

**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201802班**

**学 号： U201814734**

**姓 名： 顾时禹**

**指导教师： 班鹏新**

**实验时段： 2020年3月19日~5月7日**

**实验地点： 天津市 电传一号院**

**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： 纸质版再签名

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

[目录 1](#_Toc40083142)

[课程总体说明 1](#_Toc40083143)

[0.1 课程目标 1](#_Toc40083144)

[0.2 成绩构成 1](#_Toc40083145)

[0.3 实验任务的总体描述 2](#_Toc40083146)

[1 编程基础 4](#_Toc40083147)

[1.1 实验目的与要求 4](#_Toc40083148)

[1.2 实验内容 4](#_Toc40083149)

[1.3 任务1.1实验过程 7](#_Toc40083150)

[1.3.1 实验方法说明 7](#_Toc40083151)

[1.3.2 实验记录与分析 7](#_Toc40083152)

[1.4 任务1.2实验过程 11](#_Toc40083153)

[1.4.1 实验方法说明 11](#_Toc40083154)

[1.4.2 实验记录与分析 11](#_Toc40083155)

[1.5 任务1.3实验过程 14](#_Toc40083156)

[1.5.1 实验方法说明 14](#_Toc40083157)

[1.5.2 实验记录与分析 14](#_Toc40083158)

[1.6 任务1.4实验过程 17](#_Toc40083159)

[1.6.1 设计思想及储存单元分配 17](#_Toc40083160)

[1.6.2 源程序 20](#_Toc40083161)

[1.6.3 实验步骤 27](#_Toc40083162)

[1.6.4 实验记录与分析 27](#_Toc40083163)

[1.7 实验小结 35](#_Toc40083164)

[1.7.1 主要收获 35](#_Toc40083165)

[1.7.2 总结反思 36](#_Toc40083166)

[1.7.3 课后问题 36](#_Toc40083167)

[2 程序优化 39](#_Toc40083274)

[2.1 实验目的与要求 39](#_Toc40083275)

[2.2 实验内容 39](#_Toc40083276)

[2.3 任务2.1实验过程 40](#_Toc40083277)

[2.3.1 实验方法说明 40](#_Toc40083278)

[2.3.2 实验记录与分析 40](#_Toc40083279)

[2.4 任务2.1实验过程 45](#_Toc40083280)

[2.4.1 实验方法说明 45](#_Toc40083281)

[2.4.2 实验记录与分析 45](#_Toc40083282)

[2.5 实验小结 53](#_Toc40083283)

[2.5.1 主要收获 53](#_Toc40083284)

[2.5.2 课后问题 53](#_Toc40083285)

[3 模块化程序设计 54](#_Toc40083313)

[3.1 实验目的与要求 54](#_Toc40083314)

[3.2 实验内容 54](#_Toc40083315)

[3.3 任务3.1实验过程 55](#_Toc40083316)

[3.3.1 设计思想及存储单元分配 55](#_Toc40083317)

[3.3.2 源程序 60](#_Toc40083318)

[3.3.3 实验步骤 62](#_Toc40083319)

[3.3.4 实验记录与分析 62](#_Toc40083320)

[3.4 任务3.2实验过程 67](#_Toc40083321)

[3.4.1 实验方法说明 67](#_Toc40083322)

[3.4.2 实验记录与分析 67](#_Toc40083323)

[3.5 实验小结 74](#_Toc40083324)

[3.5.1 主要收获 74](#_Toc40083325)

[3.5.2 课后问题 74](#_Toc40083326)

[4 中断与反跟踪 78](#_Toc40083327)

[4.1 实验目的与要求 78](#_Toc40083328)

[4.2 实验内容 78](#_Toc40083329)

[4.3 任务4.1 实验过程 80](#_Toc40083330)

[4.3.1 设计思想及存储单元分配 80](#_Toc40083331)

[4.3.2 源程序 82](#_Toc40083332)

[4.3.3 实验步骤 84](#_Toc40083333)

[4.3.4 实验记录与分析 84](#_Toc40083334)

[4.4 任务4.2实验过程 87](#_Toc40083335)

[4.4.1 实验方法说明 87](#_Toc40083336)

[4.4.2 实验记录与分析 88](#_Toc40083337)

[4.5 任务4.3实验过程 92](#_Toc40083338)

[4.5.1 实验方法说明 92](#_Toc40083339)

[4.5.2 实验记录与分析 92](#_Toc40083340)

[4.6 实验小结 94](#_Toc40083341)

[4.6.1 主要收获 94](#_Toc40083342)

[4.6.2 课后问题 95](#_Toc40083343)

[5 WIN32/64编程 98](#_Toc40083396)

[5.1 实验目的与要求 98](#_Toc40083397)

[5.2 实验内容 98](#_Toc40083398)

[5.3 任务5.1实验过程 99](#_Toc40083399)

[5.3.1 实验方法说明 99](#_Toc40083400)

[5.3.2 实验记录与分析 100](#_Toc40083401)

[5.4 实验小结 113](#_Toc40083402)

[5.4.1 主要收获 113](#_Toc40083403)

[5.4.2 课后问题 113](#_Toc40083404)

**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。

# 编程基础

## 实验目的与要求

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. 熟悉操作系统的常用功能调用，如常用的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。

## 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（不需要编写完整程序），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）

## 任务1.1实验过程

### 实验方法说明

1. 准备实验环境，安装DOSBOX，MASM60。学习了解相关软件和指令的基本功能和应用方法。

2. 在TD中输入进行1.4题中运算的代码，观察每条代码执行后AH、BH、SF、OF、CF、ZF中的内容值。与自己预测的值比较。

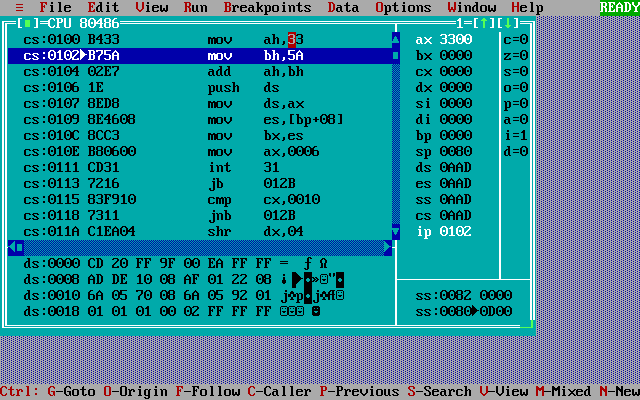
### 实验记录与分析

1. 分别对三组实验结果进行预测。预测结果如表1.1

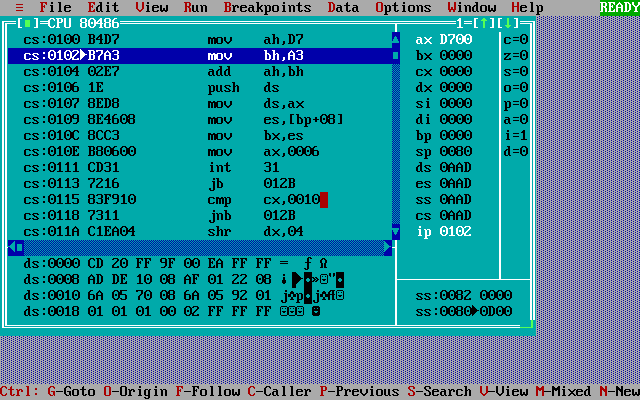
**表1.1 实验预测结果**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 组别 | AH | SF | OF | CF | ZF |
| 1 | 10001101B | 1 | 1 | 0 | 0 |
| 2 | 01111010B | 0 | 1 | 1 | 0 |
| 3 | 00001000B | 0 | 0 | 1 | 0 |

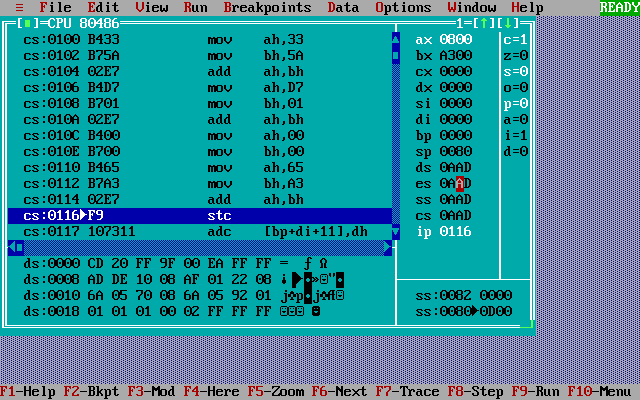
2. 逐条输入语句后运行结果如图1-3-1~1-3-3，和预期结果一致。



**图1.1 第一组运行结果**

****

**图1.2 第二组运行结果**

****

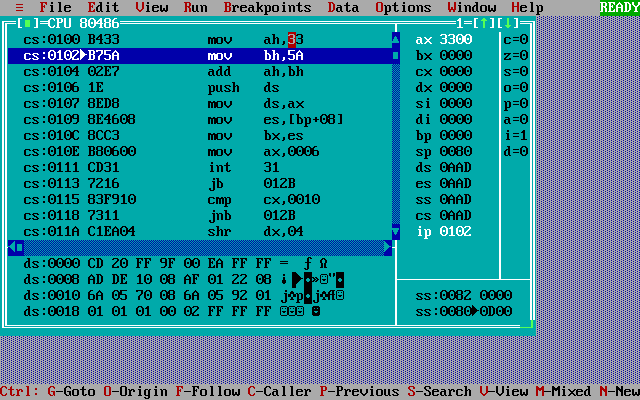
**图1.3 第三组运行结果**

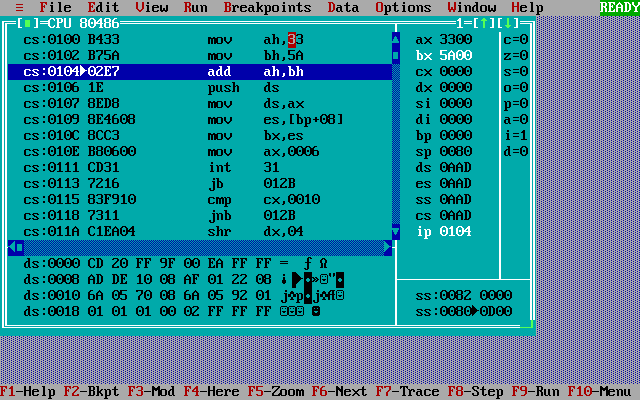
3. 在TD中选择语句后按下任意字母就可以直接输入指令，输入完毕指令后运行如果使用Run菜单会从ip=0000开始运行，但是修改的的指令不一定在0000位置，所以会引起程序崩溃。要使用F7和F8来运行代码。

4. 指令的两个参数之间要用,分割，并不是空格分隔。

5. 可以使用右键菜单点击new CS:IP来指定下一条运行的语句。

6. 在右侧的寄存器窗口可以实时监控每个寄存器的值，如图1-3-4~1-3-5所示。

**图1.4 运行第一条语句后**

**图1.5 运行第二条语句后**

## 任务1.2实验过程

### 实验方法说明

1. 按题2.3编写汇编程序。

2. 在TD中单步调试程序，观察数据段前40个字节的内容，记录并与设想的情况对比，观察是否一致

3. 运行到“MOV CX，10”和“INT 21H”之前，记录(BX),(BP), (SI), (DI)的内容，并观察他们在运行过程中的变化。

### 实验记录与分析

1. 题2.3的汇编源代码如下：

.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 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

2. 程序的功能为，把BUF1中的内容放到BUF2中，每字节+1后放到BUF3中，每字节+4后放入BUF4中。程序运行完毕DS段前40字节的内容应该为

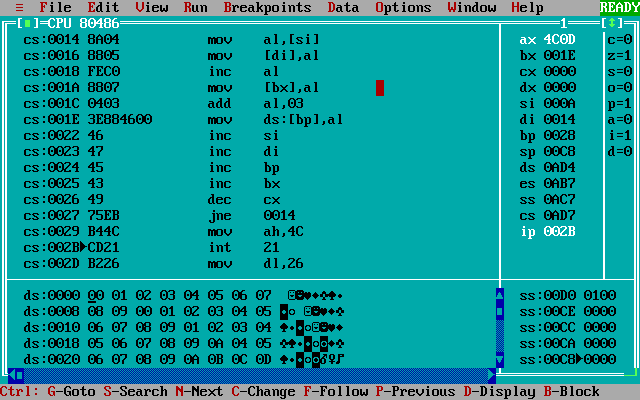
00 01 02 03 04 05 06 07 08 09

00 01 02 03 04 05 06 07 08 09

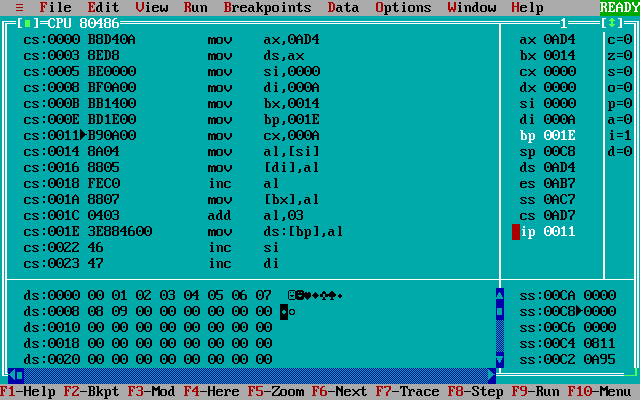
01 02 03 04 05 06 07 08 09 0A

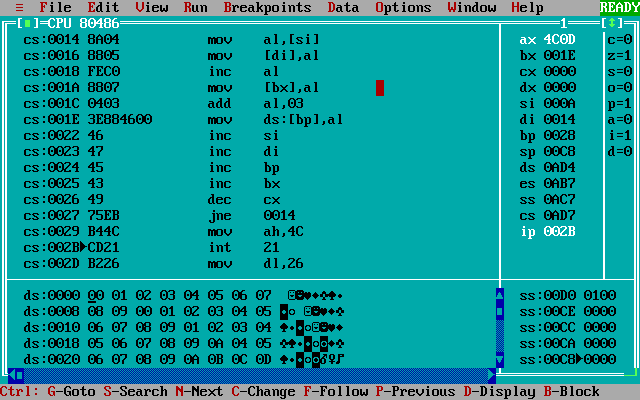
04 05 06 07 08 09 0A 0B 0C 0D

3. 程序运行完毕结果如图1.6，和预测的结果一致

**图1.6 程序运行结果**

4. 在MOV CX, 10指令之前，程序将BUF1,2,3,4的首字节在数据段中的偏移地址分别保存到SI,DI,BX,BP中，之后每次循环都将四个寄存器中的值加一，即为指向BUF1.2.3.4中的下一个字节。程序运行结束后四个寄存器的值应该比初始值大10。程序运行时寄存器的值如图1.7~1.8

**图1.7 MOV CX, 10前寄存器的值**

****

**图1.8 运行到INT 21H前寄存器的值**

## 任务1.3实验过程

### 实验方法说明

1. 修改2.3的程序，使用变址寻址方式访问存储单元，并使用32位存储器。

2. 在TD中单步调试程序，观察数据段前40个字节的内容，记录并与设想的情况对比，观察是否一致

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。重点观察汇编语句代码与机器码的联系。

4. 从不同的字节位置开始反汇编，观察结果的不同。理解EIP/IP的重要性。

### 实验记录与分析

1. 修改后的程序源代码如下：

.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, 0

LOPA: MOV AL, BUF1[ESI]

MOV BUF2[ESI], AL

INC AL

MOV BUF3[ESI], AL

ADD AL, 3

MOV BUF4[ESI], AL

INC ESI

CMP ESI, 10

JNZ LOPA

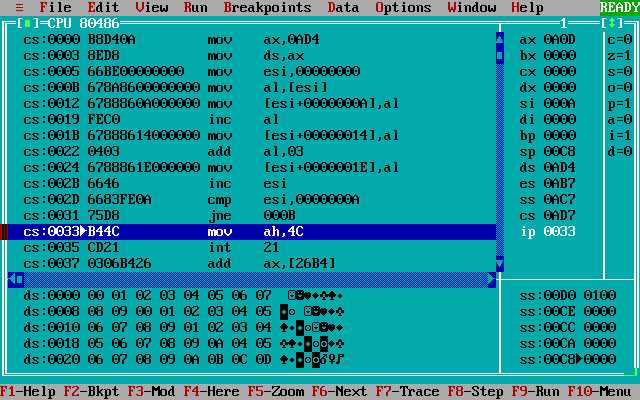
MOV AH, 4CH

INT 21H

CODE ENDS

END START

2. 实验结果和任务1.2并无区别，如图1.9所示

****

**图1.9 实验结果**

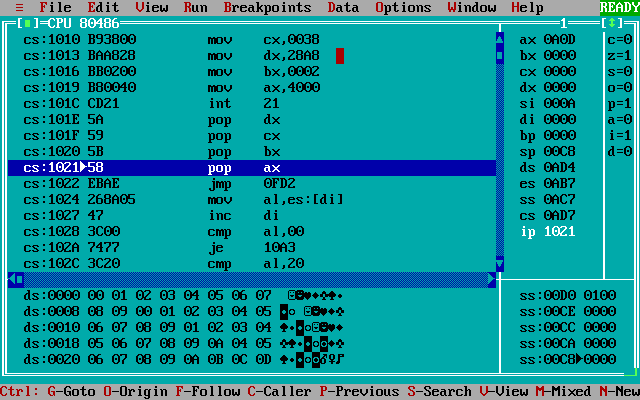
3. 本任务需要改写两个地方，一是使用32位寄存器，二是使用变址寻址。为满足要求，使用ESI存储偏移量，使用BUFX[ESI]来访问内存中的数据。

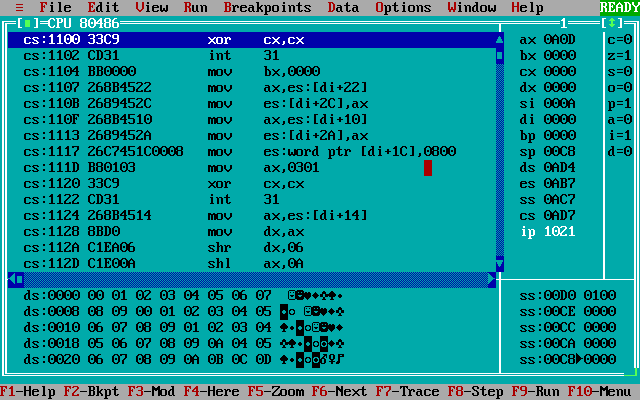
4. 通过对比反汇编代码和源代码可以得到以下结论：

DATA等标识符在汇编阶段不会被翻译为机器代码，而是会被汇编程序替换为一些偏移量，比如DATA被置换为0AD4，BUF1,2,3,4的首地址分别为0000,000A,0014,001E，和上一个任务的源代码比较，可以发现在使用32位寄存器时，机器代码的操作数也被调整为32位。

5. 通过观察生成的机器代码可以发现，同类指令的机器码有一部分共同部分，还有一部分表示了操作数，不同的寄存器也分别用不同的代码来表示

6. 如果从不同的位置开始反汇编，就会执行没有定义过的随机代码，而保证IP指针的正确保证了会执行正确的指令，如图1.10~1.11所示

**图1.10 从CS:1010位置反汇编**

**图1.11 从CS:1100位置反汇编**

## 任务1.4实验过程

### 设计思想及储存单元分配

（1）主界面

MENUT储存主界面输入命令提示字符串。

主界面部分先调用09号命令输出MENUT的内容。再调用1号命令读取一个字符，判断字符是否为1~9，是就跳转到对应功能，是0就退出程序，其他情况返回主界面。

（2）登录功能

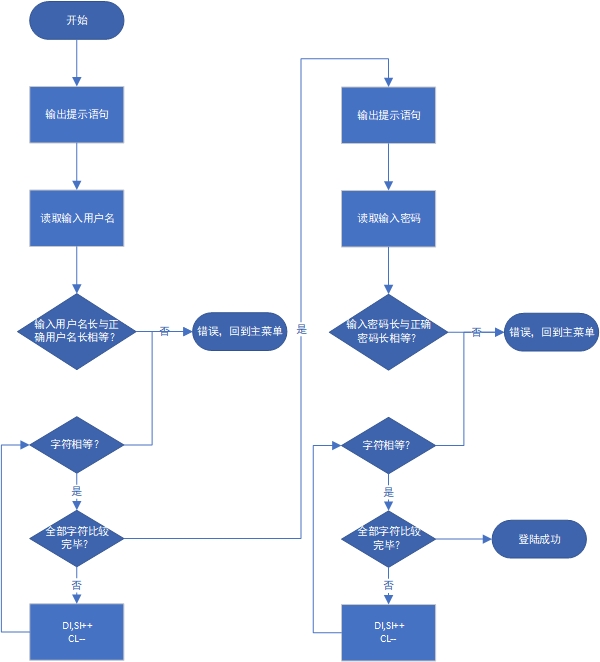
BNAME储存老板用户名，BPASS储存密码，INNAMET储存用户名输入提示信息，INNAME储存输入的用户名，WRONGNAMET储存用户名错误提示信息。INPWDT储存密码属入提示信息，INPWD储存输入的密码。WRONGPWDT储存密码错误提示信息，LOGINST储存登录成功信息。

CL用于储存输入字符串长度和正确用户名、正确密码的长度。DI和SI分别储存当前比较的输入字符和正确字符的地址。

FUNC1标号内为功能初始化部分，CHECK\_USER标号为用户名比较部分，CHECK\_PWD为密码比较部分，WRONG\_NAME，WRONG\_PWD分别为用户名和密码错误时运行的语句块，LOGIN\_S为登录成功时运行的语句。

先调用09号功能输出提示字符串，再调用10号功能读取一个字符串输入，将输入的字符串长度存入CL，与正确字符串的长度比较，不相等则直接跳转到错误时的代码语句，相等则开始比较字符串，DI初始为输入字符串第一个字节地址，SI初始为正确字符串第一个字节地址，如果两个字节不相等也跳转到错误时的代码语句，相等就将DI,SI加一，CL减一，CL为0时跳转到下一步。

登录部分流程图如图1.12：



**图1.12 登录部分流程图**

（2）查找商品功能

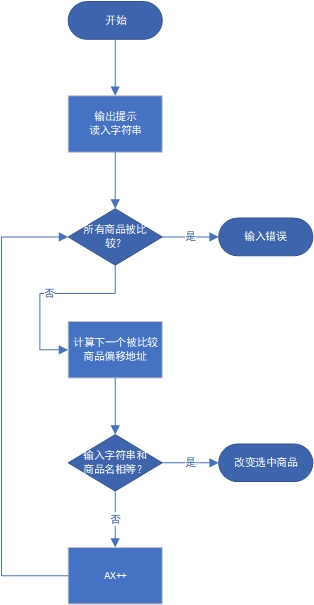
INGOODT储存商品名输入提示信息，WRONGGOODT储存错误商品名提示信息，INGOOD储存输入的商品名。CHANGEGOODT储存商品改变提示信息，GOOD储存当前选中的商品的偏移地址。

AX储存已查询的商品总数，CX用于计算当前比较的商品偏移地址时的中间储存器，CL储存输入字符串长，DI储存输入字符串第一个字节位置，SI储存当前比较的商品名的第一个字节。

FUNC2语句块为功能初始化语句，CHECK\_GOOD部分语句计算下一个比较的商品的偏移地址，CHECK\_NAME比较输入语句和当前比较的商品名是否一样，CHANGE\_GOOD将选中商品改变成输入的商品，WRONG\_GOOD是输入商品名不存在时运行的语句块。

先类似于登录部分输出提示信息和读入输入商品名，使用GA1+21\*AX计算下一个要比较的商品的偏移地址，并将偏移地址置入SI，将输入字符串第一个字符的偏移地址置入DI，输入字符串长置入CL，逐字符比较两字符串，如果有不同的字符，跳转到下一个商品，如果所有商品都被比较完，则跳转到输入错误部分。如果字符全部相同，查看被比较的商品名的下一个字符是不是0，是则将当前被比较的商品的偏移地址保存到GOOD中，不是则比较下一个商品。

查找商品部分的流程图如图1.13：

****

**图1.13 查找商品流程图**

（3）卖出商品功能

SELLGOODT储存商品已卖出提示信息，NOGOODT储存商品已卖空提示信息。

BX储存当前选中商品偏移地址，AX储存商品进货数，DX储存商品已售数。

FUNC3为功能初始化部分以及售空部分代码，SELL\_GOOD为成功售出部分代码。

比较当前商品是否还有余货，有则将当前商品售出数加一，输出卖出成功信息并调用功能四，如果没有则输出无货信息。

（4）计算推荐度功能

UPDATEPRIORT储存计算推荐度提示信息。

BP储存计算过的商品数，CX用作计算中间数存储器，SI储存当前商品偏移地址，BX和EBX为除数变量，AX和EAX为被除数变量，DI和EDI 用于存储计算常数，如128、2等

FUNC4为初始化部分，COMPUTE\_PRIOR为计算推荐度部分。

类似于商品查找部分计算下一个计算的商品的偏移地址。为了避免结果为0，先做乘法再做除法。由于售出数，进货价都为16位数，128为8位数，乘积可以为24位数，24位数做被除数尚6可以为24位数，所以除数要用EBX储存，为了保证高位的正确性，被除数和常数也使用32位寄存器。为了保证数据读入的正确性，从存储器中读取数据时使用16位寄存器。

（5）输出CS内容功能

使用BP作为计数器，CL用于循环左移，BX用于保存CS内容，DL为输出字符。

FUNC8为初始化代码，IS\_NUM为判断为数字时执行的代码，SHOW\_OVER输出换行。

现将CS内容移入BX中，在每次对BX内容循环左移4位，将内容储存到DX中，再将DX中内容与00001111做与运算，相当于取BX内容的高四位。然后再判断是数字还是字母，是数字则把DL内容加30H，字母则加37H，然后输出。

### 源程序

.386

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

DATA SEGMENT USE16

MENUT DB 'Please Enter a Number(1~9):$'

INNAMET DB 'User Name:$'

WRONGNAMET DB 'Invalid User Name$'

INNAME DB 10

DB ?

DB 10 DUP(0)

INPWDT DB 'Password:$'

WRONGPWDT DB 'Invalid Password$'

INPWD DB 10

DB ?

DB 10 DUP(0)

LOGINST DB 'Login in$'

INGOODT DB 'Good Name:$'

WRONGGOODT DB 'Invalid Good Name$'

INGOOD DB 10

DB ?

DB 10 DUP(0)

CHANGEGOODT DB 'Good Selected$'

NOGOODT DB 'No Goods$'

SELLGOODT DB 'Good Sold$'

NOTSELEVTEDT DB 'No Goods Selected$'

UPDATEPRIOT DB 'Priority Updated$'

BNAME DB 'GU SHIYU', 0 ;老板姓名

BPASS DB '153426', 0 ;密码

AUTH DB 0 ;当前登录状态，0表示顾客状态

GOOD DW 0 ;当前浏览商品地址

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, ? ;进货价，销售价，进货总数，已售数量，推荐度

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, ?, ?)

;其他商品默认值

CRLF DB 0DH, 0AH, '$' ;回车

A DB 'A$' ;调试用

B DB 'B$'

DATA ENDS

CODE SEGMENT USE16

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

START: MOV AX, DATA ;程序开始

MOV DS, AX

MENU: LEA DX, MENUT ;主菜单

MOV AH, 09H

INT 21H

MOV AH, 01H

INT 21H

MOV AH, 09H

LEA DX, CRLF

INT 21H

CMP AL, '1'

JE FUNC1

CMP AL, '2'

JE FUNC2

CMP AL, '3'

JE FUNC3

CMP AL, '4'

JE FUNC4

CMP AL, '8'

JE FUNC8

CMP AL, '9'

JMP OVER

JMP MENU

FUNC1: LEA DX, INNAMET ;功能1

MOV AH, 09H

INT 21H

LEA DX, INNAME

MOV AH, 0AH

INT 21H

LEA DX, CRLF

MOV AH, 09H

INT 21H

LEA DI, INNAME+1

MOV CL, [DI]

CMP CL, 8

JNE WRONG\_NAME

LEA DI, INNAME+2

LEA SI, BNAME

CHECK\_USER: ;逐字节比较用户名

MOV AL, [DI]

CMP AL, [SI]

JNE WRONG\_NAME

INC DI

INC SI

DEC CL

JNZ CHECK\_USER

LEA DX, INPWDT

MOV AH, 09H

INT 21H

LEA DX, INPWD

MOV AH, 0AH

INT 21H

LEA DX, CRLF

MOV AH, 09H

INT 21H

LEA DI, INPWD+1

MOV CL, [DI]

CMP CL, 6

JNE WRONG\_PWD

LEA DI, INPWD+2

LEA SI, BPASS

CHECK\_PWD: ;逐字节比较密码

MOV AL, [DI]

CMP AL, [SI]

JNE WRONG\_PWD

INC DI

INC SI

DEC CL

JNZ CHECK\_PWD

JMP LOGIN\_S

WRONG\_NAME:

LEA DX, WRONGNAMET

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

JMP MENU

WRONG\_PWD:

LEA DX, WRONGPWDT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

JMP MENU

LOGIN\_S: ;登陆成功

MOV AUTH, 1

LEA DX, LOGINST

MOV AH, 09H

INT 21H

MOV AH, 09H

LEA DX, CRLF

INT 21H

JMP MENU

FUNC2: LEA DX, INGOODT ;功能2

MOV AH, 09H

INT 21H

LEA DX, INGOOD

MOV AH, 0AH

INT 21H

LEA DX, CRLF

MOV AH, 09H

INT 21H

MOV AX, -1

CHECK\_GOOD: ;选择下一个比较商品

INC AX

CMP AX, 30

JE WRONG\_GOOD

MOV CX, AX

IMUL CX, 21

ADD CX, OFFSET GA1

MOV SI, CX

LEA DI, INGOOD+1

MOV CL, [DI]

LEA DI, INGOOD+2

CHECK\_NAME: ;比较商品名

MOV BL, [DI]

CMP BL, [SI]

JNE CHECK\_GOOD

INC DI

INC SI

DEC CL

JNZ CHECK\_NAME

MOV BL, [SI]

CMP BL, 0

JE CHANGE\_GOOD

JMP CHECK\_GOOD

WRONG\_GOOD:

LEA DX, WRONGGOODT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

JMP MENU

CHANGE\_GOOD: ;更改选中商品

MOV CX, AX

IMUL CX, 21

ADD CX, OFFSET GA1

MOV GOOD, CX

LEA DX, CHANGEGOODT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

JMP MENU

FUNC3: MOV BX, GOOD ;功能3

CMP BX, 0

JE NO\_SELECTED

MOV AX, 15[BX]

MOV DX, 17[BX]

CMP AX, DX

JNE SELL\_GOOD

LEA DX, NOGOODT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

JMP MENU

SELL\_GOOD:

INC DX

MOV 17[BX], DX

LEA DX, SELLGOODT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

JMP FUNC4

NO\_SELECTED:

LEA DX, NOTSELEVTEDT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

JMP MENU

FUNC4: MOV BP, -1 ;功能4

LEA DX, UPDATEPRIOT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

COMPUTE\_PRIO:

INC BP

CMP BP, 30

JE MENU

LEA SI, GA1

MOV CX, BP

IMUL CX, 21

ADD SI, CX

MOV BX, SI

ADD BX, 13

MOV AX, [BX-2]

MOV BX, [BX]

MOVSX EAX, AX

MOVSX EBX, BX

MOV EDI, 128

IMUL EDI

IDIV EBX

MOV ECX, EAX

MOV BX, SI

ADD BX, 15

MOV AX, [BX+2]

MOV BX, [BX]

MOVSX EAX, AX

MOVSX EBX, BX

IMUL EDI

MOV EDI, 2

IDIV EDI

MOV EDX, 0

IDIV EBX

ADD EAX, ECX

MOV BX, SI

ADD BX, 19

MOV [BX], AX

JMP COMPUTE\_PRIO

FUNC8: MOV BP, 4 ;功能8

MOV BX, CS

GET\_LETTER:

MOV CL, 4;

ROL BX, CL

MOV DL, BL

AND DL, 0FH

CMP DL, 0AH

JB IS\_NUM

ADD DL, 07H

; IS\_NUM: ADD DL, 30H

MOV AH, 02H

INT 21H

DEC BP

JZ SHOW\_OVER

JMP GET\_LETTER

SHOW\_OVER:

LEA DX, CRLF

MOV AH, 09H

INT 21H

JMP MENU

OVER: MOV AH, 4CH

INT 21H

CODE ENDS

END START

### 实验步骤

（1）准备实验环境DOSBOX以及TD

（2）编译连接源程序

（3）输入正确的用户名和密码，验证能否成功登陆

（4）输入错误的用户名和密码，验证能否提示输入错误

（5）使用TD查看AUTH是否被更改

（6）验证功能二是否会更改GOOD

（7）验证功能三能否正常卖出

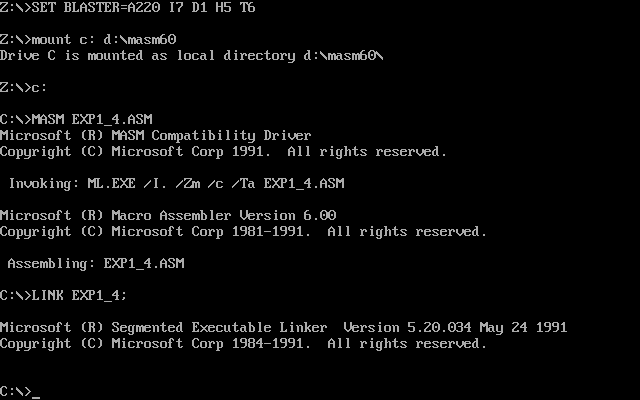
（8）验证功能三在售空时能否正常提示

（9）验证功能四推荐度计算的正确性

（10）验证功能八正确性

### 实验记录与分析

1. 编写源程序，编译，链接源程序。



**图1.14 编译链接源程序**

2. 打开TD调试源程序。

3. 验证主界面显示功能

主界面输出提示信息。



**图1.15 主界面显示**

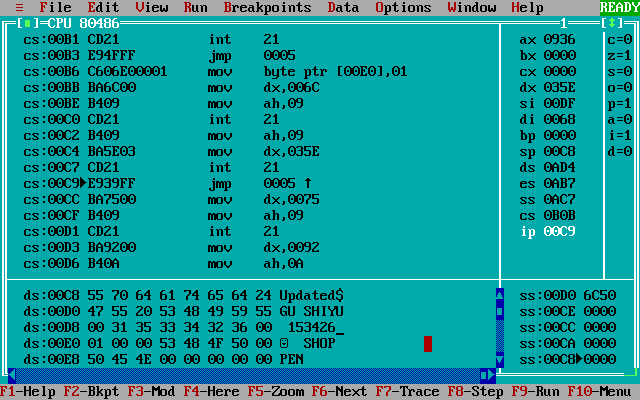
4. 验证能否成功登陆

成功登陆提示Login in



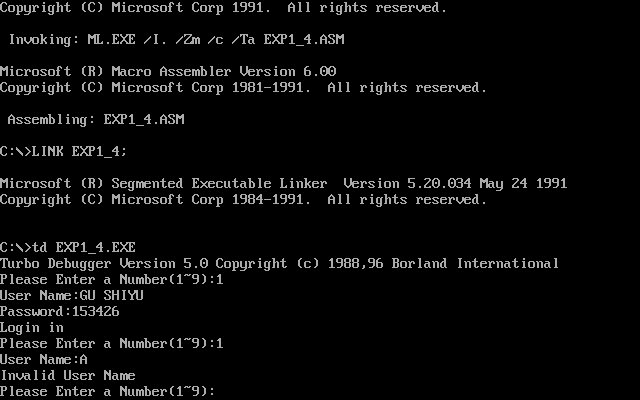
**图1.16 提示登陆成功**

AUTH值变为1。



**图1.17 DS:00E0即AUTH值为1**

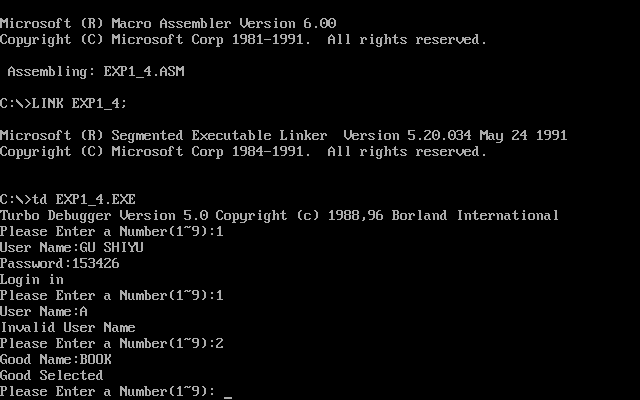
5. 验证登陆失败情况



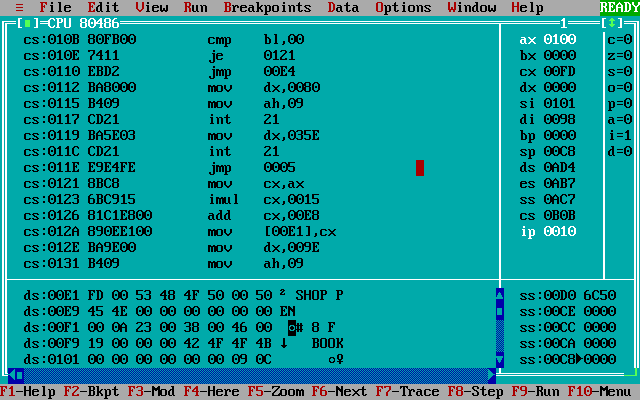
**图1.18 登陆失败**

6. 验证功能二在输入正确时的结果

输出Good Selected表示商品已选择。



**图1.19 商品已选择**



**图1.20 DS:00E1即GOOD位置内容为BOOK商品首地址**

7. 验证商品名无效的结果

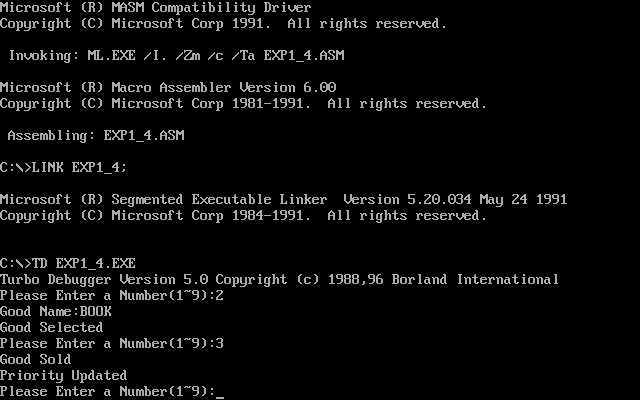
输出Invalid Name表示商品名错误。



**图1.21 商品名无效**

8. 验证功能三有存货的结果

输出Good Sold表明有存货且商品已售出



**图1.22 商品卖出**

9. 验证功能三无存货的结果

输出No Goods表明无存货



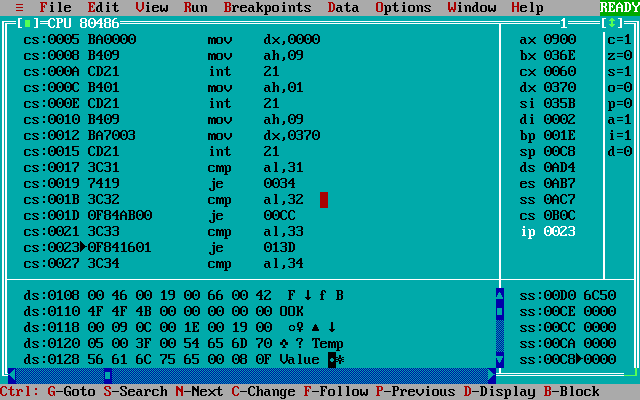
**图1.23 没有存货输出错误**

10. 验证功能四

输出Priority Update表示推荐度已更新

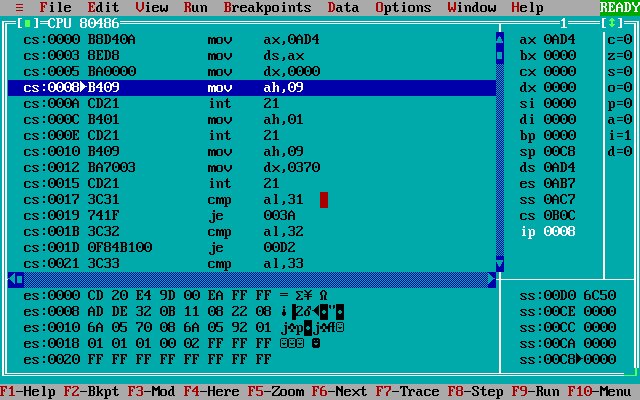


**图1.24 计算完毕**

****

**图1.25 商品1的推荐度为66H（第1行第6列）商品2的推荐度为3FH（第4行第3列）**

11. 验证功能八



**图1.26 CS内容为0B0C**



**图1.27 输出CS内容为0B0C**

11. 注意事项

（1） 系统调用功能9不会自动输出换行，要单独储存CRLF内容即0DH，0AH手动输出。

（2） 使用间接寻址和变址寻址方式时只能使用BX、BP、DI、SI四个寄存器，不能使用其他的寄存器

（3） 可以在反汇编代码中查看对应变量在数据段中定义的位置，来监视变量，也可以使用TD的监视窗口方便的监视变量。

## 实验小结

### 主要收获

在本次实验中，我主要学习运用了DOSBOX，MASM，TD等软件的使用方法和环境的搭建，基本语句的编写与执行，不同寻址方式的使用，机器码和汇编指令的转换，以及简单的程序设计。

在任务1.1中，主要练习了汇编开发环境的使用，主要是TD 的使用，TD提供了方便的寄存器监视，主存储器监视，代码反汇编等功能，还可以方便的直接修改代码运行，修改内存数据调试，也提供了断点调试，监视窗口等功能，熟练的使用TD可以极大的方便我们调试和编写程序。

在任务1.2中，通过观察程序的运行，我了解了汇编中存储器中的数据是如何被程序使用的，以及数据在存储器和寄存器中交换的过程。在实验中可以观察到，数据段一开始是未定义的默认值，在将我们定义的DATA段基址移动到DS中后，数据段的位置也发生了变化，而数据存储的位置也就是段基址+偏移量。

在任务1.3中，通过反汇编可以看出，变量在被翻译成机器代码时，直接使用偏移量常量替换了代码名，这有助于我们理解变量和标号的本质都是一个地址，同时通过观察和1.2反汇编代码的不同，以及比较类似语句的机器码，我初步归纳了汇编指令转化为机器码的规律。

最后任务1.4是一个简单的商店系统的程序设计，通过这个程序设计，我第一次尝试编写汇编语言程序，从中感受到了与C语言编程的相同点和不同点。相同点是编程的中心思想是类似的，都是注重过程，将要执行的功能一步一步翻译为汇编语句即可，但是相比于C语言，汇编语言几乎没有库函数，输入输出等功能固定执行也依赖于没有学习过的DOS指令，其次汇编语言的寄存器数量有限，需要思考如何分配，避免丢失需要的数据。指令的参数也有很多限制条件，需要记忆他们的规则。所有的内存管理都要手动进行，很容易出现数据越界的情况，这些都需要在编写程序时格外小心。

### 总结反思

本次实验中我存在很多问题，应当在之后的学习中加以改进。

首先是编程环境的使用，DOSBOX本身的兼容性较差，在使用时有很多不便，不如经常会丢失鼠标的输入。一开始我遇到这种问题只能重新开启DOSBOX，非常浪费时间，效率很低，经过这次的实验，我基本上熟悉了TD的各种快捷键和DOSBOX的各种指令，可以脱离鼠标进行调试，下次实验的调试效率应该会有所提高。

其次是自己汇编语言的熟练度还不高，还保留着很多C语言编程的思想，比如1.4的功能2需要遍历每一个商品，在计算商品地址的同时还要维护计数器，还要比较商品名，在寄存器的分配上没有足够的经验，经常发生丢失数据或者访问了其他内存区域的情况。

最后汇编的数字计算更加的底层，用到的命令也更复杂，需要考虑很多溢出的问题，需要以后特别注意。

### 课后问题

|  |  |  |  |
| --- | --- | --- | --- |
| 任务编号 | 序号 | 学生尝试的操作 | 结果 |
| 任务1.1 | 1 | 打开TD之后，如何在代码区输入一条指令，并且执行这条指令？ | 将光标移到代码区，直接输入指令，按F8运行 |
| 2 | 如何在代码区输入若干条指令后，再从输入的第一条指令开始执行？ | 可以通过new ip菜单设置下一条运行语句的位置 |
| 3 | 在输入一条指令中的数据时，若以16进制输入，需要注意什么问题？ | 在立即数后加上H |
| 4 | 输入指令MOV AH, -128并执行 查看AH寄存器的内容 | D800 |
| 5 | 执行一条指令后，如何查看寄存器的值（含32位寄存器）？如何修改寄存器的值？ | 在右侧有寄存器监视窗口，右键寄存器，选择change |
| 6 | 执行一条指令后，如何查看标志寄存器的值？ | 右侧有标志寄存器监视窗口 |
| 7 | 将A、B视为有符号数，运算后得到的结果、所有标志位信息，与将A、B视为无符号数，再次做同样运算后得到的结果、所有标志位信息进行比较，有何不同？ | 有符号数sf位会有变化，无符号数不会影响sf位 |
| 8 | 总结加减法对标志寄存器的影响？ | 会影响AF,OF,PF,SF,ZF,CF标志位 |
| 任务1.2和任务1.3 | 1 | 如何将一可执行程序调入TD，并查看代码区？理解机器码与汇编指令之间的对应关系？ | 在dos运行td 程序名指令。汇编指令按照一定规则被翻译为机器代码 |
| 2 | 如何设置断点并运行到断点？ | 使用breakpoint菜单的toggle功能 |
| 3 | 如何使程序运行到光标的当前点？（假设活动光标在代码区，指向某一条指令） | 使用F4或者run菜单中的run to cursor |
| 4 | 如何单步执行一条指令？（多种方法） | 使用run菜单的trace in或者使用F7 |
| 5 | 在数据区找到数据段的方法（包括找到某个变量在哪里）？思考：是否可以用这一方法查看代码段甚至整个程序？（至少有三种方法：(1)goto，DS：偏移地址，(2)goto, 直接输入：段寄存器的值：偏移地址；(3)直接在数据区用光标移动查找） | 使用goto菜单 |
| 6 | 修改某个指定的存储单元的值，如任务1.2中的BUF2 | 使用goto找到单元位置后直接输入数据 |
| 7 | 如何查看堆栈？ | 右下角有堆栈监视区 |
| 8 | 如何汇编一个汇编源程序并链接产生可执行代码？ | 使用MASM命令和LINK命令 |
| 9 | 如何读懂源程序在汇编过程中产生的错误？ | 在最开始有行号和错误代码提醒，可以在行号附近查找错误 |
| 10 | 使用DOS系统功能调用前应注意保存AX寄存器的值 |  |
| 11 | 查看BUF2等变量在TD中的表示形式并总结（包括在定义的地方、使用的地方等处的表现形式） | 定义的地方在机器代码中不存在，使用的地方被替换为了变量的偏移地址 |
| 12 | 查看寄存器间接寻址、变址寻址的汇编源程序经汇编、链接后在TD中的表示形式？总结源程序的指令和TD中的指令差异。 | 反汇编指令中不会包含标号名，会替换为他们的偏移地址 |
| 13 | 单步执行循环体2遍，查看数据段的变化 | 每个变量的前两个字节被改变了 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 序号 | 学生尝试的操作 | 结果 |
| 任务1.4 | 1 | 如何让9号功能调用显示的信息放在自己希望的位置？ | 在需要的地方使用9号功能 |
| 2 | 如果在9号功能调用时，带显示字符串的结尾没有“$”结束符会怎样？ | 会打印出乱码 |
| 3 | 如果在9号功能调用前，未对DS赋值，或者未对DX给予正确的值，结果会如何？ | 会打印出乱码 |
| 4 | 10号功能调用时，输入的字符数超过定义的数量时，它是如何处理的？ | 不允许输入 |
| 5 | 匹配姓名时，如何提高匹配速度？ | 一次比较多个字符 |
| 6 | 循环或转移时，是否有多种指令的组合方式实现？ | 可以使用不同的转移条件 |
| 7 | 注意观察转移指令机器码的编码方法，观察对应标号的偏移地址与该编码之间的关系。 | 编码中会包含转移指令的目的地址 |
| 8 | 检查循环程序，是否可能出现死循环的情况。 |  |
| 任务1.4 | 1 | 是否可以简化一下计算公式。 | 加号后面的部分可以改为\*64 |
| 2 | 单步观察计算过程，结果是否正确？ |  |
| 3 | 用TD直接修改数据区的数值，使得程序对应的每个分支都能得到验证。 |  |
| 4 | 选取特殊的值，观察计算结果溢出的情况。 | 溢出可能导致死循环 |

# 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。

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

（2）若无效或剩余数量为0，则提示错误，回到主菜单界面。

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。

**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）。

**注意：（1）由于本课程关注的是指令级别的影响，因此，请大家不要只考虑纯算法级别的优化。（2）在优化思想的描述中，首先要对任务2.1中可能需要优化的做法或程序片段进行分析说明，然后提出自己的优化思路。（3）对于优化问题，其实验结果的记录与分析中，必须有相同条件下优化前后的数据对比。**

## 任务2.1实验过程

### 实验方法说明

1. 准备实验环境，DOSBOX，TD等。

2. 修改任务1.4源程序，加入计时功能。

3. 通过改变M和N的值来观察计时功能的效果。

4. 通过改变DOSBOX的CPU时钟周期来观察不同硬件环境对程序运行效率的影响。

### 实验记录与分析

1. 首先在源程序中加入计时功能，更改部分的代码如下：

FUNC3: PUSH 0

MOV BX, GOOD ;功能3

CMP BX, 0

JE NO\_SELECTED

MOV AX, 15[BX]

MOV DX, 17[BX]

CMP AX, DX

JNE SELL\_GOOD

LEA DX, NOGOODT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

TIME: POP CX

CMP CX, M

JE TIME\_E

INC CX

PUSH CX

PUSH 1

JMP FUNC4

TIME\_E: MOV AX, 1

CALL TIMER

JMP MENU

SELL\_GOOD:

MOV AX, 0

CALL TIMER

INC DX

MOV 17[BX], DX

LEA DX, SELLGOODT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

PUSH 1

JMP FUNC4

NO\_SELECTED:

LEA DX, NOTSELEVTEDT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

MOV AX, 1

CALL TIMER

JMP MENU

FUNC4: MOV BP, -1 ;功能4

;LEA DX, UPDATEPRIOT

;MOV AH, 09H

;INT 21H

;LEA DX, CRLF

;INT 21H

COMPUTE\_PRIO:

INC BP

CMP BP, N

JE COMPUTE\_OVER

LEA SI, GA1

MOV CX, BP

IMUL CX, 21

ADD SI, CX

MOV BX, SI

ADD BX, 13

MOV AX, [BX-2]

MOV BX, [BX]

MOVSX EAX, AX

MOVSX EBX, BX

MOV EDI, 128

MUL EDI

DIV EBX

MOV ECX, EAX

MOV BX, SI

ADD BX, 15

MOV AX, [BX+2]

MOV BX, [BX]

MOVSX EAX, AX

MOVSX EBX, BX

IMUL EDI

MOV EDI, 2

DIV EDI

MOV EDX, 0

DIV EBX

ADD EAX, ECX

MOV BX, SI

ADD BX, 19

MOV [BX], AX

JMP COMPUTE\_PRIO

COMPUTE\_OVER:

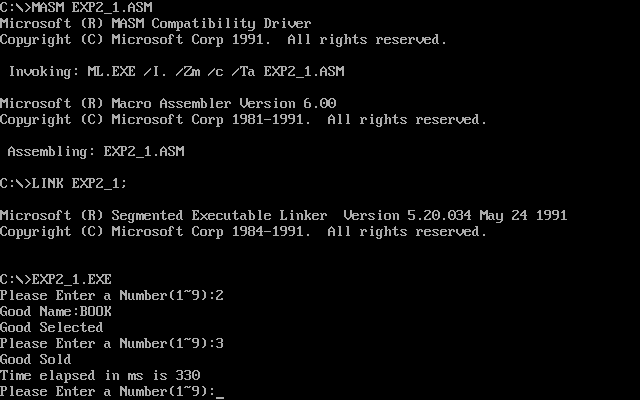
POP AX

CMP AX, 1

JE TIME

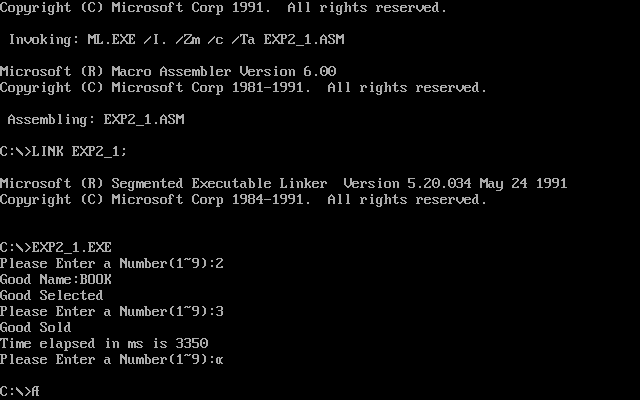
JMP MENU

2. 将商品数N设为30，计算次数设为1000，观察运行时间。



**图2.1 30商品，1000次计算所用时间**

3. 将商品数N设为30， 计算次数设为10000，观察运行时间。

****

**图2.2 30商品， 计算次10000运行时间**

4. 将商品数设为300，计算次数设为10000，观察运行时间



**图2.3 300商品，10000计算次数运行时间**

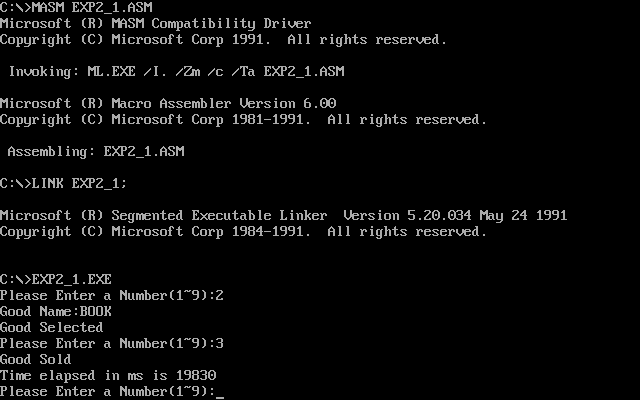
5. 可以看出程序运行所用时间与m\*n的大小基本成正比，也就是与循环体内代码的运行次数成正比。

6. 将CPU时钟频率固定为5000，商品数设为30，运行次数设为10000，观察运行时间。



**图2.4 时钟频率5000运行时间**

7. 将时钟频率设为固定8000，观察运行时间。



**图2.5 时钟频率8000运行时间**

8. 可以看出时钟频率越高，程序运行时间越短。

## 任务2.1实验过程

### 实验方法说明

1. 准备实验环境。

2. 针对循环体内指令条数进行优化，观察优化效果。

3. 使用更快的指令进行优化，观察优化效果。

4. 使用32位寄存器寻址进行优化，观察优化效果。

### 实验记录与分析

1. 将程序商品数设置为30，运行次数设置为10000,。

2. 去除循环体内的冗余语句，观察程序运行时间，优化后的代码如下：

FUNC3: PUSH 0

MOV BX, GOOD ;功能3

CMP BX, 0

JE NO\_SELECTED

MOV DX, 17[BX]

CMP DX, 15[BX]

JNE SELL\_GOOD

LEA DX, NOGOODT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

TIME: POP CX

CMP CX, M

JE TIME\_E

INC CX

PUSH CX

PUSH 1

JMP FUNC4

TIME\_E: MOV AX, 1

CALL TIMER

JMP MENU

SELL\_GOOD:

MOV AX, 0

CALL TIMER

INC DX

MOV 17[BX], DX

LEA DX, SELLGOODT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

PUSH 1

JMP FUNC4

NO\_SELECTED:

LEA DX, NOTSELEVTEDT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

MOV AX, 1

CALL TIMER

JMP MENU

FUNC4: MOV BP, -1 ;功能4

;LEA DX, UPDATEPRIOT

;MOV AH, 09H

;INT 21H

;LEA DX, CRLF

;INT 21H

COMPUTE\_PRIO:

INC BP

CMP BP, N

JE COMPUTE\_OVER

LEA SI, GA1

MOV CX, BP

IMUL CX, 21

ADD SI, CX

MOV BX, SI

MOVSX EAX, WORD PTR 11[BX]

MOVSX EBX, WORD PTR 13[BX]

MOV EDI, 128

MUL EDI

DIV EBX

MOV ECX, EAX

MOV BX, SI

MOVSX EAX, WORD PTR 17[BX]

MOVSX EBX, WORD PTR 15[BX]

MUL EDI

MOV EDI, 2

DIV EDI

MOV EDX, 0

DIV EBX

ADD EAX, ECX

MOV BX, SI

MOV 19[BX], AX

JMP COMPUTE\_PRIO

COMPUTE\_OVER:

POP AX

CMP AX, 1

JE TIME

JMP MENU



**图2.6 指令数量优化运行时间**

优化前的程序运行时间为3350ms，优化后的程序运行时间为2690ms，提升了约19.7%的效率

3. 将循环体中的乘法语句尽可能替换为左移语句，观察运行时间，优化后的代码如下：

FUNC3: PUSH 0

MOV BX, GOOD ;功能3

CMP BX, 0

JE NO\_SELECTED

MOV DX, 17[BX]

CMP DX, 15[BX]

JNE SELL\_GOOD

LEA DX, NOGOODT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

TIME: POP CX

CMP CX, M

JE TIME\_E

INC CX

PUSH CX

PUSH 1

JMP FUNC4

TIME\_E: MOV AX, 1

CALL TIMER

JMP MENU

SELL\_GOOD:

MOV AX, 0

CALL TIMER

INC DX

MOV 17[BX], DX

LEA DX, SELLGOODT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

PUSH 1

JMP FUNC4

NO\_SELECTED:

LEA DX, NOTSELEVTEDT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

MOV AX, 1

CALL TIMER

JMP MENU

FUNC4: MOV BP, -1 ;功能4

;LEA DX, UPDATEPRIOT

;MOV AH, 09H

;INT 21H

;LEA DX, CRLF

;INT 21H

COMPUTE\_PRIO:

INC BP

CMP BP, N

JE COMPUTE\_OVER

LEA SI, GA1

MOV CX, BP

IMUL CX, 21

ADD SI, CX

MOV BX, SI

MOVSX EAX, WORD PTR 11[BX]

MOVSX EBX, WORD PTR 13[BX]

SAL EAX, 7

MOV EDX, 0

DIV EBX

MOV ECX, EAX

MOV BX, SI

MOVSX EAX, WORD PTR 17[BX]

MOVSX EBX, WORD PTR 15[BX]

SAL EAX, 6

MOV EDX, 0

DIV EBX

ADD EAX, ECX

MOV BX, SI

MOV 19[BX], AX

JMP COMPUTE\_PRIO

COMPUTE\_OVER:

POP AX

CMP AX, 1

JE TIME

JMP MENU



优化后运行时间为2420ms，提升了大约10%的效率。

4. 使用32位寄存器，有更加灵活的寻址方式，优化后的代码如下：

FUNC3: PUSH 0

MOV BX, GOOD ;功能3

CMP BX, 0

JE NO\_SELECTED

MOV DX, 17[BX]

CMP DX, 15[BX]

JNE SELL\_GOOD

LEA DX, NOGOODT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

TIME: POP CX

CMP CX, M

JE TIME\_E

INC CX

PUSH CX

PUSH 1

JMP FUNC4

TIME\_E: MOV AX, 1

CALL TIMER

JMP MENU

SELL\_GOOD:

MOV AX, 0

CALL TIMER

INC DX

MOV 17[BX], DX

LEA DX, SELLGOODT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

PUSH 1

JMP FUNC4

NO\_SELECTED:

LEA DX, NOTSELEVTEDT

MOV AH, 09H

INT 21H

LEA DX, CRLF

INT 21H

MOV AX, 1

CALL TIMER

JMP MENU

FUNC4: MOV BP, -1 ;功能4

;LEA DX, UPDATEPRIOT

;MOV AH, 09H

;INT 21H

;LEA DX, CRLF

;INT 21H

COMPUTE\_PRIO:

INC BP

CMP BP, N

JE COMPUTE\_OVER

LEA ESI, GA1

MOVSX ECX, BP

IMUL ECX, 21

ADD ESI, ECX

MOVSX EAX, WORD PTR 11[ESI]

MOVSX EBX, WORD PTR 13[ESI]

SAL EAX, 7

MOV EDX, 0

DIV EBX

MOV ECX, EAX

MOVSX EAX, WORD PTR 17[ESI]

MOVSX EBX, WORD PTR 15[ESI]

SAL EAX, 6

MOV EDX, 0

DIV EBX

ADD EAX, ECX

MOV 19[ESI], AX

JMP COMPUTE\_PRIO

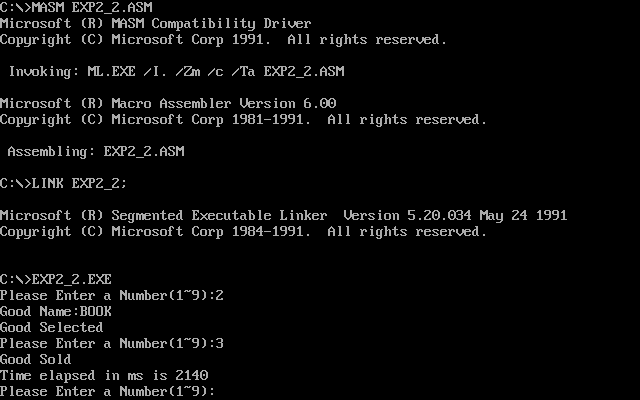
COMPUTE\_OVER:

POP AX

CMP AX, 1

JE TIME

JMP MENU



优化后的运行时间为2140ms，提升了大约11.6%的效率。

5. 可以看出，减少循环体内的语句数量是最直接的优化程序运行效率的方式，使用32位寄存器的根本目的也是通过更佳灵活的寻址方式来减少语句的数量。通过使用更快的语句也可以加快程序运行的效率，但是优化的限制比较多，优化的效果也小于直接减少指令条数，所以一般不要为了使用更快的指令而额外增加指令。

## 实验小结

### 主要收获

本次实验主要学习了计时程序的使用，以及如何在指令级别优化程序。通过本次实验，我意识到了指令级别的优化与算法级别的优化有很多不同，指令级别的优化在思路上倾向于优化到每一条语句，尽可能的使语句的数量减少，这需要我们对指令的使用特别熟悉，并掌握很多优化的技巧，比如使用LEA指令加基址变址寻址来替换一些简单的表达式。还需要我们对指令的性能有着足够的认识，了解哪些指令的运行时间比较快。在这次实验中，我对指令性能有了更加深刻的认识，也学到了一些指令使用的技巧。

### 课后问题

|  |  |  |  |
| --- | --- | --- | --- |
| 任务编号 | 序号 | 学生尝试的操作 | 结果 |
| 任务2.1 | 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？ | 程序设计越复杂，循环次数越多，CPU资源消耗越大 |
| 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？ | 信息显示的代码相比于普通代码消耗的时间要稍大 |
| 3 | m和n的最大值受到哪些限制？当商品种类的数量很多时（比如10000以上），如何处理一个数据段空间不够的问题？ | 受到段最大空间的影响，可以使用代码段来储存变量，也可使用附加段 |
| 任务2.2 | 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。 | 可以从指令数量，使用更高效的指令来优化程序。指令数量对程序运行效率的影响最为明显，指令运行效率其次 |
| 2 | 哪些指令是需要优化的关键性指令？ | 在循环体中的指令 |
| 3 | 循环体中用子程序调用与不用子程序调用会有多大影响？ | 调用子程序会增加运行时间，因为要保护和恢复现场，还增加了跳转指令 |
| 4 | 利用32位寄存器书写寻址方式具有较多的灵活性。你是否有通过此途径优化指令语句的例子？ | 可以使用LEA加基址变址寻址方式来代替一些简单的算式，还可以节省一两个寄存器，减少一些MOV指令 |

# 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。

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。注意：

1）在每个模块的开始，注明编写者的名字以及本模块中包含哪些子程序（只写名称）。

2）在程序设计思想中，要说明模块之间的通信“协议”，包括相互有关联的子程序名及其类型与参数、变量名及其类型、段的定义要求等信息。

3）对每个模块设计一套简单的测试方法，包括一个简单明了的测试框架和一组必要的测试数据，能验证被测模块符合模块之间的通信“协议”，以及该模块的功能基本正确。每个模块单独测试通过后，再一起连接生成一个程序，然后继续进行整体调试（虽然在一开始的时候就可以直接把编好的几个模块连接成一个程序，进行测试和调试，但这并不是实际中采用的最好的方法，而对单个模块先单独进行独立的测试是实际中比较有效的）。

4）需要借鉴书上（或网上）的进制转换程序：十进制转二进制的子程序F10T2和二进制转十进制的子程序F2T10（直接借鉴的程序不需要在实验报告中描述其详细内容）。

## 任务3.1实验过程

### 设计思想及存储单元分配

（1）宏定义

将原来程序中经常重复用到的语句定义为了宏，在主模块中定义了三个宏。分别为PRINT，用于打印语句，READ，用于读入键盘输入，CRLF用于输出回车换行。宏定义如下：

CRLF MACRO ;回车

LEA DX, CRLFT

MOV AH, 09H

INT 21H

ENDM

PRINT MACRO A ;打印文字

LEA DX, A

MOV AH, 09H

INT 21H

ENDM

READ MACRO A ;读取输入

LEA DX, A

MOV AH, 0AH

INT 21H

ENDM

（2）将之前实现的功能全部改为子程序

将之前实现的5个功能改为了子程序，由于每个子程序调用前没有前后文的要求，所以不需要加入保护现场和还原现场的语句，只需简单的将代码放到子程序块中，在MENU代码段，读取用户输入，判断用户输入的字符后调用对应程序，最后再返回MENU代码段。例如对于功能1：

CMP AL, '1'

JE CF1

CF1: CALL FUNC1

JMP MENU

以这样的形式调用。其余类似

（3）更改功能2

为功能2新增了显示商品信息的功能，查找商品的部分和之前一致，显示商品信息的功能使用了书上提供的F2TO10程序并加以简单的修改。

使用BPRICET储存进货价输出提示信息，SPRICET储存收货价输出提示信息。BAMOUNTT储存进货数输出提示信息，SAMOUNTT储存收货数输出提示信息，使用VALUE储存转换为十进制的数。

使用BX储存商品的首地址，根据F2TO10的定义，使用SI储存转换后的字符串储存的位置，使用AX储存被转换的二进制数，为了方便代码的编写，定义了宏SHOWVALUE，宏定义如下：

SHOWVALUE MACRO A, B

PRINT A

MOV AX, B[BX]

LEA SI, VALUE

CALL F2TO10

MOV BYTE PTR [SI], 0DH

MOV BYTE PTR [SI+1], 0AH

MOV BYTE PTR [SI+2], '$'

MOV BYTE PTR [SI+2], '$'

PRINT VALUE

ENDM

参数A为输出提示信息，参数B指示了要输出的数据相对于商品首地址的偏移量。

要增加功能只需要在原来的代码后一次SHOWVALUE四个数据即可。源代码见3.3.2。

（4）功能6

需要使用的变量和功能2新增功能使用的变量一致，再加上分隔符号SEPARATOR，以及权限不够时的提示语句NOTBOSST和没有选中商品时输出的提示语句NOTSELECTEDT。

使用BX储存商品首地址，AX储存要改变的数据，SI为要储存转换结果的空间的首地址，AX和CX用于将输入的数据转换为二进制时运算用的中间寄存器，DI用于储存是否输入了非法字符。

FUC6函数为功能6部分主函数，其中NOT\_SELECTED标号下为没有选中商品时运行的语句，NOT\_BOSS下为不是老板时运行的语句。

首先判断是否为老板，如果不是老板，输出错误语句，在判断是否选中商品，如果没选中商品输出错误语句，然后把商品首地址置入BX，定义宏CHANGEVALUE用来改变商品的值，宏定义如下：

CHANGEVALUE MACRO A, B

LOCAL C

C: PRINT A

MOV AX, B[BX]

LEA SI, VALUE

CALL F2TO10

MOV BYTE PTR [SI], '$'

PRINT VALUE

PRINT SEPARATOR

CALL GETVALUE

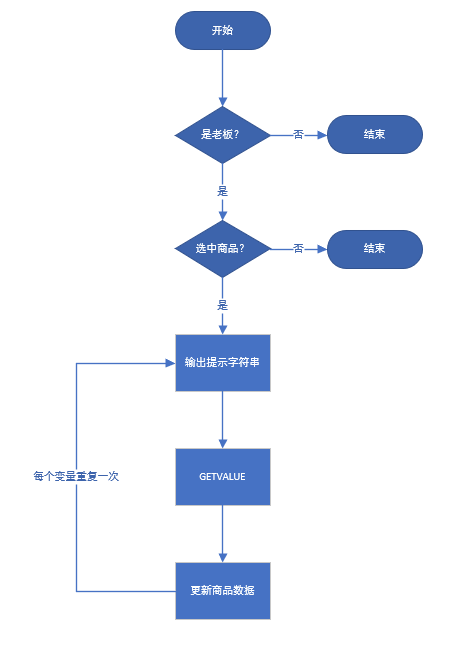
CMP DI, 1

JE C

MOV B[BX], DX

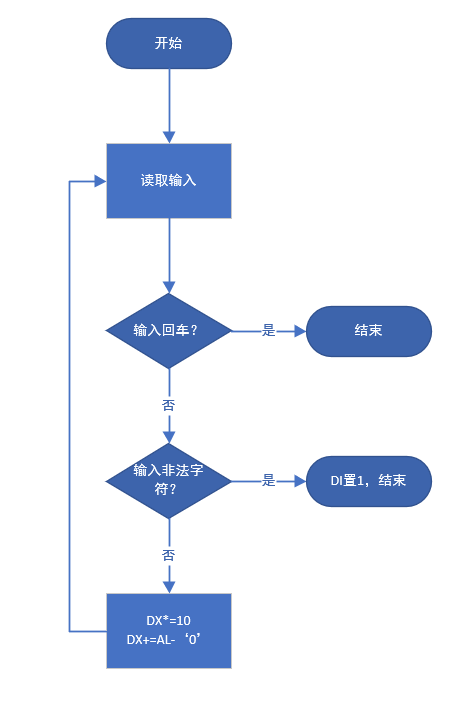
ENDM

参数A为提示语句，参数B为要改变的数据相对于商品首地址的偏移量。本地标号C为发现输入了非法字符时，返回C重新输入。下面为功能6的流程图：



**图3.1 功能6流程图**

定义了函数GETVALUE用于读取键盘输入并转换为二进制数，函数将转换后的数储存在DX中，使用DI判断是否输入了违法字符。函数内不断从键盘读入一个字符，如果读入了回车，结束输入，退出函数。如果输入了非法字符，将DI置为1，退出函数。否则将DX中内容\*10，然后在加上输入的字符与字符0的差。



**图3.2 GETVALUE流程图**

（5）使用模块化设计

将功能6的函数放在一个单独的子模块文件中，由于F2TO10函数在功能6和功能2中都有使用，所以将F2TO10也放在了单独的模块中。三个模块中，主模块含有堆栈段和数据段，三个模块的代码段都以PARA PUBLIC ‘CODE’的参数声明，为连续存放的段。

在主模块中，将其余两个模块所需要的变量声明为了PUBLIC，具体变量和类型如下:

NOTSELECTEDT:BYTE, NOTBOSST:BYTE, BPRICET:BYTE, SPRICET:BYTE, BAMOUNTT:BYTE, SAMOUNTT:BYTE, VALUE:BYTE, SEPARATOR:BYTE, AUTH:BYTE, GOOD:WORD, CRLFT:BYTE 在主函数中使用了两个外部函数，FUNC6和F2TO10都为NEAR类型。

在FUNC6模块中，将FUNC6函数声明为PUBLIC，使用了主函数中声明的所有PUBLIC变量，并使用了外部函数F2TO10.

在F2TO10模块中，声明了F2TO10函数为PUBLIC，没有使用外部的函数和变量。

### 源程序

功能2新增代码：

SHOWVALUE BPRICET, 11

SHOWVALUE SPRICET, 13

SHOWVALUE BAMOUNTT, 15

SHOWVALUE SAMOUNTT, 17

功能6 模块代码：

;编写者：顾时禹

;函数名：FUNC6, GETVALUE

NAME FUNC6

PUBLIC FUNC6

EXTERN F2TO10:NEAR, NOTSELECTEDT:BYTE, NOTBOSST:BYTE, BPRICET:BYTE, SPRICET:BYTE, BAMOUNTT:BYTE, SAMOUNTT:BYTE, VALUE:BYTE, SEPARATOR:BYTE, AUTH:BYTE, GOOD:WORD, CRLFT:BYTE

PRINT MACRO A ;打印文字

LEA DX, A

MOV AH, 09H

INT 21H

ENDM

CRLF MACRO ;回车

LEA DX, CRLFT

MOV AH, 09H

INT 21H

ENDM

CHANGEVALUE MACRO A, B

LOCAL C

C: PRINT A

MOV AX, B[BX]

LEA SI, VALUE

CALL F2TO10

MOV BYTE PTR [SI], '$'

PRINT VALUE

PRINT SEPARATOR

CALL GETVALUE

CMP DI, 1

JE C

MOV B[BX], DX

ENDM

CODE SEGMENT USE16 PARA PUBLIC 'CODE'

ASSUME CS:CODE

FUNC6 PROC

CMP AUTH, 0

JE NOT\_BOSS

CMP GOOD, 0

JE NOT\_SELECTED

MOV BX, GOOD

CHANGEVALUE BPRICET, 11

CHANGEVALUE SPRICET, 13

CHANGEVALUE BAMOUNTT, 15

CHANGEVALUE SAMOUNTT, 17

RET

NOT\_SELECTED:

PRINT NOTSELECTEDT

CRLF

RET

NOT\_BOSS:

PRINT NOTBOSST

CRLF

RET

FUNC6 ENDP

GETVALUE PROC

PUSH AX

PUSH CX

XOR DI, DI

XOR CX, CX

XOR DX, DX

LOOP1: MOV AH, 01H

INT 21H

CMP AL, 0DH

JE RETURN

CMP AL, 0AH

JE RETURN

CMP AL, '0'

JB ERR

CMP AL, '9'

JA ERR

SUB AL, '0'

MOV CL, AL

MOV AX, DX

MOV DX, 10

MUL DX

ADD AX, CX

MOV DX, AX

JMP LOOP1

ERR: CRLF

MOV DI, 1

POP CX

POP AX

RET

RETURN: POP CX

POP AX

RET

GETVALUE ENDP

CODE ENDS

END

### 实验步骤

（1）准备实验环境

（2）测试功能2新增功能

（3）模块测试F2TO10

（4）模块测试功能6

（5）整体测试

### 3.3.4 实验记录与分析

（1）准备实验环境

（2）测试功能2新增功能

**图3.3 功能2新增功能展示**

（3）编写F2TO10模块测试程序，显示100和1212两个数字，模块测试F2TO10，测试程序如下：

EXTERN F2TO10:NEAR

.386

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

DATA SEGMENT

VALUE DB 10 DUP(0)

DATA ENDS

CODE SEGMENT USE16 PARA PUBLIC 'CODE'

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

START: MOV AX, DATA

MOV DS, AX

LEA SI, VALUE

MOV AX, 100

CALL F2TO10

MOV BYTE PTR [SI], 0DH

MOV BYTE PTR [SI+1], 0AH

MOV BYTE PTR [SI+2], '$'

LEA DX, VALUE

MOV AH, 09H

INT 21H

LEA SI, VALUE

MOV AX, 1212

CALL F2TO10

MOV BYTE PTR [SI], 0DH

MOV BYTE PTR [SI+1], 0AH

MOV BYTE PTR [SI+2], '$'

LEA DX, VALUE

MOV AH, 09H

INT 21H

MOV AX, 4C00H

INT 21H

CODE ENDS

END START



**图3.4 F2TO10模块测试**

（4）编写模块6测试程序，更改PEN商品信息并展示更改后的PEN商品信息，模块测试FUNC6，测试程序如下：

EXTERN F2TO10:NEAR, FUNC6:NEAR

PUBLIC NOTSELECTEDT, NOTBOSST, BPRICET, SPRICET, BAMOUNTT, SAMOUNTT, VALUE, SEPARATOR, AUTH, GOOD, CRLFT

PRINT MACRO A ;打印文字

LEA DX, A

MOV AH, 09H

INT 21H

ENDM

SHOWVALUE MACRO A, B

PRINT A

MOV AX, B[BX]

LEA SI, VALUE

CALL F2TO10

MOV BYTE PTR [SI], 0DH

MOV BYTE PTR [SI+1], 0AH

MOV BYTE PTR [SI+2], '$'

PRINT VALUE

ENDM

.386

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

DATA SEGMENT USE16 PARA PUBLIC 'DATA'

NOTSELECTEDT DB 'No Goods Selected$'

UPDATEPRIOT DB 'Priority Updated$'

NOTBOSST DB 'Authrity Denied$'

BPRICET DB 'Buying Price:$'

SPRICET DB 'Selling Price:$'

BAMOUNTT DB 'Buying Amount:$'

SAMOUNTT DB 'Selling Amount:$'

VALUE DB 8 DUP(0)

SEPARATOR DB '->$'

AUTH DB 1 ;当前登录状态，0表示顾客状态

GOOD DW GA1 ;当前浏览商品地址

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, ? ;进货价，销售价，进货总数，已售数量，推荐度

CRLFT DB 0DH, 0AH, '$'

DATA ENDS

CODE SEGMENT USE16 PARA PUBLIC 'CODE'

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

START: MOV AX, DATA

MOV DS, AX

MOV BX, GOOD

CALL FUNC6

SHOWVALUE BPRICET, 11

SHOWVALUE SPRICET, 13

SHOWVALUE BAMOUNTT, 15

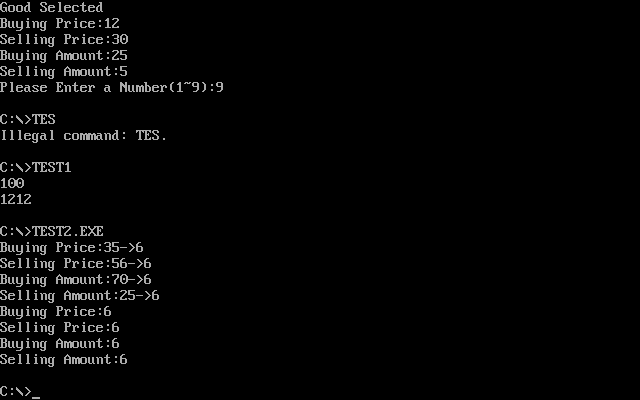
SHOWVALUE SAMOUNTT, 17

MOV AX, 4C00H

INT 21H

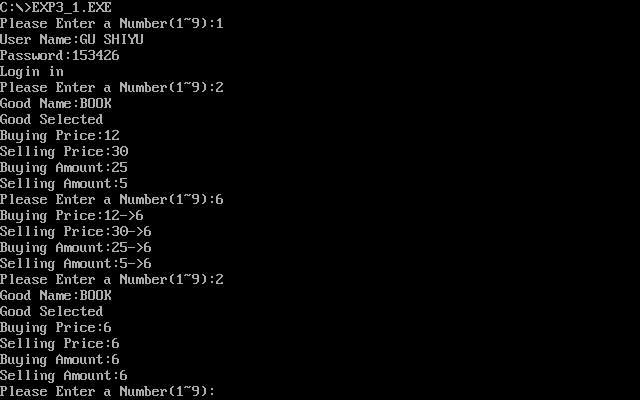
CODE ENDS

END START



**图3.5 FUNC6模块测试结果**

（5）测试在整个系统中的模块6功能，首先登陆，再查询BOOK商品，更改完商品信息后再次查找BOOK商品。



**图3.6 FUNC6模块测试结果**

可以看出

## 任务3.2实验过程

### 实验方法说明

1. 准备实验环境

2. 新建VS2019项目

3. 编写C源文件，代替原来的主界面模块

4. 修改汇编源文件

5. 编译，链接程序

6. 测试程序各项功能是否正常

### 实验记录与分析

1. 准备实验环境，新建VS2019空项目

2. 编写C语言界面源文件如下：

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

extern void FUNC1();

extern void FUNC2();

extern void FUNC3();

extern void FUNC4();

extern void FUNC6();

extern void FUNC8();

int main()

{

int op = 0;

while (op != 9)

{

printf("SHOP:MY SHOP\n");

printf("---------------------------------------------\n");

printf("1.Login in\n2.Select good\n3.Buy good\n4.Update recommendation\n6.Update good\n8.Show CS\n9.Exit\n");

printf("---------------------------------------------\n");

printf("Please Enter a Number:");

scanf("%d", &op);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

switch (op)

{

case 1:

FUNC1();

break;

case 2:

FUNC2();

break;

case 3:

FUNC3();

break;

case 4:

FUNC4();

break;

case 6:

getchar();

FUNC6();

break;

case 8:

FUNC8();

break;

default:

break;

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

system("pause");

system(("cls"));

}

return 0;

}

3. 修改汇编源文件，首先由于与C语言混合汇编使用伪指令定义段，段为32位。不能使用21号中断进行输入与输出，所以调用C语言的printf和scanf函数进行输入输出，首先加入头文件与函数声明：

printf PROTO :ptr sbyte, :VARARG

scanf PROTO :ptr sbyte, :VARARG

includelib libcmt.lib

includelib legacy\_stdio\_definitions.lib

之后修改源文件中的宏指令：

CRLF MACRO ;回车

invoke printf, offset FMT, addr CRLFT

ENDM

PRINT MACRO A ;打印文字

invoke printf, offset FMT, addr A

ENDM

READ MACRO A ;读取输入

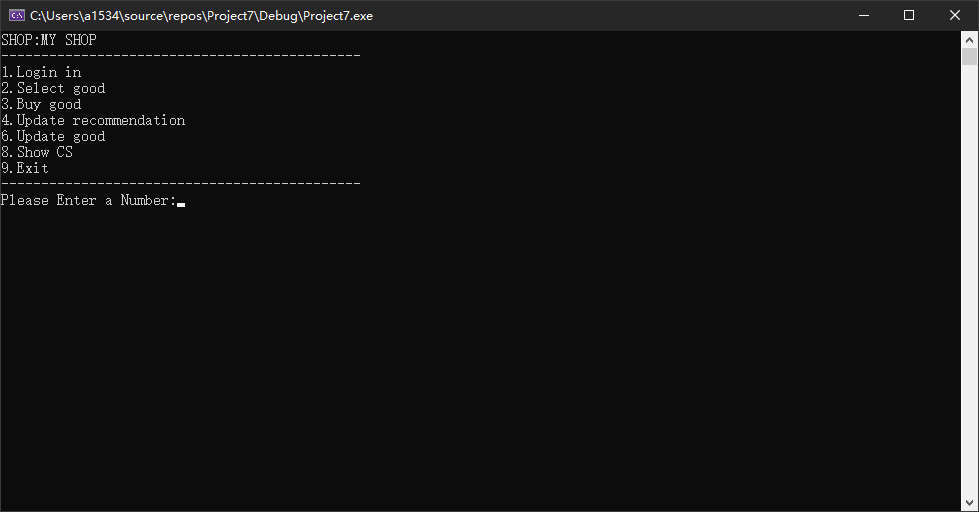
invoke scanf, offset FMT, addr A

ENDM

最后再根据32位程序的要求修改代码中储存地址的寄存器，这一步比较简单，详细代码在这里不再给出。

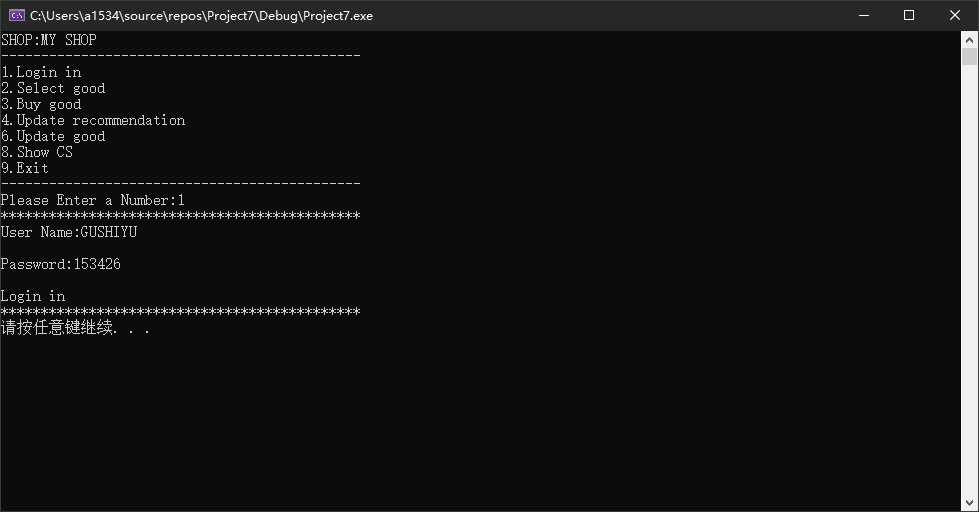
4. 测试程序功能

显示程序主界面



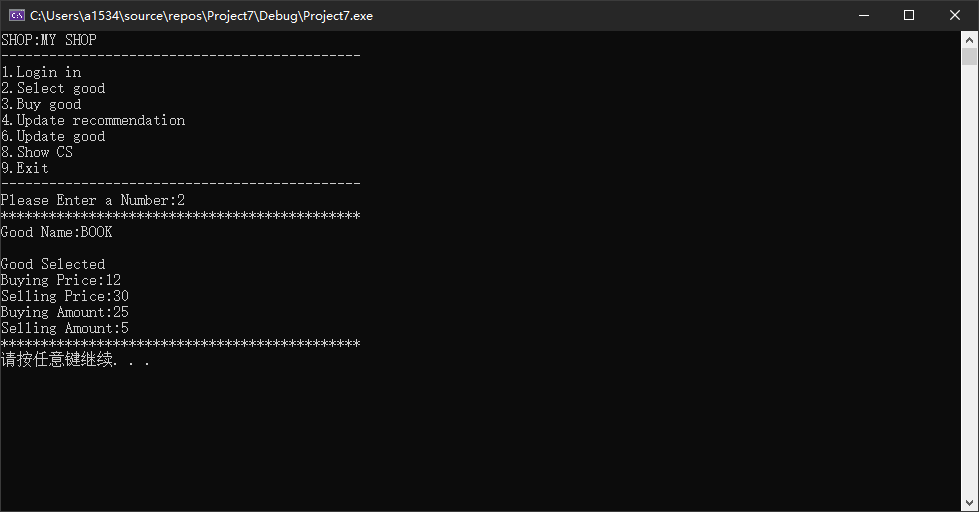
**图3.7 程序主界面**

验证功能一，登录商店



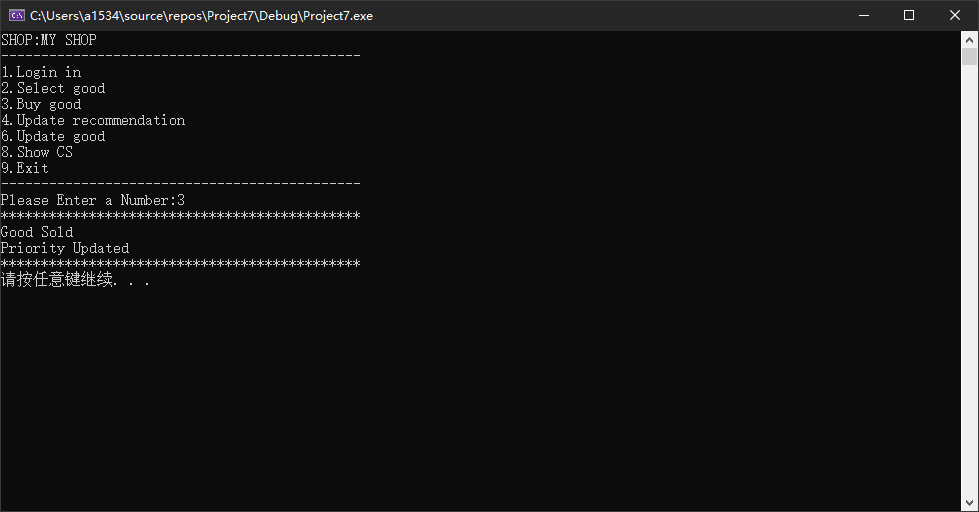
**图3.8 验证功能一**

验证功能二，查找商品



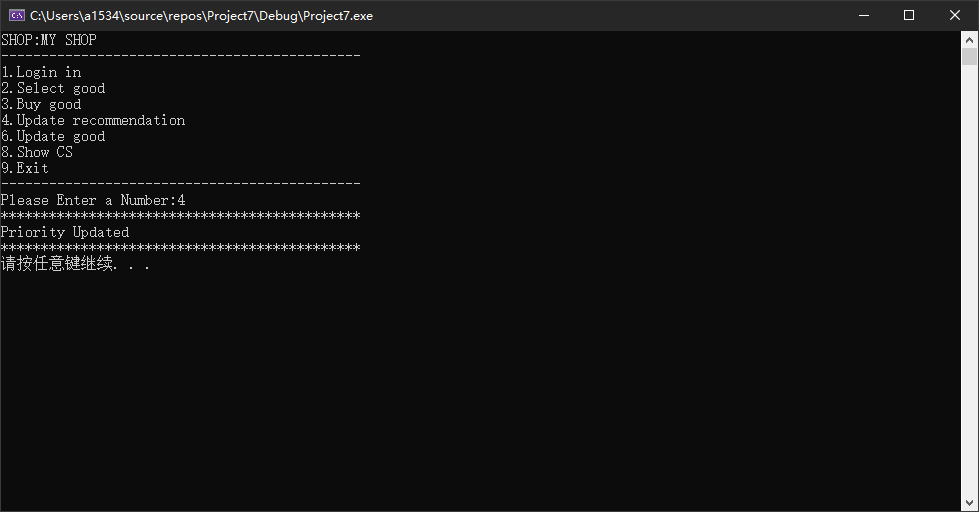
**图3.9 功能二显示商品信息**

验证功能三，购买商品



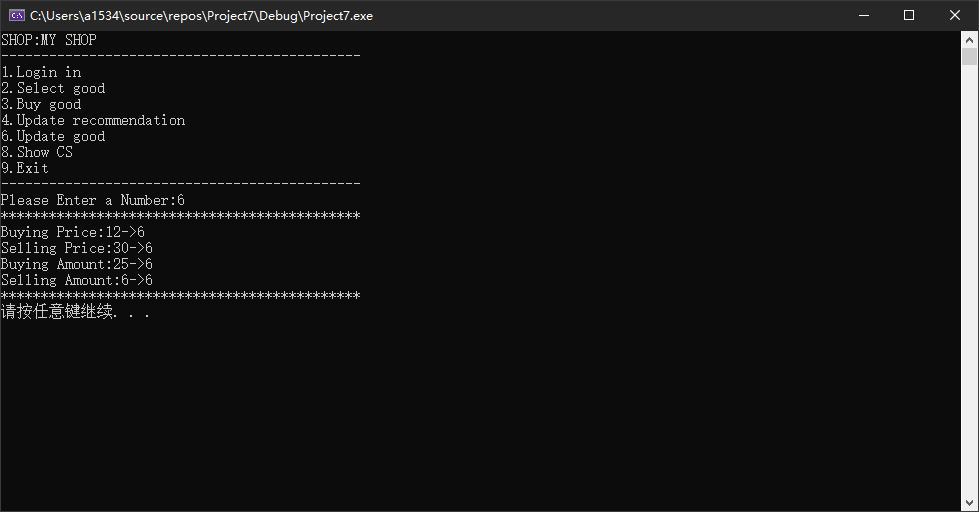
**图3.10 功能三，购买商品**

验证功能四，计算推荐度

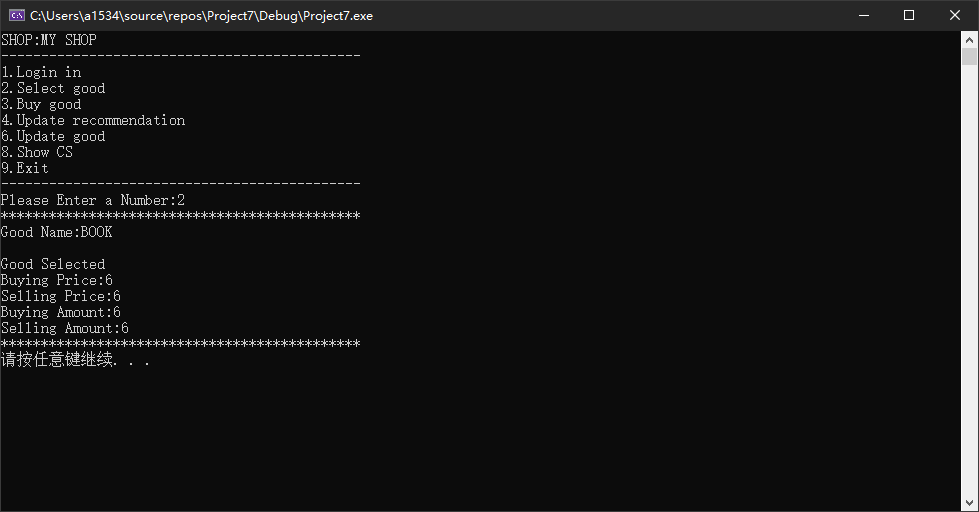


**图3.11 功能四，计算推荐度**

验证功能6， 修改商品信息

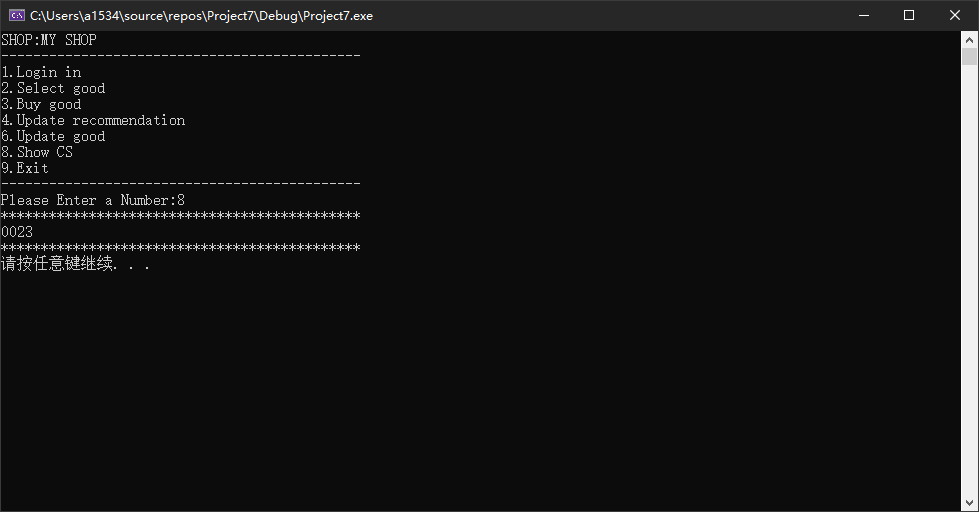


**图3.12 功能六，修改商品信息**



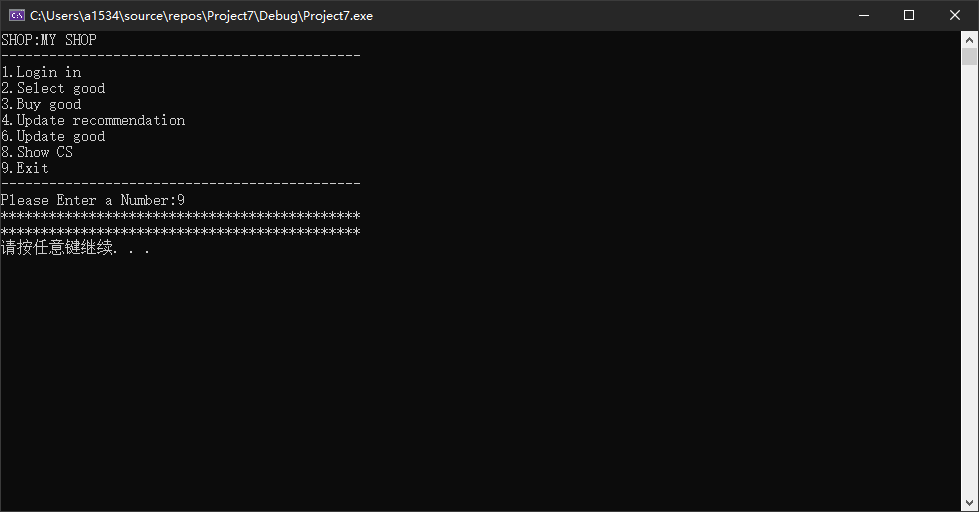
**图3.13 商品信息修改完毕**

验证功能八， 显示CS的值



**图3.14 CS的值**

退出程序



**图3.15 退出程序**

由上面的截图可知，使用C语言编写的主界面功能完全正确。

本次实验的难点集中在C语言与汇编语言的相互调用上，在C语言内，需要使用extern语句声明汇编函数原型，同样，在汇编源文件内也要声明使用到的C语言函数原型，如果使用了C的库函数，也要在汇编中加入相应的lib文件。

## 实验小结

### 主要收获

通过这次实验，我首先学会了汇编语言的模块化编程，通过将一个复杂的程序拆分成一个个简单，独立的模块，不仅使得代码更佳简洁易懂，便于维护和修改，也使得使用率高的代码可以不断被复用，减小了程序开发的成本，在调试是，一个一个模块的独立调试也可减少时间成本，所以模块化编程的思想是极其重要的，也是以后我们参与到大型项目中必须要具备的核心编程思想。同时，由于汇编语言非常贴近底层，使用汇编语言进行模块化编程可以更好的观察汇编程序和连接程序是如何处理模块之间的相互调用的，通过将函数和变量声明为extern或者public，就可以让链接程序把不同源文件当中共同使用的变量联系起来，组装成最后的可执行文件。

然后通过C语言和汇编函数的混合编程，我认识到了所有编程语言的源程序，最后都是先生成出目标文件，再通过链接程序组装在一起的，不同编程语言生成的目标文件也可以链接到一起。

### 课后问题

1. 在TD中跟踪到子程序内部有几种方法？在TD中观察子程序调用和返回时堆栈的变化。若执行RET前把栈顶的数值改掉，那么RET执行后程序返回到何处？

可以使用F8，也可以在子程序内部打上断点。会回到更改后的地址

2. 注意观察FAR、NEAR类型子程序的RET指令的机器码有何不同？观察FAR类型子程序被调用时堆栈的变化情况。对一个NEAR类型子程序强制使用FAR调用（即CALL FAR PTR 子程序名）会怎样？反之，对一个FAR类型的子程序（子程序可以与主程序在同一个代码段，也可以在不同的代码段）强制使用NEAR调用又会怎样？

FAR类型子程序在调用时会将段基址和偏移地址一起压入栈，强制使用另一种调用方式都有可能引起返回位置的错误

3. 通过把一个模块拆成多个模块或反之，体会子程序和模块化程序设计的方法，体会模块调用关系图、子程序功能说明、输入/输出说明在程序设计中的作用。注意 “功能”描述、“模块”描述、“子程序”说明等的区别与联系。

通过加入适当的说明，可以使得模块的功能一目了然，便于之后的修改和复用

4. 观察不同模块的可合并段合并后变量偏移地址的变化情况。观察不同段在内存里的放置次序。体会模块间段的定义及其对应的装配方法。

5. 观察模块间的参数的传递方法，包括公共符号的定义和外部符号的引用，若符号名不一致或类型不一致会有什么现象发生？

符号名不同可能会导致编译和链接错误，类型不一致可能会导致程序功能的错误

6. 通过TD观察宏指令在执行程序中的替换和扩展，解释宏和子程序的调用有何不同。

宏是单纯把宏所在的位置替换成一段代码，是在编译之前执行的，子程序的调用是实际的一条指令，会改变ip和栈的值。

7. EXTRN说明语句放在.386之前或者之后有什么区别？(需要用TD观察)

放在.386后面会使用32位寻址，导致寻址错误。

8. EXTRN说明的变量的段与段寄存器的关联关系（ASSUME伪指令所表达的信息）是否能带入到本模块中？

不能，需要指定段

9. 如何利用宏功能使汇编语言的程序变得更加直观易读？

可以将功能不易辨别多次出现的代码段定义为宏，并使用一个直观的名字

例：下面是一个利用宏功能直观化后的完整代码段程序，请写出对应的宏定义，并模仿该方式对自己编写的某段程序进行类似的改写。

StartProgram code，data，stack，start

Initial\_ds

GetStringTo BUF

DisplayStringFrom BUF

ExitToDOS

EndProgram code，start

data segment use16

BUF db 10 dup(0)

data ends

stack segment use16 STACK

200 dup(0)

stack ends

code segment use16

assume cs:code, ds:data., ss:stack

start:

mov ax, data

mov ds, ax

lea dx, BUF

mov ah, 0AH

int 21h

lea dx, buf+2

mov ah. 09h

int 21h

mov ax, 4c00h

int 21h

code end

end start

StartProgram code，data，stack，start

Initial\_ds

GetStringTo INNAME

Check INNAME

ExitToDOS

EndProgram code，start

10. 通过在编译器配置中设置：在编译时同时生成汇编语言源程序的方法，得到C程序对应的汇编源程序。观察C语言编译器中对各种符号的命名规则（指编译器内部可以识别的命名规则，比如，符号名前面是否加下划线“\_”，默认段名，等），主、子程序之间参数传递的机制，通过堆栈传递参数后堆栈空间回收的方法（要设计一个有多个参数需要传递的C函数）。

函数名和变量名前会加上下划线，参数传递是通过堆栈传递的，在函数时会将参数所占的空间释放。

11. 对混合编程形成的执行程序，用调试工具观察由C语言形成的程序代码与由汇编语言形成的程序代码之间的相互关系，包括段、偏移的值，汇编指令访问C的变量时是如何翻译的，等。

C语言中的一些函数会改变寄存器的值以备之后使用，但是如果强行插入了汇编语句改变了寄存器的值就会出现错误。

12. 观察C编译器的优化策略对代码的影响。通过实际观察与分析，记录本实验中汇编语言程序的效率会优于C语言程序的实例（至少给出一处的观察结果）。

在C语言中，类似if(a == b) break;类型的语句会被翻译为四句汇编语句，但实际上使用三句汇编语句就可以实现这个功能。

13. 通过调试混合编程的程序，体会与纯粹汇编语言编写的程序的调试过程的差异。

混合编程的程序在调用其他语言的函数时需要对函数的声明进行转换，并且要注意变量类型之间的对应

# 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）。

“7.迁移商店运行环境”具体实现的**提示**如下：

【前提条件】在网店商品信息管理系统（任务3.1对应的程序）中新定义了一个与堆栈段STACK大小一样的数据段空间STACKBAK作为两个堆栈段相互切换的空间。

（1）设定切换的时钟时刻（可以通过界面输入，也可以在变量里直接给定）。为便于观察，可以只设定时钟的分钟和秒（甚至只设定秒），也就是当时钟走到任一小时的对应分钟和秒的时刻时就切换一次（如果只设定秒，那么，时钟走到任一分钟的对应秒的时刻就会切换一次）。

（2）检查中断服务程序是否被安装，已经安装的话就返回主菜单界面。

（3）设置当前堆栈段信息和待转移的堆栈段信息，安装中断处理程序（接管8号时钟中断），返回到主菜单界面。

中断处理程序的主要流程：

（1）执行原中断服务程序的功能（使用PUSH+CALL组合）。

（2）用IO指令从CMOS芯片中读取当前时钟的分钟和秒信息，判断是否与设定的分钟和秒时间相同（注意CMOS分钟和秒信息的编码为BCD码），不相同则中断返回；相同则执行（3）。

（3）获取必要的当前堆栈段信息和待转移的堆栈段信息，并记录本次转移操作的状态（用于下次转移时判断转移的方向是转移到STACK还是STACKBAK）。

（4）把当前堆栈段里的有效内容复制到新的堆栈段。

（5）切换堆栈段指针。

（6）中断返回。

**任务4.2：数据加密与反跟踪 *（研究性实验）***

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 *（研究性实验）***

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4.1 实验过程

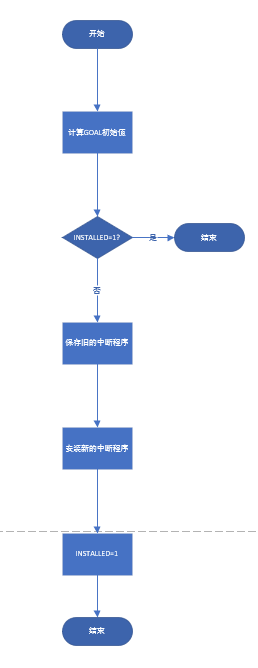
### 设计思想及存储单元分配

INSTALLT用于储存中断程序安装提示信息，INSTALLED用于记录中断程序是否安装，STACKBAK为数据段中的栈空间，在代码段中也定义了四个变量，分别为OLD\_INT用于储存原有中断程序的位置，COUNT用于控制每秒只会触发一次新中断程序，SEC用于储存设定的切换栈的秒数，GOAL用于指示要切换到的栈段基址。

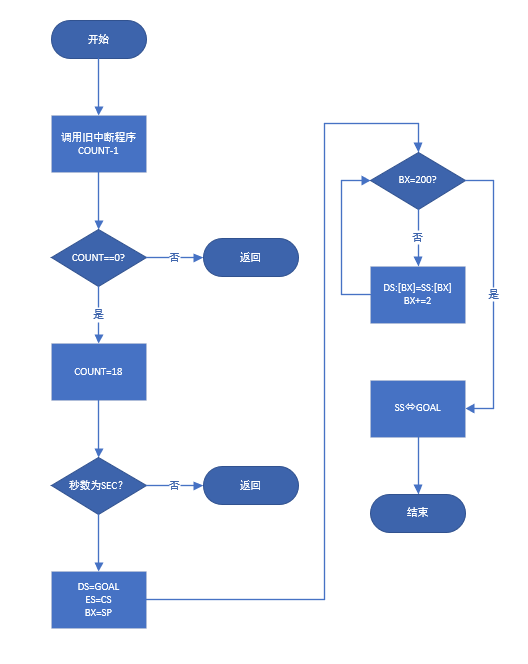
AX和BX用于将新中断程序送入中断表时的中间寄存器。在终端程序中，使用AX和BX用于复制运算中的中间寄存器。

首先在FUNC7中，先计算STACKBAK作为SS时的值，应该为STACKBACK的偏移地址加上DS的值。先判断INSTALLED是否为0，如果为0则安装新的终端程序，并将INSTALLED设置为1，否则代表已经安装新的终端程序，直接返回主界面。在FUNC9中，加入了检测INSTALLED是否为0的代码，如果不为0，代表安装了中断程序，要恢复原本的中断程序。

在新中断程序NEW08H中，首先开中断，然后保护现场，先将ES段设置为CS，再将DS设置为GOAL中的内存地址，这样两个栈即为DS段的前200个字节和SS段的前200，然后从CMOS中读取秒信息，将压缩的BCD码解压，与SEC比较，一致则进行复制操作。复制操作现将SP移入BX，然后将BX中的内容作为字指针，一次复制两个字节的内容，然后将BX+2，当BX为200时代表栈全部有效字节已经赋值完毕。将GOAL中地址设置为新的SS，将SS中的值放入GOAL中。



**图4.1 FUNC7流程图**

****

**图4.2 NEW08H流程图**

### 4.3.2 源程序

FUNC7源程序

FUNC7 PROC

MOV AX, DATA

MOV BX, OFFSET STACKBAK

ADD AX, BX

MOV GOAL, AX

POP AX

PUSH 1

PUSH 2

PUSH 3

PUSH 4

PUSH AX

CMP INSTALLED, 0

JNE INSTALL\_OVER

MOV INSTALLED, 1

XOR AX, AX

MOV ES, AX

MOV AX, ES:[08H\*4]

MOV OLD\_INT, AX

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

MOV OLD\_INT+2, AX

CLI

MOV WORD PTR ES:[08H\*4], OFFSET NEW08H

MOV ES:[08H\*4+2], CS

STI

PRINT INSTALLT

CRLF

INSTALL\_OVER:

RET

FUNC7 ENDP

NEW08H源程序

NEW08H PROC FAR

PUSHF

CALL DWORD PTR CS:OLD\_INT

DEC CS:COUNT

JZ TRANS

IRET

TRANS: MOV CS:COUNT, 18

STI

PUSHA

PUSH DS

PUSH ES

MOV AL, 0

OUT 70H, AL

JMP $+2

IN AL, 71H

MOV AH, AL

AND AL, 0FH

SHR AH, 4

XCHG AH, AL

CMP AX, WORD PTR CS:SEC

JNE INT\_OVER

MOV AX, CS

MOV ES, AX

MOV DS, GOAL

MOV BX, SP

COPY: MOV DX, SS:[BX]

MOV DS:[BX], DX

ADD BX, 2

CMP BX, 200

JB COPY

MOV AX, SS

MOV BX, DS

MOV GOAL, AX

MOV SS, BX

INT\_OVER:

POP ES

POP DS

POPA

IRET

NEW08H ENDP

### 4.3.3 实验步骤

（1）验证功能7

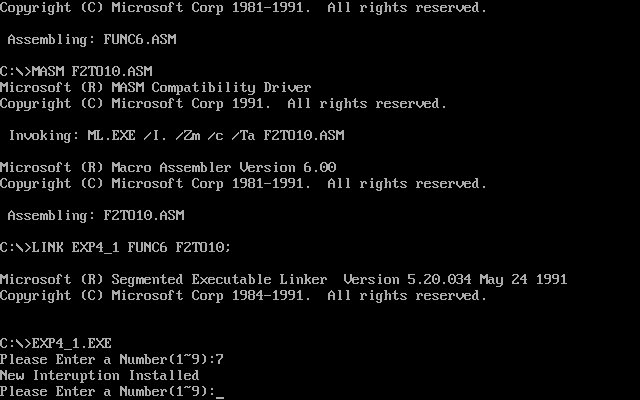
（2）验证功能7 在已经安装后是否不会重复安装

（3）验证新中断程序功能

### 4.3.4 实验记录与分析

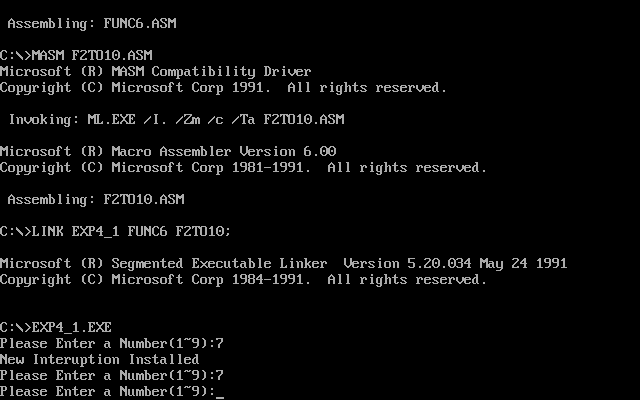
（1）准备实验环境

（2）验证功能7



**图4.3 功能7正常安装**

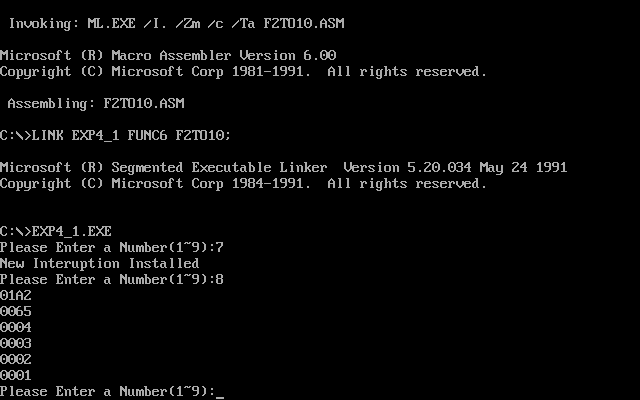
（3）已经安装新的中断程序后再运行功能7



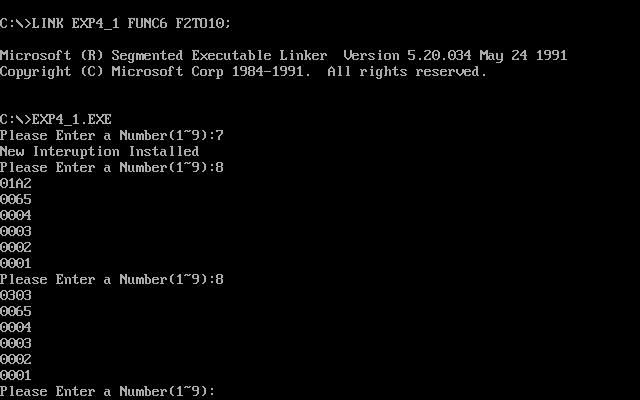
**图4.4 不会重复安装中断程序**

（4）验证中断程序功能

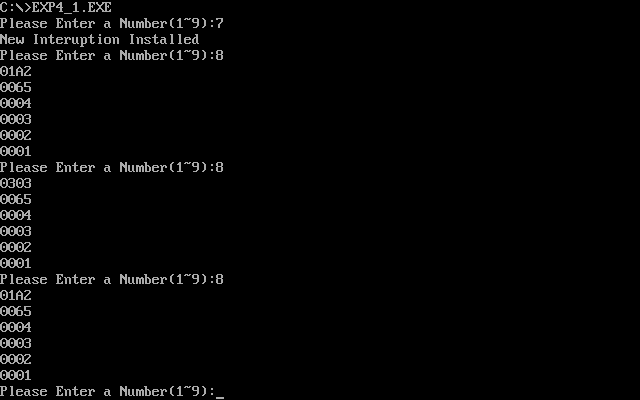
在FUNC7中将1，2，3，4依次入栈，之后在功能八中添加查看SS和栈有效内容的代码，运行如下。



**图4.5 改变栈前运行功能8**



**图4.6 改变栈后运行功能8**

****

**图4.7 再次改变栈**

可见除了SS的值其他的内容均未变化，栈内内容依次为，功能8返回地址，4,3,2,1。

由此可见功能7和新的中断程序完全正常。

在任务1的实现中有很多细节需要注意，首先是由于在代码中对于DS和ES都有做改变。所以在程序开头要保护现场，运行结束后要恢复现场，不然之后DS会错误，导致输出的内容变为乱码。其次要注意恢复和安装中断程序时。是否正确的操作，将地址放入错误的位置很可能会导致内存泄漏然后程序死机。最后要注意切换SS的值时，不是将STACKBAK的偏移地址放入SS中，因为STACKBAK定义在DS中，实际物理地址并不是从0开始的，但是作为SS时应该放入STACKBAK的物理地址。所以应该将STACKBAK的偏移值加上DS的内容放入SS中，才能保证正常的切换。

## 任务4.2实验过程

### 实验方法说明

1. 准备实验环境

2. 加密老板登录密码，密码每个字节的值+1

3. 添加新的21H号中断程序，使得在读取用户输入后将用户输入使用同样的加密方法之后再与密码比较。

4. 加密进货价，将进货价储存在新的变量中，原来储存进货价的位置储存新变量的地址。

5. 当以老板身份登录后，将进货价放回原位置。

### 实验记录与分析

1. 准备实验环境

2. 加密老板登录密码，将每一位的值都加1，加密后的老板密码为‘264537’，原密码为‘153426’

3. 编写新的21H中断程序，在读取用户输入后对用户输入的每一位加1，新的中断程序如下：

NEW0AH PROC FAR

PUSHF

CALL DWORD PTR OLD\_INT\_A

PUSH CX

PUSH BX

MOV CL, INPWD+1

LEA BX, INPWD+2

ENCODE: ADD BYTE PTR [BX], 1

INC BX

DEC CL

JNZ ENCODE

POP BX

POP CX

IRET

NEW0AH ENDP

在用户输入密码之前安装新的中断程序，并在用户密码输入完毕后恢复原来的中断程序，修改后功能1的输入密码部分代码如下：

XOR AX, AX

MOV ES, AX

MOV AX, ES:[21H\*4]

MOV OLD\_INT\_A, AX

MOV AX, ES:[21H\*4+2]

MOV OLD\_INT\_A+2, AX

CLI

MOV WORD PTR ES:[21H\*4], OFFSET NEW0AH

MOV ES:[21H\*4+2], CS

STI

READ INPWD

XOR AX, AX

MOV ES, AX

CLI

MOV AX, OLD\_INT\_A

MOV ES:[21H\*4], AX

MOV AX, OLD\_INT\_A+2

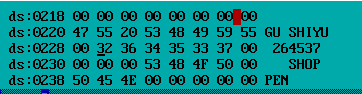
MOV ES:[21H\*4+2], AX

STI



**图4.8 正常登陆**

加密之后登陆功能可以正常使用。



**图4.9 加密后的密码在内存中显示为264537**

可以看出密码被正确加密。

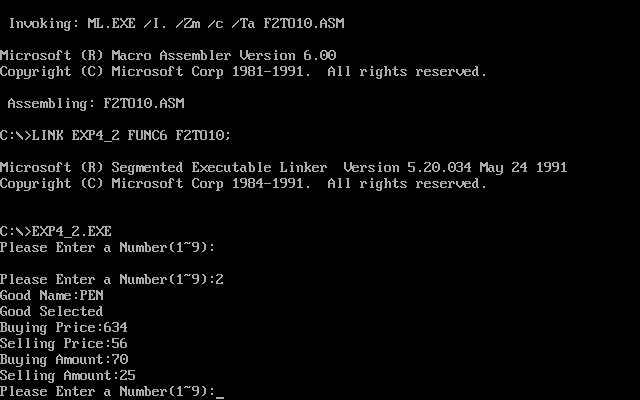
4. 将进货价储存在新的变量中，原来储存进货价的位置储存新变量的地址。



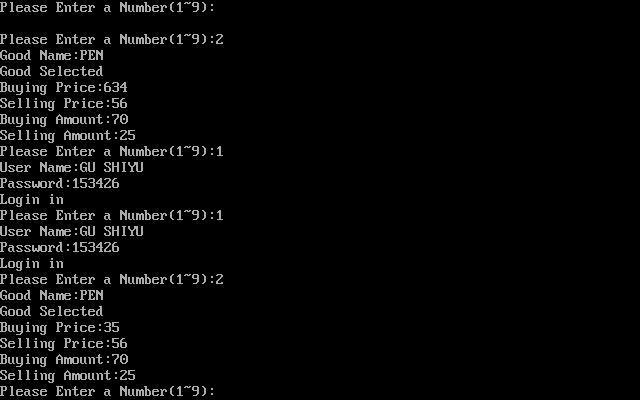
**图4.10 027A即为新变量的地址**

****

**图4.11 23H即35为真正的进货价**



**图4.12 在非老板权限下查看商品进货价为错误的进货价**



**图4.13 以老板身份登录后，进货价显示正确**

5. 在登录后加入代码，将真正的进货价放回原位。修改后的功能一登陆成功后代码如下：

MOV AUTH, 1

MOV AX, AA

MOV WORD PTR 11[GA1], AX

MOV AX, BB

MOV WORD PTR 11[GA2], AX

MOV AX, CC

MOV WORD PTR 11[GA3], AX

PRINT LOGINST

在本次实验中使用的反跟踪手段主要为中断矢量表法和时钟法，在程序一开始将1号和3号中断设为直接返回的新中断，破坏调试功能，并在一开始加入了时钟判断，如果发现两次取时钟值不同，则直接退出程序，反跟踪代码如下：

XOR AX, AX

MOV ES, AX

MOV AX, ES:[1\*4]

MOV OLD\_INT1, AX

MOV AX, ES:[1\*4+2]

MOV OLD\_INT1+2, AX

MOV AX, ES:[3\*4]

MOV OLD\_INT2, AX

MOV AX, ES:[3\*4+2]

MOV OLD\_INT2+2, AX

CLI

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

CLI

MOV AH, 2CH

INT 21H

PUSH DX

MOV AH, 2CH

INT 21H

STI

CMP DX, [ESP]

POP DX

JZ MENU

JMP OVER

在程序的最后将中断恢复，正常退出。

## 任务4.3实验过程

### 实验方法说明

1. 准备实验环境

2. 使用TD观察反汇编的代码，定位用户登录程序的位置

3. 观察用户登录程序，找出老板密码储存位置

4. 观察用户登录程序，找到解密代码

5. 破解老板密码

7. 观察反汇编代码，定位显示进货价程序位置

8. 观察显示进货价程序，找出进货价储存位置

9. 观察显示进货价程序，找到解密代码

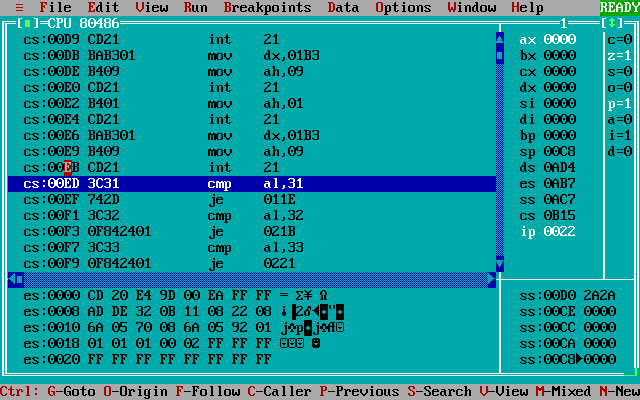
10. 破解进货价

### 实验记录与分析

本次实验与同班的吕毅东和易汉城同学组成小组完成。

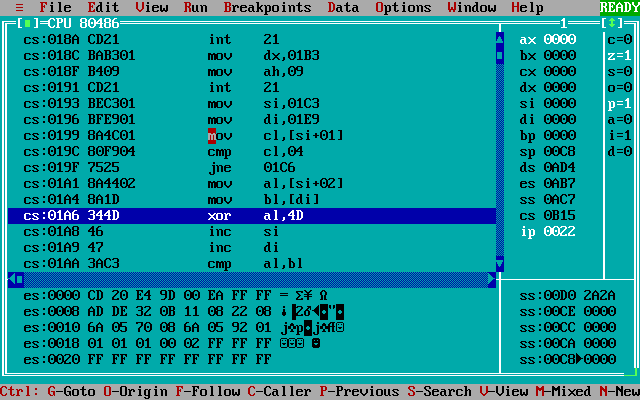
1. 准备实验环境

2. 用TD定位到功能1的代码



**图4.14 功能1代码位置为011E**

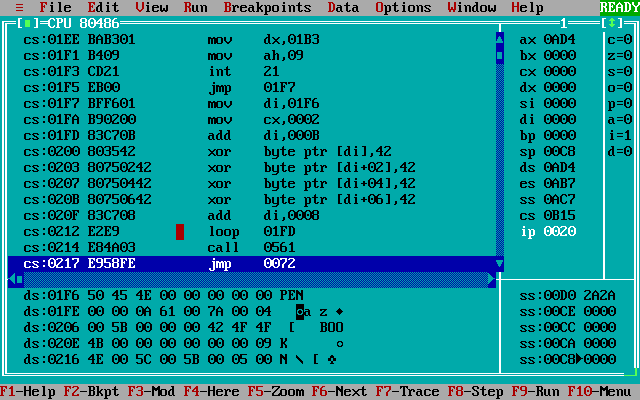
逐条观察功能1代码语句，发现在一个循环中有一句XOR AL,4D语句，推测为解密语句，密码为4DH。



**图4.15 XOR语句**

将4DH与内存中显示的密码和用户名字段进行异或，得到用户名为LVYIDONG，密码为TEST。解密成功。

2. 观察代码，找到货物储存位置为01F6，在功能一代码最后发现一系列异或语句，对01F6后的字节进行操作，且第二个操作数相同，推测为解密代码。之后对代表进货价位置的数据解密发现进货价为35，解密成功。



**图4.16 一系列XOR语句，01F6内容为商品。**

由于编写程序的同学使用了中断矢量表法阻止我进行调试，导致我只能通过逐句查看代码来猜测加密的方式和密码。如果将加密方式设计的更加复杂，将很难破解程序。

## 实验小结

### 主要收获

通过本次实验，我首先学会了如何扩展中断程序，如何安装和卸载自定义的中断程序。通过修改中断矢量表，我们可以自由的修改中断程序的功能，方便我们拓展程序的功能，同时我也认识到了保护中断矢量表的重要性，如果对中断矢量表加以修改之后不及时恢复，那么就有可能访问到非法的内存单元，导致系统死机，所以对于中断矢量表这样的系统级的内存的操作是需要十分谨慎的。其次在安装新中断程序的过程中，我也对标签的FAR类型的含义，段基址与偏移地址之间的关系有了更深刻的认识，通过灵活的指定段基址和偏移地址，我们访问同一个物理地址的方法是多种多样的，同时如果偏移地址和段基址不对应的话，我们也无法访问到正确的物理地址，所以在编写代码时需要注意默认的段基址是什么，需不需要我们显示的指定段基址，以避免访问到非法的内存单元。

其次我学会了如何加密和反跟踪，以及如何通过反汇编程序跟踪解密程序，通过大量汇编代码的编写与阅读，当我观察反汇编出来的汇编指令时，可以大致推测其功能，并尝试解密其他同学加密的数据，这也让我意识到了反跟踪的重要性，反跟踪的一大方法就是通过检查堆栈，几条代码运行的时间差，替换中断程序等方式阻止破解者使用调试程序对程序进行单步调试，失去了单步调试的破解者在定位数据和理解程序含义上都将有很大的困难，有效的反跟踪方式可以让程序破解变得困难很多。

### 4.6.2 课后问题

|  |  |  |  |
| --- | --- | --- | --- |
| 任务编号 | 序号 | 需要尝试的操作 | 结果 |
| 任务4.1  （IO） | 1 | 如何直接在TD下使用IN/OUT指令获取CMOS数据？ | 用OUT指令将想读取的内容的便宜地址送入70端口，再用IN 从71H端口读取结果 |
| 2 | CMOS里的时间信息是按照压缩BCD码的形式存放的，举例说明压缩BCD码的格式是什么？ | 用一个字节储存两个BCD码，比如12的压缩BCD码为00010010 |
| （中断矢量表） | 1 | 打开TD之后，如何在数据区切换到中断矢量表所在内存区域？ | 可以用GOTO前往0:0的内存区域 |
| 2 | 如何计算某个中断入口在中断矢量表内的偏移地址？ | 使用公式，偏移地址为es:中断号\*4，段地址为es:中断号\*4+2 |
| 3 | 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。 | 可以使用35xxH参数调用21H中断获取中断号为xx的中断入口地址 |
| 4 | 程序中如何通过直接内存访问的方法获取中断入口地址？ | 可以使用MOV AX, ES：[n\*4]和MOV AX, ES：[n\*4+2]来获取n号中断的偏移地址和段基址，其中ES为0 |
| 5 | 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） | 可能会导致系统死机 |
| 6 | 对整个中断矢量表中的入口地址进行观察，是否能看出什么特点？ | 都是2的倍数，有一些相同的中断入口地址 |
| 7 | 选择几个中断服务程序的入口地址，观察其对应的中断服务程序的代码（比如：看一下0号、2号、21号中断的处理程序干了些什么？） |  |
| （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) | PUSH F， CALL DWORF PTR OLD\_INT  和JMP DWORD PTR OLD\_INT |
| 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。 | 我编写了一个新的程序，程序的功能即为新中断程序的功能，然后进行调试发现没有问题后再放到原程序中求调试 |
| 3 | 如何判断中断服务程序已经安装过了？ | 可以设置一个变量表示是否安装过中断程序，或者将中断程序的入口地址与原入口地址比较 |
| （内存操纵 | 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？ | 要注意切换后两者的段基址改变了，但是段内变量的偏移地址是不变的，计算出来的实际物理地址改变了 |
| 2 | 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。） | 将每个段的大小都分配为4的倍数，每个段的段基址都为4的倍数 |

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？

找到密码的偏移地址，直接观察内存空间中的值

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）

先找到商品名称的位置，因为是字符串很容易辨识，再查看后几个字节的内容推断出进货价

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）

找到源程序中检测密码是否正确的代码段，在密码逐位比较时，每发现一位正确的密码就记录这位的正确值，直到每一位都正确

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）

观察反汇编代码，单步调试，推断其作用

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）

发现有改变中断矢量表的，调用获取时间功能的，检查堆栈的等代码，删除这部分代码即可。

6. 思考一下，如何用C语言（不嵌入汇编语言）实现反跟踪？是否能发现汇编语言的特殊之处? （在体会里具体描述）

C语言中也实现了操作中断程序的函数，可以通过这一点反跟踪。汇编语言更加贴近底层，可以直接操作一些高级语言无法操作的系统函数，达到一些目的

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？

使用一条指令将新的中断地址一次性送入中断矢量表，避免中途触发新的中断。

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，可以将进货价的寻址方式中加入AUTH，使得AUTH为某个值时可以正确的寻址到进货价，否则不可以。可以，可以找到计算推荐度中进货价应该出现的位置，然后反推得到进货价。

9. 接管键盘中断的方法可以用于跟踪程序，但若将接管键盘中断的方法用到反跟踪中，是否有好处？（输入密码前接管键盘中断，密码输入完毕后恢复原中断矢量。把密码字符串从明文转换成密文的转换算法写到接管后的键盘中断服务程序中，转换后的结果直接存在指定变量中，但不用破坏原键盘中断的返回值）

可以，我的源程序中就使用了这种方法

# WIN32/64编程

## 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。

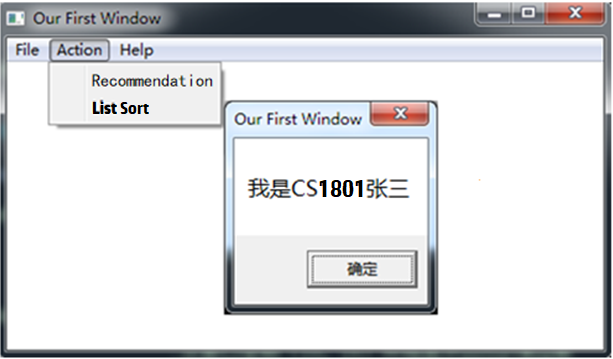
1.编写一个基于窗口的WIN32/64程序的菜单框架，具有以下的下拉菜单项：

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，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**

（不要求用中文）。

## 任务5.1实验过程

### 实验方法说明

（1）准备实验环境，安装MASM32,。

（2）使用VS2019新建一个窗口应用程序工程，添加源文件，设置编译和链接选项。

（3）编辑窗口菜单下拉选项

（4）将商品改为结构体，声明结构体变量

（5）根据32位程序的要求重新编写功能4

（6）编写排序函数

排序函数流程如下：

1. edx作为计数器1，初始置0，记录此次排序为第几趟排序

2. edc作为计数器2，初始置0，记录当前为第几件商品

3. ebx记录本趟排序推荐度最大值。edi记录最大值的商品为第几件，初始都置为0

4. 判断当前商品推荐度是否大于ebx，大于则更新ebx和edi

5. ecx++，判断ecx是否为5，不是则重复4

6. 交换第edi件商品和第edx件商品，并交换对应的输出用的商品信息

7. edx++，判断edx是否为4，不是则重复2~6

8. 结束排序

（7）在窗口中调用所需的功能

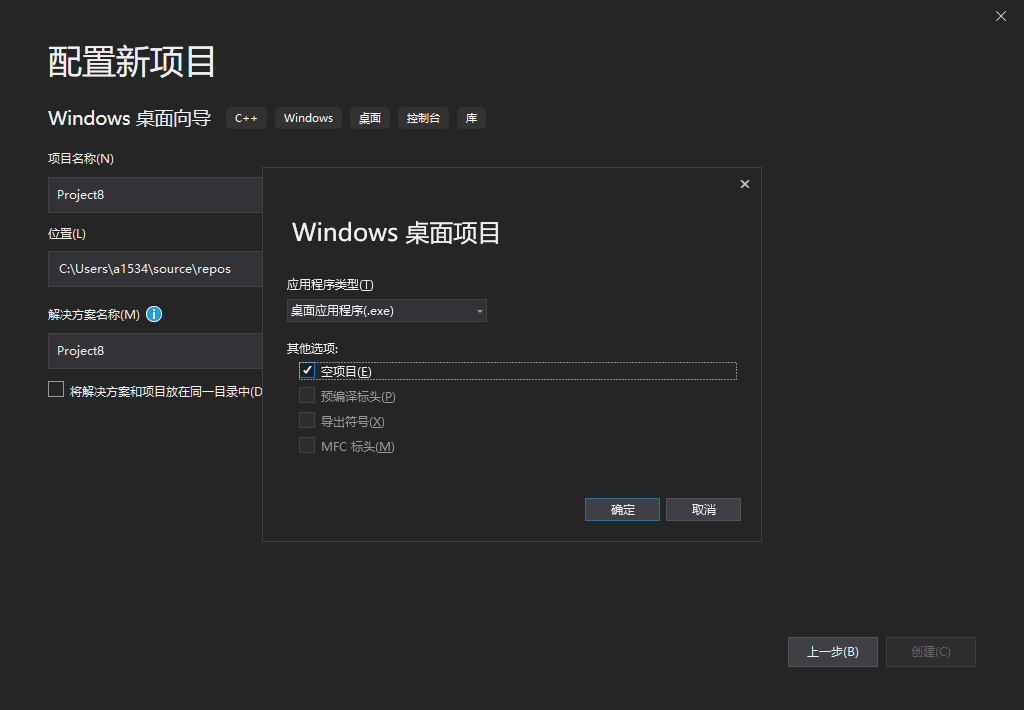
### 实验记录与分析

1. 准备实验环境，打开VS2019，新建一个Windows桌面向导项目



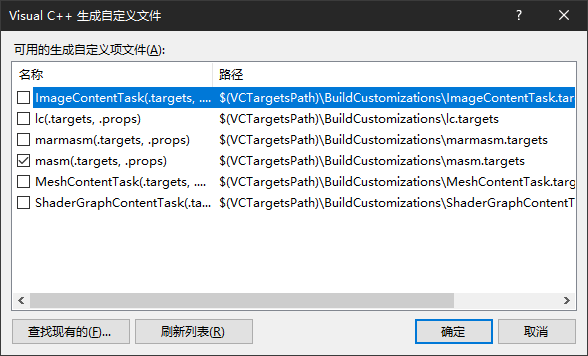
**图5.3 选择项目**

在弹出的选项框中选择桌面应用程序和空项目



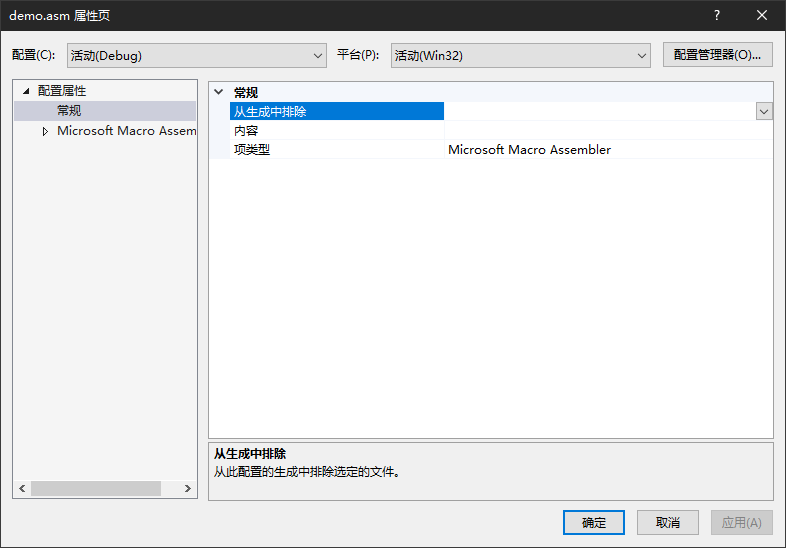
**图5.4 选择桌面应用程序**

右键项目->生成依赖项->生成自定义，勾选masm



**图5.5 生成自定义**

添加源代码文件，右键asm文件->属性->项类型，改为Microsoft Macro Assembler



**图5.6 更改项类型**

到此环境配置完成。

2. 编辑窗口菜单下拉项，使用VS打开menu.rc文件，编辑文件内容如下：

#define IDM\_FILE\_EXIT 10001

#define IDM\_ACTION\_RECO 10002

#define IDM\_ACTION\_LIST 10003

#define IDM\_HELP\_ABOUT 10101

MyMenu MENU

BEGIN

POPUP "&File"

BEGIN

MENUITEM "E&xit", IDM\_FILE\_EXIT

END

POPUP "&Action"

BEGIN

MENUITEM "R&commendation", IDM\_ACTION\_RECO

MENUITEM "L&ist Sort", IDM\_ACTION\_LIST

END

POPUP "&Help"

BEGIN

MENUITEM "&About", IDM\_HELP\_ABOUT

END

END

使用VS打开menuID.inc文件，编辑内容如下：

IDM\_FILE\_EXIT equ 10001

IDM\_ACTION\_RECO equ 10002

IDM\_ACTION\_LIST equ 10003

IDM\_HELP\_ABOUT equ 10101

3. 重新编写窗口应用程序，新的窗口应用程序如下：

.386

.model flat,stdcall

option casemap:none

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

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

Display proto :DWORD

sprintf proto C:DWORD,:DWORD,:vararg

FUNC4 proto

sort proto

include menuID.INC

include D:\Masm32\INCLUDE\windows.inc

include D:\Masm32\INCLUDE\user32.inc

include D:\Masm32\INCLUDE\kernel32.inc

include D:\Masm32\INCLUDE\gdi32.inc

include D:\Masm32\INCLUDE\shell32.inc

includelib user32.lib

includelib kernel32.lib

includelib gdi32.lib

includelib shell32.lib

good struct

gname db 10 dup(0)

off db 0

inprice dw 0

outprice dw 0

inamount dw 0

outamount dw 0

prior dw 0

good ends

len struct

os db 10 dup(0)

ips db 10 dup(0)

ops db 10 dup(0)

ias db 10 dup(0)

oas db 10 dup(0)

ps db 10 dup(0)

nl db 0

ol db 0

ipl db 0

opl db 0

ial db 0

oal db 0

pl db 0

len ends

.data

ClassName db 'TryWinClass',0

AppName db 'Our First Window',0

MenuName db 'MyMenu',0

DlgName db 'MyDialog',0

AboutMsg db 'I am CS1808 GuShiyu',0

RecoMsg db 'ComputeOver', 0

hInstance dd 0

CommandLine dd 0

msg\_name db 'name',0

msg\_off db 'Off', 0

msg\_inprice db 'BuyingPrice',0

msg\_outprice db 'SellingPrice',0

msg\_inamount db 'BuyingAmount',0

msg\_outamount db 'SellingAmount',0

msg\_prior db 'Priority',0

menuItem db 0 ;当前菜单状态, 1=处于list, 0=Clear

GA good <'PEN', 10, 35, 56, 70, 25, ?>

good <'BOOK', 9, 12, 30, 25, 5, ?>

good <'a', 10, 20, 30, 40, 50, ?>

good <'b', 11, 22, 33, 44, 55, ?>

good <'c', 111, 222, 333, 444, 555, ?>

N equ 5

GAL len <'10','35','56','70','25','102',3,2,2,2,2,2,3>

len <'9','12','30','25','5','63',4,1,2,2,2,1,2>

len <'10','20','30','40','50','165',1,2,2,2,2,2,3>

len <'11','22','33','44','55','165',1,2,2,2,2,2,3>

len <'111','222','333','444','555','165',1,3,3,3,3,3,3>

.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

LOCAL ps:PAINTSTRUCT

.IF uMsg == WM\_DESTROY

invoke PostQuitMessage,NULL

.ELSEIF uMsg == WM\_KEYDOWN

.IF wParam == VK\_F1

;;your code

.ENDIF

.ELSEIF uMsg == WM\_COMMAND

.IF wParam == IDM\_FILE\_EXIT

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

.ELSEIF wParam == IDM\_ACTION\_RECO

invoke FUNC4

invoke MessageBox,hWnd,addr RecoMsg,addr AppName,0

.ELSEIF wParam == IDM\_ACTION\_LIST

mov menuItem, 1

invoke InvalidateRect,hWnd,0,1 ;擦除整个客户区

invoke UpdateWindow, hWnd

.ELSEIF wParam == IDM\_HELP\_ABOUT

invoke MessageBox,hWnd,addr AboutMsg,addr AppName,0

.ENDIF

.ELSEIF uMsg == WM\_PAINT

invoke BeginPaint,hWnd, addr ps

mov hdc,eax

.IF menuItem == 1

invoke Display,hdc

.ENDIF

invoke EndPaint,hWnd,addr ps

.ELSE

invoke DefWindowProc,hWnd,uMsg,wParam,lParam

ret

.ENDIF

xor eax,eax

ret

WndProc endp

Display proc hdc:HDC

XX equ 10

YY equ 10

XX\_GAP equ 100

YY\_GAP equ 30

invoke sort

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\_off,3

invoke TextOut,hdc,XX+2\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_inprice,11

invoke TextOut,hdc,XX+3\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_outprice,12

invoke TextOut,hdc,XX+4\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_inamount,12

invoke TextOut,hdc,XX+5\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_outamount,13

invoke TextOut,hdc,XX+6\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_prior,8

;;

invoke TextOut,hdc,XX+0\*XX\_GAP,YY+1\*YY\_GAP,offset GA[0\*21].gname, GAL[0\*67].nl

invoke TextOut,hdc,XX+1\*XX\_GAP,YY+1\*YY\_GAP,offset GAL[0\*67].os, GAL[0\*67].ol

invoke TextOut,hdc,XX+2\*XX\_GAP,YY+1\*YY\_GAP,offset GAL[0\*67].ips, GAL[0\*67].ipl

invoke TextOut,hdc,XX+3\*XX\_GAP,YY+1\*YY\_GAP,offset GAL[0\*67].ops, GAL[0\*67].opl

invoke TextOut,hdc,XX+4\*XX\_GAP,YY+1\*YY\_GAP,offset GAL[0\*67].ias, GAL[0\*67].ial

invoke TextOut,hdc,XX+5\*XX\_GAP,YY+1\*YY\_GAP,offset GAL[0\*67].oas, GAL[0\*67].oal

invoke TextOut,hdc,XX+6\*XX\_GAP,YY+1\*YY\_GAP,offset GAL[0\*67].ps, GAL[0\*67].pl

;;

invoke TextOut,hdc,XX+0\*XX\_GAP,YY+2\*YY\_GAP,offset GA[1\*21].gname, GAL[1\*67].nl

invoke TextOut,hdc,XX+1\*XX\_GAP,YY+2\*YY\_GAP,offset GAL[1\*67].os, GAL[1\*67].ol

invoke TextOut,hdc,XX+2\*XX\_GAP,YY+2\*YY\_GAP,offset GAL[1\*67].ips, GAL[1\*67].ipl

invoke TextOut,hdc,XX+3\*XX\_GAP,YY+2\*YY\_GAP,offset GAL[1\*67].ops, GAL[1\*67].opl

invoke TextOut,hdc,XX+4\*XX\_GAP,YY+2\*YY\_GAP,offset GAL[1\*67].ias, GAL[1\*67].ial

invoke TextOut,hdc,XX+5\*XX\_GAP,YY+2\*YY\_GAP,offset GAL[1\*67].oas, GAL[1\*67].oal

invoke TextOut,hdc,XX+6\*XX\_GAP,YY+2\*YY\_GAP,offset GAL[1\*67].ps, GAL[1\*67].pl

;;

invoke TextOut,hdc,XX+0\*XX\_GAP,YY+3\*YY\_GAP,offset GA[2\*21].gname, GAL[2\*67].nl

invoke TextOut,hdc,XX+1\*XX\_GAP,YY+3\*YY\_GAP,offset GAL[2\*67].os, GAL[2\*67].ol

invoke TextOut,hdc,XX+2\*XX\_GAP,YY+3\*YY\_GAP,offset GAL[2\*67].ips, GAL[2\*67].ipl

invoke TextOut,hdc,XX+3\*XX\_GAP,YY+3\*YY\_GAP,offset GAL[2\*67].ops, GAL[2\*67].opl

invoke TextOut,hdc,XX+4\*XX\_GAP,YY+3\*YY\_GAP,offset GAL[2\*67].ias, GAL[2\*67].ial

invoke TextOut,hdc,XX+5\*XX\_GAP,YY+3\*YY\_GAP,offset GAL[2\*67].oas, GAL[2\*67].oal

invoke TextOut,hdc,XX+6\*XX\_GAP,YY+3\*YY\_GAP,offset GAL[2\*67].ps, GAL[2\*67].pl

;;

invoke TextOut,hdc,XX+0\*XX\_GAP,YY+4\*YY\_GAP,offset GA[3\*21].gname, GAL[3\*67].nl

invoke TextOut,hdc,XX+1\*XX\_GAP,YY+4\*YY\_GAP,offset GAL[3\*67].os, GAL[3\*67].ol

invoke TextOut,hdc,XX+2\*XX\_GAP,YY+4\*YY\_GAP,offset GAL[3\*67].ips, GAL[3\*67].ipl

invoke TextOut,hdc,XX+3\*XX\_GAP,YY+4\*YY\_GAP,offset GAL[3\*67].ops, GAL[3\*67].opl

invoke TextOut,hdc,XX+4\*XX\_GAP,YY+4\*YY\_GAP,offset GAL[3\*67].ias, GAL[3\*67].ial

invoke TextOut,hdc,XX+5\*XX\_GAP,YY+4\*YY\_GAP,offset GAL[3\*67].oas, GAL[3\*67].oal

invoke TextOut,hdc,XX+6\*XX\_GAP,YY+4\*YY\_GAP,offset GAL[3\*67].ps, GAL[3\*67].pl

;;

invoke TextOut,hdc,XX+0\*XX\_GAP,YY+5\*YY\_GAP,offset GA[4\*21].gname, GAL[4\*67].nl

invoke TextOut,hdc,XX+1\*XX\_GAP,YY+5\*YY\_GAP,offset GAL[4\*67].os, GAL[4\*67].ol

invoke TextOut,hdc,XX+2\*XX\_GAP,YY+5\*YY\_GAP,offset GAL[4\*67].ips, GAL[4\*67].ipl

invoke TextOut,hdc,XX+3\*XX\_GAP,YY+5\*YY\_GAP,offset GAL[4\*67].ops, GAL[4\*67].opl

invoke TextOut,hdc,XX+4\*XX\_GAP,YY+5\*YY\_GAP,offset GAL[4\*67].ias, GAL[4\*67].ial

invoke TextOut,hdc,XX+5\*XX\_GAP,YY+5\*YY\_GAP,offset GAL[4\*67].oas, GAL[4\*67].oal

invoke TextOut,hdc,XX+6\*XX\_GAP,YY+5\*YY\_GAP,offset GAL[4\*67].ps, GAL[4\*67].pl

ret

Display endp

FUNC4 PROC ;功能4

PUSH BP

MOV BP, -1

COMPUTE\_PRIO:

INC BP

CMP BP, N

JE COMPUTE\_OVER

LEA ESI, GA

MOVZX ECX, BP

IMUL ECX, 21

ADD ESI, ECX

MOV EBX, ESI

ADD EBX, 13

MOV AX, [EBX-2]

MOV BX, [EBX]

MOVZX EAX, AX

MOVZX EBX, BX

MOV EDI, 128

MUL EDI

DIV EBX

MOV ECX, EAX

MOV EBX, ESI

ADD EBX, 15

MOV AX, [EBX+2]

MOV BX, [EBX]

MOVZX EAX, AX

MOVZX EBX, BX

IMUL EDI

MOV EDI, 2

DIV EDI

MOV EDX, 0

DIV EBX

ADD EAX, ECX

MOV EBX, ESI

ADD EBX, 19

MOV [EBX], AX

JMP COMPUTE\_PRIO

COMPUTE\_OVER:

POP BP

RET

FUNC4 ENDP

sort proc

mov edx, 0

l1:

mov ecx, 0

add ecx, edx

mov ebx, 0

mov edi, 0

l2:

mov esi, ecx

imul esi, 21

add esi, 19

cmp bx, word ptr GA[esi]

jnb nm

mov bx, word ptr GA[esi]

sub esi, 19

mov edi, ecx

nm:

inc cx

cmp cx, 5

jne l2

push ecx

mov ecx,0

mov esi, edx

imul esi, 21

mov eax, edi

imul eax, 21

l3:

mov bh, byte ptr GA[eax+ecx]

mov bl, byte ptr GA[esi+ecx]

mov byte ptr GA[esi+ecx], bh

mov byte ptr GA[eax+ecx], bl

inc ecx

cmp ecx, 21

jne l3

mov ecx, 0

mov esi, edx

imul esi, 67

mov eax, edi

imul eax, 67

l4:

mov bh, byte ptr GAL[eax+ecx]

mov bl, byte ptr GAL[esi+ecx]

mov byte ptr GAL[esi+ecx], bh

mov byte ptr GAL[eax+ecx], bl

inc ecx

cmp ecx, 67

jne l4

pop ecx

inc edx

cmp edx, 4

jne l1

ret

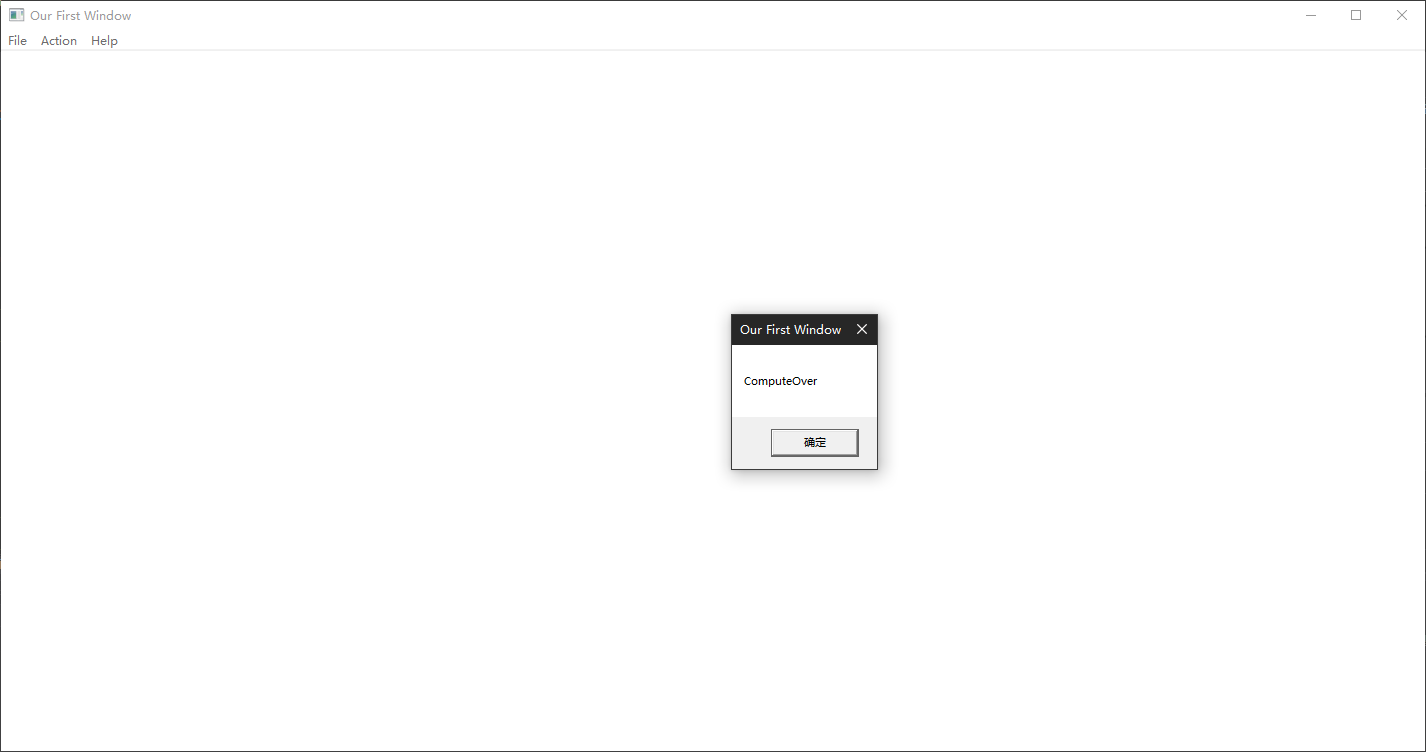
sort endp

end Start

新的窗口应用程序，在点击File->Exit时会退出程序，点击Action->Recommendation时弹出推荐度计算完毕提示框并计算推荐度，点击Action->List Sort会将商品按照推荐度由大到小排序，并打印到窗口中。点击Help->About弹出学生自我介绍窗口。

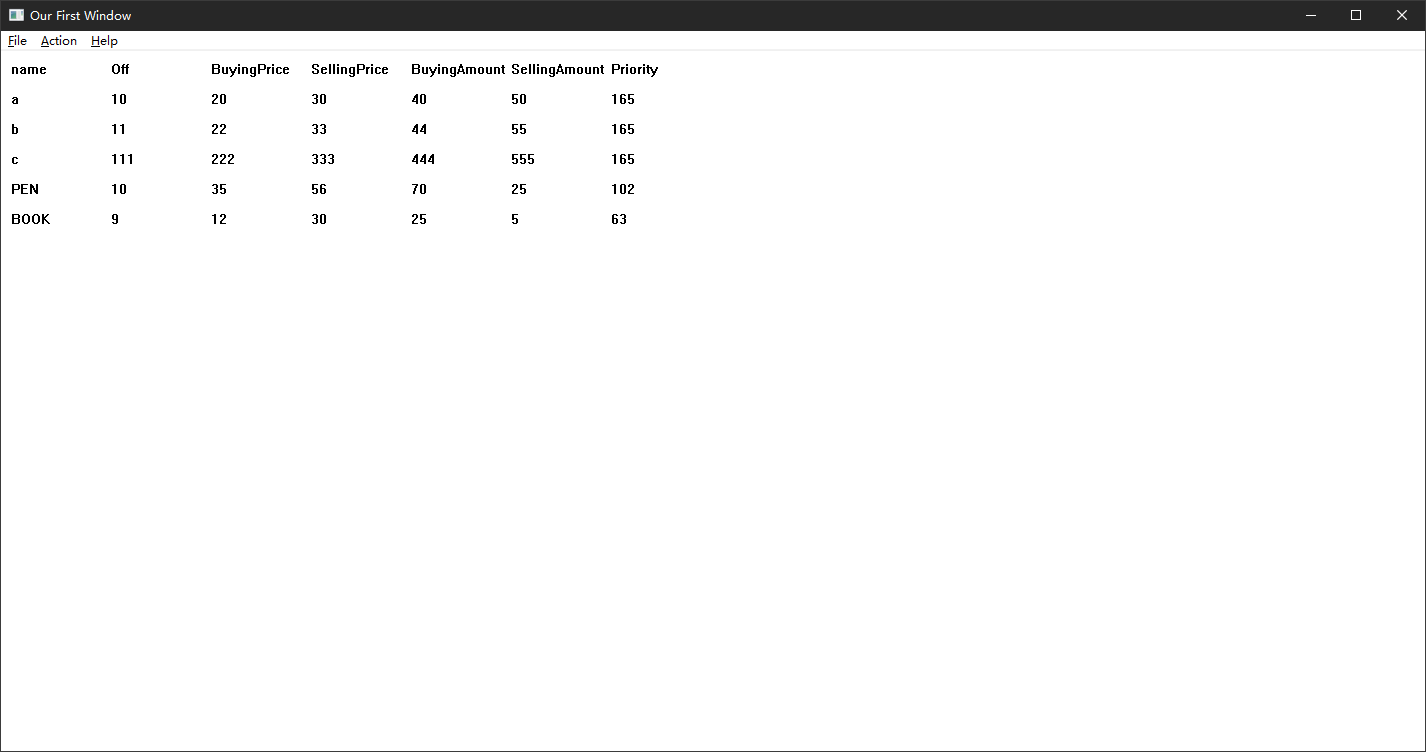
3. 测试程序功能

测试推荐度计算功能，提示推荐度计算完毕。



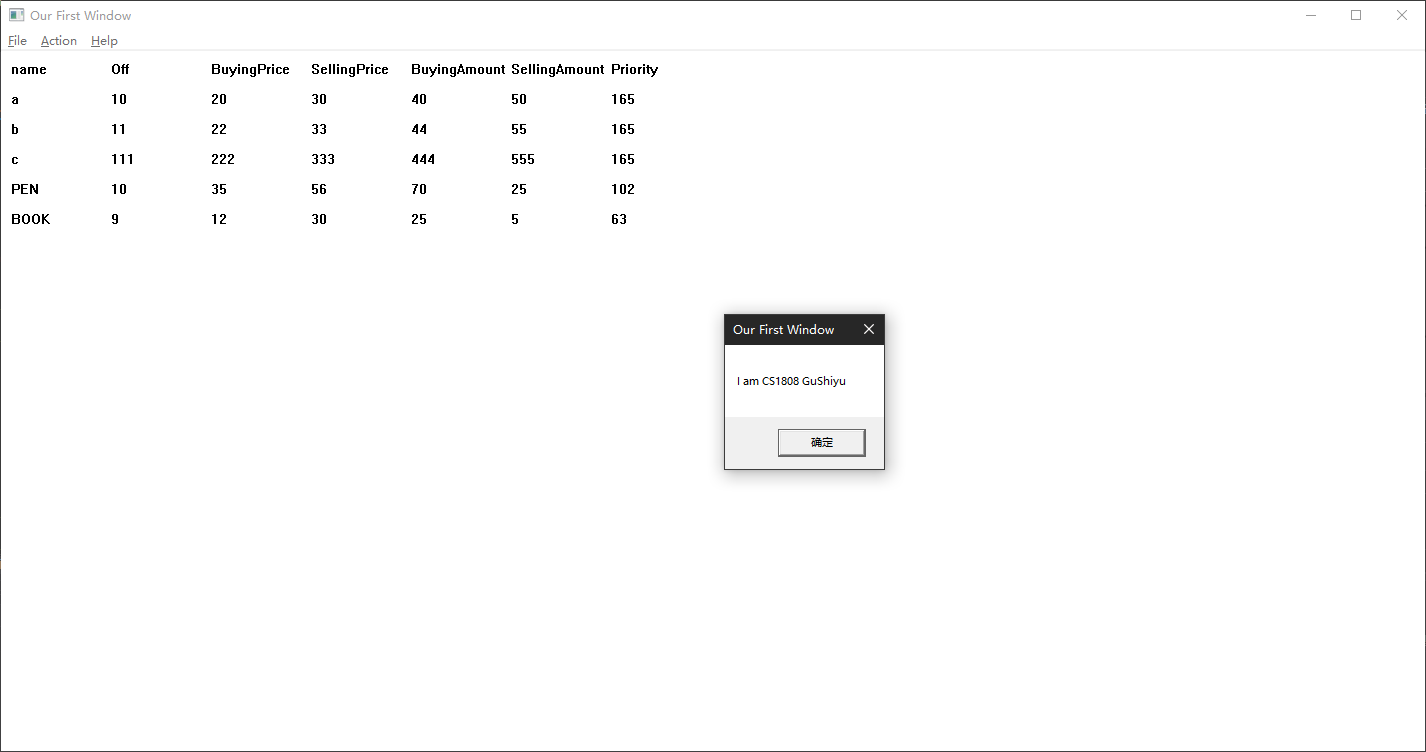
**图5.7 点击Recommendation弹出计算完毕消息框**

测试排序功能，按照推荐度从高到低排序输出信息



**图5.8 点击List Sort排序并打印**

测试About，正常弹出提示框



**图5.9 点击About弹出消息框**

可以看出程序功能全部正常。

本次实验的程序编写部分并不复杂，在已经提供了窗口程序框架的情况下，更改添加功能的过程并不困难，编写代码遇到的问题主要有两个，一是将16位程序移植到32位程序中时，已经不能再用16位寄存器储存变量的地址，需要将这些寄存器都改为32位的才能正常运行。其次是输出数字的问题，由于TextOut函数只能输出字符，不能输出整数。所以需要预先将输出内容变为字符串，并在排序时除了交换商品本身以外，还要交换储存输出信息的变量。我曾尝试使用sprintf函数来将整数转换为字符串，但是却难以得到转换了的字符串的长度，反而不如现在使用的方法简便，所以最后仍然使用了现在的方案。

## 实验小结

### 主要收获

本次实验内容为WIN32程序的设计。通过这次实验我首次接触了窗口应用程序的编写，了解到了WIN32程序开发环境的配置，程序开发流程以及Windows API的使用方法。通过学习如何使用VS2019编写汇编语言程序并进行源码级调试，我体会到了使用现代工具开发程序的便捷，与TD提供的机器码级调试不同，源码级调试可以保留源码原本的结构，保留伪指令和变量名，方便我们调试，不用再通过反汇编代码来寻找想要调试的语句，大大提高了代码编写的效率。

通过编写窗口程序，我深入了解了WIN32窗口程序编写的重点在于消息响应程序的编写，通过控制程序对不同输入消息的反馈就可以实现想要的程序功能，使得WIN32窗口程序有良好的扩展性和自定义性。

### 课后问题

1. 观察32/64位下调试工具与16位TD的异同。

32位TD的寄存器和堆栈区以及数据区的地址显示都是32位，16位TD显示的都是16位

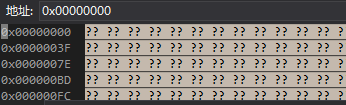
2. 调试WIN32/64程序与16位段程序的主要差异是什么？

地址都为32位，并非16位

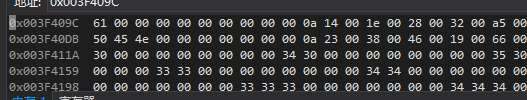
3. 尝试使用一下汇编语言程序的源码级调试工具和方法，与非源码级调试做个对比。

源码级调试工具会保留伪指令和变量名，方便我们调试，不会破坏程序的结构，非源码级调试可以精确到每一条机器指令，更佳的细致，也可以帮助我们观察伪指令到机器代码的转换关系。

4. 用调试工具观察代码区或数据区时，若所观察的地址范围不是与被调试程序相关的区间，则对应内存中的数据会因为被系统保护了而读不出来（将用“？”或其他符号代替）。请通过修改偏移地址来改变观察的区间，记录此现象。



偏移地址为0的区域不可查看



储存变量的区域正常显示

5. 观察结构变量的推荐度等字段的偏移，体会结构变量优点。

结构变量可以使用结构成员名方便的代替偏移地址，省去记忆每个变量偏移地址的消耗。

6. 观察简化段的效果。

简化段相当于使用默认参数声明了段

7. 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。

参数从右到左的顺序压栈，在系统函数内使用ebp+偏移量的形式来访问各个参数