

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实践**

**专业班级：计算机科学与技术计卓1901班**

**学 号： U201914873**

**姓 名： 刘奕康**

**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 - 3 -](#_Toc74052525)

[0.1 课程目标 - 3 -](#_Toc74052526)

[0.2 成绩构成 - 3 -](#_Toc74052527)

[0.3 实验任务的总体描述 - 3 -](#_Toc74052528)

[1 编程基础 1](#_Toc74052529)

[1.1 实验目的与要求 1](#_Toc74052530)

[1.2 实验内容 1](#_Toc74052531)

[1.3 任务1.1实验过程 5](#_Toc74052592)

[1.3.1 实验方法说明 5](#_Toc74052593)

[1.3.2 实验记录与分析 5](#_Toc74052594)

[1.4 任务1.2的实验过程 9](#_Toc74052595)

[1.4.1 实验方法说明 9](#_Toc74052596)

[1.4.2 实验记录与分析 10](#_Toc74052597)

[1.5 任务1.3的实验过程 17](#_Toc74052598)

[1.5.1 设计思想及存储单元分配 17](#_Toc74052599)

[1.5.2 源程序 19](#_Toc74052600)

[1.5.3 实验步骤 24](#_Toc74052601)

[1.5.4 实验记录与分析 24](#_Toc74052602)

[1.6 小结 26](#_Toc74052603)

[1.6.1 主要收获 26](#_Toc74052604)

[1.6.2 主要看法 27](#_Toc74052605)

[2 程序优化 27](#_Toc74052606)

[2.1 实验目的与要求 27](#_Toc74052607)

[2.2 实验内容 27](#_Toc74052608)

[2.3 任务2.1实验过程 28](#_Toc74052611)

[2.3.1 实验方法说明 28](#_Toc74052612)

[2.3.2 实验记录与分析 29](#_Toc74052613)

[2.4 小结 32](#_Toc74052614)

[2.4.1 主要收获 32](#_Toc74052615)

[2.4.2 主要看法 32](#_Toc74052616)

[3 模块化程序设计 32](#_Toc74052617)

[3.1 实验目的与要求 32](#_Toc74052618)

[3.2 实验内容 33](#_Toc74052624)

[3.2.1 任务3.1的实验内容 33](#_Toc74052625)

[3.2.2 任务3.2的实验内容 33](#_Toc74052637)

[3.3任务3.1实验过程 34](#_Toc74052649)

[3.3.1设计思想及存储单元分配 34](#_Toc74052650)

[3.3.2实验步骤 40](#_Toc74052651)

[3.3.3实验记录与分析 41](#_Toc74052652)

[3.4任务3.2实验过程 44](#_Toc74052653)

[3.4.1实验方法说明 44](#_Toc74052654)

[3.4.2实验记录与分析 44](#_Toc74052655)

[3.5 实验小结 53](#_Toc74052656)

[3.5.1主要收获 53](#_Toc74052657)

[4中断与反跟踪 53](#_Toc74052658)

[4.1实验目的与要求 53](#_Toc74052659)

[4.2实验内容 53](#_Toc74052660)

[4.3任务4.1实验过程 54](#_Toc74052661)

[4.3.1设计思想及存储单元分配 54](#_Toc74052662)

[4.3.2流程图 55](#_Toc74052663)

[4.3.3源程序 55](#_Toc74052664)

[4.3.4实验步骤 56](#_Toc74052665)

[4.3.4实验记录与分析 56](#_Toc74052666)

[4.4任务4.2实验过程 58](#_Toc74052667)

[4.4.1实验方法说明 58](#_Toc74052668)

[4.4.2实验记录与分析 59](#_Toc74052669)

[4.5任务4.3实验过程 61](#_Toc74052670)

[4.5.1实验方法说明 61](#_Toc74052671)

[4.5.2实验记录与分析 61](#_Toc74052672)

[4.6实验小结 64](#_Toc74052673)

[4.6.1主要收获 64](#_Toc74052674)

[5 16/32/64位编程比较 65](#_Toc74052675)

[5.1实验目的与要求 65](#_Toc74052676)

[5.2实验内容 65](#_Toc74052677)

[5.3任务5.1实验过程 66](#_Toc74052681)

[5.3.1实验方法说明 66](#_Toc74052682)

[5.3.2实验记录与分析 66](#_Toc74052683)

[5.4任务5.2实验过程 75](#_Toc74052684)

[5.4.1实验方法说明 75](#_Toc74052685)

[5.4.2 实验记录与分析 76](#_Toc74052686)

[5.5任务5.3实验过程 76](#_Toc74052687)

[5.5.1实验任务 76](#_Toc74052688)

[5.5.1实验结果 76](#_Toc74052689)

[5.6小结 77](#_Toc74052690)

[参考文献 78](#_Toc74052691)

# 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

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。

许向阳，《x86汇编语言程序设计》，华中科技大学出版社，2020年，第6章、第7章

## 任务1.1实验过程

### 实验方法说明

1. 从群文件中下载VS2019汇编语言上机操作，学习如何使用VS2019进行反汇编，查看寄存器和内存等。

### 实验记录与分析

1. 显示反汇编窗口

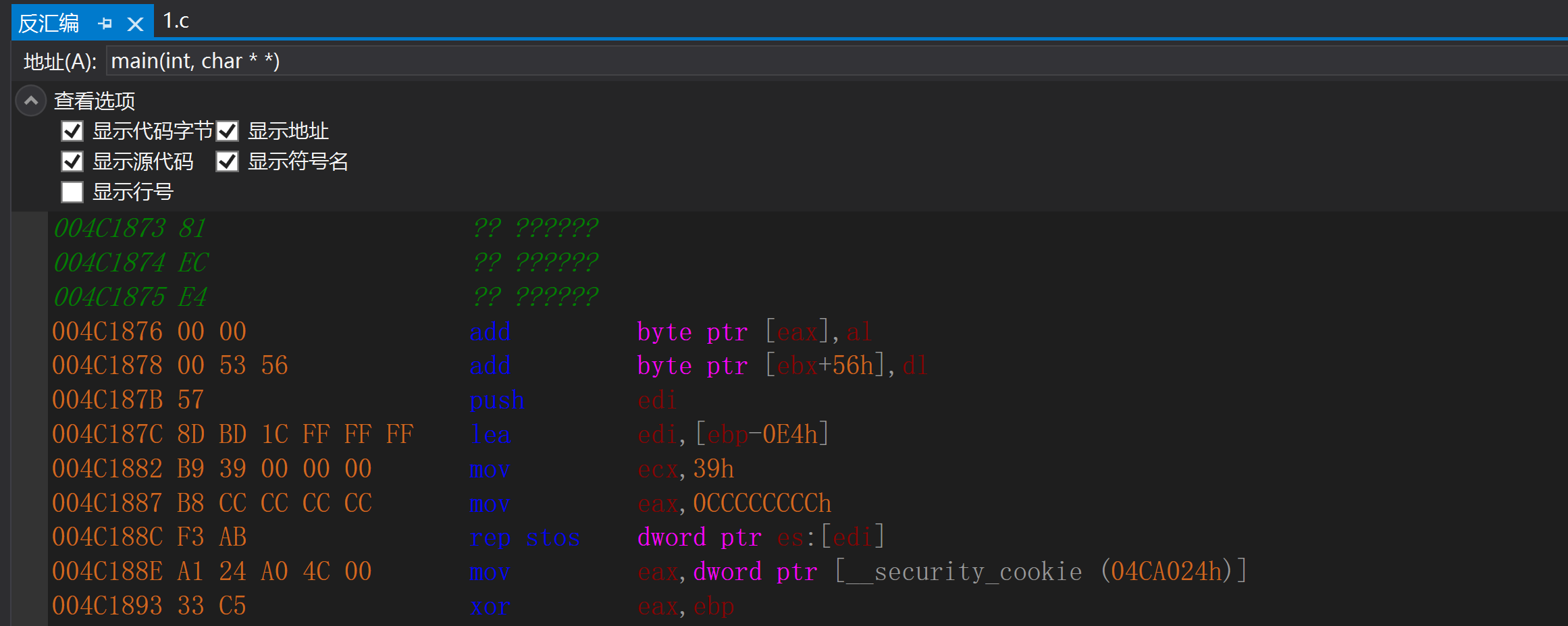


图1.1

经过观察发现，汇编语句涉及直接对内存和寄存器进行操作，一条C语言语句可能对应很多的汇编语句

勾选不显示符号名



图1.2

勾选显示符号名

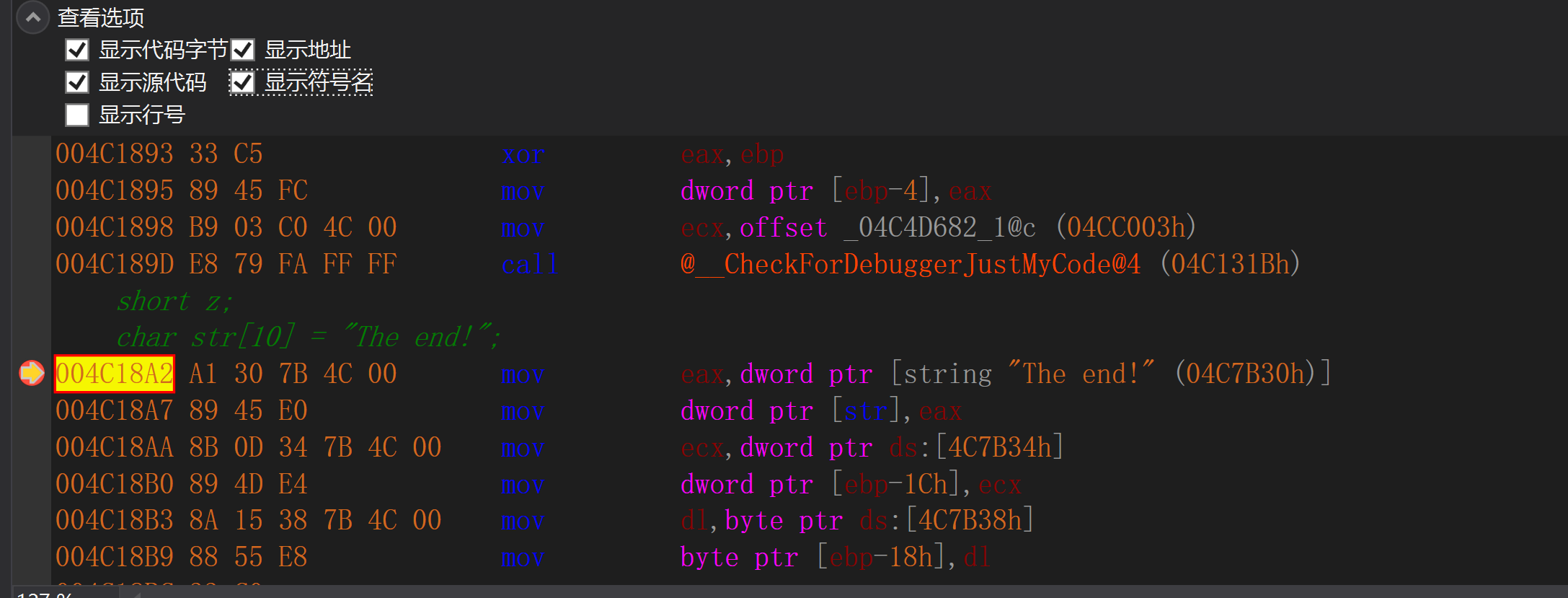


图1.3

勾选与不勾选显示符号名，显示符号名即显示出变量名，变量名是一个地址的符号表示，全局变量对应的是一个段及段内的偏移，若勾选了不显示符号名，则只显示偏移地址。局部变量对应的是一个堆栈中的存储单元。

1. 显示寄存器窗口

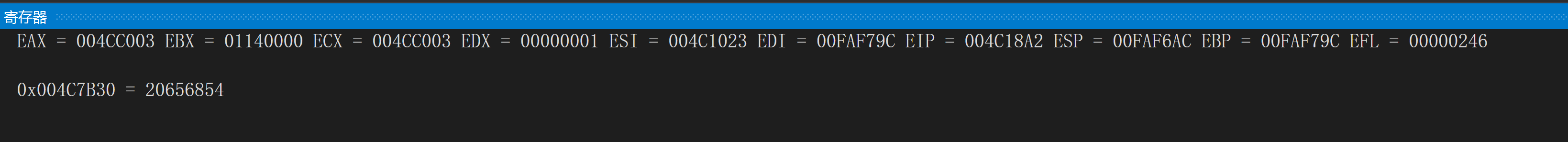


图1.4

在寄存器窗口鼠标右键，可以设置显示 寄存器、段寄存器、标志寄存器等；

其中每一个寄存器都有一定的标志

寄存器可以用于临时存储各种需要处理的数据

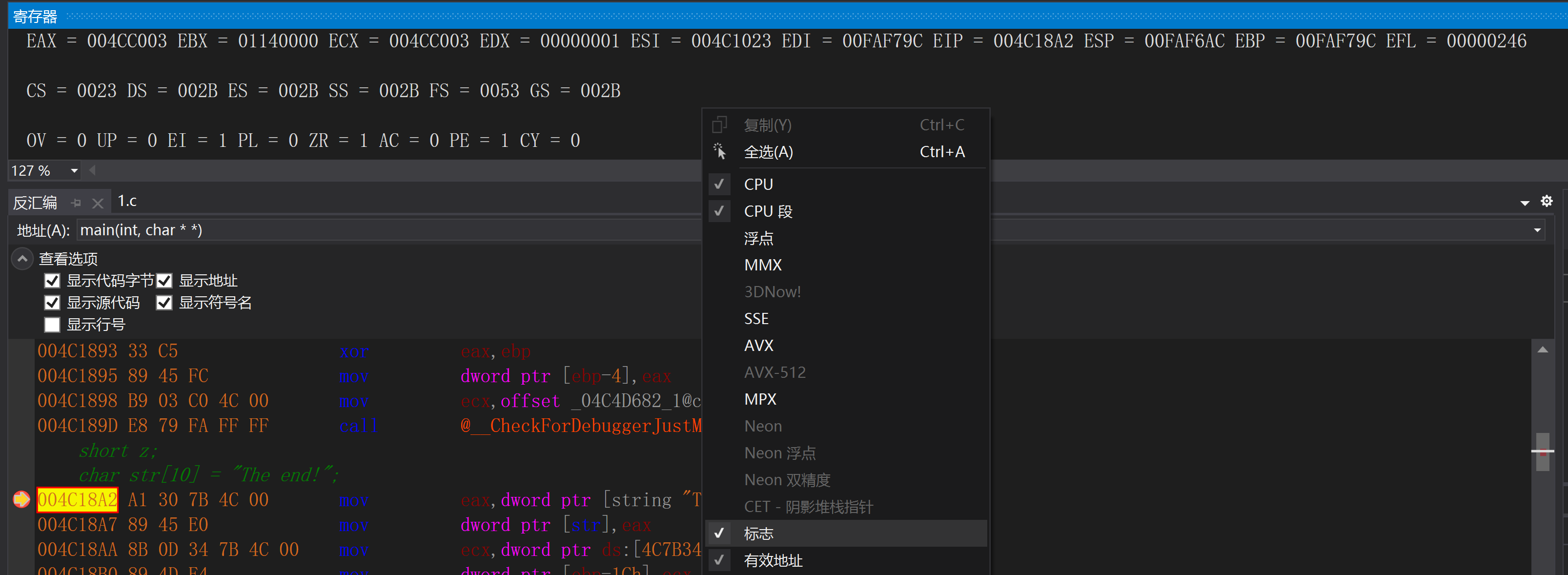


图1.5

1. 通过监视窗口，可以查看变量的值，也可以通过&变量，查看变量的地址

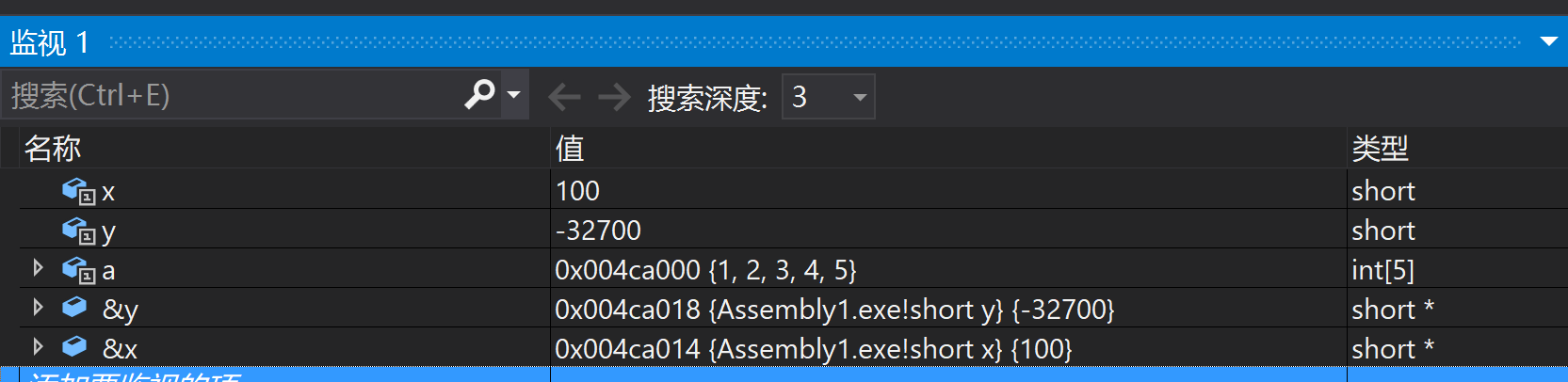


图1.6

通过监视窗口，我们可以取地址符&查看变量的在内存中的位置

如全局变量x，通过地址找到其在内存中的位置0x004ca014，由于x为short型，占两个字节，因此64 00为其值，由于Intel为小端存储，因此其值为0064，转化为二进制为0000 0000 0110 0100，最高位为0，因此为正值，转化为十进制就为100.

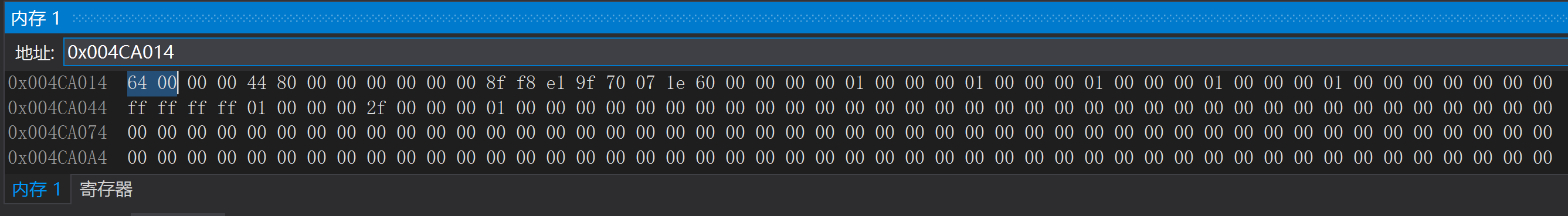


图1.7

同样我们找到y的地址0x004ca018，在内存中找到它，它在内存中为44 80 ,即8044转化为二进制为1000 0000 0100 0100，最高位为1，因此为负数，补码的值就为-32700

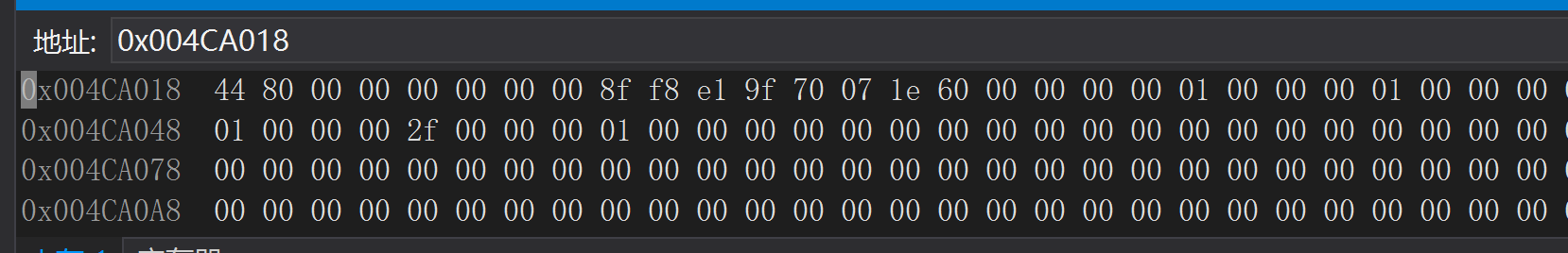


图1.8

当str被赋值之后，查找其在内存中的位置，可以看到str在内存中的存储是The end!的每个字符的ASCII码。

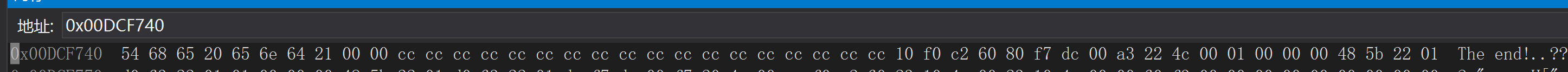


图1.9

1. 有符号数和无符号数如何存储
2. 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？

为探究以上问题，设置三个变量，为其赋值，查看其在内存中是如何存储的

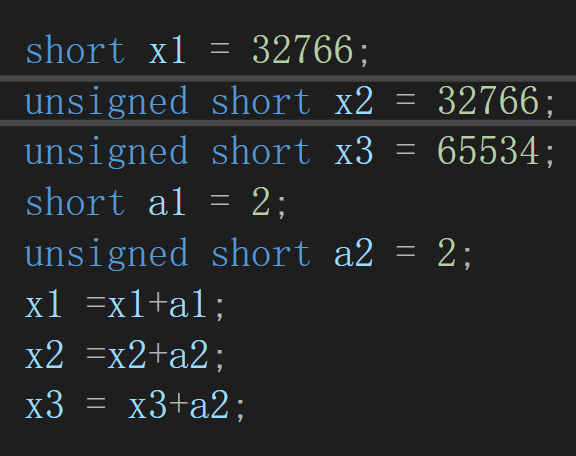


图1.10

执行完赋值语句后：

x1  0111 1111 1111 1110

x2 0111 1111 1111 1110

x3 1111 1111 1111 1110  
执行加2的语句后：

x1+2后 1000 0000 0000 0000

此时的标志寄存器如下



x2+2后1000 0000 0000 0000

此时的标志寄存器如下



x3+2后

此时的标志寄存器如下



通过比较x1，x2可以得到，有符号数和无符号数在内存中的存储方式是一样的，当最高位都为0时，两个变量的值相同，当最高位为1时，两个变量的值不同。



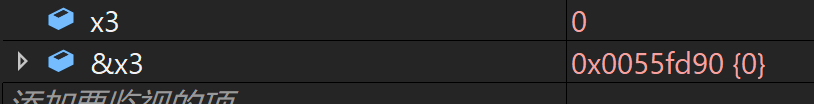


图1.11

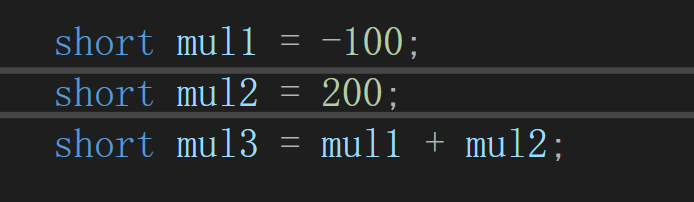


图1.12

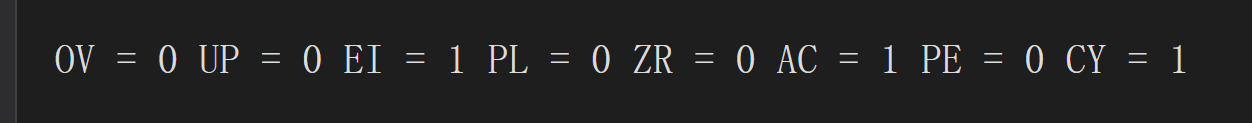
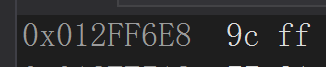


图1.13

mul1  1111 1111 1001 1100

mul2  0000 0000 1100 1000

mul3  (1) 0000 0000 0110 0100

观察执行完第三条语句后的寄存器，

CY=1，发生进位，上图可以见到结果的最高位进1

OV=0，未发生溢出，因为此操作为一正一负两个变量，不会发生溢出。

因此我们可以得到结论，无论是有符号数还是无符号数，其在内存中的存储形式都是一致的，但是编译器对于它们的解释不同，有符号数的最高位为符号位，而无符号数的最高位就是数值位。

调试时发现VS2019的标志位与书上的有一定的不同。

在此稍稍记录一下

OV 溢出标志

UP 方向标志

EI 中断标志

PL 符号标志

ZR 零标志

AC 辅助标志

PE 奇偶标志

CY 进位标志

## 任务1.2的实验过程 *（这里展示“设计型”任务的写法）*

### 实验方法说明

1. 掌握VS2019汇编的使用方法，在使用的基础上掌握调试，运行等方法。
2. 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。
3. 在DOSBox下使用“MASM 6.0，LINK.EXE，TD.EXE, 完整段定义”改造16位汇编语言程序，体会16位段程序的编译、链接和调试过程。

### 实验记录与分析

使用VS2019

1. 使用VS2019创建asm文件，编译链接调试，得到控制台输出



图1.14

同任务1.1，对1.2的程序进行反汇编窗口显示、寄存器窗口、监视窗口、内存窗口的操作。

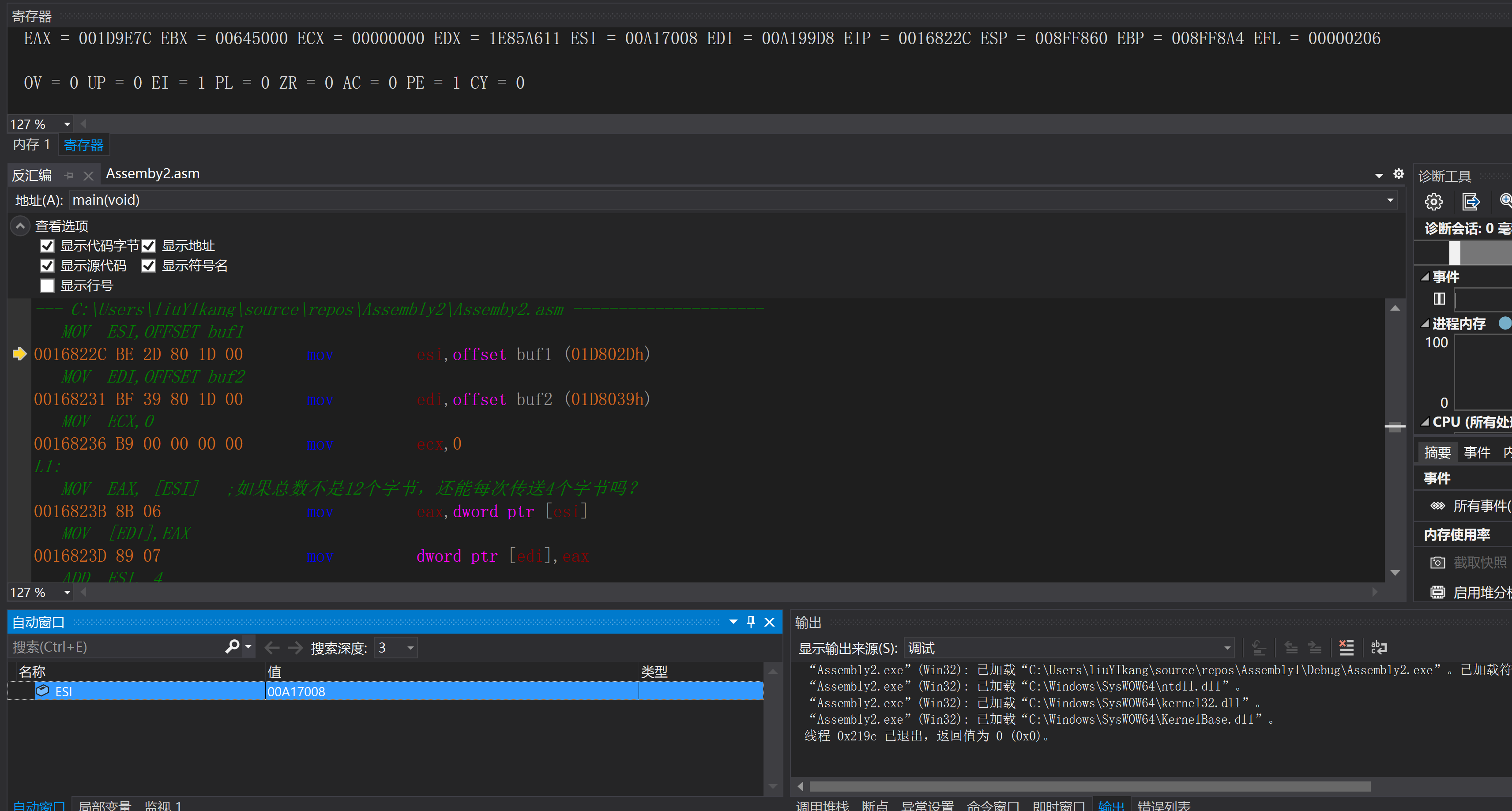


图1.15

显示符号名



图1.16

不显示符号名

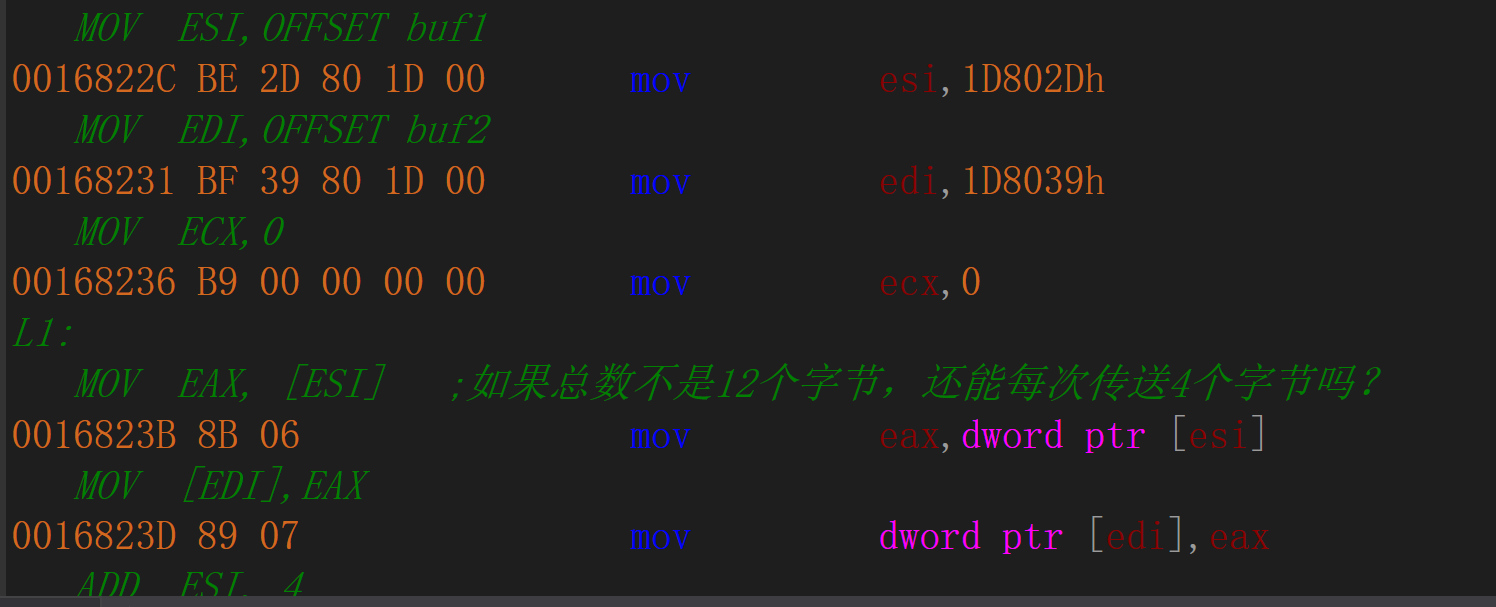
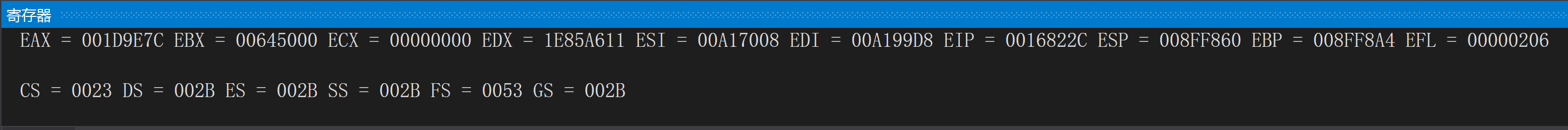


图1.17

寄存器窗口



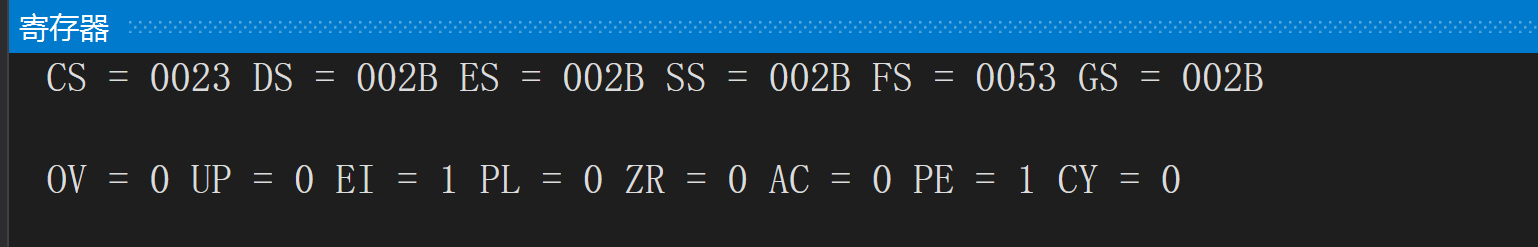


图1.18

内存窗口

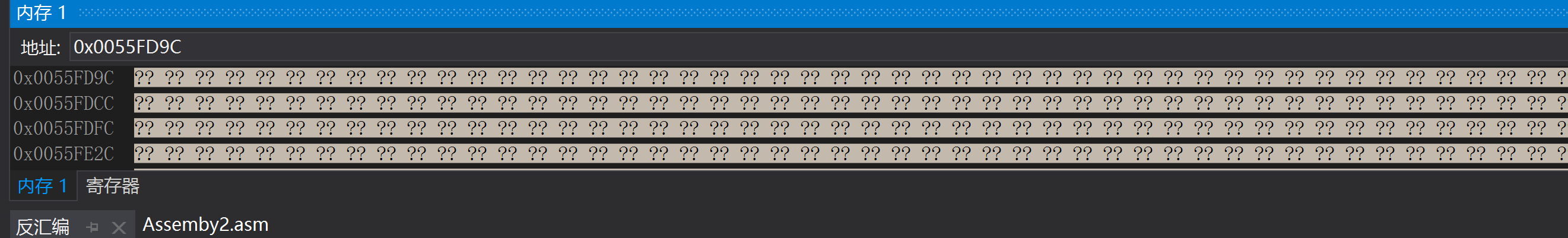
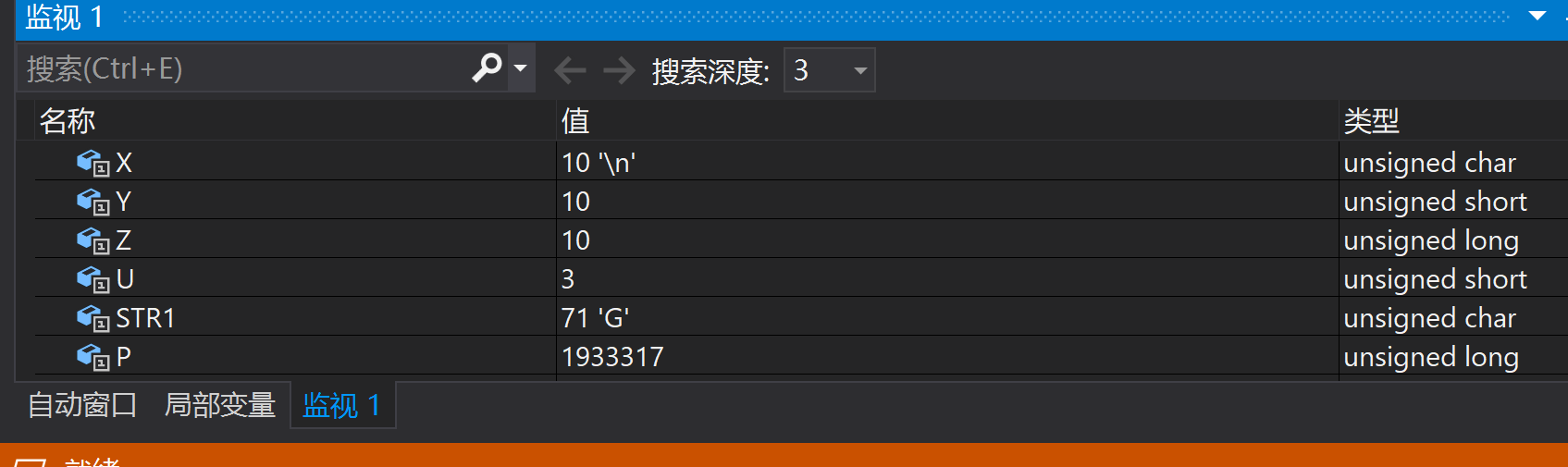


图1.19

监视窗口



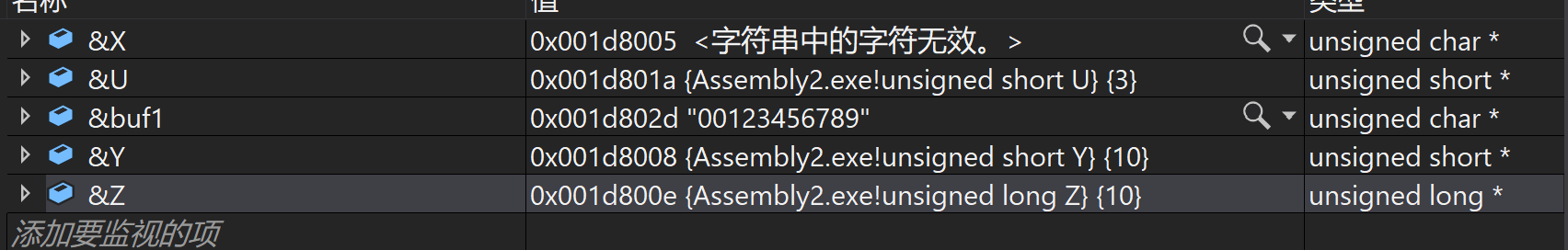
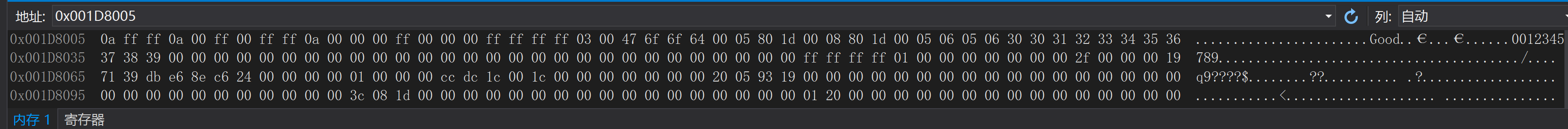


图1.20

通过监视窗口找到变量内存



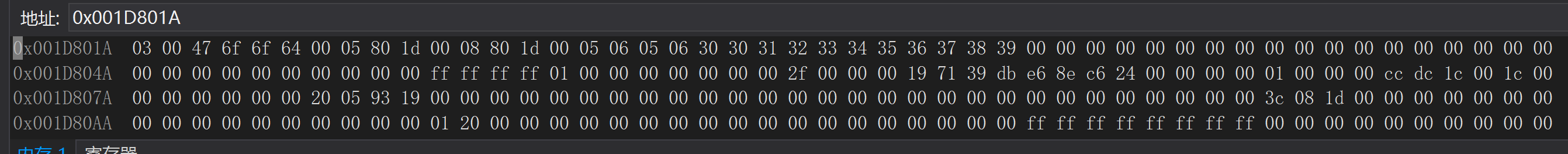


图1.21

1. 观察数据段的存储结果

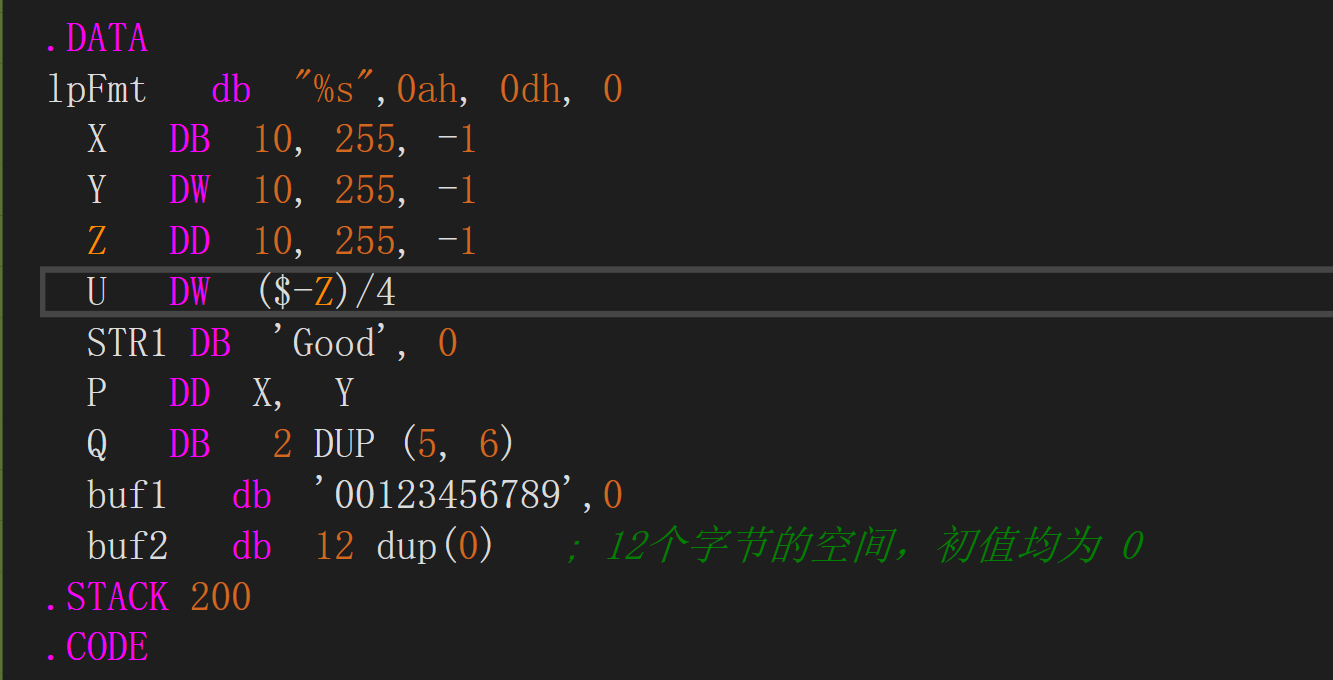


图1.22

找到数据段中第一个变量X在内存中的位置，并观察内存。

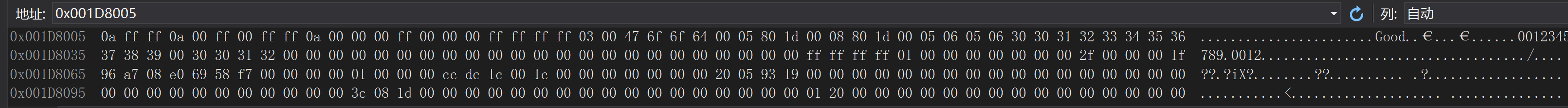


图1.23

X DB 10, 255, -1 X为单字节，后三个字节分别为0a，ff，ff ，分别为10，255，-1

Y DW 10, 255, -1 Y为双字节，Y后六个字节分别为00 0a，00 ff，ff ff，分别为10，255，-1

Z DD 10, 255, -1 Z为四字节，Z后十二个字节分别为00 00 00 0a，00 00 00 ff，ff ff ff ff

观察它们的地址，X为0x001d8005 Y为0x001d8008 Z为0x001d800e,

X与Y间隔3字节，Y与Z间隔6字节，U与Z间隔12字节。

以上观察得到，数据段中变量的存储位置是紧挨着的，相邻变量的地址也是相邻

当查看P的内存位置时，发现P存储的是X与Y的地址，验证了汇编语言中变量名为地址。

1. 观察堆栈段

堆栈地址的访问可以从寄存器中找到ESP，EBP，

ESP为扩展栈指针寄存器，用于指向栈的栈顶。EBP为扩展基址指针寄存器，用于指向栈底

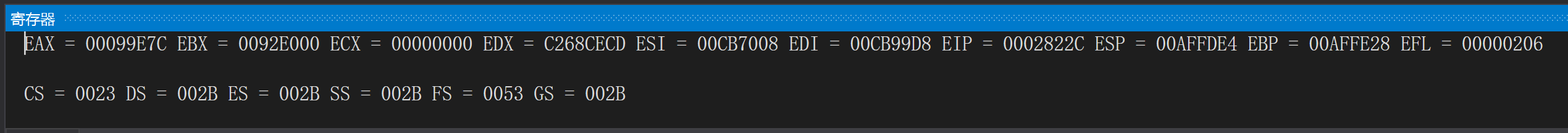


图1.24

栈基址

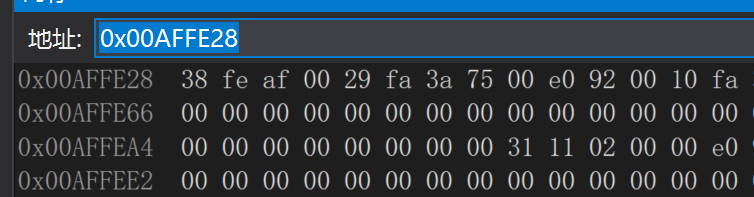


图1.25

栈顶

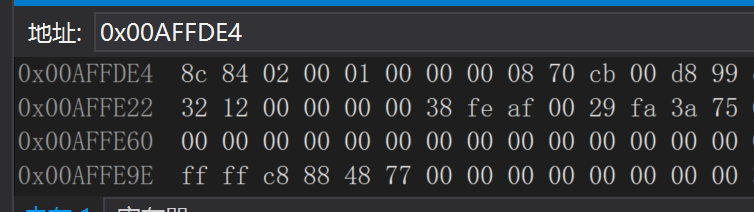


图1.26

1. 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。

修改L1段代码如下

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

MOV [EDI+ECX],EAX

ADD ECX, 4

CMP ECX,12

JNZ L1

采用变址寻址

**DOSBox**

1. 配置DOSBox

在E盘目录下建一文件夹，里面再存放如下两个文件夹



图1.27

Assembly

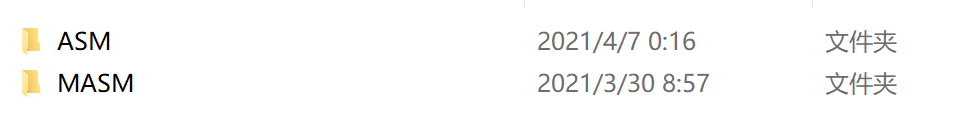


图1.28

ASM

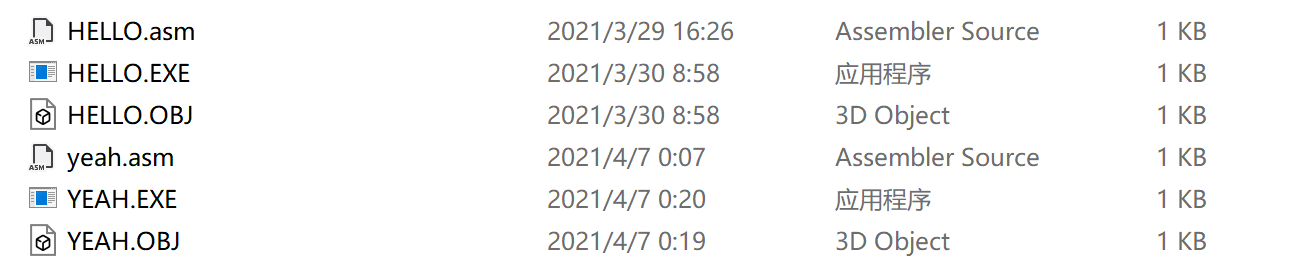


图1.29

MASM

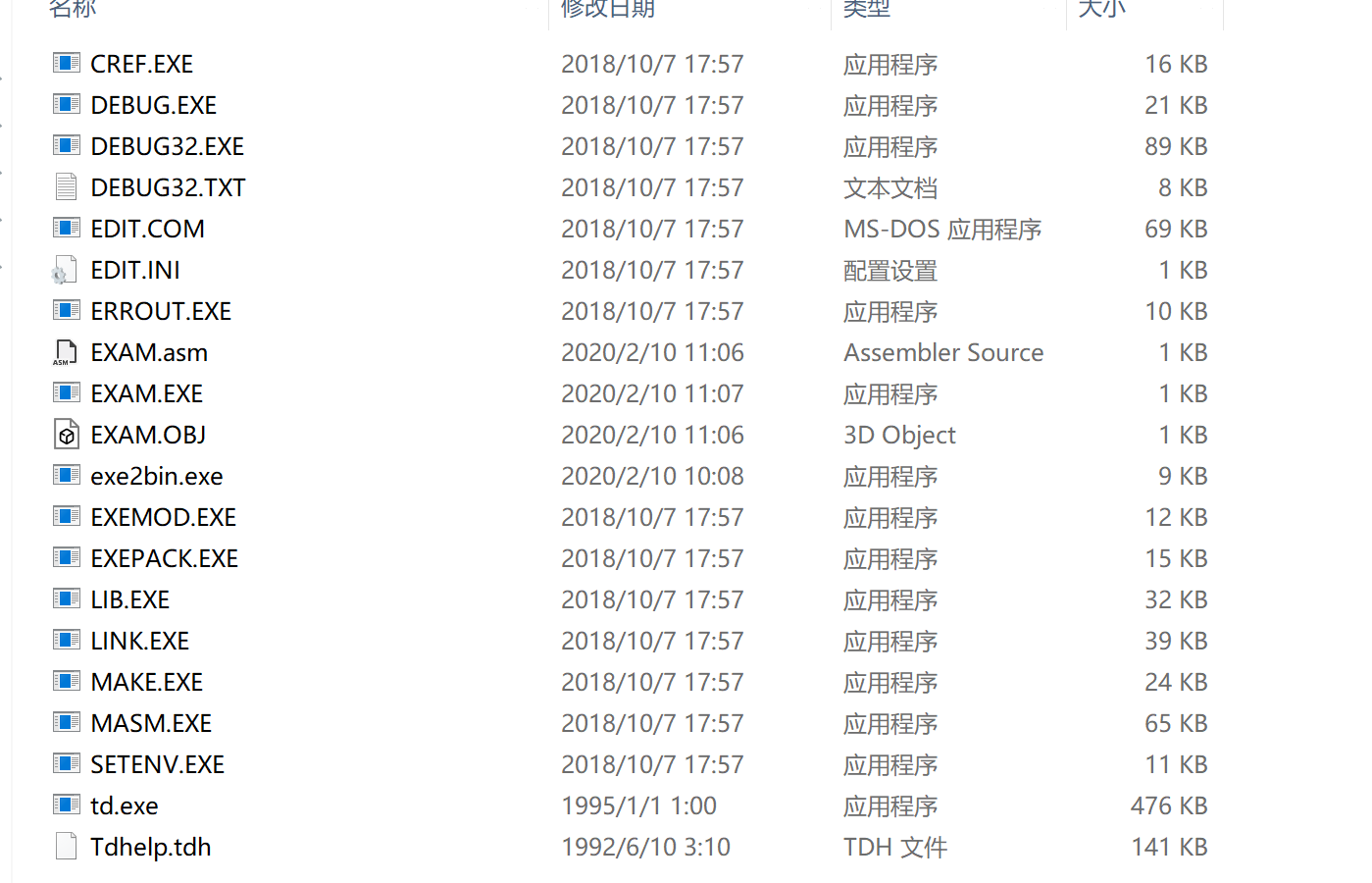


图1.30

（2） 新建yeah.asm文件，存放代码，打开DOSBox，输入如下指令

masm yeah.asm

生成yeah.obj文件

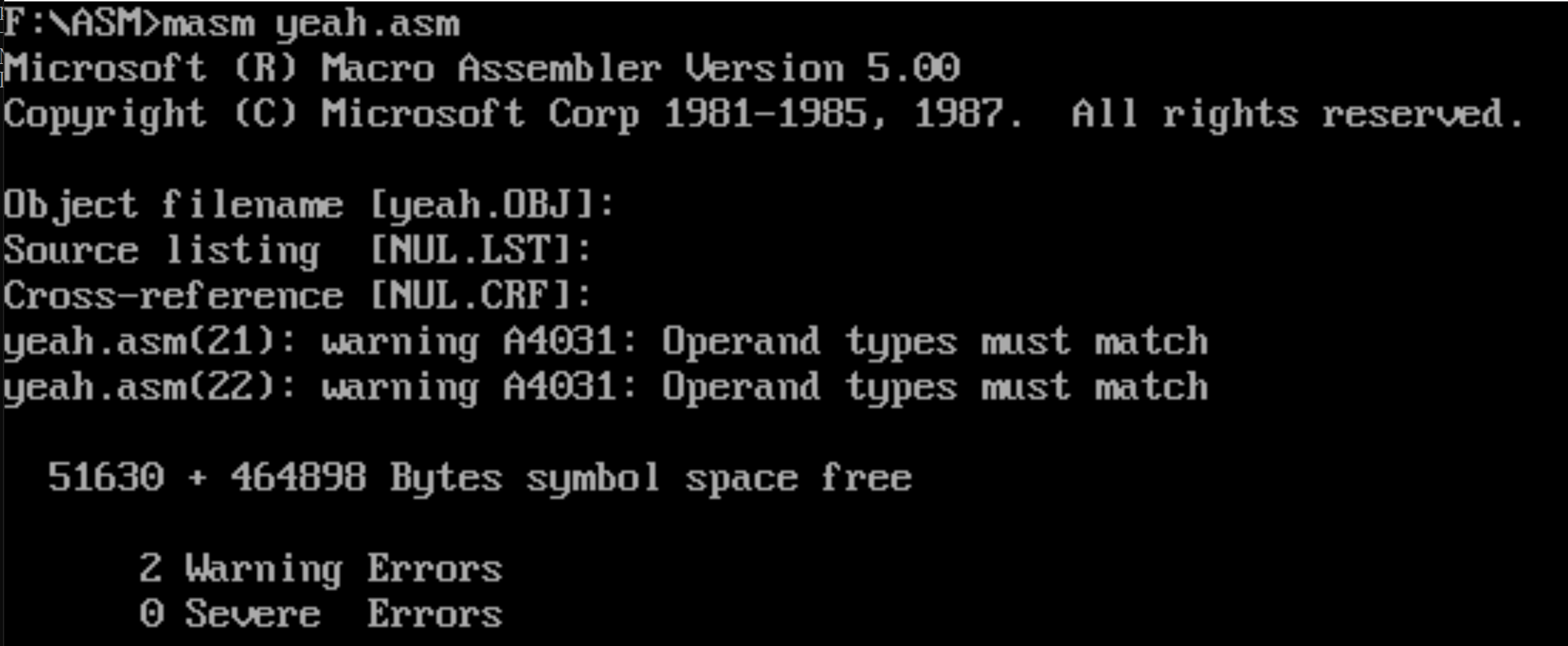


图1.31

link yeah.obj

生成yeah.exe文件

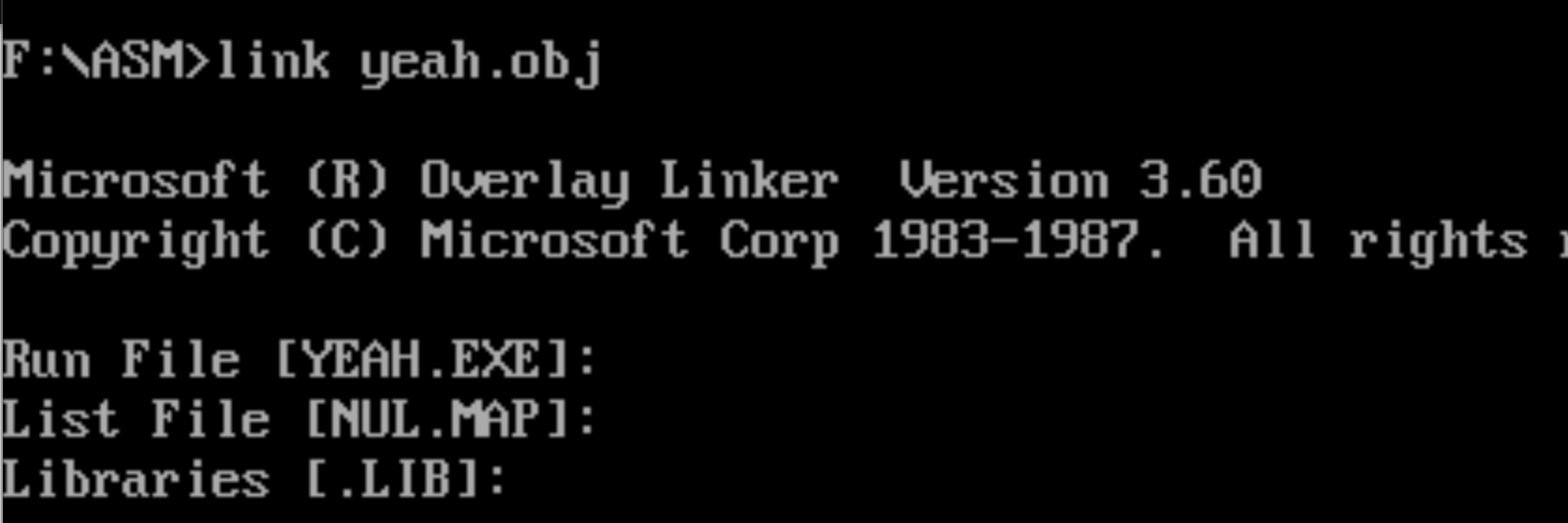


图1.32

debug yeah.exe g 运行yeah.exe文件

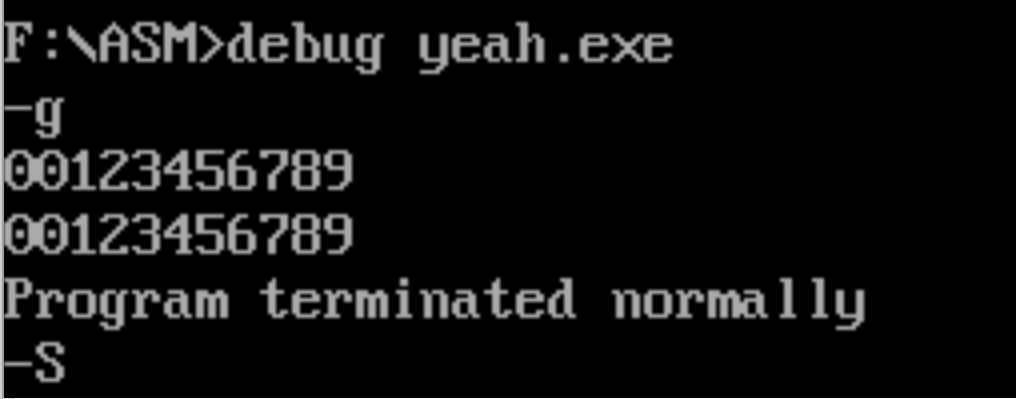


图1.33

（3） 输入td，打开TD

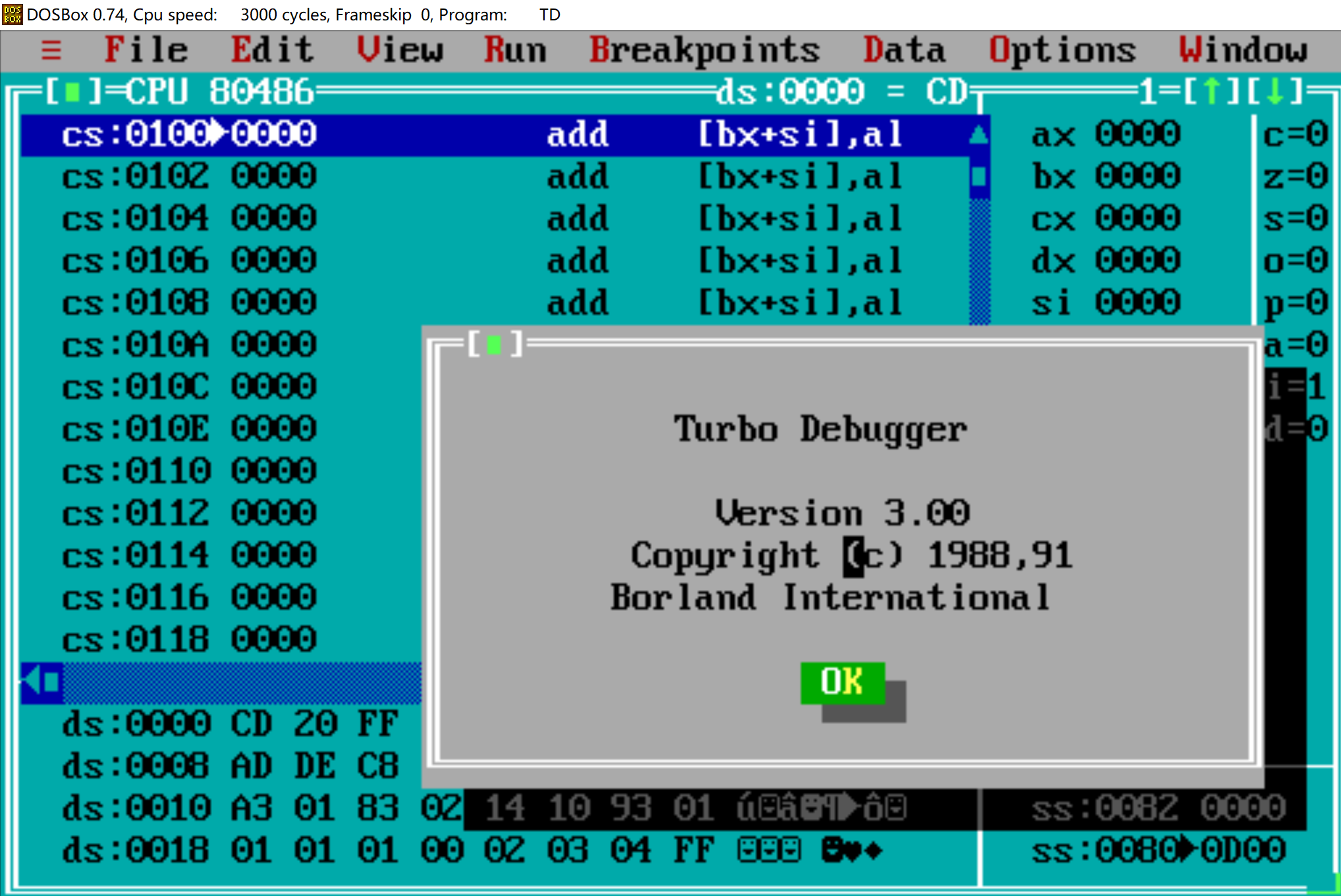


图1.34

File->open 选择之前生成的yeah.exe,可以直接打开exe文件

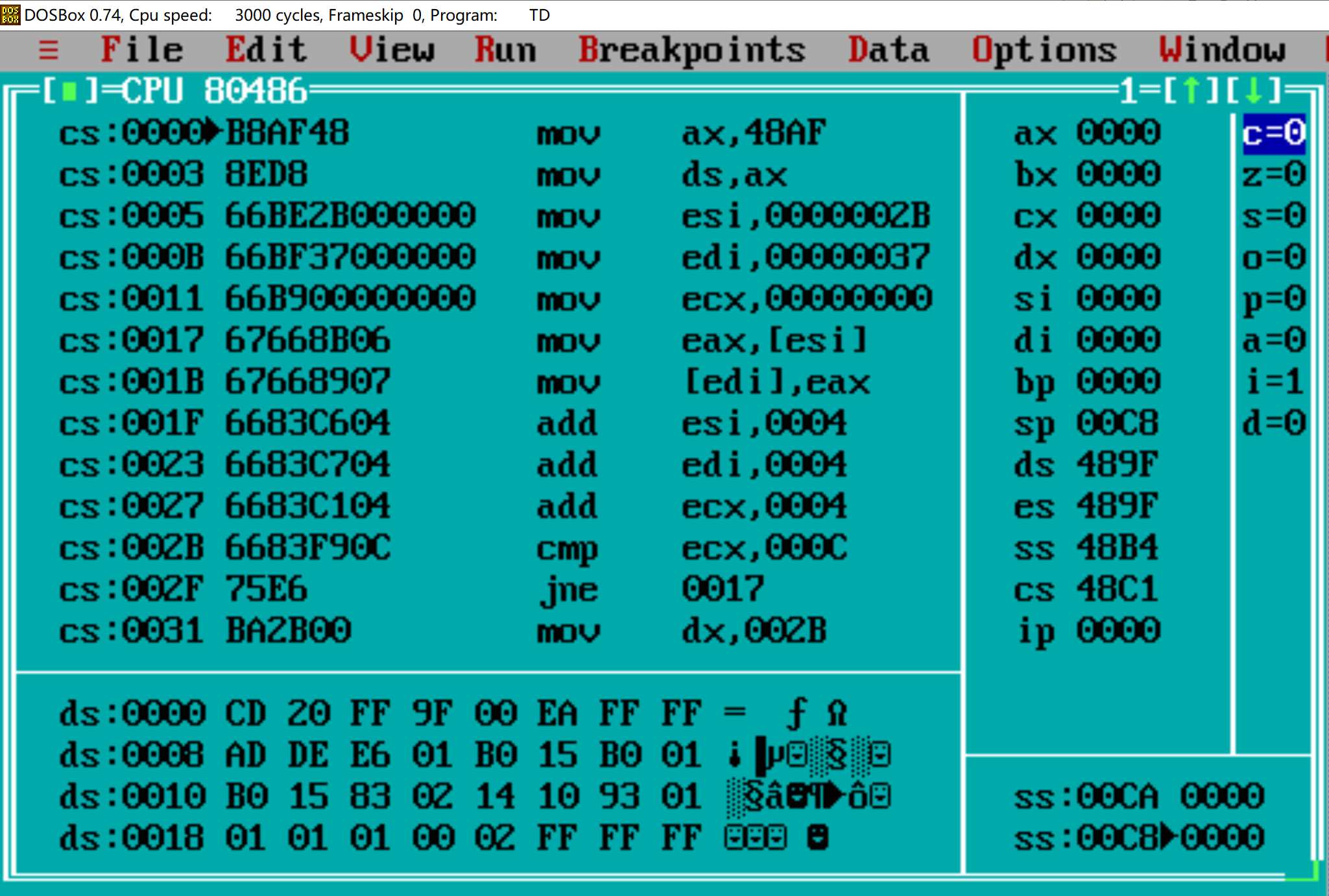


图1.35

如图，可以分别看到代码段cs，数据段ds，堆栈段ss以及寄存器

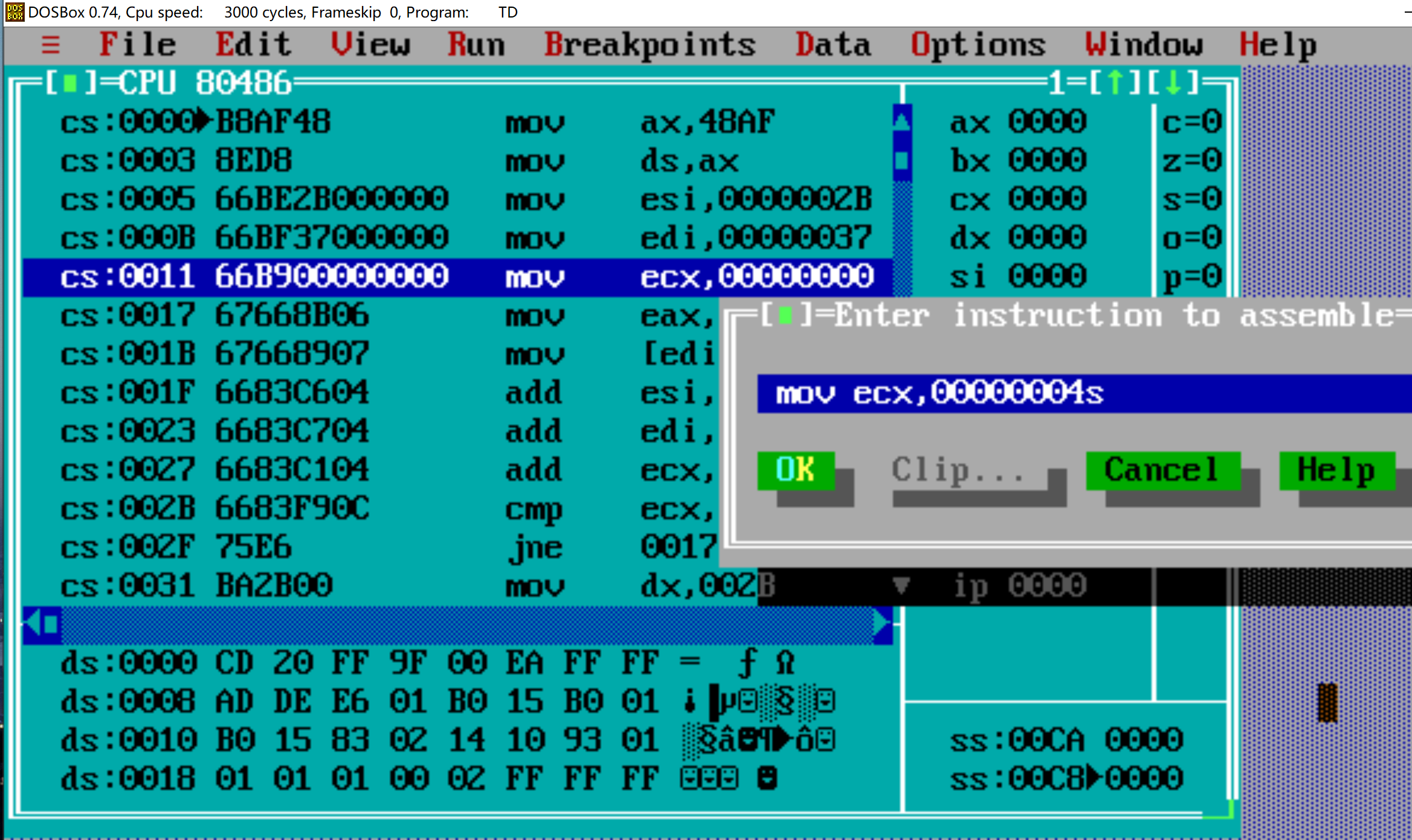


图1.36

如图可以直接编辑代码，若将上图中的ecx改为4，程序就只循环2次而不是3次

## 任务1.3的实验过程 *（这里展示“设计型”任务的写法）*

### 设计思想及存储单元分配 *（描述算法、程序框架、功能模块、接口等）*

1. 根据要求，程序运行时应该先进行身份验证，既输入姓名和学号，验证正确后才能进入系统。

数据段定义

TRUENAME储存用户正确的名字，BPASS储存正确密码。INNAME储存输入的用户名，INPWD储存输入的密码，OUTNAMEW，INPUTPWD，OUTPWDW，INPUTNAME，SUCCESSIN等分别存储提示语句

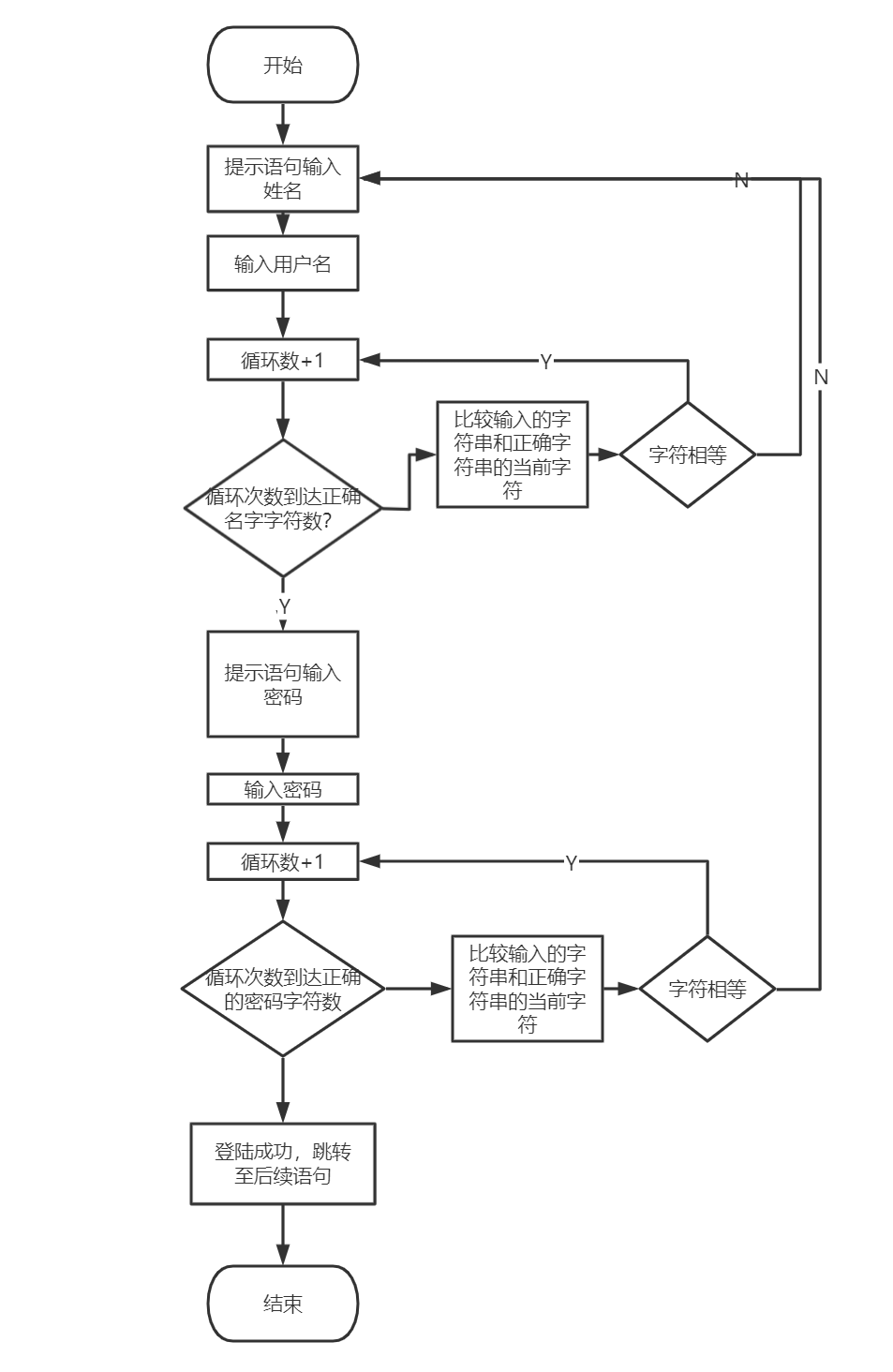


图1.37 登录流程图

1. 菜单

输出提示语句，根据提示语句用户输入要选择的功能，通过数据段中的CHOICE存储，若是功能4和9，则直接跳转至相应的位置FUNC1和OVER，若是功能123，则先不跳转，先进行商品的查找，因为功能123都要先进行商品的查找，这样可以简化代码

1. 查找商品功能

INGOOD储存输入的商品名。INGOODT，WRONGGOODT， CHANGEGOODT，GOOD。

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，WRONG\_GOOD是输入商品名不存在时运行的语句块。

先输入商品名，用变址寻址的方法，GA1+20\*BX计算4个要比较的商品的偏移地址，并将偏移地址置入ESI，将输入字符串第一个字符的偏移地址置入EDI，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1-6-2：

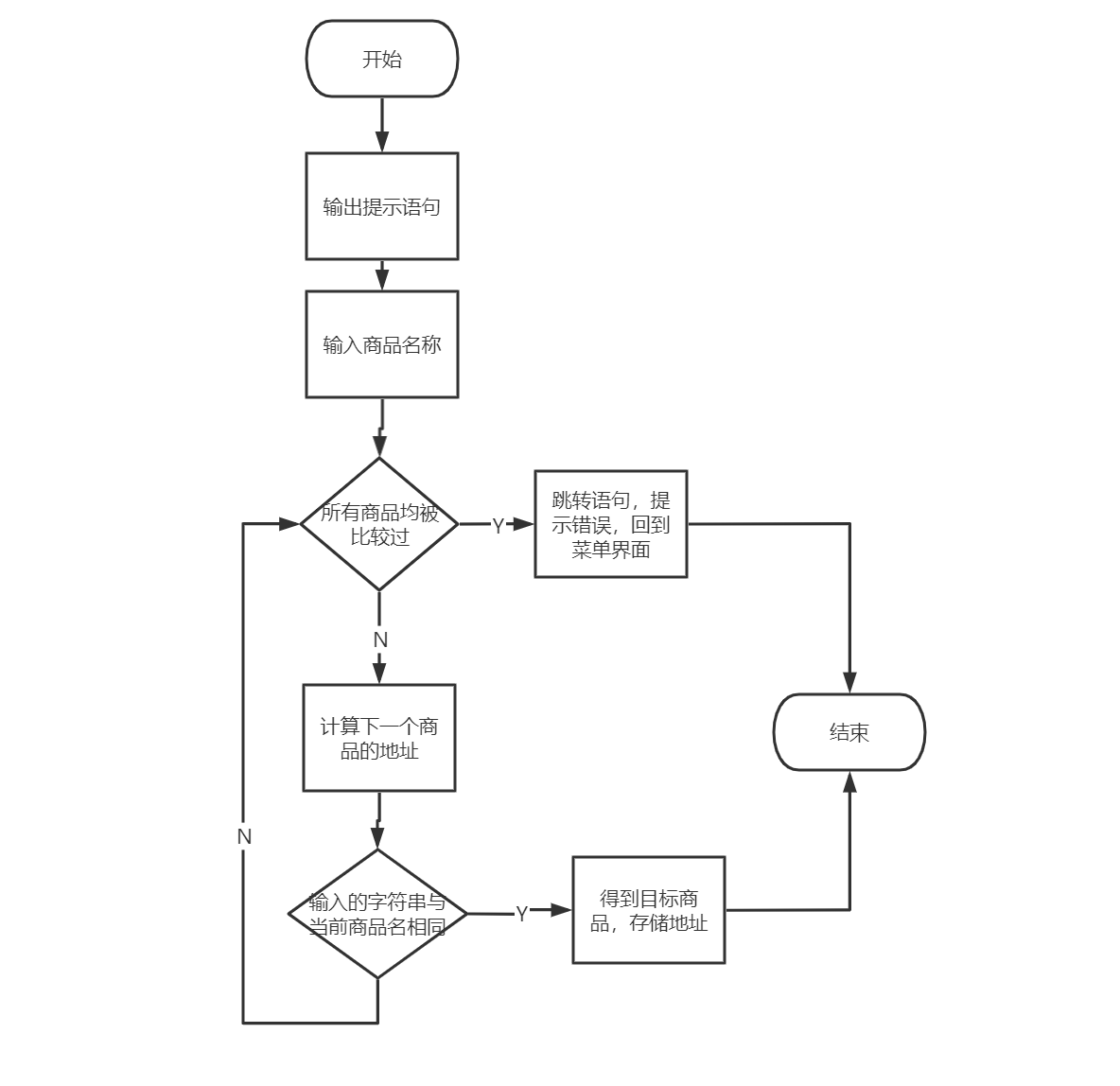
****

图1.38 查找商品功能流程图

1. 出货

输出提示语句，输入出货数量，根据得到的商品地址，计算进货量和销售量之差，再与出货数量比较，若出货数量大于剩余货物数量，则跳转提示错误的代码，回到菜单，若出货数量小于剩余货物的数量，则销售量加上出货数

（5） 进货

输出提示语句，输入进货数量，根据得到的商品地址，将进货量加上这次的进货数量

（6） 计算利润率

利用GA1+20\*EDI，分别计算四个商品的偏移地址，根据每个商品的偏移地址，取出进货价，售价，进货量，销售量，根据公式计算出利润率并保存到内存中

### 源程序

.386

.model flat, stdcall

ExitProcess PROTO STDCALL :DWORD

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

printf PROTO C : dword,:vararg;对printf函数声明.

scanf PROTO C : dword,:vararg;对scanf函数声明.

includelib libcmt.lib

includelib legacy\_stdio\_definitions.lib

.DATA

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

lpFmt1 db "%s", 0

lpFmt2 db "%d", 0

lpFmt3 db "%d",0ah,0dh, 0

INPUTNAME DB 'User Name:',0

OUTNAMEW DB 'Invalid User Name',0

INNAME DB 20 DUP(0)

BNAME DB 'LiuYiKang', 0 ;老板姓名

INPUTPWD DB 'Password:',0

OUTPWDW DB 'Invalid Password',0

INPWD DB 20 DUP(0)

BPASS DB 'U201914873', 0 ;密码

MENUT DB 'Please Enter a Number(1~9):',0

CHOICE DB 0

Shipment DB 0

Shipment\_buf DB 'Please Enter Shipment quantity:',0

Replenishment DB 0

Replenishment\_buf DB 'Please Enter Replenishment quantity:',0

SUCCESSIN DB 'Login in',0

INGOODT DB 'Good Name:',0

WRONGGOODT DB 'Invalid Good Name',0

INGOOD DB 10 DUP(0)

NOGOODT DB 'No Goods',0

Purchase\_price DB 'Purchase price:',0

Sales\_price DB 'Sales price:',0

Purchase\_quantity DB 'Purchase quantity:',0

Sold\_quantity DB 'Sold quantity:',0

SOLDGOODS DB 'The goods have been sold:',0

NENOUGH DB 'There is not enough left:',0

PURCHASEDGOODS DB 'The goods have been purchase:',0

UPDATEPRIOT DB 'Priority Updated',0

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 DB 'RULER',5 DUP(0) ;商品4 名称

DW 3,4,200,150,?

GAN DB N-4 DUP( 'Temp',6 DUP(0) ,0,15,0,20,0,30,0,2,?,?)

.STACK 200

.CODE

main proc c

Start:

;输入用户名

invoke printf,offset lpFmt,OFFSET INPUTNAME

invoke scanf,offset lpFmt1,OFFSET INNAME

MOV CL,9

LEA EDI, INNAME

LEA ESI, BNAME

CHECK\_USER: ;逐字节比较用户名

MOV AL, [EDI]

CMP AL, [ESI]

JNE WRONG\_NAME

INC EDI

INC ESI

DEC CL

JNZ CHECK\_USER

;输入密码

invoke printf,offset lpFmt,OFFSET INPUTPWD

invoke scanf,offset lpFmt1,OFFSET INPWD

MOV CL,10

LEA EDI, INPWD

LEA ESI, BPASS

CHECK\_PWD: ;逐字节比较用户名

MOV AL, [EDI]

CMP AL, [ESI]

JNE WRONG\_PWD

INC EDI

INC ESI

DEC CL

JNZ CHECK\_PWD

JMP LOGIN\_S

WRONG\_NAME:

invoke printf,offset lpFmt,OFFSET OUTNAMEW

JMP Start

WRONG\_PWD:

invoke printf,offset lpFmt,OFFSET OUTPWDW

JMP Start

LOGIN\_S: ;登陆成功

invoke printf,offset lpFmt,OFFSET SUCCESSIN

MENU:

invoke printf,offset lpFmt,OFFSET MENUT

invoke scanf,offset lpFmt1,OFFSET CHOICE

LEA EDI, CHOICE

MOV AL, [EDI]

CMP AL, '4'

JE FUNC4

CMP AL, '9'

JE OVER

invoke printf,offset lpFmt,OFFSET INGOODT

invoke scanf,offset lpFmt1,OFFSET INGOOD

MOV EAX, -1

CHECK\_GOOD: ;选择下一个比较商品

INC EAX ;查看商品是否查找完

CMP EAX, 4

JE NO\_GOOD

MOV ECX, EAX

IMUL ECX, 20

ADD ECX, OFFSET GA1 ;ECX此时存放当前商品的首地址

push ECX

MOV BL,10

MOV EDI, ECX

LEA ESI, INGOOD

CHECK\_NAME:

MOV AL, [EDI]

CMP AL, [ESI]

JNE CHECK\_GOOD

INC EDI

INC ESI

DEC BL

JNZ CHECK\_NAME

jmp NEEDGOODS

NO\_GOOD:

invoke printf,offset lpFmt,OFFSET NOGOODT

JMP MENU

NEEDGOODS:

LEA EDI, CHOICE

MOV AL, [EDI]

CMP AL, '1'

JE FUNC1

CMP AL, '2'

JE FUNC2

CMP AL, '3'

JE FUNC3

JMP MENU

;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。

FUNC1:

;显示商品信息

pop ECX

MOV EDI, ECX

invoke printf,offset lpFmt,EDI

ADD EDI,10

invoke printf,offset lpFmt1,OFFSET Purchase\_price

MOV AX,[EDI]

invoke printf,offset lpFmt3,AX

ADD EDI,2

invoke printf,offset lpFmt1,OFFSET Sales\_price

MOV AX,[EDI]

invoke printf,offset lpFmt3,AX

ADD EDI,2

invoke printf,offset lpFmt1,OFFSET Purchase\_quantity

MOV AX,[EDI]

invoke printf,offset lpFmt3,AX

ADD EDI,2

invoke printf,offset lpFmt1,OFFSET Sold\_quantity

MOV AX,[EDI]

invoke printf,offset lpFmt3,AX

JMP MENU

FUNC2:

invoke printf,offset lpFmt,OFFSET Shipment\_buf

invoke scanf,offset lpFmt2,OFFSET Shipment

POP ECX

MOV EDI, ECX

ADD EDI,14

MOV AL,[EDI]

add EDI,2

MOV BL,[EDI]

MOV DL,Shipment

SUB AL,BL

CMP AL,DL

JB NOTENOUGH

ADD [EDI],DL

invoke printf,offset lpFmt1,OFFSET SOLDGOODS

MOV AX,[EDI]

invoke printf,offset lpFmt3,AX

invoke printf,offset lpFmt,OFFSET UPDATEPRIOT

JMP MENU

NOTENOUGH:

invoke printf,offset lpFmt,OFFSET NENOUGH

JMP MENU

FUNC3:

invoke printf,offset lpFmt,OFFSET Replenishment\_buf

invoke scanf,offset lpFmt2,OFFSET Replenishment

POP ECX

MOV EDI, ECX

ADD EDI,14

MOV DL,Replenishment

ADD [EDI],DL

invoke printf,offset lpFmt1,OFFSET PURCHASEDGOODS

MOV AX,[EDI]

invoke printf,offset lpFmt3,AX

invoke printf,offset lpFmt,OFFSET UPDATEPRIOT

JMP MENU

FUNC4:

MOV ESI, -1

push ESI

ALLGOOD: ;选择下一个比较商品

pop ESI

INC ESI ;查看商品是否查找完

push ESI

CMP ESI, 4

JE MENU

IMUL ESI, 20

ADD ESI, OFFSET GA1 ;ECX此时存放当前商品的首地址

push ESI

invoke printf,offset lpFmt,ESI

POP ESI

ADD ESI,10

MOV AL,[ESI] ;进货价

ADD ESI,4

MOV BL,[ESI] ;进货数量

IMUL BL

PUSH AX ;进货

SUB ESI,2

MOV AL,[ESI] ;销售价

ADD ESI,4

MOV BL,[ESI] ;销售数量

IMUL BL

PUSH AX ;售货

MOV EAX,0

MOV EBX,0

pop AX

POP BX

SUB EAX,EBX

IMUL EAX,EAX,100

CDQ

idiv EBX

invoke printf,offset lpFmt3,EAX

jmp ALLGOOD

OVER:

invoke printf,offset lpFmt,ESI

main endp

END

### 实验步骤

1. 编写好程序代码，进入VS编译链接运行
2. 输入正确的用户名和密码，验证能否成功登陆

（3） 输入错误的用户名和密码，验证能否提示输入错

（4） 检验功能1正确情况

（6） 检验功能1错误情况

（7） 验证功能2正确情况

（8） 验证功能2错误情况

（9） 验证功能3

（10）验证功能4

### 实验记录与分析

验证登陆界面



图1.39

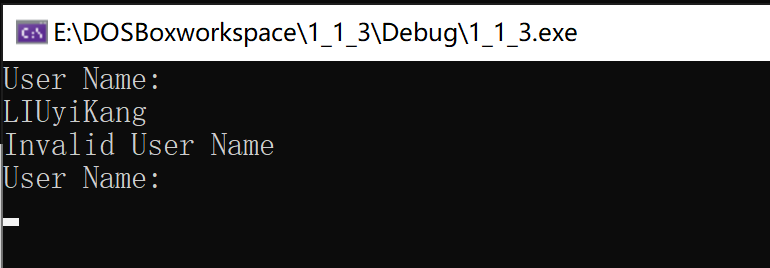


图1.40

验证功能1

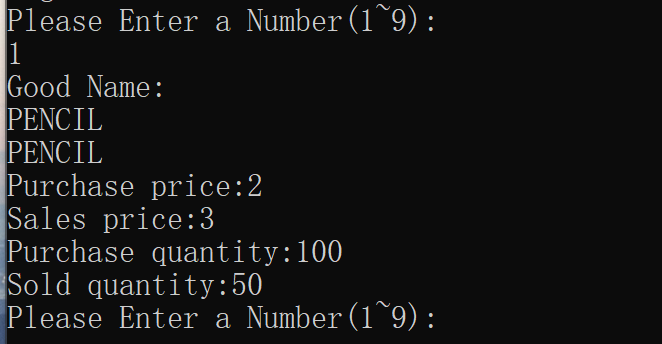


图1.41

验证功能2

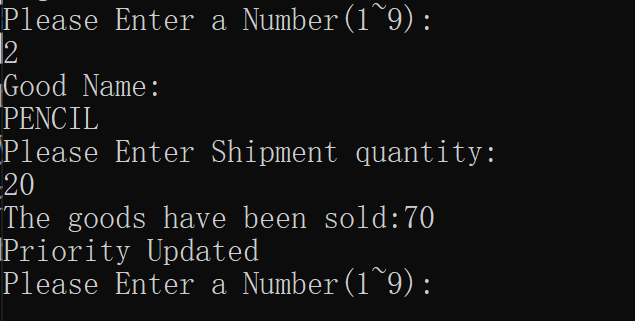


图1.42

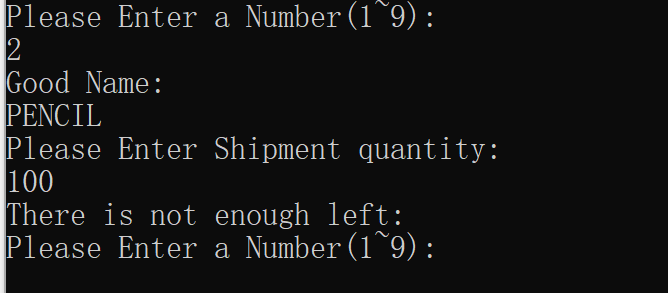


图1.43

验证功能3

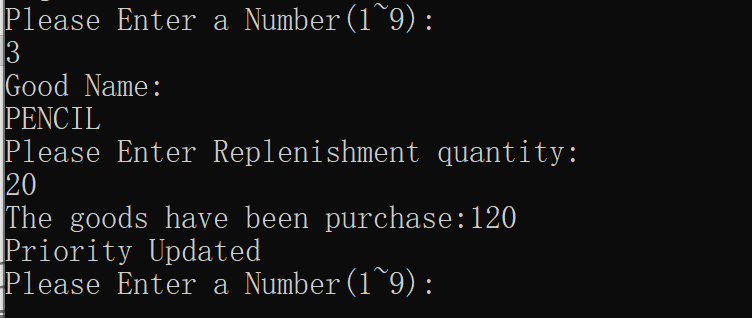


图1.44

验证功能4

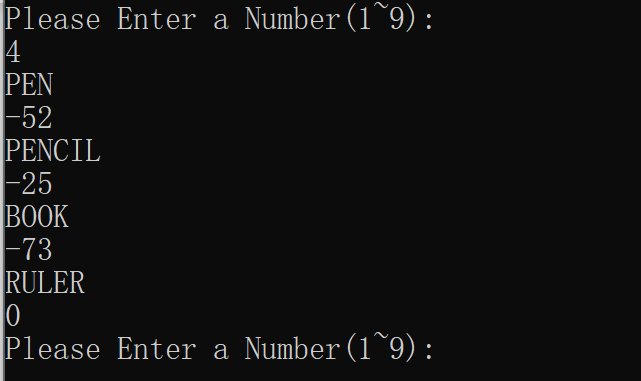


图1.45

## 小结

### 主要收获

通过任务1.1的实验，让我知道了TD不仅可以调入现成的执行程序进行调试，而且能随时输入和测试单一一条指令是否正确，执行效果如何，这将方便未来的学习过程。另外，通过观察，计算机内的加减运算，无论是否有符号数，对应的标志位都是设定好了的，如何使用这些标志，完全由程序员选择的指令来决定，这就要求我们编写程序时要理解好题目的语义，选择合适的指令语句，而不是我写了个负数交给汇编程序，系统就会自动选择有符号指令的。

任务1.2的实验表明，源程序定义的段的次序，与调到内存中段的次序是有对应关系的。但数据段开始没有被赋值到实际的数据段首址中，需要执行了我们的相关赋值语句后才变得实际有效。另外，各种寻址方式在内存中确实不如源程序直观了，但可以看到变量的偏移值都被很好的计算出来了，我们可以通过源程序中变量与内存中对应位置的信息获得该变量的偏移地址，方便我们在数据显示区输入该地址，查看对应的变量内容。

任务1.2中若输入0~9之外的字符，结果其实是没有意义的。要解决这个问题，可以增加对输入的值的范围进行判断的语句，超出范围时报错，并要求重新输入。

本次上机不仅提高了编程水平，熟悉了工具的使用，而且加深了对一些知识的理解。主要的经验教训如下：

首先，更加感受到实验前准备的意义。例如：上机前准备越充分（如先编好源程序，制定好准备做的一些步骤），上机的时候目的越明确，可以解决较多的问题。

其次，录入程序时要注意一些细节，比如中文分号、字母O等问题，虽然汇编程序指出其所在行有错，但很难发现具体是哪个符号错了，耽误了不少时间。TD在程序细节的观察、动态修改方面有很大的作用，要主动用TD的调试功能帮助自己发现、理解与解决问题。

最后，由于操作不够熟练，时间比较紧张等原因，还有些问题需要以后进一步解决，如堆栈中数据变化的原因、各个段在内存中存放的关系、是否可跟踪到INT 21H中去、多次调入程序时初始的段值是否相同等等。

总之，通过本次实验，基本掌握了16位段相关的主要编程工具的使用；直观感受了数、符号、寻址方式等在计算机内的表现形式，指令执行时标志位的改变情况；能用分支、循环程序、DOS功能调用等设计、实现、调试一个汇编语言程序。

### 主要看法

汇编语言不同于其他的高级语言，汇编语言要求对计算机结构等都要有一定的了解，也需要换一种思维方式，这对我来说是个挺大的挑战。

汇编语言实验给我的的感觉是我的基本功还很不扎实，对很多语句不熟，寄存器的使用也不是那么熟练，因此编写程序过程中bug不断。但是通过实验我得到了锻炼，编写除了简单的汇编语言程序，也学会使用VS2019进行汇编程序的编写调试，还有DOXBox这一很好用的工具。

实验中暴露的问题值得我反思，一开始我对很多东西无从下手，因为不熟悉语句和结构，导致编程过程中出现了许多的bug，这使我教训惨重，耗费了大量的时间。

通过不断地实验锻炼，我相信我的汇编水平一定会大大提高。

# 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。

参考资料：

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9章、第10章）

## 任务2.1实验过程 *（研究性任务）*

### 实验方法说明

1. 对任务1.3的程序进行完善，界面显示增加了一定的美化，将商品信息重新定义为1个结构。
2. 实现功能5：按利润率从高到低显商品信息。我的设计思路是这样的：为了便于进行重复执行，需要保存原始结构，那么就不对数据段中的的GA1，GA2等进行操作，而是设计一个新的结构体SORT，里面有两个变量：dword类型的pos，用于存放商品信息的首地址；word类型的profit，用于存放利润。

FUNC5的第一步是创建SORT，SORT1的pos存放GA1的地址，profit存放GA1的利润，依次用SORT2、SORT3等存放所有商品的信息。

使用选择排序的算法，对SORT结构数组进行排序，每次交换数据都同时交换SORT中的pos和profit。

程序结束后，SORT1中存放利润率最高的商品首地址和利润率，SORT2存放第二高的……，依次。

1. 改进思路：在FUNC4的末尾将JMP MENU删除，这样在执行FUNC4后就直接执行FUNC5。FUNC4是计算出利润率，因此我认为计时应该在FUNC5的前后，将计时开始的代码放在创建SORT之前，计时结束的代码放在结束排序后。在计时中间加一个循环控制程序，让程序运行100000次，便于观察时间查别。
2. 使用winTimer，伪代码增加timeGetTime proto stdcall，includelib Winmm.lib。计时开始时调用代码CALL timeGetTime，此时EAX为开始时间，存入内存。程序结束后再调用CALL timeGetTime，此时EAX为结束时间，打印结束时间减结束时间。
3. 优化思路：
4. SORT中加入profit，这样在算法中比较的时候，就可以直接比较利润，而不用通过寻址找到商品地址，再找到利润地址了，以空间换取一定的效率优化。

SORT STRUCT

pos DWORD 0

profit DW 0

SORT ENDS

（本次实验两次比较前后均采用此结构）

1. 将xchg指令换为mov
2. 指令，大大提高了运算效率

优化前：

MOV EDX,dword ptr[SORT1[SI]]

XCHG EDX,dword ptr[SORT1[DI]]

XCHG EDX,dword ptr[SORT1[SI]]

ADD SI,4

ADD DI,4

MOV DX,word ptr[SORT1[SI]]

XCHG DX,word ptr[SORT1[DI]]

XCHG DX,word ptr[SORT1[SI]]

优化后：

MOV EDX,dword ptr[SORT1[SI]]

push EDX

MOV EDX,dword ptr[SORT1[DI]]

MOV dword ptr[SORT1[SI]],EDX

pop EDX

MOV dword ptr[SORT1[DI]],EDX

ADD SI,4

ADD DI,4

MOV DX,word ptr[SORT1[SI]]

push DX

MOV DX,word ptr[SORT1[DI]]

MOV word ptr[SORT1[SI]],DX

pop DX

MOV word ptr[SORT1[DI]],DX

### 实验记录与分析

改进后的界面显示；

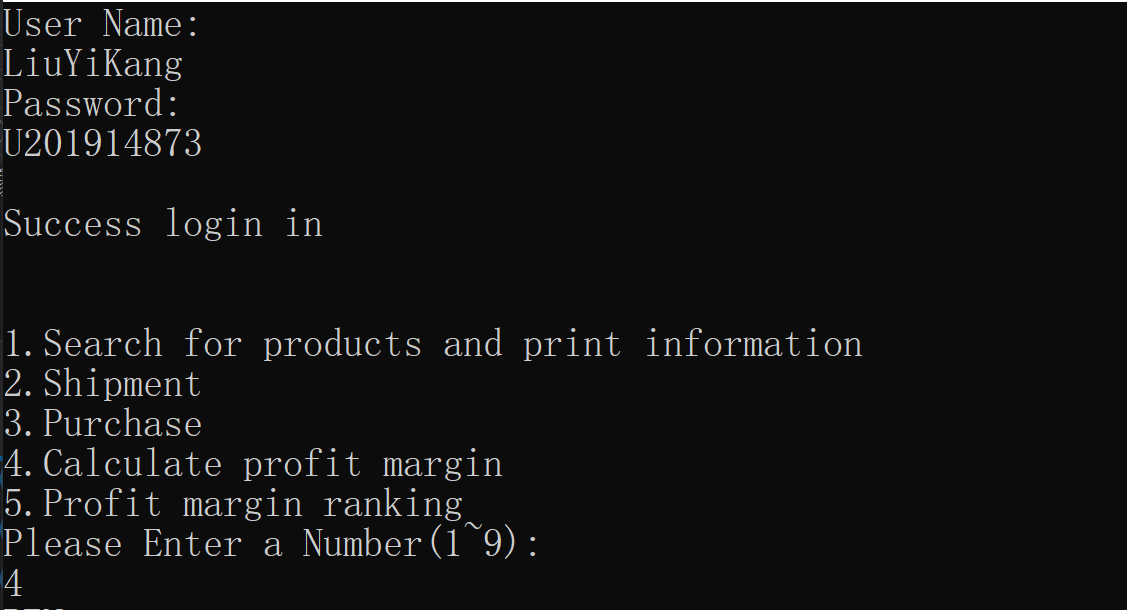


图2.1

FUNC4

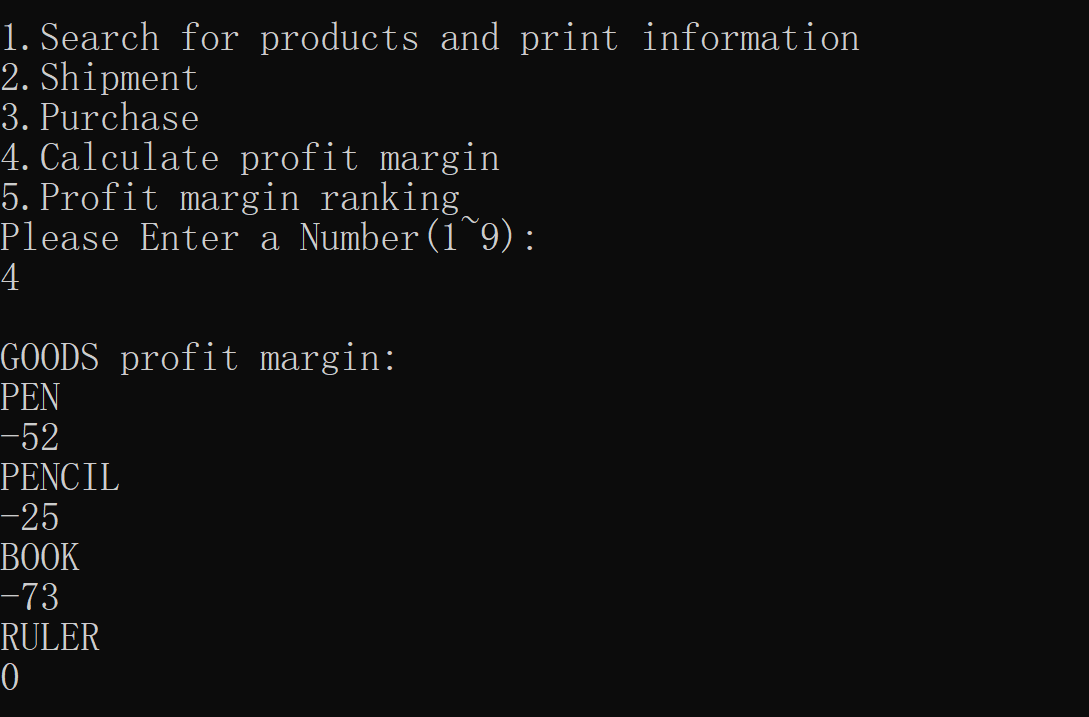


图2.2

代码改进前的FUNC5运行结果（xchg）



图2.3

代码改进后的FUNC5运行结果（MOV+push/pop）

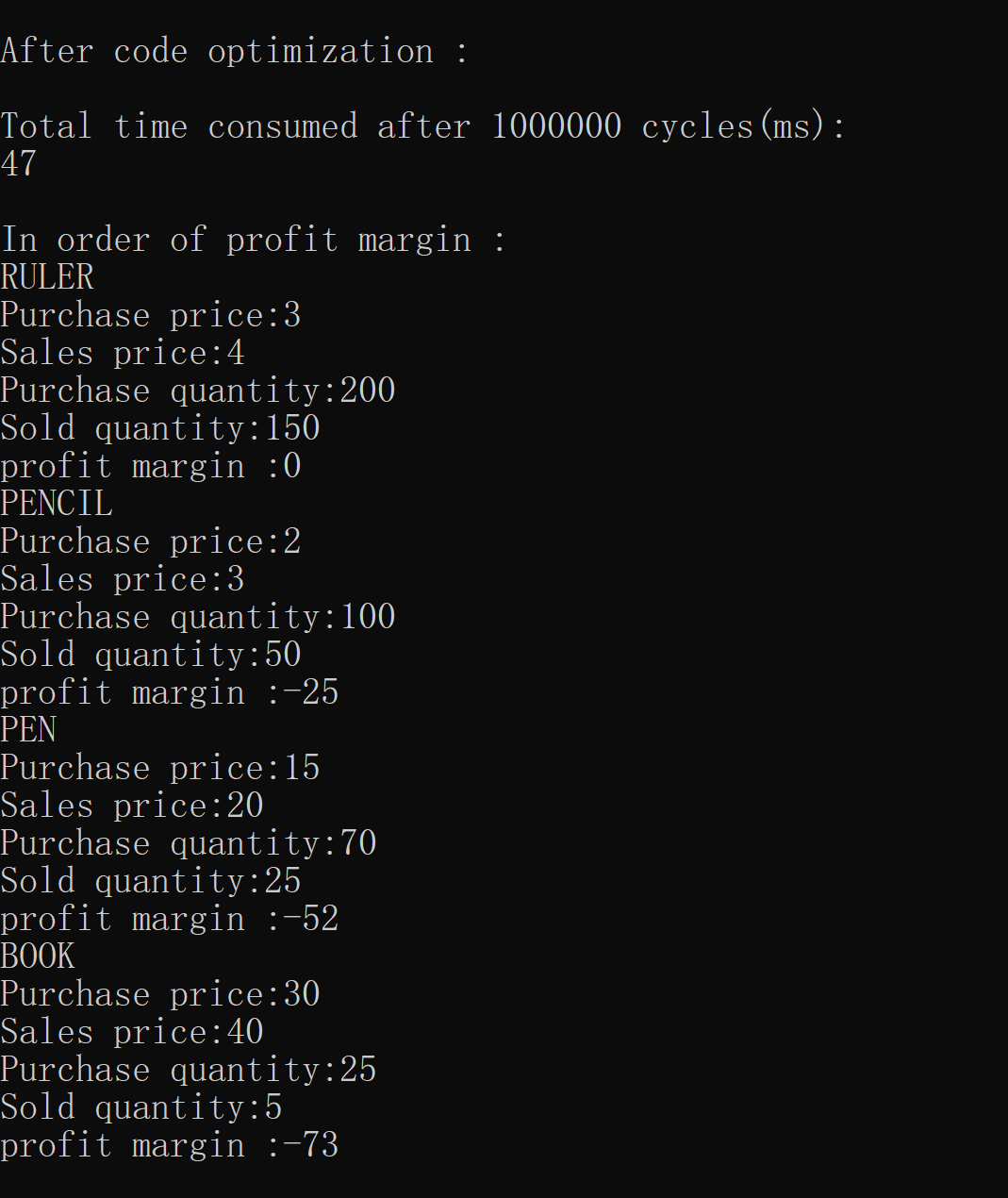


图2. 4

## 小结

### 主要收获

通过任务2的实验，我获得的第一个收获就是复习了课堂上所学习的知识点

1：将商品信息重新定义为结构，程序中我就可以通过结构访问的方式来访问数据。

2：掌握利用汇编语言实现排序算法，灵活使用寄存器和内存进行各种复杂的操作。

3： 引入外部的函数，添加计时功能

4： 尝试改进指令，通过比较两次排序的时间差别，思考实现相同功能的不同的指令的效率。

本次上机不仅提高了编程水平，熟悉了工具的使用，而且加深了对一些知识的理解。主要的经验教训如下：

首先，更加感受到实验前准备的意义。例如：上机前准备越充分（如先编好源程序，制定好准备做的一些步骤），上机的时候目的越明确，可以解决较多的问题。

其次，录入程序时要注意一些细节，比如中文分号、字母O等问题，虽然汇编程序指出其所在行有错，但很难发现具体是哪个符号错了，耽误了不少时间。

总之，通过本次实验，基本掌握了汇编语言中数据结构的定义，学习到了如何引入外部函数，以及改进算法的思路。

### 主要看法

汇编语言是底层的语言，与CPU相关，因此在做一些简单的算法，如本次实验的排序算法，较使用C语言这种较为高级的语言难度大得多。而且由于通用寄存器有限，且某些寄存器有特殊的用途，因此我们要熟练掌握寄存器和指令的应用。

现在的CPU计算速度很快，在Intel Pentium以上级别的CPU中，有一个称为“时间戳（Time Stamp）”的部件，它以64位无符号整型数的格式，记录了自CPU上电以来所经过的时钟周期数。由于目前的CPU主频都非常高，因此这个部件可以达到纳秒级的计时精度。因此我们在使用ms级的计时程序，不能直接观察到几条指令对程序执行时间的影响。可以通过大量重复（比如重复执行10000次以上等）执行几条指令的方法来解决此问题。这种方法有很强的实用性。我们在需要实现某功能时，要灵活运用各种方法。

# 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的实验内容

**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。

### 任务3.2的实验内容

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？
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。

## 3.3任务3.1实验过程

### 3.3.1设计思想及存储单元分配

（1）实现字符串比较的子程序

将之前实验中在主程序中写出的字符串比较代码转化为子程序，子程序名为stcmp，有两个参数buf1和buf2，均为双字类型，保存了地址。

strcmp proc str1:dword,str2:dword

push esi

push edi

push edx

MOV edi,str1

MOV esi,str2

strcmp\_start:

mov dl,[edi]

cmp dl,[esi]

ja strcmp\_large

jb strcmp\_little

cmp dl,0

je strcmp\_equ

inc esi

inc edi

jmp strcmp\_start

strcmp\_large:

mov eax,1

jmp strcmp\_exit

strcmp\_little:

mov eax,-1

jmp strcmp\_exit

strcmp\_equ:

mov eax,0

strcmp\_exit:

pop edx

pop edi

pop esi

ret

strcmp endp

将在登陆时比较用户名和密码，查找商品时比较商品的时候调用这个函数

（2）定义的一些带参数的子程序

1.（1）中的strcmp

2.GOODSELL，将FUNC4的内容封装为子程序，有两个参数Shipment1:word,pos:dword，分别为出货数量以及要出货的商品的地址。

代码如下

GOODSELL proc Shipment1:word,pos:dword

MOV EDI,pos

MOV AX,(GOODS1 ptr [EDI]).BUYNUM

MOV BX,(GOODS1 ptr [EDI]).SELLNUM

MOV DX,Shipment1

SUB AX,BX

CMP AX,DX

JB NOTENOUGH

ADD (GOODS1 ptr [EDI]).SELLNUM,DX

invoke printf,offset lpFmt1,OFFSET SOLDGOODS

invoke printf,offset lpFmt3,(GOODS1 ptr [EDI]).SELLNUM

invoke printf,offset lpFmt,OFFSET UPDATEPRIOT

JMP ENDGOODSSELL

NOTENOUGH:

invoke printf,offset lpFmt,OFFSET NENOUGH

ENDGOODSSELL:

ret

GOODSELL endp

将在选择功能3出货的时候调用它。

1. 定义的一些不带参数的子程序
2. CalculateRate，用于计算商品的利润率，其参数由栈传入，而不在函数名后定义。当选择功能4计算所有商品的利润率时调用它

CalculateRate proc

push ebp

mov ebp,esp

invoke printf,offset lpFmt,OFFSET NEWLINE

invoke printf,offset lpFmt,OFFSET GOODSPROFIT

MOV ESI, -1

push ESI

ALLGOOD: ;选择下一个比较商品

pop ESI

INC ESI ;查看商品是否查找完

push ESI

CMP ESI, AMOUNT

JE ENDCalculate

IMUL ESI, 20

ADD ESI, OFFSET GA1 ;ESI此时存放当前商品的首地址

MOV EAX,0

MOV EBX,0

invoke printf,ESI

invoke printf,offset lpFmt,OFFSET NEWLINE

MOV AX,(GOODS ptr[ESI]).BUYPRICE ;进货价

MOV BX,(GOODS ptr[ESI]).BUYNUM ;进货数量

IMUL BX

PUSH AX ;进货

MOV AX,(GOODS ptr[ESI]).SELLPRICE ;销售价

MOV BX,(GOODS ptr[ESI]).SELLNUM ;销售数量

IMUL BX

PUSH AX ;售货

MOV EAX,0

MOV EBX,0

pop AX

POP BX

SUB EAX,EBX

IMUL EAX,EAX,100

CDQ

idiv EBX

MOV (GOODS ptr[ESI]).RATE,AX

invoke printf,offset lpFmt3,EAX

jmp ALLGOOD

ENDCalculate:

pop ESI

pop ebp

ret

CalculateRate endp

1. SORTBEFORE，用于实现功能5排序，其参数由栈传入，而不在函数名后定义。当运行功能5，开始计时改进前排序效率的时候调用它。

;改进前的选择排序

SORTBEFORE proc

push ebp

mov ebp,esp

;以下这段代码的作用是创建一个结构数组，保存地址和利润

MOV EAX, -1 ;选择下一个比较商品

SORTKEEP:

INC EAX ;查看商品是否查找完

CMP EAX,AMOUNT

JE FUNC5KEEP

MOV EBX,EAX

IMUL EBX,6

MOV ECX,EAX

IMUL ECX,20

ADD ECX,offset GA1

ADD EBX,offset SORT1

MOV dx,(GOODS ptr [ECX]).RATE

MOV (SORT ptr [EBX]).RATE,dx

MOV [EBX],ECX

JMP SORTKEEP

;以下为选择排序算法

FUNC5KEEP:

MOV AX,0

BEGINLOOP1:

inc AX

CMP AX,word ptr[AMOUNT]

Jnl ENDLOOP1

dec AX

MOV BX,AX

MOV CX,AX

INC CX

BEGINLOOP2:

CMP CX,word ptr[AMOUNT]

jnl ENDLOOP2

MOV EDI,0

MOV ESI,0

MOV SI,CX

MOV DI ,BX

IMUL SI,6

IMUL DI,6

ADD SI,4

ADD DI,4

MOV DX,word ptr SORT1[DI]

CMP word ptr SORT1[SI],DX

jng GO

MOV BX,CX

GO:

MOV EDI,0

MOV ESI,0

MOV SI,AX

MOV DI ,BX

IMUL SI,6

IMUL DI,6

MOV EDX,dword ptr[SORT1[SI]]

XCHG EDX,dword ptr[SORT1[DI]]

XCHG EDX,dword ptr[SORT1[SI]]

ADD SI,4

ADD DI,4

MOV DX,word ptr[SORT1[SI]]

XCHG DX,word ptr[SORT1[DI]]

XCHG DX,word ptr[SORT1[SI]]

INC CX

JMP BEGINLOOP2

ENDLOOP2:

INC AX

JMP BEGINLOOP1

ENDLOOP1:

pop ebp

ret

SORTBEFORE endp

1. SORTAFTER，用于实现功能5排序，其参数由栈传入，而不在函数名后定义。当运行功能5，开始计时改进后排序效率的时候调用它。

;改进后的排序

SORTAFTER proc

push ebp

mov ebp,esp

;以下这段代码的作用是创建一个结构数组，保存地址和利润

MOV EAX, -1 ;选择下一个比较商品

SORTKEEPAFTER:

INC EAX ;查看商品是否查找完

CMP EAX,AMOUNT

JE FUNC5KEEPAFTER

MOV EBX,EAX

IMUL EBX,6

MOV ECX,EAX

IMUL ECX,20

ADD ECX,offset GA1

ADD EBX,offset SORT1

MOV dx,[ECX+18]

MOV [EBX+4],dx

MOV [EBX],ECX

JMP SORTKEEPAFTER

;以下为选择排序算法

FUNC5KEEPAFTER:

MOV AX,0

BEGINLOOP1AFTER:

inc AX

CMP AX,word ptr[AMOUNT]

Jnl ENDLOOP1AFTER

dec AX

MOV BX,AX

MOV CX,AX

INC CX

BEGINLOOP2AFTER:

CMP CX,word ptr[AMOUNT]

jnl ENDLOOP2AFTER

MOV EDI,0

MOV ESI,0

MOV SI,CX

MOV DI ,BX

IMUL SI,6

IMUL DI,6

ADD SI,4

ADD DI,4

MOV DX,word ptr SORT1[DI]

CMP word ptr SORT1[SI],DX

jng GOAFTER

MOV BX,CX

GOAFTER:

MOV EDI,0

MOV ESI,0

MOV SI,AX

MOV DI ,BX

IMUL SI,6

IMUL DI,6

MOV EDX,dword ptr[SORT1[SI]]

push EDX

MOV EDX,dword ptr[SORT1[DI]]

MOV dword ptr[SORT1[SI]],EDX

pop EDX

MOV dword ptr[SORT1[DI]],EDX

ADD SI,4

ADD DI,4

MOV DX,word ptr[SORT1[SI]]

push DX

MOV DX,word ptr[SORT1[DI]]

MOV word ptr[SORT1[SI]],DX

pop DX

MOV word ptr[SORT1[DI]],DX

INC CX

JMP BEGINLOOP2AFTER

ENDLOOP2AFTER:

INC AX

JMP BEGINLOOP1AFTER

ENDLOOP1AFTER:

pop ebp

ret

SORTAFTER endp

（4）宏定义

将程序中经常用到的语句段定义为宏，节省代码空间。

如下定义了一个打印出商品全部内容的宏，传入参数pos为商品首地址

PRINTGOOD macro POS

MOV EDI,POS

invoke printf,offset lpFmt1,EDI

invoke printf,offset lpFmt,OFFSET NEWLINE

invoke printf,offset lpFmt1,OFFSET Purchase\_price

invoke printf,offset lpFmt3,(GOODS ptr [EDI]).BUYPRICE

invoke printf,offset lpFmt1,OFFSET Sales\_price

invoke printf,offset lpFmt3,(GOODS ptr [EDI]).SELLPRICE

invoke printf,offset lpFmt1,OFFSET Purchase\_quantity

invoke printf,offset lpFmt3,(GOODS ptr [EDI]).BUYNUM

invoke printf,offset lpFmt1,OFFSET Sold\_quantity

invoke printf,offset lpFmt3,(GOODS ptr [EDI]).SELLNUM

invoke printf,offset lpFmt1,OFFSET PROFIT

invoke printf,offset lpFmt4,(GOODS ptr[EDI]).RATE

endm

（5）使用模块化设计

将strcmp和GOODSELL放在一个单独的子模块文件中，两个模块中，主模块含有堆栈段和数据段。

在主模块中，将子模块所需要的变量声明为了PUBLIC，具体变量和类型如下:

public lpFmt

public lpFmt1

public lpFmt3

public NENOUGH

public SOLDGOODS

public UPDATEPRIOT

在子模块中，将需要用的变量声明extern

extern lpFmt:sbyte

extern lpFmt1:sbyte

extern lpFmt3:sbyte

extern NENOUGH: sbyte

extern SOLDGOODS: sbyte

extern UPDATEPRIOT: sbyte

在主模块中使用了两个外部函数，stcmp和GOODSELL都为NEAR类型。

主模块引入两个函数

strcmp proto C:dword,:dword

GOODSELL proto C:word,:dword

（6）问题探究

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？

**探究：**因为当前的程序是flat扁平模式的，所以用NEAR类型的子程序的都不会报错，FAR都会报错。

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。

**探究：**不同模块的数据段和代码段自动进行了合并。

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？

**探究：**LINK类型报错。

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

可以看到宏会自动替换，而子程序是跳转。

### 3.3.2实验步骤

（1）准备实验环境

（2）测试宏指令的功能

（3）测试新增的子程序

1. 测试将源代码分散到两个不同的源文件中后能否正常运行

（5） 整体测试

### 3.3.3实验记录与分析

（1）准备实验环境

（2）测试宏指令的功能

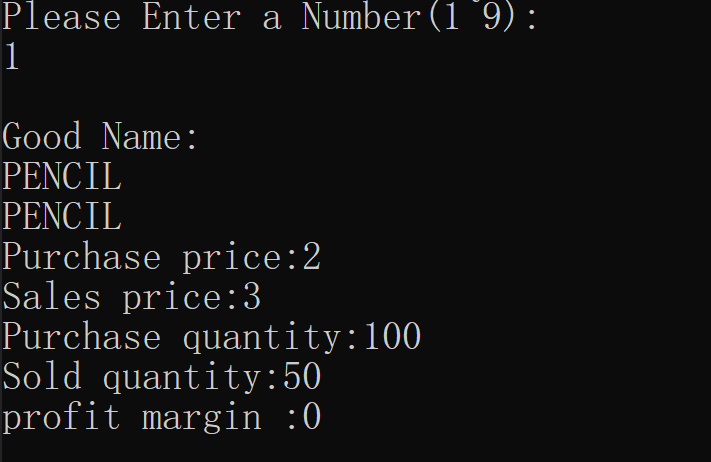


图3.1 宏指令测试

如图，宏指令测试成功，可以成功打印出商品信息

（3）测试新增的子程序

1. strcmp

测试字符串相等时能否正确返回值

****

图3.2

测试字符串不相等时能否正确返回值

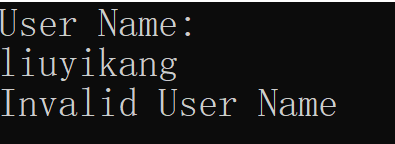
****

图3.3

因此strcmp能正确比较字符串

1. GOODSELL

当出售商品小于剩余商品的情况

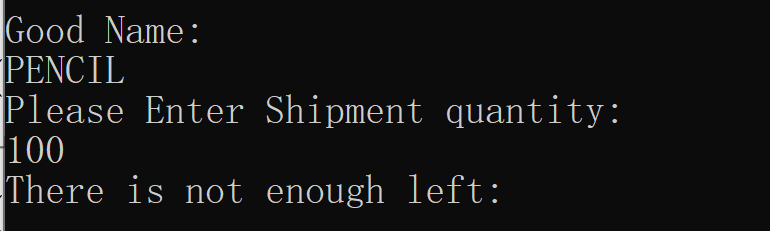
****

图3.4

当出售商品大于剩余商品的情况

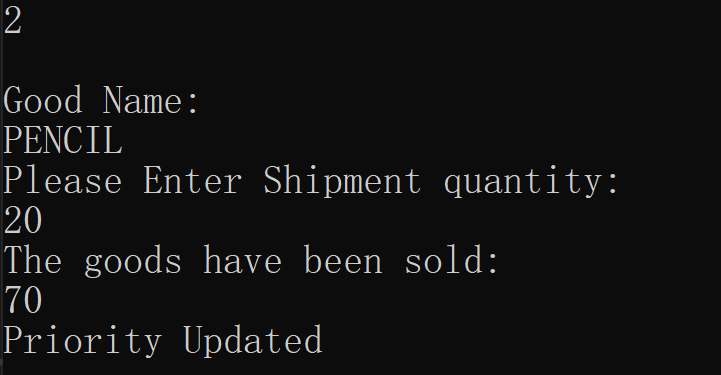
****

图3.5

GOODSELL子程序功能正确

1. CalculateRate

测试其功能，看能否计算出所有商品的利润率

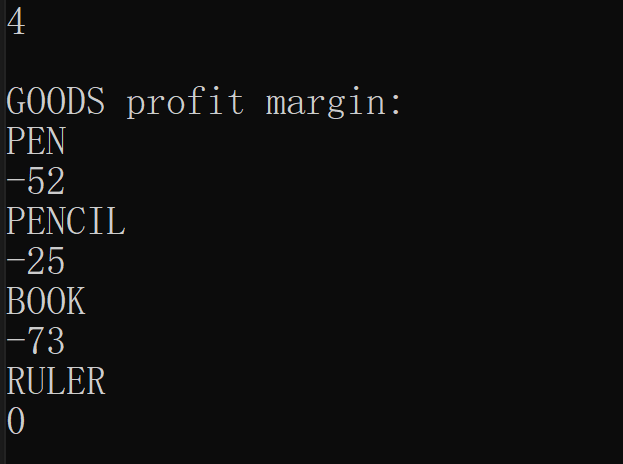
****

图3.6

功能正确

1. SOTBEFOE

通过观察输出排序结果及时间，判断此子程序的功能

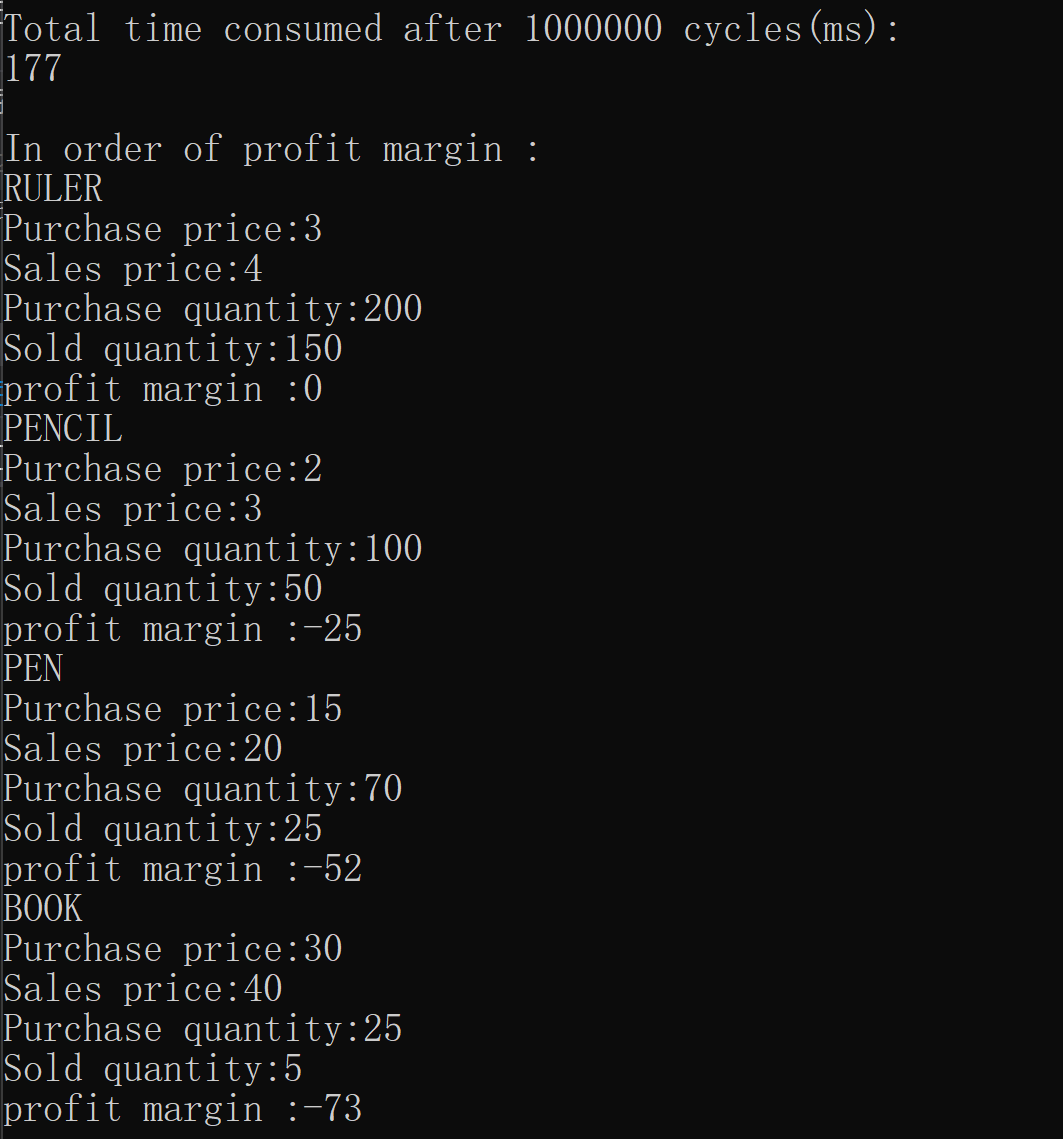
****

图3.7

1. SORTAFTER

通过观察输出排序结果及时间，判断此子程序的功能

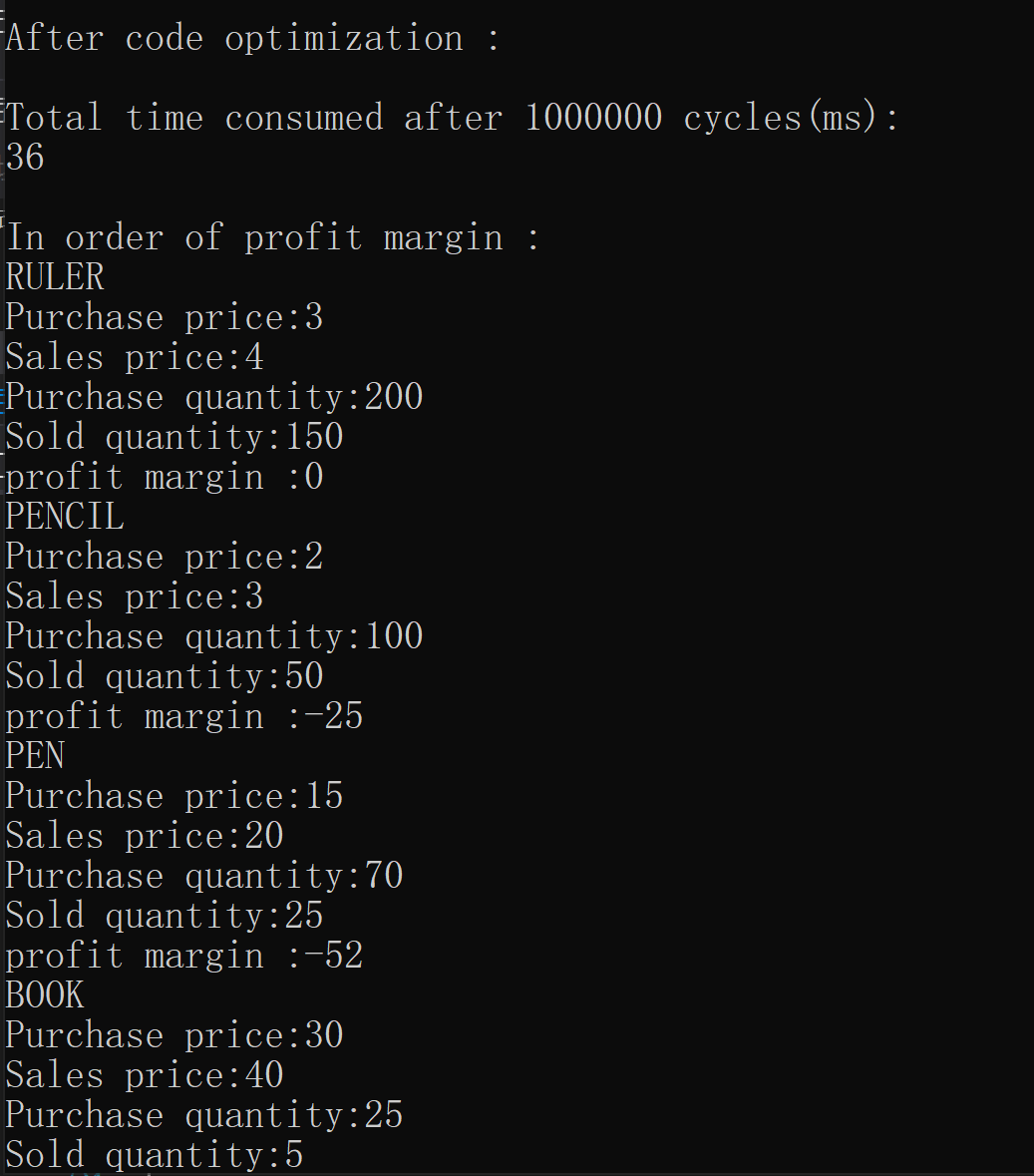
****

图3. 8

（4）测试将源代码分散到两个不同的源文件中后能否正常运行

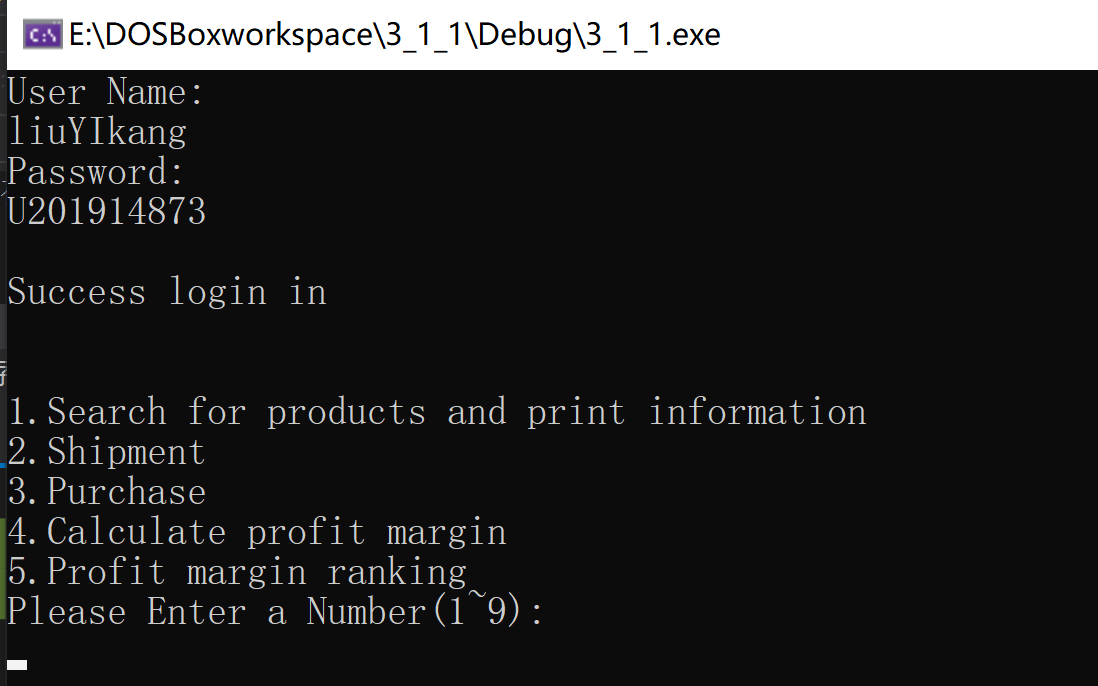


图3.9

如图，能正常运行

## 3.4任务3.2实验过程

### 3.4.1实验方法说明

1. 准备实验环境 Windows10+VS2019

2. 新建VS2019项目

3. 编写C程序的源文件，作为主界面输出和选择功能

4. 添加功能6，实现添加商品的功能

5. 添加汇编源文件，实现多项功能

6. 编译，链接程序

7. 测试程序各项功能是否正常

### 3.4.2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：**

#include<stdio.h>

#include <string.h>

#ifdef \_\_cplusplus

extern "C" {

#endif

#ifdef \_\_cplusplus

}

#endif

void FUNC1(void);

void FUNC2(void);

void FUNC3(void);

void FUNC4(void);

void FUNC6(void);

void FUNC8(void);

char trueName[10] = "liuYIkang\0";

char truePassword[15] = "U201914873\0";

int main(void)

{

char inName[10];

char inPassword[15];

printf("User Name:\n");

scanf("%s", inName);

if (strcmp(inName, trueName) != 0)

{

printf("Invalid User Name\n");

printf("Input User Name again!\n");

scanf("%s", inName);

}

printf("Password:\n");

scanf("%s", inPassword);

if (strcmp(inPassword, truePassword) != 0)

{

printf("Invalid Password\n");

printf("Input Password again!\n");

scanf("%s", inPassword);

}

printf("Success Login in!\n");

int option = 0;

while (option != 9)

{

printf("Welcome to my shop!\n");

printf("Please choice function 1-9\n");

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

printf("1.Search for products and print information\n");

printf("2.Shipment\n");

printf("3.Purchase\n");

printf("4.Calculate profit margin\n");

printf("5.Profit margin ranking\n");

printf("6.Add Good\n");

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

printf("Please Enter a Number(1~9):");

scanf("%d", &option);

printf("\n");

switch (option)

{

case 1:

FUNC1();

break;

case 2:

FUNC2();

break;

case 3:

FUNC3();

break;

case 4:

FUNC4();

break;

case 6:

FUNC6();

break;

default:

break;

}

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

system("pause");

system(("cls"));

}

return 0;

}

**3. 添加功能六，C语言实现添加商品**

void FUNC6(void)

{

AMOUNT++;

GOODS(\* ptr)[10] = &GA1;

printf("Please input new GOOD's name\n");

scanf("%s", &((\*ptr)[AMOUNT-1].GOODSNAME));

printf("Please input new GOOD's buyprice\n");

scanf("%hd", &((\*ptr)[AMOUNT - 1].BUYPRICE));

printf("Please input new GOOD's sellprice\n");

scanf("%hd", &((\*ptr)[AMOUNT - 1].SELLPRICE));

printf("Please input new GOOD's buynum\n");

scanf("%hd", &((\*ptr)[AMOUNT - 1].BUYNUM));

short a;

printf("Please input new GOOD's sellnum\n");

scanf("%hd", &a);

while (a > ((\*ptr)[AMOUNT - 1].BUYNUM))

{

printf("Sell number can't be greater than buy number!\n");

printf("Please input new GOOD's sellnum\n");

scanf("%hd", &a);

}

((\*ptr)[AMOUNT - 1].SELLNUM) = a;

((\*ptr)[AMOUNT - 1].RATE) = 0;

printf("New good %s had beem added\n", (\*ptr)[AMOUNT - 1].GOODSNAME);

}

**4. 创建汇编源文件，包括宏指令，数据段，堆栈段，代码段等，同时将c文件中需要用到的全局变量声明为public**

汇编源文件主要封装了功能1到5，功能1到5为汇编文件中的函数，可以直接让C程序调用

宏语句：打印传入地址的商品信息

PRINTGOOD macro POS

```

endm

函数：比较两字符串是否相等

strcmp1 proc str1:dword,str2:dword

```

strcmp1 endp

函数：卖出货品，判断剩余量是否足够

GOODSELL proc Shipment1:word,pos:dword

```

GOODSELL endp

函数：查找商品

SearchGOOD proc str1:dword,str2:dword

```

SearchGOOD endp

函数：计算所有商品的利润率

CalculateRate proc

```

CalculateRate endp

函数：改进前的商品利润率排序

;改进前的选择排序

SORTBEFORE proc

```

SORTBEFORE endp

函数：改进后的商品利润率排序

;改进后的排序

SORTAFTER proc

```

SORTAFTER endp

C文件中的函数FUNC1到5就是将上述函数封装得到的函数，C文件中需要对其声明

void FUNC1(void);

void FUNC2(void);

void FUNC3(void);

void FUNC4(void);

void FUNC5(void);

**5． 测试程序功能**

显示程序主界面

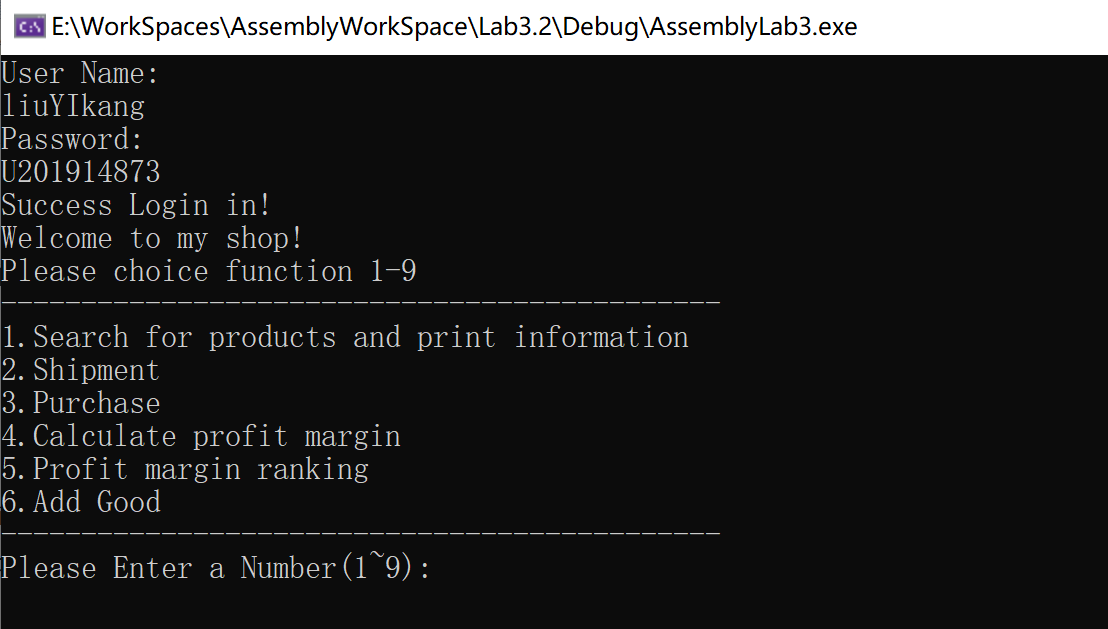


图3.9 程序主界面

验证功能一，查找商品

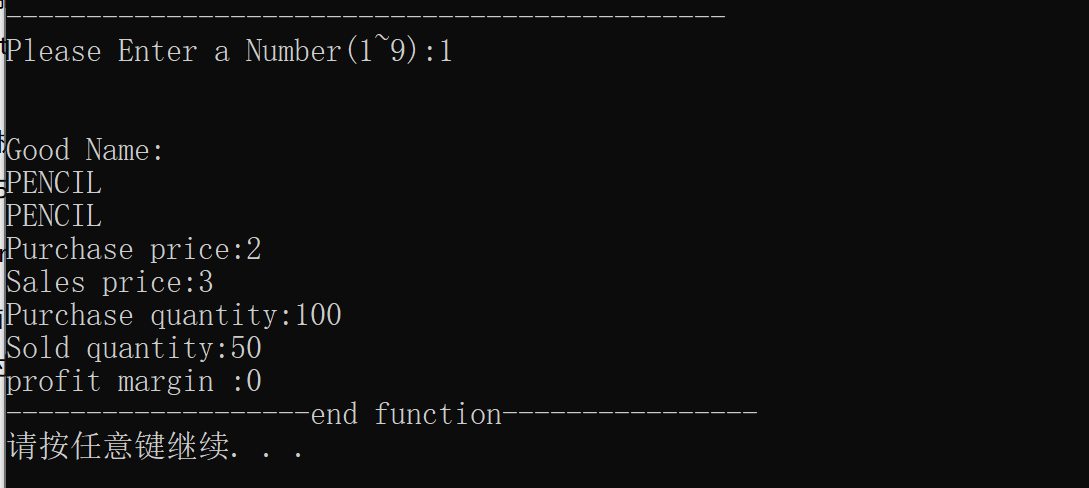


图3.10 验证功能一

验证功能二，出货

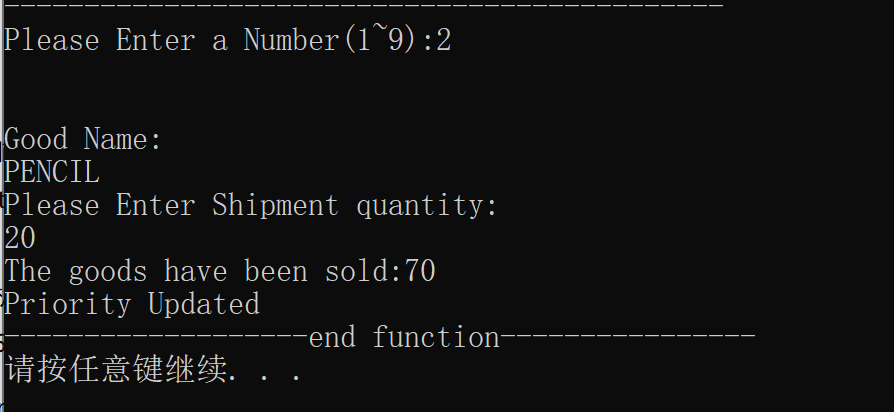


图3.11 功能二成功出货

****

图3.12 功能二货物不足

验证功能三，进货

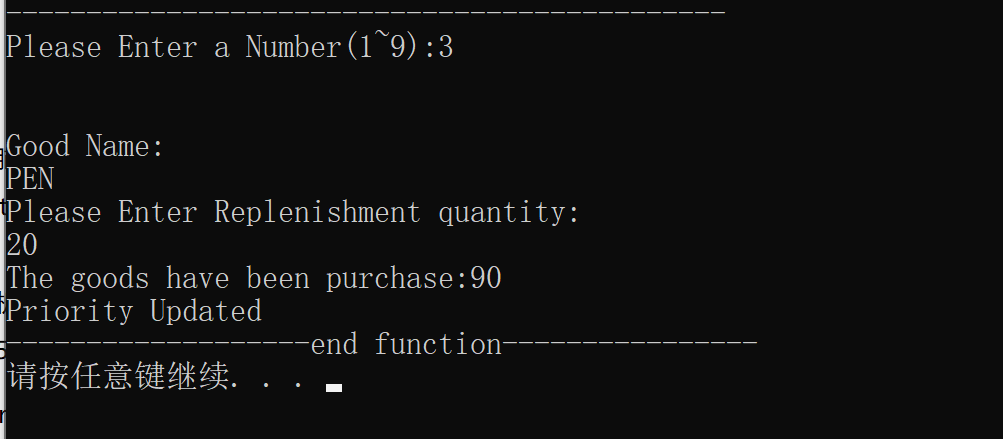


图3.13 功能三，进货

验证功能四，计算利润率

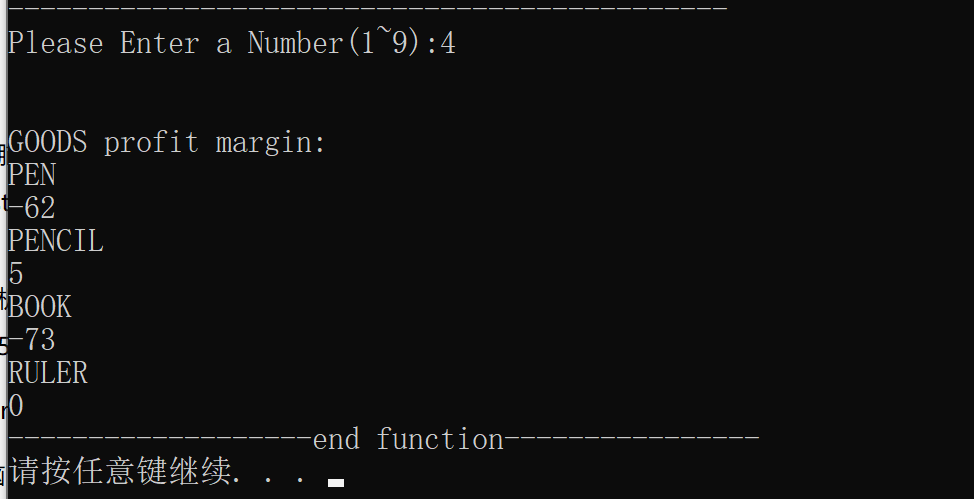


图3.14 功能四，计算利润率

验证功能5， 商品利润率排序

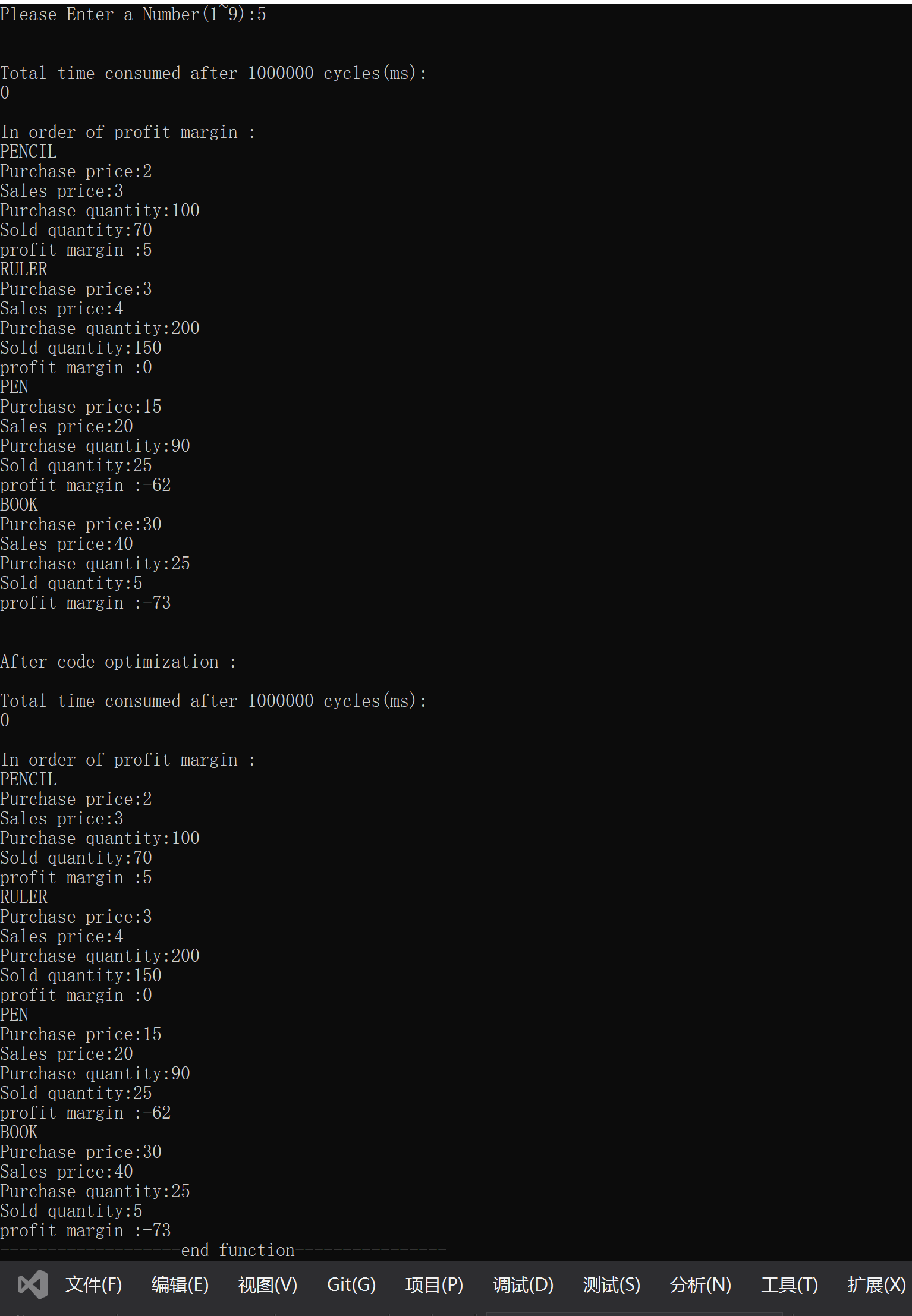


图3.15 功能五，利润率排序

验证功能六， 新增商品

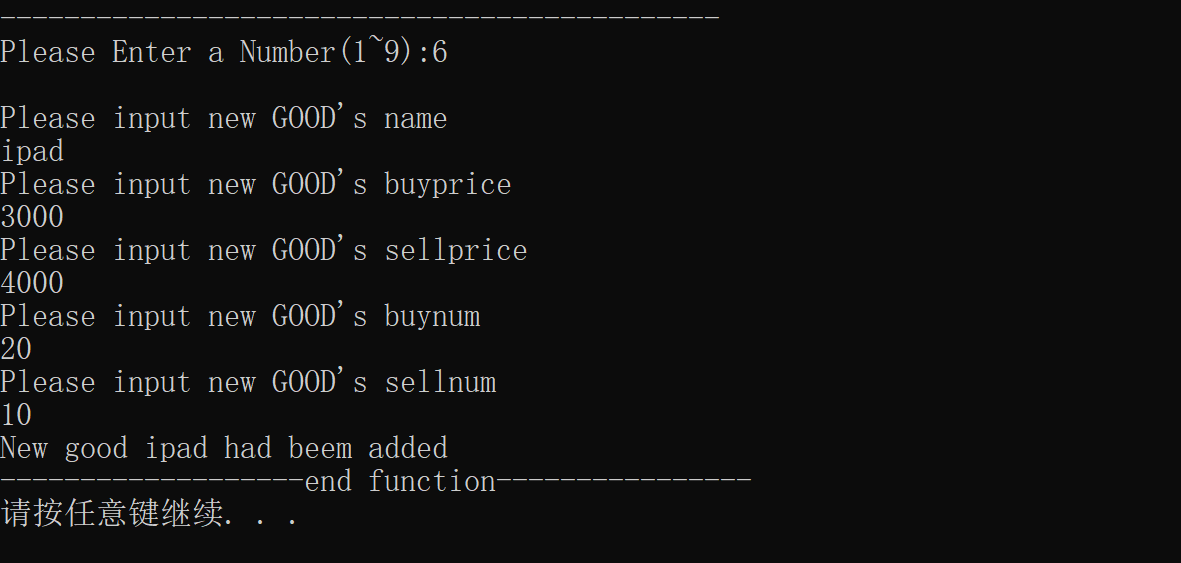


图3.16 新增商品

经过验证，C语言与汇编语言成功得链接，得到正确的商店管理系统，使用C语言编写的主界面也完成。

本次实验较为难的地方在如何使C语言与汇编语言相互调用，比如说：在C语言内，需要使用extern语句声明汇编函数原型和变量，才可以使用数据。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文件等。

**6. 回答问题：**

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？

C语言中说明外部变量时，用extern修饰；说明外部函数时候，声明外部函数即可。

汇编文件中，说明外部变量，使用extern;说明外部函数时候使用格式：函数名proto c 。

C语言语句访问汇编语言定义的变量时，反汇编后变量仍是内存中的一部分

1. 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？

在C语言中：先求出地址，然后加上偏移量就可以获得要改变的地址。所获得的地址其实是goods信息地址的地址，直接修改即可

1. 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）
2. 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；）

告诉编译器地址指向的数据是什么。

char a[10]={0，1，2，3，4，5，6，7，8，9};

char a[10] = "123456789";

此时&a上的数据如图



\*(int\*)a = 123;

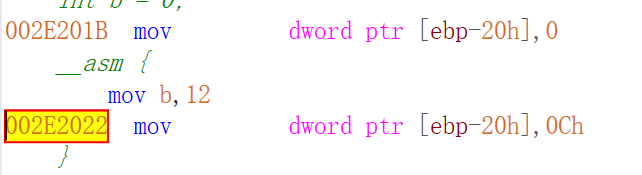
此时&a上的数据如图



1. 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？

无参数通过CALL来调用，有参数通过invoke来调用。函数之间通过栈来传递信息。

1. 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。如图3.19。



## 3.5 实验小结

### 3.5.1主要收获

这次实验，主要练习了汇编语言的模块化编程，即将复杂的程序分成模块，使得代码整齐，简洁，同时也便于维护和修改，并且可以方便的调用某些代码，减小了代码量。模块化编程是一个程序员必备的技能，这使得我们可以更好的编程，使代码可读性增强，同时也可以帮助我们提高多人协作能力。汇编语言是较为底层的语言，通过模块化编程我也能更好的学习一个程序是如何运行的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，比如声明为extern或者public的变量，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。

通过学习C语言和汇编语言的混合编程，我也更好的体会到一个程序从源文件编译成可执行文件的过程，先生成目标文件，再通过链接程序组装在一起，理解更深层的架构，使得我的编程能力有所提高

# 4中断与反跟踪

## 4.1实验目的与要求

（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；

1. 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。

## 4.2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。

**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，将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。由于进货价已经被加密后，需要进货价信息的部分需要先解密再使用。

采用计时、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）。

**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。

## 4.3任务4.1实验过程

### 4.3.1设计思想及存储单元分配

主程序需要检查中断处理程序是否已经安装，若已经安装则退出，否则安装中断处理程序接管8号时钟中断，随后驻留退出。

实方式下类型码为8的中断时系统时钟中断，在系统加电时被初始化成每秒产生18.2次中断，每次中断时，新中断处理程序首先调用原8号中断处理程序的功能，然后对中断次数计数，采用倒计数的方法，当计数值从18减到1时，直接从中断返回，当计数值减到0时，则重给计数器赋初值，同时获取系统时间，显示。

**用到的变量：**

;新INT08H使用的变量

    COUNT   DB  18          ;滴答计数

    HOUR    DB  ?,?,':'     ;时的ASCII码

    MIN     DB  ?,?,':'     ;分的ASCII码

    SEC     DB  ?,?         ;秒的ASCII码

    BUF\_LEN=$-HOUR          ;显示信息的长度

    CURSOR  DW  ?           ;原光标位置

OLD\_INT DW  ?,?         ;原INT 08H的中断矢量

  INSTALLED\_HINT DB 0DH,0AH,'Program has been installed yet.',0DH, 0AH,0DH, 0AH, '$'

    TMP      DB ?,?

可以直接修改内存中的单元来修改时间显示的位置

### 4.3.2流程图

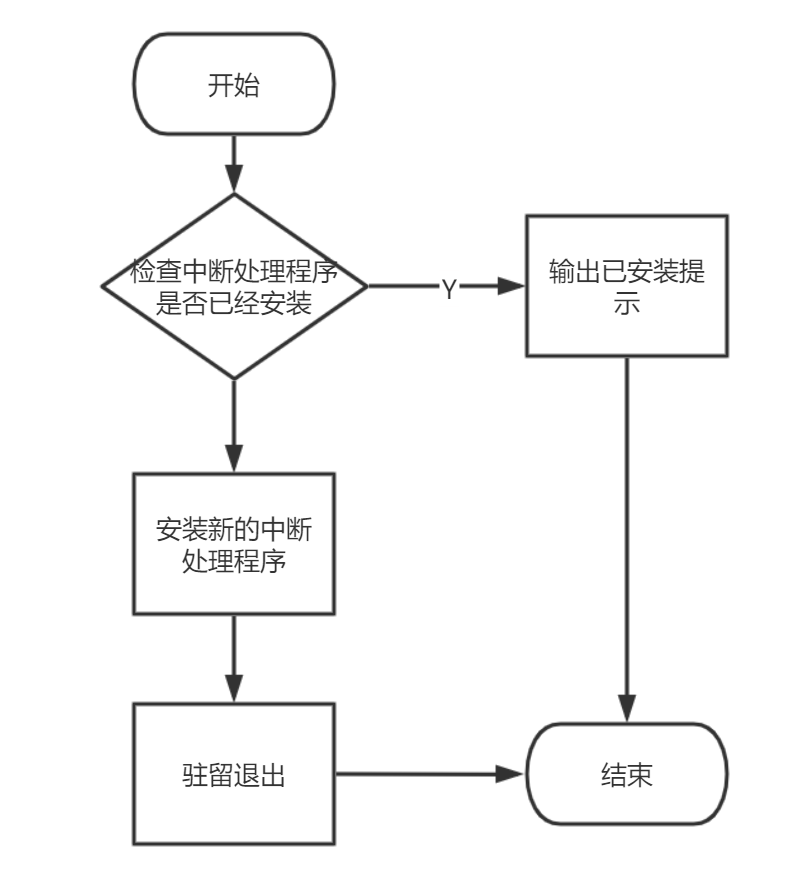


图4.1主程序流程图

### 4.3.3源程序

与书上的源程序不同的位置主要在主函数begin

 ;--------------------------------------------------------------------------

        ;INT 21 : AH=35时，获取标号为AL的中断的偏移地址，存放到BX

        MOV  AX,3508H               ;利用INT 21H 的35H入口参数获取原来08H的中断矢量

        INT  21H                    ;系统功能调用35H的入口/出口参数，返回中断向量ES:BX

        MOV  OLD\_INT,BX             ;保存中断矢量

        ;BX为当前中断为8的偏移地址，判断与NEW08H是否相同，则可以判断出NEW08H是否已被安装

        CMP  bx, OFFSET NEW08H

        JZ INSTALLED

        MOV  OLD\_INT+2,ES

        ;INT 21 : AH=25时，设置标号为AL的中断的偏移地址，设为DX

        MOV  DX,OFFSET NEW08H

        MOV  AX,2508H               ;利用25入口设置新的中断向量

        INT  21H

NEXT:   MOV  AH,0                   ;等待按键

        INT  16H

        CMP  AL,'q'

        JNE  NEXT                   ;如果按下了q则退出

       ;------------------------------------------

       ;设置驻留退出

        MOV DX,OFFSET BEGIN+15

        MOV CL,4

        SHR DX,CL

        ADD DX,10H

        MOV AL,0

        MOV AH,31H

        INT 21H

        ;-------------------------------------------

INSTALLED:

       LEA   DX, INSTALLED\_HINT  ; 显示提示串

       MOV   AH, 9

       INT   21H

       JMP   NEXT

### 4.3.4实验步骤

1. 阅读示例代码与参考资料，学习中断与驻留退出等实验需要的知识

2. 修改示例代码，实现额外的检查重复安装功能和指定位置功能

3. 使用TD调试并分析代码

### 4.3.4实验记录与分析

（一）

功能测试

打开DosBox，运行代码。

1. 未安装时，在屏幕指定位置显示时间

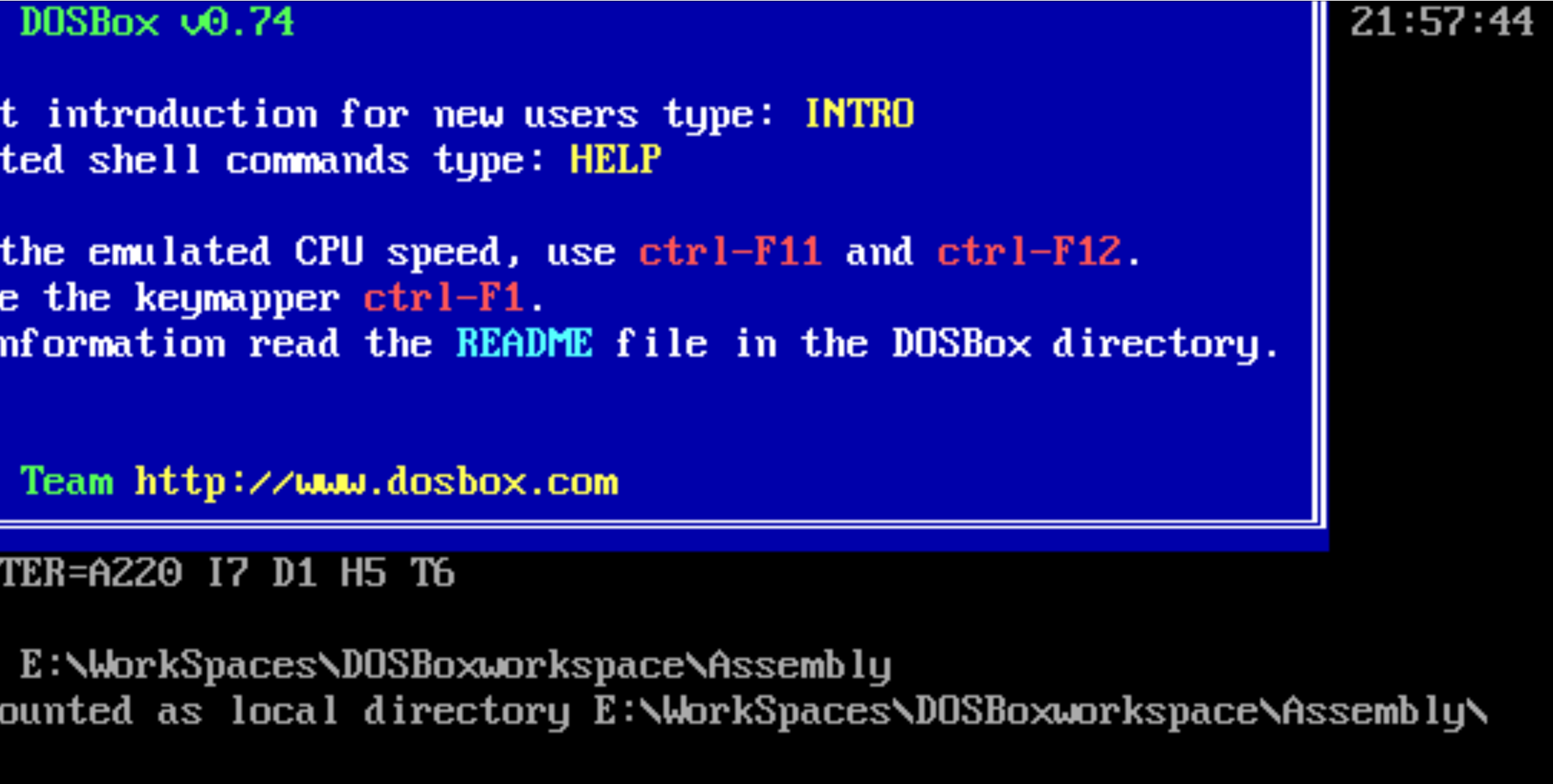


图4.2 显示时间

1. 退出后仍在屏幕指定位置显示时间

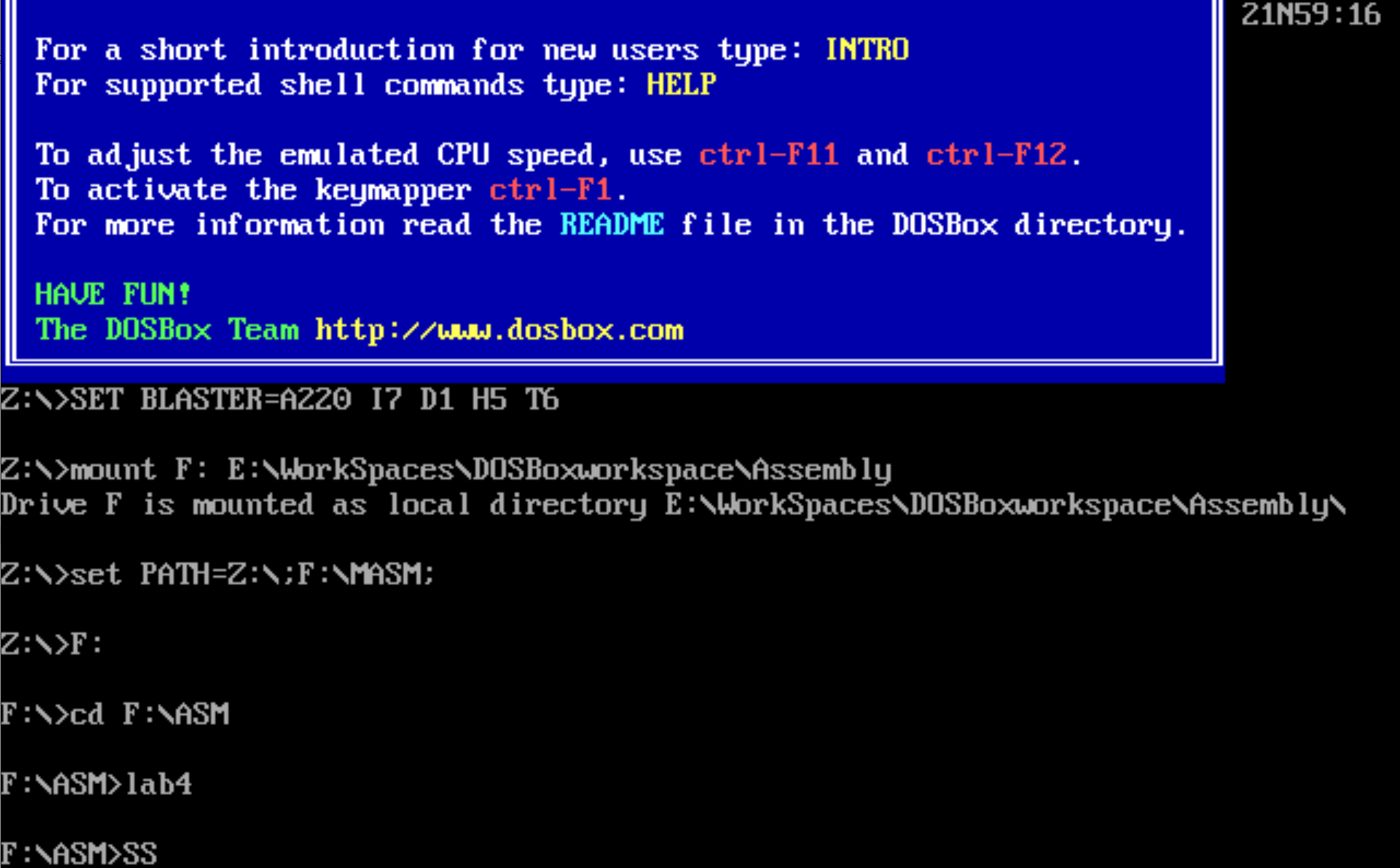


图4.3 驻留退出

1. 已安装后再运行程序会给出提示

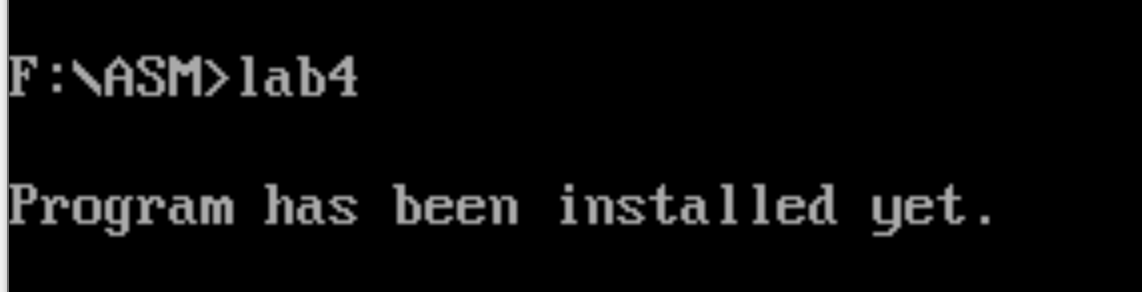


图4.4 重复安装

（二）使用TD功能调试程序并探究以下问题

1. 观察中断矢量表

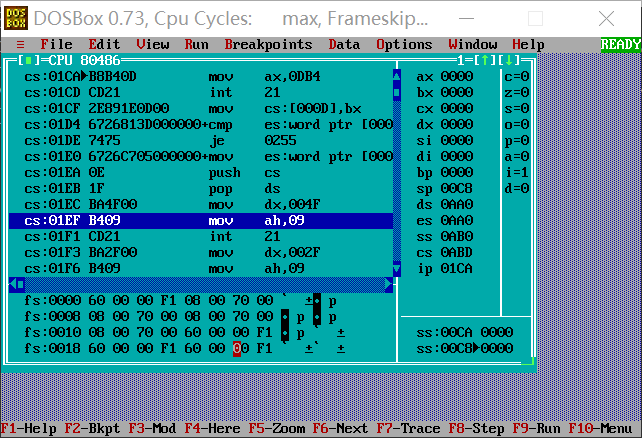


图4.5中断矢量表观察结果

在此模式下，中断矢量表起点位于0:0处，且存储着不同中断的处理函数地址，观察结果如图4.3.2所示。

2. 观察已有的某个中断处理程序的代码

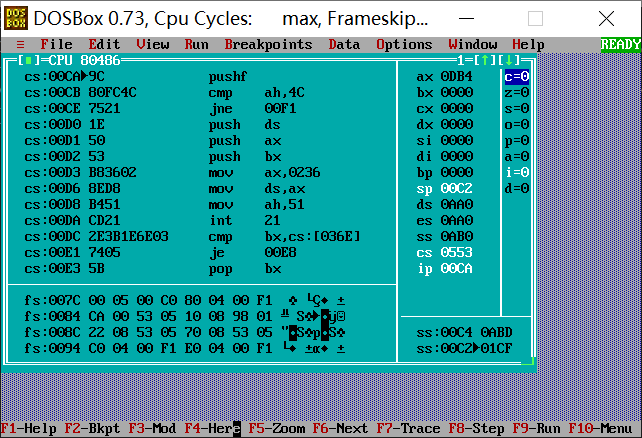


图4.6 21号中断观察结果

观察21号中断的处理代码，结果如图4.3.3所示。

3.读取CMOS中某个单元内容

利用CMOS的70h和71h两个端口和IO指令读取时钟内容，首先用OUT指令和70h端口设置数据地址，读取“时”单元，使用延时指令稳定信息，随后利用71h端口和IN指令读取信息，再将BCD码转换为对应真值，用同样的方式获取“分”“秒”信息。

4.TD的调试方法和关键的调试记录

（1）使用的调试功能：

F2添加断点，F8单步执行，F9运行至断点，goto进行跳转查看等。

（2）关键的调试记录：

观察输入的行列数是否被正确存储、中断程序是否被正确安装，检验是否被重复安装。

## 4.4任务4.2实验过程

### 4.4.1实验方法说明

加密处理方法

1. 密码和进货价采用异或某个字符的方式进行算术加密。
2. 将正确的进货价存放在某个地址中，当登陆成功后对那些地址进行解密，内容拷贝到商品结构，实现一定的混淆功能。

反跟踪处理方法

1. 动态修改执行代码，静态代码部分将密码异或一个错误的字符，动态修改执行代码后，才将密码异或正确的字符
2. 计时功能防止人工调试，在判断密码是否正确的代码区域添加计时，若此区域代码执行时间过长，就退出程序
3. 设置一些地址表，JMP语句跳转到对应变量储存的地址。

### 4.4.2实验记录与分析

1. **准备实验环境，建立工程**
2. **加密方式：**

加密登陆密码，将正确的代码异或’x’

BPASS DB '1' xor 'x','4' xor 'x','8' xor 'x','7' xor 'x','3'xor 'x', 0 ;密码

将正确的进货价异或’B’

GA1 GOODS < 'PEN',(15 XOR 'B')+4,20,70,25,0 > ;商品1 名称, 进货价、销售价、进货数量、已售数量,利润率（尚未计算）

GA2 GOODS < 'PENCIL',(2 XOR 'B')+4,3,100,50,0 >

GA3 GOODS < 'BOOK',(30 XOR 'B')+4,40,25,5,0 >

GA4 GOODS < 'RULER',3,4,200,150,0 >

GAN GOODS 6 DUP(<>)

GATEMP GOODS<>

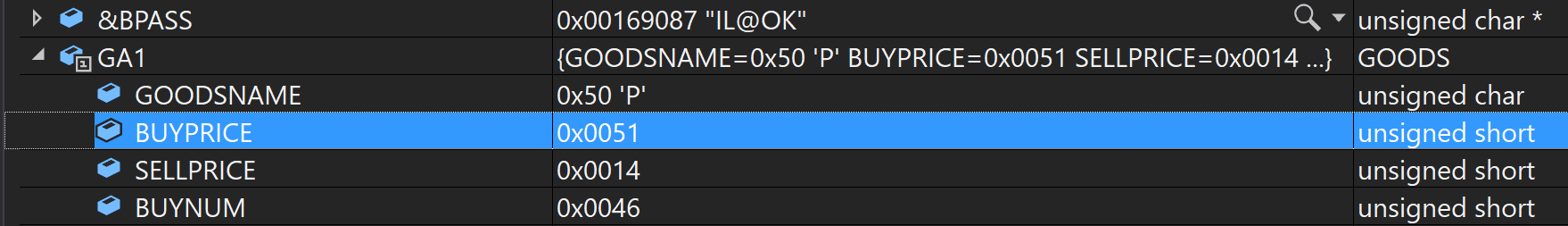


图4.7 加密后的密码和进货价

同时还额外设置了变量

offerprice1 DW 15 XOR 'A'

offerprice2 DW 2 XOR 'A'

offerprice3 DW 30 XOR 'A'

当登陆成功后，将这些变量异或’A’复原，并将其值赋给各商品进货价，起到一定的混淆作用

1. **反跟踪**

**动态修改代码**

还原密码应该异或’x’，但是我在登陆时对输入的密码不异或’x’，而是异或’D’，采用动态修改代码的方式将这句代码的机器码转换为异或’x’

.DATA

machine\_code db 80H,37H,78H

len=$-machine\_code

oldprotect dd ?

.CODE

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:

XOR byte ptr [EDI],'D'

如图：反汇编下代码显示为XOR byte ptr [EDI],'D'，但是实际上机器码 80 37 78对应的语句为XOR byte ptr [EDI],'x'

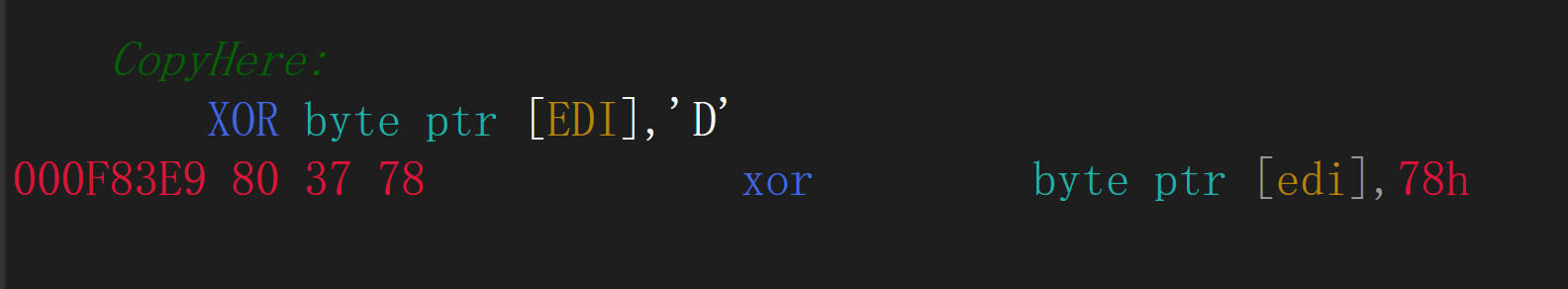
****

图4.8 动态修改代码

**时钟法反跟踪**

在判断输入密码与正确密码是否相同的代码段中添加时钟，通过这段代码执行时间的长短判断是否有人在调试，时间过长就直接退出程序

CALL timeGetTime

LEA EBX,BEGINTIME

MOV [EBX],EAX

mov EDX,0

lea EDI,INPWD

jiamimima:

CopyHere:

XOR byte ptr [EDI],'D'

inc EDI

inc EDX

cmp EDX,5

jne jiamimima

push offset INPWD

push offset BPASS

invoke strcmp,offset INPWD,offset BPASS

cmp eax,0

jne WRONG\_PWD

CALL timeGetTime

LEA EBX,ENDTIME

MOV [EBX],EAX

LEA ECX,BEGINTIME

SUB EAX,[ECX]

CMP EAX,10

JA OVER

JMP LOGIN\_S

**地址表法反跟踪**

Pmenu dword MENU

pstrcmp\_start dword strcmp\_start

pstrcmp\_exit dword strcmp\_exit

pENDGOODSSELL dword ENDGOODSSELL

将标号值（地址）存入内存变量中，JMP时不直接使用标号而是使用变量

## 4.5任务4.3实验过程

### 4.5.1实验方法说明

1. 准备实验环境

2. 使用IDA PRO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

### 4.5.2实验记录与分析

关于密码的加密

1.使用IDA PRO打开目标程序，定位到登陆程序的位置

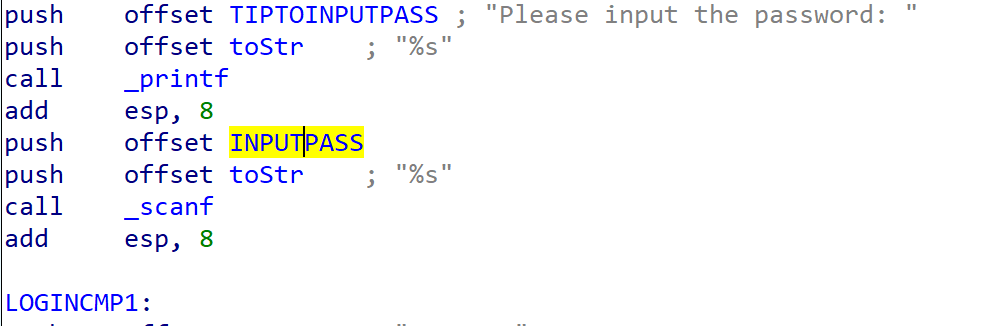


图4.9

这段程序是其输入密码的位置，我们由此输入密码

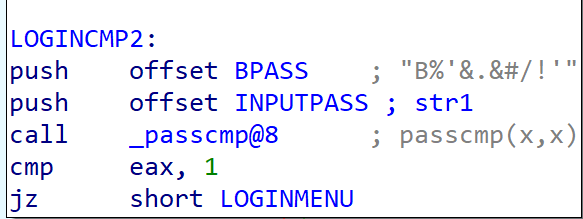


图4.10

输入密码后就要调用字符串比较函数passcmp，直接将正确密码的地址与输入的密码地址传入该函数。

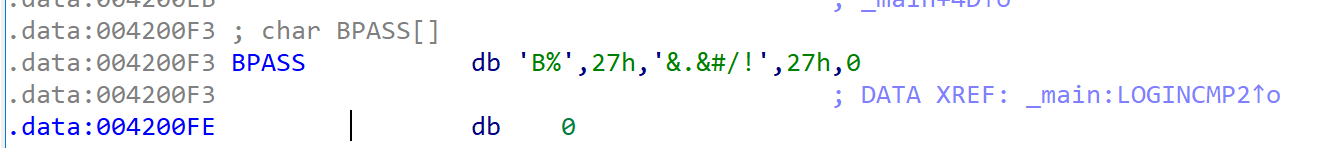


图4.11

数据段中找到该正确密码的存放地址，观察数值，显然这个密码是被加密过的



图4.12

分析代码，EDX保存INPASS的地址，EDI保存BPASS的地址，EBX初始值为0，判断当两个字符串的某个位的字符都是0的时候，字符串结束且相等，一个为0一个不为0，则不正确

取输入的字符串的某个字符，

xor ch, 15h

        xor cl, 17h

        xor ch, 16h

实际上相当于对该字符XOR 17h，ch起到了混淆作用

异或17h后，应该与BPASS的字符对应相等。

得到我们应该输入的字符串为U201914860

关于进货价的加密

跳转到功能1，功能1中应该有对进货价的解密，经过一番查找后无法找到

考虑功能2和功能3都不需要用到进货价，因此查找功能4

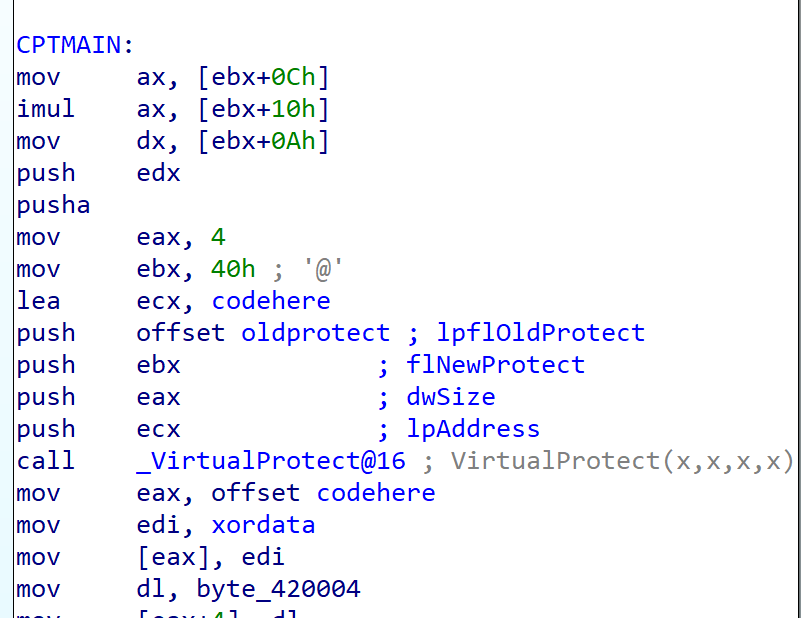


图4.13

发现功能4不太一样，调用了VirtualProtect，应该是动态修改代码，因此找到了方向

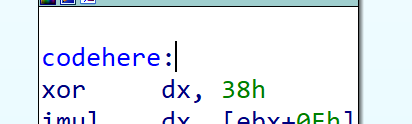


图4.14

表面上的代码时xor dx,38h，但实际上并不是异或38h，将此代码替换为xordata存储的机器指令对应的代码。



图4.15

判断此代码的指令是XOR 93H，因此只需要对每个商品的进货价XOR 93H就能得到正确的进货价

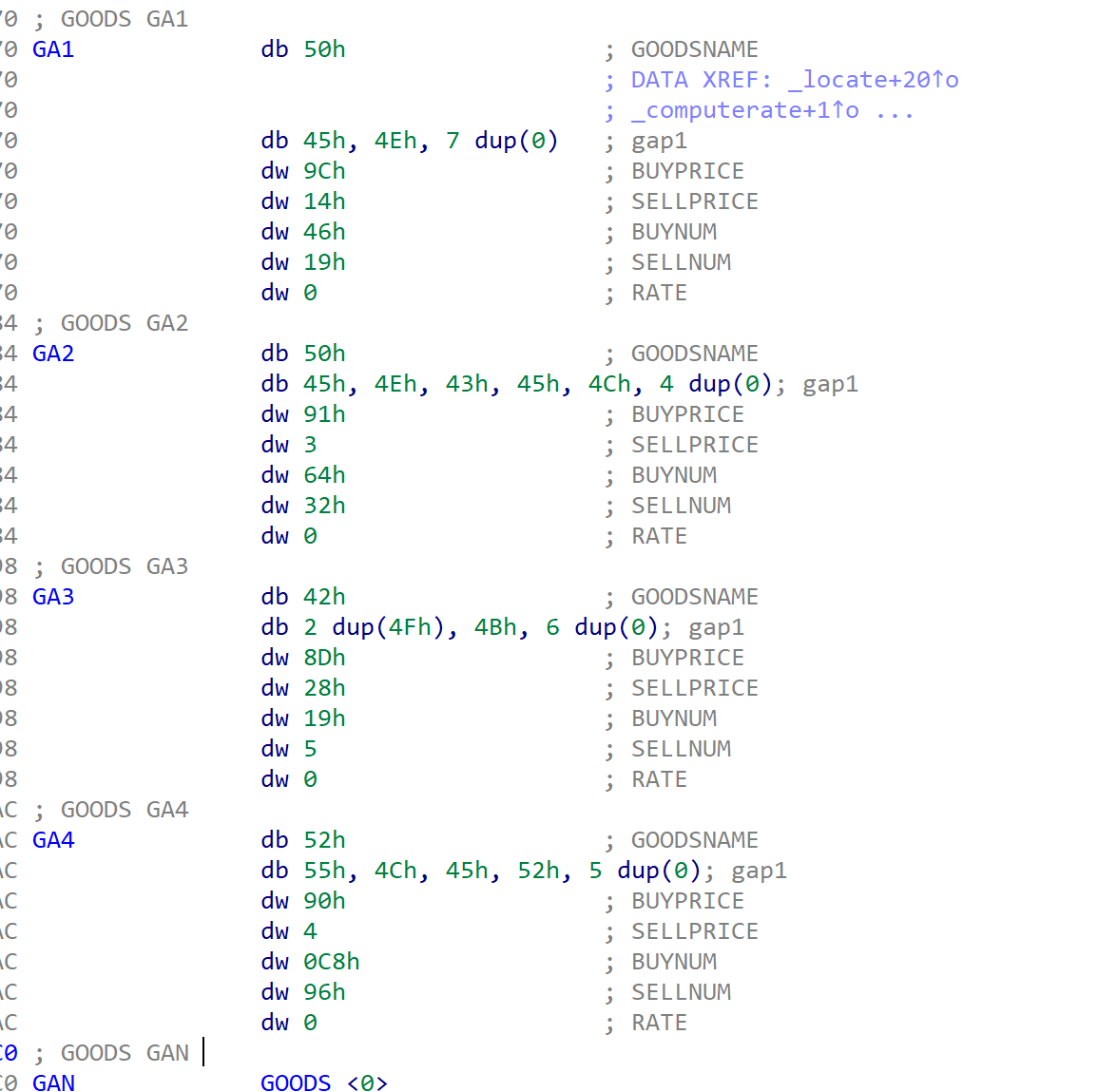


图4.16

因此几个商品的进货价分别为15，2，30，，3

## 4.6实验小结

### 4.6.1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。

# 5 16/32/64位编程比较

## 5.1实验目的与要求

（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；

1. 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。

## 5.2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。

**提示：**

1. 商品数只需定义5种左右。

2. 点菜单项List Sort时，先对所有商品按照利润率从高到低排序，然后按照排序结果在窗口中列出所有商品的各项信息（每种商品的信息占一行）。具体显示格式自行定义，不要求用中文。

3.根据消息驱动的特点，掌握基于窗口的应用程序的调试方法。注意观察：在点击菜单项时，会产生什么消息；控制台和窗口风格下的信息输出方法有什么差别。

**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）。

## 5.3任务5.1实验过程

### 5.3.1实验方法说明

1.从群文件中下载需要的inc文件等

2.使用VS2019建立窗口应用程序，添加源文件。

3.编写实验要求的窗口，将部分3.1的程序移植

### 5.3.2实验记录与分析

1.使用VS2019建立一个Windows桌面应用程序

2.建立asm汇编源文件，同时将需要使用的lib和inc等添加到当前目录下

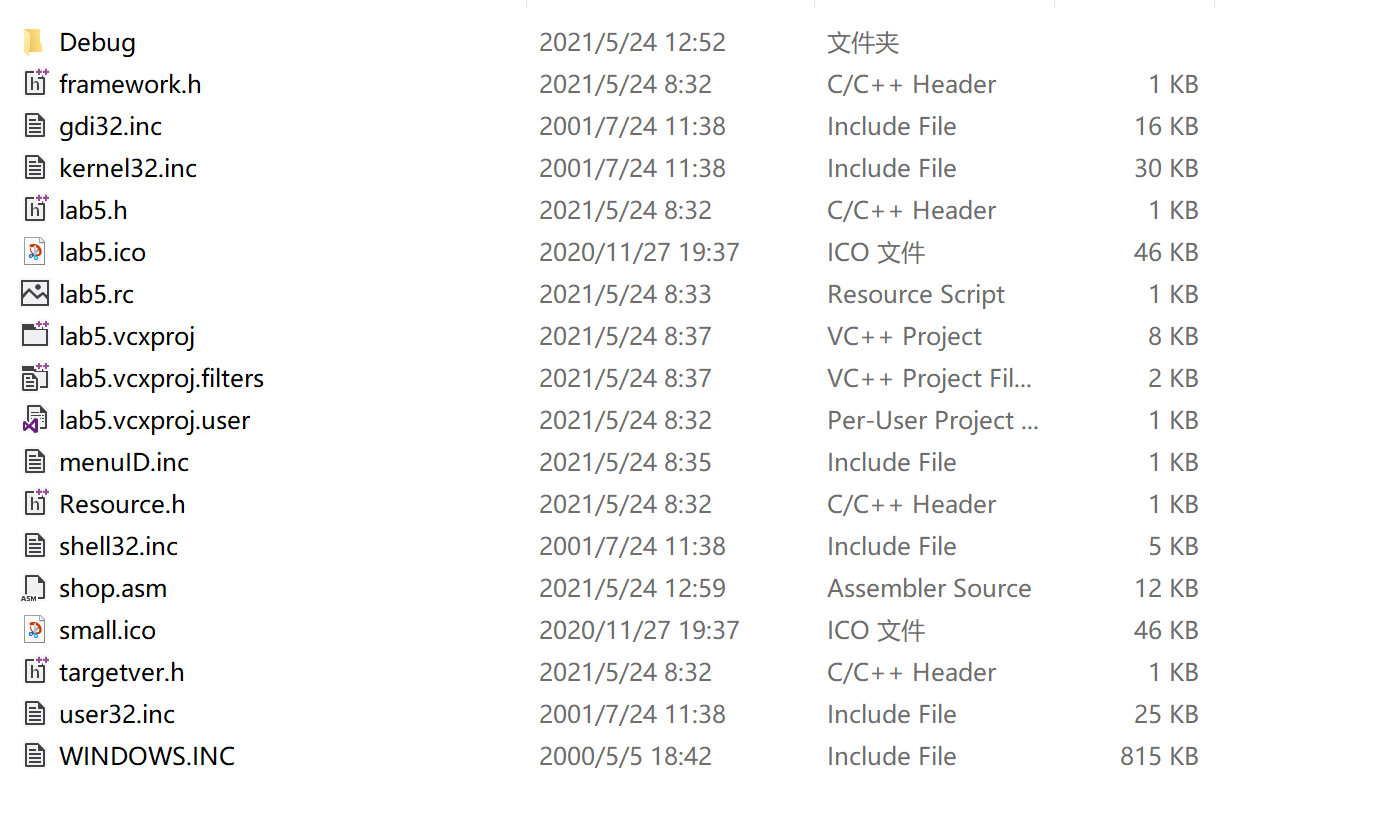


图5.1 窗口win32程序所需要的文件

3. 更改menu.rc文件，更改后的内容如下：

#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 "C&ompute Rate", IDM\_ACTION\_RECO

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

END

POPUP "&Help"

BEGIN

MENUITEM "&About", IDM\_HELP\_ABOUT

END

END

更改menuID.inc文件，更改后的内容如下：

IDM\_FILE\_EXIT equ 10001

IDM\_ACTION\_RECO equ 10002

IDM\_ACTION\_LIST equ 10003

IDM\_HELP\_ABOUT equ 10101

4. 在默认的窗口应用程序代码的基础上，进行补充与修改，要求具有如下功能：

在点击File->Exit时会退出程序，点击Action->Compute Rate时弹出利润率计算完毕提示框并计算推荐度，点击Action->List Sort会将商品按照利润率由大到小排序，并打印到窗口中。点击Help->About弹出学生自我介绍窗口。

.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

CalculateRate proto

SORT proto

include menuID.INC

include windows.inc

include user32.inc

include kernel32.inc

include gdi32.inc

include shell32.inc

includelib user32.lib

includelib kernel32.lib

includelib gdi32.lib

includelib shell32.lib

includelib ucrt.lib

includelib legacy\_stdio\_definitions.lib

includelib Winmm.lib

GOODS STRUCT

GOODSNAME db 10 DUP(0)

BUYPRICE DW 0

SELLPRICE DW 0

BUYNUM DW 0

SELLNUM DW 0

RATE DW 0

NAMELENGTH dw 0

BUYPRICELENGTH dw 0

SELLPRICELENGTH DW 0

BUYNUMLENGTH DW 0

SELLNUMLENGTH DW 0

RATELENGTH DW 0

GOODS ENDS

.data

ClassName db 'TryWinClass',0

AppName db 'Our First Window',0

MenuName db 'MyMenu',0

DlGOODSSNAME db 'MyDialog',0

AboutMsg db 'I am CE1901 LiuYiKang',0

RecoMsg db 'Calculate completed', 0

hInstance dd 0

CommandLine dd 0

msg\_name db 'name',0

msg\_off db 'Off', 0

msg\_BUYPRICE db 'BuyPrice',0

msg\_SELLPRICE db 'SellPrice',0

msg\_BUYNUM db 'BuyAmount',0

msg\_SELLNUM db 'SellAmount',0

msg\_RATE db 'RATE',0

menuItem db 0

AMOUNT Dword 4

GA1 GOODS < 'PEN',15,20,70,25,0,3,2,2,2,2,3 >

GA2 GOODS < 'PENCIL',2,3,100,50,0,6,1,1,3,2,3 >

GA3 GOODS < 'BOOK',30,40,25,5,0,4,2,2,2,1,3 >

GA4 GOODS < 'RULER',3,4,200,150,0,5,1,1,3,3,1 >

SORThelp STRUCT

POS DWORD 0

RATE DW 0

SORThelp ENDS

SORT1 SORThelp <1,0>

SORT2 SORThelp <2,0>

SORT3 SORThelp <3,0>

SORT4 SORThelp <4,0>

lpFmt db "%hd", 0

NAME dword 0

BUYPRICE DB 5 DUP(0)

SELLPRICE DB 5 DUP(0)

BUYNUM DB 5 DUP(0)

SELLNUM DB 5 DUP(0)

RATE DB 5 DUP(0)

NAMELENGTH DW 0

.code

Start: invoke GetModuleHandle,NULL

mov hInstance,eax

invoke GetCommandLine

mov CommandLine,eax

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

invoke ExitProcess,eax

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

LOCAL wc:WNDCLASSEX

LOCAL msg:MSG

LOCAL hWnd:HWND

invoke RtlZeroMemory,addr wc,sizeof wc

mov wc.cbSize,SIZEOF WNDCLASSEX

;

;CS\_HEADRAW 移动或调整更改了工作区的宽度，则重绘整个窗口

;CS\_VREDRAW 高度

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 CalculateRate

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

.ELSEIF wParam == IDM\_ACTION\_LIST

mov menuItem, 1

invoke InvalidateRect,hWnd,0,1

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

Xpos equ 10

Ypos equ 10

invoke SORT

invoke TextOut,hdc,Xpos+0\*100,Ypos+0\*30,offset msg\_name,4

invoke TextOut,hdc,Xpos+1\*100,Ypos+0\*30,offset msg\_BUYPRICE,8

invoke TextOut,hdc,Xpos+2\*100,Ypos+0\*30,offset msg\_SELLPRICE,9

invoke TextOut,hdc,Xpos+3\*100,Ypos+0\*30,offset msg\_BUYNUM,9

invoke TextOut,hdc,Xpos+4\*100,Ypos+0\*30,offset msg\_SELLNUM,10

invoke TextOut,hdc,Xpos+5\*100,Ypos+0\*30,offset msg\_RATE,4

LEA ECX,SORT1

MOV EDI, dword ptr[ECX]

invoke sprintf,OFFSET BUYPRICE,offset lpFmt,(GOODS ptr [EDI]).BUYPRICE

invoke sprintf,OFFSET SELLPRICE,offset lpFmt,(GOODS ptr [EDI]).SELLPRICE

invoke sprintf,OFFSET BUYNUM,offset lpFmt,(GOODS ptr [EDI]).BUYNUM

invoke sprintf,OFFSET SELLNUM,offset lpFmt,(GOODS ptr [EDI]).SELLNUM

invoke sprintf,OFFSET RATE,offset lpFmt,(GOODS ptr [EDI]).RATE

invoke TextOut,hdc,Xpos+0\*100,Ypos+1\*30,SORT1.POS , (GOODS ptr [EDI]).NAMELENGTH

invoke TextOut,hdc,Xpos+1\*100,Ypos+1\*30,offset BUYPRICE, (GOODS ptr [EDI]).BUYPRICELENGTH

invoke TextOut,hdc,Xpos+2\*100,Ypos+1\*30,offset SELLPRICE,(GOODS ptr [EDI]).SELLPRICELENGTH

invoke TextOut,hdc,Xpos+3\*100,Ypos+1\*30,offset BUYNUM, (GOODS ptr [EDI]).BUYNUMLENGTH

invoke TextOut,hdc,Xpos+4\*100,Ypos+1\*30,offset SELLNUM,(GOODS ptr [EDI]).SELLNUMLENGTH

invoke TextOut,hdc,Xpos+5\*100,Ypos+1\*30,offset RATE, (GOODS ptr [EDI]).RATELENGTH

LEA ECX,SORT2

MOV EDI, dword ptr[ECX]

invoke sprintf,OFFSET BUYPRICE,offset lpFmt,(GOODS ptr [EDI]).BUYPRICE

invoke sprintf,OFFSET SELLPRICE,offset lpFmt,(GOODS ptr [EDI]).SELLPRICE

invoke sprintf,OFFSET BUYNUM,offset lpFmt,(GOODS ptr [EDI]).BUYNUM

invoke sprintf,OFFSET SELLNUM,offset lpFmt,(GOODS ptr [EDI]).SELLNUM

invoke sprintf,OFFSET RATE,offset lpFmt,(GOODS ptr [EDI]).RATE

invoke TextOut,hdc,Xpos+0\*100,Ypos+2\*30,SORT2.POS , (GOODS ptr [EDI]).NAMELENGTH

invoke TextOut,hdc,Xpos+1\*100,Ypos+2\*30,offset BUYPRICE, (GOODS ptr [EDI]).BUYPRICELENGTH

invoke TextOut,hdc,Xpos+2\*100,Ypos+2\*30,offset SELLPRICE,(GOODS ptr [EDI]).SELLPRICELENGTH

invoke TextOut,hdc,Xpos+3\*100,Ypos+2\*30,offset BUYNUM, (GOODS ptr [EDI]).BUYNUMLENGTH

invoke TextOut,hdc,Xpos+4\*100,Ypos+2\*30,offset SELLNUM,(GOODS ptr [EDI]).SELLNUMLENGTH

invoke TextOut,hdc,Xpos+5\*100,Ypos+2\*30,offset RATE, (GOODS ptr [EDI]).RATELENGTH

LEA ECX,SORT3

MOV EDI, dword ptr[ECX]

invoke sprintf,OFFSET BUYPRICE,offset lpFmt,(GOODS ptr [EDI]).BUYPRICE

invoke sprintf,OFFSET SELLPRICE,offset lpFmt,(GOODS ptr [EDI]).SELLPRICE

invoke sprintf,OFFSET BUYNUM,offset lpFmt,(GOODS ptr [EDI]).BUYNUM

invoke sprintf,OFFSET SELLNUM,offset lpFmt,(GOODS ptr [EDI]).SELLNUM

invoke sprintf,OFFSET RATE,offset lpFmt,(GOODS ptr [EDI]).RATE

invoke TextOut,hdc,Xpos+0\*100,Ypos+3\*30,SORT3.POS , (GOODS ptr [EDI]).NAMELENGTH

invoke TextOut,hdc,Xpos+1\*100,Ypos+3\*30,offset BUYPRICE, (GOODS ptr [EDI]).BUYPRICELENGTH

invoke TextOut,hdc,Xpos+2\*100,Ypos+3\*30,offset SELLPRICE,(GOODS ptr [EDI]).SELLPRICELENGTH

invoke TextOut,hdc,Xpos+3\*100,Ypos+3\*30,offset BUYNUM, (GOODS ptr [EDI]).BUYNUMLENGTH

invoke TextOut,hdc,Xpos+4\*100,Ypos+3\*30,offset SELLNUM,(GOODS ptr [EDI]).SELLNUMLENGTH

invoke TextOut,hdc,Xpos+5\*100,Ypos+3\*30,offset RATE, (GOODS ptr [EDI]).RATELENGTH

LEA ECX,SORT4

MOV EDI, dword ptr[ECX]

invoke sprintf,OFFSET BUYPRICE,offset lpFmt,(GOODS ptr [EDI]).BUYPRICE

invoke sprintf,OFFSET SELLPRICE,offset lpFmt,(GOODS ptr [EDI]).SELLPRICE

invoke sprintf,OFFSET BUYNUM,offset lpFmt,(GOODS ptr [EDI]).BUYNUM

invoke sprintf,OFFSET SELLNUM,offset lpFmt,(GOODS ptr [EDI]).SELLNUM

invoke sprintf,OFFSET RATE,offset lpFmt,(GOODS ptr [EDI]).RATE

invoke TextOut,hdc,Xpos+0\*100,Ypos+4\*30,SORT4.POS , (GOODS ptr [EDI]).NAMELENGTH

invoke TextOut,hdc,Xpos+1\*100,Ypos+4\*30,offset BUYPRICE, (GOODS ptr [EDI]).BUYPRICELENGTH

invoke TextOut,hdc,Xpos+2\*100,Ypos+4\*30,offset SELLPRICE,(GOODS ptr [EDI]).SELLPRICELENGTH

invoke TextOut,hdc,Xpos+3\*100,Ypos+4\*30,offset BUYNUM, (GOODS ptr [EDI]).BUYNUMLENGTH

invoke TextOut,hdc,Xpos+4\*100,Ypos+4\*30,offset SELLNUM,(GOODS ptr [EDI]).SELLNUMLENGTH

invoke TextOut,hdc,Xpos+5\*100,Ypos+4\*30,offset RATE, (GOODS ptr [EDI]).RATELENGTH

;;

ret

Display endp

CalculateRate proc

push ebp

mov ebp,esp

MOV ESI, -1

push ESI

ALLGOOD:

pop ESI

INC ESI

push ESI

CMP ESI, AMOUNT

JE ENDCalculate

IMUL ESI, 32

ADD ESI, OFFSET GA1

MOV EAX,0

MOV EBX,0

MOV AX,(GOODS ptr[ESI]).BUYPRICE

MOV BX,(GOODS ptr[ESI]).BUYNUM

IMUL BX

PUSH AX

MOV AX,(GOODS ptr[ESI]).SELLPRICE

MOV BX,(GOODS ptr[ESI]).SELLNUM

IMUL BX

PUSH AX

MOV EAX,0

MOV EBX,0

pop AX

POP BX

SUB EAX,EBX

IMUL EAX,EAX,100

CDQ

idiv EBX

MOV (GOODS ptr[ESI]).RATE,AX

jmp ALLGOOD

ENDCalculate:

pop ESI

pop ebp

ret

CalculateRate endp

SORT proc

push ebp

mov ebp,esp

MOV EAX, -1

SORTKEEP:

INC EAX

CMP EAX,AMOUNT

JE FUNC5KEEP

MOV EBX,EAX

IMUL EBX,6

MOV ECX,EAX

IMUL ECX,32

ADD ECX,offset GA1

ADD EBX,offset SORT1

MOV dx,[ECX+18]

MOV [EBX+4],dx

MOV [EBX],ECX

JMP SORTKEEP

FUNC5KEEP:

MOV AX,0

BEGINLOOP1:

inc AX

CMP AX,word ptr[AMOUNT]

Jnl ENDLOOP1

dec AX

MOV BX,AX

MOV CX,AX

INC CX

BEGINLOOP2:

CMP CX,word ptr[AMOUNT]

jnl ENDLOOP2

MOV EDI,0

MOV ESI,0

MOV SI,CX

MOV DI ,BX

IMUL SI,6

IMUL DI,6

ADD SI,4

ADD DI,4

MOV DX,word ptr SORT1[DI]

CMP word ptr SORT1[SI],DX

jng GO

MOV BX,CX

GO:

MOV EDI,0

MOV ESI,0

MOV SI,AX

MOV DI ,BX

IMUL SI,6

IMUL DI,6

MOV EDX,dword ptr[SORT1[SI]]

push EDX

MOV EDX,dword ptr[SORT1[DI]]

MOV dword ptr[SORT1[SI]],EDX

pop EDX

MOV dword ptr[SORT1[DI]],EDX

ADD SI,4

ADD DI,4

MOV DX,word ptr[SORT1[SI]]

push DX

MOV DX,word ptr[SORT1[DI]]

MOV word ptr[SORT1[SI]],DX

pop DX

MOV word ptr[SORT1[DI]],DX

INC CX

JMP BEGINLOOP2

ENDLOOP2:

INC AX

JMP BEGINLOOP1

ENDLOOP1:

pop ebp

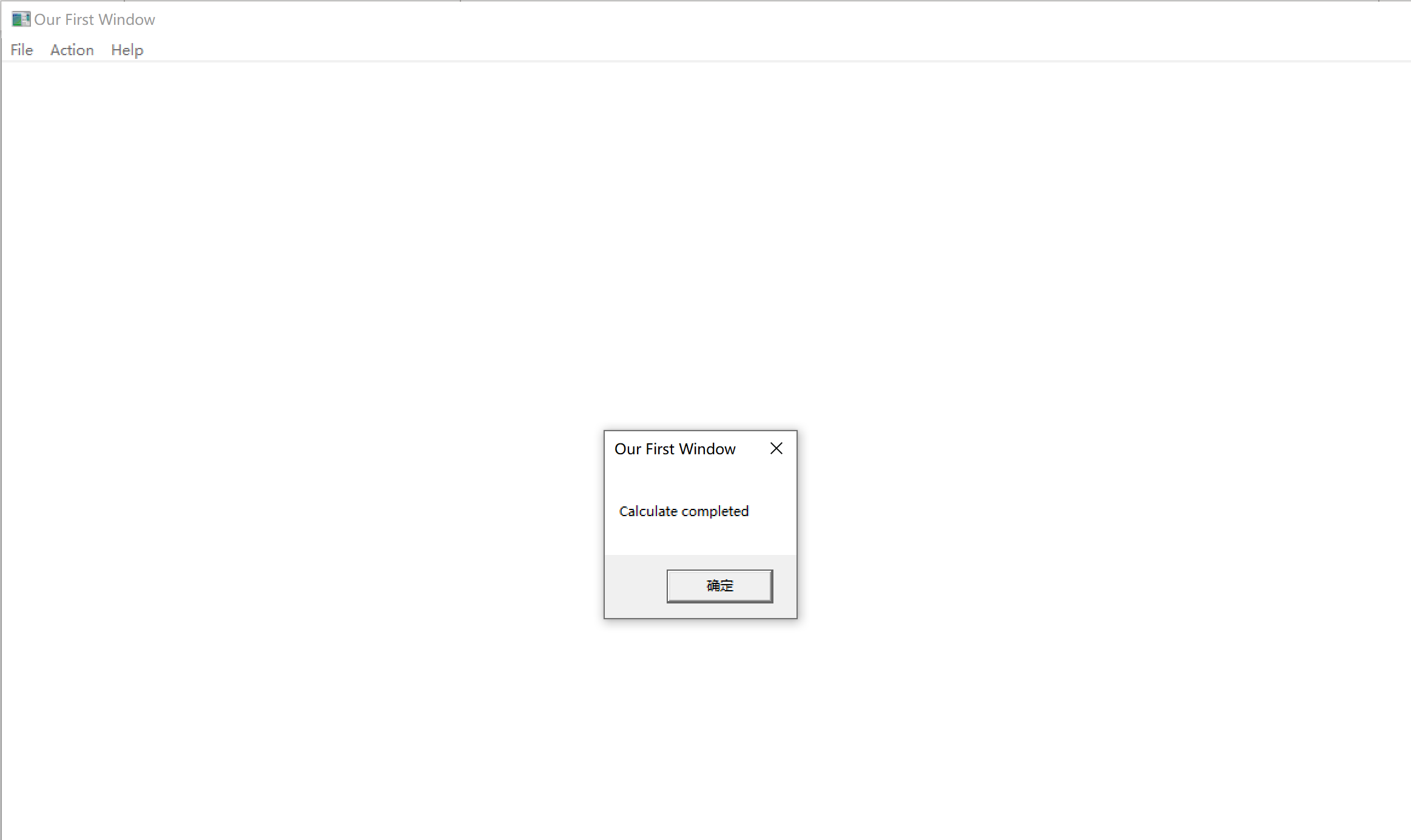
ret

SORT endp

end Start

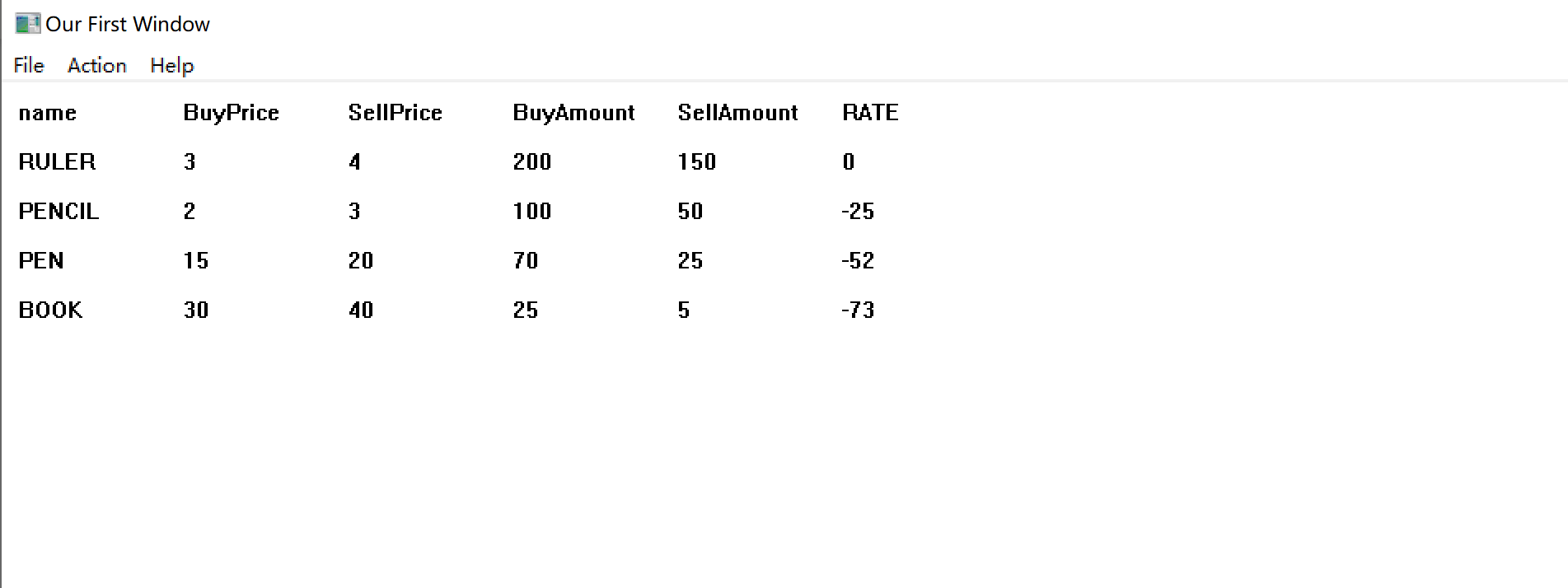
5. 测试程序功能

测试利润率计算功能



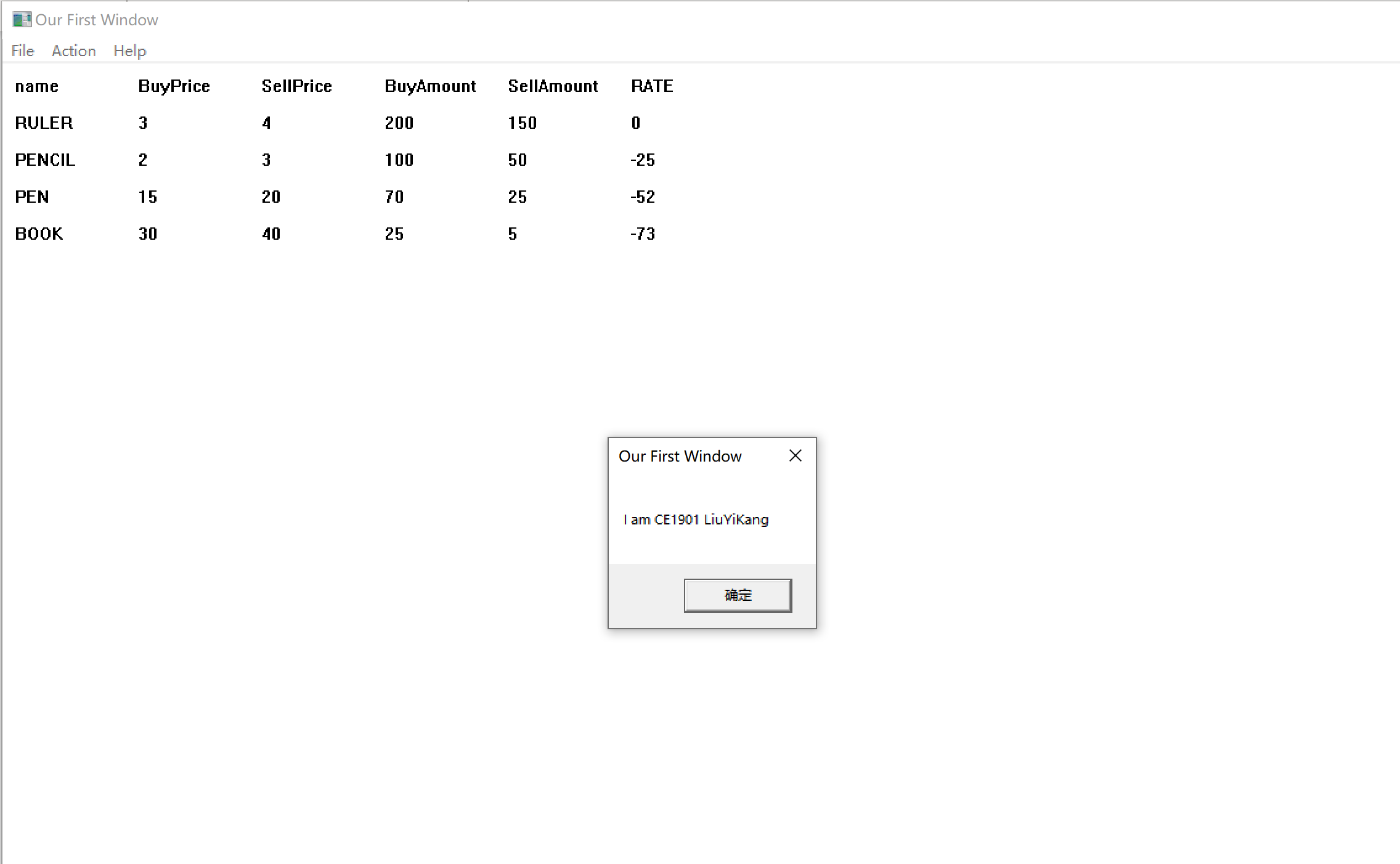
**图5.2 点击Compute Rate弹出计算完毕消息框**

测试排序功能，按照推荐度从高到低排序输出信息



**图5.3 点击List Sort排序并打印**

测试About，正常弹出提示框



**图5.4 点击About弹出消息框**

测试完毕

实验中遇到了一些问题，其中最难解决的不是修改程序框架，而是输出的一些方面 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，因此我使用了sprintf函数，单独定义了一些变量存放转换后的字符串，此外我还需要得知要输出各数据的长度，这个非常难操作，尝试了许多方法没有成功，最后在结构中单独设置了变量存放各商品对应的输出的长度。

## 5.4任务5.2实验过程

### 5.4.1实验方法说明

1. 阅读实验文件种六十四位的asm文件代码，体会64，32，16位程序的异同。

2. 通过阅读课本和上网查阅，学习其区别。

3. 查阅相关教程，使用VS建立64位的汇编工程。

4. 新建VS项目，在项目右键打开-生成依赖项-生成自定义，勾选.masm。

5. 项目-添加现有项，将64.asm文件导入源文件中。

6. 修改项目属性配置：项目-属性-配置属性-链接器下。

a) 调试-生成调试信息改为“是(/Debug)”。

b) 系统-子系统改为“控制台”。

c) 高级-入口点改为“start”。

7. 右键资源管理器中64.asm，选择属性-常规：。

a) 设置生成中排除为否。

b) 选择项类型为自定义生成工具。

8. VS菜单栏：生成-配置管理器-Win32-选择平台为X64，确定。

9. 在步骤6中的属性-生成自定义工具-常规设置

a) 命令行ml64 /c %(fileName).asm

b) 输出%(fileName).obj;%(Outputs)

c) 链接对象：是

10. 运行调试程序

### 5.4.2 实验记录与分析

1.通过观察源程序，可以观察到：

a)程序开头，64位的程序不需要声明.386指令集与.model

b)在语法格式上，64位与16、32位汇编程序差异不大

c)寄存器的名称有改变

2.通过查阅相关的书籍资料，得到如下的总结

a) 64位增加了新的汇编指令，如movq、pushq、popq

b) 64位下有16个通用寄存器，而32位下只有8个通用寄存器，且名称发生变化

c) 函数的参数传递方法发生变化：64位程序在传参与返回值时通过特定的6个寄存器，若参数过多，寄存器不够使用时则会采用栈来传参

4.根据上网查阅如何在VS上编写64位汇编程序修改配置

5.得到运行结果如图5.4.1所示

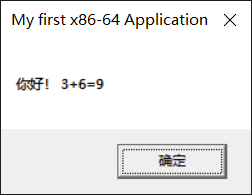


图5.5

## 5.5任务5.3实验过程

### 5.5.1实验任务

   查阅《ARM汇编技术简介》资料，体会其与所学的80X86体系的异同。

### 5.5.1实验结果

**段的定义**

相同点：分段，具有代码段、数据段等。且程序至少有一个代码段。

不同点： ARM下，使用 AREA段名 属性1，属性2，……定义一个代码段或数据段。

CODE属性：用于定义代码段，默认为READONLY

DATA属性：用于定义数据段，默认为READWRITE

READONLY属性：指定本段为只读

READWRITE属性：指定本段为可读可写

ALIGN属性：使用方式为ALIGN表达式。在默认时，ELF (可执行连接文件)的代码段和数据段是按字对齐的

COMMON属性：定义一个通用的段，不包含任何的用户代码和数据。各源文件中同名的COMMON段共享同一段存储单元

**CPU寄存器**

相同点：同时有通用寄存器以及标志寄存器

不同点：ARMv8具有更多的通用寄存器，具有31个64位寄存器，并且为了保持寄存器与32位下的通用性，每个寄存器可被当作2个32位寄存器来使用。

**指令**

相同点：操作码+操作数两部分组成

不同点：X86指令大多数为两个操作数，而ARM指令大部分是三个操作数；X86下只有跳转指令可以加条件码，而ARM指令大部分其后都可以加条件码。

**子程序调用的参数传递与返回**

相同点：都能用寄存器、堆栈传参，都能用寄存器返回值

不同点：X86下用寄存器传递是编写函数者制定一套传参通信规则，而ARM函数调用参数传递顺序是从r0-r3，当参数超过4个采用入栈出栈传参。X86下返回值保存至EAX中，而ARM返回值保存在r0中。

**与C语言混合编程**

相同点：都具有一定的寄存器使用规则、数据栈的使用规则和参数的传递规则。

不同点：规则细节不一致，可参考4子程序调用的参数传递与返回方法差异

## 5.6小结

第五次实验是最后一次实验，内容是关于16，32，64位编程的比较，实验一的内容为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。我首次接触了窗口应用程序的编程，知道了如何配置WIN32程序开发程序的配置，程序的开发流程以及Windows API的使用方法，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。使用VS编写窗口应用程序可以源码级调试，保留源码原本的结构，伪指令和变量名，调试的方便性大大提高，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。

学会了如何在VS2019下调试一个x64程序，观察与32位程序的不同之处，对汇编程序的体会加深。

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，体会了80X86体系的异同。对于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**等方面的认识得到了加深。

# 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