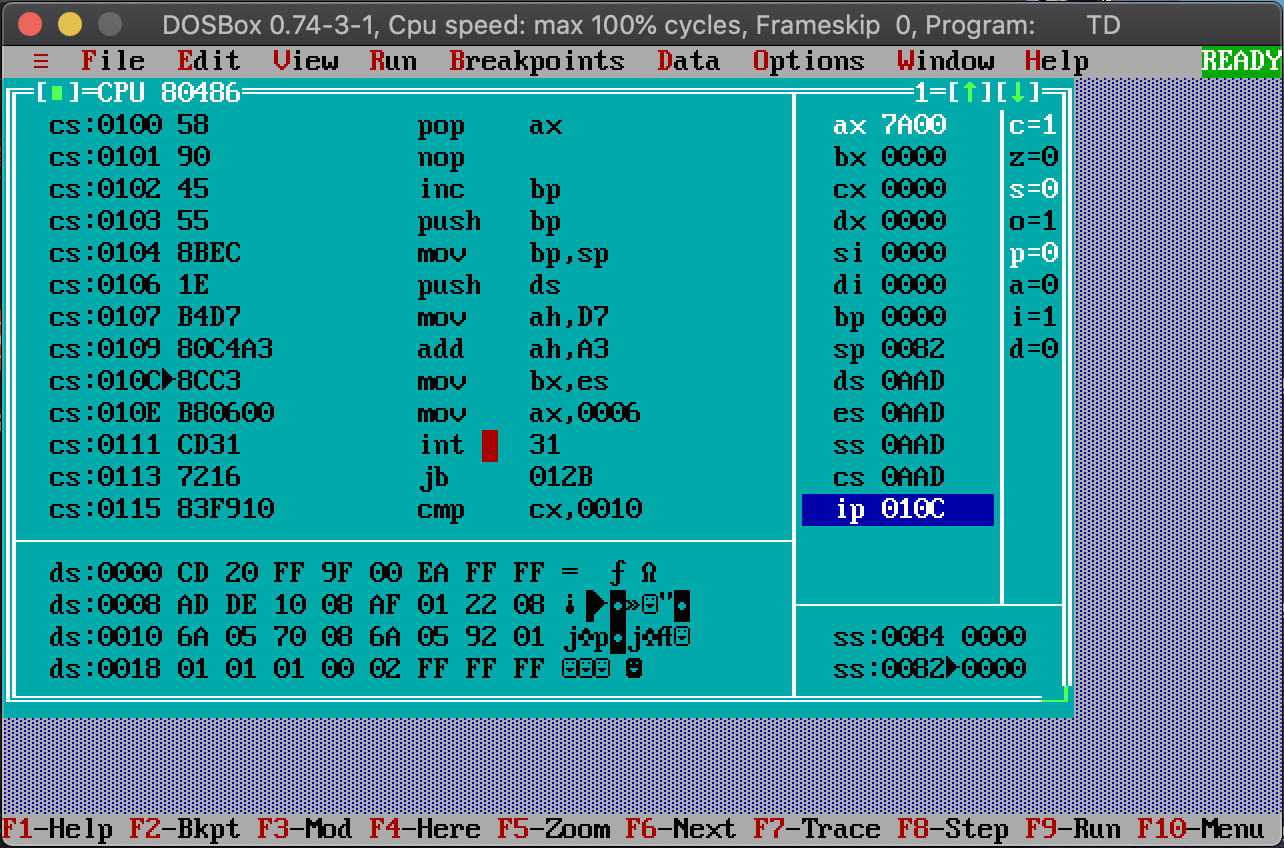


* 1. 任务1第一小题测试

1. 第（2）小题，x1=-0101001B,x2=-1011101B。

在TD之间输入指令 MOV AH,11010111 B; ADD AH,10100011B,F7进行单步调试。

调试结果如图1.1所示。可得ax即AH=7AH,CF=1,ZF=0,SF=0,OF=1。与预测相同。

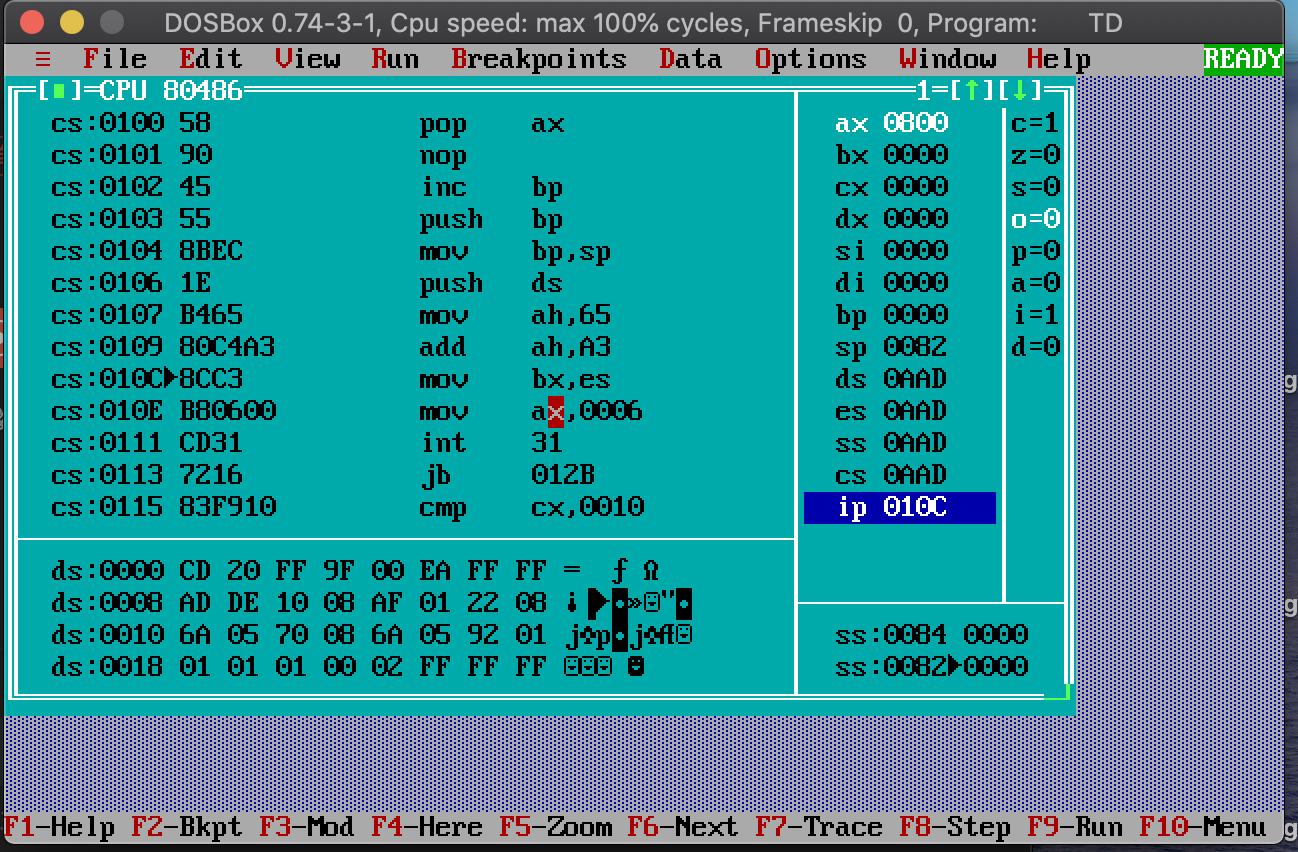


1.2任务1第二小题测试

1. 第（3）小题，x1=+1100101B,x2=-1011101B。

在TD之间输入指令 MOV AH,01100101 B; ADD AH,10100011B,F7进行单步调试。

调试结果如图1.1所示。可得ax即AH=08H,CF=1,ZF=0,SF=0,OF=0。与预测相同。



1.3 任务1第三小题测试

## 任务1.2的实验过程

## 1.4.1 实验方法说明

1. 用记事本将P45 2.3题的代码编写完成，并以.ASM格式保存，命名为45.ASM。
2. 在DOSBox中使用masm汇编源代码，生成45.OBJ文件。
3. 在DOSBox中使用LINK连接45.OBJ，生成可执行文件45.EXE。
4. 运行TD并打开45.EXE文件。

表1.2 任务2预计结果（1）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | BX | BP | SI | DI |
| MOV CX 10 | 0014H | 001EH | 0000H | 000AH |
| INT 21H | 001EH | 0028H | 000AH | 0014H |

表1.3 任务2预计结果（2）

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | DS段前40字节内容 | | | | | | | |
| 程序结束前 | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
| 08 | 09 | 00 | 01 | 02 | 03 | 04 | 05 |
| 06 | 07 | 08 | 09 | 01 | 02 | 03 | 04 |
| 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D |

### 1.4.2 实验记录与分析

1. 实验环境条件：intel i5-7360U 2.3GHz，8G内存；MacOS下DOSBox0.74； TD.EXE 5.0。
2. 汇编源程序

3. 连接过程没有发生异常。

4. 单步调试，在执行MOV CX 10之前，各寄存器值如1.5图所示，BX=0014H，BP=0014H，BP=001EH，SI=0000H，DI=000AH。符合预测值

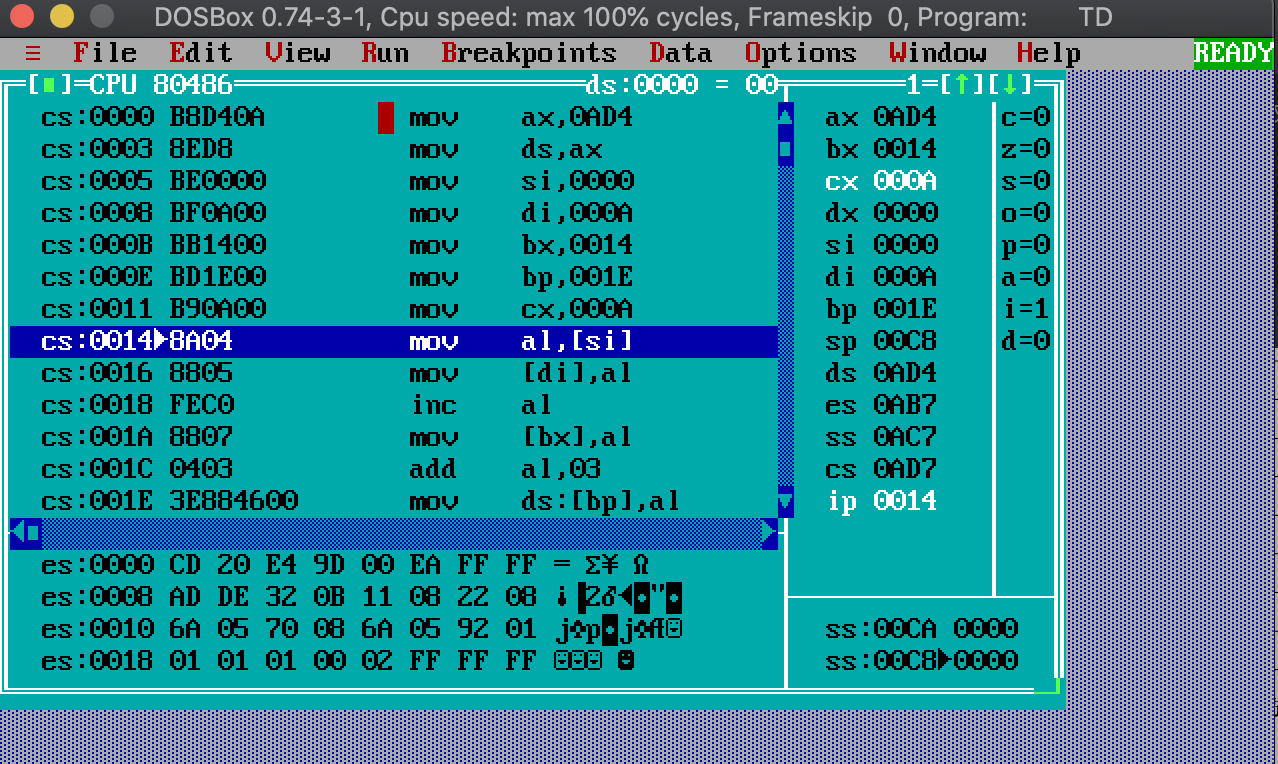


图1.4 执行MOV CX 10前情况

1. 单步调试，在执行INT 21H之前，各值如1.6图所示，BX=001EH,BP=0028H,SI=000AH,DI=0014H。符合预期值。

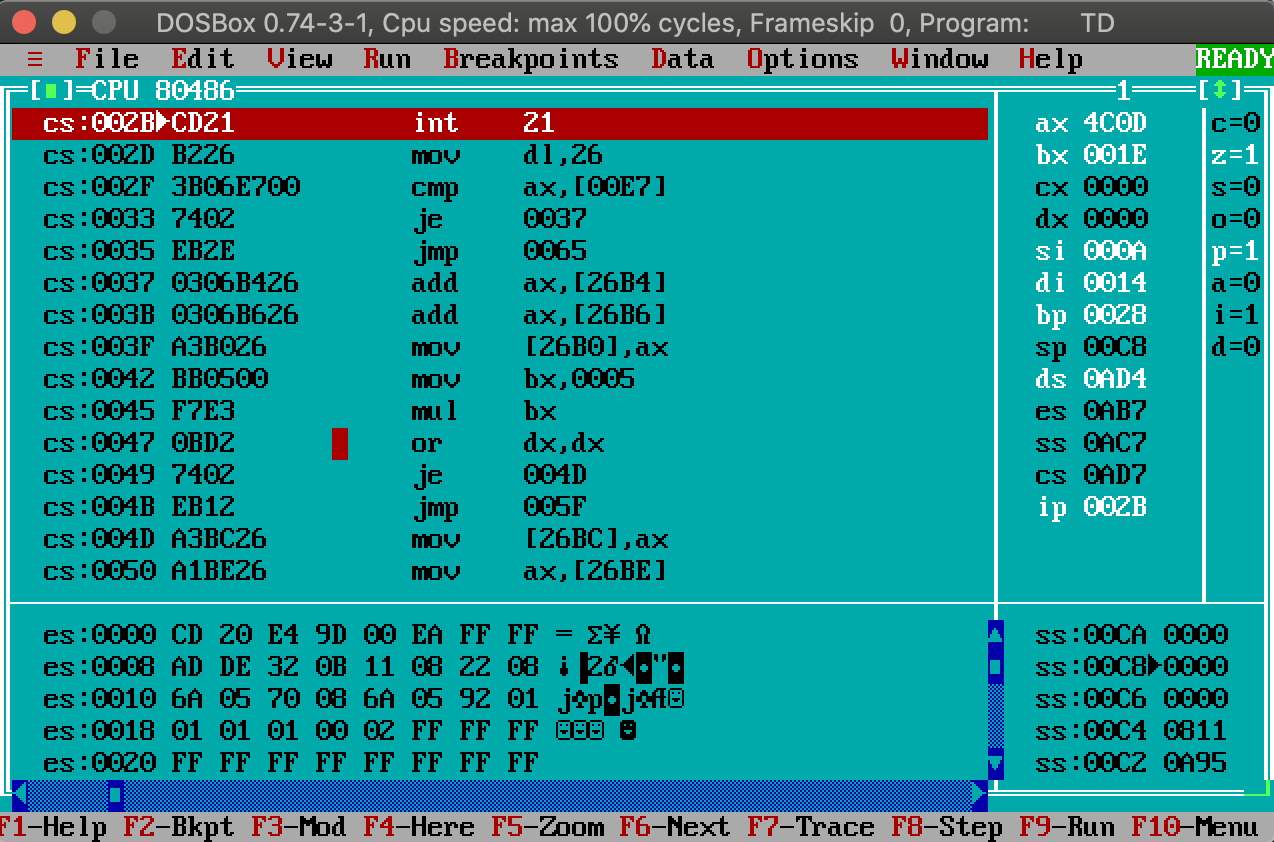


图1.5 执行INT 21H前情况

1. 跳转至DS区。转换至堆栈区后使用goto功能输入 “DS:0”，得到结果如图1.7所示。与预期结果相同。

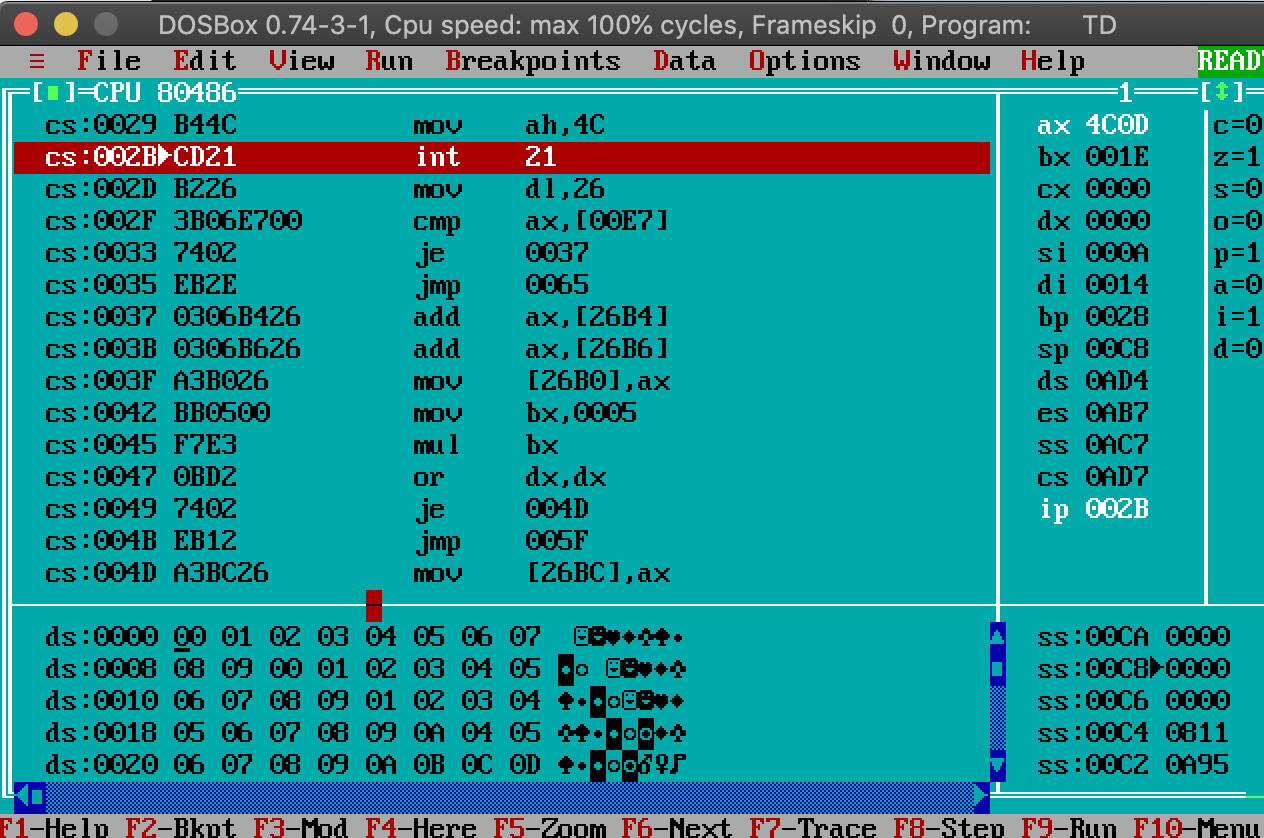


图1.6 DS段数据前40字节

1. 源程序

.386

STACK SEGMENT USE16 STACK

      DB 200 DUP(0)

STACK ENDS

DATA SEGMENT USE16

BUF1 DB 0,1,2,3,4,5,6,7,8,9

BUF2 DB 10 DUP(0)

BUF3 DB 10 DUP(0)

BUF4 DB 10 DUP(0)

DATA ENDS

CODE SEGMENT USE16

     ASSUME CS:CODE,DS:DATA,SS:STACK

START:  MOV      AX,DATA

        MOV      DS,AX

    MOV SI, OFFSET BUF1

    MOV DI, OFFSET BUF2

    MOV BX, OFFSET BUF3

    MOV BP, OFFSET BUF4

    MOV CX, 10

LOPA:   MOV AL,[SI]

    MOV [DI],AL

    INC AL

    MOV [BX],AL

    ADD AL,3

    MOV DS:[BP],AL

    INC SI

    INC DI

    INC BP

    INC BX

    DEC CX

        JNZ LOPA

    MOV AH,4CH

    INT 21H

CODE    ENDS

    END START

## 任务1.3的实验过程

### 实验步骤

* 1. 修改任务2中的代码,使其满足任务3的需求,命名为expr\_1\_3.asm保存。
  2. 生成对应asm、obj、exe文件操作与使用TD编译操作同任务2。
  3. 实验预测

表1.4任务3预计结果

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | DS段前40字节内容 | | | | | | | |
| 程序结束前 | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
| 08 | 09 | 00 | 01 | 02 | 03 | 04 | 05 |
| 06 | 07 | 08 | 09 | 01 | 02 | 03 | 04 |
| 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D |

### 实验记录与分析

1. 修改后代码

**.386**

**STACK SEGMENT USE16 STACK**

**DB 200 DUP(0)**

**STACK ENDS**

**DATA SEGMENT USE16**

**BUF1 DB 0,1,2,3,4,5,6,7,8,9**

**BUF2 DB 10 DUP(0)**

**BUF3 DB 10 DUP(0)**

**BUF4 DB 10 DUP(0)**

**DATA ENDS**

**CODE SEGMENT USE16**

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

**START:  MOV      AX,DATA**

**MOV      DS,AX**

**MOV SI, OFFSET BUF1**

**MOV DI, OFFSET BUF2**

**MOV BX, OFFSET BUF3**

**MOV BP, OFFSET BUF4**

**MOV CX, 10**

**LOPA:   MOV AL,[SI]**

**MOV [DI],AL**

**INC AL**

**MOV [BX],AL**

**ADD AL,3**

**MOV DS:[BP],AL**

**INC SI**

**INC DI**

**INC BP**

**INC BX**

**DEC CX**

**JNZ LOPA**

**MOV AH,4CH**

**INT 21H**

**CODE    ENDS**

**END START**

1. 实验环境条件：intel i5-7360U 2.3GHz，8G内存；MacOS下DOSBox0.74； TD.EXE 5.0
2. 汇编,连接过程没有发生异常。
3. 对文件进行单步调试，并使用goto跳转至DS段观察前40个字节的内容，最终成功得到结果如图1.8所示，结果与预测相同。

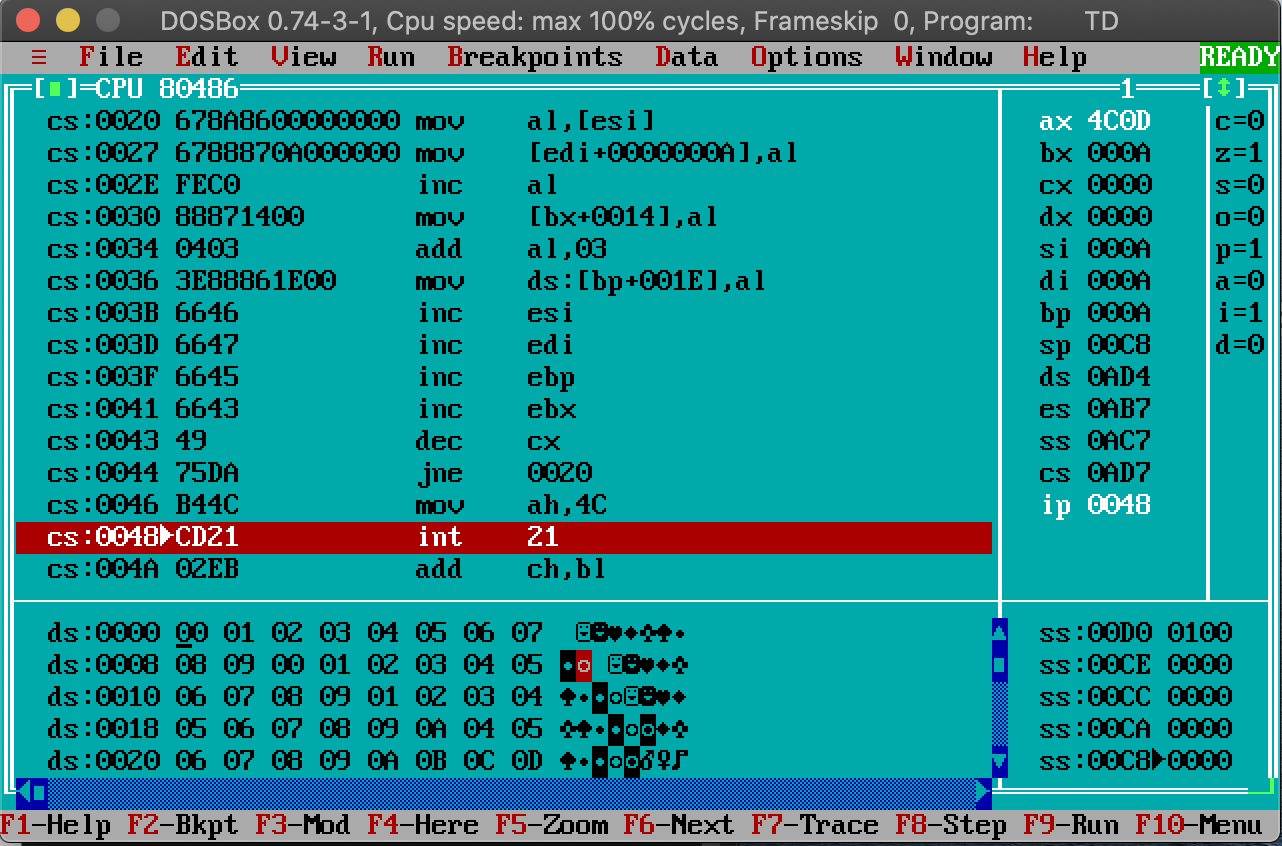


图1.7 任务3DS段前40字节

1. 任务3反汇编指令与原代码指令对比

代码段截图如图1.9所示

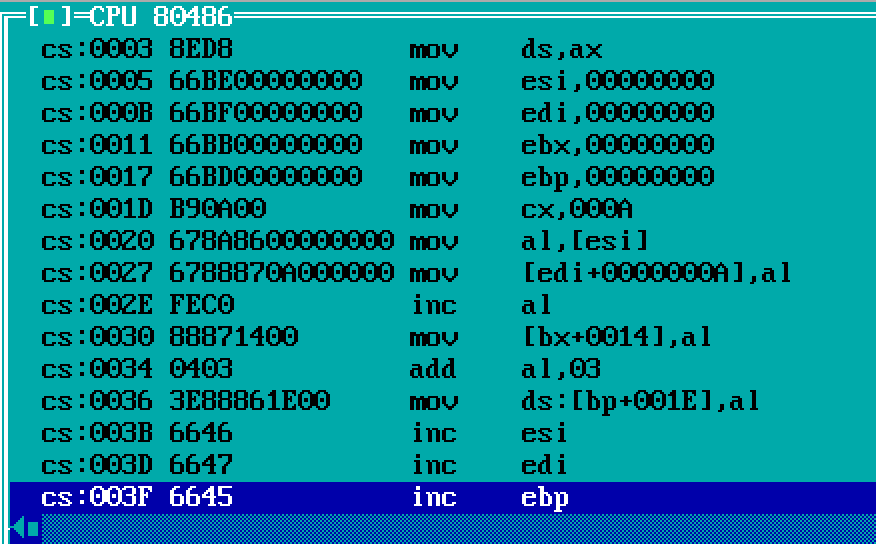
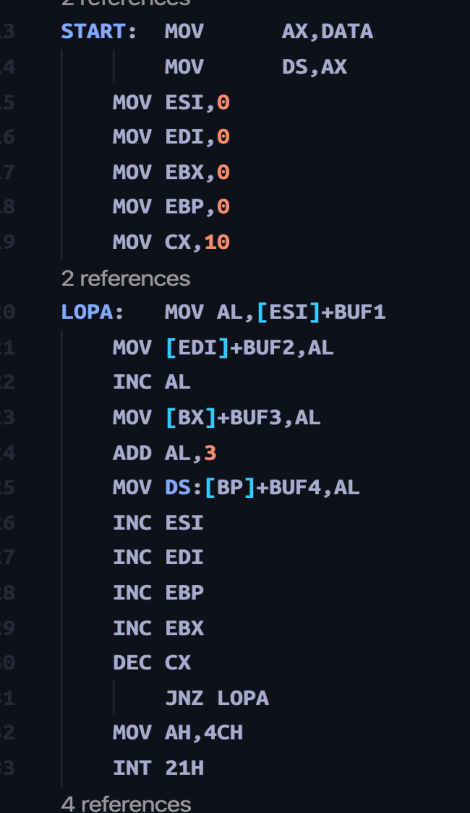
 

图1.8 任务3反汇编代码（左）与源代码（右）对比

（1）由对比可知在源代码中用十进制表示的常数在反汇编后全部转化为了十六进制。如源代码的MOV CX 10在反汇编中变成了MOV CX 000A。

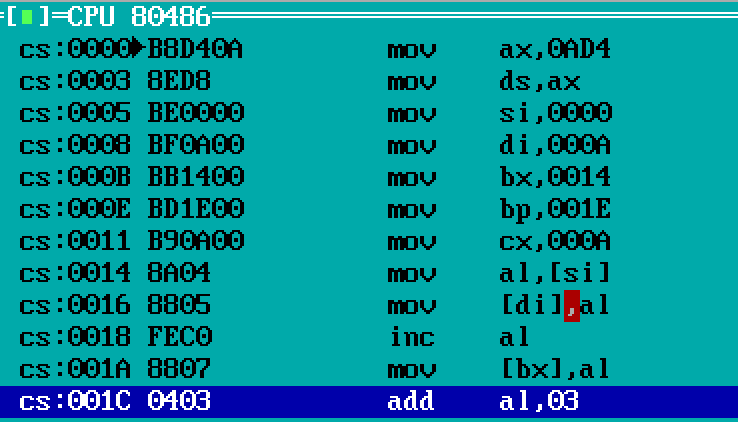
（2）在反汇编中所有的地址表示方法都变成了偏移地址。如MOV DS:[BP]+BUF4,AL在反汇编中变成了MOV DS:[bp+001E],AL

以及JNZ LOPA中LOPA标识也变成了JNE 0020。

（3）除了上述外，大部分表示反汇编任然与源程序表示方法相同。包括选择的寄存器名称，操作指令名称等，都保持不变。

1. 任务2与任务3反汇编代码对比

代码段截图如图1.9所示



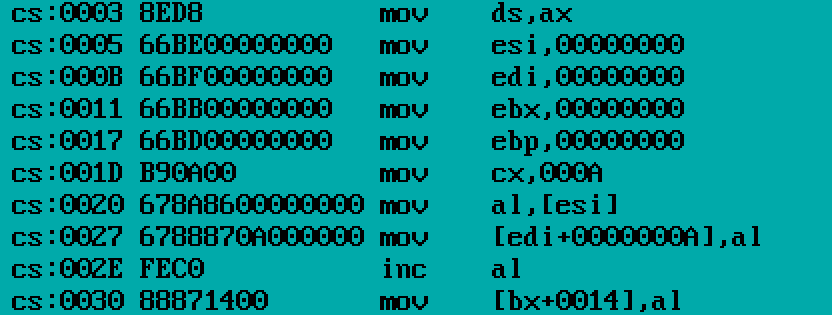


图1.9 任务2反汇编代码（上）任务3反汇编代码（下）

可以清晰对比32位寄存器与16位寄存器在偏移地址上的差别。

1. 尝试从不同位置开始反汇编

最开始仅仅使用了goto语句，并没有将该语句设置为操作句，结果发现寄存器IP并没有改变，且TRACE操作后仍然返回了GOTO之前的语句。

利用goto功能在代码段输入0014地址，并且将其设置为操作句后，观察到IP也随之更改为0014，如图1.10所示。EIP/IP是程序代码段执行的指向，计算机会从EIP/IP所指的位置开始执行代码。

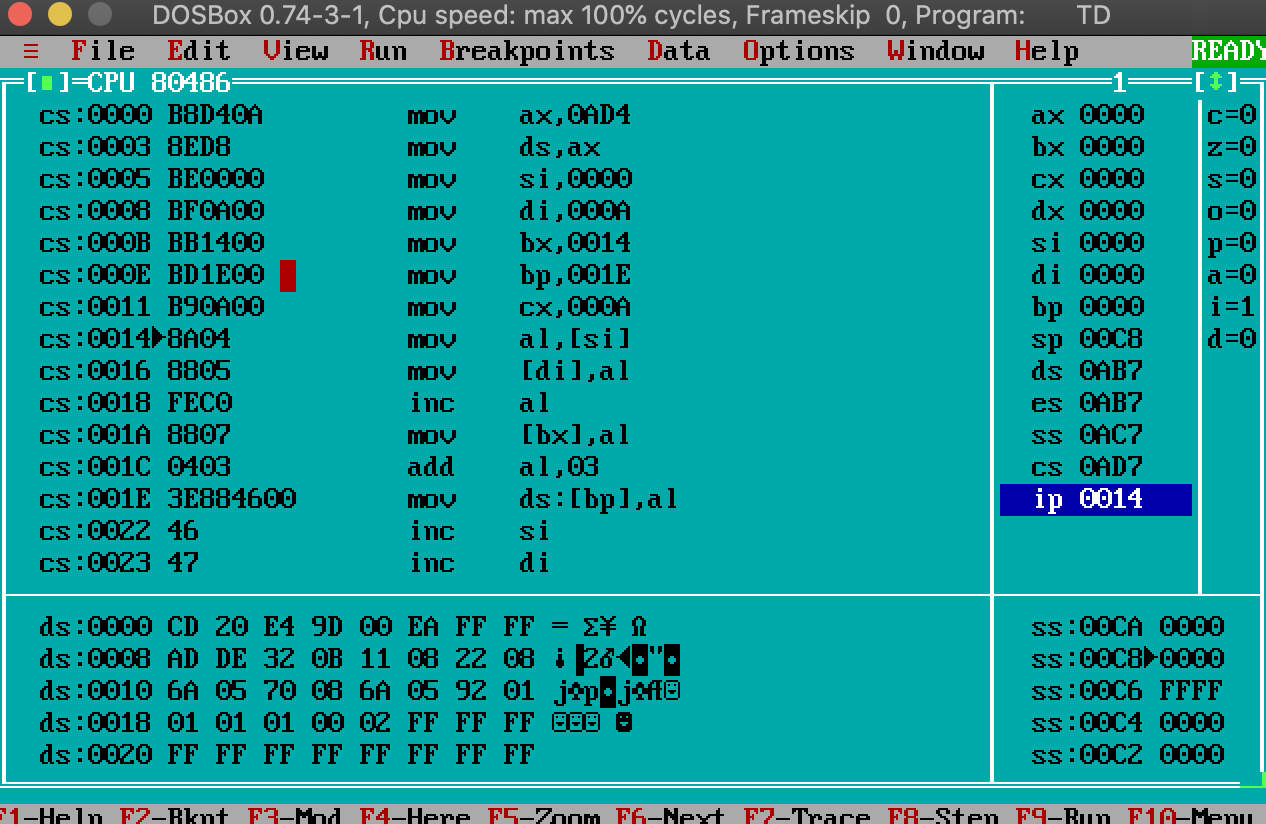


图1.10 goto语句操作

## 任务1.4的实验过程

### 实验步骤

1. 设计登录/重新登录

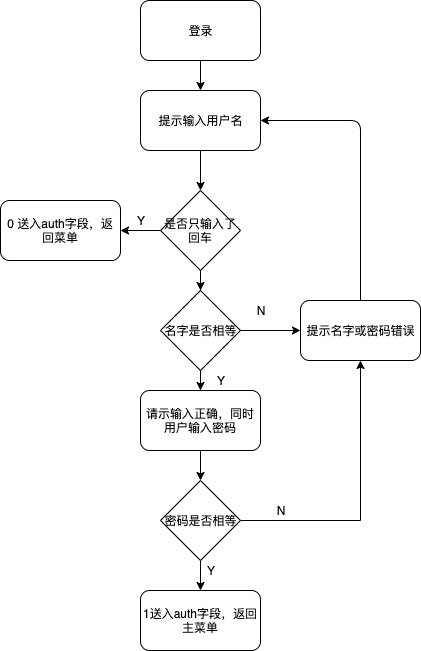


图1.11 登录操作流程图

1. 查找指定商品并显示其信息

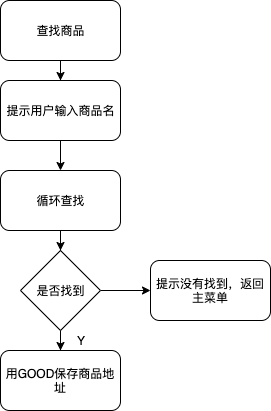


图1.12 登录操作流程图

1. 下订单

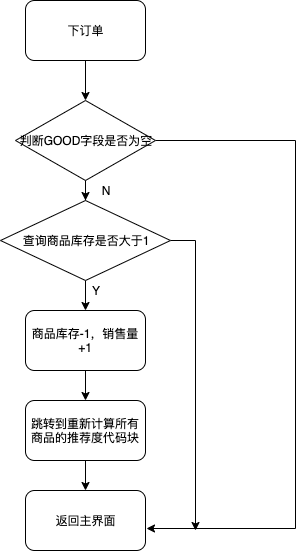


图1.13 登录操作流程图

1. 计算商品推荐度，重点在于找对偏移量

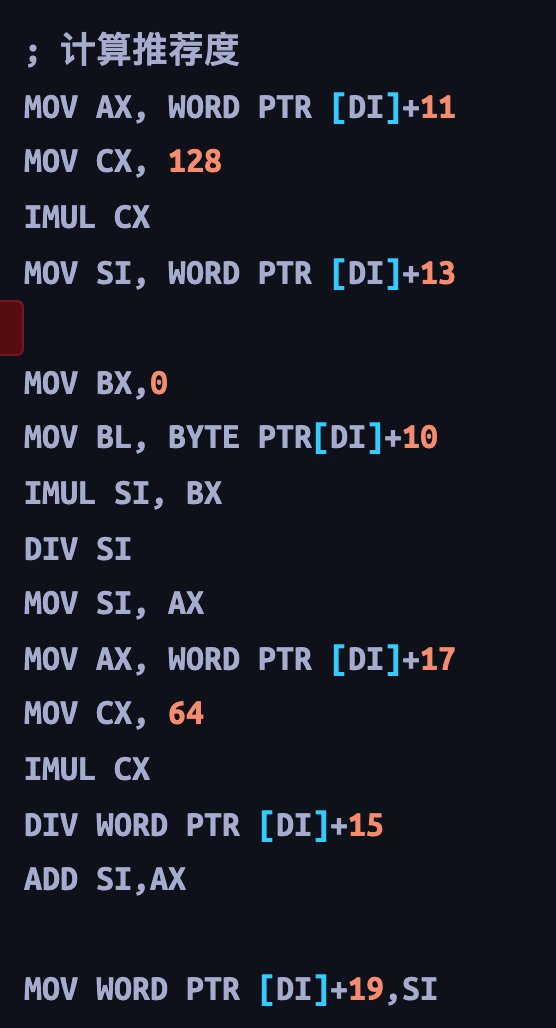


图1.14 计算推荐度代码片段

1. 显示CS的16进制，思路：定义一个buf10:‘0123456789abcdef’,对CS的值取关于16的余数，同时放入栈中，然后一次pop，通过值可以得到buf10中对应的ascii码

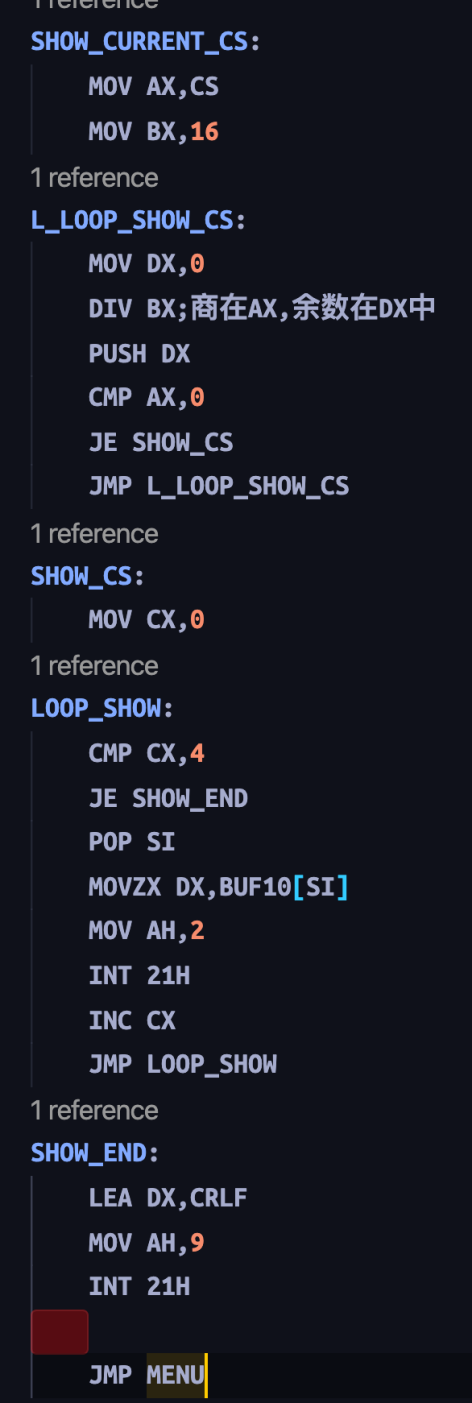


图1.15 输出CS的16进制

1. 实验代码：

**.386**

**STACK SEGMENT USE16 STACK**

**DB 200 DUP(0)**

**STACK ENDS**

**DATA SEGMENT USE16**

**BUF  DB '\*\*\*\*\*THE SHOP IS SHOP\_ONE\*\*\*\*\*$'**

**BUF1 DB 'PLEASE INPUT YOUR NAME $'**

**BUF2 DB 'PLEASE INPUT YOUR PASSWORD $'**

**BUF3 DB 'The name is wrong $'**

**BUF4 DB 'Please input the good you want $'**

**BUF5 DB 'The NAME is right $'**

**BUF6 DB 'The PWD is wrong$'**

**BUF7 DB 'Landed successfully$'**

**BUF8 DB 'Good you want not exist$'**

**BUF9 DB 'Good is not remain$'**

**BUF10 DB '0123456789abcdef'**

**BUF11 DB 'The pwd is right $'**

**BUF12 DB 'Good found!!!$'**

**BUF13 DB 'count !!!$'**

**BUF14 DB 'update success !!!$'**

**AUTH DB ?**

**CRLF DB 0DH,0AH,'$'**

**IN\_NAME DB 11**

**DB ?**

**DB 11 DUP(0)**

**IN\_PWD   DB 7**

**DB ?**

**DB 7 DUP(0)**

**IN\_GOOD DB 11**

**DB ?**

**DB 11 DUP(0)**

**BNAME DB 'yjw',3 DUP(0)**

**COUNT1 = $-BNAME**

**BPASS DB 'test',0,0**

**COUNT2 = $-BPASS**

**N EQU 30**

**S1  DB 'SHOP\_ONE',0**

**GA1 DB 'PEN',7 DUP(0),10**

**DW 35,56,70,25,?**

**GA2 DB 'BOOK',6 DUP(0),9**

**DW 12,30,25,5,?**

**;实际销售价格=销售价\*折扣/10**

**; 进货价(字类型)，**

**; 销售价（字类型），**

**; 进货总数（字类型），**

**;已售数量（字类型）**

**; 推荐度**

**GAN DB N-2 DUP('TEMP-VALUE',8,15,0,20,0,30,0,2,0,?,?)**

**GOOD DW 0**

**DATA ENDS**

**CODE SEGMENT USE16**

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

**START:**

**MOV AX,DATA**

**MOV DS,AX**

**LEA DI,GA1**

**MOV CX,10**

**MENU:**

**; 输出商店名称**

**LEA DX,BUF**

**MOV AH,9**

**INT 21H**

**; 输出空格**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**L\_INPUT:**

**;输入一个字符**

**MOV AH,1**

**INT 21H**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**CMP AL,'1'**

**JE LOGIN**

**CMP AL,'2'**

**JE LIST\_GOOD**

**CMP AL,'3'**

**JE ORDER**

**CMP AL,'4'**

**JE COUNT\_PRODUCE**

**CMP AL,'5'**

**JE RANK**

**CMP AL,'6'**

**JE MODIFY\_GOOD**

**CMP AL,'7'**

**JE CHANGE\_STORE\_RUNTIME**

**CMP AL,'8'**

**JE SHOW\_CURRENT\_CS**

**CMP AL,'9'**

**JE OVER**

**JMP MENU**

**LOGIN:**

**;请输入姓名**

**LEA DX,BUF1**

**MOV AH,9**

**INT 21H**

**LEA DX,IN\_NAME**

**MOV AH,10**

**INT 21H**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**;判断是否仅输入回车，若是，直接跳到**

**CMP IN\_NAME+1,0**

**JE MENU**

**;长度不一致**

**L\_CHECK\_NAME:**

**LEA SI,IN\_NAME**

**LEA DI,BNAME**

**MOV CL,1[SI]**

**CMP CL,3**

**JNE LOGIN\_NAME\_WRONG**

**L\_LOOP\_N:**

**MOV AL,2[SI]**

**MOV BL,[DI]**

**INC SI**

**INC DI**

**CMP AL,BL**

**JNE LOGIN\_NAME\_WRONG**

**LOOP L\_LOOP\_N**

**LEA DX,BUF5**

**MOV AH,9**

**INT 21H**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**L\_IN\_PWD:**

**LEA DX,BUF2;输入密码**

**MOV AH,9**

**INT 21H**

**LEA DX,IN\_PWD**

**MOV AH,10**

**INT 21H**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**;比较密码**

**CHECK\_PWd\_LENGTH:**

**LEA SI,IN\_PWD**

**LEA DI,BPASS**

**MOV CL,1[SI]**

**CMP CL,4**

**JNE LOGIN\_PWD\_WRONG**

**L\_CHECK\_PWD:**

**MOV AL,2[SI]**

**MOV BL,[DI]**

**INC SI**

**INC DI**

**CMP AL,BL**

**JNE LOGIN\_PWD\_WRONG**

**LOOP L\_CHECK\_PWD**

**LEA DX,BUF11**

**MOV AH,9**

**INT 21H**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**JMP L\_AUTH\_1**

**L\_AUTH\_0:**

**;auth设置为0**

**MOV AUTH,0**

**JMP MENU**

**L\_AUTH\_1:**

**;auth设置为1**

**MOV AUTH,1**

**JMP MENU**

**LOGIN\_NAME\_WRONG:**

**LEA DX,BUF3;提示输出错误的信息**

**MOV AH,9**

**INT 21H**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**JMP LOGIN**

**LOGIN\_PWD\_WRONG:**

**LEA DX,BUF6**

**MOV AH,9**

**INT 21H**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**JMP L\_IN\_PWD**

**LIST\_GOOD:**

**;;提示输入商品名称**

**LEA DX,BUF4**

**MOV AH,9**

**INT 21H**

**; 输入商品的名字**

**LEA DX,IN\_GOOD**

**MOV AH,10**

**INT 21H**

**;比较长度**

**CMP IN\_GOOD+1,0**

**JNE L\_CHECK\_GOOD**

**JMP GOOD\_NOT\_FOUND**

**L\_CHECK\_GOOD:**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**MOV DX,0**

**MOV AH,0**

**LL:**

**LEA DI,GA1**

**MOV AX,DX**

**IMUL AX,21**

**ADD DI,AX**

**LEA SI,IN\_GOOD**

**MOV CL,1[SI]**

**L\_LOOP\_G:**

**MOV AL,2[SI]**

**MOV BL,[DI]**

**INC SI**

**INC DI**

**CMP AL,BL**

**JNE L\_NEXT\_GOOD**

**DEC CL**

**CMP CL,0**

**JE  MARK\_GOOD**

**JMP L\_LOOP\_G**

**L\_NEXT\_GOOD:;进入下个商品的位置**

**INC DX**

**CMP DX,30**

**JE GOOD\_NOT\_FOUND**

**JMP LL**

**; 检查是否登陆，如果登陆**

**MARK\_GOOD:**

**LEA DI,GA1**

**IMUL DX,21**

**ADD DI,DX**

**MOV GOOD,DI**

**GOOD\_FOUND:**

**LEA DX,BUF12**

**MOV AH,9**

**INT 21H**

**; 空格**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**JMP MENU**

**GOOD\_NOT\_FOUND:**

**;提示未找到信息**

**LEA DX,BUF8**

**MOV AH,9**

**INT 21H**

**; 空格**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**JMP MENU**

**ORDER:**

**CMP GOOD,0**

**JE MENU**

**MOV DI,GOOD**

**;进货数量-1**

**MOV AX,WORD PTR [DI]+15**

**; 等于0则跳转**

**CMP AX,0**

**JE  GOOD\_NO\_REMAIN**

**SUB AX,1**

**MOV WORD PTR [DI]+15,AX**

**;销量+1**

**MOV AX,WORD PTR [DI]+17**

**ADD AX,1**

**MOV WORD PTR [DI]+17,AX**

**JMP COUNT\_PRODUCE**

**GOOD\_NO\_REMAIN:**

**LEA DX,BUF9 ;提示未找到信息**

**MOV AH,9**

**INT 21H**

**; 空格**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**JMP MENU**

**COUNT\_PRODUCE:**

**MOV DX,0**

**; 计算推荐度**

**COUNT\_LOOP:**

**PUSH DX**

**LEA DI,GA1**

**MOV AX,DX**

**IMUL AX,21**

**ADD DI,AX**

**; 计算推荐度**

**MOV AX, WORD PTR [DI]+11**

**MOV CX, 128**

**IMUL CX**

**MOV SI, WORD PTR [DI]+13**

**MOV BX,0**

**MOV BL, BYTE PTR[DI]+10**

**IMUL SI, BX**

**DIV SI**

**MOV SI, AX**

**MOV AX, WORD PTR [DI]+17**

**MOV CX, 64**

**IMUL CX**

**DIV WORD PTR [DI]+15**

**ADD SI,AX**

**MOV WORD PTR [DI]+19,SI**

**POP DX**

**INC DX**

**CMP DX,30**

**JNE COUNT\_LOOP**

**LEA DX,BUF14**

**MOV AH,9**

**INT 21H**

**; 空格**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**JMP MENU**

**RANK:**

**JMP MENU**

**MODIFY\_GOOD:**

**JMP MENU**

**CHANGE\_STORE\_RUNTIME:**

**JMP MENU**

**SHOW\_CURRENT\_CS:**

**MOV AX,CS**

**MOV BX,16**

**L\_LOOP\_SHOW\_CS:**

**MOV DX,0**

**DIV BX;商在AX,余数在DX中**

**PUSH DX**

**CMP AX,0**

**JE SHOW\_CS**

**JMP L\_LOOP\_SHOW\_CS**

**SHOW\_CS:**

**MOV CX,0**

**LOOP\_SHOW:**

**CMP CX,4**

**JE SHOW\_END**

**POP SI**

**MOVZX DX,BUF10[SI]**

**MOV AH,2**

**INT 21H**

**INC CX**

**JMP LOOP\_SHOW**

**SHOW\_END:**

**LEA DX,CRLF**

**MOV AH,9**

**INT 21H**

**JMP MENU**

**OVER:**

**MOV   AH,4CH**

**MOV   AL, 0;退出码 (如0、0FFH等)**

**INT   21H**

**CODE ENDS**

**END START**

### 实验记录与分析

1. 登录，输入正确的用户名和密码后登录成功，否则登录失败

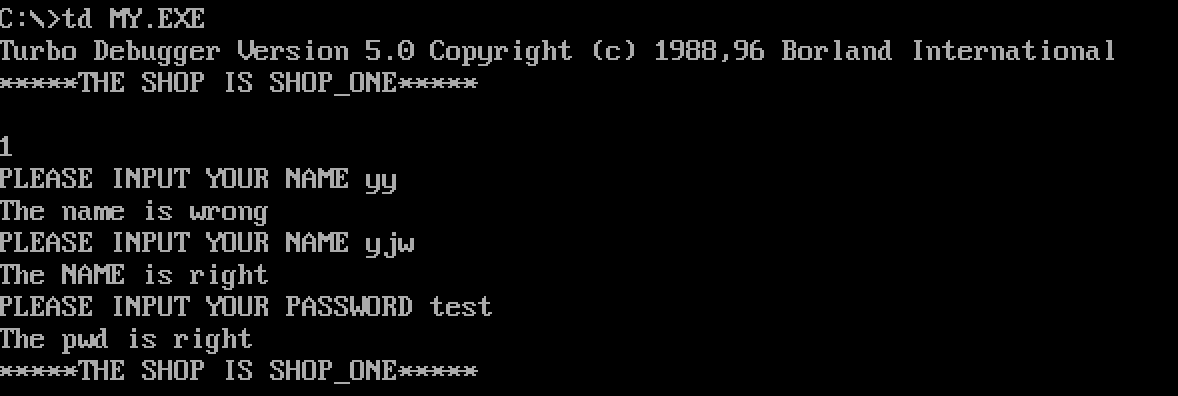


图1.16 登陆验证

1. 查找商品

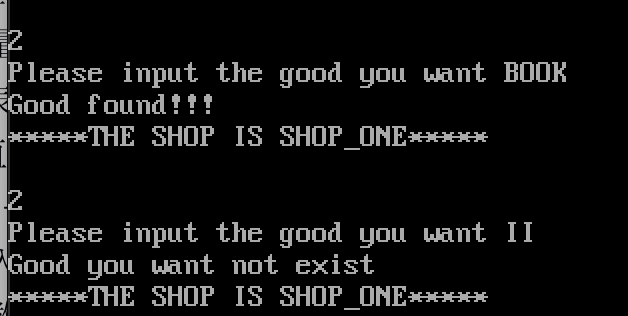


图1.17 登陆验证

1. 下订单，当没有选定商品时，直接返回，下订单后，同时修改推荐度

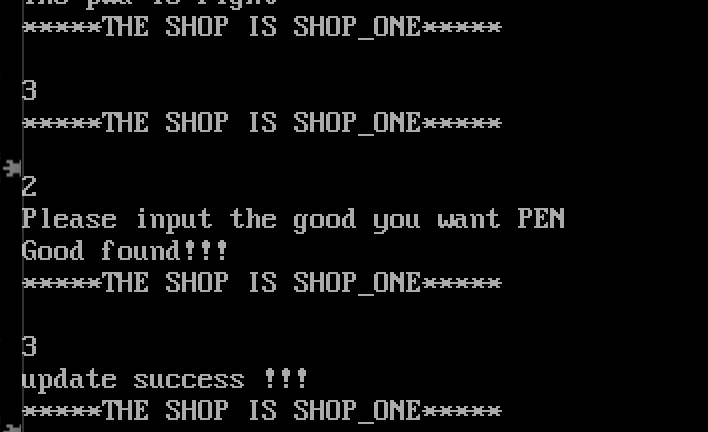


图1.18 登陆验证

1. 输出CS的16进制

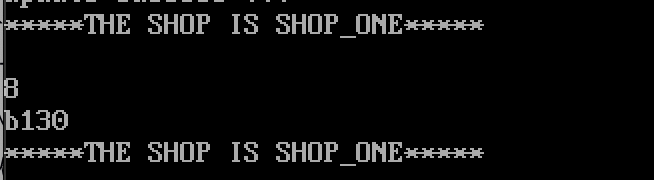


图1.19 16进制输出

## 小结

### 主要收获

通过这个实验，学会了在Doxbox环境下调试汇编程序，通过逐步调试，运行程序，对对应的汇编指令行为和寻址方式有了更直观的理解。在使用寄存器的时候一定要是否会被其他的语句的行为覆盖掉寄存器的值，比如AX,DX在做除法的时候会被覆盖的，在1.4的实验中，一开始总是出现死循环，后面debug发现是做除法的时候覆盖了DX的值

### 主要看法

汇编程序相比高级语言程序，代码逻辑更加细腻，汇编程序实现的功能没有高级程序设计语言这么直观，阅读起来需要花更多的时间，但是汇编程序然我们能够从更加高效的方式去优化程序。

# 程序优化

## 实验目的与要求

1. 了解程序计时的方法以及运行环境对程序执行情况的影响。
2. 熟悉汇编语言指令的特点，掌握代码优化的基本方法

## 实验内容

**任务2.1 观察多重循环对CPU计算能力消耗的影响。**

请通过适当修改任务1.4的程序，完成如下研究：

1.请描述并实现对一段代码的执行时间进行测量的方法。该方法应能观察到程序中的一条指令发生修改时，程序完成同样功能时的执行时间的变化。

2.通过在不同软硬件运行环境下运行同一个程序，观察程序执行时间是否会随之发生变化。

**提示** 1：程序计时方法有多种，比较直观的方法是在待测程序执行前获取一个时间标签，执行之后再获取一个时间标签，然后通过计算时间标签的差值来得到程序的执行时间。在QQ群里上传了一个计时程序timer.asm（里面有使用方法的注释），供大家学习参考。但该计时程序的精度在ms级，不能直接观察到几条指令对程序执行时间的影响。可以通过大量重复执行几条指令的方法来解决此问题。在复杂的数据处理、大量用户同时访问（如双十一购物）、大数据时代等场景下，某个功能的程序被大量地循环执行的情形还是比较常见的，因此，上述计时的做法也具有实用性，可以用来观察评估程序的执行效率。

对于任务1.4，现假设在双十一零点时，SHOP网店中的“Bag”商品共有m件，有m个顾客几乎同时下单购买了该商品。请模拟后台处理上述信息的过程并观察执行的时间。

上述场景的后台处理过程，可以理解为在同一台电脑上有m个请求一起排队使用任务1.4的程序。为了观察从第1个顾客开始进入购买至第m个顾客购买完毕之间到底花费了多少时间，我们让任务1.4的“下订单”调整后的代码重复执行m次，通过计算这m次循环执行前和执行后的时间差，来感受其影响。

**调整后的下订单的描述：**

【前提条件】用户查询到了商品“Bag”，在主菜单界面下选择了“3”对应的下订单功能。

**《下订单》**

（1）判断当前浏览商品是否有效（GOOD不为空），若有效，判断其剩余数量是否为0，不为0则将已售数量加1，重新计算所有商品的推荐度。

返回主菜单界面

1. 若无效或剩余数量为0，则提示错误，回到主菜单界面。

**任务2.** 对任务1中的汇编源程序进行优化

优化工作包括代码长度的优化和执行效率的优化，本次优化的重点是执行效率的优化。请通过优化m次循环体内的程序，使程序的执行时间尽可能减少10%以上（注意，在编写任务1的程序时，尽量不要考虑代码优化的问题）。

**优化方法提示：**首先是通过选择执行速度较快的指令来提高性能，比如，把乘除指令转换成移位指令、加法指令等；其次，内循环体中每减少一条指令，就相当于减少了m\*n条指令的执行时间，需要仔细斟酌；第三，在寻址方式中尽量把16位寄存器换成32位寄存器，能有**更多的机会和技巧**提高指令执行效率。

## 任务2.1实验过程

### 实验方法说明

1. 在1.4的基础上修改代码

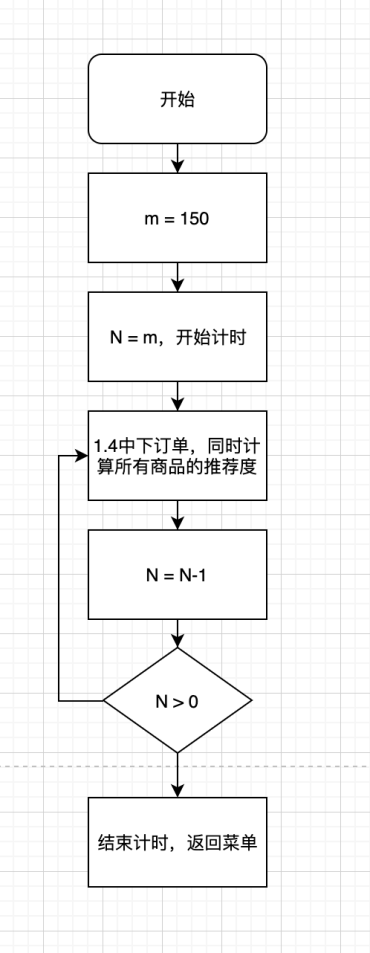


图2.1 代码逻辑

1. 相关程序代码

ORDER:

    CMP GOOD,0

    JE MENU

    MOV AX,0

    CALL TIMER

LOOP\_M:

    MOV DI,GOOD

    ;进货数量

    MOV AX,WORD PTR [DI]+15

    ;销量

    MOV BX,WORD PTR [DI]+17

    CMP AX,BX;比较

    JE GOOD\_NO\_REMAIN ;进货数量等于已售数量，显示已经售完

    MOV AX, WORD PTR [DI]+17

    ADD AX,1

    MOV WORD PTR [DI]+17,AX

COUNT\_PRODUCE:

    MOV BP,N;

    LEA DI,GA1

; 计算推荐度

COUNT\_LOOP:

    MOV AX, WORD PTR [DI]+11

    MOV CX, 10

    IMUL CX

    MOV SI, WORD PTR [DI]+13

    MOV BX,0

    MOV BL, BYTE PTR[DI]+10

    IMUL SI, BX

    DIV SI ;AX中为商，DX中为余数

    MOV SI,AX ;将商移动到SI中储存

     ;以上部分为推荐度的左半部分

    MOV AX, WORD PTR [DI]+17

    MOV CX, 64

    IMUL CX

    DIV WORD PTR [DI]+15;商储存在AX中

    ADD SI,AX;SI即为推荐度

    MOV 19[DI],SI;将推荐度储存

    ADD DI,21;将DI指向下一商品信息段

    DEC BP;查看下一商品段信息，BP加一

    CMP BP,0;

    JNE COUNT\_LOOP

    DEC COUNTM

    CMP COUNTM,0

    JNE LOOP\_M

    MOV AX,1        ;计时

    CALL TIMER

    ;LEA DX,BUF13 ;调试

    ;MOV AH,9

    ;INT 21H

    ; 空格

    ;LEA DX,CRLF

    ;MOV AH,9

    ;INT 21H

    MOV GOOD,0;

    JMP MENU

GOOD\_NO\_REMAIN:

    LEA DX,BUF9

    MOV AH,9

    INT 21H

    ; 空格

    LEA DX,CRLF

    MOV AH,9

    INT 21H

    JMP MENU

RANK:

    JMP MENU

MODIFY\_GOOD:

    JMP MENU

CHANGE\_STORE\_RUNTIME:

    JMP MENU

SHOW\_CURRENT\_CS:

    MOV AX,CS

    MOV BX,16

L\_LOOP\_SHOW\_CS:

    MOV DX,0

    DIV BX;商在AX,余数在DX中

    PUSH DX

    CMP AX,0

    JE SHOW\_CS

    JMP L\_LOOP\_SHOW\_CS

SHOW\_CS:

    MOV CX,0

LOOP\_SHOW:

    CMP CX,4

    JE SHOW\_END

    POP SI

    MOVZX DX,BUF10[SI]

    MOV AH,2

    INT 21H

    INC CX

    JMP LOOP\_SHOW

SHOW\_END:

    LEA DX,CRLF

    MOV AH,9

    INT 21H

    JMP MENU

OVER:

    MOV   AH,4CH

    MOV   AL, 0;退出码 (如0、0FFH等)

    INT   21H

TIMER   PROC

    PUSH  DX

    PUSH  CX

    PUSH  BX

    MOV   BX, AX

    MOV   AH, 2CH

    INT   21H        ;CH=hour(0-23),CL=minute(0-59),DH=second(0-59),DL=centisecond(0-100)

    MOV   AL, DH

    MOV   AH, 0

    IMUL  AX,AX,1000

    MOV   DH, 0

    IMUL  DX,DX,10

    ADD   AX, DX

    CMP   BX, 0

    JNZ   \_T1

    MOV   CS:\_TS, AX

\_T0:    POP   BX

    POP   CX

    POP   DX

    RET

\_T1:    SUB   AX, CS:\_TS

    JNC   \_T2

    ADD   AX, 60000

\_T2:    MOV   CX, 0

    MOV   BX, 10

\_T3:    MOV   DX, 0

    DIV   BX

    PUSH  DX

    INC   CX

    CMP   AX, 0

    JNZ   \_T3

    MOV   BX, 0

\_T4:    POP   AX

    ADD   AL, '0'

    MOV   CS:\_TMSG[BX], AL

    INC   BX

    LOOP  \_T4

    PUSH  DS

    MOV   CS:\_TMSG[BX+0], 0AH

    MOV   CS:\_TMSG[BX+1], 0DH

    MOV   CS:\_TMSG[BX+2], '$'

    LEA   DX, \_TS+2

    PUSH  CS

    POP   DS

    MOV   AH, 9

    INT   21H

    POP   DS

    JMP   \_T0

\_TS DW    ?

    DB    'Time elapsed in ms is '

\_TMSG   DB    12 DUP(0)

TIMER   ENDP

CODE ENDS

    END START

本实验主要通过控制变量法，通过修改m和商品数量n值（比如都大于1000等），来观察程序执行时间的变化。

通过在不同的计算机硬件、不同的虚拟机环境、以及调试工具TD等下运行该程序，观察环境对程序执行时间的影响。

### 实验记录与分析

1. 尝试改变循环次数观察运行时间

先将访问顾客数m设置为3000，商品数n设置为1000，运行程序所花费时间如图2.3所示。由进入循环前与退出循环后的两个时间段可知，m=3000,n=1000时，花费220毫秒。

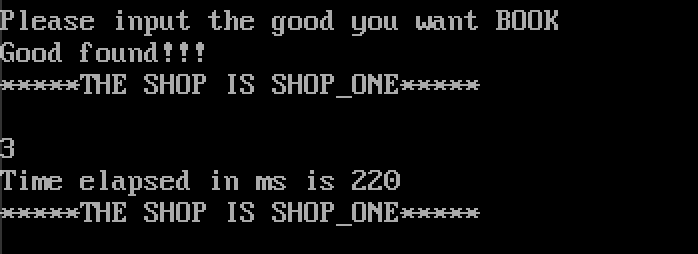


图2.2未扩大前所需时间

1. 尝试将m扩大为5000，n扩大为3000，运行程序所花费时间如图1.4所示。由进入循环前与退出循环后的两个时间段可知，m=5000,n=3000时，花费1260毫秒。

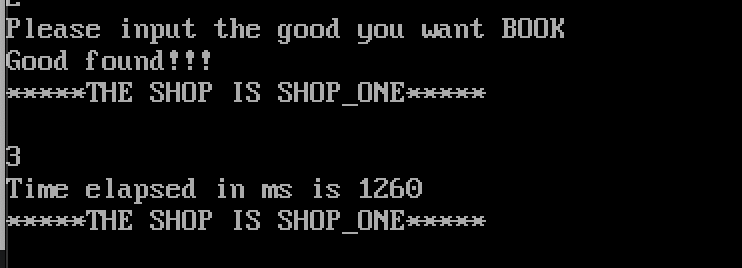


图2.3 扩大用户与商品数后所需时间

## 任务2.2实验过程

### 实验方法说明

尝试从不同的角度优化程序，考虑以下方面优化：

1. 乘除指令转换成移位指令
2. 减少循环体内指令
3. 寻址方式16位改成32位

### 实验记录与分析

### 1. 乘法指令进行优化

现为提升运算效率，将其更换为位移运算，将乘法改成位移，如图2.4所示。在m=5000,n=3000时运行程序，所花费时间如图2.7所示。利用位移代替乘法后花费时间为1210毫秒，与图2.3所测的乘法运算1260毫秒相比并无明显提速。

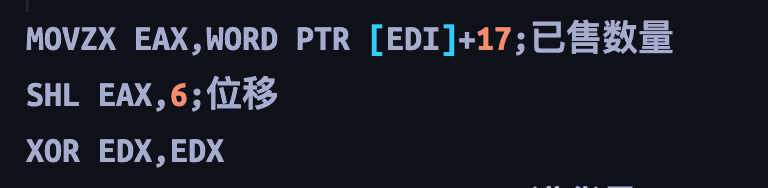


图2.4未扩大前所需时间

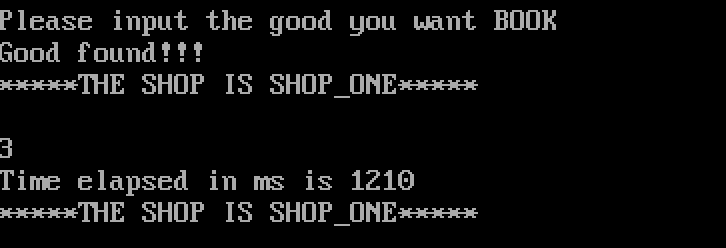


图2.5 位移运算替代后的时间

### 减少循环体内指令条数

尝试优化寻址方式的书写，将修改已售数量“MOV AX, WORD PTR [DI]+17;ADD AX,1;MOV WORD PTR [DI]+17,AX”三句修改为一句指令，在m=5000,n=3000情况下运行，花费时间如图2.6所示。计算可得花费990毫秒，有着明显优化。

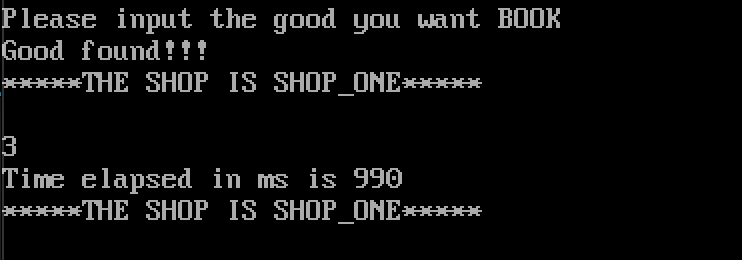


图2.6 减少指令条数后执行时间

### 寻址方式优化

将寻址方式改成32位如图2.7所示。在m=5000,n=3000时运行程序，所花费时间如图2.7所示。将寻址方式改成32位时间后耗时为1090毫秒，与图2.3所测的乘法运算1260毫秒相比有一定提速。

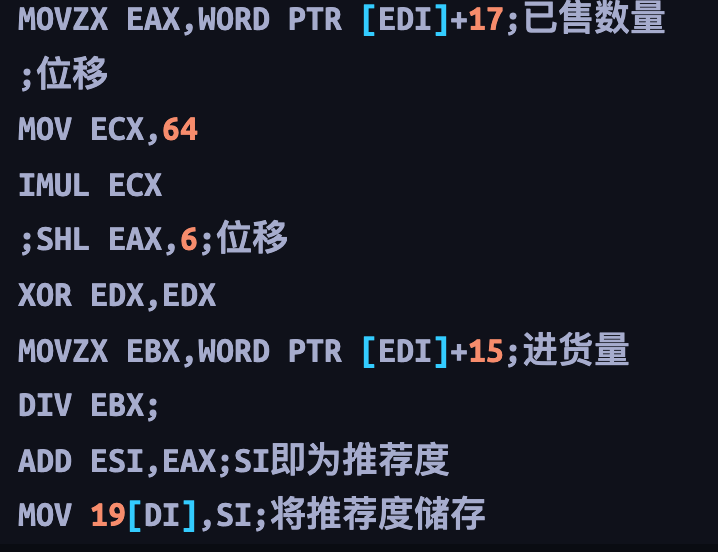


图2.7 寻址方式改成32位

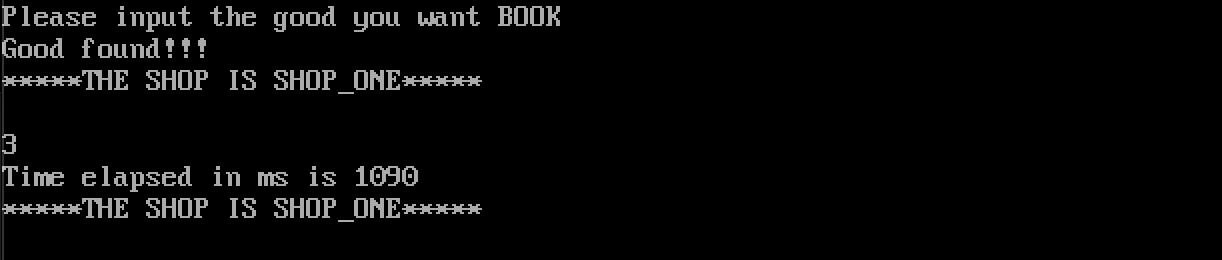


图2.8 寻址方式改成32位后运算时间

## 小结

以上优化方案中，减少指令优化带来的提升最为明显，1260ms减少到990毫秒，优化率为21%。同时发现由位操作指令代替乘法指令对于程序的优化并不明显。而在简化指令等操作对于程序的优化效果更佳明显

### 主要收获

对于指令优化有了一个直观的了解，同时提高了自己阅读调试汇编代码的能力。

1. 溢出问题，最开始并没有考虑到在计算时产生溢出的情况而导致在TD中出现FE而使程序进入了死循环。解决操作是并没有更换寄存器而是分别乘以了64与2来代替直接乘以128，解决溢出问题。
2. 跳转问题，忽视了LOOP指令跳转收到了距离的限制而报错，后选用JNZ等语句进行代替LOOP，解决的跳转的距离问题。

三、寻址方式问题，由于对于之前寻址方式掌握的不够理想，加上遗忘了很多，导致在编写寻址方式时频频出错，通过本次实验也是再一次加强了我对于寻址方式的训练。

### 主要看法

循环体内对于指令的优化，可能用高级语言体验不出来，因为有编译器的优化，但是我们应该有这种意识：在循环体内，减少冗余代码给程序带来的优化是很大的。

# 模块化程序设计

## 实验目的与要求

1. 掌握子程序设计的方法与技巧，熟悉子程序的参数传递方法和调用原理；
2. 掌握宏指令、模块化程序的设计方法;
3. 掌握较大规模程序的开发与调试方法；
4. 掌握汇编语言程序与C语言程序混合编程的方法；
5. 了解C编译器的基本优化方法;
6. 了解C语言编译器的命名方法，主、子程序之间参数传递的机制。

## 实验内容

**任务3.1 宏与子程序设计**

1.把网店商品信息管理系统的子功能尽量改成子程序的方式实现。

2.将任务1.4中重复使用的程序段尽量改成宏（至少定义一个宏指令）或子程序的方式来实现。

3.在网店商品信息管理系统中新增如下功能：

1）在“2.查找指定商品并显示其信息”的功能中，实现商品信息的显示功能。即：在找到指定商品之后，按照：“商品名称，折扣，销售价，进货总数，已售数量，推荐度”顺序显示该商品的信息。

2）实现“6.修改商品信息”的具体功能。

只有老板登录后可以使用本功能。若当前浏览商品无效，则返回；若有效，则按照：折扣，进货价，销售价，进货总数的次序，逐一先显示原来的数值，然后输入新的数值（若输入有错，则重新对该项信息进行显示与修改。若直接回车，则不修改该项信息）。

如：折扣：9》8 //符号“》”仅作为分隔符，也可以选择其他分隔符号

进货价：25》24

销售价：46》5A6 //输入了非法数值，下一行重新显示和输入

销售价：46》56

进货总数：30》 //直接回车时，对这项信息不做修改

当对这些信息都处理完毕后，回到主菜单界面。

## 任务3.1实验过程

### 实验方法说明

1. 将子功能调用改成子程序

以查找商品为例子，首先可以简单的将其代码包装成函数，即将代码放入 FIND\_GOOD PROC...FIND\_GOOD ENDP中，如果需要进一步拆分，可以按同样的思路继续拆分。

代码如下

FIND\_GOOD PROC

    ;;提示输入商品名称

    LEA DX,BUF4

    MOV AH,9

    INT 21H

    ; 输入商品的名字

    LEA DX,IN\_GOOD

    MOV AH,10

    INT 21H

    ;比较长度

    CMP IN\_GOOD+1,0

    JNE L\_CHECK\_GOOD\_F

    JMP GOOD\_NOT\_FOUND\_F

L\_CHECK\_GOOD\_F:

    LEA DX,CRLF

    MOV AH,9

    INT 21H

    MOV DX,0

    MOV AH,0

LL\_F:

    LEA DI,GA1

    MOV AX,DX

    IMUL AX,21

    ADD DI,AX

    LEA SI,IN\_GOOD

    MOV CL,1[SI]

L\_LOOP\_G\_F:

    MOV AL,2[SI]

    MOV BL,[DI]

    INC SI

    INC DI

    CMP AL,BL

    JNE L\_NEXT\_GOOD

    DEC CL

    CMP CL,0

    JE  MARK\_GOOD

    JMP L\_LOOP\_G\_F

L\_NEXT\_GOOD\_F:;进入下个商品的位置

    INC DX

    CMP DX,30

    JE GOOD\_NOT\_FOUND\_F

    JMP LL

MARK\_GOOD\_F:

    LEA DI,GA1

    IMUL DX,21

    ADD DI,DX

    MOV GOOD,DI

GOOD\_FOUND\_F:

    LEA DX,BUF12

    MOV AH,9

    INT 21H

    ; 空格

    LEA DX,CRLF

    MOV AH,9

    INT 21H

    RET

GOOD\_NOT\_FOUND\_F:

    ;提示未找到信息

    LEA DX,BUF8

    MOV AH,9

    INT 21H

    ; 空格

    LEA DX,CRLF

    MOV AH,9

    INT 21H

    RET

FIND\_GOOD ENDP

1. 将重复的代码改成宏指令，首先可看到，输入输出的系统调用实际上是有很多重复代码的，可以进行封装

代码：

IO MACRO A,B

    LEA DX,A

    MOV AX,B

    INT 21H

    ENDM

1. 增加功能，需要实现两个函数，第一个函数功能为输出商品信息，第二个函数用于修改商品信息，在查找商品的基础上，添加代码，这里需要使用而外的两个函数F2T10,F10T2用于十进制字符串和二进制数之间的转化

1. 输出商品信息代码:

FIND\_GOOD PROC

    ;;提示输入商品名称

    IO BUF4,9

    ; 输入商品的名字

    LEA DX,IN\_GOOD

    MOV AH,10

    INT 21H

    ;比较长度

    CMP IN\_GOOD+1,0

    JNE L\_CHECK\_GOOD\_F

    JMP GOOD\_NOT\_FOUND\_F

L\_CHECK\_GOOD\_F:

    IO CRLF,9

    LEA DI,GA1

    LEA SI,IN\_GOOD+2

    MOVZX CX,IN\_GOOD+1;CX保存输入商品单词的个数

    MOV DX,0;DX代表商品信息的序号（从0-29）

LL\_F:

    MOV BX,0;BX代表商品名字比较位置（从0开始计数）

L\_LOOP\_G\_F:

    CMP BX,CX

    JGE FURTHER\_CHECK;，以防输入的商品名为储存商品名的子集

    MOV AL,[DI+BX]

    MOV AH,[SI+BX]

    CMP AH,AL

    JNE L\_NEXT\_GOOD\_F;如果不相等，判断下一条商品信息

    INC BX;如果相等，判断下一个字母是否相等

    JMP L\_LOOP\_G\_F

L\_NEXT\_GOOD\_F:;进入下个商品的位置

    ADD DI,21;将DI移动到下个商品信息段

    INC DX;商品信息序号相应加一

    CMP DX,30

    JE GOOD\_NOT\_FOUND\_F

    JMP LL\_F

FURTHER\_CHECK:

    CMP BYTE PTR[DI+BX],0;判断储存字段该处是否为0

    JE MARK\_GOOD\_F;若是0，则表示输入名字不是储存名字的子集

    JMP L\_NEXT\_GOOD\_F;若不是0，则说明输入名字为储存名字的子集，还需要检查下一条商品信息段

MARK\_GOOD\_F:

    MOV GOOD,DI;将该商品信息段的首地址储存到GOOD字段中

    MOVZX BX,IN\_GOOD+1;将该商品名字单词的长度赋值给BX

    IO BUF16,9;输出提示信息

    IO CRLF,9

    MOV BYTE PTR [DI+BX],'$';在名字的末尾添加$

    IO [DI],9;输出商品的名字

    MOV BYTE PTR [DI+BX],0

    MOVZX AX,BYTE PTR 10[DI]

    MOV DX,16

    CALL F2T10;输出商品折扣

    IO CRLF,9

    WRITE\_GOOD\_INFO 11[DI];进价

    IO CRLF,9

    WRITE\_GOOD\_INFO 13[DI];售价

    IO CRLF,9

    WRITE\_GOOD\_INFO 15[DI];进货数

    IO CRLF,9

    WRITE\_GOOD\_INFO 17[DI];销售数

    IO CRLF,9

    WRITE\_GOOD\_INFO 19[DI];推荐度

    IO CRLF,9

GOOD\_FOUND\_F:

    IO BUF12,9

    IO CRLF,9

    RET

GOOD\_NOT\_FOUND\_F:

    ;提示未找到信息

    IO BUF8,9

    IO CRLF,9

    RET

FIND\_GOOD ENDP

1. 修改商品信息代码：

CHANGE\_GOOD\_INFO PROC

    PUSH BX

    PUSH BP

    PUSH DX

    PUSH AX

    PUSH SI

    PUSH CX

    CMP AUTH,1

    JNE CHANGE\_ERROR

    CMP GOOD,0

    JE  GOOD\_EMPTY

    MOV BX,GOOD

    ADD BX,10

    MOV BP,6;用于计数

L2:

    CMP BP,0

    JNE CHANGE\_INFO

    JMP FINISH\_CHANGE

CHANGE\_INFO:

    CMP BP,6

    JE L1;因为存储折扣的空间只有一个字节，所以要特殊处理

L3:

    LEA DX,CRLF

    MOV AH,9

    INT 21H

    MOV AX,WORD PTR [BX]

    MOV DX,16

    CALL F2T10;

    MOV DL,'>'

    MOV AH,2

    INT 21H

    IO NUMBER,10;输入要修改成的数字

    CMP BYTE PTR NUMBER+1,0

    JE NEXT\_INFO2;如果直接输入了回车键，则继续修改下一个商品信息

    LEA SI,NUMBER+2

    MOVZX CX,BYTE PTR NUMBER+1

    CALL F10T2

    CMP SI,-1

    JE L3

    MOV WORD PTR [BX],AX

    ADD BX,2

    DEC BP

    JMP L2

L1:

    LEA DX,CRLF

    MOV AH,9

    INT 21H

    MOVZX AX,BYTE PTR [BX]

    MOV DX,16

    CALL F2T10

    MOV DL,'>'

    MOV AH,2

    INT 21H

    IO NUMBER,10;输入要修改成的数字

    CMP BYTE PTR NUMBER+1,0

    JE NEXT\_INFO1;如果直接输入了回车键，则继续修改下一个商品信息

    LEA SI,NUMBER+2;SI为数字串的首地址

    MOVZX CX,BYTE PTR NUMBER+1;CX为数字串的数量

    CALL F10T2;调用函数，将数字串转化为二进制数字，结果存在AX中

    CMP SI,-1

    JE L1;如果输入的数字串不合法，则会要求重新输入

    MOV BYTE PTR [BX],AL;数字串合法，改变折扣的值

    ADD BX,1

    DEC BP

    JMP L2

NEXT\_INFO1:

    ADD BX,1

    DEC BP

    JMP L2

NEXT\_INFO2:

    ADD BX,2

    DEC BP

    JMP L2

CHANGE\_ERROR:

    IO BUF18,9;输出错误提示信息

    JMP RETURN

GOOD\_EMPTY:

    IO BUF\_GOOD\_EMPTY,9

    JMP RETURN

FINISH\_CHANGE:

    IO BUF19,9;提示完成商品信息的修改

    JMP RETURN

RETURN:

    POP CX

    POP SI

    POP AX

    POP DX

    POP BP

    POP BX

    IO CRLF,9

    RET

CHANGE\_GOOD\_INFO ENDP

### 实验记录与分析

1. 将输入输出改成宏指令后，代码简洁了很多
2. 输出商品信息，用masm分别编译主程序代码，F2T10，F10T2的模块，用link 链接三个obj文件（主模块obj文件需要放在最前面），输入2后输入商品名，输出商品信息

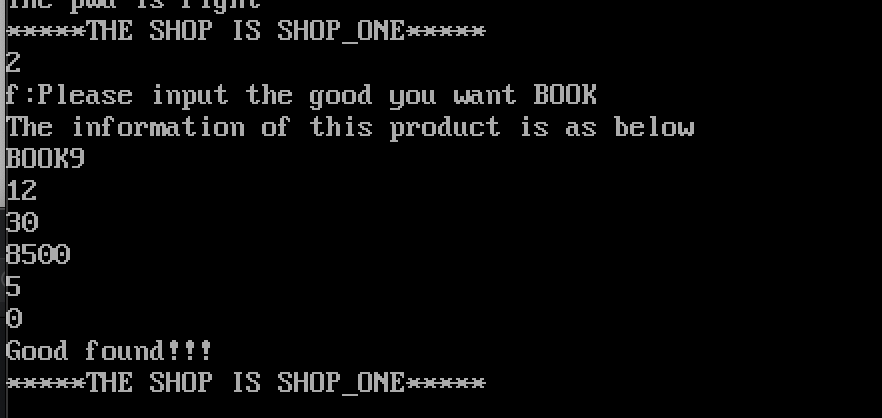


图3.1 查找商品信息

1. 修改商品信息，输入6逐个输入需要修改的信息，不需修改的项可以直接回车，如果用户没有登陆，那么无权修改。

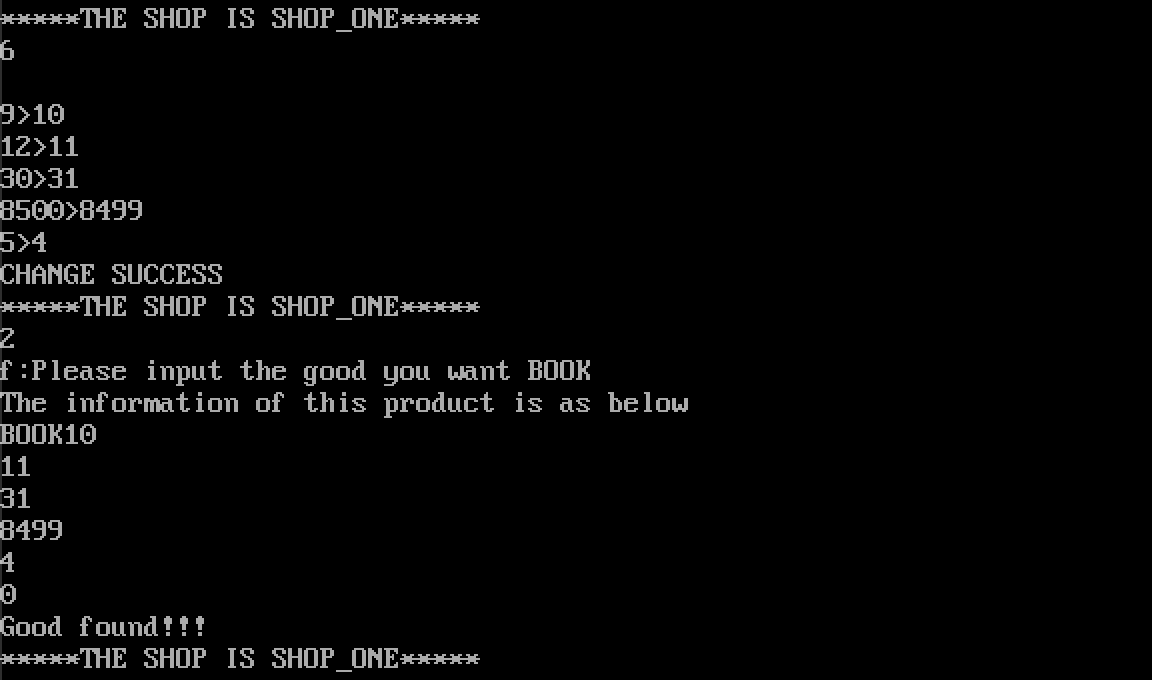


图3.2 修改商品信息

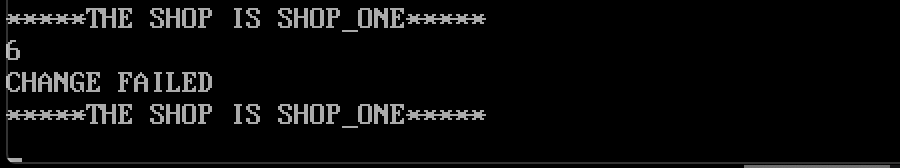


图3.3 未登陆的状态下修改

## 小结

3.1主要学会了模块化的汇编程序语言设计，同时对数据段有了一个更加直观的认识，一开始在编写代码的时候，主函数里的数据段申明为DATA SEGMENT USE16导致调用其他模块的函数后，主函数的数据段的部分数据被修改了，后面经过排查找到问题，把DATA SEGMENT USE16 PARA PUBLIC 'DATA'，防止被错误地修改数据。

### 主要看法

通过这次实验发现自己理论知识学的还不够扎实，还需要巩固。

# 中断与反跟踪

## 实验目的与要求

(1) 熟悉I/O访问，BIOS功能调用方法；

(2） 掌握中断矢量表的概念；

(3) 掌握实方式下中断处理程序的编制与调试方法；

(4) 进一步熟悉内存的一些基本操纵技术；

(5) 熟悉跟踪与反跟踪的技术以及相关的反汇编工具；

(6) 提升对计算机系统的理解与分析能力。

## 实验内容

任务4.1：实现**“7.迁移商店运行环境”的功能。**

在操作系统和虚拟机中，经常要进行内存的调度迁移。这里的迁移运行环境的含义是指将“网店商品信息管理系统”当前的数据段、堆栈段、代码段切换到另外一套数据段、堆栈段和代码段中去，并保证切换前后程序的状态一致（比如，切换前正在浏览某个商品的信息，切换后也应保留该浏览状态）。本次实验只要求切换任务3.1程序的堆栈段。切换的操作是在指定时间下，由中断服务程序完成。

另，为便于观察，需要调整“8.显示当前代码段首址”的功能为：“8.显示当前段寄存器SS的内容”（即按照16进制方式显示这个段寄存器的内容）。

任务4.2：**数据加密与反跟踪**

在任务4.1的**网店商品信息管理程序**的基础上，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（建议采用不少于两种反跟踪方法，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4.3：**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。

建议尽量使用到以下的技术：

1）利用静态反汇编工具（如SOFTICE, OLLYDBG等）将执行程序反汇编成源程序，观察源程序的特点。

2）利用二进制文件编辑工具，直接观察和修改执行文件中的信息（如老板名字信息等）。

3）动态跟踪调试，注意观察和跳过反跟踪的代码。

4）有余力的学生可以设计实现：(a)一个暴力猜解密码的程序；(b)接管键盘的中断服务程序，驻留该程序之后再运行网店商品信息管理程序，截取用户输入用户名之后的字符串信息，保存在指定内存中；退出网店商品信息管理程序之后，用TD去观察中断服务程序记录的字符串信息

## 任务3.1实验过程

### 实验方法说明

切换堆栈段主要在于切换寄存器SS的值，然后编写中断子程序实现定时切换。

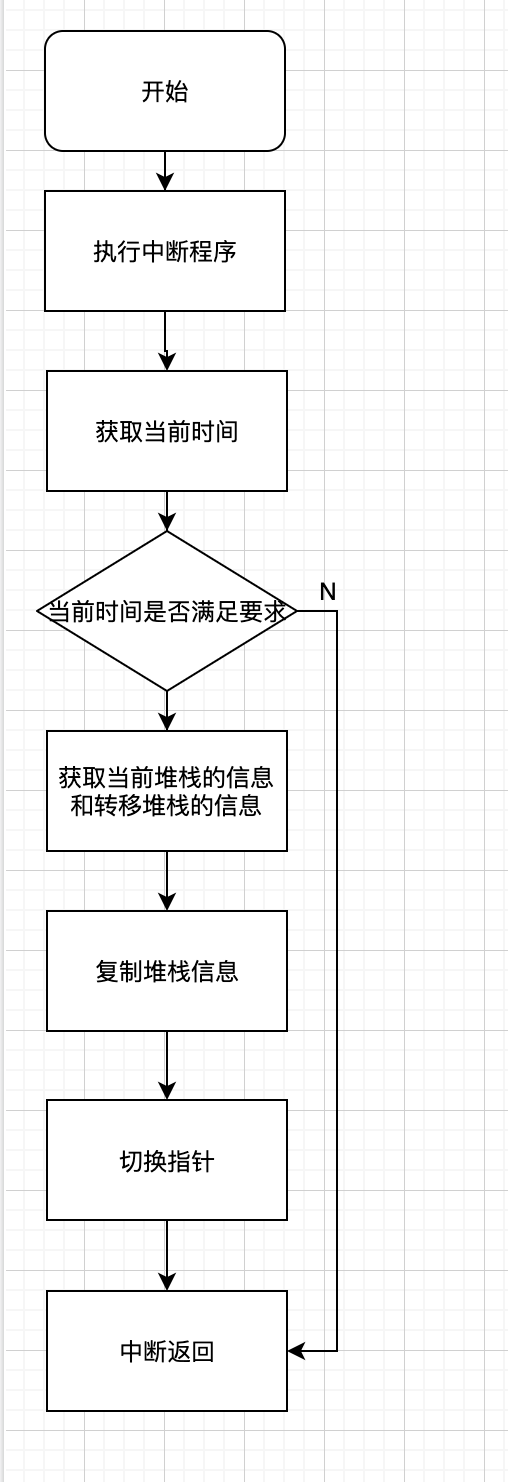


图4.1 堆栈切换逻辑

## 部分代码：

.386

STACK SEGMENT USE16 STACK

      DB 200 DUP(0)

STACK ENDS

STACKBAK SEGMENT USE16

      DB 200 DUP(0)

STACKBAK ENDS

DATA SEGMENT USE16 PARA PUBLIC 'DATA'

...

BREAK\_STATE DB 0;记录中断服务程序装载状态。0是没有装载，1是已经装载

...

DATA ENDS

CODE SEGMENT USE16

    ASSUME CS:CODE,DS:DATA,SS:STACK

  MIN    DB ?

  SEC    DB ?

OLD\_INT  DW  ?, ?

STATE DB 0;用于记录转移堆栈段的状态。

           ;0代表要转到STACK，1代表要转到STACKBAK

...

NEWO8H PROC FAR

    PUSHF

    CALL DWORD PTR CS:OLD\_INT;执行原中断程序

    CALL GET\_TIME;获取时间

    CMP BYTE PTR CS:SEC,0

    JE  CHANGE\_STACK

    IRET

CHANGE\_STACK:

    PUSHA

    CMP BYTE PTR CS:STATE,0;判断目前处于哪一个堆栈

    JE GO\_TO\_STACKBAK

    JMP GO\_TO\_STACK

GO\_TO\_STACKBAK:

    MOV BYTE PTR CS:STATE,1

    MOV DX,STACKBAK

    CALL COPY\_STACK

    POPA

    IRET

GO\_TO\_STACK:

    MOV BYTE PTR CS:STATE,0

    MOV DX,STACK

    CALL COPY\_STACK

    POPA

    IRET

NEWO8H ENDP

GET\_TIME PROC

        PUSH AX

        MOV AL,0

        OUT 70H,AL

        JMP $+2

        IN AL,71H

        MOV BYTE PTR CS:SEC,AL

        POP AX

        RET

GET\_TIME ENDP

COPY\_STACK PROC

;输入参数为DX，DX存储新堆栈段的SS

;该函数功能为将当前堆栈段切换到新堆栈段，并且将旧堆栈段的内容复制到新堆栈段中

    PUSH ES

    MOV ES,DX

    MOV BX,199

CSL:

    MOV AL,SS:[BX]

    MOV ES:[BX],AL

    DEC BX

    CMP BX,0

    JNE CSL

    MOV SS,DX

    POP ES

    RET

COPY\_STACK ENDP

...

;功能7

CHANGE\_STORE\_RUNTIME PROC

    CMP BYTE PTR BREAK\_STATE,0

    JNE ALREADY\_LOAD

    PUSH DS;

    PUSH CS

    POP DS

    MOV AX,3508H

    INT 21H

    MOV WORD PTR OLD\_INT,BX;偏移地址

    MOV WORD PTR OLD\_INT+2,ES;

    MOV DX,OFFSET NEWO8H

    MOV AX,2508H

    INT 21H;

    POP DS

    MOV BYTE PTR BREAK\_STATE,1

    IO BUF21,9

    IO CRLF,9

    RET

ALREADY\_LOAD:

    IO BUF20,9

    IO CRLF,9

    RET

CHANGE\_STORE\_RUNTIME ENDP

4.2加密与反跟踪：通过对数据异或的方式对数据加密，如加密老板密码：

BPASS DB  5 XOR 'Y';密码长度的加密方法，和老板姓名的首字母异或

      DB ('j'-30H)\*2

      DB ('o'-30H)\*2

      DB ('k'-30H)\*2

      DB ('e'-30H)\*2

      DB ('r'-30H)\*2;密码为JOKER，长度为5位

      DB 0A1H,5FH,0D3H;随意填充防止被猜出密码的长度

同时对数据的进价进行加密：

GA1 DB 'PEN',7 DUP(0),10

    DW 35 XOR 'P',56,8500,25,?

对数据进行加密后需要在使用数据时进行解密，如输出商品进价时：

MOV AX, WORD PTR 11[DI]

    MOVZX BX,bYTE PTR [DI]

    XOR AX,BX

    WRITE\_GOOD\_INFO AX;进价

反跟踪手段：主要采用计时反跟踪、检查中断矢量表、堆栈反跟踪来抵制动态调试跟踪，添加冗余信息，

CHECK\_PWd\_LENGTH:

    cli

    mov ah,2ch

    int 21h

    push dx;获取时间并保存

    MOV CL,IN\_PWD+1

    XOR CL,'Y'

    SUB CL,BYTE  PTR BPASS

    MOVSX BX,CL

    ADD BX,OFFSET P1;如果串长相同，BX中存储的是P1的偏移地址

    mov ah,2ch

    int 21h

    sti

    cmp dx,[esp]

    pop dx

    jz OK1;通过时间的检验

    mov bx,[bx];BX可能是PASS1也可能不是PASS1

    add bx,10h;BX一定不是PASS1

    jmp bx;如果时间不同，则说明程序正在被调试，那么就选择跳出程序的正常运行

OK1:

    mov bx,[bx];如果时间符合要求，串长相同的话，BX中存储的是PASS1

               ;如果时间不符合要求，串长不容，BX中存储的不是PASS1

    cmp bx,PASS1;判断BX中存储的是否为PASS1，其实也就是判断是否时间相符并且字符串相同

    je  OK2

    jmp OVER

OK2:jmp bx;跳转到PASS1，对密码进行更进一步的判断

PASS1:

    cli;堆栈检查反跟踪

    push P2;PASS2的地址压栈

    LEA DI,BPASS+1

    LEA SI,IN\_PWD+2

    pop ax

    mov bx,[esp-2];把栈顶上面的字（PASS2的地址）取到

    sti

    jmp bx;如果被跟踪，将不会转移到PASS2

    db ' come on !'

PASS2:

    MOV CL,BYTE PTR BPASS

    XOR CL,'Y';CL此时存储的是真实密码的长度

L\_CHECK\_PWD:

    MOV AL,[DI]

    MOV BL,[SI]

    SUB BL,30H

    ADD BL,BL

    CMP AL,BL

    JNE LOGIN\_PWD\_WRONG

    INC SI

    INC DI

    DEC CL

    JZ L\_AUTH\_1

    LOOP L\_CHECK\_PWD

L\_AUTH\_0:

    ;auth设置为0

    MOV AUTH,0

    JMP MENU

L\_AUTH\_1:

    IO BUF11,9

    IO CRLF,9

    ;auth设置为1

    MOV AUTH,1

    JMP MENU

LOGIN\_NAME\_WRONG:

    IO BUF3,9;提示输出错误的信息

    IO CRLF,9

    JMP LOGIN

LOGIN\_PWD\_WRONG:

    IO BUF6,9

    IO CRLF,9

    JMP L\_IN\_PWD

    db 'come on !'

### 实验记录与分析

1. 为了观察方便，设置每60秒切换一下堆栈，进入dosbox，使用masm编译，link连接后，td运行，定时调用8号功能，

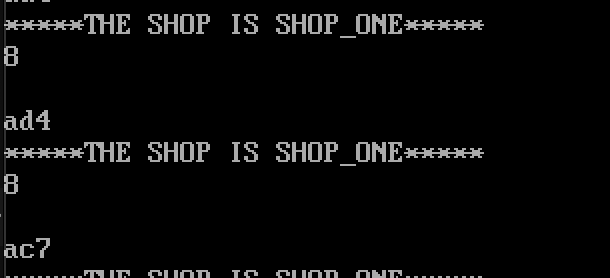


图4.2 SS地址改变

1. 相关问题的回答
   * 1. 如何直接在TD下使用IN/OUT指令获取CMOS数据？

先查CMOS中的数据地址对照表，然后在TD中输入指令段，如读取分钟

MOV AL,2

OUT 70H,AL

JMP $+2

IN AL,71H

* + 1. 压缩BCD码：一个字节表示两个十进制数

如：10010110B表示96D。

* + 1. 如何计算某个中断入口在中断矢量表内的偏移地址？

通过中断号和一个中断矢量的表项（实模式下一个表项为4byte,保护模式下一个表项为8byte）得出偏移地址

* + 1. 程序中如何使用系统功能调用获取中断入口地址？可以在TD中录入指令语句或编写完整程序来尝试。

MOV AX, 3508H;获取8号中断的地址

INT 21H

MOV OLD\_INT,BX

MOV OLD\_INT+2,ES

程序中如何通过直接内存访问的方法获取中断入口地址？

通过直接内存访问

XOR AX,AX

MOV DS,AX

MOV AX,DS[08H\*4]

MOV OLD\_INT,AX

MOV AX,DS[08H\*4+2]

MOV OLD\_INT+2,AX

* + 1. 用TD把中断矢量表里的中断矢量的值随意改成其他值（或改成其他中断的中断矢量）会有什么现象发生？（比如修改21H,1H,3H等的中断矢量，修改后再做些其他操作，比如打开一个执行程序等）

中断矢量中保存的是CS和IP的值，如果修改，在执行中断处理程序的时候可能造成程序的崩溃

* + 1. 有哪两种方式进入原中断服务程序

用CALL和JMP

* + 1. 如何判断中断服务程序已经安装过了？

可以读取对应中断的中断向量表，查询其中断程序地址入口是否和指定的中断服务程序地址一致

* + 1. 如果要切换数据段和代码段，是否会有新的问题需要注意？

切换数据段的过程中不能修改数据段中的值，同时切换后，需要将代码段的CS:IP指向正确的位置，防止死循环。

* + 1. 如果给你的是一片连续的存储空间，你如何将它们定义成合适的堆栈段、数据段和代码段（即给出段首址和分配的空间大小），以满足程序的相应段的迁移要求？（比如，待迁移程序的堆栈段200字节，数据段509字节，代码段321字节；给你的一片存储空间为首址为2000H，大小为1100字节。）

代码段：2000H， 324字节

数据段：2144H, 512字节

堆栈段：2344H, 196字节

1. 加密解密测试，将老板的密码和商品进价加密后，测试功能是否还正确。

测试用户登陆

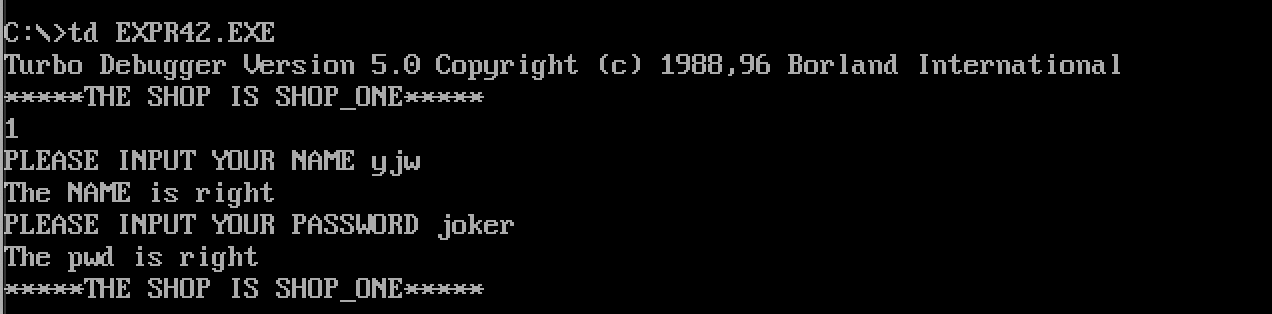


图4.3 测试密码加密后用户登陆功能

测试商品进价输出

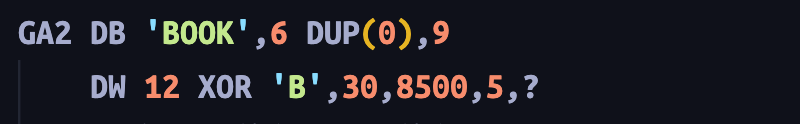


图4.4 BOOK数据定义

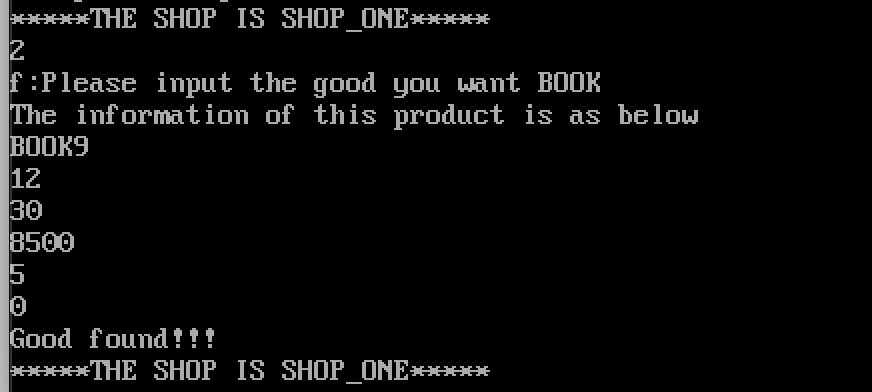


图4.5 解密进价输出。

1. 跟踪和数据解密

对程序进行单步调试，设置完新1号和3号中断矢量后，td进入卡死状态，单步调试无法进行。合作成员：CS1706干佳琪，CS1706白哲

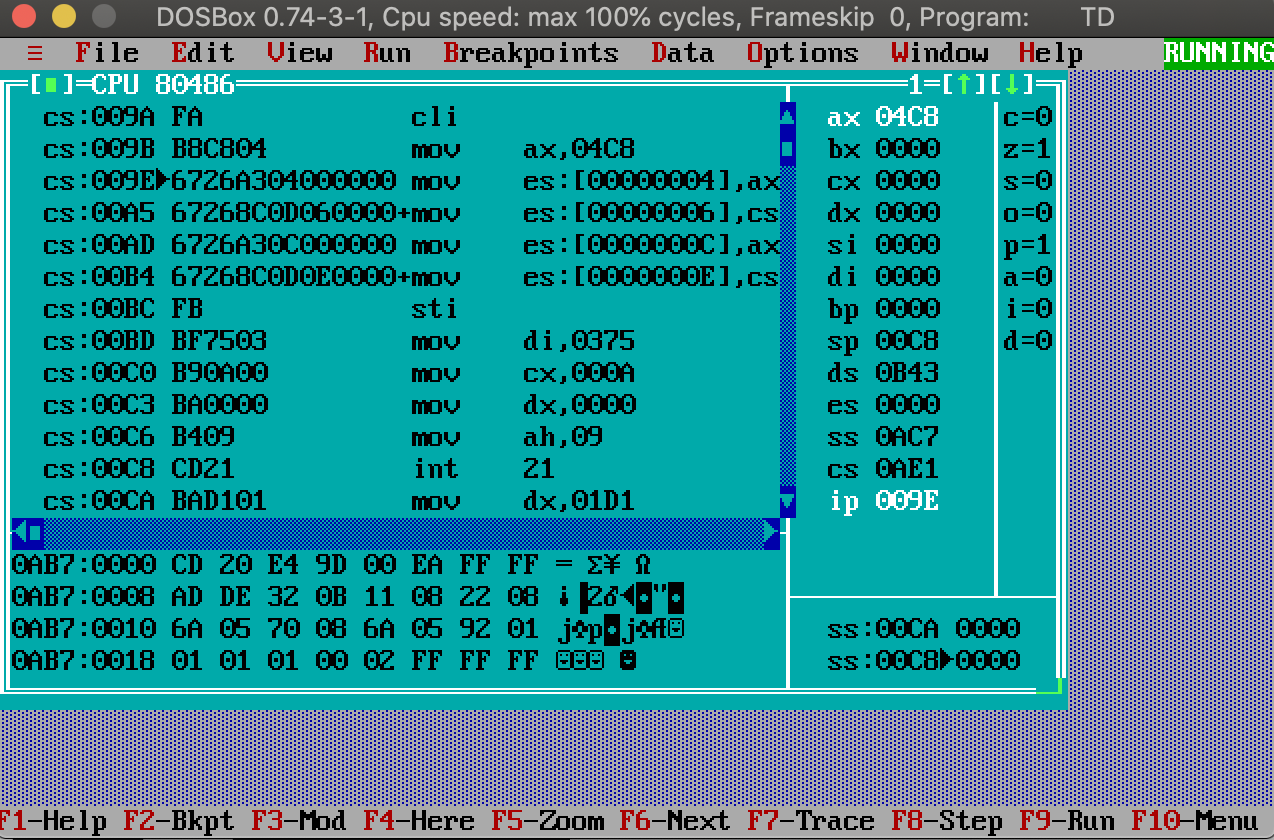


图4.6 td尝试单步调试。

1. 尝试破解同学的程序

因为单步调试无法进行，使用TD进行动态调试，跟踪到cli和sti，发现有修改中断向量表的代码，将其修改为nop。向下跟踪可以观察到，输入密码后，可以发现循环异或后进行比较的操作，可能是进行加密。进过观察，发现该同学的加密方法为：(明文-29h)\*3。

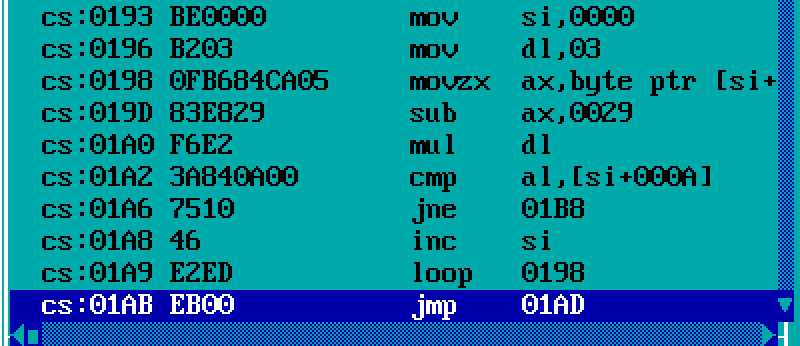


图4.7 密码比较循环

通过上图可以找到密码存放的地址，DX=000AH，查看对应ds区域内存，

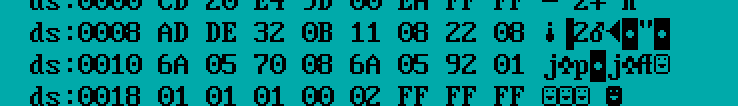


图4.8 加密后的密码

## 小结

4.1主要学会了中断的基本使用，对于中断有了一个基本的认识，但是在开发中仍然碰到了很多问题，最后通过仔细阅读书本得以解决，理论只是还是不够扎实。

4.2 实验是让我们利用之前学习的各种汇编知识运用到加密以及解密上，是一次全新的体验，让我们体会到了汇编在加密以及解密上的巨大作用，汇编语言由于可以更加直接的对硬件进行操作，所以在反跟踪和加密等方面比高级语言更具有优势。

4.3 实验解密同学的代码，因为同学的代码比较简单所以很容易即可解出来，越难发现规律的

加密方式是越难被察觉的。

同时，通过加密与解密的过程，让我重新审视之前编写的代码，让我对程序编写的安全性有了更加深刻的理解。

# WIN32程序设计

## 实验目的与要求

（1）熟悉WIN32/64程序的设计和调试方法；

（2）了解不同操作系统环境下开发工具的特点；

（3）熟悉宏汇编语言中INVOKE、结构变量、简化段定义等功能；

（4）进一步理解机器语言、汇编语言、高级语言之间以及实方式、保护方式之间的一些关系；了解16位段程序移植到32/64位段程序时需要注意的问题。

## 实验内容

**任务5.1** 编写一个基于窗口的WIN32/64程序，实现**网店商品信息管理系统**的部分功能。也即：以任务3.1为基础，将其部分功能移植过来，具体要求如下描述。

1.编写一个基于窗口的WIN32/64程序的菜单框架，具有以下的下拉菜单项：

File Action Help

Exit Recommendation About

List Sort

点菜单File下的Exit选项时结束程序；点菜单Help下的选项About，弹出一个消息框，显示本人信息。点菜单Action下的选项Recommendation、List Sort将分别实现计算推荐度或显示排序后的SHOP所有商品信息的功能。

2.要求采用结构变量存放商品的相关信息。商品数定义5种左右。

（1）点菜单项Recommendation时，按照任务3.1的方法计算所有商品的推荐度。

（2）点菜单项List Sort时，先对所有商品按照推荐度从高到低排序，然后按照排序结果在窗口中列出SHOP的所有商品的信息。具体显示格式自行定义，可以参照图5.2的样式（不要求用中文）。

## 5.3 实验过程

### 5.3.1 设计思想与变量分配

1．该基于窗口的程序是基于WIN32的标准框架实现的。该程序中还用到了WIN32的标准框架，其中包含了主程序、窗口主程序、窗口消息、处理程序以及用户处理程序。操作系统首先执行主程序，待主程序获得与本程序有关的基本信息后，在调用窗口主程序。窗口主程序创建窗口后，将说道的消息通过操作系统发送给窗口消息处理程序。窗口消息处理程序判断收到消息的类型，完成相应的功能

2.菜单栏的实现用到了.rc文件。在主程序winmain中，程序将对菜单资源进行装载。再在窗口过程中对相应的菜单功能进行设计，即可完成相关菜单功能的实现。在实现菜单功能时，需要对子函数进行调用。在本次实验中，需要编写的子程序为计算推荐度和显示商品信息的子函数。

3.使用结构体存放商品的相关信息：



图5.1 商品结构体定义

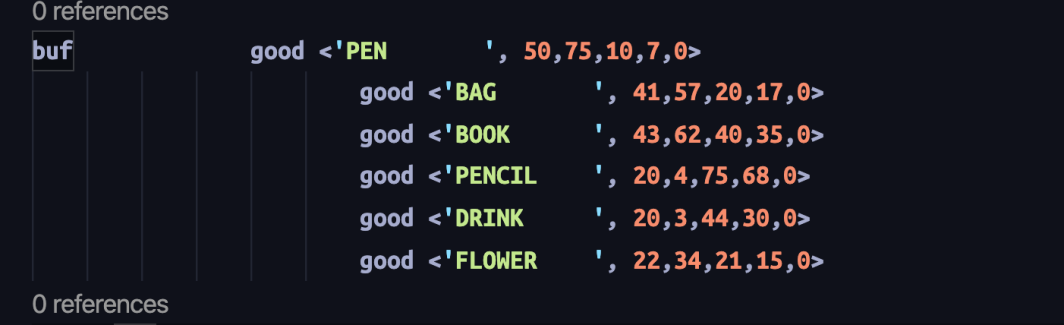


图5.2 商品数据

### 5.3.2 源程序

**.386**

**.model   flat,stdcall**

**option   casemap:none**

**WinMain  proto :DWORD,:DWORD,:DWORD,:DWORD**

**WndProc  proto :DWORD,:DWORD,:DWORD,:DWORD**

**Display  proto :DWORD**

**GetNumber proto:DWORD,:DWORD**

**Average  proto**

**include      newmenuID.INC**

**include      windows.inc**

**include      user32.inc**

**include      kernel32.inc**

**include      gdi32.inc**

**include      shell32.inc**

**includelib   user32.lib**

**includelib   kernel32.lib**

**includelib   gdi32.lib**

**includelib   shell32.lib**

**good         struct**

**goodname   db  10 dup(0)**

**invalue  db  0**

**outvalue     db  0**

**insum  db  0**

**outsum  db  0**

**profit    db  0**

**good      ends**

**.data**

**ClassName    db       'TryWinShop',0**

**AppName      db       'Our First Window',0**

**MenuName     db       'MyMenu',0**

**DlgName      db       'MyDialog',0**

**AboutMsg     db       'Gan Jiaqi, CS1706,U201714058',0**

**hInstance    dd       0**

**CommandLine  dd       0**

**buf             good <'PEN       ', 50,75,10,7,0>**

**good <'BAG       ', 41,57,20,17,0>**

**good <'BOOK      ', 43,62,40,35,0>**

**good <'PENCIL    ', 20,4,75,68,0>**

**good <'DRINK     ', 20,3,44,30,0>**

**good <'FLOWER    ', 22,34,21,15,0>**

**numberbuf        db     100 dup(0)**

**msg\_name     db       'name',0**

**msg\_inval  db       'in\_value',0**

**msg\_outval     db       'out\_value',0**

**msg\_insum  db       'in\_sum',0**

**msg\_outsum  db       'out\_sum',0**

**msg\_profit    db       'profit',0**

**list\_flag           db          0**

**.code**

**Start:       invoke GetModuleHandle,NULL**

**mov    hInstance,eax**

**invoke GetCommandLine**

**mov    CommandLine,eax**

**invoke WinMain,hInstance,NULL,CommandLine,SW\_SHOWDEFAULT**

**invoke ExitProcess,eax**

**;;**

**WinMain      proc   hInst:DWORD,hPrevInst:DWORD,CmdLine:DWORD,CmdShow:DWORD**

**LOCAL  wc:WNDCLASSEX**

**LOCAL  msg:MSG**

**LOCAL  hWnd:HWND**

**invoke RtlZeroMemory,addr wc,sizeof wc**

**mov    wc.cbSize,SIZEOF WNDCLASSEX**

**mov    wc.style, CS\_HREDRAW or CS\_VREDRAW**

**mov    wc.lpfnWndProc, offset WndProc**

**mov    wc.cbClsExtra,NULL**

**mov    wc.cbWndExtra,NULL**

**push   hInst**

**pop    wc.hInstance**

**mov    wc.hbrBackground,COLOR\_WINDOW+1**

**mov    wc.lpszMenuName, offset MenuName**

**mov    wc.lpszClassName,offset ClassName**

**invoke LoadIcon,NULL,IDI\_APPLICATION**

**mov    wc.hIcon,eax**

**mov    wc.hIconSm,0**

**invoke LoadCursor,NULL,IDC\_ARROW**

**mov    wc.hCursor,eax**

**invoke RegisterClassEx, addr wc**

**INVOKE CreateWindowEx,NULL,addr ClassName,addr AppName,\**

**WS\_OVERLAPPEDWINDOW,CW\_USEDEFAULT,\**

**CW\_USEDEFAULT,CW\_USEDEFAULT,CW\_USEDEFAULT,NULL,NULL,\**

**hInst,NULL**

**mov    hWnd,eax**

**INVOKE ShowWindow,hWnd,SW\_SHOWNORMAL**

**INVOKE UpdateWindow,hWnd**

**;;**

**MsgLoop:     INVOKE GetMessage,addr msg,NULL,0,0**

**cmp    EAX,0**

**je     ExitLoop**

**INVOKE TranslateMessage,addr msg**

**INVOKE DispatchMessage,addr msg**

**jmp    MsgLoop**

**ExitLoop:    mov    eax,msg.wParam**

**ret**

**WinMain      endp**

**WndProc      proc   hWnd:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD**

**LOCAL  hdc:HDC**

**.IF     uMsg == WM\_DESTROY**

**invoke PostQuitMessage,NULL**

**;.ELSEIF uMsg == WM\_KEYDOWN**

**;.IF     wParam == VK\_F1**

**;invoke MessageBox,hWnd,addr AboutMsg, addr AppName,MB\_OK**

**;.ENDIF**

**.ELSEIF uMsg == WM\_COMMAND**

**.IF     wParam == IDM\_FILE\_EXIT**

**invoke SendMessage,hWnd,WM\_CLOSE,0,0**

**.ELSEIF wParam == IDM\_ACT\_LIST**

**invoke Display,hWnd**

**.ELSEIF wParam == IDM\_ACT\_AVE**

**invoke Average**

**.ELSEIF wParam == IDM\_HELP\_ABOUT**

**invoke MessageBox,hWnd,addr AboutMsg,addr AppName,MB\_OK**

**.ENDIF**

**.IF     wParam == IDM\_ACT\_LIST**

**mov list\_flag, 1**

**.ENDIF**

**;.ELSEIF uMsg == WM\_PAINT**

**.IF list\_flag != 0**

**invoke Display,hWnd**

**.ENDIF**

**.ELSE**

**invoke DefWindowProc,hWnd,uMsg,wParam,lParam**

**ret**

**.ENDIF**

**xor    eax,eax**

**ret**

**WndProc      endp**

**Display      proc   hWnd:DWORD**

**XX     equ  10**

**YY     equ  10**

**XX\_GAP equ  100**

**YY\_GAP equ  30**

**LOCAL  hdc:HDC**

**LOCAL  Ycount:DWORD**

**LOCAL  goods:DWORD**

**push ecx**

**mov Ycount, YY+YY\_GAP**

**;mov Xcount, XX**

**invoke GetDC,hWnd**

**mov    hdc,eax**

**invoke TextOut,hdc,XX+0\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_name,4**

**invoke TextOut,hdc,XX+1\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_inval,5**

**invoke TextOut,hdc,XX+2\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_outval,6**

**invoke TextOut,hdc,XX+3\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_insum,5**

**invoke TextOut,hdc,XX+4\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_outsum,6**

**invoke TextOut,hdc,XX+5\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_profit,6**

**mov ecx, 6**

**mov goods, offset buf[0].goodname**

**DLOOP:               push ecx**

**;;**

**invoke TextOut,hdc,XX+0\*XX\_GAP,Ycount,goods,10**

**add goods, 10**

**invoke GetNumber, goods, addr numberbuf**

**invoke TextOut,hdc,XX+1\*XX\_GAP,Ycount,addr numberbuf,2**

**add goods, 1**

**invoke GetNumber, goods, addr numberbuf**

**invoke TextOut,hdc,XX+2\*XX\_GAP,Ycount,addr numberbuf,2**

**add goods, 1**

**invoke GetNumber, goods, addr numberbuf**

**invoke TextOut,hdc,XX+3\*XX\_GAP,Ycount,addr numberbuf, 2**

**add goods, 1**

**invoke GetNumber, goods, addr numberbuf**

**invoke TextOut,hdc,XX+4\*XX\_GAP,Ycount,addr numberbuf, 2**

**add goods, 1**

**invoke GetNumber, goods, addr numberbuf**

**invoke TextOut,hdc,XX+5\*XX\_GAP,Ycount,addr numberbuf, 2**

**add Ycount, YY\_GAP**

**add goods, 1**

**pop ecx**

**dec ecx**

**jnz near ptr DLOOP**

**pop ecx**

**ret**

**Display      endp**

**Average proc**

**push eax**

**push ebx**

**push ecx**

**push edx**

**push esi**

**mov ecx, 6**

**lea esi, buf**

**add esi, 10**

**AVELOOP:**

**mov al, [esi]**

**mov ah, [esi+2]**

**mul ah**

**mov bx, ax**

**mov al, [esi+1]**

**mov ah, [esi+3]**

**mul ah**

**sub ax, bx**

**imul ax, 100**

**cwd**

**idiv bx**

**mov [esi+4], al**

**add esi, 15**

**loop AVELOOP**

**pop esi**

**pop edx**

**pop ecx**

**pop ebx**

**pop eax**

**ret**

**Average endp**

**GetNumber proc numberAddr:DWORD, bufAddr:DWORD**

**push esi**

**push eax**

**push ebx**

**push edx**

**mov esi, numberAddr**

**mov al, byte ptr [esi]**

**mov dl, 10**

**div dl**

**mov esi, bufAddr**

**movzx ebx, al**

**add ebx, 30H**

**mov [esi], ebx**

**movzx ebx, ah**

**add ebx, 30H**

**mov [esi+1], ebx**

**pop edx**

**pop ebx**

**pop eax**

**pop esi**

**ret**

**GetNumber endp**

**end  Start**

### 5.3.3 实验步骤

1.根据实验要求下载MASM32和窗口程序框架。

2.运行VS2019创建一个空项目（名称为vc\_win32ASM）

3.设置工程的INCLUDE目录，增加汇编语言依赖项,导入汇编源文件。

4. 根据功能要求修改窗口程序框架的代码，修改结构体，添加求推荐度的程序，数字转换程序以及推荐度排序展示的程序等等。

5.编译生成vc\_win32ASM.exe，进行源码级的调试

6.根据程序运行情况进行调试，记录情况

## 。。。。。。。。

**实验四代码：**

EXTERN F2T10:FAR

EXTERN F10T2:FAR

.386

IO MACRO A,B

    LEA DX,A

    MOV AH,B

    INT 21H

    ENDM

WRITE\_GOOD\_INFO MACRO A;输出商品的数字信息

    MOV AX,A

    MOV DX,16

    CALL F2T10

    ENDM

STACK SEGMENT USE16 STACK

    DB 200 DUP(0)

STACK ENDS

STACKBAK SEGMENT USE16

      DB 200 DUP(0)

STACKBAK ENDS

DATA SEGMENT USE16 PARA PUBLIC 'DATA'

BUF  DB '\*\*\*\*\*THE SHOP IS SHOP\_ONE\*\*\*\*\*$'

BUF1 DB 'PLEASE INPUT YOUR NAME $'

BUF2 DB 'PLEASE INPUT YOUR PASSWORD $'

BUF3 DB 'The name is wrong $'

BUF4 DB 'f:Please input the good you want $'

BUF5 DB 'The NAME is right $'

BUF6 DB 'The PWD is wrong$'

BUF7 DB 'Landed successfully$'

BUF8 DB 'Good you want not exist$'

BUF9 DB 'Good is not remain$'

BUF10 DB '0123456789abcdef'

BUF11 DB 'The pwd is right $'

BUF12 DB 'Good found!!!$'

BUF13 DB 'count !!!$'

BUF14 DB 'update success !!!$'

BUF15 DB 'hear!$'

BUF16 DB 'The information of this product is as below$';提示：提示用户以下是查找到的商品的信息

BUF17 DB 'Find\_GOOD$';

BUF18 DB 'CHANGE FAILED $';

BUF19 DB 'CHANGE SUCCESS $';

BUF20 DB 'INTERRUPT SERVICE ALREADY LOADED! $';

BUF21 DB 'INTERRUPT SERVICE LOADED! $';

BUF22 DB 'Please enter the number 1 to 9 to select the function',0DH,0AH

     DB '1. Log in / re log in',0DH,0AH

     DB '2. Find the specified product and display its information',0DH,0AH

     DB '3. Placing an order',0DH,0AH

     DB '4. Calculate commodity recommendation',0DH,0AH

     DB '5. Ranking', 0DH,0AH

     DB '6. Change product information', 0DH,0AH

     DB '7. Change the store operation environment', 0DH,0AH

     DB '8. Display the current stack segment first address',0DH,0AH

     DB '9. Exit',0DH,0AH,'$'

BUF\_GOOD\_EMPTY DB 'GOOD EMPTY $';

AUTH DB ?

CRLF DB 0DH,0AH,'$'

IN\_NAME DB 11

    DB ?

    DB 11 DUP(0)

IN\_PWD   DB 7

    DB ?

    DB 7 DUP(0)

IN\_GOOD DB 11

    DB ?

    DB 11 DUP(0)

BNAME DB 'yjw',3 DUP(0)

COUNT1 = $-BNAME

BPASS DB  5 XOR 'Y';密码长度的加密方法，和老板姓名的首字母异或

      DB ('j'-30H)\*2

      DB ('o'-30H)\*2

      DB ('k'-30H)\*2

      DB ('e'-30H)\*2

      DB ('r'-30H)\*2;密码为JOKER，长度为5位

      DB 0A1H,5FH,0D3H;随意填充防止被猜出密码的长度

COUNT2 = $-BPASS

N EQU 3000

S1  DB 'SHOP\_ONE',0

GA1 DB 'PEN',7 DUP(0),10

    DW 35 XOR 'P',56,8500,25,?

GA2 DB 'BOOK',6 DUP(0),9

    DW 12 XOR 'B',30,8500,5,?

    ; 实际销售价格=销售价\*折扣/10

    ; 进货价(字类型)，

    ; 销售价（字类型），

    ; 进货总数（字类型），

    ; 已售数量（字类型）

    ; 推荐度

GA3 DB 'PAD',7 DUP(0),10

    DW 12 XOR 'M',30,8500,5,?

GAN DB N-3 DUP('TEMP-VALUE',8,15,0,20,0,30,0,2,0,?,?)

NUMBER DB 10;用于保存修改商品信息时输入的数字串

       DB ?

       DB 10 DUP(0)

GOOD DW 0

M EQU 100

COUNTM DW 5000

P1   DW PASS1

P2   DW PASS2

BREAK\_STATE DB 0;记录中断服务程序装载状态，0是没有装载，1是已经装载

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

DATA ENDS

CODE SEGMENT USE16

    ASSUME CS:CODE,DS:DATA,SS:STACK

MIN DB ?;分钟

SEC DB ?;秒

OLD\_INT DW ?,?;记录原中断程序信息

STATE DB 0;用于记录转移堆栈段的状态

SSEG       DW    ?,  ?  ;用于保存2个堆栈段的段地址

NEWO8H PROC FAR

    PUSHF

    CALL DWORD PTR CS:OLD\_INT;执行原中断程序

    CALL GET\_TIME;获取时间

    CMP BYTE PTR CS:SEC,0

    JE  CHANGE\_STACK

    IRET

CHANGE\_STACK:

    PUSHA

    CMP BYTE PTR CS:STATE,0;判断目前处于哪一个堆栈

    JE GO\_TO\_STACKBAK

    JMP GO\_TO\_STACK

GO\_TO\_STACKBAK:

    MOV BYTE PTR CS:STATE,1

    MOV DX,STACKBAK

    CALL COPY\_STACK

    POPA

    IRET

GO\_TO\_STACK:

    MOV BYTE PTR CS:STATE,0

    MOV DX,STACK

    CALL COPY\_STACK

    POPA

    IRET

NEWO8H ENDP

GET\_TIME PROC

    PUSH AX

    MOV AL,0;秒的偏移

    OUT 70H,AL

    JMP $+2

    IN AL,71H;读取秒的信息

    MOV BYTE PTR CS:SEC,AL

    POP AX

    RET

GET\_TIME ENDP

COPY\_STACK PROC

    PUSH ES

    MOV ES,DX

    MOV BX,199

CSL:

    MOV AL,SS:[BX];循环拷贝堆栈

    MOV ES:[BX],AL

    DEC BX

    CMP BX,0

    JNE CSL

    MOV SS,DX;切换堆栈

    POP ES

    RET

COPY\_STACK ENDP

START:

    MOV AX,DATA

    MOV DS,AX

    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

    LEA DI,GA1

    MOV CX,10

MENU:

    ; 输出商店名称

    IO BUF,9

    ; 输出空格

    IO BUF22,9

    IO CRLF,9

L\_INPUT:

    ;输入一个字符

    MOV AH,1

    INT 21H

    IO CRLF,9

    CMP AL,'1'

    JE LOGIN

    CMP AL,'2'

    JNE FUNC3

    CALL FIND\_GOOD

    JMP MENU

FUNC3:

    CMP AL,'3'

    JE ORDER

FUNC4:

    CMP AL,'4'

    JE COUNT\_PRODUCE

    CMP AL,'5'

    JE RANK

    CMP AL,'6'

    JNE FUNC7

    CALL CHANGE\_GOOD\_INFO

    JMP MENU

FUNC7:

    CMP AL,'7'

    JNE FUNC8

    CALL CHANGE\_STORE\_RUNTIME

    JMP MENU

FUNC8:

    CMP AL,'8'

    JNE FUNC9

    CALL SHOW\_CURRENT\_CS

    JMP MENU

FUNC9:

    CMP AL,'9'

    JNE MENU

    JMP OVER

LOGIN:

    ;请输入姓名

    LEA DX,BUF1

    MOV AH,9

    INT 21H

    ;LEA DX,IN\_NAME

    ;MOV AH,10

    ;INT 21H

    IO IN\_NAME,10

    LEA DX,CRLF

    MOV AH,9

    INT 21H

    ;判断是否仅输入回车，若是，直接跳到

    CMP IN\_NAME+1,0

    JE MENU

   ;长度不一致

L\_CHECK\_NAME:

    LEA SI,IN\_NAME

    LEA DI,BNAME

    MOV CL,1[SI]

    CMP CL,3

    JNE LOGIN\_NAME\_WRONG

L\_LOOP\_N:

    MOV AL,2[SI]

    MOV BL,[DI]

    INC SI

    INC DI

    CMP AL,BL

    JNE LOGIN\_NAME\_WRONG

    LOOP L\_LOOP\_N

    LEA DX,BUF5

    MOV AH,9

    INT 21H

    LEA DX,CRLF

    MOV AH,9

    INT 21H

L\_IN\_PWD:

    LEA DX,BUF2;输入密码

    MOV AH,9

    INT 21H

    LEA DX,IN\_PWD

    MOV AH,10

    INT 21H

    LEA DX,CRLF

    MOV AH,9

    INT 21H

    ;比较密码

CHECK\_PWd\_LENGTH:

    cli

    mov ah,2ch

    int 21h

    push dx;获取时间并保存

    MOV CL,IN\_PWD+1

    XOR CL,'Y'

    SUB CL,BYTE  PTR BPASS

    MOVSX BX,CL

    ADD BX,OFFSET P1;如果串长相同，BX中存储的是P1的偏移地址

    mov ah,2ch

    int 21h

    sti

    cmp dx,[esp]

    pop dx

    jz OK1;通过时间的检验

    mov bx,[bx];BX可能是PASS1也可能不是PASS1

    add bx,10h;BX一定不是PASS1

    jmp bx;如果时间不同，则说明程序正在被调试，那么就选择跳出程序的正常运行

OK1:

    mov bx,[bx];如果时间符合要求，串长相同的话，BX中存储的是PASS1

               ;如果时间不符合要求，串长不容，BX中存储的不是PASS1

    cmp bx,PASS1;判断BX中存储的是否为PASS1，其实也就是判断是否时间相符并且字符串相同

    je  OK2

    jmp OVER

OK2:jmp bx;跳转到PASS1，对密码进行更进一步的判断

PASS1:

    cli;堆栈检查反跟踪

    push P2;PASS2的地址压栈

    LEA DI,BPASS+1

    LEA SI,IN\_PWD+2

    pop ax

    mov bx,[esp-2];把栈顶上面的字（PASS2的地址）取到

    sti

    jmp bx;如果被跟踪，将不会转移到PASS2

    db ' come on !'

PASS2:

    MOV CL,BYTE PTR BPASS

    XOR CL,'Y';CL此时存储的是真实密码的长度

L\_CHECK\_PWD:

    MOV AL,[DI]

    MOV BL,[SI]

    SUB BL,30H

    ADD BL,BL

    CMP AL,BL

    JNE LOGIN\_PWD\_WRONG

    INC SI

    INC DI

    DEC CL

    JZ L\_AUTH\_1

    LOOP L\_CHECK\_PWD

L\_AUTH\_0:

    ;auth设置为0

    MOV AUTH,0

    JMP MENU

L\_AUTH\_1:

    IO BUF11,9

    IO CRLF,9

    ;auth设置为1

    MOV AUTH,1

    JMP MENU

LOGIN\_NAME\_WRONG:

    IO BUF3,9;提示输出错误的信息

    IO CRLF,9

    JMP LOGIN

LOGIN\_PWD\_WRONG:

    IO BUF6,9

    IO CRLF,9

    JMP L\_IN\_PWD

    db 'come on !'

ORDER:

    CMP GOOD,0

    JE MENU

    MOV AX,0

    CALL TIMER

LOOP\_M:

    MOV DI,GOOD

    ;进货数量

    MOV AX,WORD PTR [DI]+15

    ;销量

    MOV BX,WORD PTR [DI]+17

    CMP AX,BX;比较

    JE GOOD\_NO\_REMAIN ;进货数量等于已售数量，显示已经售完

    MOV AX, WORD PTR [DI]+17

    ADD AX,1

    MOV WORD PTR [DI]+17,AX

COUNT\_PRODUCE:

    MOV EBP,N;

    XOR EDI,EDI

    LEA DI,GA1

; 计算推荐度

COUNT\_LOOP:

    MOV AX, WORD PTR [EDI]+11

    ;解密

    MOVZX BX,BYTE PTR [EDI]

    XOR AX,BX

    MOV CX, 10

    IMUL CX

    MOV SI, WORD PTR [EDI]+13

    MOVZX BX, BYTE PTR[EDI]+10

    IMUL SI, BX

    DIV SI ;AX中为商，DX中为余数

    MOVZX ESI,AX ;将商移动到SI中储存

     ;以上部分为推荐度的左半部分

    MOVZX EAX,WORD PTR [EDI]+17;已售数量

    ;位移

    MOV ECX,64

    IMUL ECX

    ;SHL EAX,6;位移

    XOR EDX,EDX

    MOVZX EBX,WORD PTR [EDI]+15;进货量

    DIV EBX;

    ADD ESI,EAX;SI即为推荐度

    MOV 19[DI],SI;将推荐度储存

    ADD DI,21;将DI指向下一商品信息段

    DEC EBP;查看下一商品段信息，BP加一

    CMP EBP,0;

    JNE COUNT\_LOOP

    MOV AX,1        ;计时

    CALL TIMER

    ;LEA DX,BUF13 ;调试

    ;MOV AH,9

    ;INT 21H

    ; 空格

    ;LEA DX,CRLF

    ;MOV AH,9

    ;INT 21H

    MOV GOOD,0;

    JMP MENU

GOOD\_NO\_REMAIN:

    LEA DX,BUF9

    MOV AH,9

    INT 21H

    ; 空格

    LEA DX,CRLF

    MOV AH,9

    INT 21H

    JMP MENU

RANK:

    JMP MENU

MODIFY\_GOOD:

    JMP MENU

CHANGE\_STORE\_RUNTIME PROC

    CMP BYTE PTR BREAK\_STATE,0

    JNE ALREADY\_LOAD

    PUSH DS;

    PUSH CS

    POP DS

    MOV AX,3508H

    INT 21H

    MOV WORD PTR OLD\_INT,BX;偏移地址

    MOV WORD PTR OLD\_INT+2,ES;

    MOV DX,OFFSET NEWO8H

    MOV AX,2508H

    INT 21H;

    POP DS

    MOV BYTE PTR BREAK\_STATE,1

    IO BUF21,9

    IO CRLF,9

    RET

ALREADY\_LOAD:

    IO BUF20,9

    IO CRLF,9

    RET

CHANGE\_STORE\_RUNTIME ENDP

SHOW\_CURRENT\_CS PROC

    PUSH CX

    PUSH AX

    PUSH DX

    PUSH BX

    IO CRLF,9

    MOV AX,SS;改写

    MOV BX,16

    MOV CX,0;

L\_LOOP\_SHOW\_CS:

    MOV DX,0

    DIV BX;商在AX,余数在DX中

    PUSH DX;

    INC CX;

    CMP AX,0

    JE SHOW\_CS

    JMP L\_LOOP\_SHOW\_CS

SHOW\_CS:

    CMP CX,0

    JE SHOW\_END

    POP BX

    MOVZX DX,BUF10[BX];

    MOV AH,2

    INT 21H

    DEC CX

    JMP SHOW\_CS

SHOW\_END:

    IO CRLF,9

    POP BX

    POP DX

    POP AX

    POP CX

    RET

SHOW\_CURRENT\_CS ENDP

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

    MOV   AH,4CH

    MOV   AL, 0;退出码 (如0、0FFH等)

    INT   21H

FIND\_GOOD PROC

    ;;提示输入商品名称

    IO BUF4,9

    ; 输入商品的名字

    LEA DX,IN\_GOOD

    MOV AH,10

    INT 21H

    ;比较长度

    CMP IN\_GOOD+1,0

    JNE L\_CHECK\_GOOD\_F

    JMP GOOD\_NOT\_FOUND\_F

L\_CHECK\_GOOD\_F:

    IO CRLF,9

    LEA DI,GA1

    LEA SI,IN\_GOOD+2

    MOVZX CX,IN\_GOOD+1;CX保存输入商品单词的个数

    MOV DX,0;DX代表商品信息的序号（从0-29）

LL\_F:

    MOV BX,0;BX代表商品名字比较位置（从0开始计数）

L\_LOOP\_G\_F:

    CMP BX,CX

    JGE FURTHER\_CHECK;，以防输入的商品名为储存商品名的子集

    MOV AL,[DI+BX]

    MOV AH,[SI+BX]

    CMP AH,AL

    JNE L\_NEXT\_GOOD\_F;如果不相等，判断下一条商品信息

    INC BX;如果相等，判断下一个字母是否相等

    JMP L\_LOOP\_G\_F

L\_NEXT\_GOOD\_F:;进入下个商品的位置

    ADD DI,21;将DI移动到下个商品信息段

    INC DX;商品信息序号相应加一

    CMP DX,30

    JE GOOD\_NOT\_FOUND\_F

    JMP LL\_F

FURTHER\_CHECK:

    CMP BYTE PTR[DI+BX],0;判断储存字段该处是否为0

    JE MARK\_GOOD\_F;若是0，则表示输入名字不是储存名字的子集

    JMP L\_NEXT\_GOOD\_F;若不是0，则说明输入名字为储存名字的子集，还需要检查下一条商品信息段

MARK\_GOOD\_F:

    MOV GOOD,DI;将该商品信息段的首地址储存到GOOD字段中

    MOVZX BX,IN\_GOOD+1;将该商品名字单词的长度赋值给BX

    IO BUF16,9;输出提示信息

    IO CRLF,9

    MOV BYTE PTR [DI+BX],'$';在名字的末尾添加$

    IO [DI],9;输出商品的名字

    MOV BYTE PTR [DI+BX],0

    MOVZX AX,BYTE PTR 10[DI]

    MOV DX,16

    CALL F2T10;输出商品折扣

    IO CRLF,9

    ;

    MOV AX, WORD PTR 11[DI]

    MOVZX BX,bYTE PTR [DI]

    XOR AX,BX

    WRITE\_GOOD\_INFO AX;进价

    IO CRLF,9

    WRITE\_GOOD\_INFO 13[DI];售价

    IO CRLF,9

    WRITE\_GOOD\_INFO 15[DI];进货数

    IO CRLF,9

    WRITE\_GOOD\_INFO 17[DI];销售数

    IO CRLF,9

    WRITE\_GOOD\_INFO 19[DI];推荐度

    IO CRLF,9

GOOD\_FOUND\_F:

    IO BUF12,9

    IO CRLF,9

    RET

GOOD\_NOT\_FOUND\_F:

    ;提示未找到信息

    IO BUF8,9

    IO CRLF,9

    RET

FIND\_GOOD ENDP

CHANGE\_GOOD\_INFO PROC

    PUSH BX

    PUSH BP

    PUSH DX

    PUSH AX

    PUSH SI

    PUSH CX

    CMP AUTH,1

    JNE CHANGE\_ERROR

    CMP GOOD,0

    JE  GOOD\_EMPTY

    MOV BX,GOOD

    ADD BX,10

    MOV BP,6;用于计数

L2:

    CMP BP,0

    JNE CHANGE\_INFO

    JMP FINISH\_CHANGE

NEWINT:IRET

CHANGE\_INFO:

    CMP BP,6

    JE L1;因为存储折扣的空间只有一个字节，所以要特殊处理

L3:

    LEA DX,CRLF

    MOV AH,9

    INT 21H

    MOV AX,WORD PTR [BX]

    MOV DX,16

    CALL F2T10;

    MOV DL,'>'

    MOV AH,2

    INT 21H

    IO NUMBER,10;输入要修改成的数字

    CMP BYTE PTR NUMBER+1,0

    JE NEXT\_INFO2;如果直接输入了回车键，则继续修改下一个商品信息

    LEA SI,NUMBER+2

    MOVZX CX,BYTE PTR NUMBER+1

    CALL F10T2

    CMP SI,-1

    JE L3

    MOV WORD PTR [BX],AX

    ADD BX,2

    DEC BP

    JMP L2

L1:

    LEA DX,CRLF

    MOV AH,9

    INT 21H

    MOVZX AX,BYTE PTR [BX]

    MOV DX,16

    CALL F2T10

    MOV DL,'>'

    MOV AH,2

    INT 21H

    IO NUMBER,10;输入要修改成的数字

    CMP BYTE PTR NUMBER+1,0

    JE NEXT\_INFO1;如果直接输入了回车键，则继续修改下一个商品信息

    LEA SI,NUMBER+2;SI为数字串的首地址

    MOVZX CX,BYTE PTR NUMBER+1;CX为数字串的数量

    CALL F10T2;调用函数，将数字串转化为二进制数字，结果存在AX中

    CMP SI,-1

    JE L1;如果输入的数字串不合法，则会要求重新输入

    MOV BYTE PTR [BX],AL;数字串合法，改变折扣的值

    ADD BX,1

    DEC BP

    JMP L2

NEXT\_INFO1:

    ADD BX,1

    DEC BP

    JMP L2

NEXT\_INFO2:

    ADD BX,2

    DEC BP

    JMP L2

CHANGE\_ERROR:

    IO BUF18,9;输出错误提示信息

    JMP RETURN

GOOD\_EMPTY:

    IO BUF\_GOOD\_EMPTY,9

    JMP RETURN

FINISH\_CHANGE:

    IO BUF19,9;提示完成商品信息的修改

    JMP RETURN

RETURN:

    POP CX

    POP SI

    POP AX

    POP DX

    POP BP

    POP BX

    IO CRLF,9

    RET

CHANGE\_GOOD\_INFO ENDP

TIMER   PROC

    PUSH  DX

    PUSH  CX

    PUSH  BX

    MOV   BX, AX

    MOV   AH, 2CH

    INT   21H        ;CH=hour(0-23),CL=minute(0-59),DH=second(0-59),DL=centisecond(0-100)

    MOV   AL, DH

    MOV   AH, 0

    IMUL  AX,AX,1000

    MOV   DH, 0

    IMUL  DX,DX,10

    ADD   AX, DX

    CMP   BX, 0

    JNZ   \_T1

    MOV   CS:\_TS, AX

\_T0:    POP   BX

    POP   CX

    POP   DX

    RET

\_T1:    SUB   AX, CS:\_TS

    JNC   \_T2

    ADD   AX, 60000

\_T2:    MOV   CX, 0

    MOV   BX, 10

\_T3:    MOV   DX, 0

    DIV   BX

    PUSH  DX

    INC   CX

    CMP   AX, 0

    JNZ   \_T3

    MOV   BX, 0

\_T4:    POP   AX

    ADD   AL, '0'

    MOV   CS:\_TMSG[BX], AL

    INC   BX

    LOOP  \_T4

    PUSH  DS

    MOV   CS:\_TMSG[BX+0], 0AH

    MOV   CS:\_TMSG[BX+1], 0DH

    MOV   CS:\_TMSG[BX+2], '$'

    LEA   DX, \_TS+2

    PUSH  CS

    POP   DS

    MOV   AH, 9

    INT   21H

    POP   DS

    JMP   \_T0

\_TS DW    ?

    DB    'Time elapsed in ms is '

\_TMSG   DB    12 DUP(0)

TIMER   ENDP

CODE ENDS

    END START

# 参考文献

[1] 作者.书名.版本(第×版).译者.出版地:出版者,出版年：起页-止页

[序号] URL: 网络地址，如URL:http://www.cbs.dtu.dk/services