

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实验**

**实验名称： 实验一 编程基础**

**实验时间： 2018-3-26，14：00-17：30 实验地点： 南一楼**

**指导教师： 朱虹**

**专业班级： 计算机科学与技术1601班**

**学 号： U201614531 姓 名： 刘本嵩**

**同组学生： 无 报告日期： 2018年 3 月 26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**

[1 实验目的与要求 2](#_Toc1275167533)

[2 实验内容 3](#_Toc219193243)

[2.1 任务1：《80X86汇编语言程序设计》教材中 P31的 1.14题 3](#_Toc1327341636)

[2.2 任务2：《80X86汇编语言程序设计》教材中 P45的 2.3题 3](#_Toc763973014)

[2.3 任务3：《80X86汇编语言程序设计》教材中 P45的 2.4题的改写 4](#_Toc2119631114)

[2.4 任务4：内存单元的访问 4](#_Toc712642282)

[2.5 任务5：网店商品信息查询 5](#_Toc2079517283)

[2.5.1 实验背景 5](#_Toc336588420)

[2.5.2 功能一：提示并输入登录用户的姓名与密码 6](#_Toc1831018573)

[2.5.3 功能二：登录信息认证 6](#_Toc1440881484)

[2.5.4 功能三：计算指定商品的利润率。 7](#_Toc636752152)

[2.5.5 功能四：将功能三计算的平均利润率进行等级判断，并显示判断结果。 7](#_Toc1091235989)

[3 实验过程 8](#_Toc1767545562)

[3.1 任务1 8](#_Toc1631228973)

[3.1.1 实验步骤 8](#_Toc181938177)

[3.1.2 实验记录与分析 9](#_Toc865310198)

[3.2 任务2 11](#_Toc906664936)

[3.2.1 实验步骤 11](#_Toc49597364)

[3.2.2 修改后源程序 11](#_Toc394006413)

[3.2.3 实验记录与分析 12](#_Toc1481186837)

[3.3 任务3 15](#_Toc892757746)

[3.3.1 实验步骤 15](#_Toc1365380522)

[3.3.2 修改后源程序 15](#_Toc1681446813)

[3.3.3 实验记录与分析 16](#_Toc1589484034)

[3.4 任务4 19](#_Toc860806134)

[3.4.1 实验步骤 19](#_Toc244220883)

[3.4.2 源程序 19](#_Toc342478668)

[3.4.3 实验记录与分析 21](#_Toc1007865421)

[3.5 任务5 21](#_Toc1346160861)

[3.5.1 实验步骤 21](#_Toc615297885)

[3.5.2 源程序 22](#_Toc350241971)

[3.5.3 实验记录与分析 29](#_Toc473844746)

[4 总结与体会 31](#_Toc834491128)

[5 参考文献 32](#_Toc1677583607)

# 实验目的与要求

本次实验的主要目的与要求有下面6点，所有的任务都会围绕这6点进行，希望大家事后检查自己是否达到这些目的与要求。

(1) 掌握汇编源程序编辑工具、汇编程序、连接程序、调试工具TD的使用；

(2) 理解数、符号、寻址方式等在计算机内的表现形式；

(3) 理解指令执行与标志位改变之间的关系；

(4) 熟悉常用的DOS System Call；

(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

要求：

(1) 直接在TD中输入指令，完成两个数的求和、求差的功能。求和/差后的结果放在(AH)中。

(2) 请事先指出执行指令后(AH)、标志位 SF、OF、CF、ZF的内容。

(3) 记录上机执行后的结果，与（2）中对应的内容比较。

(4) 求差运算中，若将A、B视为有符号数，且A>B, 标志位有何特点？

若将A、B视为无符号数，且A>B, 标志位又有何特点？

## 任务2：《80X86汇编语言程序设计》教材中 P45的 2.3题

要求：

(1) 分别记录执行到“MOV CX, 10”和“INT 21H”之前的(BX)，(BP)，(SI)，(DI)各是多少。

(2) 记录程序执行到退出之前数据段开始40个字节的内容，指出程序运行结果是否与设想的一致。

(3) 在标号LOPA前加上一段程序，实现新的功能：先现实提示信息“Press any key to begin！”，然后，再按了一个键之后继续执行LOPA处的程序。

## 任务3：《80X86汇编语言程序设计》教材中 P45的 2.4题的改写

要求：

(1) 实现的功能不变，对数据段中变量访问时所用到的寻址方式中的寄存器改成32位寄存器。

(2) 内存单元中数据的访问采用变址寻址方式。

(3) 记录程序执行到退出之前数据段开始40个字节的内容，检查程序运行结果是否与设想的一致。

(4) 在TD代码窗口中观察并记录机器指令代码在内存中的存放形式，并与TD中提供的反汇编语句及自己编写的源程序语句进行对照，也与任务2做对比。（相似语句记录一条即可，重点理解机器码与汇编语句的对应关系，尤其注意操作数寻址方式的形式）。

(5) 观察连续存放的二进制串在反汇编成汇编语言语句时，从不同字节位置开始反汇编，结果怎样？理解 IP/EIP指明指令起始位置的重要性。

## 任务4：内存单元的访问

以四种不同的内存寻址方式，将自己学号的后四位依次存储到 以 XUEHAO开头的存储区中，要求学号的存放以字符方式存放。

要求：在报告中给出完整的程序；给出运行效果截图；（不需要画流程图）；在程序注释中，明确指出访问存储单元时，用的是什么寻址方式。

## 任务5：网店商品信息查询

### 实验背景

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。

例如：

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

BPASS DB ‘test’，0，0 ；密码

N EQU 30

S1 DB ‘SHOP1’,0 ;网店名称，用0结束

GA1 DB ‘PEN’, 7 DUP(0) ; 商品名称

DW 35，56，70，25，？ ；利润率还未计算

GA2 DB ‘BOOK’, 6 DUP(0) ; 商品名称

DW 12，30，25，5，？ ；利润率还未计算

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

S2 DB ‘SHOP2’,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

GB2 DB ‘PEN’, 7 DUP(0) ; 商品名称

DW 35，50，30，24，？ ；利润率还未计算

……

### 功能一：提示并输入登录用户的姓名与密码

（1）使用9号DOS系统功能调用，先后分别提示用户输入姓名和密码。

（2）使用10号DOS系统功能调用，分别输入姓名和密码。输入的姓名字符串放在以in\_name为首址的存储区中，密码放在以in\_pwd为首址的存储区中，进入功能二的处理。

（3）若输入姓名时只是输入了回车，则将0送到AUTH字节变量中，跳过功能二，进入功能三；若在输入姓名时仅仅输入字符q，则程序退出。

### 功能二：登录信息认证

（1）使用循环程序结构，比较姓名是否正确。若不正确，则跳到（3）。

（2）若正确，再比较密码是否相同，若不同，跳到（3）。

（3）若名字或密码不对，则提示登录失败，并回到“功能一（1）”的位置，提示并重新输入姓名与密码。

（4）若名字和密码均正确，则将1送到AUTH变量中，进到功能三。

提示：字符串比较时，当采用输入串的长度作为循环次数时，若因循环次数减为0而终止循环，则还要去判断网店中定义的字符串的下一个字符是否是结束符0，若是，才能确定找到了（这样做是为了避免输入的字符串仅仅是数据段中所定义字符串的子集的误判情况）。

### 功能三：计算指定商品的利润率。

（1）提示用户输入要查询的商品名称。若未能在第一个网店中找到该商品，重新提示输入商品名称。若只输入回车，则回到功能一（1）。

（2）判断登录状态，若是已经登录的状态，转到（3）。否则，转到（4）。

（3）首先计算第一个网店该商品的利润率PR1，然后在第二个网店中寻找到该商品，也计算其利润率PR2。最后求出该商品的平均利润率APR=(PR1+PR2)/2。进入功能四。

（4）若是未登录状态，则只在下一行显示该商品的名称，然后回到功能一（1）。

要求尽量避免溢出。

提示：使用循环程序结构，注意寻址方式的灵活使用。结果只保留整数部分。

### 功能四：将功能三计算的平均利润率进行等级判断，并显示判断结果。

（1）等级显示方式：若平均利润率大于等于90%，显示“A”；大于等于50%，显示“B”；大于等于20%，显示“C”；大于等于0%，显示“D”；小于0%，显示“F”。

提示：使用分支程序结构，采用2号DOS系统功能调用显示结果（注意，“%”是不要出现在计算式子和指令语句中的）。

（2）使用转移指令回到“功能一（1）”处（提示并输入姓名和密码）。

# 实验过程

## 任务1

### 实验步骤

1. 准备上机实验环境。

2. 在TD的代码窗口中的当前光标下输入第一个运算式对应的两个8位数值对应的指令语句MOV AH,33；MOV AL,5AH；ADD AH,AL；观察代码区显示的内容与自己输入字符之间的关系；然后确定CS:IP指向的是自己输入的第一条指令的位置，单步执行三次，观察寄存器内容的变化，记录标志寄存器的结果。然后依次修改第一和第二条指令，完成全部三组数的求和功能。

3. 然后修改第三条指令为 SUB AH,AL，然后依次重新修改第一和第二条指令中的立即数，完成全部三组数的求差的功能。

4. 下面给出所有数求和求差的预期结果。

第一组数+33H和+5AH

求和的结果为8DH SF=1、OF=1、CF=0、ZF=0

求差的结果为D9H SF=1、OF=0、CF=1、ZF=0

第二组数-29H和-5DH

求和的结果为7AH SF=0、OF=1、CF=1、ZF=0

求差的结果为34H SF=0、OF=0、CF=0、ZF=0

第三组数+65H和-5DH

求和的结果为08H SF=0、OF=0、CF=1、ZF=0

求差的结果为C2H SF=1、OF=1、CF=1、ZF=0

### 实验记录与分析

第一组数的求和：

屏幕快照%202017-03-15%20下午3

屏幕快照%202017-03-15%20下午3

屏幕快照%202017-03-15%20下午3

第一组数的求差：

屏幕快照%202017-03-15%20下午3

屏幕快照%202017-03-15%20下午3

屏幕快照%202017-03-15%20下午3

第二组数的求和：

屏幕快照%202017-03-15%20下午3

屏幕快照%202017-03-15%20下午3

屏幕快照%202017-03-15%20下午3

第二组数的求差：

屏幕快照%202017-03-15%20下午3

屏幕快照%202017-03-15%20下午3

屏幕快照%202017-03-15%20下午3

第三组数的求和：

屏幕快照%202017-03-15%20下午3

屏幕快照%202017-03-15%20下午3

屏幕快照%202017-03-15%20下午3

第三组数的求差：

屏幕快照%202017-03-15%20下午3

屏幕快照%202017-03-15%20下午3

屏幕快照%202017-03-15%20下午3

显然，得到的结果和预期结果相符，结果正确。

## 任务2

### 实验步骤

(1) 准备上机环境，编写实验程序

(2) 经MASM汇编，LINK连接后并且确认源程序正确无误后，使用WD对连接后的EXE程序进行调试，打断点，执行至mov cx,10后记录(BX)，(BP)，(SI)，(DI)的值。继续打断点至int 21h处，记录(BX)，(BP)，(SI)，(DI)的值。然后记录DS段的前40个字节的数据。

(3) 然后重新更改程序，通过系统21H中断的9号调用显示相应的字符串，然后再通过系统21H中断的1号调用读取输入字符，中断返回，继续执行LOPA处的程序。

### 修改后源程序

.386

stack segment use16 stack

db 200 dup(0)

stack ends

data segment use16

msg db “Press any key to begin!$”

buf1 db 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

buf2 db 10 dup (0)

buf3 db 10 dup (0)

buf4 db 10 dup (0)

data ends

code segment use16

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

include rlib.inc

start: mov ax, data

mov ds, ax

mov si, offset buf1

mov di, offset buf2

mov bx, offset buf3

mov bp, offset buf4

mov cx, 10

lea dx, msg

push dx

call rlib\_print

call rlib\_pause

lopa: mov al, [si]

mov [di], al

inc al

mov [bx], al

add al, 3

mov ds:[bp], al

inc si

inc di

inc bp

inc bx

dec cx

jnz lopa

mov ah, 4ch

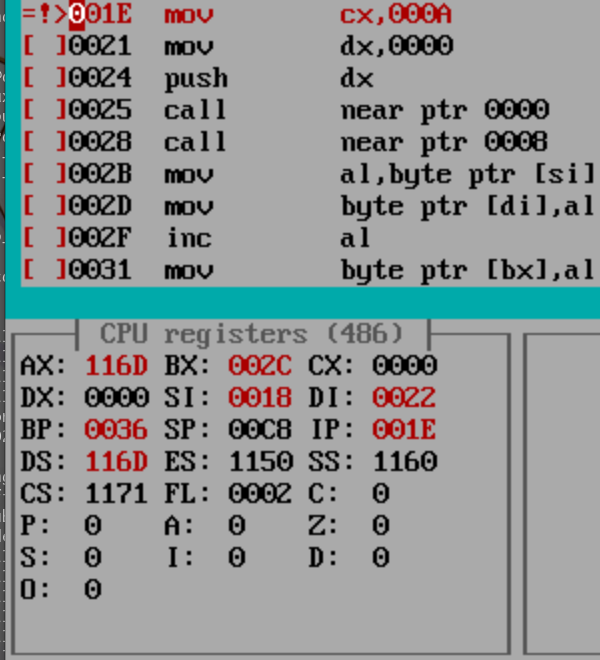
int 21h

code ends

end start

### 实验记录与分析

（1）使用wd进行调试。命中断点后，有

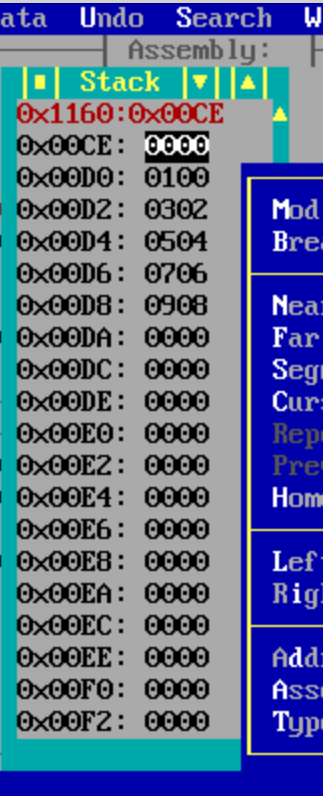


下一个断点处，有



不难从图中直接读出所需的x16寄存器值。

1. 直接看ds指向的内存区。



1. 只需调用rlib.inc::rlib\_print和rlib.inc::rlib\_pause，即可。

rlib\_print proc

; print string to stdout(dos, 16)

; Arg: word ptr to\_print

pop ax

pop dx

push ax

mov ah, 9

int 21h

ret

rlib\_print endp

rlib\_pause proc

; pause

; Arg:

mov ah, 1

int 21h

ret

rlib\_pause endp

## 任务3

### 实验步骤

(1) 根据任务的要求以及任务2的相关程序修改程序源代码，经确认无误后汇编连接。

(2) 在WD中加载连接后的可执行文件，观察数据段开始的前40个字节，并打断点，在程序退出前观察数据段开始的前40个字节。

(3) 在程序运行的过程中，观察机器指令在内存中的存放方式，同时比较反汇编代码与自己的源程序之间的区别。

(4) 使用GOTO语句，观察不同字节位置开始反汇编后程序的不同。

### 修改后源程序

.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:stack

start: mov ax, data

mov ds, ax

mov ebx, 0

mov cx, 10

lopa: mov al, [ebx + buf1]

mov [ebx + buf2], al

inc al

mov [ebx + buf3], al

add al, 3

mov [ebx + buf4], al

inc ebx

dec cx

jnz lopa

mov ax, 4c00h

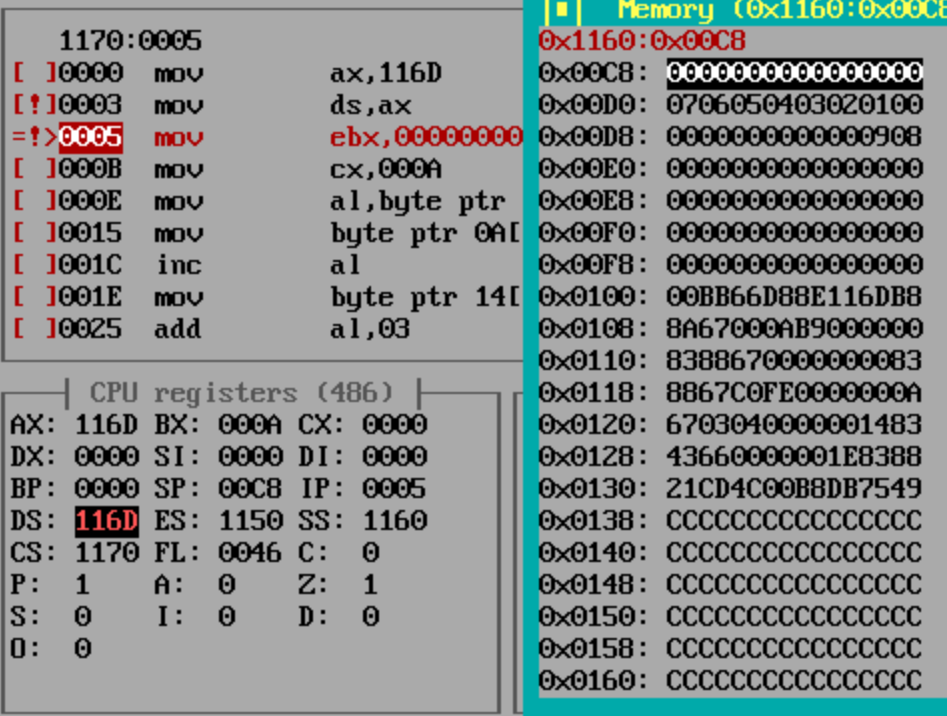
int 21h

code ends

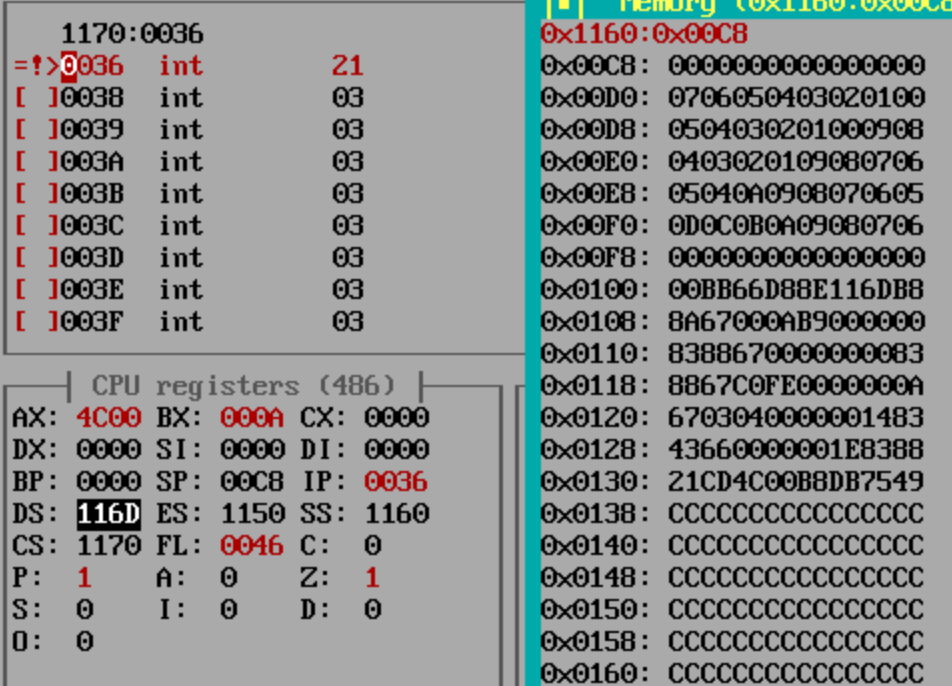
end start

### 实验记录与分析

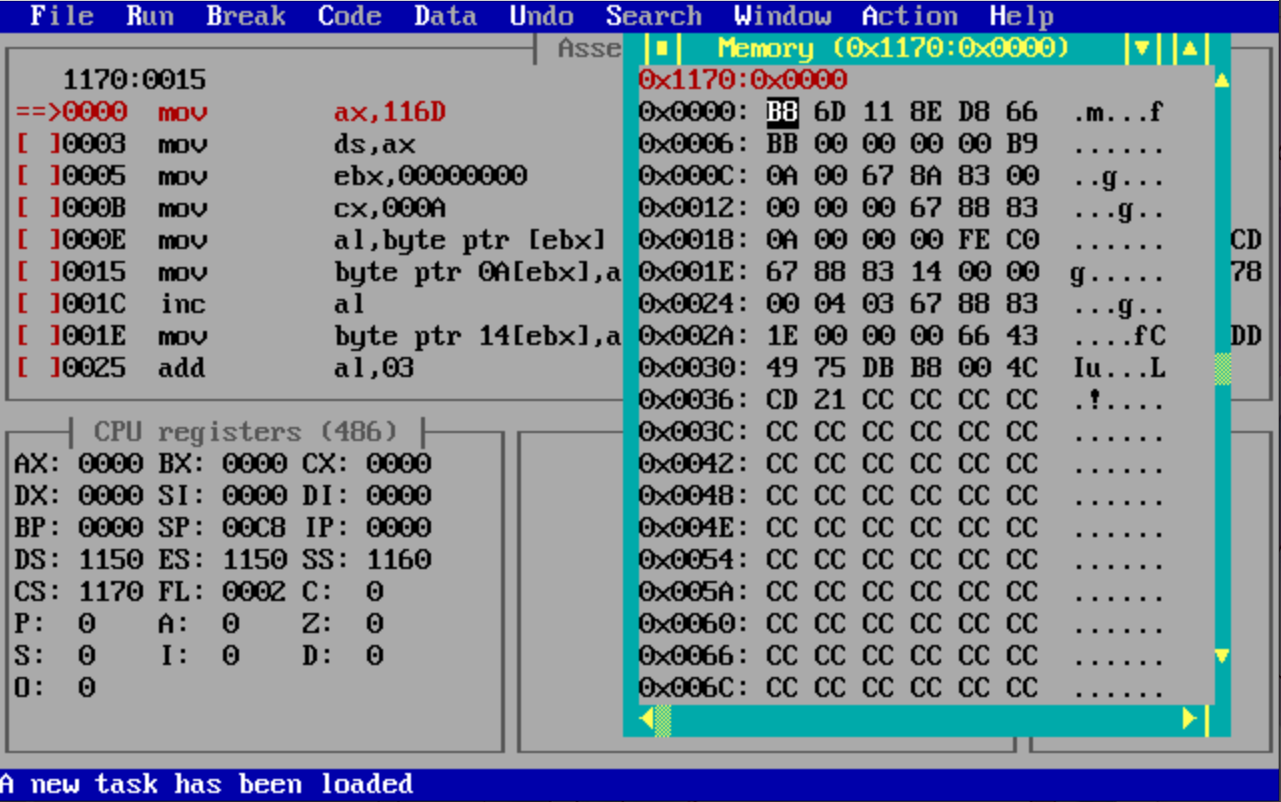
1. 记录程序执行到退出之前数据段开始40个字节的内容，检查程序运行结果是否与设想的一致。



程序退出之前，栈数据区内容如图。



1. 在TD代码窗口中观察并记录机器指令代码在内存中的存放形式，并与TD中提供的反汇编语句及自己编写的源程序语句进行对照，也与任务2做对比。（相似语句记录一条即可，重点理解机器码与汇编语句的对应关系，尤其注意操作数寻址方式的形式）。



All instruction encodings are subsets of the general instruction format

shown in Figure 17-1. Instructions consist of optional instruction

prefixes, one or two primary opcode bytes, possibly an address specifier

consisting of the ModR/M byte and the SIB (Scale Index Base) byte, a

displacement, if required, and an immediate data field, if required.

Smaller encoding fields can be defined within the primary opcode or

opcodes. These fields define the direction of the operation, the size of the

displacements, the register encoding, or sign extension; encoding fields

vary depending on the class of operation.

Most instructions that can refer to an operand in memory have an addressing

form byte following the primary opcode byte(s). This byte, called the ModR/M

byte, specifies the address form to be used. Certain encodings of the ModR/M

byte indicate a second addressing byte, the SIB (Scale Index Base) byte,

which follows the ModR/M byte and is required to fully specify the

addressing form.

Addressing forms can include a displacement immediately following either

the ModR/M or SIB byte. If a displacement is present, it can be 8-, 16- or

32-bits.

If the instruction specifies an immediate operand, the immediate operand

always follows any displacement bytes. The immediate operand, if specified,

is always the last field of the instruction.

1. 观察连续存放的二进制串在反汇编成汇编语言语句时，从不同字节位置开始反汇编，结果怎样？理解 IP/EIP指明指令起始位置的重要性。

结果不同。因此必须有IP/EIP/RIP指明指令起始位置。

## 任务4

### 实验步骤

(1) 根据题目的要求编写程序，完成相应的功能。

(2) 将程序经过汇编，链接，确认没有错误后运行。

(3) 用4种mov指令分别进行寻址写入，检查没有错误即可。

### 源程序

首先给出rlib.inc

rlib\_print proc

; print string to stdout(dos, 16)

; Arg: word ptr to\_print

pop ax

pop dx

push ax

mov ah, 9

int 21h

ret

rlib\_print endp

rlib\_pause proc

; pause

; Arg:

mov ah, 1

int 21h

ret

rlib\_pause endp

rlib\_exit proc

; libc exit

; Arg:

mov ah, 4ch

int 21h

rlib\_exit endp

然后给出4.asm

name rt4

.model small

.stack 1024

.data

buf db 'xuehao',?,?,?,?,0ah,'$'

num db '4531'

.code

jmp \_\_rlib\_start

include rlib.inc

\_\_rlib\_start:

mov ax, @data

mov ds, ax

;寄存器间接寻址

mov si, offset num

mov bx, offset buf + 6

mov di, [si]

mov [bx], di

;变址寻址

mov al, num + 1

mov 1[bx], al

;基址加变址寻址

mov si, 1

mov al, num + 2

mov 1[bx][si], al

;直接寻址

mov al, num + 3

mov byte ptr buf + 9, al

lea bx, buf

push bx

call rlib\_print

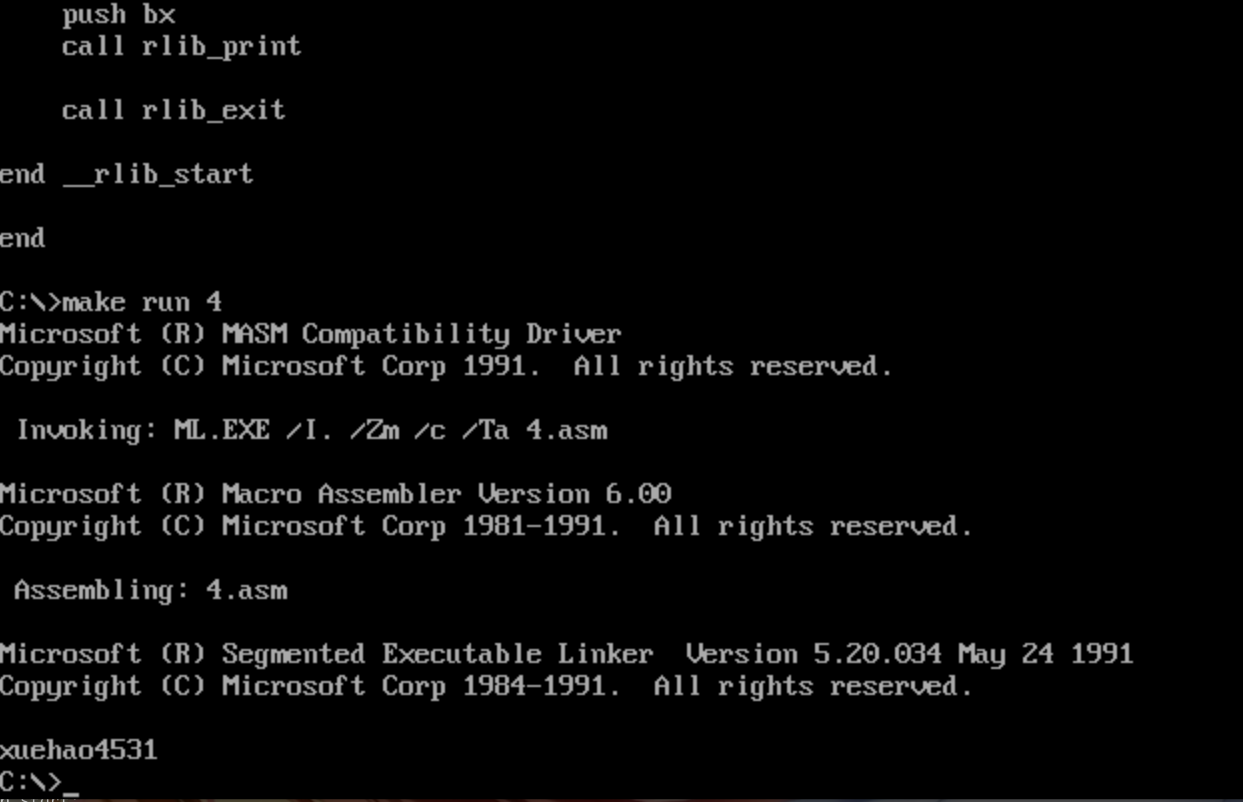
call rlib\_exit

end \_\_rlib\_start

end

### 实验记录与分析

下面分别使用4种方法进行写1字节内存测试。



显然，结果正确。

## 任务5

### 实验步骤

(1) 根据题目的要求编写程序，完成相应的功能。

(2) 将程序经过汇编，链接，确认没有错误后运行。

(3) 确认程序行为正常。

### 源程序

name rt5

.model small

.stack 1024

data segment use16

bname db 'bensongliu',0

bpass db 'test',0,0

n equ 30

s1 db 'shop1',0

ga1 db 'pen', 7 dup(0)

dw 35,56,70,25,?

ga2 db 'book', 6 dup(0)

dw 12,30,25,5,?

gan db n-2 dup( 'temp-value',15,0,20,0,30,0,2,0,?,?)

s2 db 'shop2',0

gb1 db 'book', 6 dup(0)

dw 12,28,20,15,?

gb2 db 'pen', 7 dup(0)

dw 35,50,30,24,?

m\_name db 12

db ?

db 12 dup (0)

m\_pwd db 8

db ?

db 8 dup(0)

m\_product db 12

db ?

db 12 dup(0)

pr1 dw 0

pr2 dw 0

pr\_sum dw 0

tmpbuf2 db 0ah,0dh,'$'

tmpbuf1 db 0h

info\_name db 'User: $'

info\_pswd db 'Password: $'

info\_invalid db 'Wrong password.$'

info\_prompt db 'The name of product ? $'

data ends

code segment use16

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

start:

jmp \_\_rlib\_start

include rlib.inc

tmp11:

lea bx,tmpbuf2

push bx

lea bx,info\_invalid

push bx

lea bx,tmpbuf2

push bx

call rlib\_print

call rlib\_print

call rlib\_print

jmp \_\_rlib\_start

tmp31:

jmp tmp7

tmp30:

cmp bx,1

je tmp6

; fallthrough warning

tmp8:

sub bx,1

jmp tmp22

tmp1:

mov ah,2

mov dx,0

mov dl,al

int 21h

lea ax,tmpbuf2

push ax

call rlib\_print

jmp \_\_rlib\_start

tmp24:

inc ch

mov cl,[bname+bx-1]

mov ch,1[di][bx]

cmp ch,cl

jne tmp11

dec si

jne tmp24

; fallthrough warning

tmp12:

mov di,offset m\_pwd

mov bx,di

mov bx,[bx+1]

mov bh,0

cmp bx,4

mov si,4

jne tmp11

tmp23:

mov ch,1[di][bx]

mov cl,[bpass+bx-1]

cmp ch,cl

je tmp29

jmp tmp11

tmp29:

dec si

jne tmp23

mov tmpbuf1,1

jmp tmp10

tmp10:

lea bx,tmpbuf2

push bx

lea bx,offset m\_product

push bx

lea bx,info\_prompt

push bx

lea bx,tmpbuf2

push bx

call rlib\_print

call rlib\_print

call rlib\_readstr

call rlib\_print

jmp tmp28

tmp27:

jmp tmp31

tmp28:

sub m\_product[1],0

je \_\_rlib\_start

; fallthrough warning

tmp9:

mov di,offset s1

inc di

mov bl,[m\_product+1]

mov si,offset m\_product

mov bh,0

cmp bx,10

add di,5

je tmp22

mov dx,[di+bx]

sub dx,0

je tmp22

jmp tmp27

tmp3:

mov cx,offset m\_name

add di,20

cmp cx,di

mov bh,0

mov bl,[m\_product+1]

jne tmp21

jmp tmp10

tmp16:

mov cx,1h

jcxz tmp17

jmp tmp18

push cx

call rlib\_print

tmp22:

mov cl,[di+bx-1]

inc cl

dec cl

mov ch,[si+bx+1]

je tmp25

cmp ch,cl

je tmp30

tmp25:

jmp tmp31

tmp7:

mov bl,[m\_product+1]

mov cx,offset s1

add di,20

cmp cx,di

mov bh,0

jne tmp22

jmp tmp10

tmp17:

mov bh,m\_name+1

cmp bh,1

jne tmp16

; warning: fallthrough

tmp18:

call rlib\_exit

jmp tmp11

tmp14:

mov tmpbuf1,0

jmp tmp10

tmp6:

mov bh, tmpbuf1

dec bh

je tmp26

mov ch,24h

mov bl,[1+m\_product]

mov bh,0

mov 2[si][bx],ch

lea bx,tmpbuf2

push bx

lea bx,m\_product+2

push bx

call rlib\_print

call rlib\_print

jmp \_\_rlib\_start

tmp26:

; not finished

call rlib\_exit

\_\_rlib\_start:

lea bx,data

mov ds,bx

lea bx,tmpbuf2

push bx

lea bx,offset m\_name

push bx

lea bx,info\_name

push bx

call rlib\_print

call rlib\_readstr

call rlib\_print

cmp m\_name[1],0

je tmp14

mov bh,m\_name+2

cmp bh,71h

je tmp17

cmp bh,0h

je tmp26

lea bx,offset m\_pwd

push bx

lea bx,info\_pswd

push bx

call rlib\_print

call rlib\_readstr

jmp tmp13

mov ax, 12h

clc

cmp dx, 1h

jbe tmp11

call rlib\_exit

tmp5:

tmp4:

mov si,offset m\_product

mov di,offset s2

mov bl,[m\_product+1]

mov bh,0

add di,6

cmp bx,10

je tmp21

mov dx,[di+bx]

inc dx

dec dx

jne tmp3

tmp21:

mov cl,[di+bx-1]

mov ch,[si+bx+1]

cmp ch,cl

jne tmp3

inc cl

dec cl

je tmp3

cmp bx,1

je tmp20

dec bx

jmp tmp21

tmp20:

; not finished

call rlib\_exit

tmp2:

cmp dx,5ah

mov ax,61h

jge tmp19

inc ax

cmp dx,50

jge tmp19

inc ax

cmp dx,20

jge tmp19

inc ax

cmp dx,0

jge tmp19

inc ax

tmp19:

jmp tmp1

test ax,1h

jne tmp13

call rlib\_readchar

tmp13:

mov di,offset m\_name

mov bx,di

add bx,1

mov si,10

cmp bx,0

je tmp18

mov dx,[bx]

mov dh,0

cmp si,dx

je tmp24

jmp tmp11

code ends

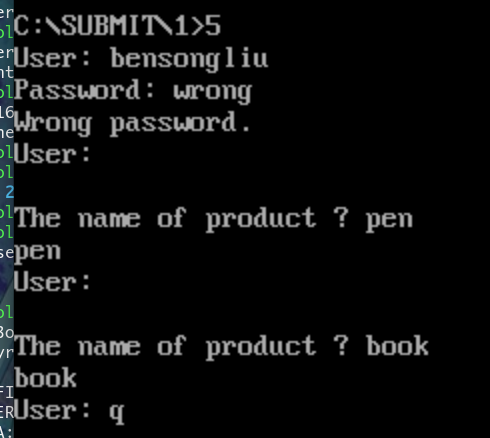
end start

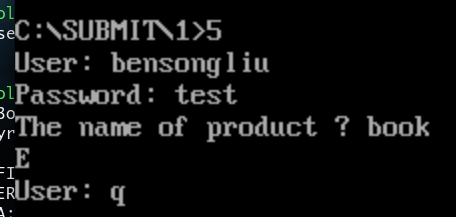
end

### 实验记录与分析

依次进行汇编，链接，然后运行。

功能演示如下 (正确密码为test图1为错误密码 图2为正确密码)





测试结果和预期一致。

# 总结与体会

通过实验中的任务1，我主要熟悉了调试工具的使用，通过在网上搜索帮助文档的方法，我掌握了在TD中实现运行程序，单步执行指令，修改指令，重新定向CS:IP等等功能的操作。同时通过向内存中的cs段实时地写入指令的方式了解了TD的各种监视工具，也加深了自己对于add，sub这两条指令以及和运算相关的标志位的理解，同时通过任务的引导，我注意到了在使用sub命令的时候，当a和b分别为有符号或是无符号数的时候，根据它们的大小不同，标志寄存器也会出现一定的规律，我的疑问在学习到了cmp的时候得到了解答，cmp正是根据了不同标志位的值来确定两个数的大小。

通过实验中的任务2，我主要熟悉了使用TD加载一个程序并且调试一个程序的方法，并且能更加熟练地观察不同寄存器的变化了以及数据段的内容了。在该任务中，我还学会了使用系统调用，通过系统调用实现从输入流读入字符，或者是在屏幕（标准输出流）中显示字符的功能。

通过实验中的任务3，我主要加深了对于不同寻址方式的认识，同时也理解了在内存中，程序和数据本都是二进制数，关键是如何让计算机解读这样的观点。我还了解了16位寄存器和32位寄存器之间的不同，以及有它们的不同所导致的寻址需要的字节数的扩大，这一点在指令的长度上可以反映出来。

通过任务5,我使用masm完成了高级语言应当完成的程序功能，学会了更灵活的使用汇编语言。

在本次实验中，我初步熟悉了dosbox的操作环境，学会了汇编工具，链接工具以及调试工具的使用，对符号，数字，寻址方式在计算机中的表现形式有了逐渐深入的理解。

# 参考文献

[1] Intel(R) 64 and IA-32 Architectures Software Developer’s Manual(https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf)

[2] (out-of-date) INTEL 80386 PROGRAMMER'S REFERENCE MANUAL 1986(https://css.csail.mit.edu/6.858/2015/readings/i386.pdf)

[3] (out-of-date) Open Watcom Toolset (http://www.openwatcom.org/)

[4] (out-of-date) Microsoft ASM Language for MS-DOS (https://en.wikipedia.org/wiki/Microsoft\_Macro\_Assembler)

[5] 80386 instruction set indexed by MIT.edu (https://pdos.csail.mit.edu/6.828/2017/readings/i386/c17.htm)

[6] (out-of-date) MASM directives (http://stanislavs.org/helppc/directives.html)

[7] (out-of-date) DOS Interrupts Reference by SCU.edu.au (http://spike.scu.edu.au/~barry/interrupts.html)

[8] DOSBox: DOS Simulator for modern computer(not a VM) (https://www.dosbox.com/wiki/)

[9] x86 arch introduction by wikibooks.org (https://en.wikibooks.org/wiki/X86\_Assembly/X86\_Architecture)