

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实践**

**专业班级：计算机科学与技术201806班**

**学 号： U201814655**

**姓 名： 杨雨鑫**

**指导教师： 张勇**

**实验时段： 2020年3月19日~5月7日**

**实验地点： 武汉大学测绘社区**

**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 - 4 -](#_Toc41156710)

[课程目标 - 4 -](#_Toc41156711)

[成绩构成 - 4 -](#_Toc41156712)

[实验任务的总体描述 - 4 -](#_Toc41156713)

[1编程基础 1](#_Toc41156714)

[1.1 实验目的与要求 1](#_Toc41156715)

[1.2 实验内容 1](#_Toc41156716)

[1.3 任务1.1实验过程 3](#_Toc41156717)

[1.3.1 实验方法说明 3](#_Toc41156718)

[1.3.2 实验记录与分析 3](#_Toc41156719)

[1.4 任务1.2的实验过程 4](#_Toc41156720)

[1.4.1实验方法说明 4](#_Toc41156721)

[1.4.2实验记录分析 5](#_Toc41156722)

[1.5 任务1.3的实验过程 6](#_Toc41156723)

[1.5.2实验记录分析 7](#_Toc41156724)

[1.6 任务1.4的实验过程 9](#_Toc41156725)

[1.6.1设计思想及单元分配 9](#_Toc41156726)

[1.6.2源程序 13](#_Toc41156727)

[1.6.3实验步骤 16](#_Toc41156728)

[1.6.4实验记录分析 16](#_Toc41156729)

[1.7 小结 19](#_Toc41156730)

[1.7.1 主要收获 19](#_Toc41156731)

[1.7.2 主要看法 20](#_Toc41156732)

[2程序优化 20](#_Toc41156733)

[2.1 实验目的与要求 20](#_Toc41156734)

[2.2 实验内容 20](#_Toc41156735)

[2.3任务2.1实验过程 20](#_Toc41156736)

[2.3.1 实验方法说明 20](#_Toc41156737)

[2.3.2 实验记录与分析 21](#_Toc41156738)

[2.4 任务2.2实验过程 27](#_Toc41156739)

[2.4.1实验方法说明 27](#_Toc41156740)

[2.4.2 实验记录与分析 29](#_Toc41156741)

[2.5小结 30](#_Toc41156742)

[2.5.1 主要收获 30](#_Toc41156743)

[2.5.2 主要看法 31](#_Toc41156744)

[3模块化程序设计 31](#_Toc41156745)

[3.1实验目的与要求 31](#_Toc41156746)

[3.2实验内容 31](#_Toc41156747)

[3.3 任务3.1实验过程 32](#_Toc41156748)

[3.3.1 设计思想及单元分配 32](#_Toc41156749)

[3.3.2 源程序 32](#_Toc41156750)

[3.3.3 实验步骤 34](#_Toc41156751)

[3.3.4 实验记录和分析 34](#_Toc41156752)

[3.4 任务3.2实验过程 38](#_Toc41156753)

[3.4.1 设计思想及单元分配 38](#_Toc41156754)

[3.4.2 源程序 39](#_Toc41156755)

[3.4.3 实验步骤 40](#_Toc41156756)

[3.4.4 实验记录和分析 40](#_Toc41156757)

[3.5 小结 46](#_Toc41156758)

[3.5.1 主要收获 46](#_Toc41156759)

[3.5.2 主要看法 47](#_Toc41156760)

[4中断与反跟踪 47](#_Toc41156761)

[4.1实验目的与要求 47](#_Toc41156762)

[4.2实验内容 47](#_Toc41156763)

[4.3任务4.1实验过程 48](#_Toc41156764)

[4.3.1设计思想及单元分配 48](#_Toc41156765)

[4.3.2 实验步骤 50](#_Toc41156766)

[4.3.3实验记录与分析 50](#_Toc41156767)

[4.4任务4.2和4.3实验过程 54](#_Toc41156768)

[4.4.1设计思想及单元分配 54](#_Toc41156769)

[4.4.2 实验步骤 56](#_Toc41156770)

[4.4.3实验记录与分析 57](#_Toc41156771)

[4.5 小结 65](#_Toc41156772)

[4.5.1 主要收获 65](#_Toc41156773)

[4.5.2 主要看法 66](#_Toc41156774)

[5 WIN32程序设计 67](#_Toc41156775)

[5.1实验目的与要求 67](#_Toc41156776)

[5.2实验内容 67](#_Toc41156777)

[5.3 任务5.1实验过程 68](#_Toc41156778)

[5.3.1实验方法说明 68](#_Toc41156779)

[5.3.2实验记录与分析 68](#_Toc41156780)

[5.4 小结 73](#_Toc41156781)

[5.4.1 主要收获 73](#_Toc41156782)

[5.4.2 主要看法 74](#_Toc41156783)

[参考文献 75](#_Toc41156784)

[附录1 76](#_Toc41156785)

# 课程总体说明

## 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。 |

## 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。

## 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。

# 1编程基础

## 1.1 实验目的与要求

本次实验的主要目的与要求有以下几点，所有的任务都会围绕这几点进行，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；
7. 掌握设计实现一个原型系统的基本方法。

## 1.2 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1.4 设计实现一个网店商品信息管理系统。

该系统的基本需求见“2020实验报告样例”文档中的“实验任务的总体描述”。

根据系统的基本需求，可以制定如下的数据段的定义（供参考）：

BNAME DB ‘ZHANG SAN’,0 ；老板姓名（本实验要求必须是自己名字的拼音）

BPASS DB ‘test’，0，0 ，0 ；密码

AUTH DB 0 ；当前登录状态，0表示顾客状态

GOOD DB/DW … ；当前浏览商品名称或地址（自行确定）

N EQU 30

SNAME DB ‘SHOP’,0 ；网店名称，用0结束

GA1 DB ‘PEN’, 7 DUP(0) ，10 ；商品名称及折扣

DW 35，56，70，25，？ ； 推荐度还未计算

GA2 DB ‘BOOK’, 6 DUP(0) ，9 ； 商品名称及折扣

DW 12，30，25，5，？ ；推荐度还未计算

GAN DB N-2 DUP( ‘TempValue’ ,0,8，15，0，20，0，30，0，2，0，？，？) ;除了2个已经具体定义了的商品信息以外，其他商品信息暂时假定为一样的。

本次实验主要是利用分支、循环程序的结构，实现该系统的基本功能，并能熟悉全周期、全流程地设计实现一个原型系统的基本方法。本次实验要具体实现的功能要求如下：

0.主菜单界面

完整显示“实验任务的总体描述”中给出的界面信息。等待用户输入数字（可使用1号DOS系统功能调用）。对用户输入的字符进行判断，看是否是1~9的数字；是的话就转移到对应功能的程序标号，不是的话就提示错误，回到主菜单界面。

1.登录/重新登录

（1）先后分别提示用户输入姓名和密码（可使用9号DOS系统功能调用）。

（2）分别获取输入的姓名和密码（可使用10号DOS系统功能调用）。输入的姓名字符串放在以in\_name为首址的存储区中，密码放在以in\_pwd为首址的存储区中。

（3）若输入姓名时只是输入了回车，则将0送到AUTH字节变量中，回到主菜单界面。

（4）进行身份认证：

（a）使用循环程序结构，比较姓名是否正确。若不正确，则跳到（c）。

（b）若正确，再比较密码是否相同，若相同，跳到（d）。

（c）若名字或密码不对，则提示登录失败，并转到“（3）”的位置。

（d）若名字和密码均正确，则将1送到AUTH变量中，回到主菜单界面。

2.查找指定商品并显示其信息

（1）提示用户输入商品名称。

（2）在商店中寻找是否存在该商品。

（3）若存在，则将商品名称或地址记录到GOOD字段中。商品信息的显示暂时不做。返回到主菜单界面。

（4）若没有找到，提示没有找到，返回到主菜单界面。

3.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（目前不是用子程序实现的，所以，跳转之前，要把返回地址送到指定变量中），返回主菜单界面。

（2）若无效或剩余数量为0，则提示错误，回到主菜单界面。

4.计算商品推荐度

按照给出的公式计算所有商品的推荐度，返回到指定的位置（JMP 含返回地址的指定变量）。

要求尽量避免溢出。结果只保留整数部分。

5.排名

暂不实现，直接返回主菜单界面。

6.修改商品信息

暂不实现，直接返回主菜单界面。

7.迁移商店运行环境

暂不实现，直接返回主菜单界面。

8.显示当前代码段首址

将当前代码段寄存器CS里面的内容按照16进制的方式显示到屏幕上，返回主菜单界面。

9.退出

退出本系统（可使用4CH号DOS系统功能调用）

## 1.3 任务1.1实验过程

### 1.3.1 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,0110011B；MOV AL,1011010B；ADD AH,AL(SUB 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。三次预计的实验结果见表1.1:

表1.1预期试验结果表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | AH | AL | 加法预期结果 | 减法预期结果 |
| 数据1 | +0110011B | +1011010B | AH:8DH  SF:1  OF:1  CF:0  ZF:0 | AH:D9  SF:1  OF:0  CF:1  ZF:0 |
| 数据2 | -0101001B | -1011101B | AH:7AH  SF:0  OF:1  CF:1  ZF:0 | AH:34H  SF:0  OF:0  CF:0  ZF:0 |
| 数据3 | +1100101B | -1011101B | AH:08H  SF:0  OF:0  CF:1  ZF:0 | AH:C2H  SF:1  OF:1  CF:1  ZF:0 |

### 1.3.2 实验记录与分析

三次实验结果截图见表1.2:

表1.2实验结果表

|  |  |  |
| --- | --- | --- |
|  | 相加 | 相减 |
| 数据1 |  |  |
| 数据2 |  |  |
| 数据3 |  |  |

与上机前的预测结果进行比较后发现和预测相符合，第一组数据相加发生了溢出，所以OF=1，第八位为1，所以SF=1，结果不为0，所以ZF=0,没有发生进位，所以CF=0。第二组数据相加发生了溢出，所以OF=1,第八位为0，所以SF=0，结果不为0，所以ZF=0，同时发生了进位，所以CF=1。第三组数据相加没有发生溢出，所以OF=0,第八位为0，所以SF=0，结果不为0，所以ZF=0，同时发生了进位，所以CF=1。

## 1.4 任务1.2的实验过程

### 1.4.1实验方法说明

1.将需要运行的代码存储在demo.asm文件中，打开td开始分步调试程序，通过f8单步调试，在需要停止的语句前观察需要观察的寄存器的存储的数值，预期如下：

(1)运行到MOV CX,10前，(BX),(BP),(SI),(DI)各是0014H,001EH,0000H,000AH

运行INT 21H前，(BX),(BP),(SI),(DI)各是001EH,0028H,0000H,000AH

(2)设想的开始的40个字节的内容为：

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

BUF2 DB 0, 0，0，0，0，0，0，0，0，0

BUF3 DB 0，0，0，0，0，0，0，0，0，0

BUF4 DB 0，0，0，0，0，0，0，0，0，0

程序结束时的40个字节的内容为：

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

BUF2 DB 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

BUF3 DB 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

BUF4 DB 4, 5, 6, 7, 8, 9, 0AH, 0BH, 0CH, 0DH

### 1.4.2实验记录分析

运行到MOV CX,10前，(BX),(BP),(SI),(DI)各是0014H,001EH,0000H,000AH,截图见图1.1:

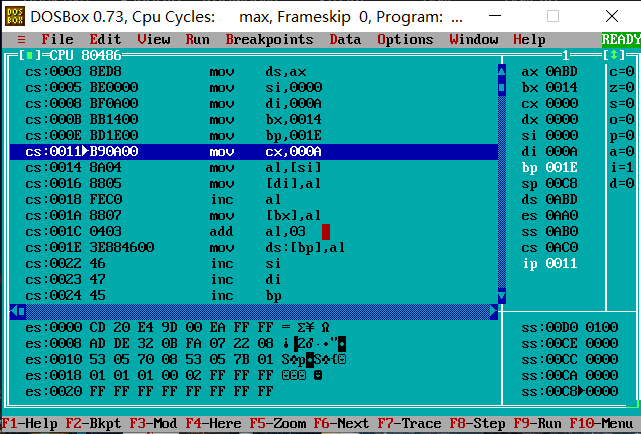


图1.1 运行之前的图

运行到INT 21前，(BX),(BP),(SI),(DI)各是001EH,0028H,0000H,000AH，截图见图1.2:

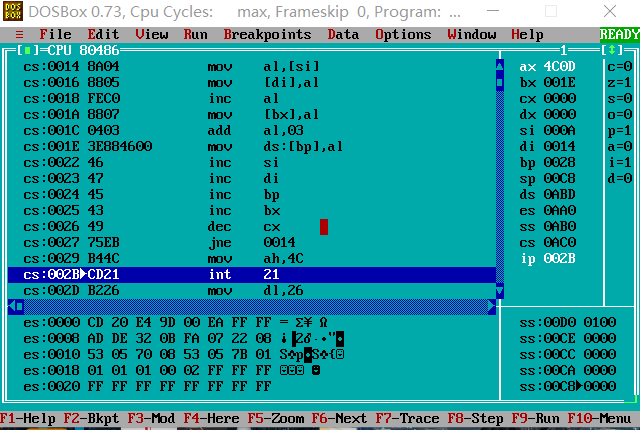


图1.2 运行之后的截图

在程序结束时ds数据段中存储的数据截图见图1.3，1.4:

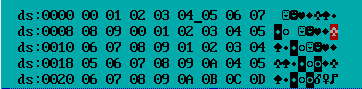


图1.3 数据段显示截图（1）

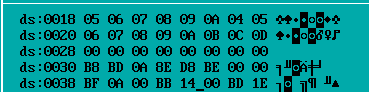


图1.4 数据段显示截图（2）

通过对ds数据段的观察发现与我之前预想的操作结果一致。

问题：如何设置断点？

答案：使用f2来设置或取消断点，f9运行程序。

问题：如何让程序运行到光标的所在语句？

答案：使用f4运行至光标所在语句。

对于断点以及调试的操作我专门去网上查阅使用方法，会在收获中提到。

## 1.5 任务1.3的实验过程

1.5.1 实验方法说明

程序目的是将BUF1的内容复制到BUF2，BUF3中的数据比BUF1中的依次大

1,BUF4中对应数据比BUF1中大3，所以它们的数据理论上如下：

BUF1：00，01，02，03，04，05，06，07，08，09，

BUF2：00，01，02，03，04，05，06，07，08，09，

BUF3：01，02，03，04，05，06，07，08，09，0A，

BUF4：04，05，06，07，08，09，0A，0B，0C，0D。

实验中使用了32位的寄存器进行变址寻址，因此寄存器都得使用32位寄存器表示ESI，因为BUF1- BUF4依次相差10个字节，因此可以根据BUF1的偏移地址计算出其他几个，不用调用多个寄存器，可节约使用的寄存器数量。源程序代码如下：

.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 ESI, OFFSET BUF1 ;只取BUF1的偏移地址

MOV CX, 10

LOPA: MOV AL, [ESI]

MOV [ESI + 0AH], AL ;通过变址寻址的方式定位内存单元BUF2

INC AL

MOV [ESI + 14H], AL ;通过变址寻址的方式定位内存单元BUF3

ADD AL, 3

MOV [ESI + 1EH], AL ;通过变址寻址的方式定位内存单元BUF4

INC ESI

DEC CX

JNZ LOPA

MOV AH, 4CH

INT 21H

CODE ENDS

END START

### 1.5.2实验记录分析

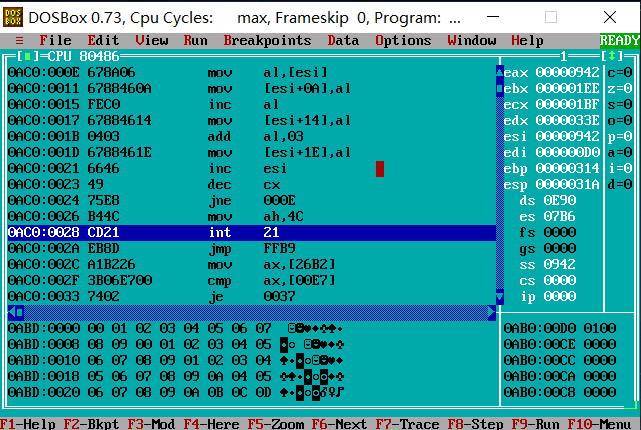


图1.5 前40个字节信息截图

通过观察ds数据段中存储的元素发现40个字节的内容分别代表操作之后的四个数组的元素，与我的预测相同。

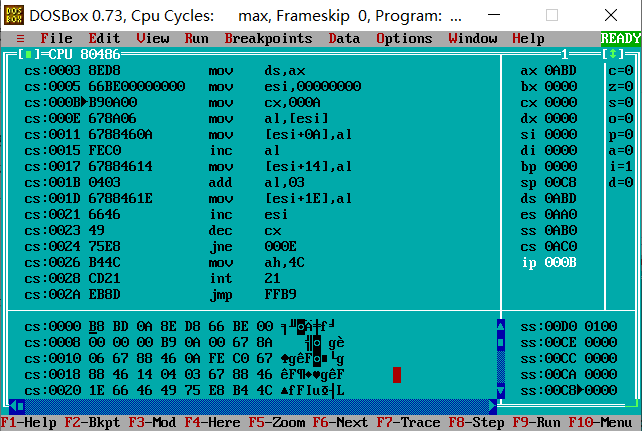


图1.6 机器指令观察截图

分析：如果我们仅仅是观察这句指令的机器码B90A00并不能理解这个指令正在干什么，但是通过后面现实的反汇编指令mov cx,000A我们就可以知道是把000A赋值给cx寄存器。接下来我们发现原来的代码段地址只有0011和0015，于是我们在代码段选择goto并输入cs:0013对该语句进行反汇编：

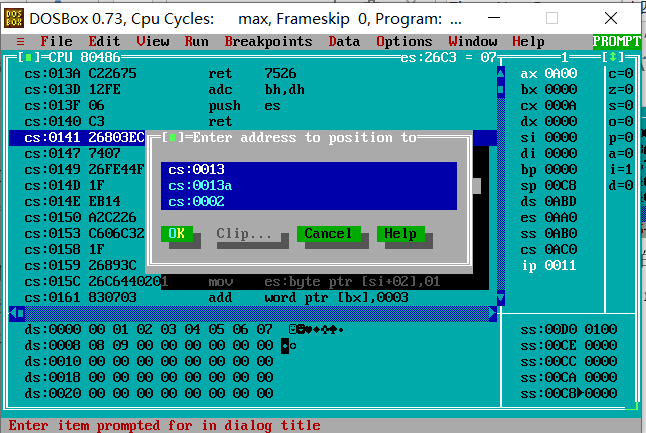


图1.7 反汇编结果截图

得到的操作结果如下：

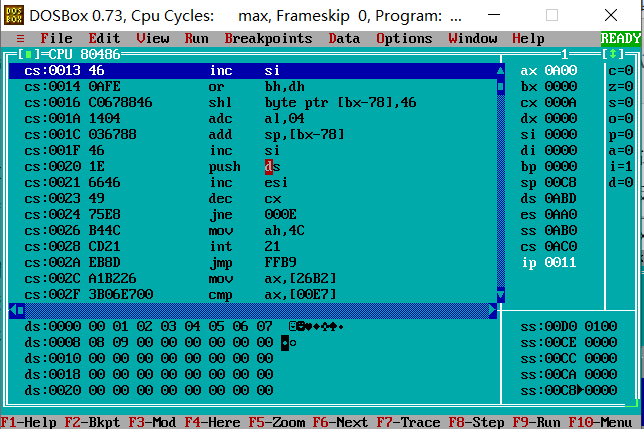


图1.8 操作结果截图

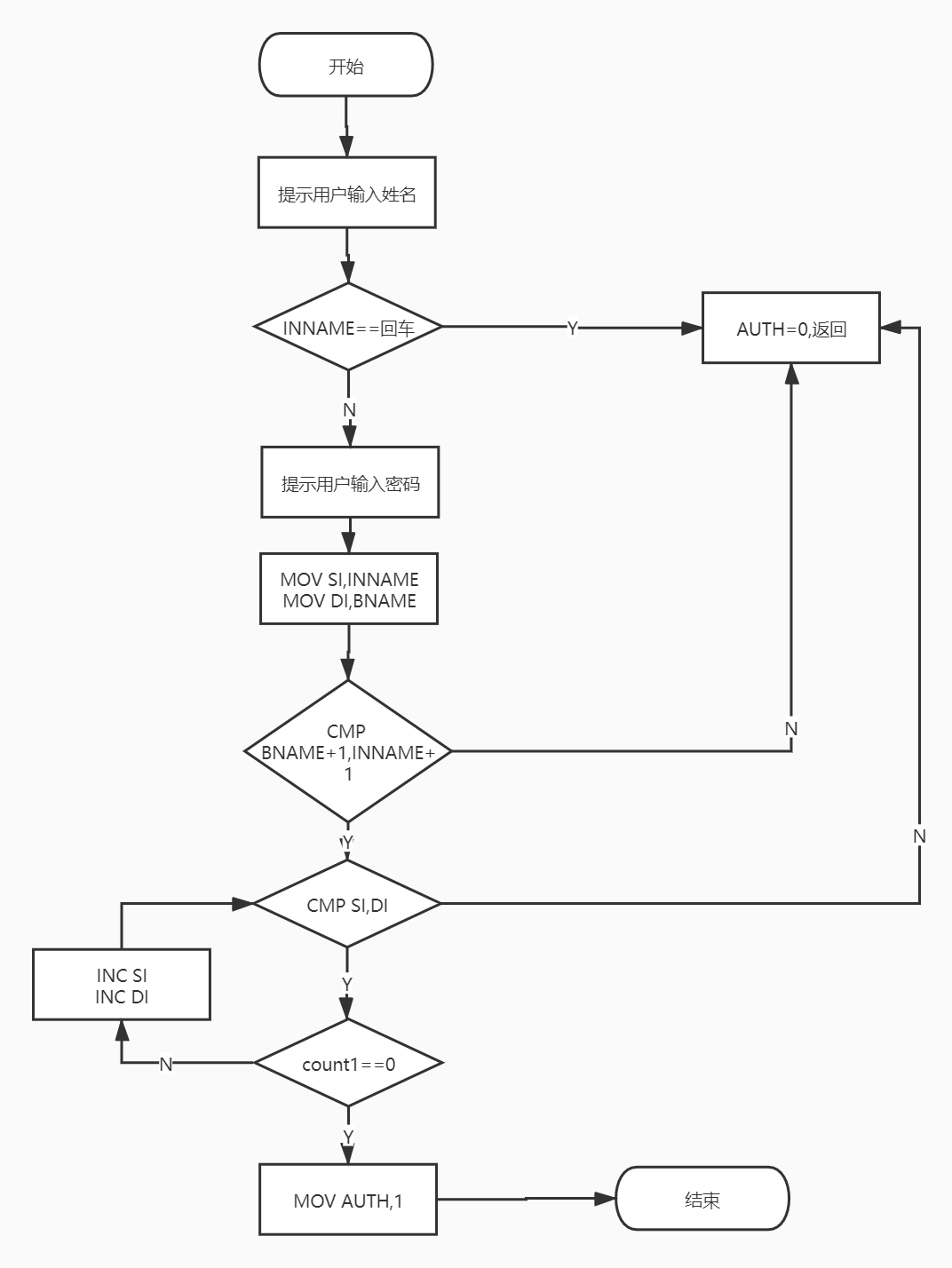
此处我们可以发现机器码段只显示了46，我们通过观察cs:0011的机器码为:6788460A，于是我们发现我们访问的地址为0013的代码段正好在0011~0015之间，这里得到的反汇编语句却变成了si寄存器自增，我们可以得出结论，如果我们随意从某一条语句的中间位置开始汇编会导致程序表达出完全不同的意思，因此ip寄存器的作用就是告诉计算机下一条语句的起始地址，防止产生像上面这种错误的编译方式。

## 1.6 任务1.4的实验过程

### 1.6.1设计思想及单元分配

本任务一共有四个功能，故我决定使用四个模块来分别实现这四个功能，数据段的定义采用参考的数段定义。

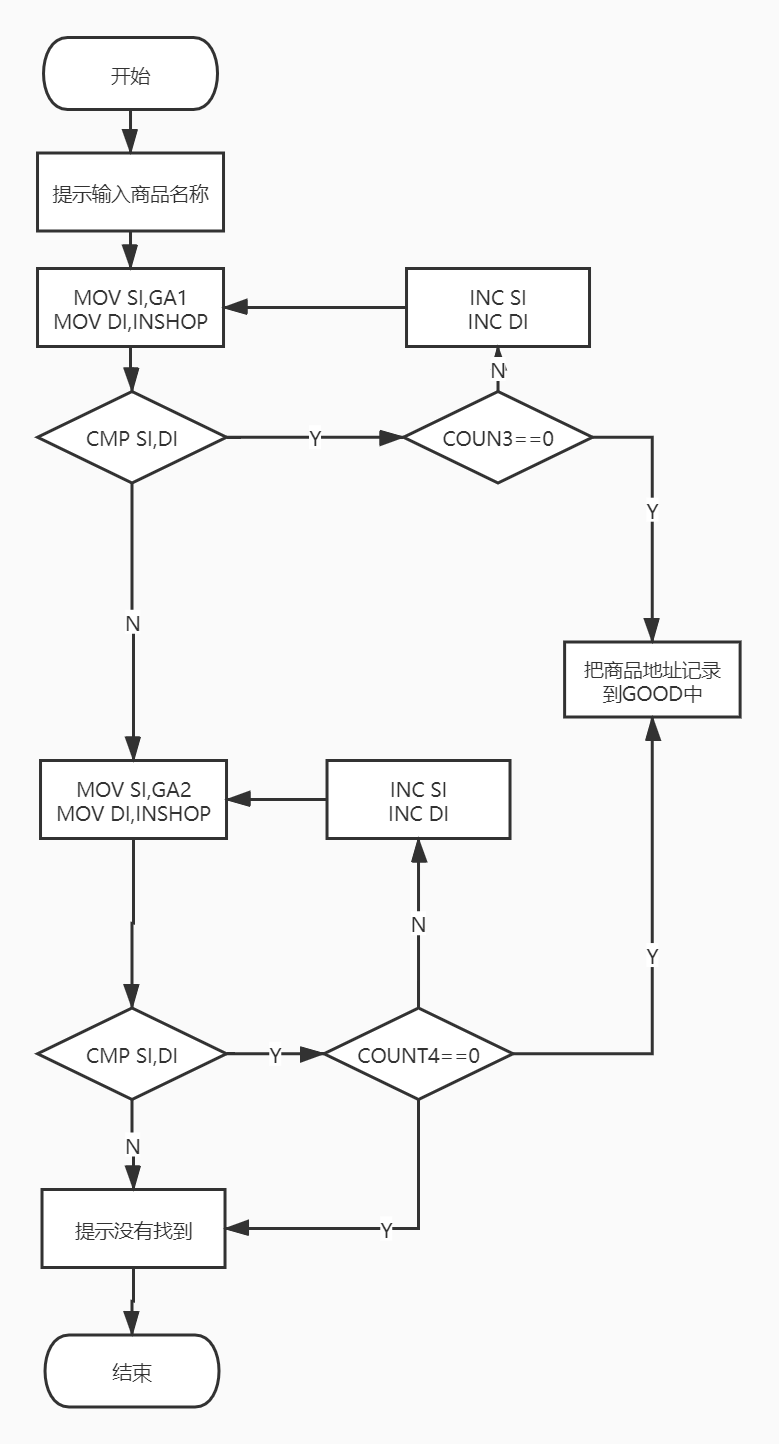
（1）功能一是实现登录和进行身份认证的，流程图如下：



1.9 功能一流程图

首先会调用出提示，提醒用户输入姓名，使用MOV AH,9和INT 21来调用9号dos系统功能。把用户输入的姓名传入INNAME字符串中，如果只是输入了回车，把AUTH置为0并返回主菜单。接下来调用出提示，提醒用户输入密码。将SI,DI寄存器分别指向INNAME和BNAME的首地址，使用CMP比较SI,DI指向地址里的字符，其中COUNT1为INNAME的长度，首先比较两个字符串长度是否相同，再进入循环比较每次CMP比较之后，如果两个字符相同，则把COUNT1自减1，如果此时COUNT1不为0，则把SI,DI分别往后移一位，跳转到SI,DI的比较指令。如果CMP比较发现两个字符不同，则把AUTH置为0，返回菜单。当COUNT1为0时，如果不是，把AUTH置为0，返回主菜单。

（2）功能二是为了查找指定商品并显示信息，流程图如下：



1.10 功能二流程图

调用提示，提示用户输入商品名称，先将第一个产品名字与输入的产品名字进行比对，如果比较成功则把第一个商品的名称地址存储存储到GOOD中，如果中间比较失败，则开始比较第二个商品名称和输入名称，如果比较成功，则把第二个商品的名称的地址存储到GOOD中，如果比较失败，则输出提示没有找到，返回主菜单。

（3）功能三是为了实现下定单的功能，即把售出数量加一，流程图如下：

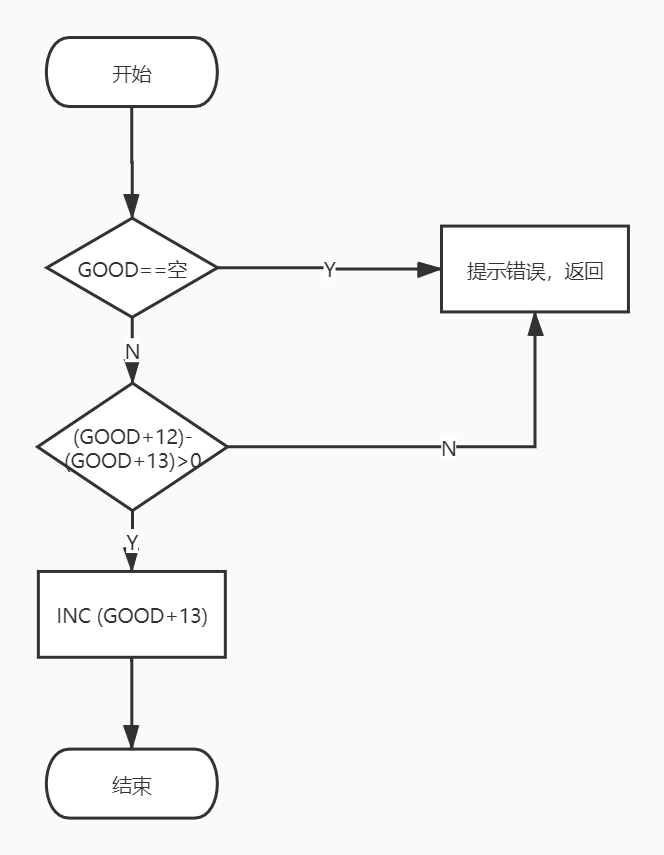


图1.11 功能三流程图

首先判断GOOD中存储的是否为空，如果不为空，则访问该商品的第12和第13个字，把两数相减所得的值与0比较，如果为0，则提示错误并返回，如果不为0，则把第13个字即已售商品数量加一并且跳转到功能4重新对商品推荐度进行计算，最后返回主菜单。

（4）功能4是为了计算商品推荐度，流程图如下：

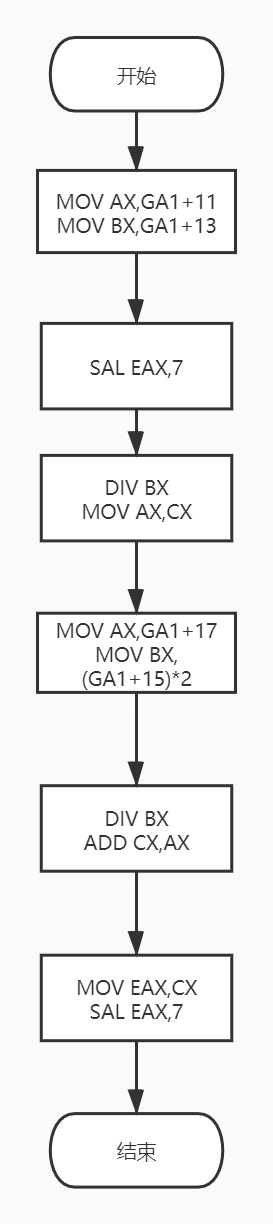


图1.12 功能四流程图

首先将该商品的进货价和实际销售价格进行相除操作，在把已售数量和两倍的进货数量相除，最后把两个结果相加，对于\*128，则直接将相加结果左移7位，把所得到的推荐度存储到该商品的推荐度所在内存单元中，返回。

### 1.6.2源程序

本次一共设计了四个函数分别实现四个功能：

**功能一函数关键部分（进行比较循环部分）：**

MOV CL,COUNT1 ;验证姓名

LEA SI,BNAME

LEA DI,INNAME+2

CMP CL,INNAME+1 ;长度不相等，姓名肯定错误

JNZ WRONG1

FLAG1:

MOV BL,[DI]

CMP BYTE PTR [SI],BL

JNZ WRONG1

INC SI

INC DI

DEC CX

CMP CX,0 ;比较计数器是否为0

JNZ FLAG1

MOV CL,COUNT2 ;验证密码

LEA SI,BPASS

LEA DI,INPASSWORD+2

CMP CL,INPASSWORD+1 ;长度不相同，密码肯定错误

JNZ WRONG2

FLAG2:

MOV BL,[DI]

CMP BYTE PTR [SI],BL

JNZ WRONG2

INC SI

INC DI

DEC CX

CMP CX,0

JNZ FLAG2

JMP LOGIN\_SUCCESSFUL ;成功登入

**功能二函数关键部分（进行字符串比对寻找商品）：**

LEA SI,GA1

LEA DI,INCOMMODITY+2

MOV CL,COUNT3

CMP CL,INCOMMODITY+1 ;判断字符数是否相同

JNZ NEXTGOOD

FLAG3:

MOV BL,[DI] ;移动到第一个字符

MOV BH,[SI]

CMP BL,BH

JNZ NEXTGOOD ;如果两个字符不同，则搜索下一个商品

INC SI

INC DI

DEC CX

CMP CX,0

JNZ FLAG3 ;如果CX不为0，则继续循环

LEA DX,TIP8

MOV AH,9

INT 21H

MOV GOOD,1 ;1号商品即为所求

**功能三函数关键部分（以购买商品1为例）：**

CMP GOOD,1

JZ GOOD1

CMP GOOD,2

JZ GOOD2

RET

GOOD1:

MOV AX,WORD PTR GA1+17 ;销售数量

MOV BX,WORD PTR GA1+15 ;进货数量

SUB BX,AX

CMP BX,0

JNZ PURCHASE1

RET

PURCHASE1:

INC WORD PTR GA1+17

LEA DX,TIP9

MOV AH,9

INT 21H

RET

**功能四函数关键部分（以计算商品1为例）：**

CMP GOOD,1 ;是否为第一个产品

JNZ NEXT

LEA SI,GA1

XOR BX,BX

MOV AX,WORD PTR GA1+17 ;销售数量

MOV BX,WORD PTR GA1+15 ;进货数量

SAL BX,1

SAL EAX,7 ;已售数量\*128

CWD ;字转双字

IDIV BX

MOV CX,AX ;存储结果

MOV AX,WORD PTR GA1+11 ;进货价

MOV BX,WORD PTR GA1+21 ;销售价

SAL EAX,7 ;进货价\*128

CWD

IDIV BX

ADD AX,CX ;把两个结果相加

MOV WORD PTR [GA1+19],AX

RET

**功能八函数关键部分：**

MOV AX, CS

AND AX, 0F000H

SHR AX, 12

XLAT TAB

MOV DL, AL

MOV AH, 2 ;输出CS的最高位

INT 21H

MOV AX, CS

AND AX, 0F00H

SHR AX, 8

XLAT TAB

MOV DL, AL

MOV AH, 2 ;输出CS的次高位

INT 21H

MOV AX, CS

AND AX, 00F0H

SHR AX, 4

XLAT TAB

MOV DL, AL

MOV AH, 2 ;输出CS的次低位

INT 21H

MOV AX, CS

AND AX, 000FH

XLAT TAB

MOV DL, AL

MOV AH, 2 ;输出CS的最低位

INT 21H

### 1.6.3实验步骤

准备上机环境，在vscode中编写程序。

在编写程序的过程中自己也遇到了很多问题，并且成功的解决了问题。

首先对于程序的读取和输出，我上网查阅了有关资料明白了dos9号指令可以用来输出字符串，dos10号指令可以从键盘读取字符串。具体使用方法如下：

LEA DX,(需要输出或是读取的字符串地址)

MOV AH,9 (调用的指令)

INT 21H

针对在定义字符串的时候要在结尾是否加上$符号，我决定通过观察输出在菜单栏的字符进行判断，可以分别比较加了$符号和没有加$符号的输出情况。

对于换行符，空格符之类的符号，我事先得先了解一些符号的asc码方便进行编写。

针对任务中建议的匹配方法，我通过了解dos10号命令发现可以得到已经输入的字符串长度，于是我觉得可以在此基础上对于匹配循环进行改进，最先匹配字符数目，如果字符数目不同就直接返回，不必要进入循环，最后也没有必要去访问BNAME的下一个是否为0，可以加快匹配速度。

对于功能四我觉得可以先把128乘进括号里面去，不然在四舍五入的情况下25/58会出现0的情况，但如果先把128乘进去，就不会出现特别大的偏差。

对于乘法操作，我发现乘的数全是2的整数次方，于是我决定用移位操作代替乘法操作，效率更快。

### 1.6.4实验记录分析

实验环境条件：WINDOWS 10下的dosbox0.73，LINK.EXE,MASM.EXE

汇编程序过程中出现了这些报错

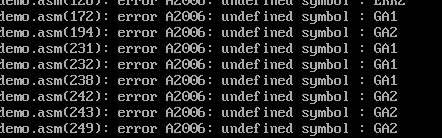


图1.13 报错一截图

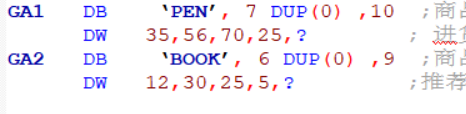


图1.14 报错二截图

最后我发现原来是在自己编写程序的时候没有注意中英文切换，导致各种未定义警告出现，修改后报错消失。

连接过程中没有出现异常

先输出商店名称，再显示菜单：

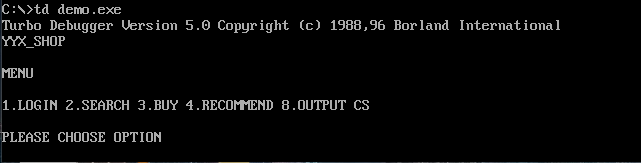


图1.15 菜单显示

选择登录，输入1：



图1.16 功能一

这里我们输入正确的姓名和密码，系统验证后返回菜单。

我们选择第二个功能，输入2,接着输入我们需要查找的产品名：



图1.17 功能二

这里的提示代表查找成功。

我们选择第三个功能，输入三，进行购买操作：

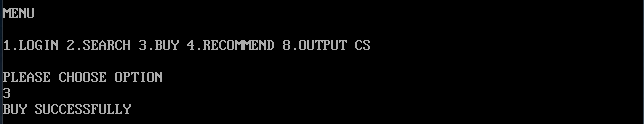


图1.18 功能三

这里代表购买成功。

我们选择第四个功能，对GOOD所存储的商品进行推荐度计算：

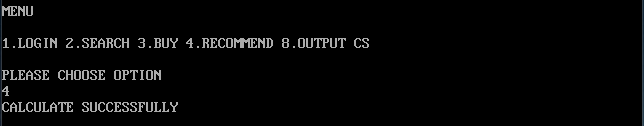


图1.19 功能四

这里提示计算成功。

我们选择第八个功能，输出CS首址：

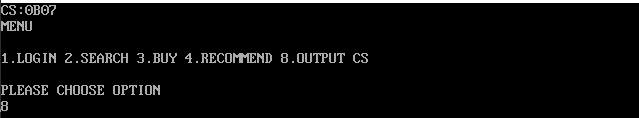




图1.20 功能8

对于上述四个功能的纠错检查测试：

（1）功能1：

输入的姓名错误：



图1.21 输入错误姓名

输入的密码有误：

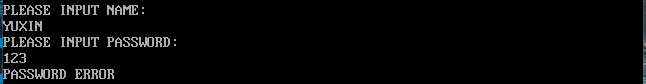


图1.22 输入错误密码

（2）功能2：

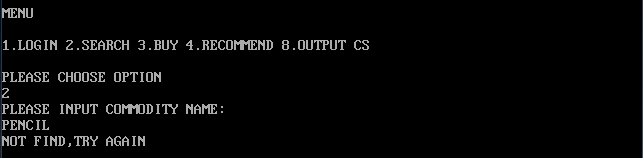


图1.23 输入错误商品名

输入的商品名是pencil，不在商品表中。

（3）功能3：

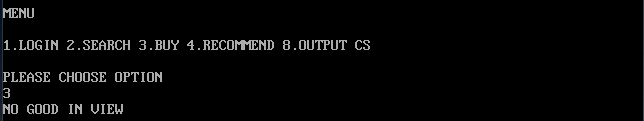


图1.24 未进行搜索操作

没有进行搜索操作，此时GOOD中为0。

## 1.7 小结

### 1.7.1 主要收获

对于任务一，主要的收获是懂得如何把文件挂载在dosbox中，并且学会了如何生成obj，exe类型的文件，并且能够在td编译器中打开。知道了可以直接在代码段输入指令并且对指令进行编译。直接输入的步骤如下:

首先使用方向键把光标移到期望的地址处，打开指令编辑窗口。有两种方法:一是直接输入汇编指令，在输入汇编指令的同时屏幕上就会自动弹出指令的临时编辑窗口；二是激活代码区局部菜单，选择其中的汇编命令，屏幕上也会自动弹出指令的临时编辑窗口。在临时编辑窗口中输入/编辑指令，每输入完一条指令，按回车，输入的指令即可出现在光标处，同时光标自动下移一行，以便输入下一条指令。

我在程序验证过程中能够清楚地知道td中的每个区域代表的含义，能快速找到寄存器的位置。利用goto指令可以直接快速从我们想要执行的语句处开始执行，了解了标志位的含义与进位之间的关系SF代表最高位的数字，OF代表是否发生溢出，ZF代表结果是否为0，CF代表是否向高位进位。

对于任务二和任务三，我能够了解到关于td的使用技巧，并且总结出了许多经验，f7,f8可以进行单步调试，f4可以让程序运行到光标所在的语句处，f9直接执行程序，f2可以用来设置或者取消断点，可以把光标切换到存储段，再通过goto指令访问不同的寄存器来观察数据在内存中的存储状态，方便我们对照上方代码段的单步执行来一步步观察执行结果，也是通过这两个任务了解到了循环结构在汇编语言中的编写方式，同时巩固了自己对于寄存器以及偏移地址章节的知识，增强了自己的动手操作能力。

对于任务四，我这次花费了很长的时间在查找指令名称以及指令的使用方法。但是真正在编写程序的过程中，我是直接在纸上画出了内存存储方式，尽量避免了变址寻址方式，而是直接访问对应的内存单元，不容易出错，当然这个任务的重点，我觉得还是跳转指令的应用。此外，我通过网络搜索，成功的学会了使用函数定义的方式把程序实现模块化调用，感觉和C语言十分类似，因此在熟悉基本操作后我上手很快，快速的编写好了程序，并且调试过程中基本没有遇见太多不能解决的bug，感觉对于自己的编程能力有了很大的提升。

### 1.7.2 主要看法

任务一二三感悟：这次使用的dosbox是基于命令行的，和cmd操作方式很像。对于编译工具td感觉自己上手比较快，在了解一些经常使用的操作技巧后，学会使用键盘进行操作了，总体感觉就是非常直观的反映了计算机的执行步骤，当然界面的美化还需要做出提高，相比于其他的诸如vs之类的平台还是显得过于简陋，但适合帮助我这种初学者理解程序。

任务四感悟：这次使用dosbox调试的时候我就可以很轻松的查看相应的数据是否与我所期待的一致，希望能够在单步执行的同时可以看到输出的效果图，方便我判断程序执行到了哪里。

# 2程序优化

## 2.1 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。

## 2.2 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。

**任务2.2 对任务2.1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2.1的程序时，尽量不要考虑代码优化的问题）。

## 2.3任务2.1实验过程

### 2.3.1 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。

2. 打开老师在qq中发送的time函数文件，了解计时函数的使用方法。

3. 在自己的asm文件中，把两个商品的进货量修改成100。

4. 在原来的功能三函数中加入循环，使得功能三操作变为购买1000次。

5. 把time函数嵌入功能三中，完成计时功能，记录下程序的运行时间显示。具体代码如下：

P3:

MOV AX, 0

CALL TIMER ;开始计时

MOV CX,30000 ;初始化计数器

XUNHUAN1:

CMP GOOD,0 ;观察GOOD是否为0

JNZ NOTZERO1

JZ XUNHUAN1

NOTZERO1:

CALL FUN3

CALL FUN4

DEC CX

CMP CX,0

JNZ XUNHUAN1

P3LAST:

MOV AX,1

CALL TIMER ;输出计时结果

JMP INSTRUCT

6.在执行过程中调节虚拟机的频率并观察运行时间的变化。

### 2.3.2 实验记录与分析

这次试验，我使用了两台设备，具体配置如下：

标准设备：

系统：windows7

处理器：Intel Core i7-4700MQ CPU @ 2.40 GHz八核

内存：8G

对比设备：

系统：windows10

处理器：Intel Core i7-7Y75 CPU @ 1.30 GHz 四核

内存：16G

（1）为了让时间误差比较小，我将虚拟机频率降到了25%，重复了三次，得到了如下的三组数据：

表2.1 25%频率下的运行时间

|  |  |  |
| --- | --- | --- |
| 序号 | 结果 | 用时/ms |
| 第一次 |  | 500 |
| 第二次 |  | 440 |
| 第三次 |  | 490 |
| 在另一台设备的运行结果 |  | 110 |

（2）我提升了虚拟机的频率到40%，再次进行测试，所得三组数据如下：

表2.2 40%频率下的运行时间

|  |  |  |
| --- | --- | --- |
| 序号 | 结果 | 用时/ms |
| 第一次 |  | 110 |
| 第二次 |  | 110 |
| 第三次 |  | 110 |
| 在另外一台设备的运行结果 |  | 50 |

通过在不同cpu频率下的工作时间，我可以很直观的看出cpu频率对于运行时间的影响，这里通过在不同设备上的运行时间很好地反映出了硬件对于运行速度的影响，cpu代数高的4核还是碾压低代八核。

问题1：改变循环程序的设计和循环次数，对CPU资源消耗的影响有多大？

我在任务管理器中观察cpu的使用量：

循环10000次时：

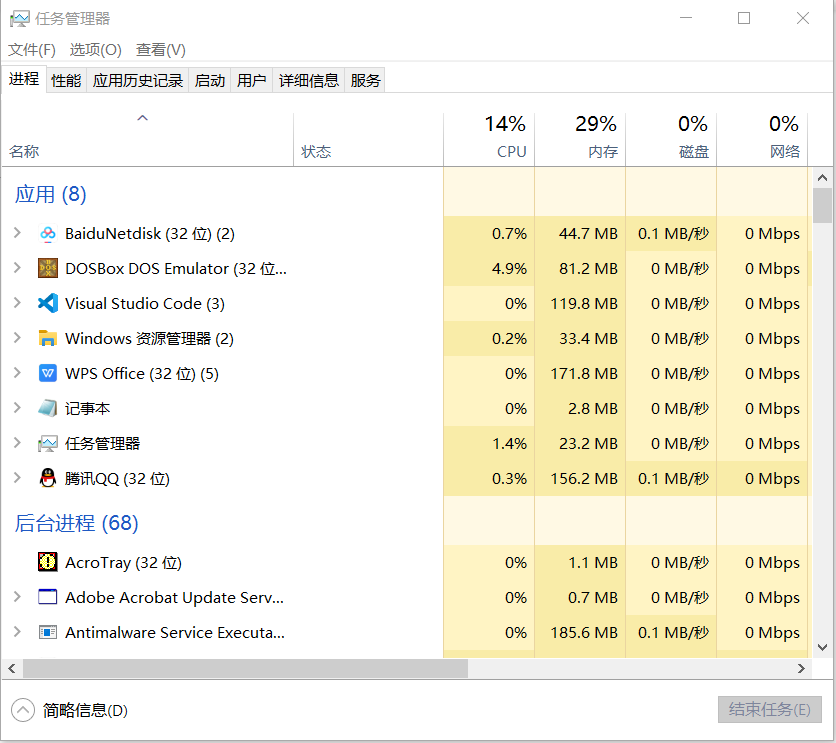


图2.1 1000次循环cpu使用量

循环100000次时：

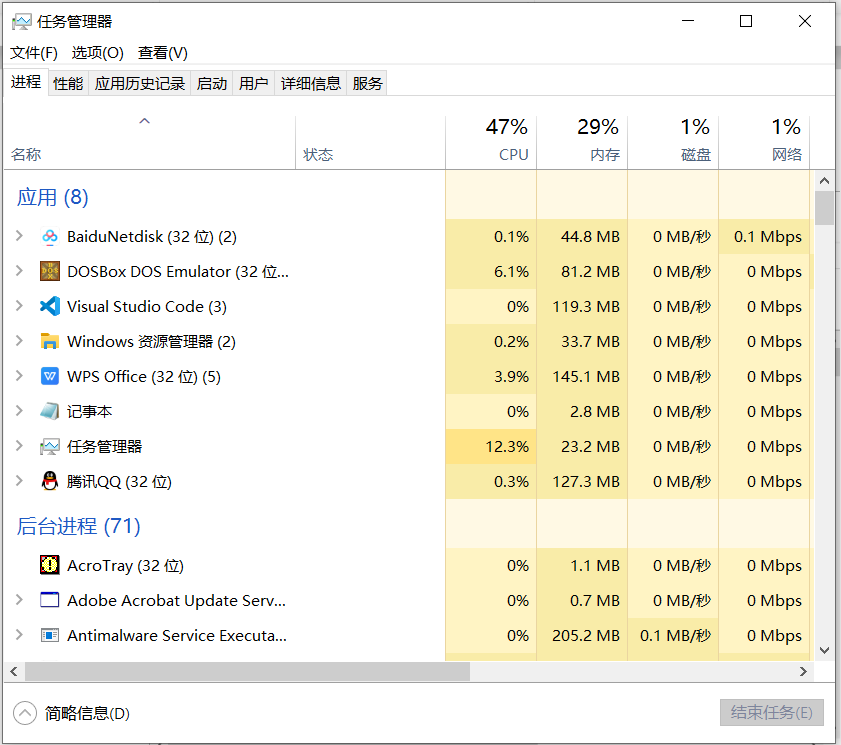


图2.2 循环10000次cpu使用量

正常情况下的cpu占比：

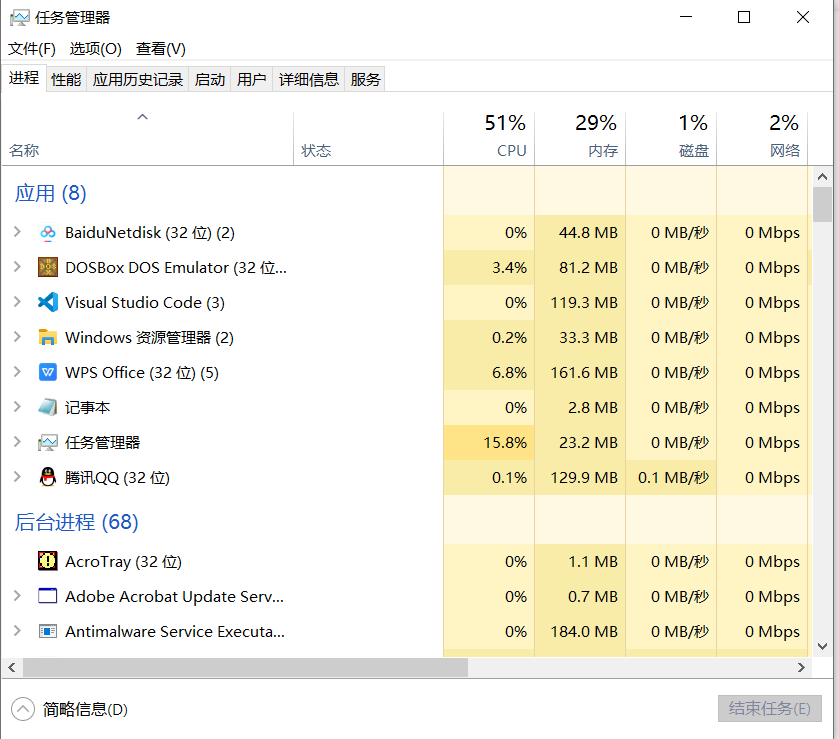


图2.3 正常情况下cpu使用量

我可以从上面的数据中发现，循环次数越多，cpu占比越大，占用资源更多。

问题2：内循环体中若有信息显示的代码（比如2号或9号功能调用），程序执行时间会有多大影响？

如果我在循环中添加一段输出字符串的指令：

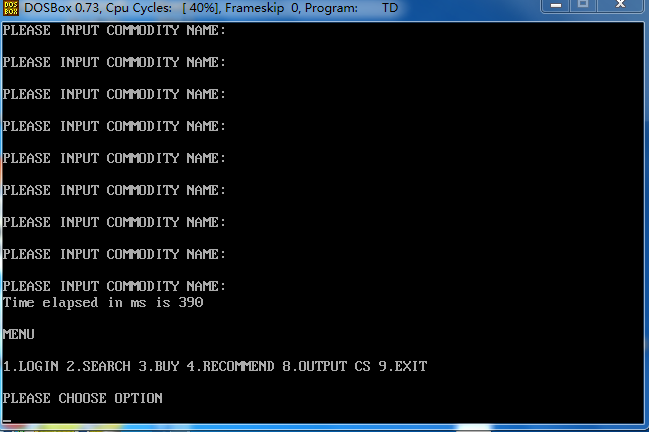


图2.4 输出字符串时运行时间

我会发现程序的执行时间忽然增加了很多，因此我可以总结出输出字符需要花费比较多的时间。

## 2.4 任务2.2实验过程

### 2.4.1实验方法说明

对于推荐度计算函数，我最开始使用的是移位指令，进行\*128的操作，已经是最优的算法，这里我将会把移位指令和乘法指令进行一个对比。此外，对于循环体内部的指令，如果能够合并，尽量合并。把能够从循环体拿出来的指令都拿出来加快执行速度。

（1）对计算推荐度函数代码优化前后对比如下(以优化商品1为例)：

表2.3 推荐度优化前后代码比较

|  |  |
| --- | --- |
| 优化前 | 优化后 |
| FUN4 PROC  LEA SI,GA1  XOR BX,BX  MOV AX,WORD PTR GA1+17 ;销售数量  MOV BX,WORD PTR GA1+15 ;进货数量  SAL BX,1  IMUL EAX,128 ;已售数量\*128  CWD ;字转双字  IDIV BX  MOV DI,AX ;存储结果  MOV AX,WORD PTR GA1+11 ;进货价  MOV BX,WORD PTR GA1+21 ;销售价  IMUL EAX,128 ;进货价\*128  CWD  IDIV BX  ADD AX,DI ;把两个结果相加  MOV WORD PTR [GA1+19],AX  LEA SI,GA2  XOR BX,BX  MOV AX,WORD PTR GA2+17  MOV BX,WORD PTR GA2+15  SAL BX,1  IMUL EAX,128  CWD  IDIV BX  MOV DI,AX  XOR BX,BX  MOV AX,WORD PTR GA2+11  MOV BX,WORD PTR GA2+21  IMUL EAX,128  CWD  IDIV BX  ADD AX,DI  MOV WORD PTR [GA2+19],AX  RET  FUN4 ENDP | FUN4 PROC  PUSH EDX  MOV EDX,0  LEA SI,GA1  XOR EBX,EBX  XOR EAX,EAX  MOV AX,WORD PTR GA1+17 ;销售数量  MOV BX,WORD PTR GA1+15 ;进货数量  SAL EBX,1  SAL EAX,7 ;已售数量\*128  IDIV EBX  MOV CX,AX ;存储结果  XOR EBX,EBX  XOR EAX,EAX  MOV AX,WORD PTR GA1+11 ;进货价  MOV BX,WORD PTR GA1+21 ;销售价  MOV EDX,0  SAL EAX,7 ;进货价\*128  IDIV EBX  ADD AX,CX ;把两个结果相加  MOV WORD PTR [GA1+19],AX  LEA SI,GA2  XOR EBX,EBX  XOR EAX,EAX  MOV AX,WORD PTR GA2+17  MOV BX,WORD PTR GA2+15  SAL EBX,1  SAL EAX,7  IDIV EBX  MOV CX,AX  XOR EBX,EBX  XOR EAX,EAX  MOV AX,WORD PTR GA2+11  MOV BX,WORD PTR GA2+21  MOV EDX,0  SAL EAX,7  IDIV EBX  ADD AX,CX  MOV WORD PTR [GA2+19],AX  POP EDX  POP CX  RET  FUN4 ENDP |

对计算推荐度函数时间优化前后对比如下(以优化商品1为例)：

表2.4 推荐度优化前后代码比较

|  |  |
| --- | --- |
| 优化前 | 优化后 |
|  |  |
| 500ms | 330ms |

分析：该子函数中将多处32位寄存器换为64位寄存器，运算速度理论上也得到提高，两处比较，代码长度也缩短不少，执行时间理论上有所减少，此外我优化后的代码考虑了防溢出，可以让进货数量有更大的初始值。

（2）对循环内部指令代码优化前后对比如下：

表2.5 循环指令优化前后比较

|  |  |
| --- | --- |
| 优化前 | 优化后 |
| P3:  MOV AX, 0  CALL TIMER ;开始计时  MOV CX,1000 ;初始化计数器  XUNHUAN1:  CMP GOOD,0 ;观察GOOD是否为0  JNZ NOTZERO1  JZ XUNHUAN1  NOTZERO1:  CALL FUN3  CALL FUN4  DEC CX  CMP CX,0  JNZ XUNHUAN1  P3LAST:  MOV AX,1  CALL TIMER ;输出计时结果  JMP INSTRUCT | P3:  MOV AX, 0  CALL TIMER ;开始计时  MOV CX,1000 ;初始化计数器  XUNHUAN1:  CMP GOOD,0 ;观察GOOD是否为0  JNZ NOTZERO1  JZ XUNHUAN1  NOTZERO1:  （把两个函数粘贴在这里）  DEC CX  CMP CX,0  JNZ XUNHUAN1  P3LAST:  MOV AX,1  CALL TIMER ;输出计时结果  JMP INSTRUCT |

### 2.4.2 实验记录与分析

（1）对计算推荐度函数时间优化前后对比如下(以优化商品1为例)：

表2.6 计算推荐度优化前后运行时间比较

|  |  |
| --- | --- |
| 优化前 | 优化后 |
|  |  |
| 500ms | 330ms |

分析：该子函数中将多处乘128替换成左移7位，运算速度理论上也得到提高，两处比较，执行时间理论上有所减少。

（2）对循环内部指令时间优化前后对比如下：

表2.7 循环内部指令优化前后运行时间比较

|  |  |
| --- | --- |
| 优化前 | 优化后 |
|  |  |
| 330ms | 210ms |

此处将子程序放入功能三模块中，避免了在循环中多次调用子程序造成时间浪费，优化后的程序执行效率明显提升，说明了调用函数虽然方便我们进行模块化设计，但是会极大地消耗程序执行的时间。

## 2.5小结

### 2.5.1 主要收获

问题1：汇编语言程序的优化可以从哪些方面进行？总结不同类别的优化措施对效率的影响程度。

汇编语言的优化可以从程序的指令的改写，结构的优化和合并，也可以直接对算法进行优化。我们这里的程序算法优化空间较小。我发现避免多次调用子函数可以极大地减少程序执行时间可以达到这里可以优化20%左右，此外，程序语句的改写优化率也比较有效，这里优化率可以达到10~20%。

问题2：循环体中用子程序调用与不用子程序调用会有多大影响？

此程序的优化率可以达到30%，十分出乎意料，说明调用子程序效率不高。

问题3：哪些指令是需要优化的关键性指令？

我把乘128在除以2优化成了乘64，再把乘128换成移位。效果比较明显，但是如果仅仅是把MOV换成XCHG这样的话优化并没有多少甚至还会出现负优化。

这次试验，对于我来说并不轻松，通过加入了多次循环，我发现了第一次写程序的时候没有考虑到的溢出在这次增加了循环次数和商品件数之后发现了程序的不足，并利用32寄存器进行了优化。这次的实验主要是观察执行时间和程序优化，让我更加深刻的了解到了每条指令的执行速度，也对于程序优化有了自己的经验。

### 2.5.2 主要看法

这次实验中我使用了老师所给的计时函数，但是在td过程调试中，输出的结果并不是十分的稳定，甚至有的时候同一个程序会直接输出0的情况。感觉是环境不稳定造成的，每次执行的时候调节了频率，可能是虚拟机并没有很快的做出反应，导致了这种结果的出现。另外个人觉得寄存器是真的不够用，在结束一段程序后必须立刻对寄存器做好回收，对于寄存器不够的情况，可以开辟新的内存单元来作为中间变量来存储信息，才能让寄存器腾出来进行操作，比高级语言麻烦了不少。

# 3模块化程序设计

## 3.1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。

## 3.2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。

4.将本次新增功能的子程序放到另外单独的模块中，按照模块化程序设计的方法搭建系统。

任务3.2：在C语言程序中调用汇编语言实现的函数

将网店商品信息管理系统中菜单界面的功能用C语言实现，其他子功能可以仍采用汇编语言实现。在C语言程序中调用汇编语言子程序。

## 3.3 任务3.1实验过程

### 3.3.1 设计思想及单元分配

（1）一开始我就使用了子程序来设计这个管理系统，因此这次可以省去这个步骤

（2）把重复的地方拿宏来实现

1.读取字符串指令替换为WRITE指令：

2.输出字符串指令替换为READ指令：

表3.1 宏定义

|  |  |
| --- | --- |
| WRITE | READ |
| WRITE MACRO A  MOV DX,A  MOV AH,9  INT 21H  ENDM | READ MACRO A  LEA DX,A  MOV AH,10  INT 21H  ENDM |

（3）在网店商品信息管理系统中添加显示商品信息的功能

改进后的功能二得流程如下：

1.首先调用FUN2来匹配商品

2.在FUN2中成功匹配到商品，调用PRINT\_INFO函数来输出相应商品的信息

3.分别输出商品名称，折扣，销售量等信息，在每个信息输出的时候先把数据赋值给EAX作为参数传入转换为10进制的子程序中。

4.输出完成返回主菜单

（4）实现功能六对商品进行信息修改的操作

功能六的设计流程如下：

1.首先比较AUTH是否为0来判断是否是老板登录状态

2.比较GOOD中数字是否为0来判断当前是否在浏览商品

3.按照折扣，进货价，销售价，进货总数分别显示原来的数值，进行修改操作，中间调用RADIX函数来实现进制转换。

### 3.3.2 源程序

下面展示输出信息的函数和进制转换输出函数：

（1）输出商品信息函数：

PRINT\_INFO PROC

MOV CX,GOOD

DEC CX

LEA SI,GA1

CMP CX,0

JZ PRINT\_NAME

LOOP4:

ADD SI,23

LOOP LOOP4

PRINT\_NAME:

PUSH SI

MOV CL,BYTE PTR [SI+9]

LOOP5:

MOV DL,BYTE PTR [SI]

MOV AH,2

INT 21H

INC SI

LOOP LOOP5 ;输出商品名称

POP SI

ADD SI,10

WRITE TIP14

MOV EAX,0

MOV AL,[SI]

CALL RADIX

INC SI

WRITE TIP15

MOV EAX,0

MOV AX,WORD PTR [SI]

CALL RADIX

ADD SI,2

WRITE TIP16

MOV EAX,0

MOV AX,WORD PTR [SI]

CALL RADIX

ADD SI,2

WRITE TIP17

MOV EAX,0

MOV AX,WORD PTR [SI]

CALL RADIX

ADD SI,2

WRITE TIP18

MOV EAX,0

MOV AX,WORD PTR [SI]

CALL RADIX

ADD SI,2

WRITE TIP19

MOV EAX,0

MOV AX,WORD PTR [SI]

CALL RADIX

ADD SI,2

MOV DL,0AH

MOV AH,2

INT 21H

RET

PRINT\_INFO ENDP

RADIX PROC ;需要事先把EAX赋值

PUSH CX

PUSH EDX

PUSH SI

MOV SI,NUM

XOR CX,CX

MOV EBX,10

LOP1:

XOR EDX,EDX

DIV EBX

PUSH DX

INC CX

OR EAX,EAX

JNZ LOP1

LOP2:

POP AX

CMP AL,10

JB L1

ADD AL,7

L1:

ADD AL,30H

MOV DL,AL

MOV AH,2

INT 21H

INC SI

LOOP LOP2

POP SI

POP EDX

POP CX

RET

RADIX ENDP

### 3.3.3 实验步骤

1. 准备上机环境，在vscode中编写程序。

2. 这次试验我决定把第一次试验的代码全部重构一遍，主要是之前没有系统的学习循环和堆栈的操作，之前的步骤过于的繁琐，并且这一次还充分的考虑了程序溢出的情况。

3. 这里我为了计算方便，专门给实际售价分配了存储空间，方便我直接调用，并且在修改折扣后也可以快速计算好当前售价，方便了我后面的操作。

4. 这次试验使用的RADIX函数改编自课本上的程序。

5. 这次试验把输入输出调用dos21号操作改成了宏定义，方便了程序的编写，增加了可读性，但是注意在宏定义中不可以使用强制转换操作，因为很有可能之后使用的时候强制转换的类型不对，给自己带来不必要的麻烦。

6. 这次实验还需要观察EXTERN变量，感觉会和c语言的有些类似。

### 3.3.4 实验记录和分析

1.实验环境条件：windows10下的dosbox0.73，masm.exe，link.exe。

2.试验流程：

（1）首先我把第一次实验的功能1234调整成为循环，缩短了代码长度，商品容量也更改为了30，并且有了第二次试验的教训，我这次特意留意了程序溢出问题。

（2）编写的第一个函数是输出商品信息，这里调用了书上的RADIX函数，先把需要转换的数放入EAX寄存器中，在根据asc码转换成字符串存入INNUM数组中。

（3）编写的第二个功能是修改商品信息的函数，我自己编写了LOAD\_DATA函数来把二进制数转换为10进制数，最终的结果放在EAX寄存器中，在FUN6函数中存入相应的内存单元中，完成修改。

（4）完成模块化设计：首先，我先建立了一个名为MACRO.LIB的宏库放在文件中，把需要用到的宏放在里面。

我这次把程序分为了两个模块，一个是包含123489功能的TEST3.ASM，命名为：



图3.1 主商店

第二个是包含功能二的部分函数和功能6的TEST4.ASM，命名为：



图3.2 分模块名

关于TEST3.ASM的全局定义和接口如下：

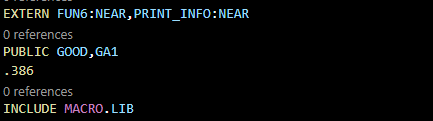


图3.3 模块一声明

这里的FUN6和PRINT\_INFO函数在TEST4.ASM中声明为PUBLIC，这个函数的GOOD和GA1作为全局变量参数和TEST4.ASM共享。

关于TEST4.ASM的全局定义和接口如下：

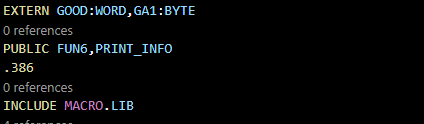


图3.4 模块二声明

这里的FUN6，PRINT\_INFO被声明为全局变量和TEST3.ASM共享，同时这里接收GOOD,GA1两个变量，定位方式分别为WORD和BYTE。

（5）先对两个ASM文件使用MASM生成OBJ文件，再将两个OBJ文件用LINK联合生成TEST3.EXE：



图3.5 链接两个obj文件

运行TEST3.EXE：



图3.6 运行exe文件

（6）程序运行效果：

首先直接进入功能2，查找BOOK商品：

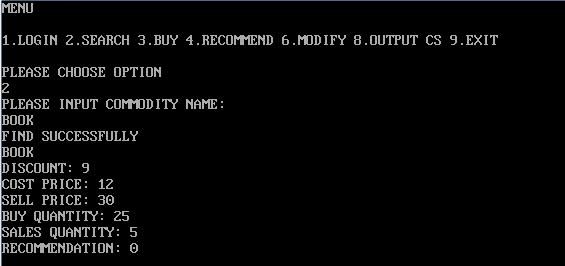


图3.7 查找商品

这里发现推荐度还没有计算，接下来调用功能4，再调用功能2：

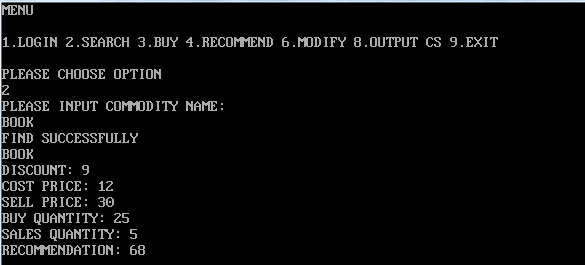


图3.8 计算推荐度

发现推荐度被计算出来，商品的完整信息全部输出。

接下来进入功能6对BOOK商品进行信息修改：



图3.9 修改商品信息

显示提示信息，未登录，不是老板状态，于是我们先登陆，再进入功能6：

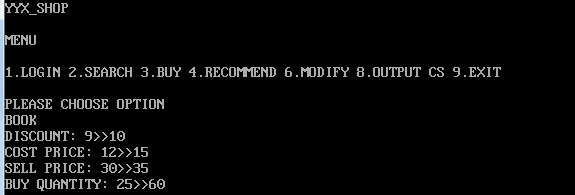


图3.10 登陆后修改商品信息

这里我们把折扣修改为10，进货价格修改为15，销售价格更改为35，进货量改为60，再调用功能4，重新计算所有商品推荐度，再进入功能2搜索BOOK：

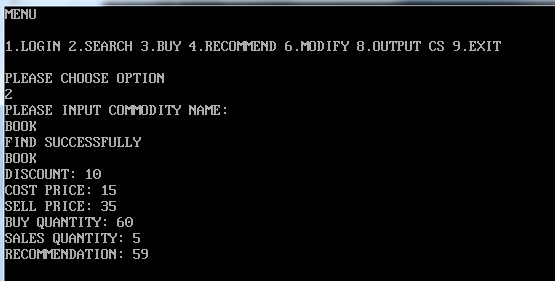


图3.11 重新搜索该商品

发现值被成功修改，推荐度重新计算并且经验证为正确的。

3.汇编过程中，遇到的报错如下：

C:\Users\WidenSun\AppData\Roaming\Tencent\Users\2056404413\QQ\WinTemp\RichOle\}[RQ`}))402H[RVHBB2W%XA.png

图3.12 报错一截图

这是在调用宏库的时候，LIB文件末尾没有加上一个回车符，导致的报错，我对这个原理还不太清楚。

此外在刚开始我的第二个程序并没有使用EXTERN，我以为把两个程序的数据段合并之后就可以直接调用第一个程序里面数据段的内容，但是编译一直通过不了，最后我才意识到为什么课本上说每个模块要有独立性，可以直接通过编译，未知的参数得先把接口先写好，才能调用。相当于虽然最终EXE文件是两个文件融合在一起，但其实每个文件单独是一个完整的，可以通过编译的文件。

问题回答：

1. 在跟踪调试的时候，我使用F7进入了子程序内部进行单步调试，每次进入子程序的时候，堆栈段会自动把调用子程序的下一个代码段的地址压入栈中，在执行RET语句的时候弹出，把地址值赋给IP寄存器，和课本上的流程一致。

2. NEAR子程序和FAR调用子程序，我观察堆栈的时候，发现NEAR只是把偏移地址压入栈中，如果是FAR，则会把偏移地址和段地址会被压入（即CS和IP）。

3. 如果我强行对一个NEAR函数执行CALL FAR FUN6，会报错如下：

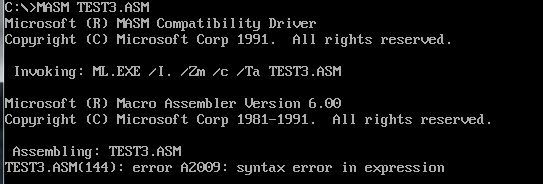


图3.13 报错二截图

如果我对一个FAR函数强行执行CALL NEAR FUN6，会报错如下：

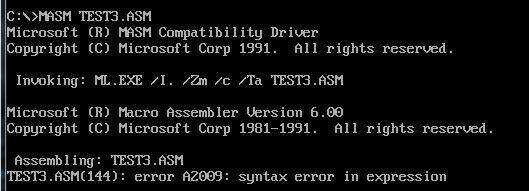


图3.14 报错四截图

4. 我尝试了堆栈传递和直接寄存器传递参数，最后发现直接寄存器传比较浪费寄存器，但是简单明了。堆栈法传递参数，更容易出错，经常会漏算地址导致错误访问参数。

5. 通过观察td中反汇编语句，我发现宏定义就是一个程序替换过程。

6. 模块间传递参数的时候如果参数名打错了会直接报错，提示该参数UNRESLOVE。

7. EXTERN和EXTRN貌似可以通用，我尝试了两种写法都得到了一样的效果，关于EXTRN在.386前面和后面的区别如下：

在.386后面：

C:\Users\WidenSun\Desktop\QQ图片20200406212053_WPS图片.jpg

图3.15 EXTERN在.386后面

在.386前面：

C:\Users\WidenSun\Desktop\QQ图片20200406212053_WPS图片1.jpg

图3.16 EXTERN在.386前面

最后我发现就是.386的作用是32位寻址，和16位寻址的区别，在.386后面就是16位，在.386前面就是32位寻址。

## 3.4 任务3.2实验过程

### 3.4.1 设计思想及单元分配

这次使用的是Visual Studio平台，系统是windows7.

首先在源文件中建立两个源文件，一个使用c语言实现菜单，汇编语言编写的子程序放在.asm程序中。最后点击生成解决方案，就可以得到一个可执行文件。



图3.17 工程建立

重点在于体会c语言如何调用汇编语言编写的子程序。

### 3.4.2 源程序

商店的商品存储结构体如下：

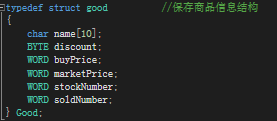


图3.18 商品存储结构申明

汇编语言编写的函数在c语言程序中的声明如下：

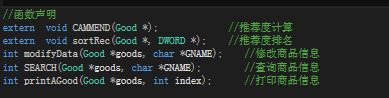


图3.19 函数声明

C语言编写的菜单程序如下：

system("cls");

printf("\t\tYYX\_SHOP\t\n1.SEARCH 2.BUY 3.RECOMMEND 4.PRINTGOOD 5.EXIT\n");

printf("Please input your option:");

scanf("%d", &op\_);

switch (op\_)

{

case 1:

printf("Please input the name of the commodity: ");

scanf("%s", GNAME);

temp\_int = SEARCH(goods, GNAME);

if (temp\_int == 0)

{

printf("!\n");

}

else

{

/\* 输出商品信息子程序 \*/

printAGood(goods, temp\_int - 1);

}

break;

case 2:

printf("Please input the commodity you want to buy: ");

scanf("%s", GNAME);

temp\_int = SEARCH(goods, GNAME);

if (temp\_int == 0 || goods[temp\_int - 1].stockNumber == goods[temp\_int - 1].soldNumber)

printf("Not found!\n");

else{

printf("Buy successfully!\n");

}

break;

case 3: //计算推荐度

CAMMEND(goods);

printf("Calculate successful!\n");

break;

case 4:

for (int i = 0; i < N; i++)

{

printAGood(goods, i);

}

break;

case 5:

return 0;

default:

break;

}

system("pause");

}

### 3.4.3 实验步骤

首先根据老师的提示，现在平台上编写好c语言的菜单，这里我把第一个功能使用c语言实现了，基本流程和之前汇编编写的程序是一样的，准备之后观察反汇编语句。

接着使用c语言编写了菜单，感觉switch结构很方便。

接着，我开始把之前编写的汇编函数放在demo.asm文件中，修改了原来函数的一些操作，但是设计流程保持一样，但是这里，由于我么你又数据段，因此需要我传递参数，于是我在每个函数的声明后面加上了参数定义：



图3.20 传参定义

这里定义了参数的类型。

### 3.4.4 实验记录和分析

接下来就是测试我所写的程序是否能够运行：

首先测试密码输入：

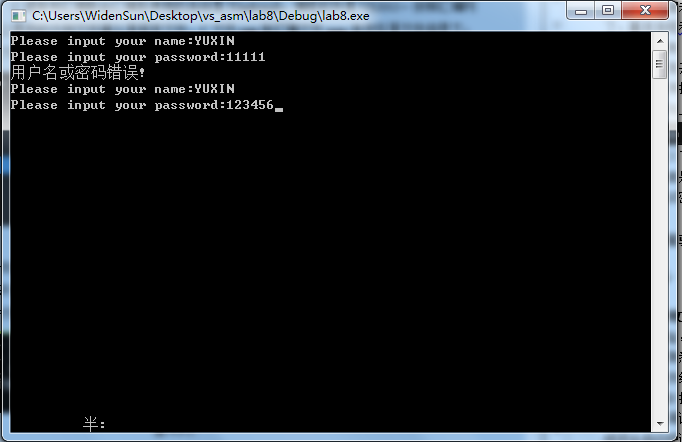


图3.21 登陆功能

这里我的密码是YUXIN,123456，可以发现程序可以识别出错误输入，我登陆进入老板界面：

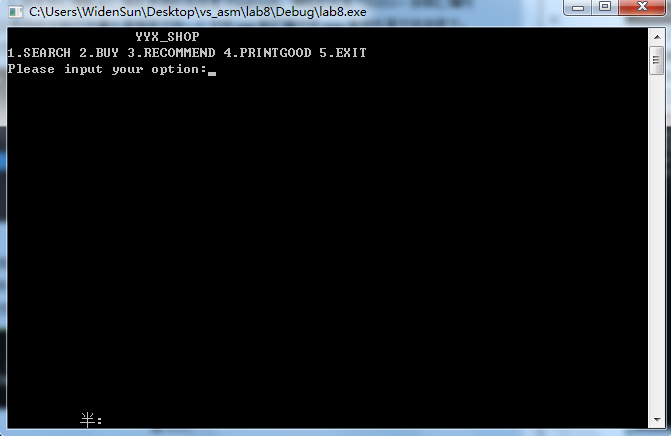


图3.22 成功登陆后进入主页面

首先测试第一个查找功能：

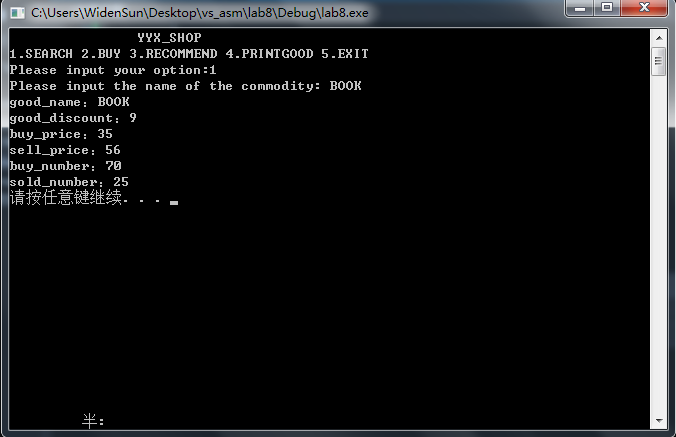


图3.23 查找功能

成功的找到了该商品并且输出了商品信息。

接下来测试购买功能：

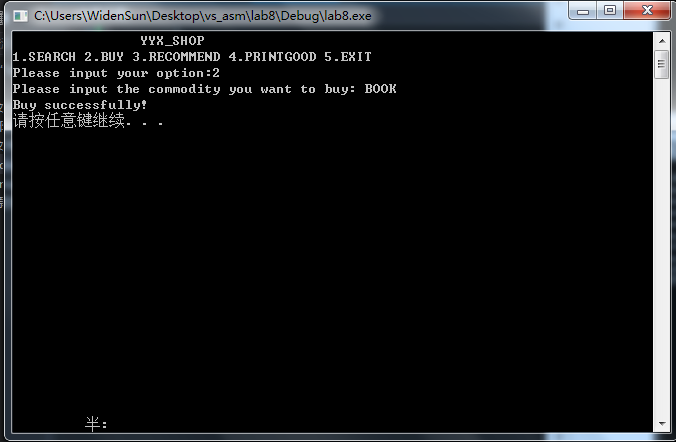


图3.24 购买功能

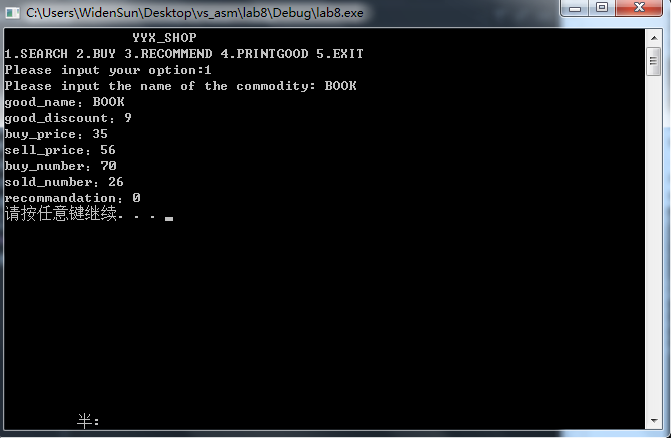


图3.26 查看是否购买成功

发现这里的sold\_number变成了26，加了一，代表购买成功。

接下来测试第三个功能推荐度计算：

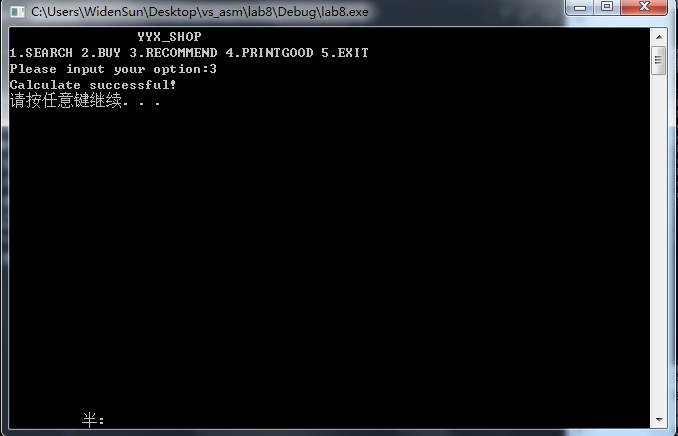


图3.27 计算推荐度

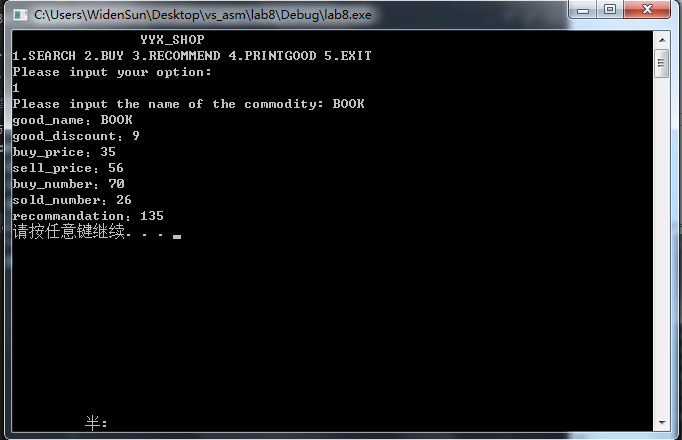


图3.28 观察推荐度是否成功计算

这里我们发现推荐度被成功计算。

接下来我们测试功能四输出所有的商品信息：

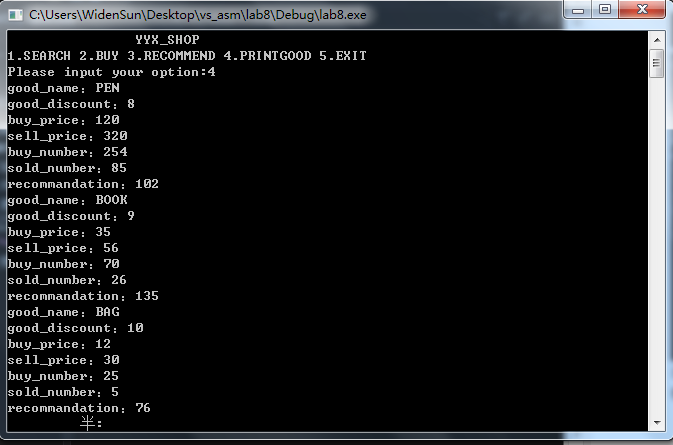


图3.29 输出所有商品信息

证明我编写的程序可以对所有商品进行操作，我也可以看到所有商品信息。

最后测试功能5，成功退出：

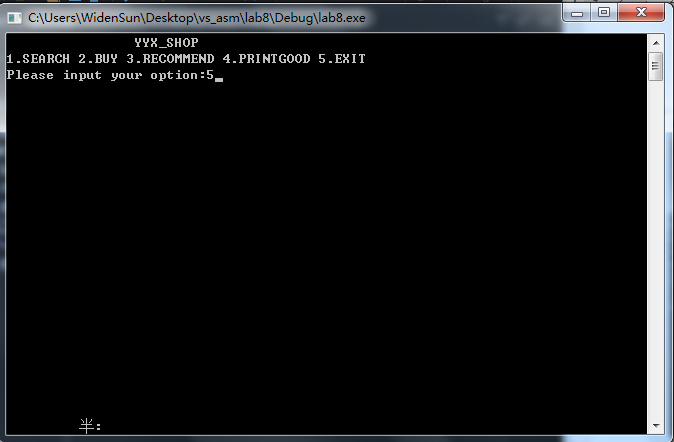


图3.30 退出

到此，程序所有功能测试完成！

打开lab8的debug文件夹，我们可以很快的就找到生成的obj文件：

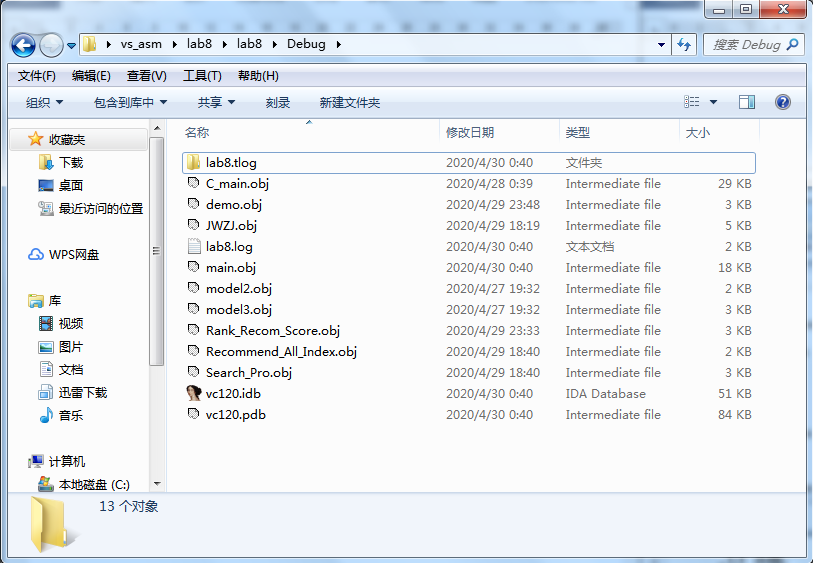


图3.31 查看生成的文件

根据我所学到的知识，我可以知道这里是先生成了obj文件，最后再把两个文件link成了exe文件，到此 ，实验操作完毕。

## 3.5 小结

### 3.5.1 主要收获

任务3.1这次实验，主要就是添加了两个功能，重点是如何实现模块化编程，对于代码段数据段以及堆栈段的接口熟悉一遍，也是更加深入的了解了生成EXE文件的步骤，对于不同模块间的参数传递有了一定的操作经验。此外在编写宏文件的时候，感觉和其他高级语言比较类似，直接编写好一个库，使用INCLUDE指令进行调用，极大地减少了代码的大量的重复。

在跟踪调试的时候，我使用F7进入了子程序内部进行单步调试，每次进入子程序的时候，堆栈段会自动把调用子程序的下一个代码段的地址压入栈中，在执行RET语句的时候弹出，把地址值赋给IP寄存器，和课本上的流程一致。NEAR子程序和FAR调用子程序，我观察堆栈的时候，发现NEAR只是把偏移地址压入栈中，如果是FAR，则会把偏移地址和段地址会被压入（即CS和IP）。如果我强行对一个NEAR函数执行CALL FAR FUN6，会报错，如果我对一个FAR函数强行执行CALL NEAR FUN6，会报错。我尝试了堆栈传递和直接寄存器传递参数，最后发现直接寄存器传比较浪费寄存器，但是简单明了。堆栈法传递参数，更容易出错，经常会漏算地址导致错误访问参数。通过观察td中反汇编语句，我发现宏定义就是一个程序替换过程。模块间传递参数的时候如果参数名打错了会直接报错，提示该参数UNRESLOVE。EXTERN和EXTRN貌似可以通用，我尝试了两种写法都得到了一样的效果，最后我发现就是.386的作用是32位寻址，和16位寻址的区别，在.386后面就是16位，在.386前面就是32位寻址。

我通过已经完成的几次试验，充分地发现了自己能力的欠缺，每次实验之前都得花一天时间熟悉实验，做好预习，并且在上课之前基本完成实验内容，在课堂上就可以编写实验报告的同时留意群里面同学和老师的讨论，有更加充裕的时间去了解不同同学遇到的不同的问题，防止自己以后把时间浪费在这些问题上，对于自己的提升是巨大的。

任务3.2，这次试验虽然只是修改一下从前的汇编程序，我却做得并不轻松，但是在这个过程中，我通过对于反汇编指令的观察，也可以得到汇编语言效率会优于C语言：

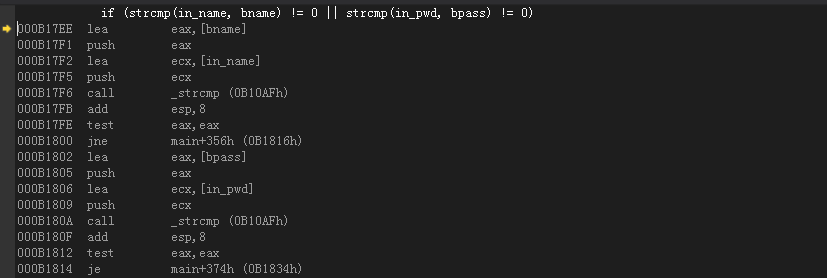


图3.32 查看反汇编语句和c语言对比

这里我调用了字符串比较函数，在原来的汇编程序中：

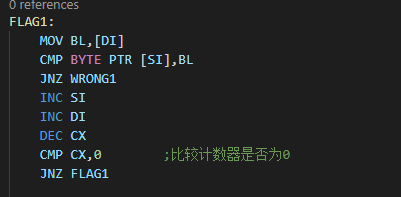


图3.33 原来的字符串匹配

我主要就是简单地使用循环进行字符串匹配，但是他这里却是要老老实实地调用函数，涉及到的传参，就会导致程序变慢很多，此外，函数的进入与返回还需要多占用堆栈空间，更是减慢了操作速度，通过了解我得知，许多程序在进行优化的时候，最好的结果是重新拿汇编语言重写某一个片段，可以取到事半功倍的效果。

通过上面的这个例子，我也发现了c语言在使用寄存器的时候也是十分随意的，比较容易造成后续的汇编语言子程序产生寄存器被破坏的情况，更加提醒我要在子程序编写的过程中十分小心提防寄存器没保护的情况，只有这样才能够防止程序出现意想不到的bug。

### 3.5.2 主要看法

这次实验使用的依旧是dosbox和td界面，不过我这次在vscode中安装了代码插件，编写程序的时候会有彩色关键字提示，避免了拼写错误，界面也十分养眼，希望能够专门为汇编语言做一套开源支持各种插件的软件，加快编写和调试程序的同时，对代码编辑器风格也可以进行个性化设置，可以把各种集成开发平台都嵌入其中的软件，真正做到个性化开发调试。

这次试验虽然使用了VS开发平台，但是我本人还是觉得，vs经常会出现一些我意想不到的报错，并且有的报错还会有多种可能，这里我还在c语言程序编写的时候经常碰到该函数不安全，才知道还有更安全的函数可以使用，也是增加了我的见识。但是不得不说，这个反汇编确实很直观的把每个高级语言的语句都翻译成了汇编语句，方便了我的阅读和比较，还是有很大的优点的。

# 4中断与反跟踪

## 4.1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。

## 4.2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。

## 4.3任务4.1实验过程

### 4.3.1设计思想及单元分配

（1）任务I/O：如何在TD下使用IN/OUT指令获取CMOS数据

根据老师所给的提示，我决定读取机器的时分秒三个数据并且输出在屏幕上。

分配三个变量来分别存储时分秒：

HOUR DB ?,?,':' ;时的信息

MIN DB ?,?,':' ;分的信息

SEC DB ?,?,' ' ;秒的信息

在TD下使用IN/OUT指令获取CMOS数据，并且解压缩之后得到系统的时分秒，关键代码如下：

MOV AL,2

OUT 70H,AL

JMP $+2

IN AL,71H ;获取数据

MOV AH,AL ;解压

AND AL,0FH

SHR AH,4

ADD AX,3030H

XCHG AH,AL

MOV WORD PTR SEC,AX ;获取秒

（2）中断矢量表

首先直接在TD中观察中断矢量表中的信息。

接下来利用程序获取中断矢量表中的数据，设计思路：分别将3501H和3513H赋值给AX，35H功能能够获取中断信息，中断号保存在AL中，用指令INT 21H进行系统功能调用即可，程序代码如下：

.386

STACK SEGMENT STACK USE16

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT USE16

ASSUME CS:CODE,SS:STACK

START:

XOR AX,AX

MOV DS,AX

MOV AX,3501H

INT 21H

MOV AX,3513H

INT 21H

MOV AH,4CH

INT 21H

CODE ENDS

END START

接下来编写程序，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）。

设计：计算出中断号1H和13H所在的地址，赋值给AX,BX即可。代码如下：

.386

STACK SEGMENT STACK USE16

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT USE16

ASSUME CS:CODE,SS:STACK

START:

XOR AX,AX

MOV DS,AX

MOV AX,DS:[1H\*4]

MOV BX,DS:[13H\*4]

MOV AH,4CH

INT 21H

CODE ENDS

END START

（3）中断服务程序：

观察

核心代码如下：

XOR AX,AX

MOV DS,AX

MOV AX,DS:[16H\*4]

MOV OLD\_INT,AX ;保存旧的中断矢量

MOV AX,DS:[16H\*4+2]

MOV OLD\_INT+2,AX

CLI ;关中断，防止被打断

MOV WORD PTR DS:[16H\*4],OFFSET NEW13H

MOV DS:[16H\*4+2],CS

STI ;开中断

MOV DX,OFFSET START+15

MOV CL,4

SHR DX,CL

ADD DX,10H

MOV AL,0

MOV AH,31H ;退出时，程序驻留

INT 21H

（4）内存操作

关于堆栈段迁移，我决定进行堆栈段的复制，把第一个堆栈段里的数据复制到第二个堆栈段里，在数据段中设置一个CHANGE\_STACK\_TIP内存用来判断迁移方向。

### 4.3.2 实验步骤

1. 准备上机环境，在vscode中编写程序。

2. 这次试验涉及到了中断的读写，我首先重新复习了一遍中断的读写。

3. 这次实验需要使用IN/OUT指令读取CMOS中的数据，需要我先把任务书中的附录部分仔细研究一下。

4. 这次试验我决定使用书上的计时器例子作为我的中断重新写入的子程序

5. 对于8号中断，我逐渐理解了为什么是每秒18.2次以及这是一个硬件中断程序，机器会自动每55ms调用一次8号中断，我们这次利用的就是这个软中断指令，在中断的同时不会影响到程序的进程。

6. 这次实验延伸出来的切换堆栈段的任务，我认为复制的工作需要十分的小心，防止由于复制的不到位导致程序死机甚至错误的返回。

### 4.3.3实验记录与分析

任务4.1：

1.如何在TD下使用IN/OUT获取CMOS数据：

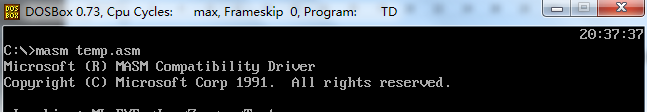


图4.1 输出系统时间

在屏幕的左上角输出的是电脑的时间



图4.2 实际系统时间

对比之后发现与机器时间相同，代表输出成功。

在菜单中添加加载新中断：



图4.3 添加新的中断

之后我开始编写切换段的中断，关键代码如下：

MOV AX,WORD PTR SEC

CMP AH,'0'

JNZ QUIT

CMP AL,'0'

JNZ QUIT

MOV AX,WORD PTR CHANGE\_STACK\_TIP

CMP AX,0

JZ J1\_2

JNZ J2\_1

J1\_2:

MOV CX,150

MOV BX,STACKBACK

MOV ES,BX

MOV SI,298

LOP20:

MOV AX,SS:[SI]

MOV ES:[SI],AX

SUB SI,2

DEC CX

LOOP LOP20

MOV AX,STACKBACK

MOV SS,AX

ADD WORD PTR CHANGE\_STACK\_TIP,1

JMP QUIT

J2\_1:

MOV CX,150

MOV BX,STACK

MOV ES,BX

MOV SI,298

LOP21:

MOV AX,SS:[SI]

MOV ES:[SI],AX

SUB SI,2

DEC CX

LOOP LOP21

MOV AX,STACK

MOV SS,AX

SUB WORD PTR CHANGE\_STACK\_TIP,1

JMP QUIT

在编写程序的过程中我遇到了很多的困难：

1.首先我先把切换段的功能写在子程序中，调试通过之后在嵌入进8号中断中，但是又出现了错误，这个时候，我不知道应该如何跟进到中断里面进行调试。在询问了老师之后，只能先设好断点之后才能进入中断中调试，我使用的是f2和f4来进行跟踪调试。

2.

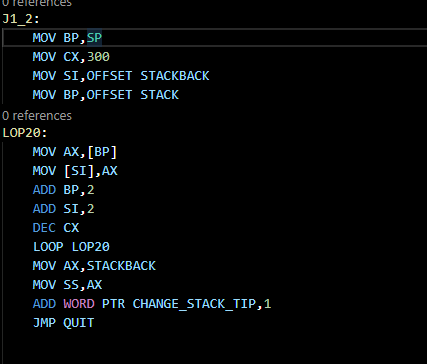


图4.4 段的切换错误示范

在编写段切换的程序中，我刚开始想当然的直接使用BP和SI进行间接寻址，却忽略了这两个寄存器的默认段，最后我使用了ES附加段寄存器完成了段的切换：

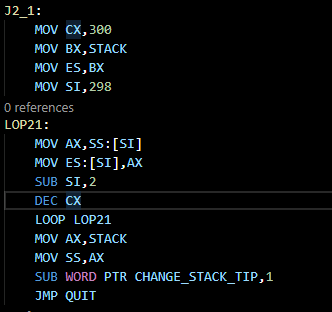


图4.5 段的切换正确示范

首先输入功能7，加载新中断之后，我把原来程序的功能8修改成了输出SS的首地址，在秒数为00的时候发生段的切换。

得到的运行结果如下图：

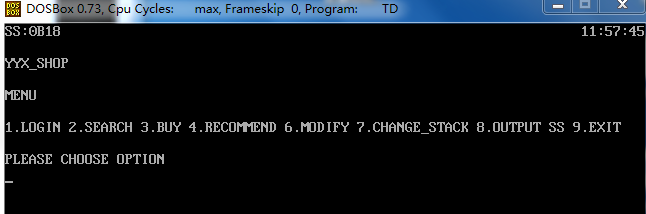


图4.6 切换前的堆栈段地址

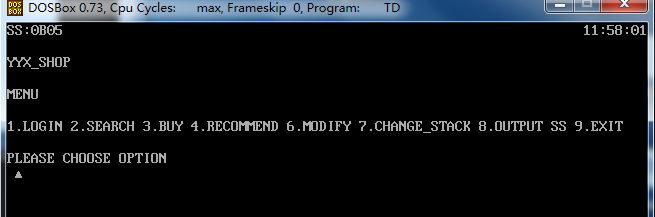


图4.7 切换后的堆栈段地址

第一次显示的是0B18，第二次显示的是0B05，堆栈段明显发生了切换，跟踪进入中断程序中，发现切换之后的堆栈段内的数据和切换前的一样，证明切换成功，堆栈段数据成功迁移：

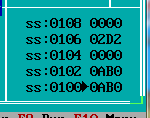


图4.8 dosbox中切换前的堆栈段地址

（切换前的堆栈段）

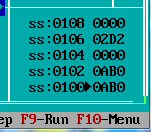


图4.9 dosbox中切换后的堆栈段地址

（切换后的堆栈段）

此外，在每相邻分钟各运行一个商店功能，发现功能可以正常使用，证明做到了全部数据迁移。

任务4.1圆满完成。

## 4.4任务4.2和4.3实验过程

### 4.4.1设计思想及单元分配

首先对于把密码商品进价等信息进行加密，我准备选择把信息和固定的字母进行异或操作。

具体设计是：把老板姓名和‘X’进行异或，把商品进货价和‘Y’进行异或。

进行反跟踪过程中，我决定首先使用修改中断向量表的方法，把九个子程序的入口地址存入地址表中，这样必须得执行程序的时候才会把子程序地址送到寄存器中，可以扰乱跟踪的思路。此外，我还使用了检查堆栈和计时来防止对方对我的程序进行跟踪。存储密码的时候我准备采取函数（X-30H）\*3对密码进行加密。

这里根据所给出的参考程序，我决定再使用计时的方法来抵制动态调试跟踪。如果时间发生了较大的偏差，便跳转到误导区域，破坏程序原来的进程，防止队友利用动态跟踪破译出我的密码。

（1）加密部分源代码：

BNAME  DB  'Y' XOR 'X','U' XOR 'X','X' XOR 'X','I' XOR 'X','N' XOR 'X',5 DUP(0)  ;老板姓名

BPASS  DB ('1'-30H)\*3      ;这里采用(X-30H)\*3来对密码进行加密

    DB ('2'-30H)\*3

    DB ('3'-30H)\*3

    DB ('4'-30H)\*3

    DB ('5'-30H)\*3

    DB ('6'-30H)\*3

    DB 0B2H,0FFH ,3BH  ;密码

GA1   DB 'PEN', 6 DUP(0) ,3 ,10  ;商品名称及折扣

    DW 35 XOR 'Y',56,70,25,?,?   ; 进货价格，销售价格，进货数量，销售数量，推荐度还未计算

GA2  DB 'BOOK', 5 DUP(0) ,4 ,9  ;商品名称及折扣

   DW  12 XOR 'Y',30,25,5,?,?   ;推荐度还未计算

GAN  DB N-2 DUP( 'TempValue' ,0,8,15 XOR 'Y',0,20,0,30,0,2,0,?,0,?,0)

（2）修改中断向量表部分源代码：

MOV AX,WORD PTR ES:[1\*4]        ;存储旧的1,3中断

    MOV OLD\_INT1,AX

    MOV AX,WORD PTR ES:[1\*4+2]

    MOV OLD\_INT1+2,AX

    MOV AX,WORD PTR ES:[3\*4]

    MOV OLD\_INT3,AX

    MOV AX,WORD PTR ES:[3\*4+2]

    MOV OLD\_INT3+2,AX

    CLI                             ;新的1,3中断

    MOV ES:[1\*4],OFFSET NEW\_INT

    MOV ES:[1\*4+2],CS

    MOV ES:[3\*4],OFFSET NEW\_INT

    MOV ES:[3\*4+2],CS

    STI

（3）添加计时检查部分源程序：

CLI

MOV AH,2CH

INT 21H

PUSH DX

LEA SI,BNAME

STI

MOV AH,2CH

INT 21H

CMP DX,[ESP]

POP DX

JNZ GO3

MOV CH,0

MOV CL,COUNT1 ;验证姓名

LEA DI,INNAME+2

CMP CL,INNAME+1 ;长度不相等，姓名肯定错误

JNZ WRONG1

FLAG1:

MOV BL,[DI]

XOR BL,'X'

CMP BYTE PTR [SI],BL

JNZ WRONG1

INC SI

INC DI

DEC CX

CMP CX,0 ;比较计数器是否为0

JNZ FLAG1

MOV CL,COUNT2 ;验证密码

LEA SI,BPASS

LEA DI,INPASSWORD+2

CMP CL,INPASSWORD+1 ;长度不相同，密码肯定错误

JNZ WRONG2

（4）堆栈检查部分源代码：

PUSH SI

PUSH CX

PUSH EBX

PUSH EAX

PUSH EDX

CLI

PUSH XUNHUAN

MOV CX,N

LEA SI,GA1

POP AX

MOV BX,[ESP-2]

STI

JMP BX

XUNHUAN:

PUSH CX

MOV EDX,0

XOR EBX,EBX

XOR EAX,EAX

MOV AX,WORD PTR [SI+17] ;销售数量

MOV BX,WORD PTR [SI+15] ;进货数量

SAL EBX,1

SAL EAX,7 ;已售数量\*128

IDIV EBX

MOV CX,AX ;存储结果

XOR EBX,EBX

XOR EAX,EAX

MOV AX,WORD PTR [SI+11] ;进货价

XOR AX,'Y'

MOV BX,WORD PTR [SI+21] ;销售价

MOV EDX,0

SAL EAX,7 ;进货价\*128

IDIV EBX

ADD AX,CX ;把两个结果相加

MOV WORD PTR [SI+19],AX

ADD SI,23

POP CX

LOOP XUNHUAN

WRITE TIP12

POP EDX

POP EAX

POP EBX

POP CX

POP SI

### 4.4.2 实验步骤

1. 准备上机环境，在vscode中编写程序。

2. 这次试验涉及到了中断的读写，我首先重新复习了一遍中断的读写。

3. 首先根据已经给出的提示程序，理解思路。

4. 根据自己之前设计的函数和加密方式来对程序的数据段进行加密。

5. 修改完成之后，跟踪调试，验证程序在跟踪的时候是否做到了迷惑破解者的功能。

6. 把exe程序发送给队友，让队友进行破译。

7. 接受队友的exe程序使用IDA反汇编软件进行分析，看看是否能够破解出密码。

### 4.4.3实验记录与分析

任务4.2 加密自己的程序

这次试验，主要是需要我先把老师所给我的程序先放在td中执行一遍，了解各种反跟踪的原理。

首先执行到重新写入中断向量表的时候，为了防止被反跟踪，我选择修改ip来绕过这里。

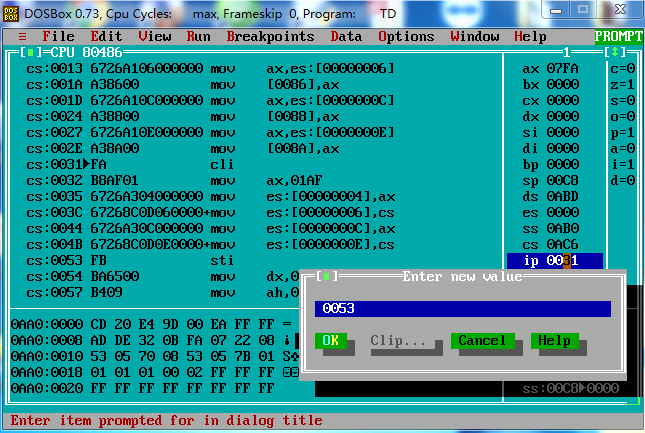


图4.10 修改ip跳过反跟踪中断

同样的方法帮助我绕过了后面的计时和堆栈检查，最后我理解了老师所给的示例程序。

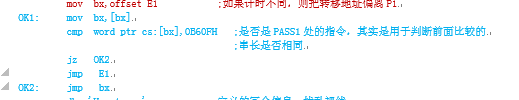


图4.11 老师给的程序片段

老师这里的0B60FH是PASS1的入口地址，很好地混淆了跟踪者的视线。

我开始对我的代码进行了修改：

首先加入了修改中断向量表的片段之后，进入td开始单步调试：

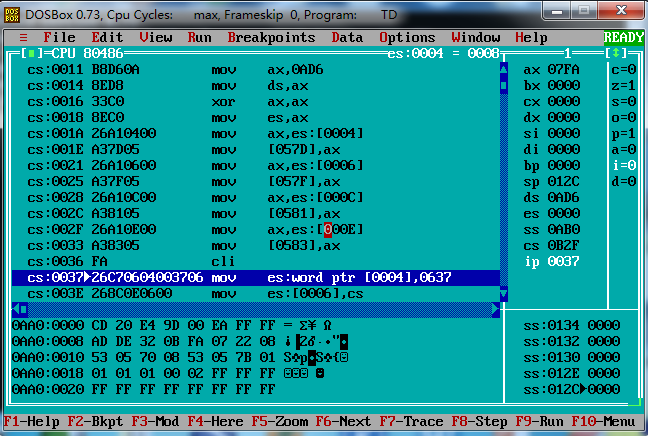


图4.12 验证自己的反跟踪中断

在重写中断向量表的那一步执行完之后，程序自动的退出，证明该反跟踪调试方法有效。

接下来给我的程序加入时间检验，进入td开始调试，绕开重新写入中断向量表的指令。

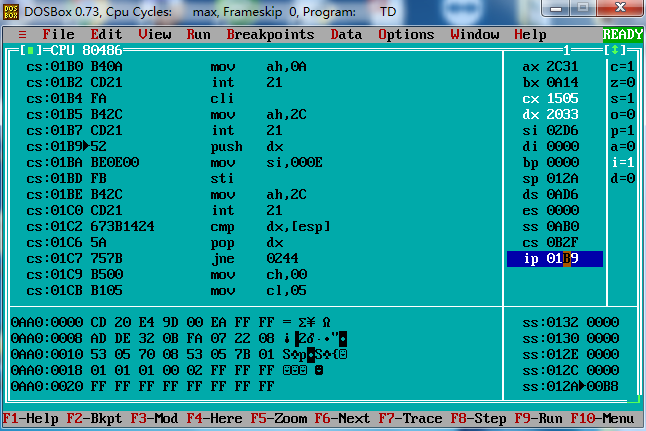


图4.13 绕开写入中断向量但是不绕开时间中断检查

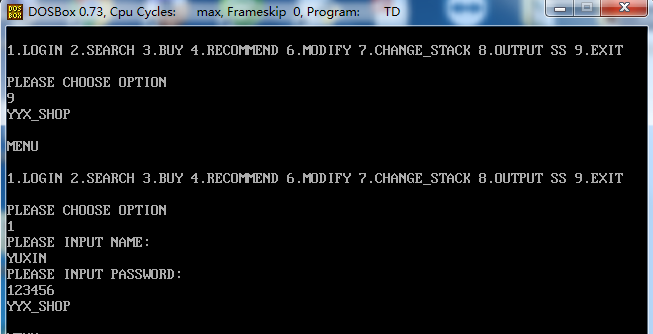


图4.14 未绕过时间中断

这里我可以发现如果执行了单步调试并且没有绕开时间中断检查，就直接返回了菜单并且没有任何提示信息显示。

如果直接按f9执行程序的话，我们可以发现程序正常使用：

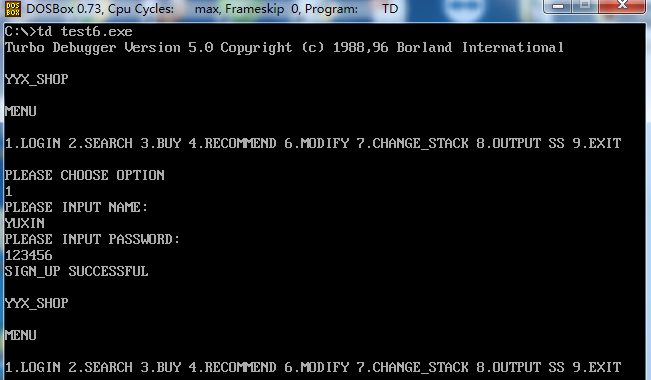


图4.15 直接运行程序

证明我的时间中断程序成功了。

下面证明堆栈段检查是否有效：

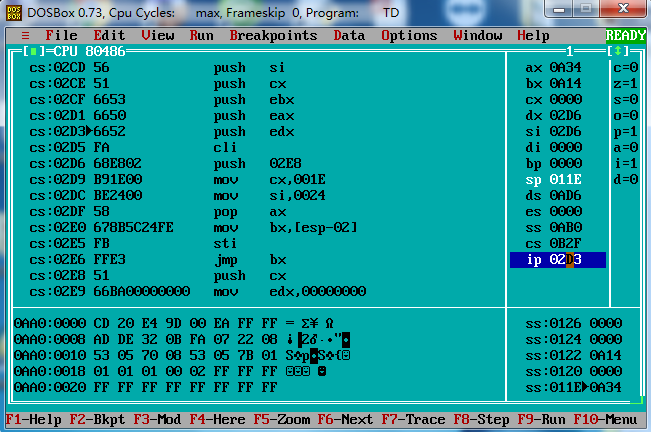


图4.16 绕过前两个反跟踪操作进入堆栈检查

这里我们先进入了功能4，先单步执行功能4：

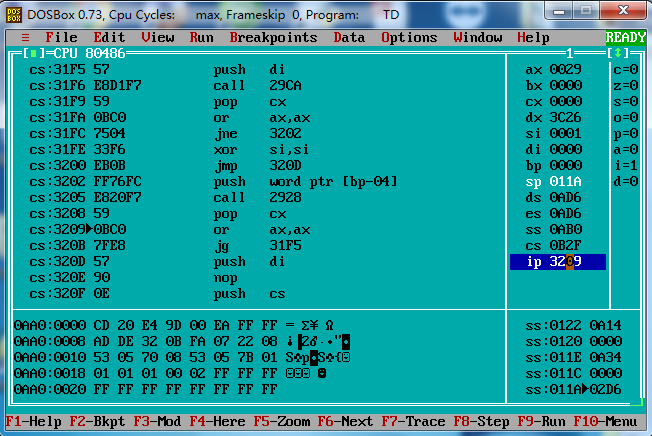


图4.17 单步进入堆栈检查

最后程序进入了上图中的死循环。

接下来我们按f9直接执行程序：

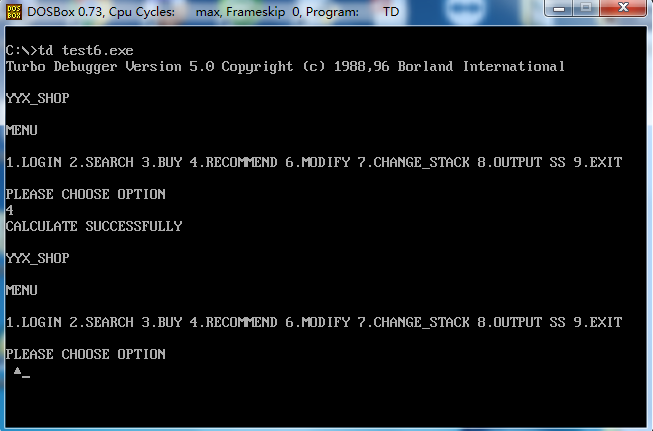


图4.18 直接执行程序

程序正常执行了功能四,接下来再进入功能2观察功能4是否成功计算了推荐度：

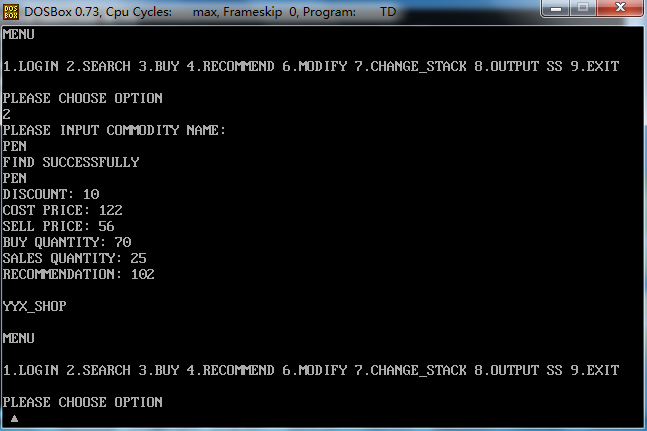


图4.19 功能四测试

发现推荐度被成功的计算了，证明我编写的堆栈段验证反跟踪成功。

在编写上面的程序中，我遇到了一个问题，一直不能解决，最后发现原来时间中断会破坏我的CX值，但是我后面只给CL重新赋值了，并没有考虑到CH，导致程序一直是死循环，最后我才想起来时间中断会破坏CX和DX寄存器，这里也是提醒了我保护寄存器的必要性。

任务4.5：破解同学的程序

我的队友：张鹏 U201814659

首先我先接收队友的exe程序，先放在ida中进行反汇编，观察队友的数据段：

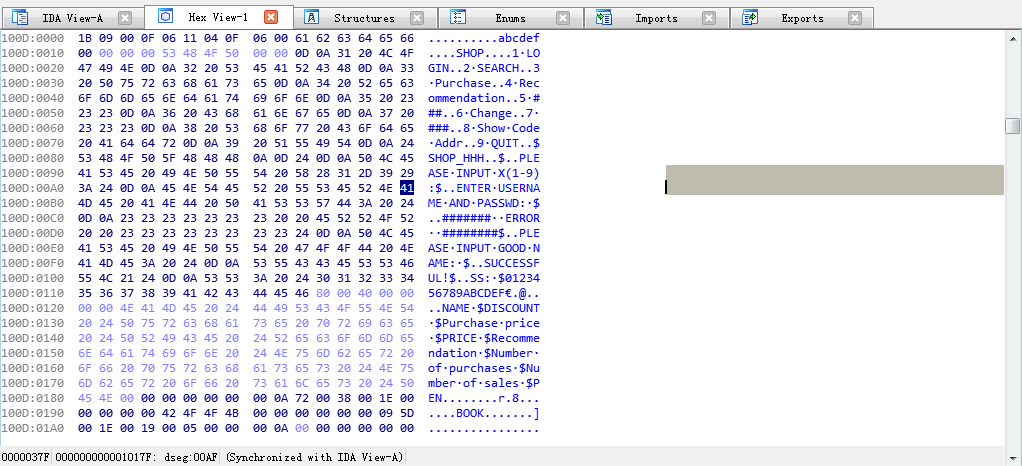


图4.20 利用反汇编后得到的数据段

并没有发现密码，因此密码被加密了。

在ida工具中添加队友的exe文件进行分析：

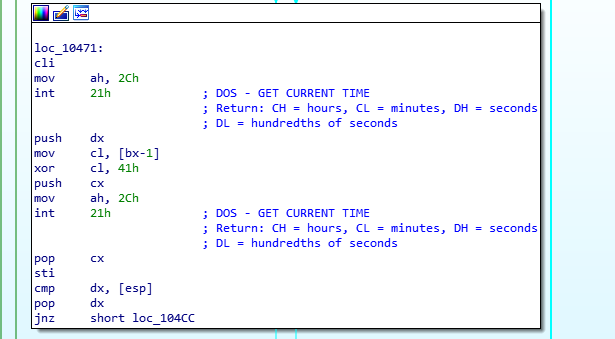


图4.21 在ida中找到的计时检查模块

在功能一中找到了队友的计时检查和堆栈检查的代码段。

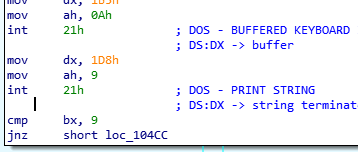


图4.22 分析出姓名长度

通过这一步我发现队友的姓名长度为9。

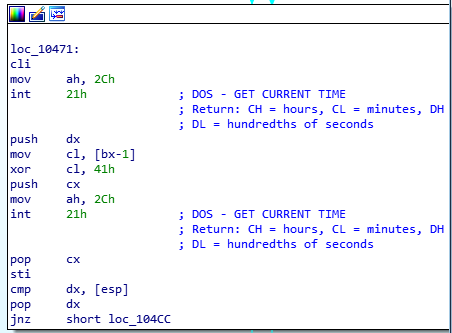


图4.23 分析出加密方式

发现时间中断在循环中，因此我锁定了验证输入的姓名的程序在时间中断中，这里我发现他的加密方式是XOR 41H，因此找到了他的密钥。

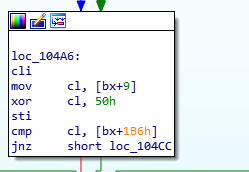


图4.24 分析出加密方式

接着在后续的代码段中发现了解密的过程，找到了队友的密码密钥为XOR 50H。

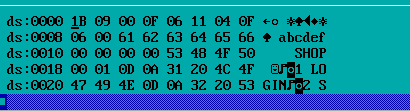


图4.25 使用td观察

整理好了程序的执行思路以及总结了什么时候应该跳过某写反跟踪手段，我开始使用td，跳过之前在ida中发现的反跟踪手段，单步调试。

在密码比对的代码段里，通过所访问的代码段的地址，我找到了存储姓名和密码的地址是ds：0000~000AH和000BH~0014H。

知道了存储地址，又知道了存储密钥，于是我手动破解出密码，把存储密码的每个bit XOR 41H，把存储密码的每个bit XOR 50H，最后破解出密码。

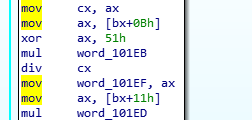


图4.26 破解出密码的位置7

破解出进货价密钥是51H

破译出的密码是：

姓名：ZHANGPENG 加密：XOR 41H

密码：123456 加密：XOR 50H

进货价：35,12 加密：XOR 51H

最后输入破解的密码：

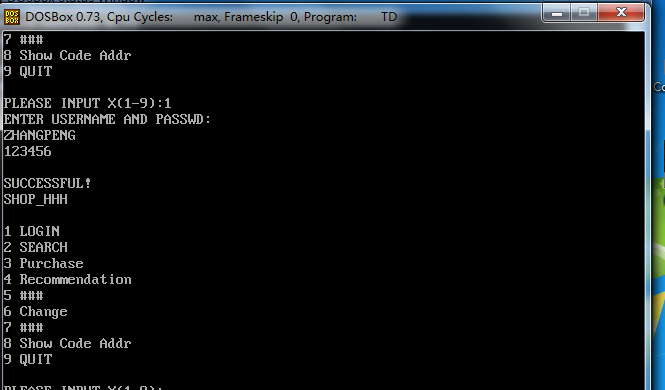


图4.27 破解成功

成功进入管理员页面，破解成功！

## 4.5 小结

### 4.5.1 主要收获

**查看中断矢量表：**

1. 打开TD之后，如何在数据区切换到中断向量表所在内存区域？

在td数据区中输入0000:0000就可以直接查看中断矢量表：

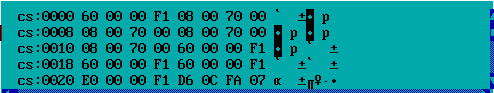


图4.28 查看中断矢量表

2. 如何计算某个中断入口在中断向量表内的偏移地址？

中断入口的偏移地址就是中断号乘4.

3. 程序中如何使用系统功能调用获取中断入口地址？

DOS可以通过调用系统35号功能来获得中断处理程序的入口地址。AL中存放中断码。获取到的中断入口地址的段地址放在ES中，偏移地址放在 BX中。

4. 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）

随着值的修改，中断程序也进入了修改后的中断子程序执行。

**中断服务程序：**

1.有哪两种方式进入原中断服务程序？(CALL和JMP)

这里有两种方法进入原中断程序，一种是CALL原中断程序，需要加上IRET来配对原来的CALL，但是如果使用JMP，就不需要再添加IRET作为配对了。

2. 为避免未调试好的中断服务程序接管时钟、键盘中断时使系统时间或键盘操作失灵，可以先用其他方法（比如：先不安装到中断矢量表中，仅当作子程序调用来调试；或者先安装到其他非硬件的中断号上，利用软中断来调试等。基本思路是：先把中断服务程序中与时钟、键盘操作没有直接相关的部分调试好，最后再把时钟、键盘操作相关的部分加上去）调试该中断服务程序，调试好后再安装成接管时钟或键盘中断的状态。

请给出你采用的“其他”调试方法的具体描述并实施一下。

我开始也是先把子程序写好，最后再加上去，但是可能还是会出现键盘锁死的情况，这个时候我就只能使用设置断点的方法进入终端内部进行调试。

3. 如何判断中断服务程序已经安装过了？

我这里新编写的中断程序添加了输入q才能够退出的，调用中断的时候，如果是之前的8号中断就不会需要我输入q来退出中断。

**内存操作：**

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？

我觉得主要是切换的时候要使用附加段寄存器，在使用寄存器寻址的时候注意段，不然直接使用默认段寄存器，会产生访问错误。

**任务4.2任务4.3**

1. 若密码是用明文存放在数据段中的，如何更快地获取密码？

如果密码是使用明文的话，直接访问内存段，找到数据段，如果不清楚位置，可以单步调试程序，找到检查密码的时候访问内存的位置，就可以确定密码位置，根据asc码表直接读取密码。

2. 若商品进货价是用明文存放在数据段中的，如何更快地获取进货价？（除了用调试工具在内存中去看，还可以将执行程序文件用二进制编辑工具打开，直接在文件里寻找所定义的商品信息）

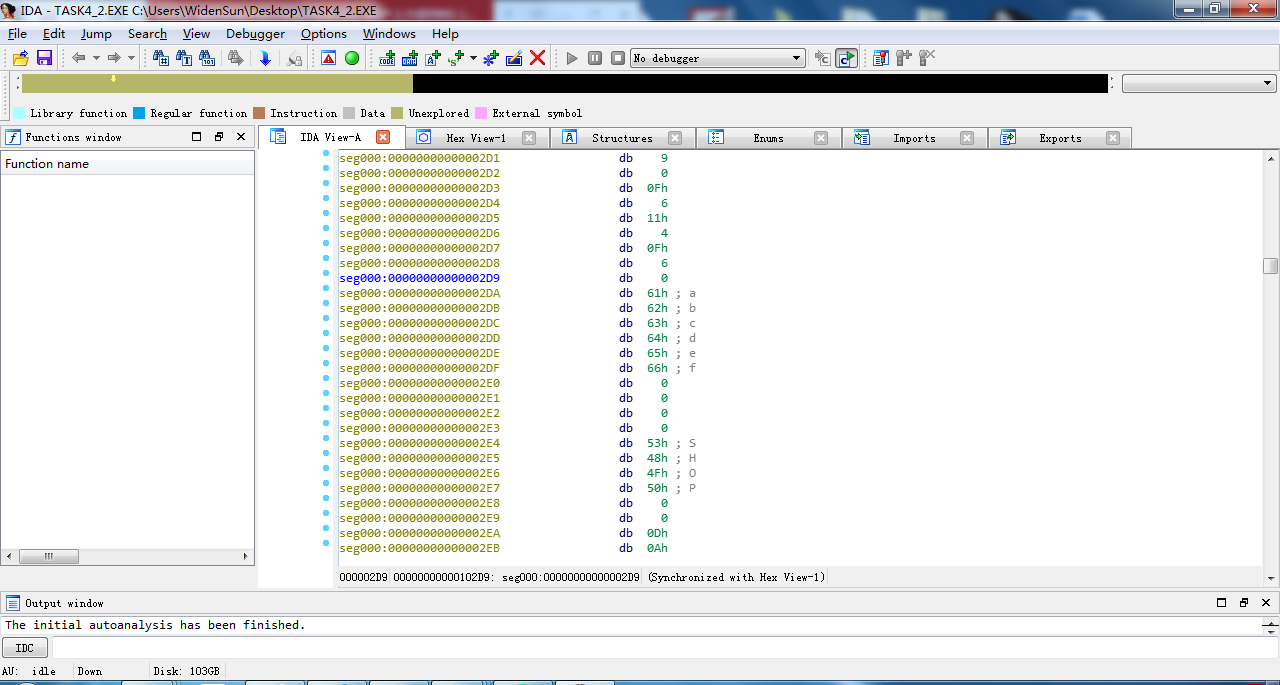


图4.29 二进制文件

我在IDA中把可执行文件转换成了二进制文件之后可以快速的找到密码abcdef。

3. 如何对密码实现快速的暴力破解？（可以编写程序，也可以描述一下实现的具体思路）

这次试验我没有做暴力破解，但是我的思路还是得先结合人工分析程序至少先知道密码是多少位的，再去根据26个字母和十个数字的自由组合去暴力破解密码。

4. 如何综合利用静态反汇编和动态反汇编的信息破解程序？（要使用反汇编工具得到汇编源程序，对其进行观察分析）

根据我上面的实验步骤，我这次使用的是IDA，不得不说这是一款功能强大的反汇编软件，能够快速对程序进行模块化分割，并且会在屏幕上显示出模块之间的联系。我这次首先是观察IDA中的程序，了解了程序大致的工作流程，再使用td来对程序进行单步调试，方便我知道在什么时候需要绕过反跟踪手段，以及什么时候会出现关键代码段，方便我对于密码存储位置进行定位。

5. 举例说明如何观察到程序中存在反跟踪的代码?举例说明如何应对反跟踪程序? （在记录分析里具体描述）

我主要是看到了STI这种中断标志就开始小心调试，此外，如果忽然出现异常的跳转，我也可以立刻察觉出异常。当然这些意识前提就是必须了解这个程序在干什么才能够清楚地知道什么时候应该绕过这些反跟踪程序，我是选择修改ip寄存器的值直接跳转，防止发生错误跳转。

7. 当存在修改中断矢量表的代码时，一般会先关掉中断（也即执行CLI指令）。如果不想因为关中断指令的出现让跟踪者容易判断出后续存在反跟踪代码，应如何设计修改中断矢量表的代码，达到不用关中断的目的？

可以先把1压入栈中，在popf来修改IF标志位，从而修改中断状态。

8. 是否可以通过修改AUTH的值来达到获取进货价的目的？是否可以通过观察该程序计算推荐度的过程来获取进货价？

可以，观察了队友的程序，发现只要我修改了auth就可以直接更改登录状态，可以在之后加上是否计算了推荐度来判断是否正常登陆，当然我个人认为还是不能够防住别人。

### 4.5.2 主要看法

任务4.1我个人觉得难点就在于如何理解中断，编写中断程序其实并不算是特别的复杂，当然中断调试过程中还是碰到了一些麻烦的，希望以后的编译器对于这种硬件终端也可以进行单步执行。

任务4.2和4.3，我感觉做起来十分的有趣，可以和队友互相交流程序的破解思路，也是第一次使用IDA这种强大的反汇编程序，软件我稍微捣鼓了半小时就摸清了功能，左边的任务框可以进入我想进的子程序，最上面的HEX VIEW还可以像td一样查看数据段和代码段，对于我分析队友的程序思路十分有帮助，并且IDA还支持把可执行文件转换成二进制文件的形式，方便我快速定位出我需要的一些数据。

这次破解同伴的程序花了接近50分钟，感觉要是流程清晰的话，分析起来并不算很难，此外，这里同伴并没有为难我，都是一些简单的加密方法，我理解起来也比较快，如果稍微出的复杂一点的加密算法或者编写程序的时候混乱一些，可能我的读程序步骤就会困难很多了。

# 5 WIN32程序设计

## 5.1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。

## 5.2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，类似图5.1所示。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（详见要求“2”的描述）。

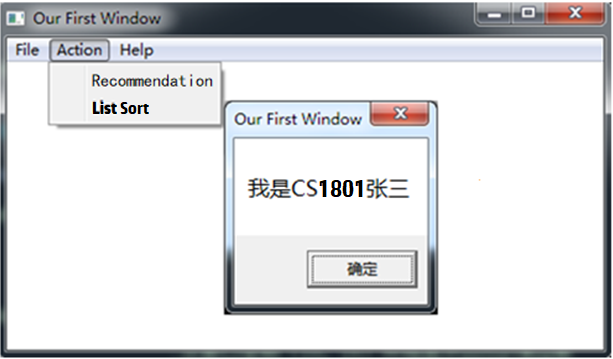


图5.1 菜单示例

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）。



图5.2 商品信息显示示意图

## 5.3 任务5.1实验过程

### 5.3.1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. 打开VS2013，按照老师的要求对平台进行配置

3. 熟悉各种api函数调用，懂得应该如何编写代码

4. List Sort流程描述：

（1）把BUF地址赋给ESI

（2）比较第一个和第二个商品的推荐度

（3）如果第一个比第二个小，则交换两个商品的所有信息

（4）重复5次，把最小的放在最后

（5）再重复3次，最后得到从高到低的排序

5. 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

6. 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还有排序的子函数。

### 5.3.2实验记录与分析

1. 实验环境条件：Visual Studio 2013，masm32。

2. 首先我熟悉了VS环境下的32位编程，发现这里编写的程序需要调用很多的库函数，这里的子程序需要在程序的最开始声明，不然会报错：



图5.3 声明函数

解压MASM32压缩包，在解压后的文件目录下，可以看到在BIN目录下有汇编（ML.exe/RC.exe）和连接（LINK.exe）等程序。这样我就可以直接在程序最开始的地方调用这些库。

3. 接下来我进入了数据段定义，这里大部分和之前的类似，但是这次实验要求我使用结构体来表示商品信息：

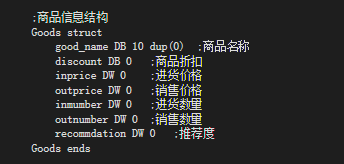


图5.4 商品信息结构

这里我声明了商品名称，商品折扣，进货价格，销售价格，进货数量，销售数量还有推荐度这几个变量，代表了所有的商品都要按照这个结构体的结构来存储信息，接下来，我需要给这几件商品信息进行赋初值：

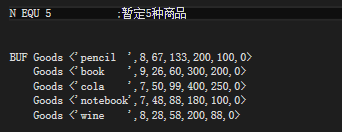


图5.5 商品信息定义

这里我决定在商店中存放五个商品信息。

接下来是主程序的编写和窗口主程序的编写，这个我主要参考了老师所给的实验范例和书上的演示，成功的输出了一个窗口：

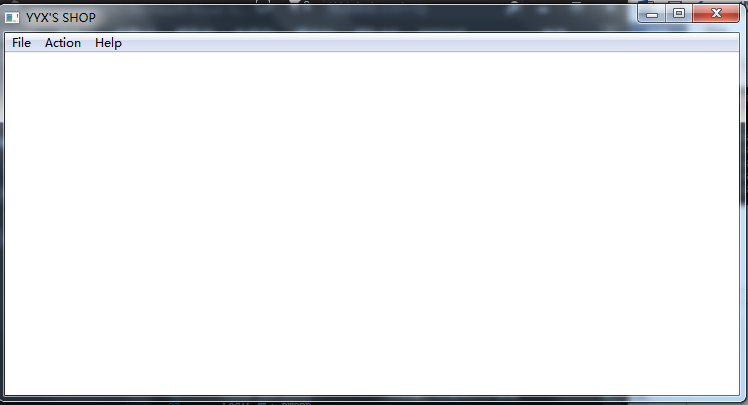


图5.6 首页

接下来我就需要在rc文件中添加每个目录下的选项，这里在file目录下，我添加了exit选项，点击就会立刻退出。在action目录下，我添加了三个选项，第一个是推荐度计算recommandation，第二个是list用来显示现在的商店商品信息，最后一个list sort用来对商品推荐度。

此外，我的rc文件编辑如下：

600 MENUEX MOVEABLE IMPURE LOADONCALL DISCARDABLE

BEGIN

POPUP"&File", , , 0

BEGIN

MENUITEM"&Exit", 1000

END

POPUP"&Action", , , 0

BEGIN

MENUITEM"&Recommendation", 1100

MENUITEM"&List", 1200

MENUITEM"&List Sort", 1300

END

POPUP"&Help", , , 0

BEGIN

MENUITEM "&About", 1900

END

END

接下来就是编写汇编代码实现商品信息显示。

注意到，在win32编程环境下，不能够使用21号中断，所以这里需要使用textout函数来进行信息输出。

接下来把自己之前编写的程序进行迁移到win32环境下，注意到这里的寄存器操作大部分都要修改成32位的，最后编写自己的基于冒泡排序的排序函数源程序片段如下：

CHANGE PROC

PUSHA

MOV ECX,4

LEA ESI,BUF

MOV EDX,0

MOV EAX,0

MOV EBX,0

SORT\_LOP1:

MOV AX,WORD PTR [ESI+19]

MOV BX,WORD PTR [ESI+40]

CMP AX,BX

JB CHANGE\_TWO

BACK:

ADD ESI,21

XOR AX,AX

XOR BX,BX

LOOP SORT\_LOP1

POPA

RET

CHANGE\_TWO:

PUSHA

MOV ECX,5

CHANGE\_LOP:

MOV EAX,0

MOV EBX,0

MOV EAX,DWORD PTR [ESI]

MOV EBX,DWORD PTR [ESI+21]

MOV DWORD PTR [ESI],EBX

MOV DWORD PTR [ESI+21],EAX

ADD ESI,4

LOOP CHANGE\_LOP

POPA

JMP BACK

CHANGE ENDP

这里在交换的过程中，我选择的是每4位4位一交换，最后在屏幕上进行输出测试：

（1）直接显示list：



图5.7 显示list

（2）计算推荐度之后再显示list：



图5.8 计算推荐度后显示list

（3）对推荐度进行排序后再输出list：



图5.9 排序后显示list

这里我们可以看到，程序确实是按照推荐度从高到低显示商品信息的，说明我们的排序函数成功工作了。

（4）个人信息窗口显示：

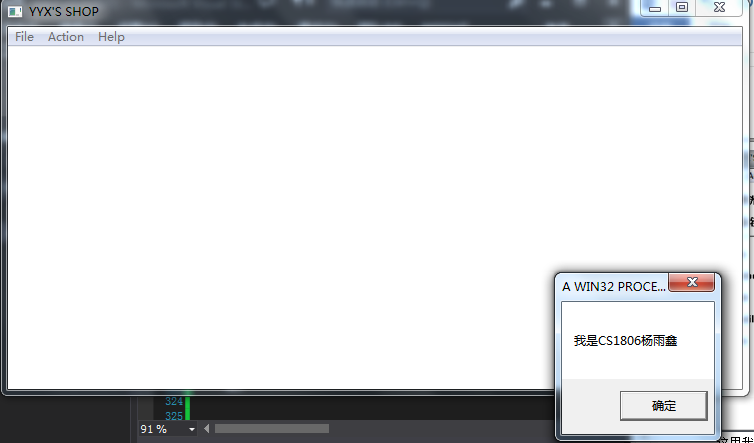


图5.10 个人信息显示

（5）退出测试：

点击file栏目的exit选项，程序成功退出。

到此，任务所需要完成的任务全部完成。

报错总结：

刚开始使用vs编写程序的时候，我遇到了一个报错困扰了我很久，报错显示我写入错误，后来我上网搜了好久，最开始别人说是堆栈段满了，后来改了堆栈段大小还是不行，最后发现是INVOKE textout会破坏我的寄存器，最后加了保护就好了。

## 5.4 小结

### 5.4.1 主要收获

问题描述和解答：

1. 观察32/64位下调试工具与16位TD的异同。

在32位下的调试观察到的寄存器都是32位的：

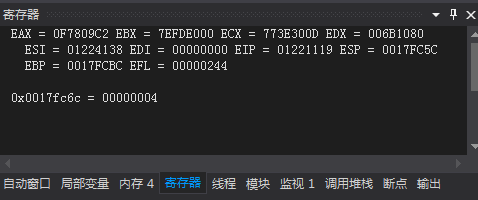


图5.11 观察的寄存器信息

并且这里把内存段分成了四个区间，可以让我们很方便的就观察到堆栈段数据段的信息。

2. 观察Invoke语句翻译成机器码后的特点，记录参数压栈顺序。单步跟踪到调用系统API函数的位置，观察相关代码的特点。

在INVOKE语句处设置断点，观察反汇编语句：

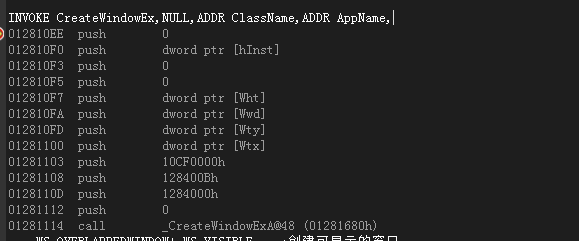


图5.12 观察反汇编语句

发现INVOKE被翻译成了这么长的一大段指令。发现这里的操作基本上是把函数的参数压入栈中，接着调用一个api函数。

3. 关于VS编程，你有什么发现？

在使用VS的过程中，我发现VS里面需要设置的东西是真的多，稍微一个不注意，可能就会产生意想不到的报错，使用起来更像ide，把很多种功能都整合到这一个平台里面，是一个很大的成功。

### 5.4.2 主要看法

Visual Studio是一款功能强大的集成开发环境，功能很多，对于编程者的环境配置能力要求也比较高。

这次试验的难点我觉得还是api函数的调用十分的不熟悉，不清楚api函数的具体工作原理，很容易发生写入内存错误，我在编写程序的过程中，碰到了很多次内存写入报错，但其实最后都是寄存器没有保护或者是程序操作溢出了，感觉这个报错方式还是不能够具有很高的区分度，最好能够在中断的地方标明是进入了死循环还是其他的情况之类的，既然已经是这么庞大的平台了，不如做的再更加细致。

# 参考文献

[1] 王元珍等.80X86汇编语言程序设计.华中科技大学出版社.2012:1-356

# 附录1

.386

DATA SEGMENT USE16

SNAME DB 'YYX\_SHOP',0,0AH,0DH,'$' ;网店名称，用0结束

BNAME DB 'YUXIN',0 ;老板姓名

BPASS DB '123456',0,0 ,0 ;密码

AUTH DB 0 ;当前登录状态，0表示顾客状态，1表示登陆成功，2表示退出程序

GOOD DB 0 ;当前浏览商品名称或地址（自行确定）

N EQU 30 ;商品总数

GA1 DB 'PEN', 7 DUP(0) ,10 ;商品名称及折扣

DW 35,56,70,25,?,? ; 进货价格，销售价格，进货数量，销售数量，推荐度还未计算

GA2 DB 'BOOK', 6 DUP(0) ,9 ;商品名称及折扣

DW 12,30,25,5,?,? ;推荐度还未计算

GAN DB N-2 DUP( 'TempValue' ,0,8,15,0,20,0,30,0,2,0,?,?);

TEMP DB 20 ;用来存储输入的指令

DB 0

DB 10 DUP(0)

OUTPUT\_CS DB 'CS:$'

TAB DB '0123456789ABCDEF'

COUNT1 EQU 5 ;用来验证老板姓名

COUNT2 EQU 6 ;用来验证密码

COUNT3 EQU 3 ;用来验证第一个商品名

COUNT4 EQU 4 ;用来验证第二个商品名

INNAME DB 20 ;代表最多输入20个字母

DB 0 ;用来记录输入的字母数量

DB 20 DUP(0) ;建立一个数组来存储输入的姓名

INPASSWORD DB 10 ;最多输入10个数字

DB 0 ;用来记录输入的数字数量

DB 10 DUP(0) ;建立一个数组用来存储输入的密码

INCOMMODITY DB 10 ;用来存储输入的商品名称

DB 0

DB 10 DUP(0)

TIP1 DB 0AH,0DH,'NAME ERROR',0AH,0DH,'$' ;提示信息

TIP2 DB 0AH,0DH,'PLEASE INPUT NAME:',0AH,0DH,'$'

TIP3 DB 0AH,0DH,'PLEASE INPUT PASSWORD:',0AH,0DH,'$'

TIP4 DB 0AH,0DH,'SING\_UP SUCCESSFUL',0AH,0DH,'$'

TIP5 DB 0AH,0DH,'PASSWORD ERROR',0AH,0DH,'$'

TIP6 DB 0AH,0DH,'PLEASE INPUT COMMODITY NAME:',0AH,0DH,'$'

TIP7 DB 0AH,0DH,'NOT FIND,TRY AGAIN',0AH,0DH,'$'

TIP8 DB 0AH,0DH,'FIND SUCCESSFULLY',0AH,0DH,'$'

TIP9 DB 0AH,0DH,'BUY SUCCESSFULLY',0AH,0DH,'$'

TIP10 DB 0AH,0DH,'PLEASE CHOOSE OPTION',0AH,0DH,'$'

TIP11 DB 0AH,0DH,'NO GOOD IN VIEW',0AH,0DH,'$'

TIP12 DB 0AH,0DH,'CALCULATE SUCCESSFULLY',0AH,0DH,'$'

MENU DB 0AH,0DH,'MENU',0AH,0DH,0AH,0DH,'1.LOGIN 2.SEARCH 3.BUY 4.RECOMMEND 8.OUTPUT CS 9.EXIT',0AH,0DH,'$'

DATA ENDS

STACK SEGMENT USE16 STACK

DB 300 DUP(0)

STACK ENDS

CODE SEGMENT USE16

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

START:

MOV AX,DATA

MOV DS,AX

MOV ES,AX

MOV AL,BYTE PTR GA1+10

MOV BL,BYTE PTR GA1+13

MOV BH,10

IMUL BL

IDIV BH

MOV [GA1+21],AL

MOV AL,BYTE PTR GA2+10

MOV BL,BYTE PTR GA2+13

MOV BH,10

IMUL BL

IDIV BH

MOV [GA2+21],AL

;计算出实际销售价格

LEA DX,SNAME ;把DX寄存器放在商店名的地址处

MOV AH,9 ;9号中断，输出商店名

INT 21H

JMP INSTRUCT

INSTRUCT:

LEA DX,MENU

MOV AH,9

INT 21H

LEA DX,TIP10

MOV AH,9

INT 21H

LEA DX,TEMP

MOV AH,10

INT 21H

MOV AL,TEMP+2

CMP AL,'0'

JZ EXIT

CMP AL,'1'

JZ P1

CMP AL,'2'

JZ P2

CMP AL,'3'

JZ P3

CMP AL,'4'

JZ P4

CMP AL,'8'

JZ P8

CMP AL,'9'

JZ EXIT

P1:

CALL FUN1 ;进入第一个功能函数

CMP AUTH,0 ;判断是否为顾客状态

JZ INSTRUCT

CMP AUTH,1 ;判断是否为登录状态

JZ INSTRUCT

CMP AUTH,2 ;判断是否为输入错误

JZ P1

CMP AUTH,3 ;判断是否为退出程序

JZ EXIT

P2: ;寻找商品

LEA DX,TIP6 ;提示输入商品名称

MOV AH,9

INT 21H

LEA DX,INCOMMODITY ;读取输入的商品名称

MOV AH,10

INT 21H

CALL FUN2

JMP INSTRUCT

P3:

CMP GOOD,0 ;观察GOOD是否为0

JNZ NOTZERO1

LEA DX,TIP11

MOV AH,9

INT 21H

JZ INSTRUCT

NOTZERO1:

CALL FUN3

JMP INSTRUCT

P4:

CALL FUN4

JMP INSTRUCT

P8:

CALL FUN8

JMP INSTRUCT

EXIT:

MOV AH,4CH

INT 21H

FUN1 PROC ;登陆验证

LEA DX,TIP2 ;提示用户输入姓名

MOV AH,9

INT 21H ;显示提示信息

LEA DX,INNAME ;接受姓名

MOV AH,10

INT 21H

CMP INNAME+2,0DH ;比较输入的是否为回车

JZ GO0

CMP INNAME+2,'q' ;比较是否为退出字符

JZ GO3

LEA DX,TIP3 ;提示用户输入密码

MOV AH,9

INT 21H

LEA DX,INPASSWORD

MOV AH,10

INT 21H

MOV CL,COUNT1 ;验证姓名

LEA SI,BNAME

LEA DI,INNAME+2

CMP CL,INNAME+1 ;长度不相等，姓名肯定错误

JNZ WRONG1

FLAG1:

MOV BL,[DI]

CMP BYTE PTR [SI],BL

JNZ WRONG1

INC SI

INC DI

DEC CX

CMP CX,0 ;比较计数器是否为0

JNZ FLAG1

MOV CL,COUNT2 ;验证密码

LEA SI,BPASS

LEA DI,INPASSWORD+2

CMP CL,INPASSWORD+1 ;长度不相同，密码肯定错误

JNZ WRONG2

FLAG2:

MOV BL,[DI]

CMP BYTE PTR [SI],BL

JNZ WRONG2

INC SI

INC DI

DEC CX

CMP CX,0

JNZ FLAG2

JMP LOGIN\_SUCCESSFUL ;成功登入

WRONG1: ;姓名错误

MOV AUTH,2

LEA DX,TIP1 ;输出提示信息

MOV AH,9

INT 21H

RET

WRONG2: ;密码错误

MOV AUTH,2

LEA DX,TIP5

MOV AH,9

INT 21H

RET

LOGIN\_SUCCESSFUL:

MOV AUTH,1

LEA DX,TIP4

MOV AH,9

INT 21H

RET

GO0:

MOV AUTH,0 ;代表为顾客状态

RET

GO1:

MOV AUTH,1

RET

GO2:

MOV AUTH,2 ;输入错误

RET

GO3:

MOV AUTH,3 ;退出

RET

FUN1 ENDP ;结束子程序

FUN2 PROC ;查找商品，把商品序号返回到GOOD中存储

LEA SI,GA1

LEA DI,INCOMMODITY+2

MOV CL,COUNT3

CMP CL,INCOMMODITY+1 ;判断字符数是否相同

JNZ NEXTGOOD

FLAG3:

MOV BL,[DI] ;移动到第一个字符

MOV BH,[SI]

CMP BL,BH

JNZ NEXTGOOD ;如果两个字符不同，则搜索下一个商品

INC SI

INC DI

DEC CX

CMP CX,0

JNZ FLAG3 ;如果CX不为0，则继续循环

LEA DX,TIP8

MOV AH,9

INT 21H

MOV GOOD,1 ;1号商品即为所求

RET

NEXTGOOD:

LEA SI,GA2

LEA DI,INCOMMODITY+2

MOV CL,COUNT4

CMP CL,INCOMMODITY+1 ;判断字符数是否相同

JNZ FAILED

FLAG4:

MOV BL,[DI] ;移动到第一个字符

MOV BH,[SI]

CMP BL,BH

JNZ FAILED ;如果两个字符不同，则搜索下一个商品

INC SI

INC DI

DEC CX

CMP CX,0

JNZ FLAG4 ;如果CX不为0，则继续循环

LEA DX,TIP8

MOV AH,9

INT 21H

MOV GOOD,2 ;2号商品即为所求

RET

FAILED:

LEA DX,TIP7

MOV AH,9

INT 21H

MOV GOOD,0

RET

FUN2 ENDP

FUN3 PROC

CMP GOOD,1

JZ GOOD1

CMP GOOD,2

JZ GOOD2

RET

GOOD1:

MOV AX,WORD PTR GA1+17 ;销售数量

MOV BX,WORD PTR GA1+15 ;进货数量

SUB BX,AX

CMP BX,0

JNZ PURCHASE1

RET

PURCHASE1:

INC WORD PTR GA1+17

LEA DX,TIP9

MOV AH,9

INT 21H

RET

GOOD2:

MOV AX,WORD PTR GA2+17 ;销售数量

MOV BX,WORD PTR GA2+15 ;进货数量

SUB BX,AX

CMP BX,0

JNZ PURCHASE2

RET

PURCHASE2:

INC WORD PTR GA2+17

LEA DX,TIP9

MOV AH,9

INT 21H

RET

FUN3 ENDP

FUN4 PROC

PUSH EDX

MOV EDX,0

CMP GOOD,1 ;是否为第一个产品

JNZ NEXT

LEA SI,GA1

XOR EBX,EBX

XOR EAX,EAX

MOV AX,WORD PTR GA1+17 ;销售数量

MOV BX,WORD PTR GA1+15 ;进货数量

SAL EBX,1

SAL EAX,7 ;已售数量\*128

IDIV EBX

MOV CX,AX ;存储结果

XOR EBX,EBX

XOR EAX,EAX

MOV AX,WORD PTR GA1+11 ;进货价

MOV BX,WORD PTR GA1+21 ;销售价

MOV EDX,0

SAL EAX,7 ;进货价\*128

IDIV EBX

ADD AX,CX ;把两个结果相加

MOV WORD PTR [GA1+19],AX

LEA DX,TIP12

MOV AH,9

INT 21H

POP EDX

RET

NEXT:

LEA SI,GA2

XOR EBX,EBX

XOR EAX,EAX

MOV AX,WORD PTR GA2+17

MOV BX,WORD PTR GA2+15

SAL EBX,1

SAL EAX,7

IDIV EBX

MOV CX,AX

XOR EBX,EBX

XOR EAX,EAX

MOV AX,WORD PTR GA2+11

MOV BX,WORD PTR GA2+21

MOV EDX,0

SAL EAX,7

IDIV EBX

ADD AX,CX

MOV WORD PTR [GA2+19],AX

LEA DX,TIP12

MOV AH,9

INT 21H

POP EDX

RET

FUN4 ENDP

FUN8 PROC

F8:

MOV AH, 15

INT 10H

MOV AH, 0

INT 10H

LEA DX, OUTPUT\_CS

MOV AH, 9

INT 21H

LEA BX, TAB ;16进制字符转换表放入BX

MOV AX, CS

AND AX, 0F000H

SHR AX, 12

XLAT TAB

MOV DL, AL

MOV AH, 2 ;输出CS的最高位

INT 21H

MOV AX, CS

AND AX, 0F00H

SHR AX, 8

XLAT TAB

MOV DL, AL

MOV AH, 2 ;输出CS的次高位

INT 21H

MOV AX, CS

AND AX, 00F0H

SHR AX, 4

XLAT TAB

MOV DL, AL

MOV AH, 2 ;输出CS的次低位

INT 21H

MOV AX, CS

AND AX, 000FH

XLAT TAB

MOV DL, AL

MOV AH, 2 ;输出CS的最低位

INT 21H

MOV AH, 1

INT 21H

JMP INSTRUCT

FUN8 ENDP

CODE ENDS

END START