

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实验**

**实验名称： 编程基础**

**实验时间： 2018-4-8，18:30-21:50 实验地点： 南一楼804室16号实验台**

**指导教师： 李海波 专业班级：计算机科学与技术201607班**

**学 号： U201614700 姓 名： 王亚宁**

**同组学生： 无 报告日期： 2018年3月30日**

**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：

日期：

成绩评定

|  |  |  |
| --- | --- | --- |
| 实验完成质量得分（70分）（实验步骤清晰详细深入，实验记录真实完整等） | 报告撰写质量得分（30分）（报告规范、完整、通顺、详实等） | 总成绩（100分） |
|  |  |  |

指导教师签字：

                    日期：

**目录**

[实验目的与要求 1](#_Toc510121407)

[实验内容 1](#_Toc510121408)

[实验过程 1](#_Toc510121409)

[任务1 1](#_Toc510121410)

[1. 设计思想及存储单元分配 1](#_Toc510121411)

[2. 流程图 1](#_Toc510121412)

[3. 源程序 1](#_Toc510121413)

[4. 实验步骤 1](#_Toc510121414)

[5. 实验记录 1](#_Toc510121415)

[体会 1](#_Toc510121416)

[参考文献 1](#_Toc510121417)

# 实验目的与要求

1. 掌握汇编源程序编辑工具、汇编程序、连接程序、调试工具TD的使用；
2. 理解数、符号、寻址方式等在计算机内的表现形式；
3. 理解指令执行与标志位改变之间的关系；
4. 熟悉常用的DOS功能调用；
5. 熟悉分支、循环程序的结构及控制方法，掌握分支、循环程序的调试方法；
6. 加深对转移指令及一些常用的汇编指令的理解。

# 实验内容

任务1. 《80X86汇编语言程序设计》教材中 P31的 1.14题。

任务2. 《80X86汇编语言程序设计》教材中 P45的 2.3题。

任务3. 《80X86汇编语言程序设计》教材中 P45的 2.4题的改写。

任务4：有一个老板在网上开了2个网店SHOP1,SHOP2；每个网店有n种商品销售，不同网店之间销售的商品种类相同，但数量和销售价格可以不同。每种商品的信息包括：商品名称（10个字节，名称不足部分补0），进货价(字类型)，销售价（字类型），进货总数（字类型），已售数量（字类型），利润率（%）【=（销售价\*已售数量-进货价\*进货总数）\*100/（进货价\*进货总数），字类型】。老板管理网店信息时需要输入自己的名字（10个字节，不足部分补0）和密码（6个字节，不足部分补0），登录后可查看商品的全部信息；顾客（无需登录）可以查看所有网店中每个商品除了进货价、利润率以外的信息。

# 实验过程

## 任务1

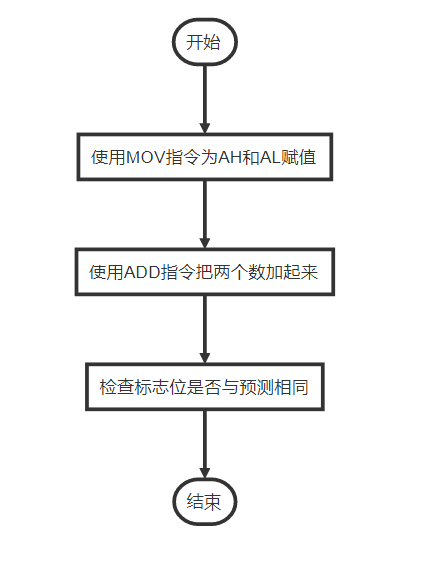
任务详细内容：已知二进制数x1和x2的值，求x1和x2的补码的和，并指出符号，判断是否发生了溢出和进位。

### 设计思想

设计思路：任务一的内容是对两个数进行加减运算，指出结果的符号，可以使用AX寄存器的高低位存放两个数的值，运算即可。

寄存器分配：AH存放x1，AL存放x2。

### 流程图



### 实验步骤

1. 打开dosbox，将masm的等文件目录挂载为c，输入c:切换到c盘。
2. 输入td打开td的窗口。
3. 直接编写要执行的汇编代码：首先把数据x1和x2的补码使用MOV指令到AX寄存器的高位和低位。
4. 使用ADD AH, AL命令计算AL加AH并将结果存放在AH中。
5. 观察此时的标志位SF, OF, CF, ZF，记录下来和自己实验前预测的对比并分析原因。

### 实验记录

实验环境条件：i5 6200U，2.8GHz，8GB内存；dosbox命令行窗口；visual studio 2017；MASM.EXE 6.0； LINK.EXE 5.2; TD.EXE 5.0。

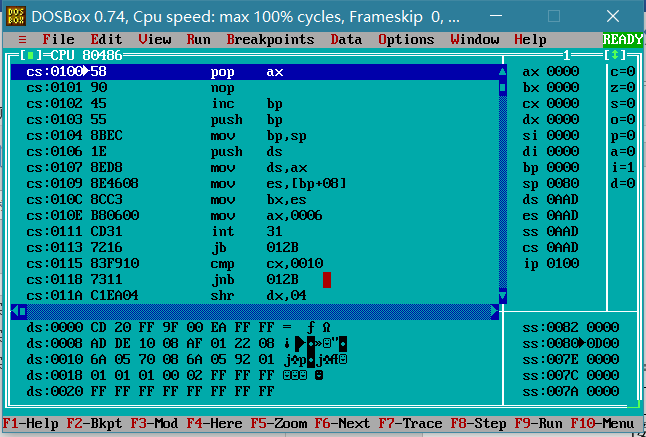
首先是实验使用的数据x1，x2，它们的补码表示，如表1-1所示：

先计算求和的标志位：

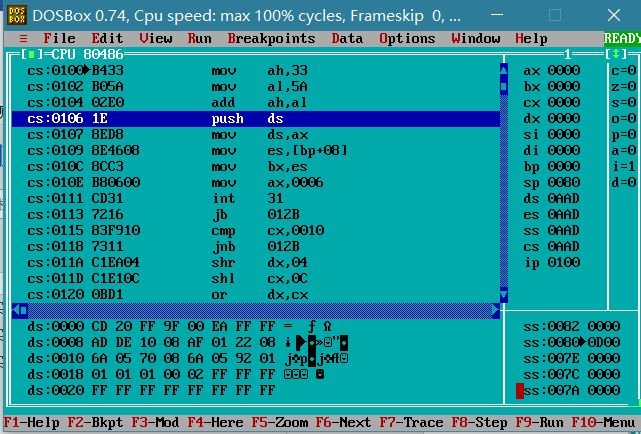
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X1 | X2 | 和 | SF | OF | CF | ZF |
| 33H | 5AH | 8DH | 1 | 1 | 0 | 0 |
| D7H | A3H | 7AH | 0 | 1 | 1 | 0 |
| 65H | A3H | 08H | 0 | 0 | 1 | 0 |

接下来测试在CPU上执行汇编代码的真实情况。

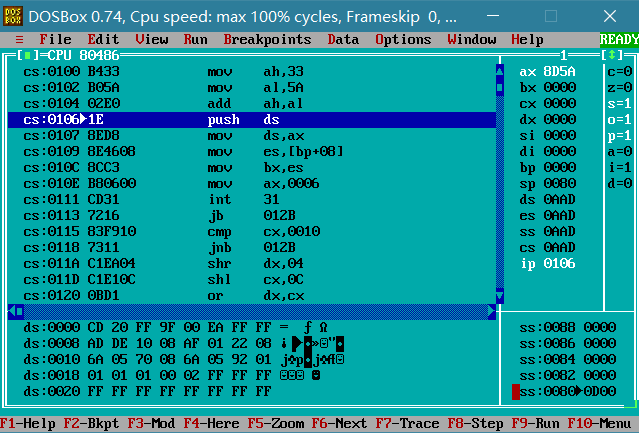
计算第一个表达式的值，先打开dosbox，挂载磁盘，切换磁盘，输入td打开程序窗口，界面如图所示：



此时直接将光标移到箭头所指处，直接输入指令MOV AH, 33H和MOV AL, 5AH，输入指令如图所示：

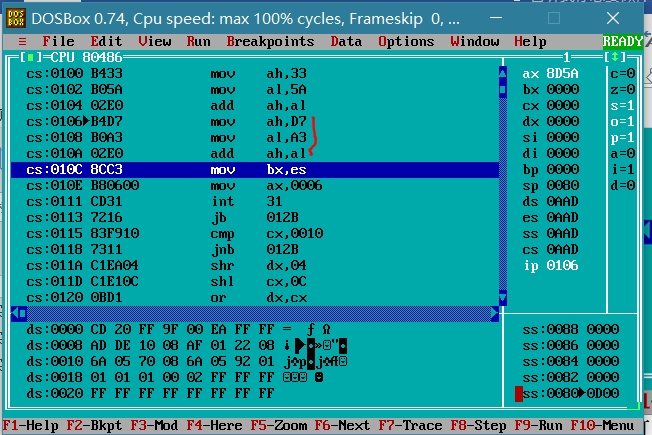


按F8单步运行指令，记录四个指示标志位，结果如图所示：

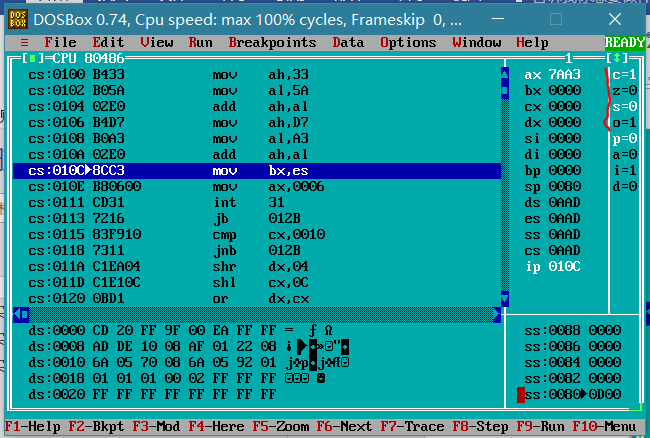


可见SF为1，表示结果是负数，ZF为0表示结果不为0，OF为1表示发生了溢出，CF为0表示没有发生进位或者借位，与预测相同。

测试第二个表达式，指令如图所示：

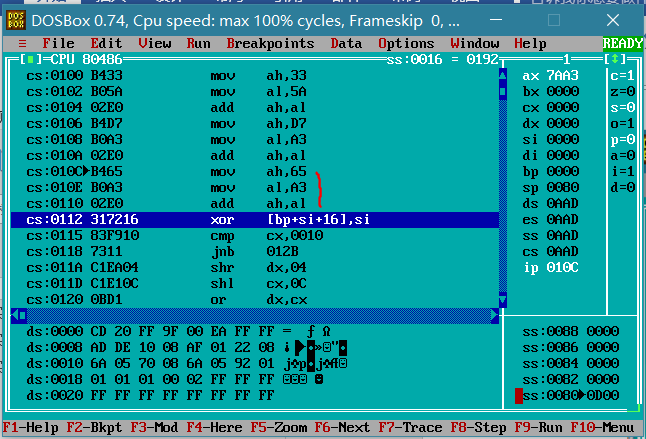


运行指令如图所示：

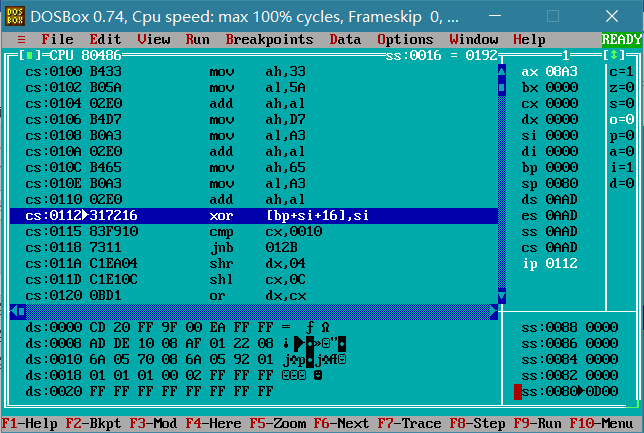


可见CF为1表示产生了进位，OF表示发生了溢出，ZF为0表示结果不为0，SF为0表示结果为整数，与预测结果相同。

输入第三个表达式的指令，如图所示：



运行指令如图所示：

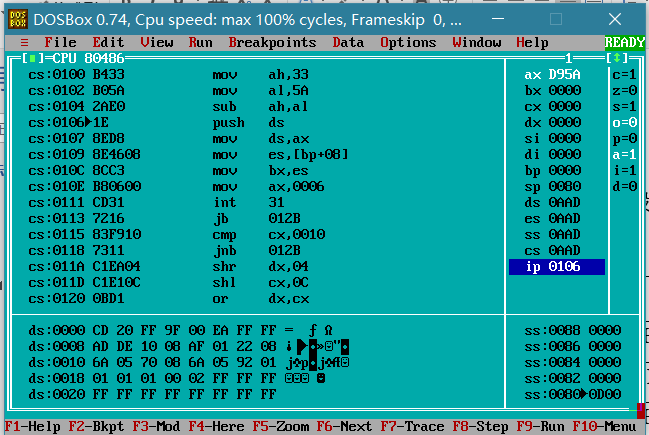


CF为1表示发生了进位，ZF为0表示结果不为0，SF为0表示结果为正数，OF为0表示运算没有发生溢出，预测正确。

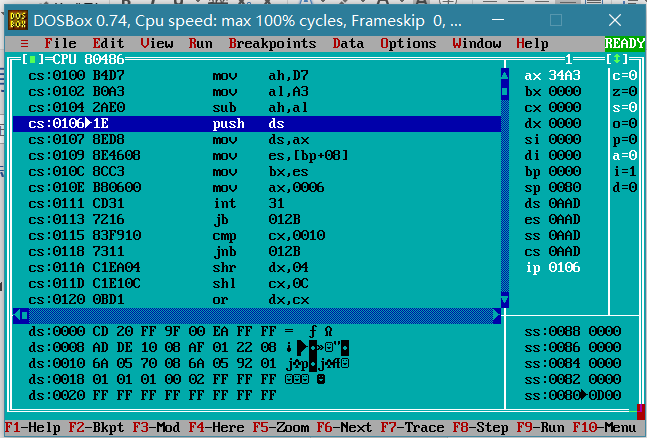
在进行减法运算时结果的预测：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X1 | X2 | 差 | SF | OF | CF | ZF |
| 33H | 5AH | D9H | 1 | 0 | 1 | 0 |
| D7H | A3H | 34H | 0 | 0 | 0 | 0 |
| 65H | A3H | C2H | 1 | 1 | 1 | 0 |

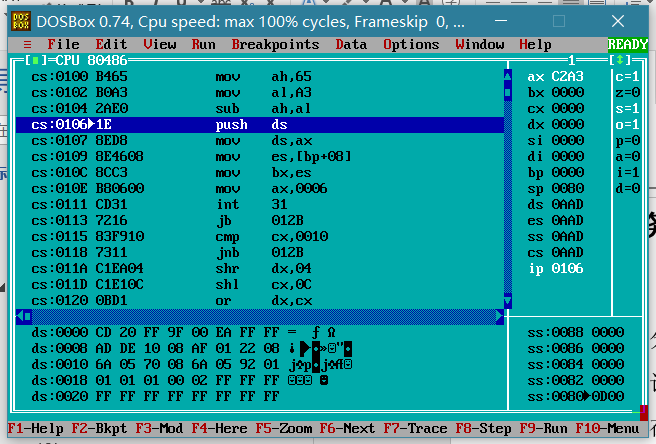
进行第一组数据：



第二组数据：



第三组数据：



标志位的改变和预测的相同。

## 任务二

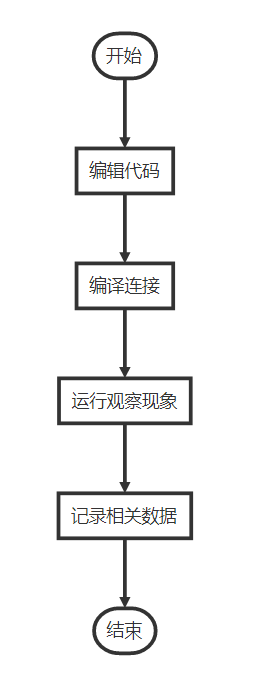
任务二是运行课本上45页题2.3的汇编代码，观察现象。

1. 分别记录执行到“MOV CX，10”和“INT 21H”之前的(BX), (BP),(SI),(DI)各是多少。
2. 记录程序执行到退出之前数据段开始40个字节的内容，指出程序运行结果是否与设想的一致。
3. 在标号LOPA前加上一段程序，实现新的功能：先显示提示信息“Press any key to begin!”, 然后，在按了一个键之后继续执行LOPA处的程序。

### 设计思想

由于课本上已经给出了代码，只需要将代码写入文本文件保存并使用ml编译链接，使用td打开程序进行运行。

### 流程图



### 源程序

.386

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

DATA SEGMENT USE16

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

BUF2 DB 10 DUP(0)

BUF3 DB 10 DUP(0)

BUF4 DB 10 DUP(0)

DATA ENDS

CODE SEGMENT USE16

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

START:

MOV AX, DATA

MOV DS, AX

; code after here

MOV SI, OFFSET BUF1

MOV DI, OFFSET BUF2

MOV BX, OFFSET BUF3

MOV BP, OFFSET BUF4

MOV CX, 10; count number

LOPA: MOV AL, [SI]; let ax = BUF1[0]

MOV [DI], AL

INC AL; add 1

MOV [BX], AL

ADD AL, 3; add 3

MOV DS:[BP], AL

INC SI

INC DI

INC BP

INC BX

DEC CX

JNZ LOPA; check jmp or not

;stop here

MOV AH, 4CH

INT 21H

CODE ENDS

END START

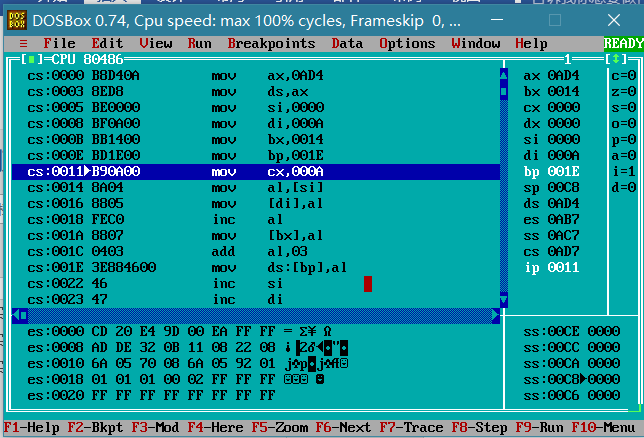
### 实验步骤

1. 编写代码：使用visual studio编写汇编代码并保存为asm文件。
2. 打开dosbox，挂载d:\github\asm\asm为c盘。
3. 输入c:切换到c盘，输入ml mission2.asm完成编译和连接。
4. 输入td mission2运行程序。
5. 单步运行程序，分别记录执行到“MOV CX，10”和“INT 21H”之前的(BX), (BP),(SI),(DI)各是多少，记录程序执行到退出之前数据段开始40个字节的内容，指出 程序运行结果是否与设想的一致。
6. 修改程序：在标号LOPA前加上一段程序，实现新的功能：先显示提示信息“Press any key to begin!”, 然后在按了一个键之后继续执行LOPA处的程序。

### 实验记录

使用visual studio编写代码并保存到d:\github\asm\asm文件夹，运行dosbox执行编译连接后运行程序。记录如下。

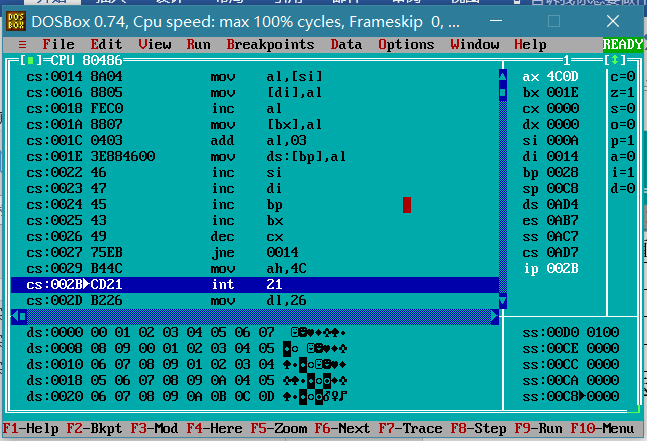
程序执行到MOV CX, 10之前的BX，BP，SI，DI如图所示：



值分别为：

|  |  |
| --- | --- |
| BX | 0014H |
| BP | 001EH |
| SI | 0000H |
| DI | 000AH |

运行程序到INT 21H，记录此时的BX，BP，SI，DI如图所示：



数据：

|  |  |
| --- | --- |
| BX | 001EH |
| BP | 0028H |
| SI | 000AH |
| DI | 0014H |

阅读汇编代码可知代码的功能是将BUF1中的值进行操作后存储到BUF2,BUF3,BUF4中，BUF2中的为原值，BUF3中为加1后的值，BUF4为加上4之后的值，而且是顺序写入，数据应该是：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BUF1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 9BUF2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| BUF3 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A |
| BUF4 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D |

参考上方截图中的数据可知程序正确。

修改代码让运行LOPA前输出一个字符串并等待用户按键以继续，本功能需要使用BIOS的功能，主要为9号调用和1号调用，需要注意的是输出的字符串必须是’$’结尾的，我尝试了一下如果不是’$’结尾的字符串将会一直输出直到遇到内存中一个’$’字符。修改后的代码：

.386

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

DATA SEGMENT USE16

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

BUF2 DB 10 DUP(0)

BUF3 DB 10 DUP(0)

BUF4 DB 10 DUP(0)

WAIT\_STR DB "Press any key to begin!$"; dolar in the end

DATA ENDS

CODE SEGMENT USE16

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

START:

MOV AX, DATA

MOV DS, AX

; code after here

MOV SI, OFFSET BUF1

MOV DI, OFFSET BUF2

MOV BX, OFFSET BUF3

MOV BP, OFFSET BUF4

MOV CX, 10; count number

LEA DX, OFFSET WAIT\_STR;load offset

MOV AH, 9;set function

INT 21H ; call function

; wait for key

MOV AH, 1

INT 21H

; begin loop

LOPA: MOV AL, [SI]; let ax = BUF1[0]

MOV [DI], AL

INC AL; add 1

MOV [BX], AL

ADD AL, 3; add 3

MOV DS:[BP], AL

INC SI

INC DI

INC BP

INC BX

DEC CX

JNZ LOPA; check jmp or not

;stop here

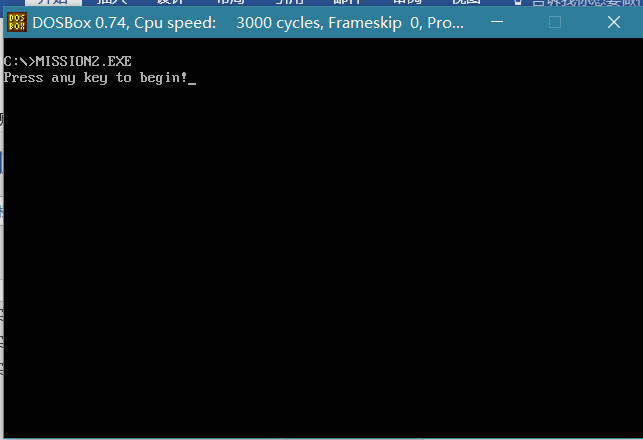
MOV AH, 4CH

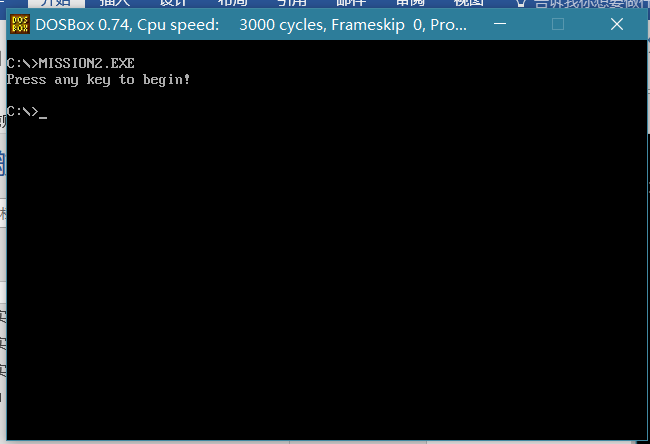
INT 21H

CODE ENDS

END START

运行截图：





修改正确。

## 任务三

对教材45页2.4题代码的改写，实现使用32位寄存器变址寻址方式运行，记录数据区前40个字节的内容。

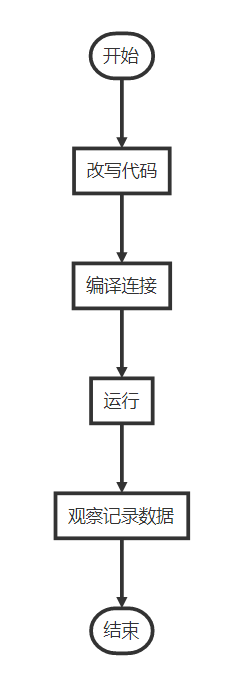
### 设计思想

本任务为使用32位寄存器变址寻址运行程序，记录数据段前40个字节的内容，所以只需要将4个寻址使用的寄存器替换为32位的即可。

变址寻址的实现：使用EBX存放BUF1的地址，根据他们在内存中的位置可以知道BUF2的地址位EBX+10，BUF3地址为EBX+20，BUF4地址为EBX+30，由此就可以改写为变址寻址。

寄存器和内存空间的分配：由于只是修改代码使其能够使用32位寄存器，还有就是使用变址寻址，那么我直接将两个功能结合到一起，使用32位寄存器对数据进行变址寻址，只使用一个ESI寄存器存放地址，CX计数。

### 流程图



### 源程序

.386

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

DATA SEGMENT USE16

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

BUF2 DB 10 DUP(0)

BUF3 DB 10 DUP(0)

BUF4 DB 10 DUP(0)

DATA ENDS

CODE SEGMENT USE16

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

START:

MOV AX, DATA

MOV DS, AX

; code after here

MOV ESI, OFFSET BUF1

MOV CX, 10

LOPA: MOV AL, [ESI]

MOV [ESI + 10], AL; according memory order

INC AL

MOV [ESI + 20], AL

ADD AL, 3

MOV [ESI + 30], AL

INC ESI

DEC CX

JNZ LOPA

;stop here

MOV AH, 4CH

INT 21H

CODE ENDS

END START

### 实验步骤

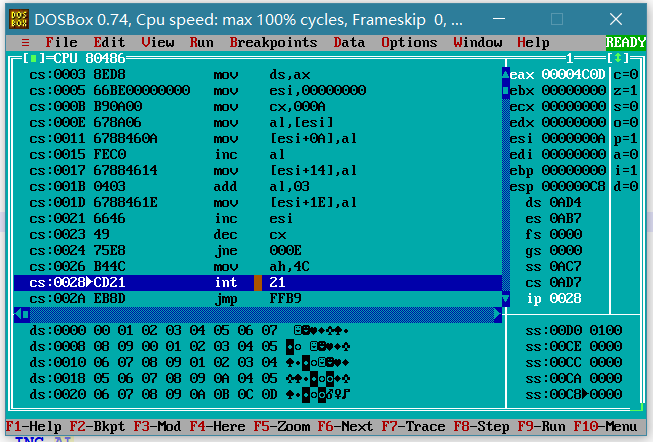
1. 实验步骤和任务二相同。

### 实验记录

记录主要是记录代码运行前后数据区前40个字节的内容。

编译连接没有报错，开始使用td运行程序。

单步运行程序直到INT 21H，记录截图如下：



运行结果与使用4个16寄存器表示地址时的结果相同。

## 任务4

### 设计思想

本次的实验任务有些复杂，需要实现准备重组，画好流程图对写程序帮助很大。

变量分配：

NOTICE\_NAME变量存放提示输入的变量，是一个以’$’结尾的字符串，与此类似的还有提示输入密码和商品名的字符串，也是以’$’结尾的。

BNAME和BPASS是程序存放的用户名和密码，长度是10个字节，IN\_NAME，IN\_PWD和IN\_GOODS是输入缓冲区，长度是22字节，前两字节存放长度和输入字符个数，分别是用户名，密码和商品的输入段。

G\_CNT是一个常量，表示每个商店里的商品数目，S\_CNT常量表示商店的总个数，定义商店时首先一个10个字节的字符串表示商店名，然后是每个商品的信息，10个字节的字符串表示商品名，然后是5个字的数据分别代表进价，售价，进货量，出货量，利润率，一共G\_CNT个商品。

G\_SIZE常量就是表示一个商品的字节大小，常量S\_SIZE表示一个商店占用的内存大小。

AUTH为一个字节的数据，标志是否通过授权，通过授权值为1，否则值为0。

GG\_INDEX类型是双字的数组，总共S\_CNT个元素，指的是所有的同种商品在每个商店中的地址，用于最后的求平均利润。

G\_INDEX是一个临时双字变量，用于在每个商店中遍历货物时使用的临时地址变量。

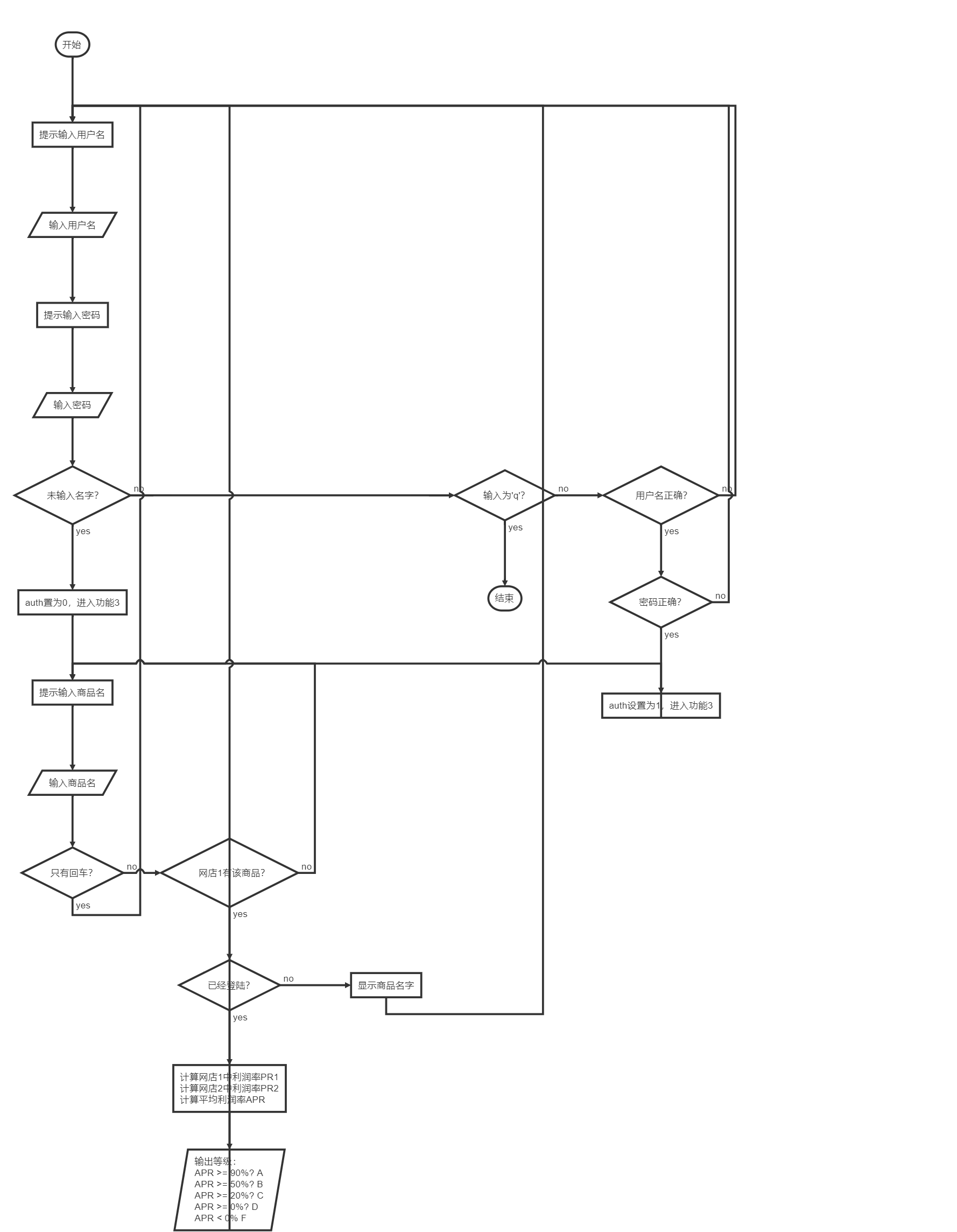
SHIFTLINE为一个字符串，用于换行。

S\_CNT是表示商店数量的一个常量，S\_TEMP\_CNT和G\_TEMP\_CNT用于循环中的累加计数，类型是字节，S\_INDEX存放的是商店的地址。

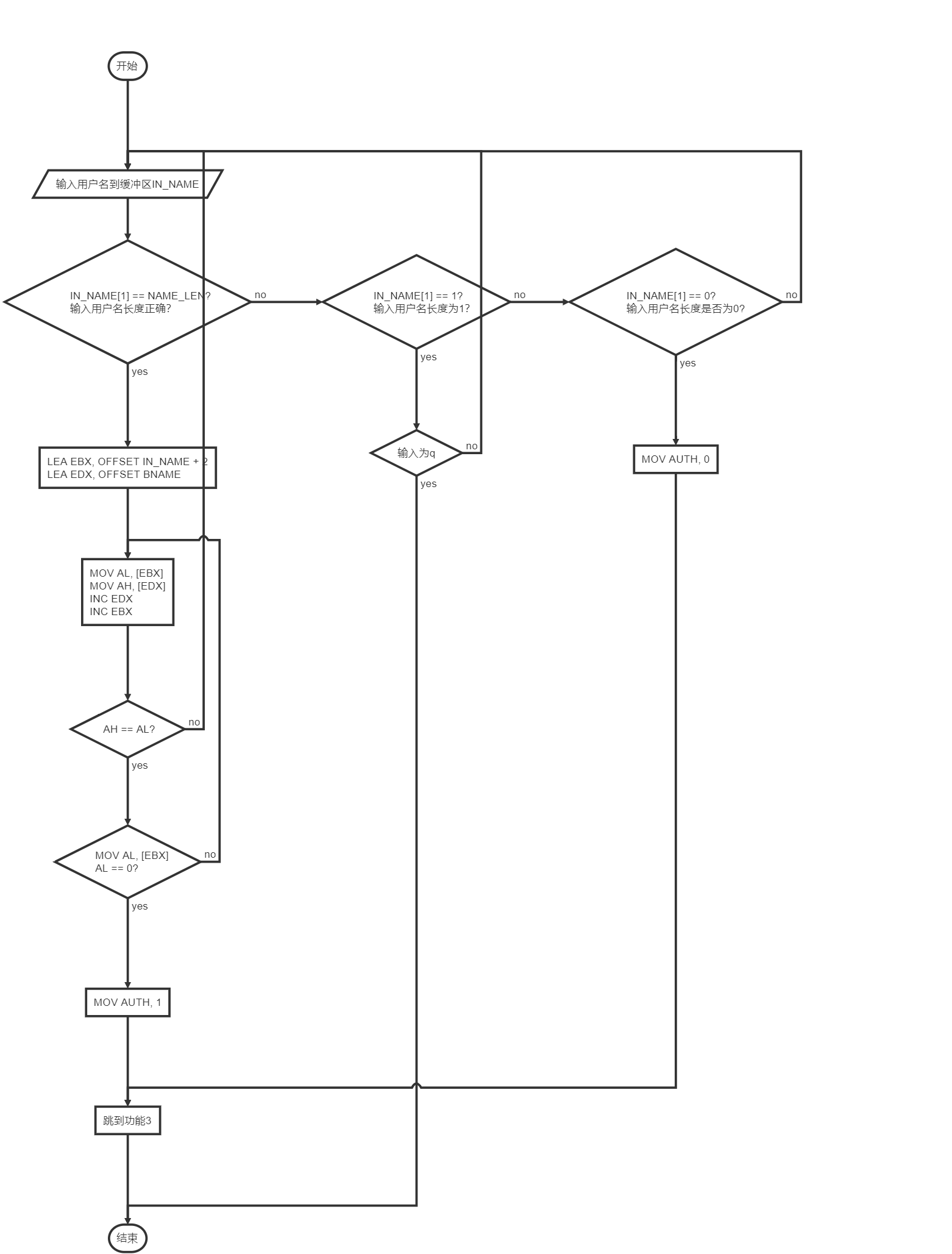
最后两个临时变量G\_COST和G\_PRO存放计算利润率中存放进货总花费和售出总销售额，大小是一个字。

### 流程图

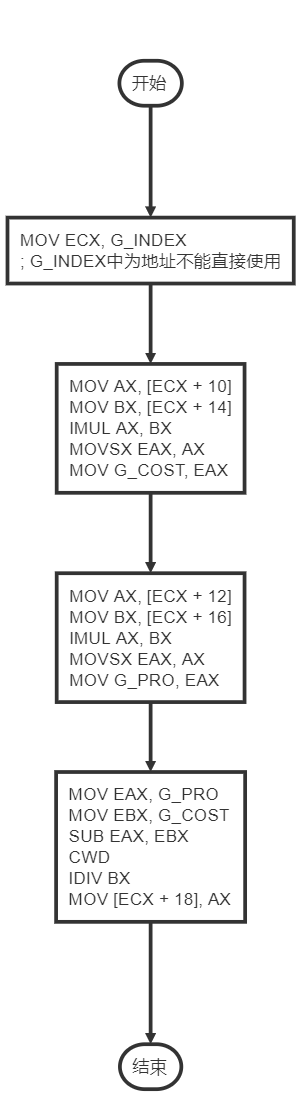
总的整体的流程图：



检测输入用户名或者密码是否正确的流程图：



计算利润率的流程图



### 源程序

.386

STACK SEGMENT USE16 STACK

DB 200 DUP(0)

STACK ENDS

DATA SEGMENT USE16

NOTICE\_NAME DB 'Please enter user name: $', 5 DUP(0)

NOTICE\_PWD DB 'Please enter user password: $', 2 DUP(0)

NOTICE\_GOODS DB 'Please enter goods name: $', 3 DUP(0)

BNAME DB 'yaning', 4 DUP(0); 用户名

BPASS DB 'passwd', 4 DUP(0); 密码

IN\_NAME DB 20, ?, 20 DUP(0)

IN\_PWD DB 20, ?, 20 DUP(0)

IN\_GOODS DB 20, ?, 20 DUP(0)

G\_CNT = 30

S1 DB 'SHOP1', 5 DUP(0)

GA1 DB 'PEN', 7 DUP(0)

DW 35, 56, 70, 25, ? ; 利润率未计算

GA2 DB 'BOOK', 6 DUP(0)

DW 12, 30, 25, 5, ? ; 利润率未计算

GAN DB G\_CNT - 2 DUP('Temp-Value', 15, 0, 20, 0, 30, 0, 2, 0, ?, ?); 其他商品暂时未知

S\_SIZE = $ - S1

S2 DB 'SHOP2', 5 DUP(0) ;网店名称，用0结束

GB1 DB 'BOOK', 6 DUP(0) ; 商品名称

DW 12, 28, 20,15, ? ;利润率还未计算

G\_SIZE = $ - GB1

GB2 DB 'PEN', 7 DUP(0) ;商品名称

DW 35, 50, 30, 24, ? ;利润率还未计算

GBN DB G\_CNT - 2 DUP('Temp-Value', 15, 0, 20, 0, 30, 0, 2, 0, ?, ?)

AUTH DB 0; 标志是否通过验证

GG\_INDEX DD 2 DUP(0); 一共两家店,存放查询的物品的地址在内存中

G\_INDEX DD 0

SHIFTLINE DB 13, 10, '$'

S\_CNT = 2

S\_TEMP\_CNT DB 0

G\_TEMP\_CNT DB 0

SG\_INDEX DD 0

S\_INDEX DD 0

G\_COST DD 0

G\_PRO DD 0

DATA ENDS

CODE SEGMENT USE16

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

START:

MOV AX, DATA

MOV DS, AX

; code after this

FUNC1: ; 功能1开始

LEA DX, OFFSET NOTICE\_NAME; 提示输入用户名

MOV AH, 9

INT 21H

LEA DX, OFFSET IN\_NAME ; 输入用户名

MOV AH, 10

INT 21H

; 换行

LEA DX, OFFSET SHIFTLINE

MOV AH, 9

INT 21H

LEA DX, OFFSET NOTICE\_PWD;提示输入密码

MOV AH, 9

INT 21H

LEA DX, OFFSET IN\_PWD;输入密码

MOV AH, 10

INT 21H

LEA DX, OFFSET SHIFTLINE

MOV AH, 9

INT 21H

;LEA DX, OFFSET NEWLINE

;MOV AH, 9

;INT 21H

FUNC2: ;功能2开始

; 用户名输入长度为0？

CMP IN\_NAME[1], 0

; 调到功能3

JZ FUNC3

; 长度为1，检测是否是q

CMP IN\_NAME[1], 1

; 不是1跳check\_name

JNZ CHECK\_NAME

CMP IN\_NAME[2], 'q'

; 输入为q，退出

JZ QUIT

CHECK\_NAME:

; 检查用户名

; 已知用户名长度为6，作为已知量使用

CMP IN\_NAME[1], 6

; 不为6错误

JNZ FUNC1

MOV CX, 6

LEA EBX, OFFSET IN\_NAME + 2

LEA EDX, OFFSET BNAME

LOOP\_NAME: ; 偷懒从高到低检查

MOV AH, [EBX]

MOV AL, [EDX]

CMP AH, AL

JNZ FUNC1

INC EBX

INC EDX

DEC CX

JNZ LOOP\_NAME

; 密码也是6位

CMP IN\_PWD[1], 6

JNZ FUNC1

MOV CX, 6

LEA EBX, OFFSET IN\_PWD

ADD EBX, 2

LEA EDX, OFFSET BPASS

LOOP\_PWD:

MOV AH, [EBX]

MOV AL, [EDX]

CMP AH, AL

; 密码不同跳功能1

JNZ FUNC1

INC EBX

INC EDX

DEC CX

JNZ LOOP\_PWD

; 通过用户名和密码检测

MOV AUTH, 1

; 进入功能3

FUNC3:

INPUT\_GOODS:

LEA DX, OFFSET NOTICE\_GOODS; 提示输入商品名

MOV AH, 9

INT 21H

LEA DX, OFFSET IN\_GOODS; 输入商品名

MOV AH, 10

INT 21H

LEA DX, OFFSET SHIFTLINE

MOV AH, 09

INT 21H

; 只有回车跳转功能1

CMP IN\_GOODS[1], 0

JZ FUNC1

; MOV AH, 0

; 检测网店1是否有此商品

LEA EBX, OFFSET S1

ADD EBX, 10; 跳到商品名处

LEA EDX, OFFSET IN\_GOODS + 2

MOV G\_INDEX, EBX

MOV CL, 0

S1\_CK: ; 检测物品是否存在第一个商店

MOV AL, [EBX]

MOV AH, [EDX]

CMP AL, AH

JNZ NEXT\_SG

INC EBX

INC EDX

MOV AL, [EBX]

CMP AL, 0; 检测商品名是否走到结尾

JNZ S1\_CK

; 走到这里说明网店1有该商品

; 判断授权

; 未授权, 输出商品名跳FUNC1

CMP AUTH, 1

JZ CALCU\_PR

LEA EBX, OFFSET IN\_GOODS

XOR EAX, EAX

MOV AL, [EBX + 1]

ADD EBX, EAX

ADD EBX, 2

MOV BYTE PTR [EBX], '$'

LEA DX, OFFSET IN\_GOODS + 2

MOV AH, 9

INT 21H

; 换行

LEA DX, OFFSET SHIFTLINE

MOV AH, 09

INT 21H

JMP FUNC1

NEXT\_SG:

MOV EBX, G\_INDEX

ADD EBX, G\_SIZE

INC CL

CMP CL, G\_CNT

JZ INPUT\_GOODS

MOV G\_INDEX, EBX

JMP S1\_CK

CALCU\_PR:

; 计算利润率,结果乘以100保证整数显示

LEA EBX, OFFSET S1

MOV S\_INDEX, EBX

; 已检测商店个数

MOV S\_TEMP\_CNT, 0

LOOP\_S:

MOV EBX, S\_INDEX

ADD EBX, 10

MOV G\_INDEX, EBX

MOV G\_TEMP\_CNT, 0

LOOP\_G:

; 首先找到商品位置

; EBX记录商品位置

MOV EBX, G\_INDEX

LEA EDX, OFFSET IN\_GOODS + 2

; 判断商品名是否为所求

CHG\_S: ; 检测循环

MOV AL, [EBX]

MOV AH, [EDX]

CMP AL, AH; 不是这件商品

JNZ NEXT\_G

INC EBX

INC EDX

MOV AL, [EBX]

; 检测是否到头

CMP AL, 0;

JNZ CHG\_S;没到头继续

; 到头了, 说明就是这个商品，计算利润率

MOV ECX, G\_INDEX

MOV AX, [ECX + 10]

MOV BX, [ECX + 14]

IMUL AX, BX

; 80x86低位在前

LEA ESI, OFFSET G\_COST

MOVSX EAX, AX

MOV DS:[ESI], EAX

;MOV WORD PTR DS:[ESI + 2], 0

;MOV ECX, G\_INDEX

MOV AX, [ECX + 12]

MOV BX, [ECX + 16]

IMUL AX, BX

LEA ESI, OFFSET G\_PRO

MOVSX EAX, AX

MOV DS:[ESI], EAX

;MOV DS:[ESI + 2], DX

MOV EAX, G\_PRO

MOV EBX, G\_COST

; 我怎么知道结果和0的大小？

SUB EAX, EBX

IMUL EAX, 100

MOV EDX, 0

CMP EAX, 0

JGE L1

MOV EDX, -1

;MOVSX EDX, EAX

L1: IDIV EBX

; 由于结果8个字节，一个寄存器放不下,抛弃高位EDX

MOV EBX, G\_INDEX

MOV [EBX + 18], AX;EAX也只取前两个字节

; 纠结：如何将地址放入GG\_INDEX数组

XOR EAX, EAX

MOV AL, G\_TEMP\_CNT

MOV CL, 4

IMUL CL

LEA BX, OFFSET GG\_INDEX

ADD AX, BX

MOV EBX, G\_INDEX

MOV [EAX], EBX

;LEA EAX, OFFSET GG\_INDEX

;MOV ECX, G\_TEMP\_CNT

;LEA EAX, OFFSET GG\_INDEX

;XOR ECX, ECX

;MOV CL, S\_TEMP\_CNT

;ADD EAX, ECX

;MOV [EAX], EBX

; 利润率计算完成，下面算平均

NEXT\_G:

ADD G\_INDEX, G\_SIZE

INC G\_TEMP\_CNT

MOV CL, G\_TEMP\_CNT

CMP CL, G\_CNT

; 未完成商品继续LOOP\_G

JL LOOP\_G

NEXT\_S:

MOV G\_TEMP\_CNT, 0

INC S\_TEMP\_CNT

ADD S\_INDEX, S\_SIZE

MOV CL, S\_TEMP\_CNT

CMP CL, S\_CNT

; 计算完毕去计算平均数目

JNZ LOOP\_S

AVE: ; 计算平均利润

MOV CL, 0

LEA EDX, OFFSET GG\_INDEX

MOV EBX, [EDX]

XOR AX, AX

NEXT\_IDX:

ADD AX, [EBX + 18]

; 移到下一个同种商品位置

ADD EDX, 4

MOV EBX, [EDX]

; 计数加一

INC CL

CMP CL, S\_CNT

JL NEXT\_IDX

MOV EDX, 0

CMP AX, 0

JGE L2

MOV EDX, -1

L2:

IDIV CL

; 计算后AL内存放结果的商，AH存放余数

FUNC4:

; 检测利润率范围

CMP AL, 90

JGE SHOWA

CMP AL, 50

JGE SHOWB

CMP AL, 20

JGE SHOWC

CMP AL, 0

JGE SHOWD

JMP SHOWF

SHOWA:

MOV DL, 'A'

JMP SHOW

SHOWB:

MOV DL, 'B'

JMP SHOW

SHOWC:

MOV DL, 'C'

JMP SHOW

SHOWD:

MOV DL, 'D'

JMP SHOW

SHOWF:

MOV DL, 'F'

SHOW:

MOV AH, 2

INT 21H

; 换行

LEA DX, OFFSET SHIFTLINE

MOV AH, 09

INT 21H

JMP FUNC1

QUIT:

;stop here

MOV AH, 4CH

INT 21H

CODE ENDS

END START

### 实验步骤

1. 打开dosbox，挂载d:\github\asm\asm为c盘，输入c:切换到c盘，输入masm mission4编译代码文件，输入link mission4连接文件。
2. 输入mission4运行程序，根据提示输入用户名和密码。
3. 输入用户名YANING，密码是PASSWD，enter登录。
4. 输入要查询的商品名如PEN，就会计算商品的利润率，并据此计算商品的等级并输出。
5. 回到用户名输入界面，输入q退出程序。

### 实验记录

本次实验前我已经完成了实验的大部分内容，实验中我主要是修复了几个难缠的bug，实验中的记录如下：

1. 输入姓名和密码登录后输入商品名，然而我输入商品名后程序陷入了死循环，死循环最常见的原因最有可能是没有及时更新变量或者程序计算出错。进入td找到比较输入货物名处，输入F4直接运行到此处，循环几轮后用于比较的AH和AL没有改变，由此推断未及时对存储地址的EBX和EDX及时递增造成错误，修改后可以正确判断输入的商品名是否存在于商店中。
2. 计算利润率的逻辑出现错误，计算进货量和进货价的乘积时我使用的是字乘字的方法，结果是双字结果是存放在EAX里的，我以为高位存放在DX中，因此写入到内存中的数据肯定是错误的，直接导致了我在做除法时程序直接崩溃了，修改后我直接将EAX存放在了G\_COST和G\_PRO的双字中，计算的销售额和进货成本为正确的。
3. 计算完进货成本和销售额，两者相减，结果存放在EAX中，接下来需要利润除以进货成本，运行除法时使用的是双字除法，即(EDX,EAX) / ECX，我未将EAX的符号拓展到EDX中，导致结果错误，改正的方法就是CWD;MOVSX EDX, DX，然后进行运算即可。
4. 还有一个困难的地方就是多重循环的设计，我在网上借鉴了一些blog后总结出了一个规律，就是for循环先变成while循环，然后根据条件判断，满足继续运行，循环结束处更新循环变量，然后跳转到开始的地方判断是否继续运行，这样下来就比较简单了。

上面就似乎我在实验里遇到的主要困难部分，只要分配好空间、理清逻辑还是可以解决问题的。

# 体会

这次的汇编实验总体来说由于我的准备的比较充分，任务书上说的也比较清楚，实验时没有遇到什么较大的困难。通过这次的汇编实验我知道了在内存中代码和数据属于同一个地位，都只是内存中的0101而已，只是被不同的CPU部件读取而导致意义不同，这也就是冯·诺依曼架构的计算机的同一个概念，数据与指令没有区别。这次实验还见识到了计算机对于数据表示并没有什么正负之分，都是存储的补码，加就是直接加，减就是加补码，对我理解计算机系统有很大帮助。

第一次实验做的任务一到任务3都是比较简单的东西，任务4要求我们去设计实现一个基础的应用，难度还是有点大的。本次实验的主要难点主要有寻址方式的使用和算数运算指令的使用，特别是除法指令和符号拓展指令，需要记住操作数的位置和结果的位置，借助课本还是记住了常用的操作符的使用方法。

实验中很多地方都陷入了死循环，导致程序输入后便陷入了无响应状态，根据一步步的debug还是成功找到了问题所在处。

本次实验的时候由于还没有学习子程序的操作方法，所以在写程序的时候很多地方都不能抽象出来变成函数，变成函数可以极大的减少劳动量，程序的修改和更正也会变得简单，下一次的实验内容就是使用子程序简化代码和逻辑，希望在接下来的学习中可以更好地熟悉汇编。

# 参考文献

[1] 王元珍等.80x86汇编语言程序设计.版本(第1版)

[2] 王晓虹等.汇编语言程序设计教程.版本(第1版)