

**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ACM1901班**

**学 号： U201915035**

**姓 名： 邹雅**

**指导教师： 曹忠升**

**实验时段： 2021年3月29日~5月21日**

**实验地点： 南一楼808室**

**原创性声明**

  本人郑重声明：本报告的内容由本人独立完成，有关观点、方法、数据和文献等的引用已经在文中指出。除文中已经注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品或成果，不存在剽窃、抄袭行为。

特此声明！

学生签名：

报告日期：2021.6.7

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 - 7 -](#_Toc74073222)

[0.1 课程目标 - 7 -](#_Toc74073223)

[0.2 成绩构成 - 7 -](#_Toc74073224)

[0.3 实验任务的总体描述 - 7 -](#_Toc74073225)

[1 编程基础 1](#_Toc74073226)

[1.1 实验目的与要求 1](#_Toc74073227)

[1.2 实验内容 1](#_Toc74073228)

[1.3 任务1.1实验过程 6](#_Toc74073286)

[1.3.1 实验方法说明 6](#_Toc74073287)

[1.3.2 实验记录与分析 6](#_Toc74073288)

[1.4 任务1.2的实验过程 8](#_Toc74073289)

[1.4.1 实验方法说明 8](#_Toc74073290)

[1.4.2 实验记录与分析 9](#_Toc74073291)

[1.5 任务1.3的实验过程 10](#_Toc74073292)

[1.5.1 设计思想及存储单元分配 10](#_Toc74073293)

[1.5.2 流程图 11](#_Toc74073294)

[1.5.3 实验步骤 11](#_Toc74073295)

[1.5.4 实验记录与分析 11](#_Toc74073296)

[1.6 小结 15](#_Toc74073297)

[1.6.1 主要收获 15](#_Toc74073298)

[1.6.2 主要看法 15](#_Toc74073299)

[2 程序优化 17](#_Toc74073300)

[2.1 实验目的与要求 17](#_Toc74073301)

[2.2 实验内容 17](#_Toc74073302)

[2.3 任务2.1实验过程 17](#_Toc74073303)

[2.3.1 实验方法说明 17](#_Toc74073304)

[2.3.2 实验记录与分析 18](#_Toc74073305)

[2.4 小结 20](#_Toc74073306)

[2.4.1 主要收获 20](#_Toc74073307)

[2.4.2 主要看法 20](#_Toc74073308)

[3 模块化程序设计 21](#_Toc74073309)

[3.1 实验目的和要求 21](#_Toc74073310)

[3.2 实验内容 21](#_Toc74073316)

[3.3 任务3.1实验过程 22](#_Toc74073347)

[3.3.1 设计思想及存储单元分配 22](#_Toc74073348)

[3.3.2 流程图 24](#_Toc74073349)

[3.3.3 实验步骤 24](#_Toc74073350)

[3.3.4 实验记录与分析 24](#_Toc74073351)

[3.4 任务3.2实验过程 27](#_Toc74073352)

[3.4.1 实验方法说明 27](#_Toc74073353)

[3.4.2 实验记录与分析 27](#_Toc74073364)

[3.5 小结 31](#_Toc74073365)

[3.5.1 主要收获 31](#_Toc74073366)

[3.5.2 主要看法 31](#_Toc74073367)

[4 中断与反跟踪 32](#_Toc74073368)

[4.1 实验目的和要求 32](#_Toc74073369)

[4.2 实验内容 32](#_Toc74073376)

[4.3 任务4.1实验过程 32](#_Toc74073380)

[4.3.1 设计思想及存储单元分配 32](#_Toc74073381)

[4.3.2 流程图 33](#_Toc74073382)

[4.3.3 源程序 33](#_Toc74073383)

[4.3.4 实验步骤 34](#_Toc74073384)

[4.3.5 实验记录与分析 34](#_Toc74073385)

[4.4 任务4.2实验过程 36](#_Toc74073386)

[4.4.1 实验方法说明 36](#_Toc74073387)

[4.4.2 实验记录与分析 36](#_Toc74073388)

[4.5 任务4.3实验过程 38](#_Toc74073389)

[4.5.1 实验方法说明 38](#_Toc74073390)

[4.5.2 实验记录与分析 38](#_Toc74073391)

[4.6 小结 40](#_Toc74073392)

[4.6.1 主要收获 40](#_Toc74073393)

[4.6.2 主要看法 41](#_Toc74073394)

[5 16/32/64位编程比较 42](#_Toc74073395)

[5.1 实验目的和要求 42](#_Toc74073396)

[5.2 实验内容 42](#_Toc74073397)

[5.3 任务5.1实验过程 42](#_Toc74073401)

[5.3.1 实验方法说明 42](#_Toc74073402)

[5.3.2 实验记录与分析 42](#_Toc74073403)

[5.4 任务5.2实验过程 44](#_Toc74073404)

[5.4.1 实验方法说明 44](#_Toc74073405)

[5.4.2 实验记录与分析 44](#_Toc74073406)

[5.5 任务5.3实验过程 45](#_Toc74073407)

[5.5.1 实验方法说明 45](#_Toc74073408)

[5.5.2 实验记录与分析 45](#_Toc74073409)

[5.6 小结 47](#_Toc74073410)

[5.6.1 主要收获 47](#_Toc74073411)

[5.6.2 主要看法 47](#_Toc74073412)

[参考文献 49](#_Toc74073413)

# 课程总体说明

## 0.1 课程目标

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

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

## 0.2 成绩构成

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

## 0.3 实验任务的总体描述

本课程安排了8次4学时的课内实验课时，将实现一个具有一定复杂程度的系统。对该系统的相关要求被划分成了**5个主题**：1）搭建原型系统；2）在原型系统基础上探索程序指令级别的优化；3）通过模块化方法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实现。

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（16/32/64位编程比较）**安排4个课内学时，熟悉在不同软硬件平台上移植实现指定功能的基本方法。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。

# 编程基础

## 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)熟练掌握程序开发平台(VS2019) 的基本用法，包括程序的编译、链接和调试；掌握DOSBox下16位汇编语言程序开发工具的基本用法；

(2)熟悉编程的基础知识，包括数据在计算机内的表现形式、寻址方式、常用指令等；

(3) 熟悉程序运行的基本原理；

(4) 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；

(5) 加深对转移指令及一些常用的汇编指令的理解；

(6) 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 从C语言到汇编语言**

对于下列给定的C语言程序，使用VS2019进行编译、链接和调试。通过实验，回答如下问题：

（1） 显示反汇编窗口，了解C语言与汇编语句的对应关系。在反汇编窗口中的“查看选项”下有“显示符号名”，指出勾选与不勾该项选时，反汇编窗口显示内容的差异；

（2）显示寄存器窗口。在该窗口中设置显示 寄存器、段寄存器、标志寄存器等；

（3）显示监视窗口，观察变量的值；显示内存窗口，观察变量的值（整型值、字符串等）在内存中的具体表现细节。

（4）有符号与无符号整型数是如何存储的；

（5）有符号数和无符号数的加减运算有无差别，是如何执行的？执行加法运算指令时，标志寄存器是如何设置的？执行比较指令时又有什么差异？

#include <stdio.h>

int a[5] = { 1, 2, 3, 4, 5 };

short x = 100;

short y = -32700; //注意观察初始值较大带来的问题

int psub;

int sum(int a[], unsigned length)

{

int i;

int result = 0;

for (i = 0; i < length ; i++)

result += a[i];

return result;

}

int main(int argc, char\* argv[])

{

short z;

char str[10] = "The end!";

z = sum(a, 5);

printf("sum : %d \n", z);

if (x > y)

printf("condition1: %d > %d \n", x, y);

else

printf("condition1: %d < %d \n", x, y);

z = x - y;

printf("condition2: (%d) - (%d) = %d \n", x, y, z);

psub= &x - &y;

if (psub< 0)

printf("condition3: & %d < & %d \n", x, y);

printf(str);

return 0;

}

**任务1.2 观察汇编语言程序**

对下列汇编语言源程序（其功能是：定义了一个数据段，并用指定的内存寻址方式，将buf1缓冲区中的12个字节内容拷贝到 buf2中,并显示两个缓冲区中的字符串），完成下列要求：

（1）使用VS2019进行编译、链接和调试。完成反汇编窗口显示，了解汇编源程序中的语句与反汇编语句之间的关系。同时，完成同任务1.1的寄存器窗口、监视窗口、内存窗口的操作。观察数据段的存储结果，说明存储规律，说明各个变量地址之间的关系。如何观察堆栈段？尝试将访问buf1的寻址方式由寄存器间接寻址方式改成其他的寻址方式。

给定的汇编语言源程序如下：

.386

.model flat, stdcall

ExitProcess PROTO STDCALL :DWORD

includelib kernel32.lib ; ExitProcess 在 kernel32.lib中实现

printf PROTO C :VARARG

includelib libcmt.lib

includelib legacy\_stdio\_definitions.lib

.DATA

lpFmt db "%s",0ah, 0dh, 0

X DB 10, 255, -1

Y DW 10, 255, -1

Z DD 10, 255, -1

U DW ($-Z)/4

STR1 DB 'Good', 0

P DD X, Y

Q DB 2 DUP (5, 6)

buf1 db '00123456789',0

buf2 db 12 dup(0) ; 12个字节的空间，初值均为 0

.STACK 200

.CODE

main proc c

MOV ESI,OFFSET buf1

MOV EDI,OFFSET buf2

MOV ECX,0

L1:

MOV EAX, [ESI] ;如果总数不是12个字节，还能每次传送4个字节吗？

MOV [EDI],EAX

ADD ESI, 4

ADD EDI, 4

ADD ECX, 4

CMP ECX,12

JNZ L1

invoke printf,offset lpFmt,OFFSET buf1

invoke printf,offset lpFmt,OFFSET buf2

invoke ExitProcess, 0

main endp

END

（2）在DOSBox下使用“MASM 6.0，LINK.EXE，TD.EXE, 完整段定义”改造上述汇编语言程序（源程序可以采用记事本等编辑），体会16位段程序的编译、链接和调试过程。尝试在调试状态下直接录入或修改代码。

.386

DATA SEGMENT USE16

lpFmt db 0ah, 0dh, "$"

X DB 10, 255, -1

Y DW 10, 255, -1

Z DD 10, 255, -1

U DW ($-Z)/4

STR1 DB 'Good', 0

P DD X, Y

Q DB 2 DUP (5, 6)

buf1 DB '00123456789','$' ;结束符号为$

buf2 DB 12 dup(0)

DATA ENDS

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

CODE SEGMENT USE16

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

START: MOV AX,DATA

MOV DS,AX

MOV ESI,OFFSET buf1

MOV EDI,OFFSET buf2

MOV ECX,0

L1:

MOV EAX, [ESI]

MOV [EDI],EAX

ADD ESI, 4

ADD EDI, 4

ADD ECX, 4

CMP ECX,12

JNZ L1

MOV DX, OFFSET buf1 ;采用DOS功能调用显示字符串

MOV AH,9

INT 21H

MOV DX, OFFSET lpFmt

MOV AH,9

INT 21H

MOV DX, OFFSET buf2

MOV AH,9

INT 21H

MOV AH,4CH

INT 21H

CODE ENDS

END START

**任务1.3 设计实现一个网店商品信息后台管理系统**

有一个老板在网上开了一个网店，通过后台管理系统管理相关信息。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）、进货价(字类型)、销售价（字类型）、进货数量（字类型）、已售数量（字类型）、利润率（%）【=（销售价\*已售数量-进货价\*进货数量）\*100/（进货价\*进货数量），字类型（有符号数）】。老板管理网店信息时需要输入自己的名字和密码，老板登录后可查看指定商品的信息、出货（出售指定数量的某种商品）、补货（给某种商品增加一定的进货数量）、计算商品的利润率，按利润率从高到低显示商品信息等。

该系统被执行后，首先提示输入用户名（即老板名称）和密码，在用户名和密码正确时，显示一个主菜单界面。当用户名和密码错误时，显示用户名错误或者密码错误的提示信息后，退出系统。主菜单界面信息包括：

请输入数字1…9选择功能：

1.查找指定商品并显示其信息

2.出货

3.补货

4.计算商品的利润率

5.按利润率从高到低显示商品的信息（本次暂不实现）

9.退出

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 ‘ZHANGSAN’,0 ；老板姓名（要求必须是自己名字的拼音）

BPASS DB ‘U20190001’，0 ；密码（必须是自己的学号）

N EQU 30

GA1 DB ‘PEN’, 7 DUP(0) ；商品1 名称

DW 15，20，70，25，？ ； 进货价、销售价、进货数量、已售数量，利润率（尚未计算）

GA2 DB ‘PENCIL’, 4 DUP(0) ；商品2 名称

DW 2，3，100，50，？

GA3 DB ‘BOOK’, 6 DUP(0) ；商品3 名称

DW 30，40，25，5，？

GA4 DW ‘RULER’,5 DUP(0) ；商品4 名称

DW 3，4，200，150，？

GAN DB N-4 DUP( ‘TempValue’ ,0,15,0,20,0，30，0，2，0，？，？) ;除了4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，在VS2019下实现该系统的部分功能，并熟悉全周期、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：

***查找指定商品并显示其信息：***提示用户输入商品名称；用户输入名称后，在商店中寻找是否存在该商品；若存在，显示找到的商品信息；若没有找到**，提示没**有找到。最后都返回到主菜单界面。

***出货：***输入商品的名称及本次销售数量。判断输入数据的有效性，即剩余数量应大于等于本次销售数量，修改商品的已售数量。若商品未找到或数据无效，则提示错误。最后都返回到主菜单界面。

***补货：***输入商品的名称及本次增加的数量。找到该商品，修改商品的进货数量。若商品未找到，则提示错误。最后都返回到主菜单界面。

***计算商品的利润率：***按照利润率计算公式依次计算所有商品的利润率。

## 任务1.1实验过程

### 实验方法说明

1. 准备上机实验环境，利用VMware Fusion搭建Win10环境，进行VS2019环境的安装、运行，参考书上第19章并且通过试用初步了解软件的基本功能、操作等。

2. 将程序载入VS2019，调试编译，使用反汇编窗口、寄存器窗口、监视窗口完成前三项任务并记录回答问题。

3．深入探讨内部底层结构，思考有符号和无符号整型数是如何存储的。

4. 尝试按照自己想的其他语句及输入格式等进行操作，积累更多的经验。

### 实验记录与分析

1. 实验环境条件：2.3 GHz Quad-Core Intel Core i5，8 GB内存；虚拟机环境下WINDOWS 10 VS2019。

2. 显示反汇编窗口，可以看到对应每一句源代码而生成的汇编语句。观察阅读每一句C语言语句，当转化成汇编当勾选“显示符号名”时，从图1.1.1可以看到，对应每个操作数地址都显示了地址的符号表示，如对于当前运行时的地址0A9A018h处是y变量。当不勾选“显示符号名”时，如图1.1.2所示，可见只显示了操作数的物理地址，而没有显示地址的符号名。

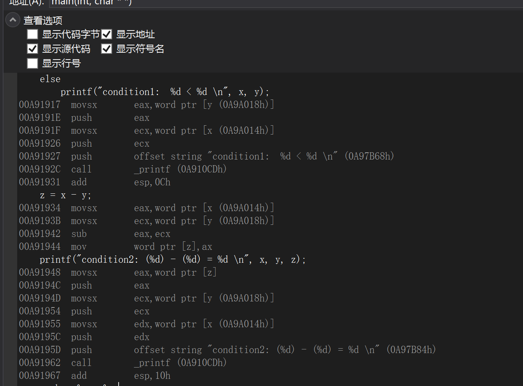


图1.1.1 反汇编勾选“显示符号名”

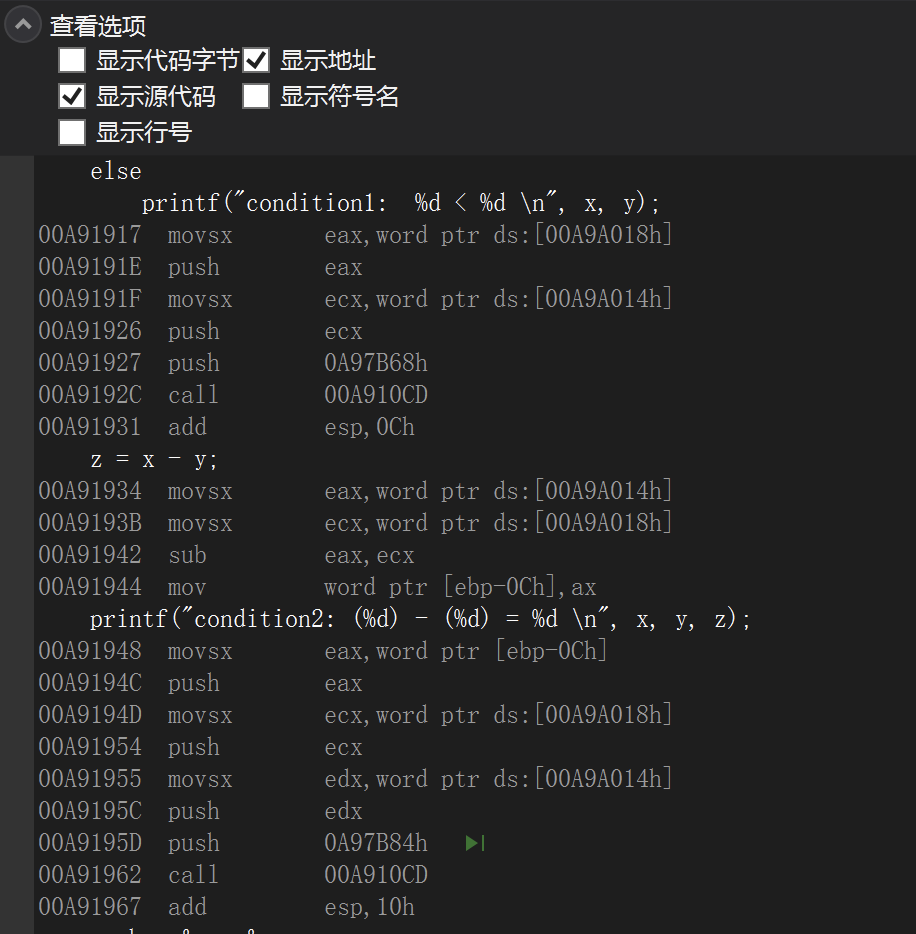


图1.1.2 反汇编未勾选“显示符号名”

3.使用寄存器窗口（图1.1.3）。显示了八个的通用寄存器以及其中的值，以及EIP指令指示器和EFL标志寄存器，由其中的值显示可知这些寄存器都是32位的。在窗口中右键，通过勾选“CPU段”显示出段寄存器CS、DS、ES、SS、FS、GS，段寄存器是16位的；通过勾选“标志”显示出几个标志的值，其中OV表示溢出标志、AC是辅助进位标志、CY是进位借位标志、UP是增量标志、EI允许中断标志、PL表示正、ZR表示零、PE表示偶校验。

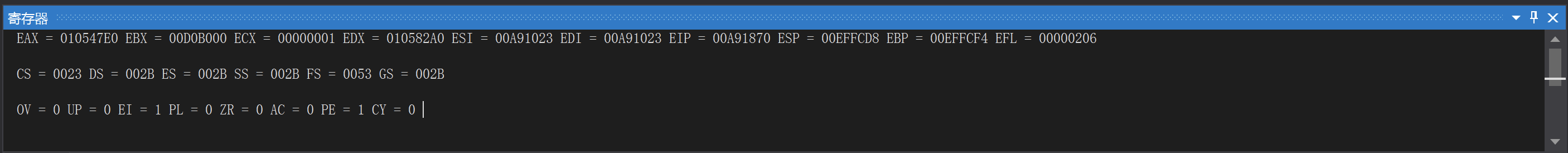


图1.1.3 寄存器窗口

4.利用监视窗口可以显示变量的值，利用内存窗口可以显示变量的值在内存中的具体表现形式。这里我选用了y变量，在上文反汇编是我们知道了运行时y 变量的地址是0A9A018h，在内存窗口中输入即可显示y变量在内存中的保存形式。y变量是short类型，需要两个字节的内存。从监视窗口（图1.1.4）中我们看到y变量的值是0x8044，而在内存窗口（图1.1.5）中我们看到44保存在低地址中中，80保存在下一个地址0A9A019h，采用小端存储的数据存储方式。

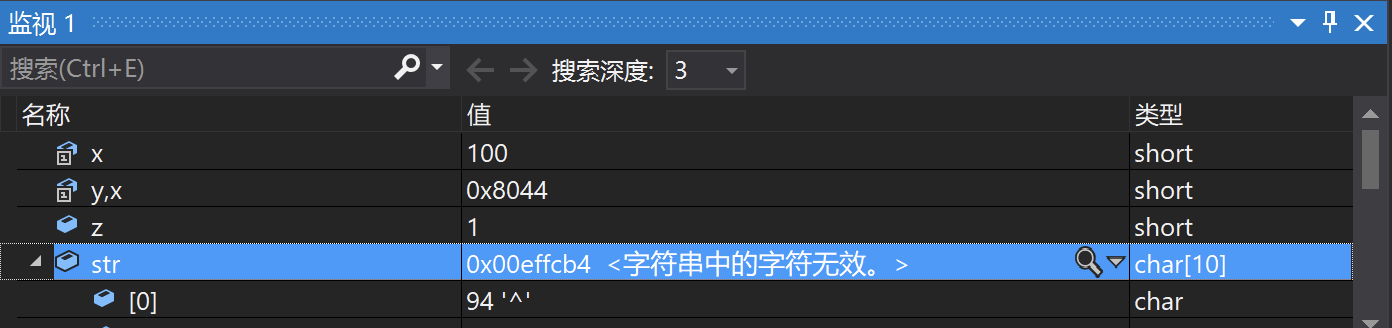


图1.1.4 监视窗口

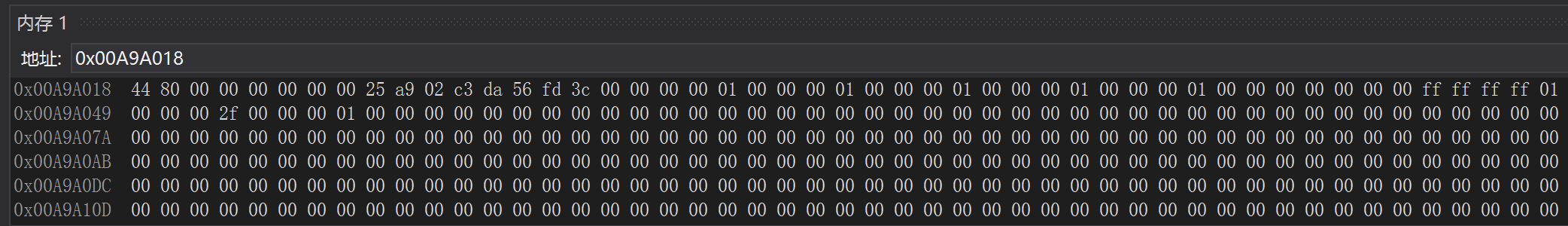


图1.1.5 内存窗口

5.探究有符号整型数和无符号整型数是如何存储的。我们知道无符号数是从0开始一直到操作数各位都是1的范围；而有符号数在计算机中用补码表示，正数的补码是其本身。

我们进行试验。定义了一个unsigned short类型的x变量，令x为-100，在此时运行中地址为01BA014h；定义了一个short类型的y变量，令y为-100，此时运行的地址为01BA018h，我们打开内存窗口来查看在内存中保存形式。

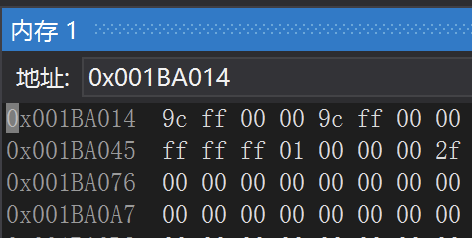


图1.1.6 有符号整型数与无符号存储对比

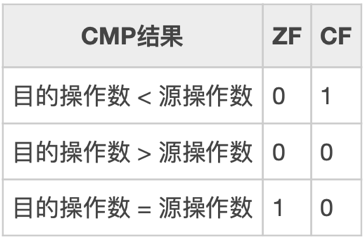
可见即使x和y一个为无符号整型数一个为有符号整型数，但是在保存一个负数时，都是保存这个数的补码表示，并无区别。但是在如何解释时是有区别的，x会被解释成65436，而y会被解释成-100.

6.CPU在执行加减运算的时候，不区分有符号数还是无符号数，数的最高位都和后面的二进制位一样参与运算，按照相同的规律进行运算。在程序中标明的负数都会以补码在计算机中保存，加减法被统一用加法来进行运算。

在执行加法指令时：如果执行前两个操作数的最高位相同，而运算结果的最高位和它们相反，那么溢出标志OF=1，其他情况下OF=0。如果运算中从最高位再向前面产生进位时，则把进位标志CF置为1，否则置0。

在执行比较指令时：

如果比较的是两个无符号数，则零标志ZF、进位标志CF结果如下图1.1.7所示



1.1.7 无符号数比较时标志位结果

如果比较的是有符号数，则符号标志SF、溢出标志OF、零标志ZF结果如下图1.1.8所示



1.1.8 有符号数比较时标志位结果

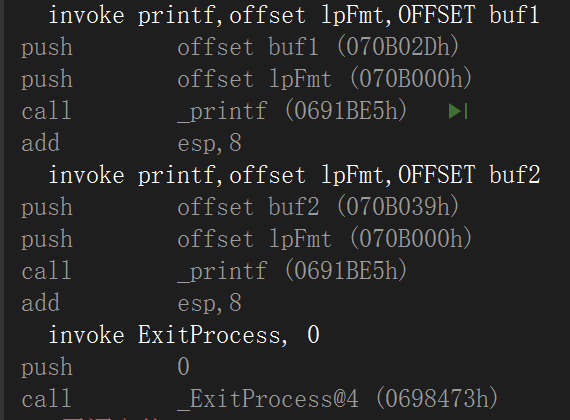
## 任务1.2的实验过程

### 实验方法说明

1. 准备上机实验环境，进行DOSBox环境的安装、运行，通过试用初步了解软件的基本功能、操作等。
2. 用TextEdit编辑源程序，体会16位段程序的编译、链接和调试过程，并尝试在调试状态下直接录入或修改代码。

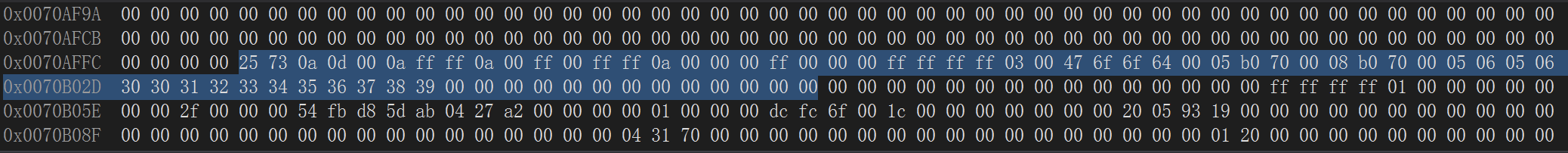
### 实验记录与分析

1.在反汇编窗口中可以看出，汇编源程序中的语句和反汇编语句在一些基本指令如MOV、ADD等基本是一致的。在反汇编语句中，寄存器间接寻址会指出该地址的类型。而在invoke伪指令会在反汇编语句中翻译成一系列直接执行的语句。



1.2.1 汇编源程序与反汇编语句的对比

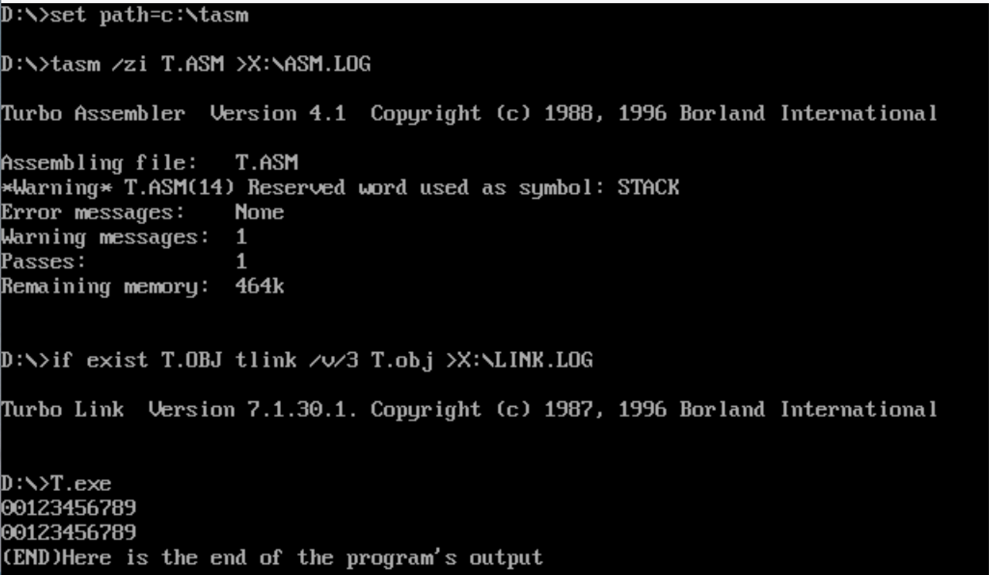
2.数据段存储结果如下图1.2.2中所选择的数据部分所示



1.2.2 数据段存储结果

数据存储时起始地址会遵循内存对齐的原则。按照变量定义的类型如DB、DW、DD来为每一个数值字面量来分配内存中所占的字节数。数值翻译成补码来进行存储。各个变量地址之间所间隔的就是该变量所占的字节数。如X为DB类型，其值为10，255，-1；在内存中存储0a ff ff，占三个字节，所以Y-X就是3.

1. 了解如何使用DOSBox编译16位段程序，并链接运行。



## 任务1.3的实验过程

### 设计思想及存储单元分配

1.存储单元分配

在DATA段准备一些会用到的信息，程序运行时需要的提醒字符串，登陆时需要的老板姓名、密码，四种现有的货物GA1、GA2、GA3和GA4，以及程序运行时所需的一些变量。

2.寄存器分配（包括功能中必要的变量说明）

用户信息检验：ESI记录输入信息的偏移地址，EDI记录所要比对的正确信息的偏移地址，AL、BL，分别寄存输入信息和段中正确信息的每一个字符值。

商品名称比对：ESI记录输入商品信息的偏移地址，EDI记录商店中现有商品的信息的偏移地址，offset1保存当前比对商品距离商店中第一个商品的偏移地址，AL、BL分别寄存商品名称和输入名称的每一个字符值。

打印商品信息：ESI记录商店中现有商品的信息的偏移地址，offset1保存当前比对商品距离商店中第一个商品的偏移地址

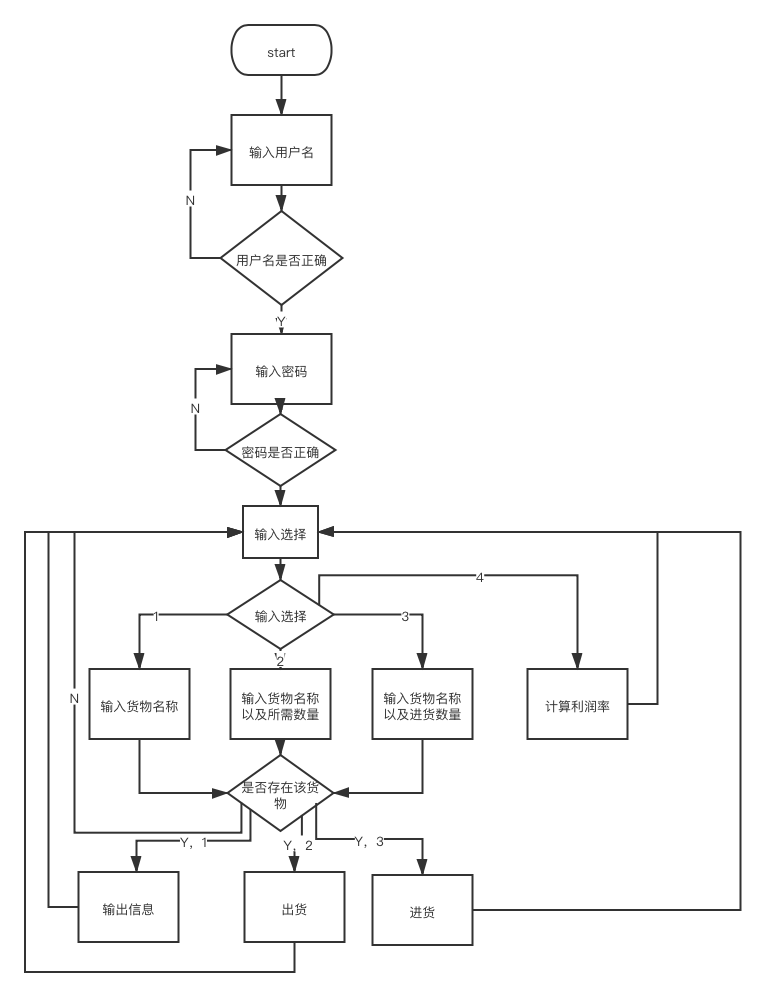
出货：ESI记录商店中现有商品的信息的偏移地址，offset1保存当前比对商品距离商店中第一个商品的偏移地址，AX保存输入所需要的货物数量，BX保存当前货物的进货数量，CX保存当前货物的销售数量，后BX变为当前货物的现有数量（进货数量减去销售数量）

进货：ESI记录商店中现有商品的信息的偏移地址，offset1保存当前比对商品距离商店中第一个商品的偏移地址，AX保存输入进货的数量，BX保存当前货物的进货数量

计算利润率：ESI记录商店中现有商品的信息的偏移地址；AX一开始保存销售价格，而后与销售数量相乘，保存销售额；BX一开始保存进货价格，而后与进货数量相乘，保存进货所需的金额；之后计算过程，AX减去BX并保存在AX中，再与BX相除，最后利润率保存在AX中。

3.程序模块。程序模块分为主菜单模块，查找、出货、补货模块，计算利润率模块。

### 流程图



1.5.1 程序流程图

### 实验步骤

1.准备上机实验环境，VMware Fusion下的Windows 10 x64。

2.直接在Visual Studio 2019上创建编辑源文件,在源文件目录下命名为system.asm。对源文件生成解决方案，观察提示信息，根据错误提示对源代码进行修改，重新生成解决方案，直至不在报错为止。

3. 执行该程序，观察运行情况。对每个功能进行单独的测试，测试输入正常用例和异常用例，查看实现的功能是都符合要求和预期，根据调试过程中出现的错误信息进行修改。如果出现运行上的问题，则按照以下方法进行修改和调试。直至实现的功能符合预期，完成程序功能的实现

(1) 查看对应的流程图，检查有没有程序流程里的逻辑问题。

(2) 使用“逐过程”功能开始调试程序，并打开寄存器窗口，边调试边观察寄存器的值的变化。

(3) 如果是代码在逻辑实现上的错误，思考并修改。

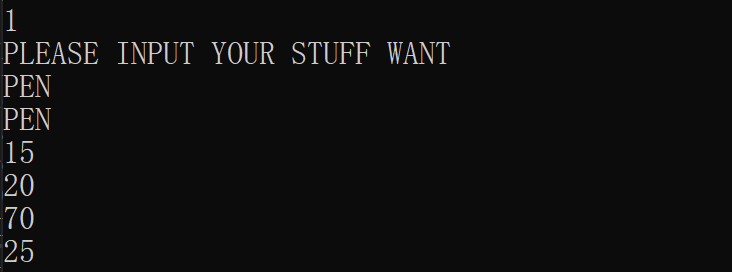
### 实验记录与分析

1. 实验环境条件：2.3 GHz Quad-Core Intel Core i5，8 GB内存；虚拟机环境下WINDOWS 10 VS2019

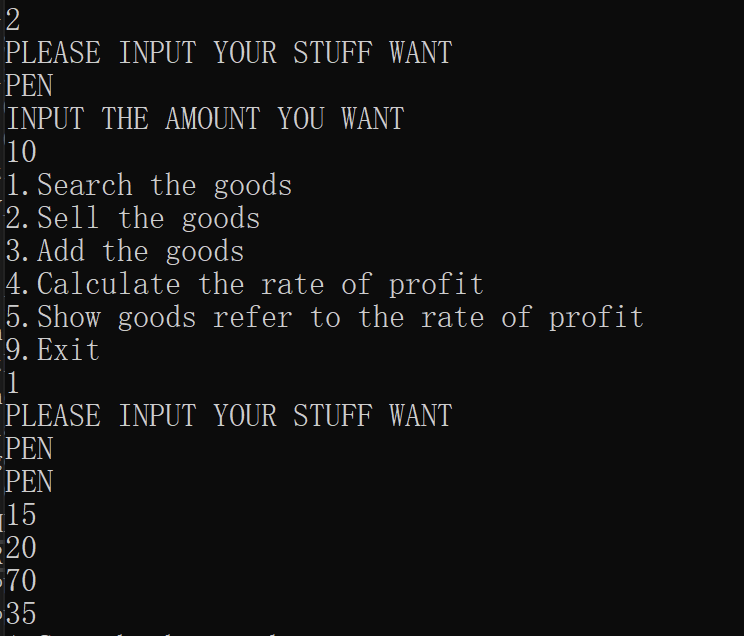
2. 实验流程



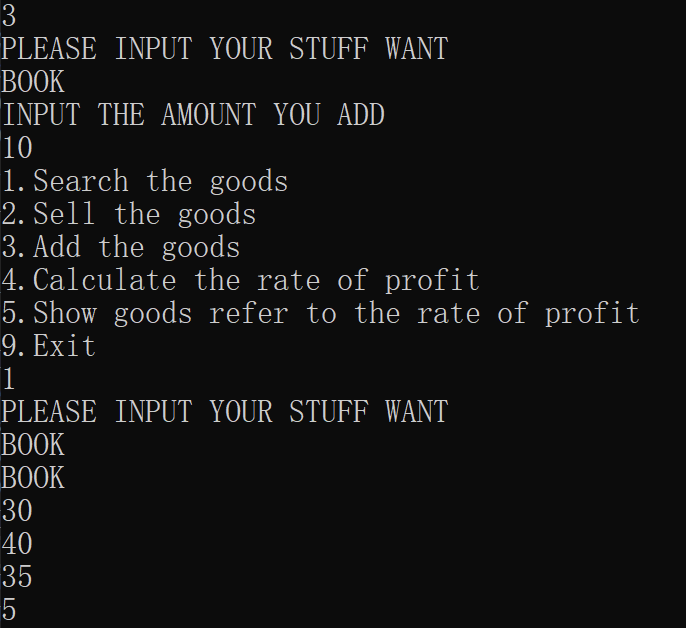
1.5.2 登陆并进入选择程序



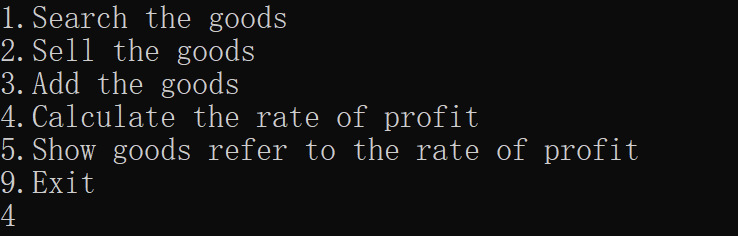
1.5.3 选择1 输出信息



1.5.4 选择2 出货

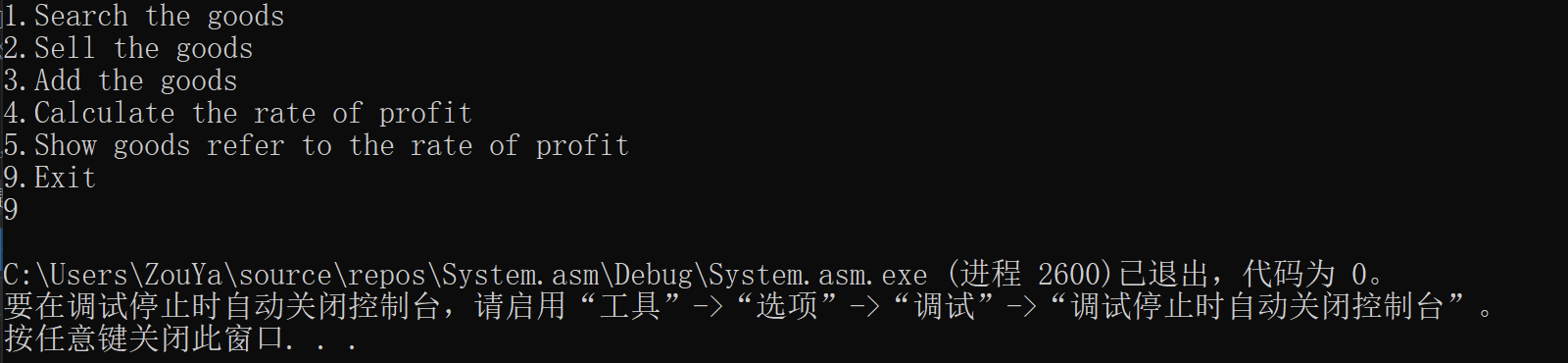


1.5.5 选择3 进货



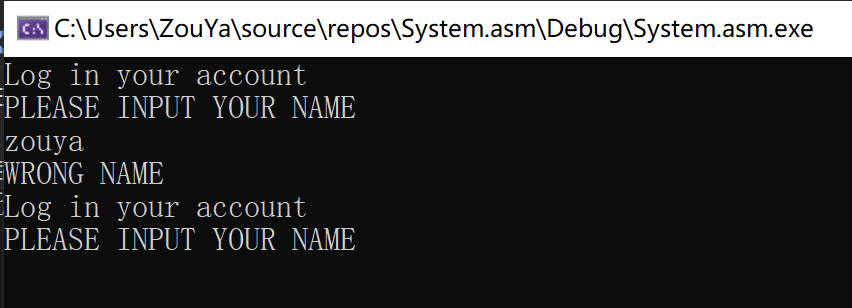
1.5.6 选择4 计算利润率

利润率计算结果为PEN -52%，PENCIL -25%，BOOK -73%，RULER 0.

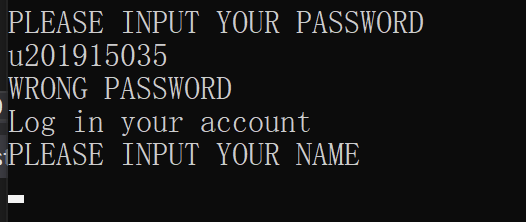


1.5.7 选择9 退出程序

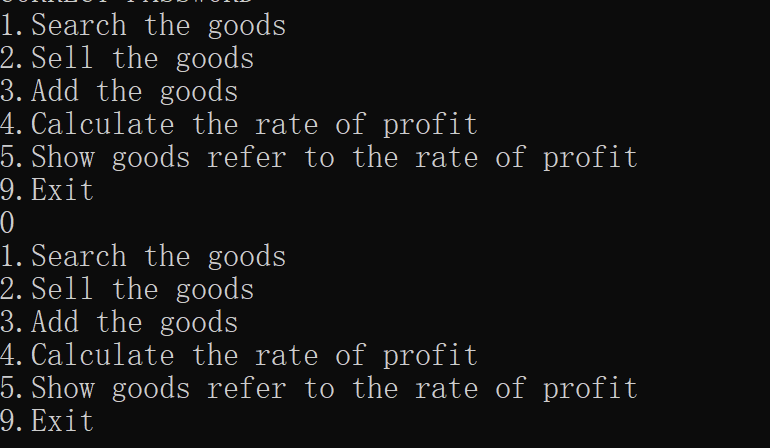
3.输入信息有误



1.5.8 输入错误的用户名

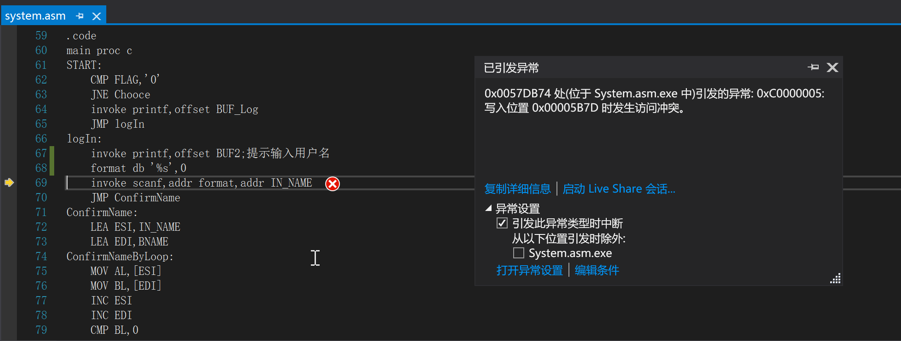


1.5.9 输入错误的密码



1.5.10 输入不正确的选项

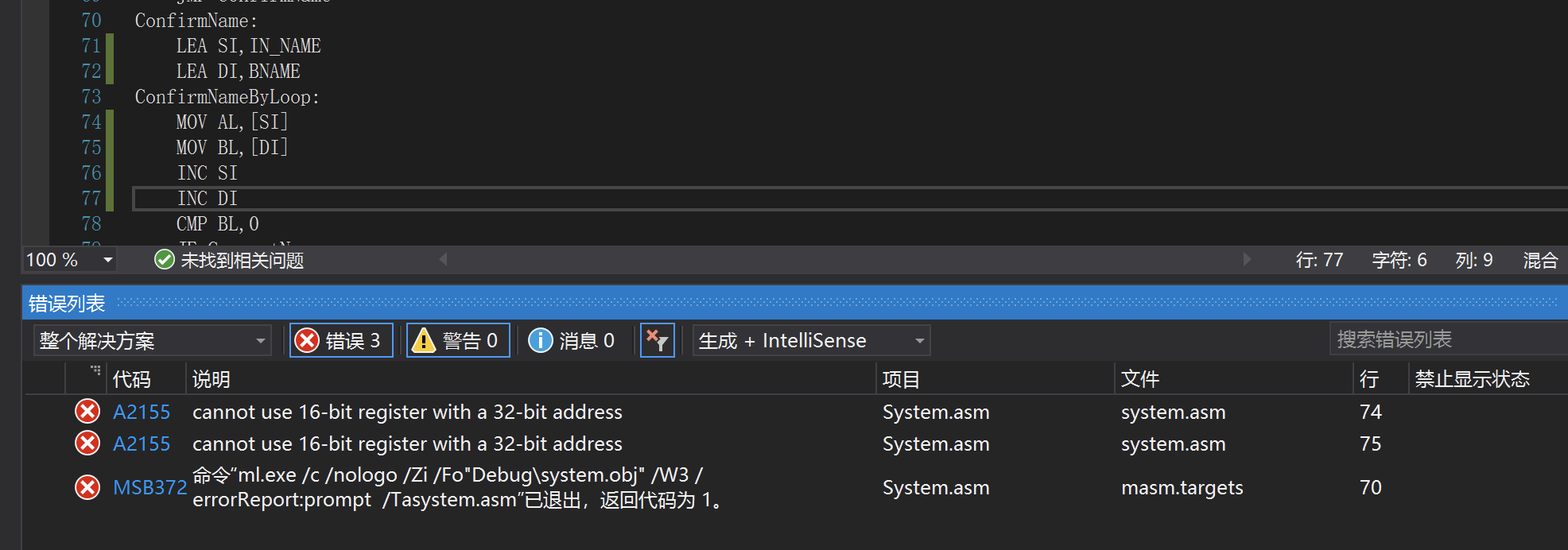
4.在调用scanf函数时出现的错误



1.5.11 调C语言程序时的错误

一开始我以为是调函数的语法错误，但是查了之后发现语法并没有问题。问了老师和同学也没有找到解决方案，但在后来我突然意识到，可能是我把format变量定义在了程序段中的错误，改了之后果然可以进行下去了。因为在C语言包括别的编程语言中，我们都可以直接定义局部变量，导致我在汇编里也下意识这么认为。

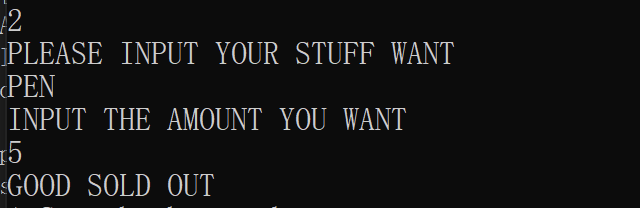
1. 寄存器保存地址不匹配问题



1.5.12 寄存器保存地址不匹配

如上图所示，32位程序里，用16位寄存器是不能来处理32位地址的。所以不能用SI、DI，而是应该使用ESI、EDI。

1. 跳转不匹配问题



1.5.13 跳转不匹配

在明显输入所需的货物量小于现存数量时，仍然显示已卖完。我一开始以为是寄存器保存变量值的问题，但在调试后发现并没有问题。后来检查了程序才发现是跳转指令使用错误。熟悉掌握跳转指令在程序编程中非常重要。

1. 寻址问题。

我在编程中经常卡住的一个地方就是寻址，比如说寄存器ESI中保存着一个变量的偏移地址，我想在这个变量后18位的地方存入AX寄存器中的值，我应该使用偏移地址还是寄存器间接寻址呢？在实验后我发现应该是寄存器间接寻址，语句应该是这样：MOV[ESI+18],AX。其实仔细思考之后这样才是合理的，但是对于汇编编程新手来说这并不很直观，体现了实验的重要性。

## 小结

### 主要收获

收获知识点如下：

1.32位编程中，可以使用C语言函数，具体如下所示

ExitProcess PROTO STDCALL :DWORD

includelib kernel32.lib ; ExitProcess 在 kernel32.lib中实现

printf PROTO C :ptr sbyte,: VARARG

scanf PROTO C :ptr sbyte,: VARARG

includelib libcmt.lib

includelib legacy\_stdio\_definitions.lib

includelib是包含进C语言的库函数，上述代码让我们在程序中可以调用（invoke）ExitProcess退出函数、printf打印函数、scanf输入函数。printf和scanf的使用格式和C语言是一样的，非常便捷。

2.在32位编程中，字符串以0AH,0DH,0结尾即可在输出时自带换行。而在我们需要单独输出换行时，可以定义如下变量CRLF DB 0AH,0DH,0。输出CRLF即可输出一个单独的换行。

3.循环执行一块代码可以用LOOP 标号来实现。

4.转移指令：JMP无条件转移，JE相等转移，JNE不相等转移，JS为负转移，JNS为正转移。

5.出现循环时，检查程序的跳转逻辑。

6.在程序无法出现正常逻辑输出时，学会了使用“逐过程”以及寄存器窗口观察程序的运行。比如，我在出货时总是不能得到正确结果，在查看了寄存器变化后才发现，当调用C语言函数后，EAX、ECX、EDX都会出现变化。如果此时用这三个寄存器保存变量的话，自然不能正确计算。

### 主要看法

在这次的汇编实验中，我收获了很多。

首先，作为macOS的用户，一开始为了汇编我先装配了虚拟机，在查阅资料后选择了VMware Fusion，中间在虚拟机搭建上也出过一些环境变量的错误，后来慢慢解决了，然后在虚拟机中下载了VS2019，并学会使用。后来我发现可以在macOS环境下使用Visual Studio Code，并且搭配虚拟机DOSBox，可以直接调试。但是我在两个环境里还是用Windows下的VS2019更加顺手。虽然原生VS2019没有代码高亮、自动补全等功能，但是对比在DOSBox里调试，我还是更喜欢VS2019.

在任务1.1中，我跟着教科书学会了调试，使用寄存器窗口、监视窗口、反汇编窗口、内存窗口。一开始我并没有意识到这些的重要性，直到在1.3编写程序报错时，我才发现这些窗口的好处。在C语言程序里，一般调试时我会用断点或者直接printf中间变量来调试。而监视窗口让我不需要再去printf，直接可以看到变量的值；寄存器窗口也直接显示了当前寄存器的值，在语句执行出现错误的当下我就可以发现：这些窗口在我编程时起了很大的作用。

在检查预习报告的时候，老师告诉我，我平时这种直接开始写程序，写完了再写报告的思维是有错误的。我们应该在动手写程序之前，就先构思好这个程序的框架和流程，画好流程图，规划好程序的大致步骤，再等我们开始写程序的时候就能更加容易也更加快速了。这是我在整个任务1.3中学到的最大的点。程序思路由面向程序开发到面向设计开发的转变，我想这也是我们今后要努力的方向。

在本次实验中，由于我最近事情特别忙，没有安排好实验时间，没有在上机前及时完成预习，导致我在后面补的时间花费比较大。后续事情调整了之后，我会把实验的时间安排再调整过来，在上机的时候就可以直接开始实验、调试、记录等工作了。

# 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. 深刻理解CPU执行指令的过程，不同特点的编程技巧和指令序列组合对程序长度及执行效率的影响，掌握代码优化的基本方法。

## 实验内容

**任务2.1 对任务1.3的程序进行完善和优化**

先实现“5.按利润率从高到低显示商品信息”的功能（该功能先按照商品利润率的大小排序，然后显示排序后的商品信息，显示完毕后返回主菜单），然后对功能“4”和“5”的程序做些改造，以便计时与多次循环（多次循环的目的仅仅是为了更明显地观察程序执行的时间）。

改造之后的程序流程为：当输入“4”时，在原来“4”的代码之前先执行新增的计时开始的代码，然后再执行新增的一段计数的循环控制程序，接着执行原来“4”的功能，执行完后不要返回到主菜单，而是直接执行“5”的功能，“5”的功能执行完后，也不要返回到主菜单，而是回到计数的循环控制程序；当计数未结束时，重复执行“4”和“5”功能，直到计数完毕，才跳转到新增的计时结束的代码处，显示了计时的时间之后回到主菜单。

优化工作包括代码长度的优化和执行效率的优化等等（本次以执行效率/性能的优化为主）。

同时，为增强程序的可读性，要求将商品信息定义成一个结构，参考下列示例。

GOODS STRUCT

GOODSNAME db 10 DUP(0)

BUYPRICE DW 0

SELLPRICE DW 0

BUYNUM DW 0

SELLNUM DW 0

RATE DW 0

GOODS ENDS

对商品信息的访问都通过该结构进行。

## 任务2.1实验过程

### 实验方法说明

1．利润率排序。采用冒泡排序的思想。

建立一个偏移地址数组GoodArray，把各个商品的偏移首地址保存进去，然后按照商品的利润率把偏移地址进行排序，最后在输出的时候遍历这个数组即可。

排序过程中用变量i、j作为内外循环的控制变量。在寄存器分配上，用EDI保存GoodArray的偏移地址，用ESI和EDX保存GoodArray中的前后两个偏移地址值。最后输出时用AX来暂存各个数据。

2.程序改造思路。使用变量timesCount进行计数，在判断输入选择为’4’的地方，跳转到一个新增的计时开始的代码，调用winTimer开始计时，接着执行原来“4”的功能，执行完后不要返回到主菜单选择的语句，而是直接跳转到执行‘5’的功能，‘5’的功能执行完后，也不要返回到主菜单，而是跳到新增的计数的循环控制程序；当计数未结束时，重复执行‘4’和‘5’功能，直到计数完毕，才跳转到新增的计时结束的代码处，调用winTimer结束计时，显示了计时的时间之后回到主菜单。

3. 所采用的计时方法。用winTimer.asm中定义的计时子程序进行计时。

4. 主要的优化思路：

1. 指令优化（例如将功能4和5中的乘法指令换成移位指令）
2. 减少循环体中的指令

### 实验记录与分析

执行次数为10,000,000次。

为了使计时的效果更加明显，把商品的数量改为10个。

1.优化思想一：把变量经过寄存器计算改成直接用变量进行计算。

在计算利润率的循环里：

MOV ECX,Offset1

ADD ECX,20

MOV Offset1,ECX

CMP ECX,200

JE SortAndShowMessage

改成

ADD Offset1,20

CMP Offset1,200

JE SortAndShowMessage

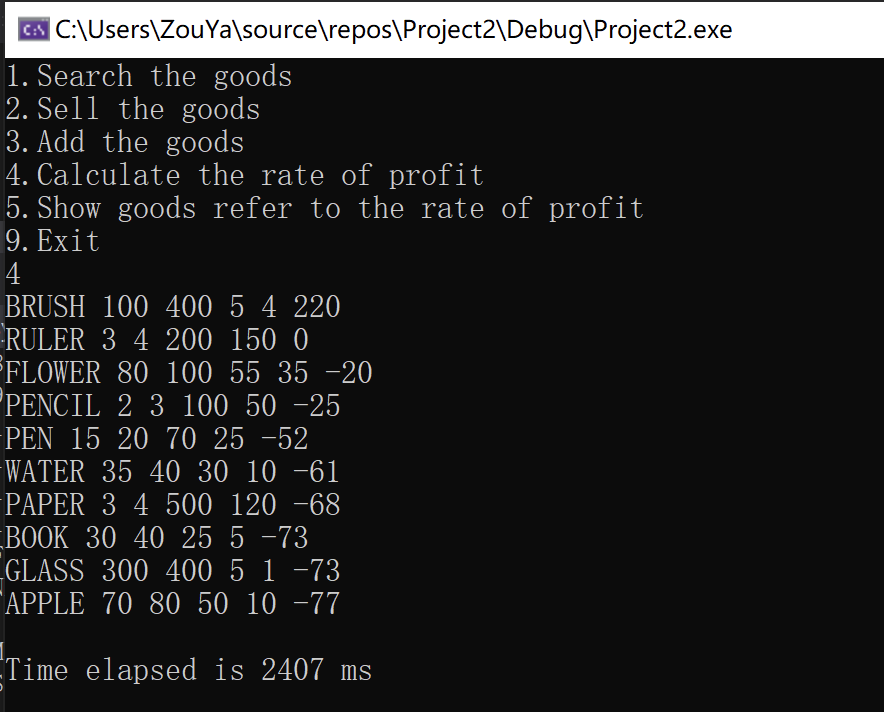


图2.3.1 优化前的执行时间

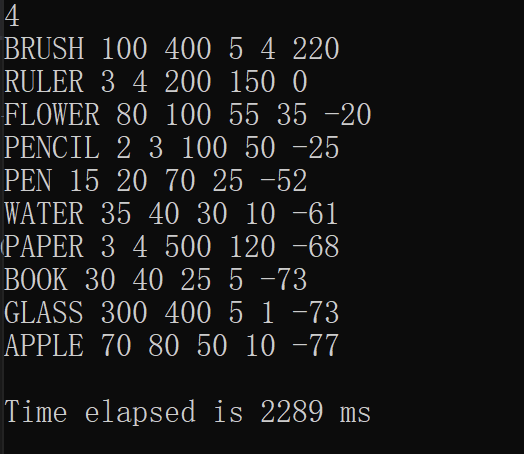


图2.3.2 优化后的执行时间

2.优化思想二：把乘法指令改成左移指令

在排序算法中：

MOV ECX,j

imul ECX,4

改成

MOV ECX,j

sal ECX,2

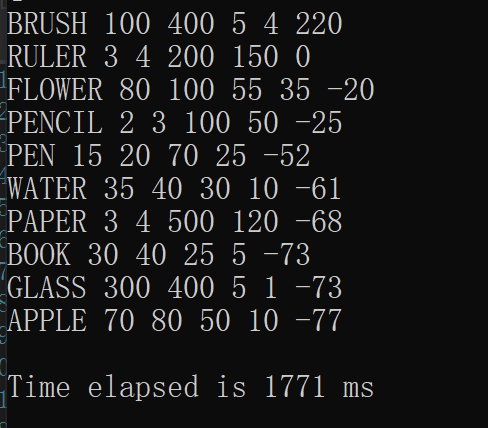


图2.3.3 第二次优化后的执行时间

3.在两次优化的基础上更换另一个电脑进行实验。

英特尔 Core i5-8265U @ 1.60GHz；WINDOWS 10下 Visual Studio 2019

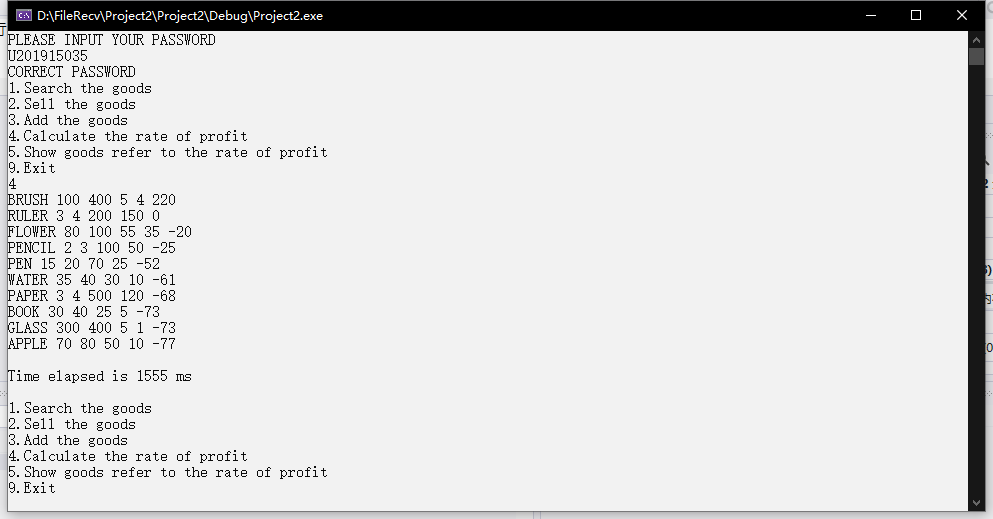


图2.3.4 更换电脑后的运行时间

## 小结

### 主要收获

在设计功能5的程序时，我学会了串操作指令和结构变量的使用，同时也加深了我对汇编的指令、结构概念的理解。

在编写冒泡排序的过程中，我花费了大量的时间。因为我是采用建立一个偏移地址数组GoodArray，把各个商品的偏移首地址保存进去，然后对数组进行排序的思想来进行该功能的实现的，但是因为两次寄存器间接寻址的问题，导致我的思维比较混乱。对地址偏移的控制也很混乱。而且以前用高级语言编程时，当需要变量时就直接创建一个新的变量来使用，但是在汇编语言中，寄存器的数量是固定的，使用时不仅要考虑是否会由于Loop指令或者调用C函数使得寄存器的值发生变化，还要考虑寄存器的值是否使用冲突的问题。这也体现了高级语言的优势。

在优化时，我发现了只是减少几个指令在大量循环后对于程序执行时间的影响是很大的。同时我还发现移位指令比乘法指令来的高效，只是更改了一个乘法指令就可以把时间缩短22.63%。

最后我在另一台电脑上运行该程序，测试结果也发生了变化。由此可以得出，影响一个系统最终是否符合要求的影响因素是有很多方面的，不仅有功能问题、算法问题、程序结构问题、指令选择问题，还有运行环境问题。

### 主要看法

VS2019写汇编程序的时候没有代码高亮也没有补全提示，作为IDE来说使用感受非常糟糕。写完实验后发现可以配一个插件，比如说AsmDude。这个插件让ASM文件能与C文件一样不同类型单词都有不同的颜色，且当光标指到一个单词时，在代码中其他位置的同一个单词也能被高亮。

# 模块化程序设计

## 实验目的和要求

（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）掌握汇编语言程序与C语言程序混合编程的方法；

（4）掌握较大规模程序的开发与调试方法；理解模块之间的信息传递与组装的基本方法；

（5）完成指定功能的程序设计与调试。

## 实验内容

**任务3.1 采用子程序、宏指令、多模块等编程技术调整经过任务2.1完善后的商品信息后台管理系统。**

要求：

1. 实现字符串比较的子程序，用户名、密码及商品名的比较均调用该子程序。
2. 出售商品、计算商品的利润率、利润排序等均用子程序实现。
3. 要有一个有参数的子程序，其调用使用invoke伪指令，并且在该子程序有形参，定义并使用了局部变量。
4. 至少定义一个带有形参的宏指令。
5. 将汇编源代码至少分解到两个不同的源文件中。
6. 在实验中，要对回答如下问题：子程序与主程序之间是如何传递信息的？刚进入堆栈时，堆栈栈顶及之下存放了一些什么信息？执行 CALL 指令及RET指令，CPU完成了哪些操作？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？invoke 伪指令对应的汇编语句有哪些？子程序中的局部变量的存储空间在什么位置？如何确定局部变量的地址？访问局部变量时的地址表达式有何限制？
7. 探究：

a)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？

b)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。观察段合并与不合并时对程序的影响。

c)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？

d)通过调试工具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。

e)利用宏功能使汇编语言的程序变得更加直观易读。请观察下面的例子，体会和思考相关的宏定义。

例：下面是一个利用宏功能直观化后的完整代码段程序（16位段程序）。

StartProgram code，data，stack，start

Initial\_ds

GetStringTo BUF

DisplayStringFrom BUF

ExitToDOS

EndProgram code，start

**任务3.2 C语言和汇编语言混合编程**

对任务3.1中的程序，进行改造。

1. 用户登录功能、主菜单的显示功能用 C语言程序实现；
2. 用C语言实现，增加“6.添加新商品”的功能；
3. 其他功能模块仍用汇编语言实现。

回答如下问题：

1. 在C语言程序、汇编语言程序中，分别是如何说明外部变量和函数的？汇编指令访问C的变量时是如何翻译的（观察对应的反汇编代码）？C语言语句访问汇编语言定义的变量时是如何翻译的？
2. 如何保证在C语言程序和汇编语言程序中，正确访问商品信息的结构数组？
3. 观察不同变量地址之间的关系；根据该关系，实现一个变量名称不出现在语句中的情况下，修改该变量值的功能。（比如，已知int x，y; 假设这两个变量在内存中相邻，就可以用表达式 \*(&x-1)=20修改 y的值）
4. 地址类型转换的含义是什么？ （比如，char a[10]; 一种地址类型转换的做法： \*(int \*)a=123；）
5. 函数调用语句对应的汇编语句有哪些？ 调用函数与被调用函数之间是如何传递信息的？

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、偏移的值等。

## 任务3.1实验过程

### 设计思想及存储单元分配

1. 汇编源代码分为两个源文件，对应两个模块。第一个文件system.asm保存主程序，第二个文件subProgram.asm保存字符串比较子程序、货物名字比较子程序和四个功能性子程序，分别对应两个模块中的内容。所有的程序都定义在.code代码段中，在WIN32系统中，不存在需要更改段寄存器的问题。

在主程序中，要使用另一个文件的子程序，需要如下的声明：

strcmp proto :dword,:dword

GoodCompare proto :dword

PrintMessage proto :dword

SellProc proto :dword,:word

CalculateProc proto

SortProc proto

为了在另一个文件里调用主程序数据段中定义的变量，需要在该文件对变量进行public声明，而在另一文件里对变量进行extern声明。

2. 测试框架和测试数据。

对主程序模块的测试就是对程序正常运行以及输入用户名和密码的测试，但该输入测试需要用到字符串比较函数，可以先用原来写的汇编指令代替。测试数据为正确的用户名和密码，还有错误的用户名和密码。

对子程序模块的测试可以分步进行，一个个函数实现的过程中一个个进行测试。比如说先实现了字符串比较函数，那么在主程序中调用该函数进行比较用户名和密码来测试。每一个函数编写完成后都用主程序来调用它进行测试，最后即可得到完整的程序。如此测试简单有效，又防止了二次整合程序。

3.将程序功能设计成六个子程序，分别为字符串比较子程序、货物名字比较子程序、输出商品信息子程序、出售商品子程序、计算商品利润率子程序、利润排序子程序。后四个子程序对应货物系统管理系统的功能。进货商品功能由于执行较为较为简单，定义成带形参的宏定义进行实现。

4.字符串比较子程序负责实现用户名、密码及商品名的比较。该子程序比较两个字符串，入口参数保存于堆栈中，后一个字符串的地址先入栈，出口参数保存在EAX中，0表示两个字符串完全相等，1表示不相等。算法采用从字符串左端开始逐一比较单个字符。如果提前检测到输入的字符串已经到了结尾直接结束；如果比较的两个字符不相等也则直接结束，如果当前两个字符相等，则继续往下比较。寄存器分配上，用EDI和ESI分别保存两个字符串，dl用于缓存当前读到的字符。

5.货物名字比较子程序负责从现有的货物信息中比较寻找出用户输入的货物名字。入口参数为当前输入的货物名字首地址；出口参数保存在EAX中，如果找到对应的货物，那么输出当前货物的偏移量，如果未找到，那么输出1。在寄存器分配上，ESI保存入口参数的货物地址，EDI保存货物信息的偏移地址，EBX用来保存偏移量以及比较的次数。

6.输出商品信息子程序负责实现输出单个商品的所有信息。入口参数为当前所要输出的商品信息离第一个商品信息的偏移长度，没有出口参数，结束直接返回。寄存器分配上，用ESI保存商品信息地址，用AX暂存商品数据。

7．出售商品子程序负责修改商品的已售数量信息。入口参数为当前所要出售的商品信息离第一个商品信息的偏移长度和当前所需求的商品数量，没有出口参数，结束直接返回。首先要先把商品的进货数量与已售数量相减，与输入所需求的商品数量比较，如果不够则直接输出商品售空的错误信息，如果仍有足够数量，则将已售数量加上当前所需的商品数量。寄存器分配上，用ESI保存商品信息偏移地址，AX保存所需商品数量，BX保存进货数量，CX保存已售数量。

8.进货宏定义负责修改商品的进货数量信息。入口参数为当前所要进货的商品信息离第一个商品信息的偏移长度和所进货的数量，没有出口参数，结束直接返回。把商品信息中的进货数量加上当前输入的数量即可。寄存器分配上，用ESI保存商品信息偏移地址，AX保存所需商品数量，BX保存进货数量。

9.计算商品利润率负责计算当前所有商品的利润率。没有入口参数也没有出口参数。ESI保存商品信息偏移地址，AX计算并保存售出金额，BX计算并保存进货金额，利用EDX保存可能的溢出，最后计算结束利润率保存于AX中。

10.利润排序子程序用于将商品的偏移地址按照利润率排序保存，最后依照次序输出商品信息。没有入口参数也没有出口参数。用GoodArray连续保存商品的各个偏移地址，程序采用冒泡排序的方式，i、j变量为中间局部控制流程变量，经过Sort和SortLoop两个内外循环完成排序。寄存器分配上，ECX、EDX负责局部计算数据的保存，EDI保存GoodArray的偏移地址，EDX、ESI负责保存对应比较的商品偏移地址，用AX、BX保存商品的利润率。

### 流程图

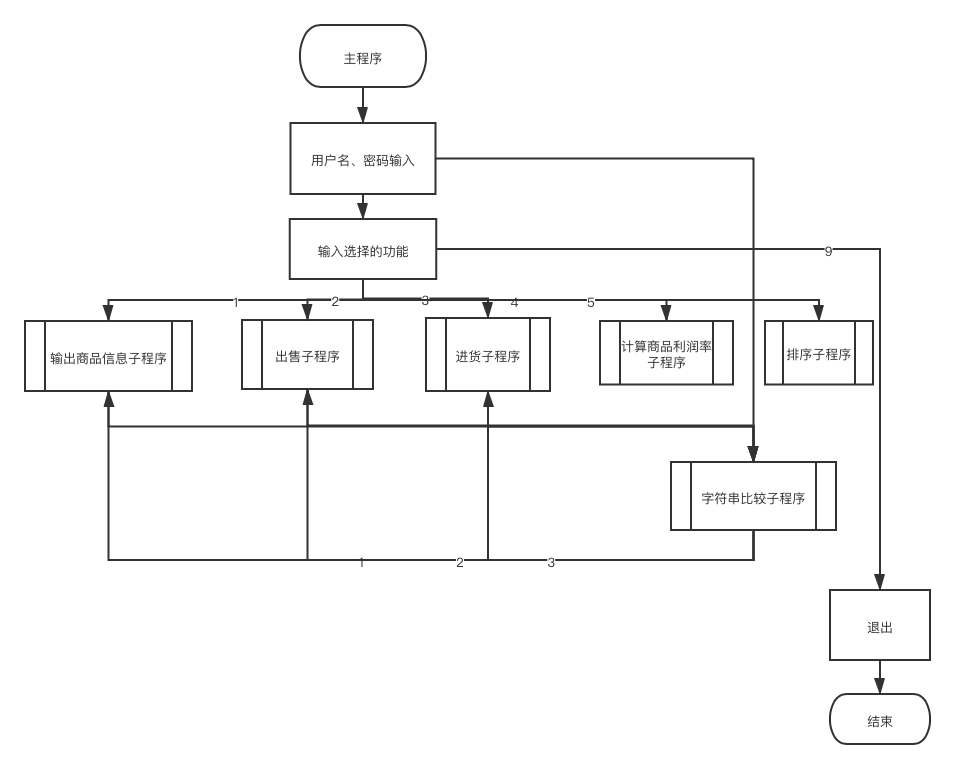


图3.3.1 子程序流程图

### 实验步骤

1.编写字符串比较函数，并在输入用户名、输入密码、输入商品名时都调用该字符串比较函数进行实现。

2.将之前程序里的各个功能改写成功能性子程序或者宏定义的形式。在改写的过程中学习如何声明外部文件的程序、使用invoke指令调用子程序、如何使用形式参数、如何使用局部变量、如何使用宏定义等等内容。

3.在实现程序的过程中，使用调试工具：反汇编窗口、内存窗口、监视窗口、寄存器窗口来对程序执行过程进行观察分析，尝试回答实验任务中的问题，并进行深入探究。

### 实验记录与分析

1.字符串比较函数的子程序设计如下。

strcmp proc uses esi edi edx,x:dword,y:dword

mov esi,x

mov edi,y

strcmpStart:

mov dl,[esi]

cmp dl,[edi]

jne strcmpNot

cmp dl,0

je strcmpEqual

inc esi

inc edi

jmp strcmpStart

strcmpEqual:

mov eax,0

jmp strcmpExit

strcmpNot:

mov eax,1;eax output

strcmpExit:

ret

strcmp endp

2.进货功能采用带形参的宏指令来完成。其代码如下所示：

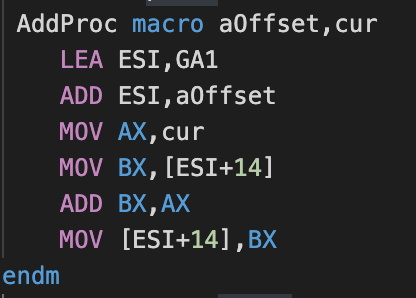


图3.3.2 带形参的宏指令

3.子程序定义的源代码框架如下，其中strcmp为字符串比较子程序，GoodCompare为货物名字比较子程序，PrintMessage为单个货物信息打印子程序，SellProc为出货子程序，CalculateProc为计算利润率子程序，SortProc为根据货物利润率排序子程序：

strcmp proc uses ESI EDI EDX,x:dword,y:dword

strcmp endp

GoodCompare proc uses ESI EDI EBX,input:dword

GoodCompare endp

PrintMessage proc uses ESI EAX EBX,aOffset:dword

PrintMessage endp

SellProc proc uses ESI EAX EBX ECX,aOffset:dword,cur:word

SellProc endp

CalculateProc proc uses ESI EAX EBX ECX EDX

CalculateProc endp

SortProc proc uses ESI EAX EBX ECX EDX

SortProc endp

4. 子程序与主程序之间传递信息是通过参数传递来实现的。主程序在调用子程序需要把子程序需要处理的数据传给子程序，即提供入口参数；子程序把处理完的数据返回给主程序，即返回出口参数。主程序和子程序之间传递参数是需要实现定义好的。常见的参数传递方式有寄存器法、约定单元法、堆栈法。寄存器法就是把参数放在约定好的寄存器中。约定单元法就是把参数放到约定好的存储单元中，但是使用全局变量会影响模块间的独立性。堆栈法就是通过堆栈来传递信息，这是传递参数最常使用的方法。

5.刚进入子程序时，堆栈栈顶保存的是程序断点地址，栈顶之下依次放着压入的参数，最接近栈顶的是最后压入的参数。这是因为在使用堆栈法传递参数时，一定是先将参数压入堆栈，然后执行子程序调用call语句；call语句执行时，先会保存断点地址到堆栈的栈顶，然后把子程序过程名对应的偏移地址传给EIP。在退出子程序之前执行ret指令，会从堆栈栈顶弹出一个双字送给EIP，所要实现的功能就是把栈顶中保存的断点地址传入EIP便于程序返回到原断点处继续执行。

6.若执行RET前把栈顶的数值改掉，那么RET执行后，EIP中的值就是栈顶的数值，程序会去寻找该值对应的内存空间，会出现不可预知的操作或者报错。

7. invoke 伪指令对应的汇编语句是先把参数push进堆栈，然后执行call指令，最后会在执行完返回原断点时把esp的地址add参数所占的内存数，以释放堆栈内存防止内存泄漏。查看一个利用invoke调printf的函数可看到如下的汇编指令：

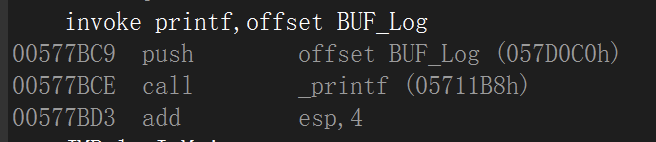


图3.3.3 invoke指令对应的汇编语句

8. 子程序中的局部变量的存储空间是在堆栈中，在断点地址之上。可以用原esp中（用ebp保存后就是ebp中）的地址减相应的偏移量来访问，偏移量是由局部变量所占的内存字节数以及局部变量压入堆栈的次序所决定的。访问局部变量时的地址表达式只能用寄存器变址寻址方式，比如说用ebp保存了原esp地址之后，访问局部变量的地址表达式应该是这样的形式：[ebp-4]。

9.探究：

(1)对于子程序使用NEAR和FAR的阐述如下：如果子程序和主程序在同一个代码段，则使用NEAR，调用发生后，主程序堆栈中只压入IP值；如果子程序和主程序不在一个代码段，则使用FAR，调用发生后，主程序堆栈中将压入CS、IP值。

而在WIN32中，这两种内存模式是相同的。此时的NEAR和FAR没有意义。

在16位体系中：对一个NEAR类型子程序强制使用FAR调用时，由于FAR需要使用32位地址而无法进行；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调用时，不能改变CS寄存器的值，当调用同一个代码段的子程序时可以正常进行，而调用不同代码段的子程序时会出现错误，因为该子程序本来是需要通过CS：IP访问的，而只通过IP访问，访问的是当前CS下的IP，自然会出现错误。

(2) 解释宏和子程序的调用有何不同。宏定义指令在执行之后展开如下所示：

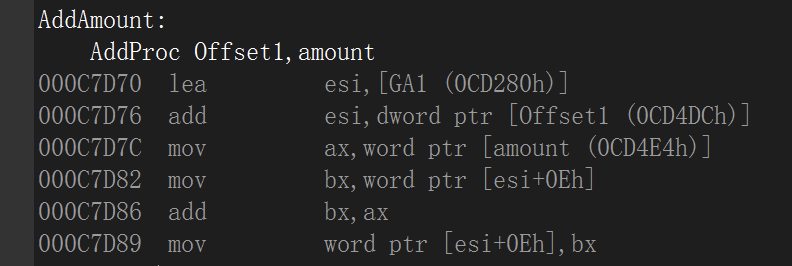


图3.3.4 宏指令执行时替换

宏指令和子程序都可以用来处理程序中重复使用的程序段，但它们有着本质的区别。宏指令是在汇编期间由编译器处理的，所以必须先定义，后调用；而子程序调用是在目标程序运行期间由CPU直接执行的。宏调用是用宏体置换宏指令名，实参替换形参，汇编结束时宏定义也就消失了。子程序调用时不发生这种代码和参数的置换，而是通过EIP和堆栈由CPU控制将程序运行从主程序转向子程序。

## 任务3.2实验过程

### 实验方法说明

1.对任务3.1中的程序，进行改造。首先用C语言程序实现用户登录功能、主菜单的显示功能。创建一个新文件CFunction.c，保存C语言程序。

2.增加“6.添加新商品”的功能并用C语言实现。C语言模块中实现了用户登录、主菜单显示、添加新商品三个函数，而程序的其他部分都是由汇编语言程序实现的。C语言模块和汇编语言模块都要使用GOODS结构体，分别定义。其中C语言程序需要用到的变量都用形参的形式传入。

3.在汇编语言程序中调用C语言程序。

4.在实验过程中，观察分析并回答以下问题

1)观察在C语言程序、汇编语言程序中，分别是如何说明外部变量和函数的；汇编指令访问C的变量时是如何翻译的（观察对应的反汇编代码），C语言语句访问汇编语言定义的变量时是如何翻译的。

2)尝试保证在C语言程序和汇编语言程序中，正确访问商品信息的结构数组。

3)观察不同变量地址之间的关系；根据该关系，实现一个变量名称不出现在语句中的情况下，修改该变量值的功能。（比如，已知int x，y; 假设这两个变量在内存中相邻，就可以用表达式 \*(&x-1)=20修改 y的值）

4)地址类型转换的含义。

5)函数调用语句对应的汇编语句有哪些， 调用函数与被调用函数之间是如何传递信息的。

6)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、偏移的值等。

### 实验记录与分析

1.加入“6.添加新商品”的功能后的测试截图如下所示。

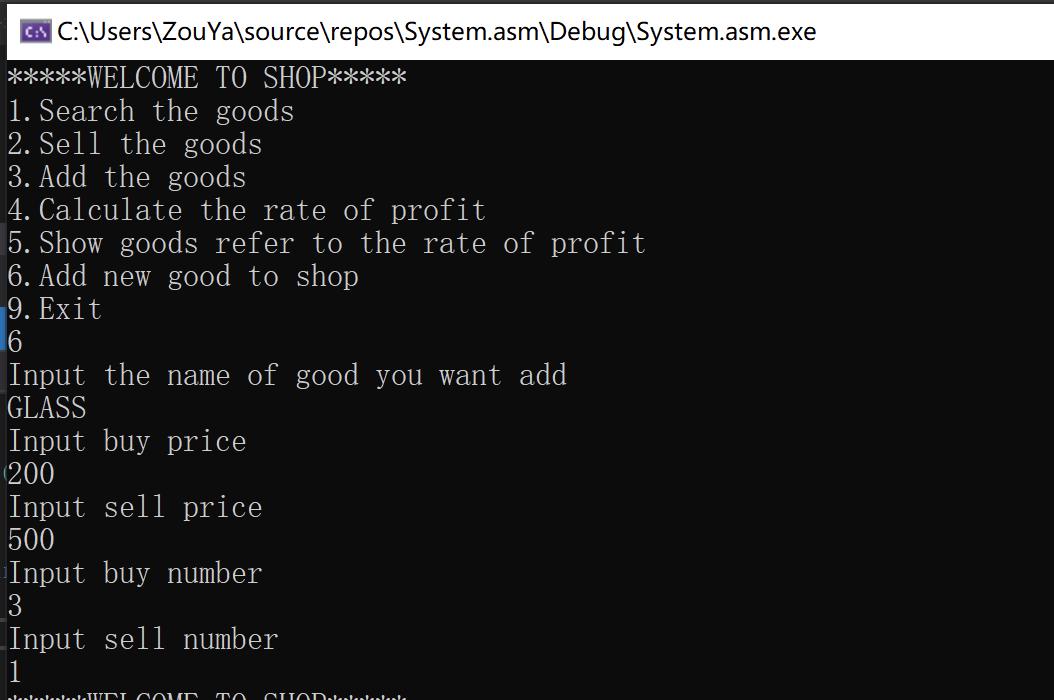


图3.4.1 输入新加入的货物信息

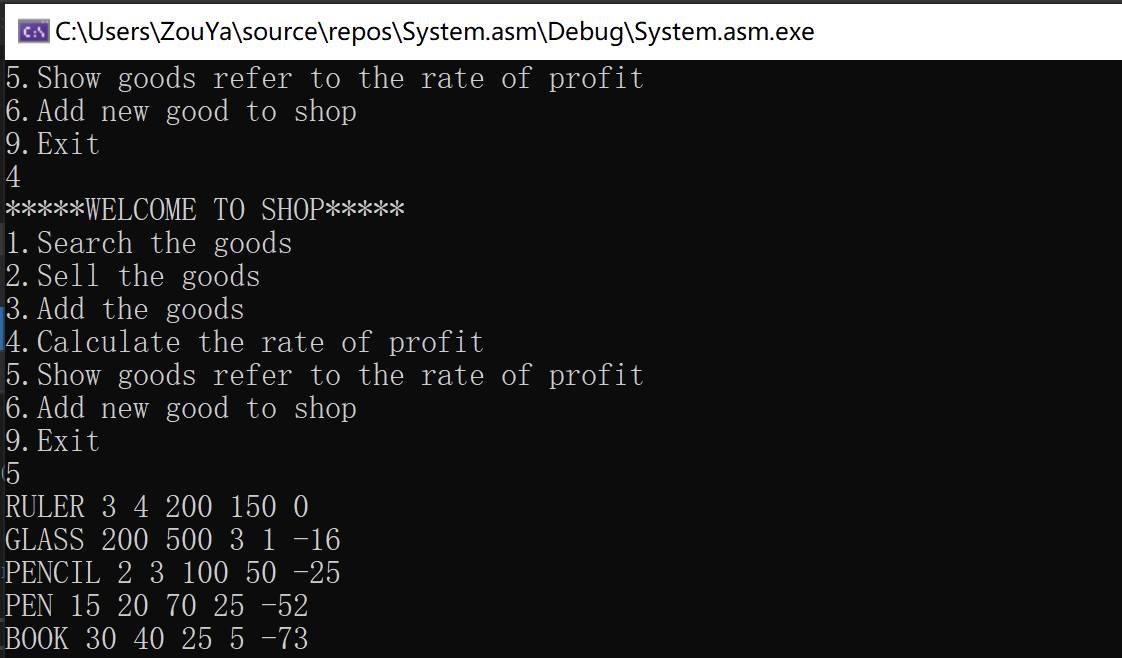


图3.4.2 输入货物信息后计算利润率并排序输出

2.在C语言程序中用include加入外部文件中来使用外部文件中的函数，用extern声明要使用的外部变量，用public声明要提供给外部使用的变量。在汇编语言程序中用includelib加入外部文件，并且要声明外部函数，用extern声明要使用的外部变量，用public声明该变量需要被外部文件引用。在本程序中，从汇编语言程序中调用C语言程序，其声明如下所示：

includelib CFunction.c

Login PROTO C :ptr sbyte,:ptr sbyte

ShowMenu PROTO C

NewGoodToSystem PROTO C :dword

3.观察反汇编窗口，汇编指令访问C的变量时直接访问变量地址，并且标注出该变量地址的类型。如C语言中的int类型，反汇编后将其地址标注成dword ptr来访问。C语言语句访问汇编语言定义的变量时是将其翻译成C语言中定义的变量类型。比如在汇编语句中定义的word类型，在C语言中定义为short。

4. 要保证在C语言程序和汇编语言程序中正确访问商品信息的结构数组，分别在汇编语言程序和C语言程序中定义结构体，结构体中的变量类型一一对应。在汇编语言中调用同一个属性的语句分别如下[ESI].GOODS.RATE（ESI保存一个结构体的起始地址）和&curGood->RATE（curGood指向一个GOODS结构体）。本程序中的定义如下所示：

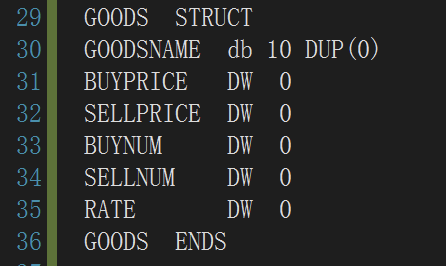


图3.4.3 汇编语言程序中结构体定义

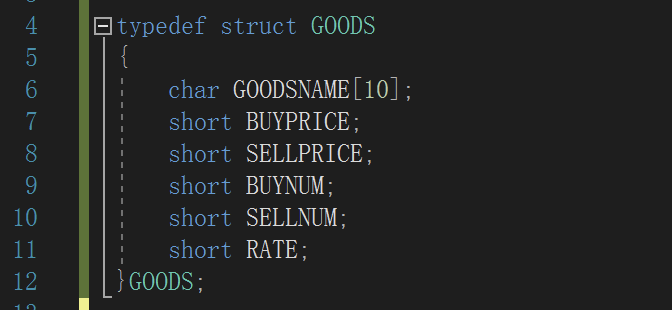


图3.4.4 C语言程序中结构体定义

5. 相邻的不同变量地址之间相差他们变量所占字节数。比如在出货的宏定义中，售出数量与货物信息首地址相差了14个字节数，可以直接用代表地址的ESI+14进行寄存器间接寻址。

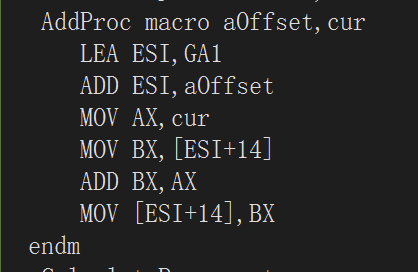


图3.4.5 出货宏定义

6. 地址类型转换是指将同一块内存中的值解释成不同的类型。比如，char a[10];一种地址类型转换的做法是 \*(int \*)a=123。原来的a被解释成char\*，由于char是一个字节的，所以访问a所代表的内存空间也是一个个字节去访问的。但地址类型转换之后，把a解释成了int\*，是以四个字节为单位来访问的。

7. 函数调用语句对应的汇编语句就是首先把函数形式参数从右到左压栈，然后call函数，也就是保存断电地址到堆栈的栈顶，并且把过程名对应的地址传入EIP，进入子程序，退出程序时自动把形参局部变量出栈。调用函数的汇编语句如下所示：

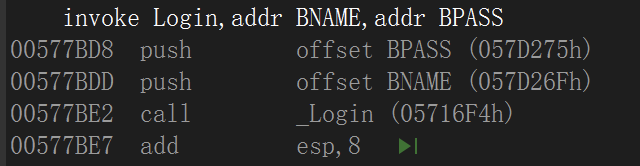


图3.4.6 调用函数对应的汇编语句

主程序调用子程序时，把子程序处理的数据传给子程序需要用到入口参数，子程序处理之后需要返回处理结果即提供出口参数给主程序。参数传递主要有三种方式：寄存器法、约定单元法、堆栈法，无论哪种方式传递参数都需要事先约定好并按照此要求来。高级语言编译器一般在函数调用时的参数传递都采用堆栈法。在进入函数时的一系列汇编指令也是对堆栈进行处理，如下所示：

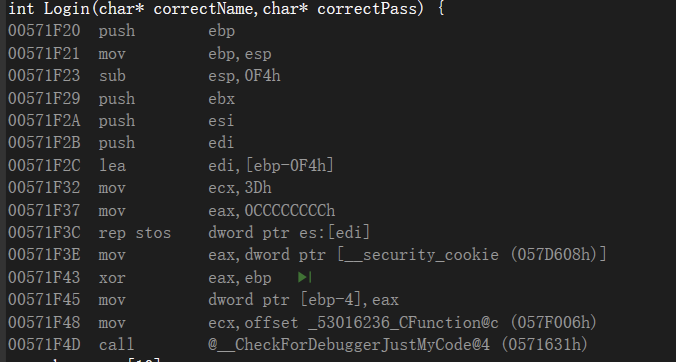


图3.4.7 进入子程序时的汇编指令

使用ebp寄存器的原因是在子程序中不能保证esp不发生变化，这时候用esp获取元素就会出错，所以首先保护ebp寄存器，然后把esp送给ebp，通过ebp来访问局部变量。

8.汇编语言程序编译之后，用反汇编窗口查看，发现同一个汇编语言程序的文件中的语句都会保存在同一块连续的内存地址空间；而C语言程序编译之后，用反汇编窗口查看，发现在同一个C语言程序文件中的函数分布在了不连续的内存地址空间。



图3.4.8 C语言程序代码内存分布情况

在C语言程序中内嵌汇编语言指令，两者的代码会连续保存在一起，情况如下图所示：



图3.4.9 C语言程序内嵌汇编指令内存情况

9. 在C语言程序中，若不考虑上下语句翻译成怎样的机器码而随意嵌入汇编指令语句时，有可能存在出错的风险。因为C语言程序翻译成汇编语言之后，会用到一些寄存器，如果随意改变寄存器的值，就会有出错的风险。比如在C语言程序中随意改变了esp或者ebp的值，导致堆栈中某些元素无法被访问。

10. 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。在C语言中调用scanf时，要先把变量的偏移地址送入寄存器，再把寄存器中的值压栈；而汇编语言中直接可以把变量地址压栈。

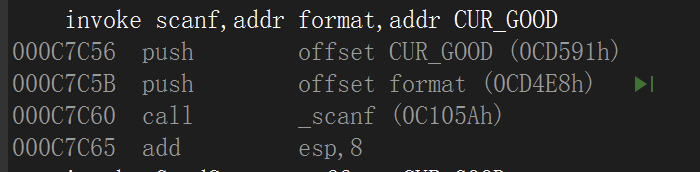


图3.4.10 汇编语言的scanf调用

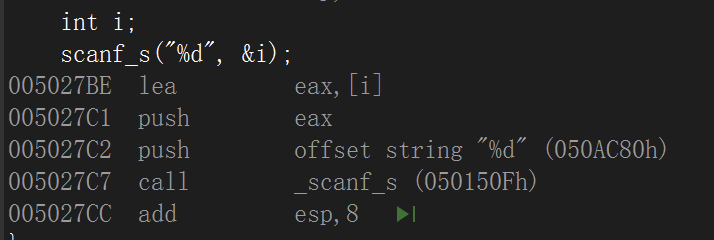


图3.4.11 C语言的scanf\_s调用

## 小结

### 主要收获

本次实验设计面非常广。

在任务3.1中我学习了模块化的思想，把各个功能用子程序的方式来实现，并且分文件保存，比起先前的全用标号和跳转指令而言代码的结构清晰了很多，也更好阅读调试了。学习了如何在子程序和主程序之间传递信息。我最开始用局部变量的时候，对堆栈的操作非常不熟悉，导致每次使用都会思考很久到底如何使用。还学习了使用宏指令，体会了子程序调用和宏指令展开之间的区别。通过探究我还知道了NEAR和FAR的区别，更好地理解了代码段的意义。

在任务3.2中我尝试了C语言和汇编语言混合编程。学会了如何在汇编语言程序中调用C语言函数。学会了在C语言程序和汇编语言程序中，正确访问商品信息的结构数组，更加深刻的明白了C语言变量类型定义和汇编语言中声明的关系，以及变量在内存中究竟是如何保存的。通过探究函数调用对应的汇编指令，我对于调用函数之后究竟是如何跳转如何返回有了更加深入的了解。同时我也更加清晰了C语言和汇编语言之间的区别和关系，体会了两者在实现同一个功能上可能出现的差别和效率问题。

通过本次实验我开始明白了不同的编程语言是可以协同解决一个问题的，而且可以利用不同语言的特点来更好地解决问题；利用汇编语言的知识，能够更好地理解高级语言的内部处理原理与策略，为编写更好的C语言程序、用好C编译器提供支持。

### 主要看法

我的电脑操作系统是macOS，而我用虚拟机WIN10里的VS2019进行汇编语言程序的编写和调试，体验感非常差，每一次编译运行起来都需要消耗较长时间，而且操作起来蛮卡顿的。而用macOS上的VSCode编写配合Dosbox调试，对我来说不太习惯这样的方式。

# 中断与反跟踪

## 实验目的和要求

（1）通过观察与验证，理解中断矢量表的概念；

（2）熟悉I/O访问，BIOS功能调用方法；

（3）掌握实方式下中断处理程序的编制与调试方法；

（4）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，熟悉动态与静态反汇编工具，深刻理解汇编语言的特殊能力；

（6）完成指定功能的程序设计与调试，提升对计算机系统的理解与分析能力。

## 实验内容

任务4.1：利用中断实现**实时时间显示。**

在DOSBox环境中实现时分秒信息在窗口指定位置的显示。其中，指定位置信息来源于程序中定义的变量的内容；所实现的程序运行后需要驻留退出，并能避免重复安装。

要求能在TD下观察中断矢量表、观察已有的某个中断处理程序的代码、读取CMOS中某个单元内容；能在TD下调试中断处理程序(给出调试方法和关键的调试记录)。

**任务4.2：数据加密与反跟踪**

在任务3.1完成的程序的基础上，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）。当进货价被加密后，原来对该数据进行处理的功能在执行处理操作前就需要解密进货价。

可以采用计时、动态修改执行代码、间接寻址、代码中穿插数据定义或无关代码、堆栈检查等反跟踪方法中的几种方法组合起来进行反跟踪（要求:采用包括动态修改执行代码在内的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。

**注意：**两人一组（在实验记录中说明同组是谁），每人实现一套自己选择的加密与反跟踪方法，把执行程序交给对方解密（解密时间超过半小时的，说明反跟踪方法基本有效）。如何设计反跟踪程序以及如何跟踪破解，是本次实验报告中重点需要突出的内容。

## 任务4.1实验过程

### 设计思想及存储单元分配

在NEW08H子程序中，CS寄存器表示所在的代码段，用AX、BX作为各种设置的中间变量，DX中为光标的位置，DS、ES都指向了CS。

在GET\_TIME子程序中，用AL作为存取时、分、秒的中间量保存寄存器，当转换成ASCII码值时，需要扩充到AH，即使用整个AX来保存时、分、秒的值。

在主程序中，同样用AX作为各种中间值的保存寄存器，用DS、ES指向相应代码段的地址。

其他的数据变量如下所示：

COUNT：计数

HOUR：时的ASCII码

MIN ：分的ASCII码

SEC： 秒的ASCII码

BUF\_LEN：计算显示信息长度

CURSOR：原光标位置

OLD\_INT：原INT 08H的终端矢量

### 流程图

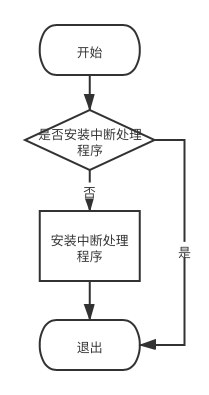


图4.3.1 主程序流程图

### 源程序

NEW08H PROC FAR

…….

NEW08H ENDP

GET\_TIME PROC

……

GET\_TIME ENDP

BEGIN:

XOR AX,AX

MOV DS,AX

MOV AX,OFFSET NEW08H

CMP AX,DS:[08H\*4]

JNE BEGIN\_N

MOV AX,ES

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

JE NEXT

BEGIN\_N:PUSH CS

POP DS

MOV AX,3508H

INT 21H

MOV OLD\_INT,BX

MOV OLD\_INT+2,ES

MOV DX,OFFSET NEW08H

MOV AX,2508H

INT 21H

NEXT: MOV AH,0

INT 16H

CMP AL,'q'

JNE NEXT

MOV DX,OFFSET BEGIN+15

SHR DX,4

ADD DX,10H

LDS DX,DWORD PTR OLD\_INT

MOV AX,2508H

INT 21H

MOV AH,4CH

INT 21H

CODE ENDS

END BEGIN

### 实验步骤

1. 参考“参考资料【2】的P224例6.2，以及【2】的P330 INT 10H的功能，学习并编写源程序。
2. 主程序中加入功能：判断中断处理程序是否已经安装。
3. 在TD下观察中断矢量表、观察已有的某个中断处理程序的代码、读取CMOS中某个单元内容，在TD下调试中断处理程序；并给出调试方法和简要记录关键的调试记录。

### 实验记录与分析

1.用DOSBox运行程序，调用TD调试程序，设置时间显示位置为窗口右上角，且时间显示颜色为白色，其显示效果如下：

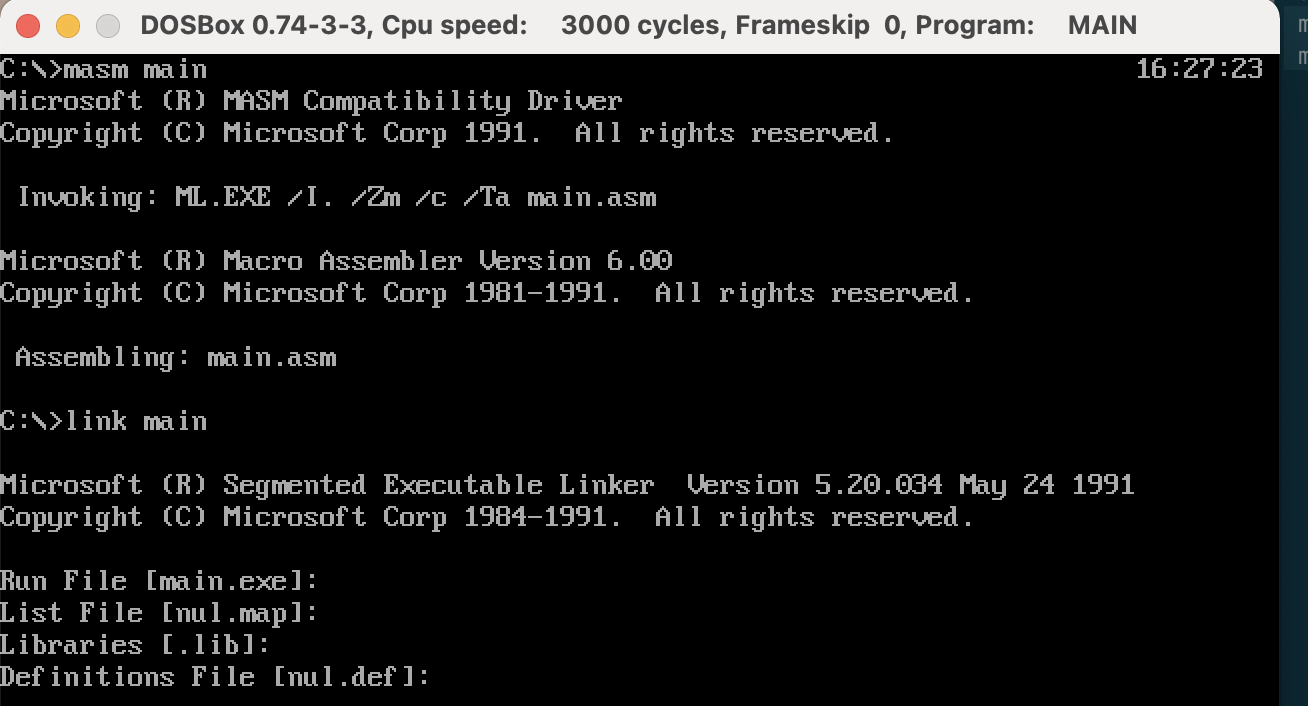


图4.3.2 时间显示效果截图

2. 在TD下观察中断矢量表。首先直接TD运行程序，在数据区中输入00:20，观察8号中断的中断矢量表如下

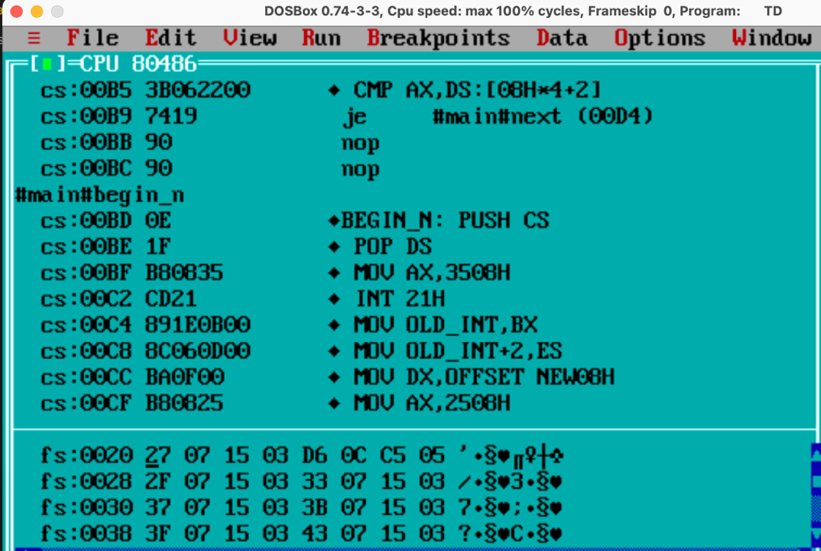


图4.3.3 8号中断矢量表观察截图

在运行一次自定义中断程序后，观察BX的值（程序会把8号中断的中断矢量表值送入BX寄存器）：

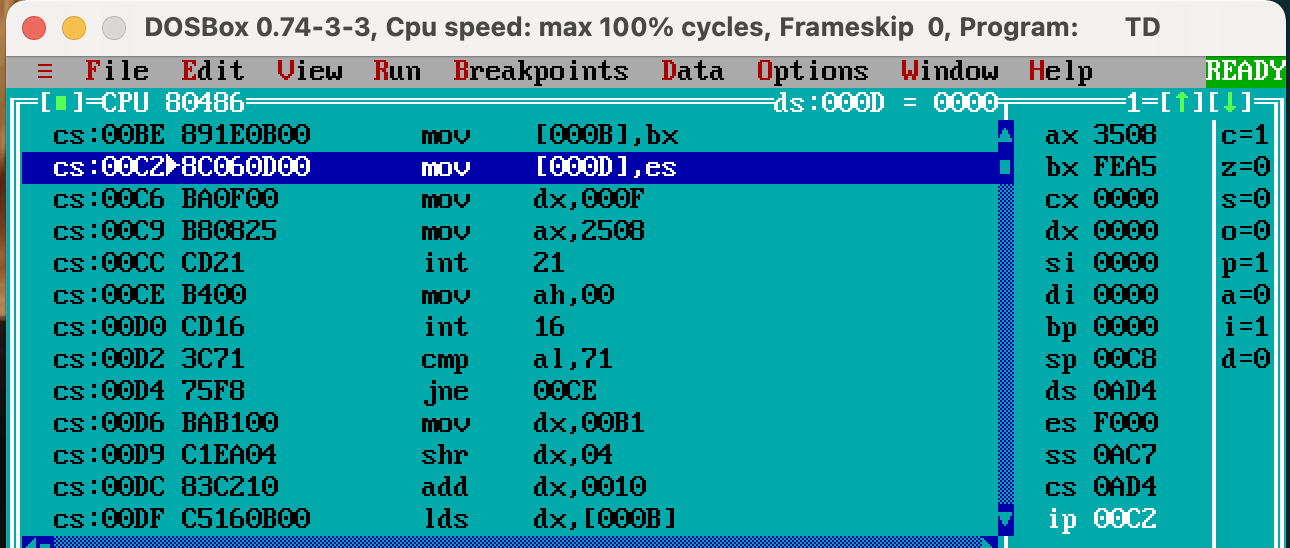


图4.3.4 程序运行后8号中断矢量表观察截图

5.观察已有的某个中断处理程序的代码可以通过中断矢量表中保存的数据，也就是对应程序入口地址，在数据区输入该地址即可观察。

4.在TD中调试程序，基本方法是用F8进行单步调试，F9进行run，也可以用F2在特定语句上设置断点来运行程序，设置断点后截图如下所示（红色行即为断点位置）：

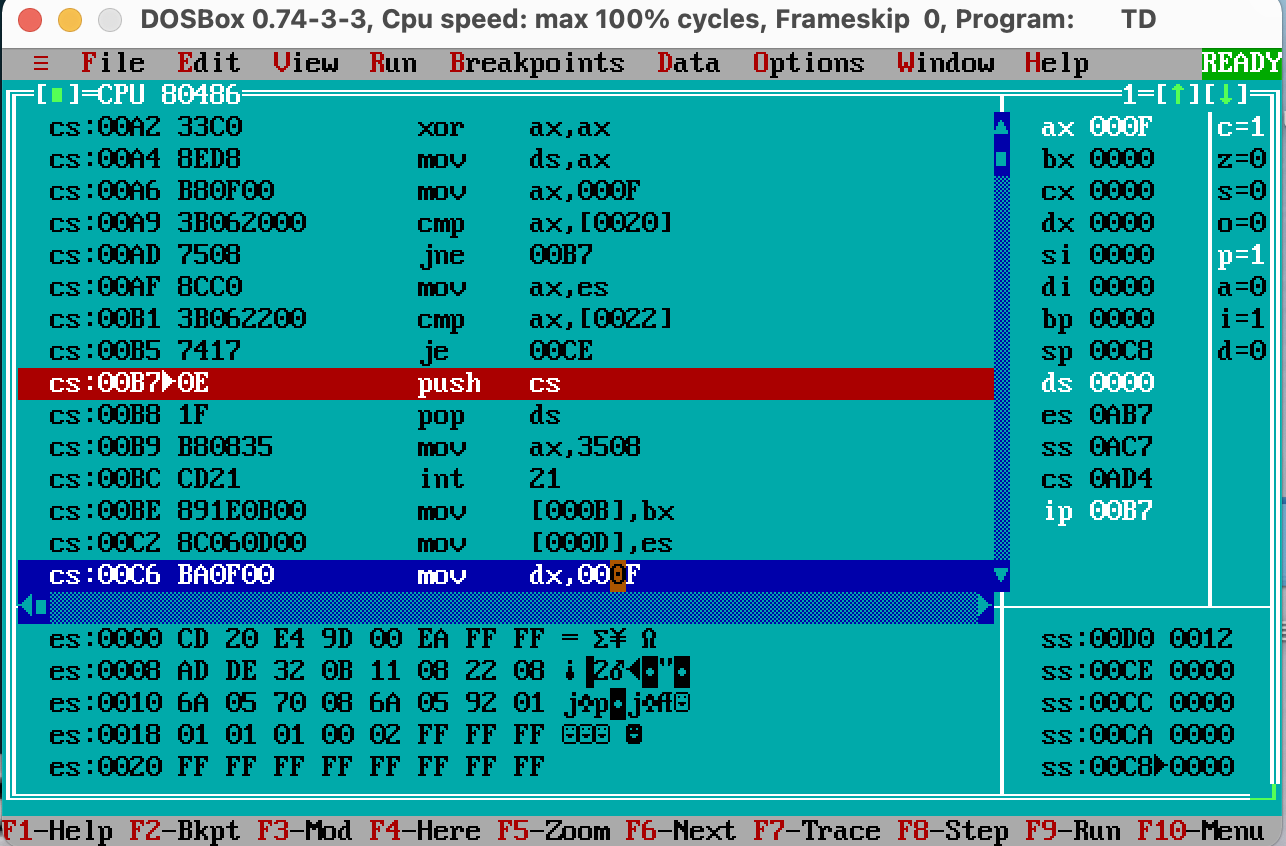


图4.3.5 设置断点截图

## 任务4.2实验过程

### 实验方法说明

1. 对密码以及商品进货价信息进行加密。加密方式采用的是与数据段中定义的老板姓名‘Z’第一个字符来进行异或得到。
2. 设计相应的解密函数，在验证输入正确的用户名和密码后进行解密再进行之后的程序运行。
3. 反跟踪设计：采用动态修改执行代码、代码中穿插数据定义或无关代码这两种反跟踪方法组合进行程序反跟踪。

### 实验记录与分析

1.对密码以及商品进货价进行异或加密，如下所示：

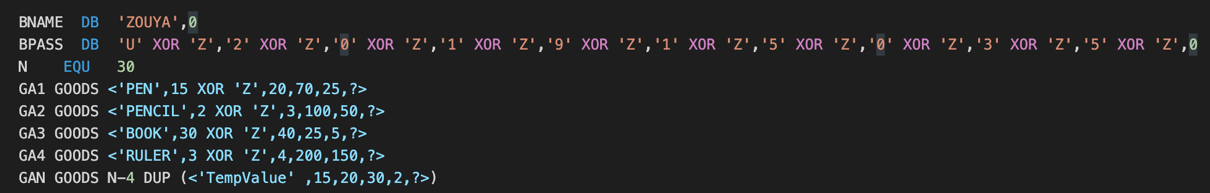


图4.4.1 异或加密

打开VS2019内存调试，查看在内存里的数据，确认BPASS以及进货价的内容不是以明文显示的。

2.编写解密函数，并在解密函数中穿插入动态修改执行代码、代码中穿插数据定义或无关代码这两种反跟踪方法进行程序反跟踪。结合入实验3中的内容，所编写解密函数在subProgram.asm文件中。

在数据段中增加如下数据定义，用于动态修改执行代码。

machine\_code db 0E8H,1DH,0,0,0

db 'Very Good',0DH,0AH,0

db 0E8H,0CH,0,0,0

db '12345',0DH,0AH,0

len = $ - machine\_code

解密函数以及所要用到的display函数的代码如下所示：

Cracker proc uses ESI EAX EBX

call display

msg1 db 'Very Good',0DH,0AH,0

call display

msg2 db '123456',0DH,0AH,0 ;代码中穿插数据定义或无关代码

Begin:

LEA ESI,GA1

MOV BX,0

LoopToCrack:

ADD ESI,10

MOV AX,word ptr [ESI]

XOR AX,'Z'

MOV word ptr [ESI],AX

ADD ESI,10

ADD BX,1

CMP BX,4

JZ CopyCodeStart

JMP LoopToCrack

CopyCodeStart:

mov eax,len

mov ebx,40H

lea ecx,CopyHere

invoke VirtualProtect,ecx,eax,ebx,offset oldprotect

mov ecx,len

mov edi,offset CopyHere

mov esi,offset machine\_code

CopyCode:

mov al,[esi]

mov [edi],al

inc esi

inc edi

loop CopyCode

CopyHere:

db len dup(0)

ret

Cracker endp

display proc

pop ebx

p1:

cmp byte ptr [ebx],0

je exitDisplay

invoke putchar,byte ptr [ebx]

inc ebx

jmp p1

exitDisplay:

inc ebx

push ebx

ret

display endp

通过穿插display函数，输出“Very Good”和“123456”两个字符串而完成反跟踪设计。

3.运行Cracker函数之后，观察反汇编代码。

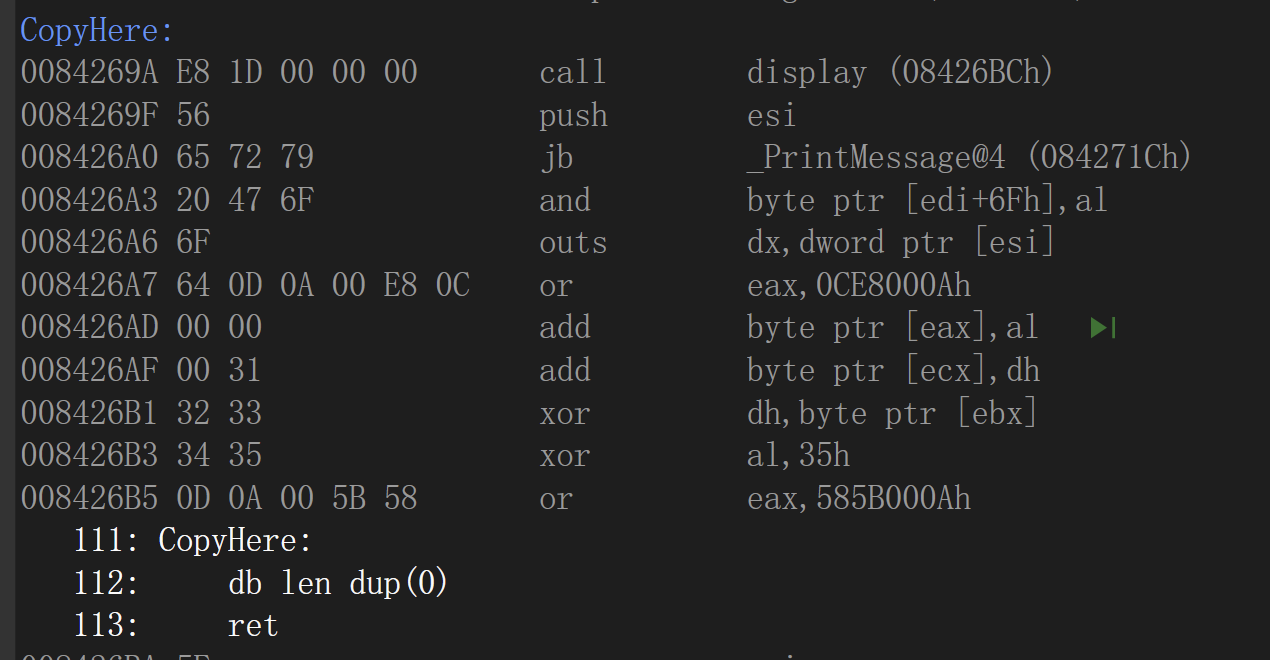


图4.4.2 Cracker运行完后CopyHere的内容

## 任务4.3实验过程

### 实验方法说明

1.利用静态反汇编工具IDA将执行程序反汇编成源程序，观察源程序的特点。

2.利用文件编辑工具进行暴力破解，比如直接观察和修改执行文件中的数据信息；或者直接修改执行程序的机器码，绕过输入密码后进行密码正确性检查的代码。

3. 静态反汇编。利用IDA直接处理待破解程序.exe文件，将可执行文件翻译成比较直观易读的汇编语言的源程序，再由人工阅读该源程序代码，推断上下文程序含义。

4.利用VS2019动态跟踪调试，注意观察反跟踪的代码。若碰到反跟踪程序段，则需要配合设置断点的方法，绕过反跟踪程序指令。

### 实验记录与分析

1.我解密的程序是蓝冰瑛的。（我、蓝冰瑛、张弘弢三人一组）

2.用IDA反汇编后直接观察程序源代码以及数据栈内容，得到用户名为“LANBINGYING”。

3.继续观察IDA中反汇编的汇编源程序，在观察了很久之后，只能找到到如下指令，这是反跟踪解密的部分：

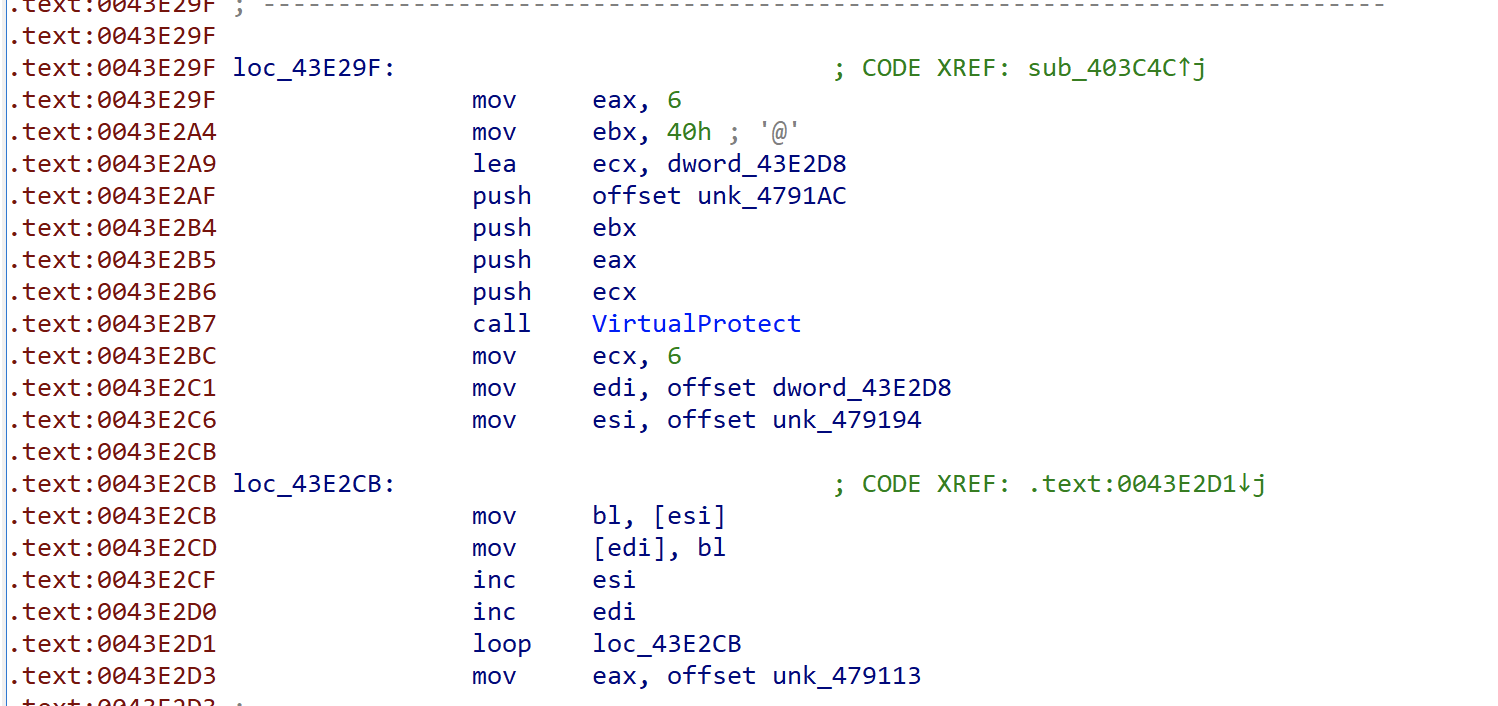


图4.5.1 解密及反跟踪部分

接着往下观察，发现以下代码，由无意义的内容组成，无法从此解密程序。

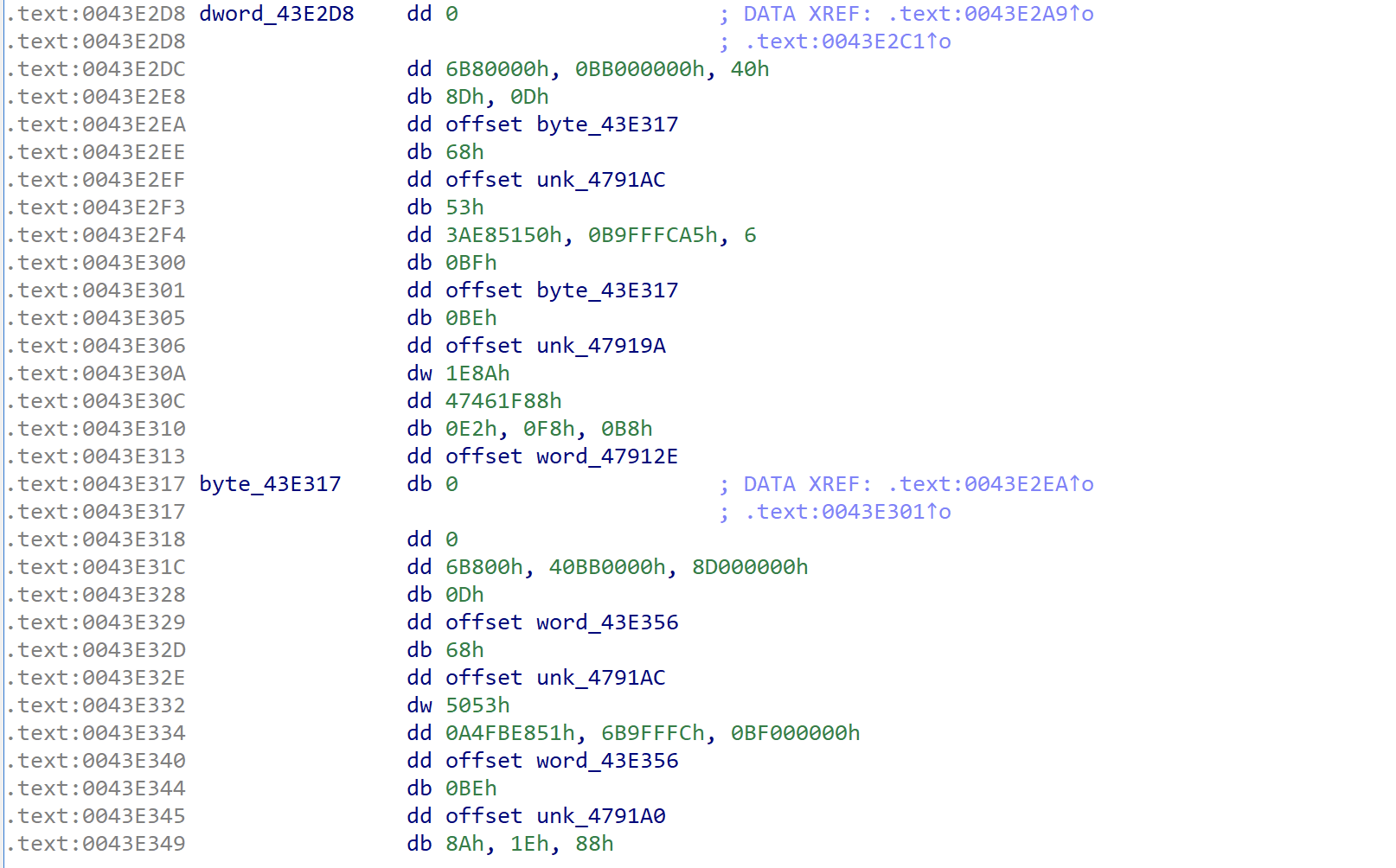


图4.5.2反跟踪内容

4.接着我想尝试继续通过这个源代码找出解密部分指令，尝试了直接寻找xor异或指令，或者是寻找有意义指令程序，都没有成功。然后尝试用动态反汇编的方式。

根据该函数在main函数中的调用关系，利用VS2019动态调试运行，结合上推测的位置打上断点，反复寻找解密程序对应的指令，最终得到如下：

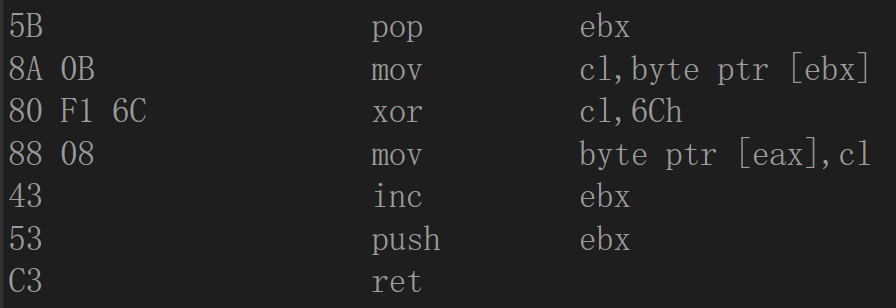


图4.5.3 解密指令

观察得到异或的值为0x6c，也就是字符‘l’。根据这个，再结合到在内存中观察到保存的密码内容，可以得出密码为‘3’。输入成功进入系统：

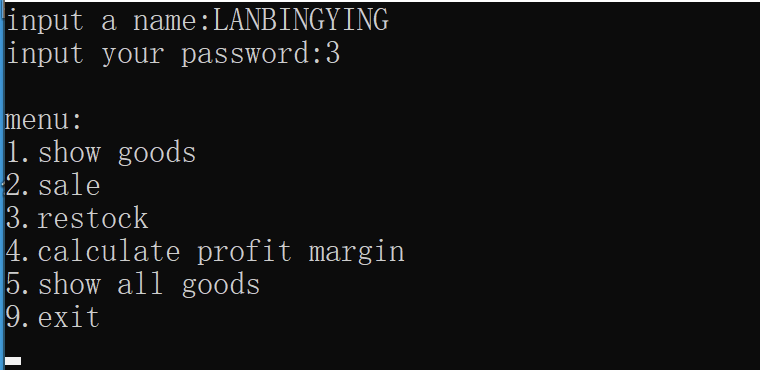


图4.5.4 进入系统

5.思考题。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?

（1）C语言反跟踪：插入无关代码；调用调试监测函数，调试即退出；动态函数指针；动态修改代码

（2）汇编语言的特殊之处：汇编语言是更加底层的语言，在反跟踪中可以做到在更底层来改变指令语句，因为汇编语句基本上就是机器指令的翻译，所以和机器指令的交互度很高。

## 小结

### 主要收获

在本节的学习过程中学习并理解了中断矢量表的概念；了解I/O访问，尝试了BIOS功能调用；学习了实方式下中断处理程序的编制与调试方法；熟悉了内存的一些基本操纵技术；熟悉了跟踪与反跟踪的技术，使用并学习了动态与静态反汇编工具，深刻理解汇编语言的特殊能力。

在任务4.1中了解了如何调用中断处理程序，学会了如何获取以及显示系统时间，知道了需要从用IO指令从CMOS芯片中读取当前时钟的时、分钟和秒信息，并且要实现所实现的程序运行后需要驻留退出并能避免重复按照的功能，对时钟中断和CMOS中保存的数据有了直观的了解。同时通过这次任务，让我更加熟练的使用TD来调试程序。

在任务4.2中，我学会了动态修改执行代码、代码中穿插数据定义或无关代码两种反跟踪方式，并且将它改变放入我的程序中。一开始由于我的程序和书上给的例子中的偏移地址不一致，导致运行不了，在我的单步调试和观察反汇编代码之下，终于找到了准确的偏移地址，实现了反跟踪。

在任务4.3中，我花费了大量的时间，一方面是同学的反跟踪方法确实达到了效果，另一方面我对于反汇编代码也没有熟悉到可以看到并且直接推断的程度。但在这次实验里，我阅读了大量的反汇编代码，不仅学到了反汇编之后的指令，也发现了我们的程序在编译链接之后会有大量的指令被编译器自动加上，这是以前没有接触到的。通过静态反汇编、动态反汇编几种方式，以及大量时间的投入，我对反汇编指令有了更深刻的了解。同时我也明白了反跟踪语句的作用。

### 主要看法

本次实验中最有意思的事情就是进行跟踪和反跟踪，使用了一些针对性的混淆技巧，而反跟踪则需要针对对方的手段做出应对。这些其实也是从另外一个角度去了解汇编本身，只有对汇编语言和反汇编的方法了解清楚，才能正确的实现自己的目的。

我有听说网络安全专业的同学的课程会大量接触到汇编语言，我想这就是其中的原因之一吧。现在网络空间安全是一个比较热门的领域，相信学好汇编语言、了解反汇编、了解反跟踪会是我们涉足这个领域的第一步。

本次实验中用到了IDA的反汇编软件，也尝试了VS2019的反汇编功能。总体来说对于反汇编来说都不是特别好用。

# 16/32/64位编程比较

## 实验目的和要求

（1）了解16/32/64位环境下程序设计的不同特点及配套的开发工具；

（2）通过完成指定的程序设计，观察并理解汇编语言在不同环境下的基本特点。

## 实验内容

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

编写一个基于窗口的WIN32程序的菜单框架，具有以下的下拉菜单项：

File Action Help

Exit Compute Rate About

List Sort

点菜单File下的Exit选项时结束程序；点菜单Help下的选项About，弹出一个消息框，显示本人信息。点菜单Action下的选项Compute Rate、List Sort将分别实现计算利润率或在窗口中显示按利润率排序后的所有商品各项信息的功能。

**任务5.2** 在VS2019下调试一个x64程序，观察与32位程序的不同之处。**参考程序见附件**。

**任务5.3** 查阅华为鲲鹏服务器所采用的CPU （即ARMv8系列）的汇编语言编程资料，体会与80X86体系的异同。主要关注CPU内寄存器、**段的定义方法**、指令语句及格式的特点、子程序调用的**参数传递与返回**方法、与C语言混合编程、**开发环境**等方面。**参考阅读材料见附件**（如：ARM汇编技术简介.pdf， ARM基础实验手册.pdf等）。

## 任务5.1实验过程

### 实验方法说明

1. 实验环境条件：2.3 GHz Quad-Core Intel Core i5，8 GB内存；虚拟机环境下WINDOWS 10 VS2019。

2.基于模版“Win32汇编窗口程序环境与例-bpx2021.5”进行WIN32程序编程改写，依次实现功能，包括更改resource.h和resource.inc中对常量的设置以及Menu\_Dialog.rc对菜单模式的设置。

3.加入网店商品信息后台管理系统中的相关货物信息以及相应将分别实现计算利润率或在窗口中显示按利润率排序后的所有商品各项信息的功能函数，并与相应菜单项对应。

### 实验记录与分析

1. 所实现的基于窗口的WIN32程序的菜单框架如下所示：

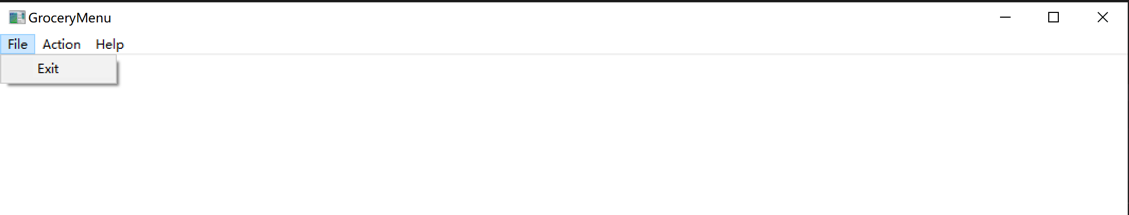


图5.3.1基于窗口的WIN32程序的菜单框架

点击“File”中的Exit选项，弹出确认消息框，点击右上角关闭也同样会产生确认消息框，点击确认即关闭程序：

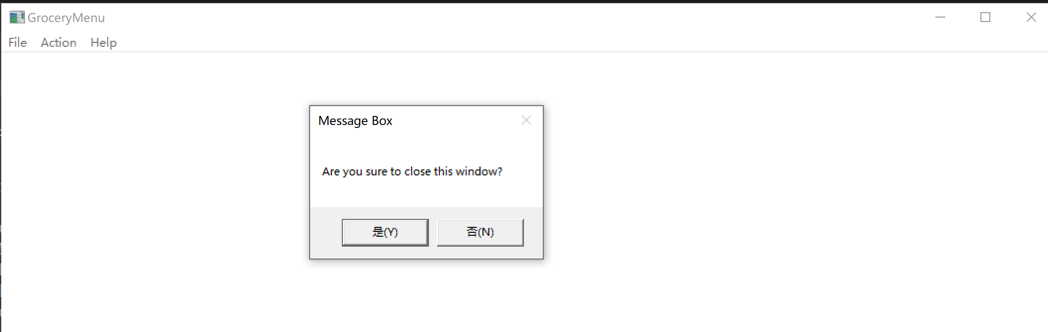


图5.3.2 确认关闭消息框

点击“Action”下的选项 Compute Rate将实现计算利润率，点击List Sort选项在窗口中显示按利润率排序后的所有商品各项信息的功能。

“Action”下的选项如下所示：

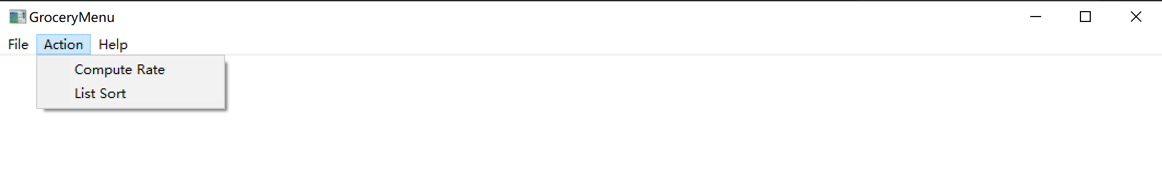


图5.3.3 “Action”下的选项

计算利润率之前直接点击List Sort：

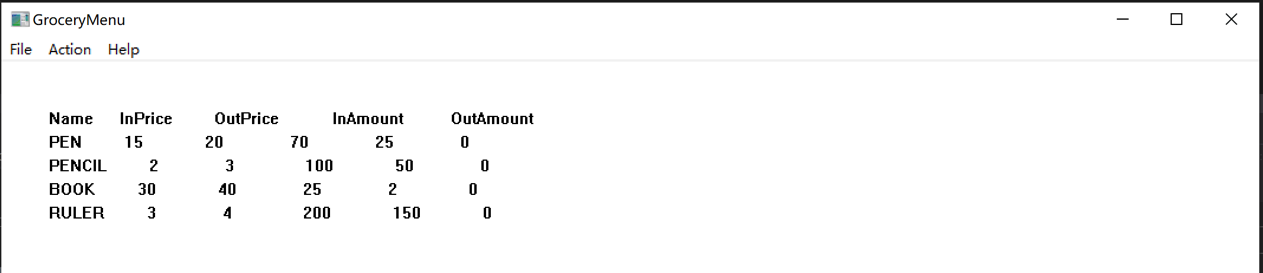


图5.3.4 直接点击List Sort选项

点击选项 Compute Rate实现计算利润率，再点击List Sort，得到排序后的信息如下所示：

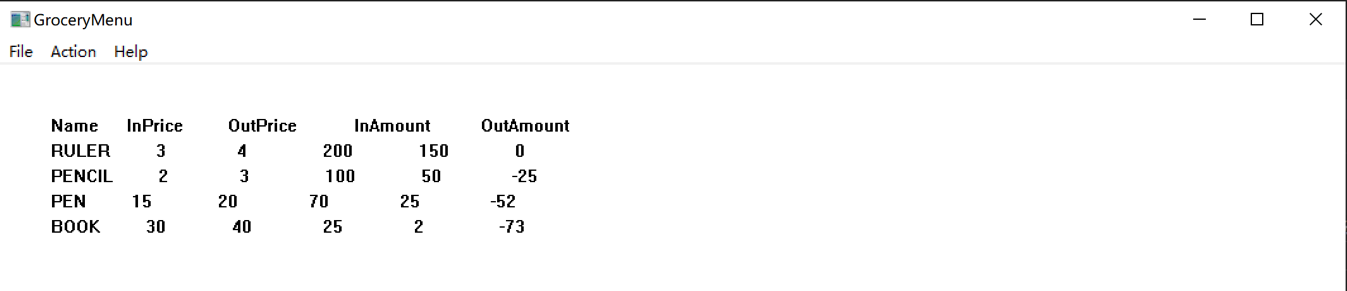


图5.3.5 计算利润率后排序

“Help”下的About选项如下所示：

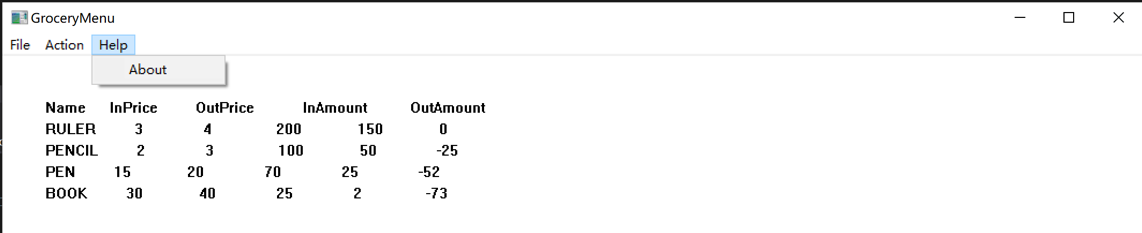


图5.3.6 “Help”下的About选项

点击About选项显示系统本人信息：

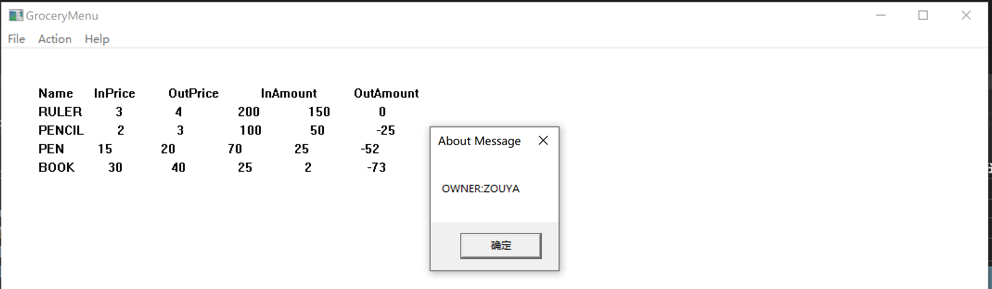


图5.3.7 点击About选项弹出消息框

2.一开始我没有使用WIN32程序模版来编写程序，而是直接自己创建文件照着书编写程序。但是运行的时候却发现报错打不开“windows.inc”“user32.inc”等文件，这时候才发现问题。

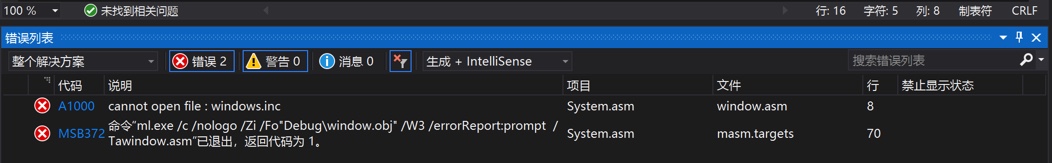


图5.3.8 程序报错

后来发现给了的模版，于是将写好的程序移过去，并修改了resource.h和resource.inc中对常量的设置以及Menu\_Dialog.rc对菜单模式的设置之后，便可以正确运行了。虽然自己编写程序过程中花费了大量时间，但是因为自己手敲过每一条语句而对WIN32编程有了更深刻的理解，对语句的功能更加了解了，对句柄等概念也有了更清晰的认识。

## 任务5.2实验过程

### 实验方法说明

1. 准备实验环境VS2019.
2. 运行所给的x64程序，观察x64汇编的特点。

### 实验记录与分析

1. 将解决方案平台切换至X64，在项目“属性-链接-高级”里指定start为入口点。

2. 寄存器名称的构成体系有很大不同，开头字母都变成了R，出现了R加上数字的寄存器，如下图；但是常见指令基本都一样，没有变化。

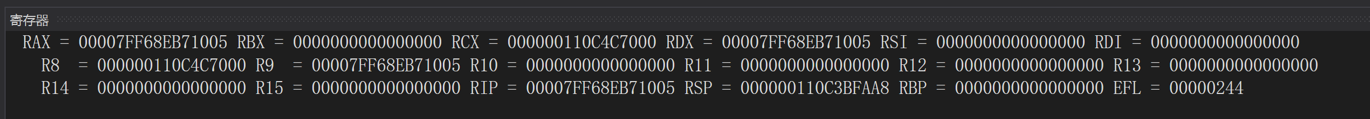


图5.4.1 x64下寄存器的值

观察可见寄存器中的内容是由16位十六进制数字组成的，也就是寄存器中保存了64位二进制。

3．观察反汇编代码

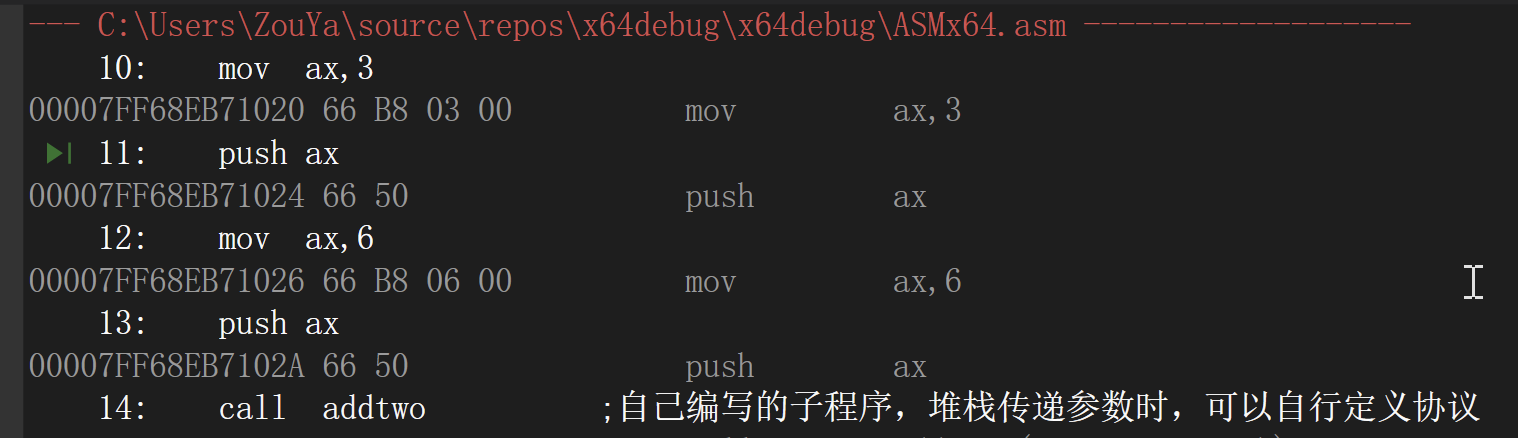


图5.4.2 x64程序反汇编代码

可以看出指令的保存地址也变成了64位的。

4.该程序的运行结果如下所示：



图5.4.3 x64程序运行结果

## 任务5.3实验过程

### 实验方法说明

1.查阅华为鲲鹏服务器所采用的CPU （即ARMv8系列）的汇编语言编程资料，体会与80X86体系的异同。

2.观察CPU内寄存器、段的定义方法、指令语句及格式的特点、子程序调用的参数传递与返回方法、与C语言混合编程、开发环境等.

### 实验记录与分析

1.段定义

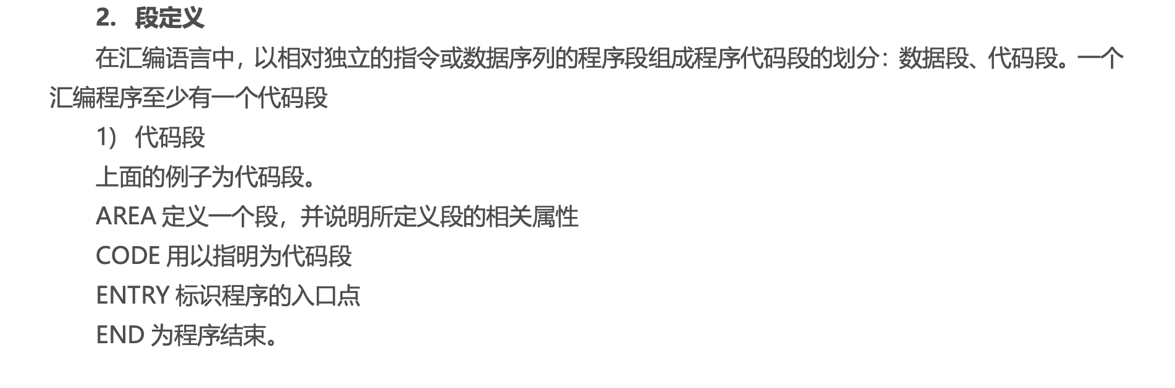


图5.5.1 段定义（包括代码段）



图5.5.2 数据段

2.子程序参数传递和返回的方法如下

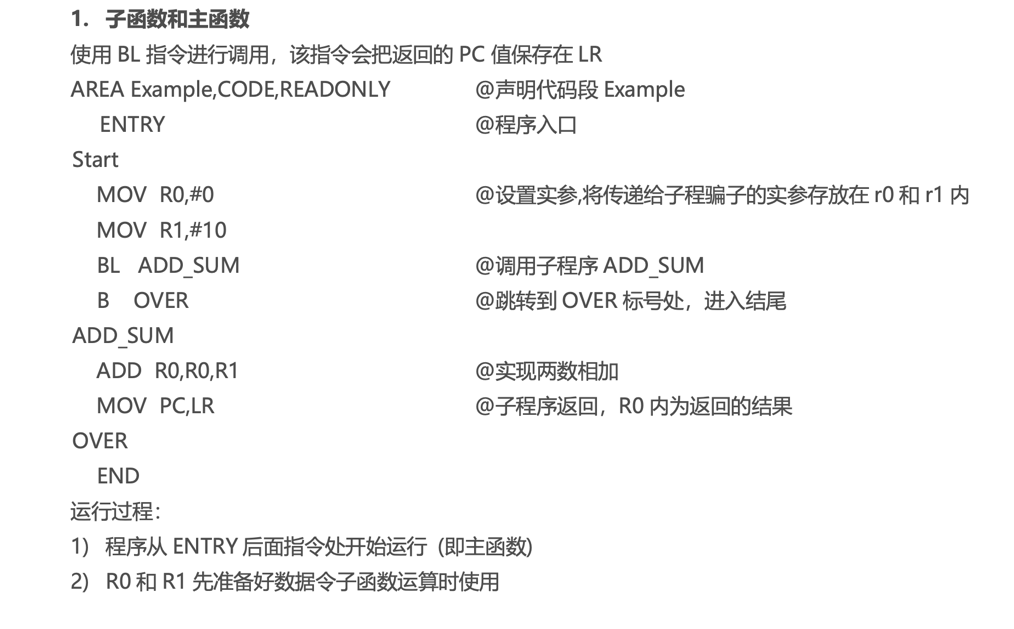


图5.5.3 子程序调用方法1

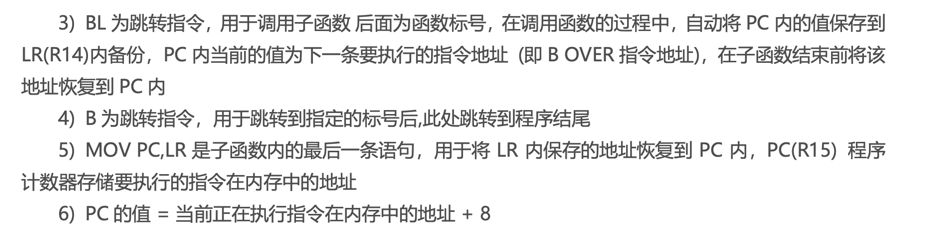


图5.5.4 子程序调用方法2

3.开发环境为ARMv8环境。

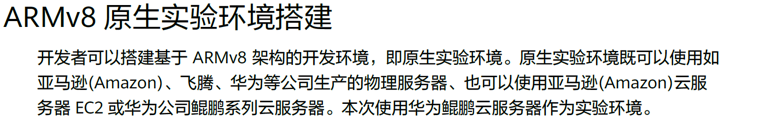


图5.5.5 开发环境

## 小结

### 主要收获

在任务5.1的实验过程中，我了解了32位窗口程序如何编写，并且自己实现了常见的基础窗口程序，如消息提示窗口等。同时对WIN32编程有了更深刻的理解，对语句的功能更加了解了，对句柄等概念也有了更清晰的认识。

经过了任务5.2的实验，我学习了X64汇编语言程序的特点以及了解到了如何运行和编写基本的X64汇编语言程序。

通过任务5.3的实验，我学习了ARM架构的相关知识：基本对ARMv8的体系机构有一个初步的认识，了解了段（包括代码段、数据段）的定义方法、指令语句及格式的特点、子程序调用的参数传递与返回方法、与C语言混合编程方式、ARM开发环境等知识。同时也对ARM64和X86的差异点有了直观的感受。

### 主要看法

16位程序和32位程序的比较：

16位程序实方式下段地址和偏移地址的寻址方式在32位程序中不再存在，在32位中统一采用flat的内存地址模式寻址。16位操作系统中的程序运行在RING0级，也就是说普通程序和操作系统程序运行在同一个级别并且拥有最高权限；而32位操作系统中的程序一般只拥有3级别运行权限，程序的所有操作都受到操作系统控制。16位操作系统的可执行文件格式和32位操作系统的可执行文件格式不同。32位的Windows操作系统运行在CPU的保护模式之上，而16位的系统则运行在CPU的实模式上。

32位程序和64位程序的比较：

从功能上讲，64位汇编较之32位汇编最大优势就是能够支持64位整数的处理，32位汇编较之16位汇编最大优势就是能够支持32位整数的处理。数据位数的增大，意味着数据处理能力的增强，通常也会增加处理速度。64位程序和32位程序使用编译和链接器将源程序翻译成机器语言程序时，两者的编译链接器不同，两种编译器对伪指令的支持有较大的区别。64位有16个64位通用寄存器，而32位只有8个32位寄存器。相应的标志寄存器和指令指针也会由32位扩展成64位，数据的处理能力明显增强。

对DOSBox下的编程环境、WINDOWS下的VS2019、支持ARM的gcc等进行简要的比较：

（1）DOSBox是一个使用SDL库的DOS模拟器，这使得DOSBox可以很容易地移植到不同的平台。同时使用MASM，LINK，TD等程序对汇编程序进行编译和连接等。可以使用TD对汇编程序进行观察和Debug。

（2）VS2019是Windows平台下的APP，创建程序需要单独建立一个项目，然后选择MASM作为自定义生成依赖项。VS2019是一个集成开发环境，可以更方便的管理文件和编译器，调试更加方便直观。

（3）支持ARM的gcc用于编译 ARM架构下的汇编语言程序，用于编译 ARM 架构的裸机系统，一般适合 ARM7、Cortex-M 和 Cortex-R 内核的芯片使用。

# 参考文献

[1]许向阳. x86汇编语言程序设计. 武汉：华中科技大学出版社，2020（第19章，第3章、第4章、第5章）

[2]许向阳. 80X86汇编语言程序设计上机指南. 武汉：华中科技大学出版社， 2007

[3]王元珍，曹忠升，韩宗芬. 80X86 汇编语言程序设计. 武汉：华中科技大学出版社，2005

[4]汇编语言课程组. 《汇编语言程序设计实践》任务书与指南，2021

[5]……