微机系统

实 验 报 告

**第一次实验**

（对应《汇编语言实验教程》 **实验一** “一、基础性实验”、“二、加强性实验”）

首先，在PC机使用MASM环境，**练习**“一、**基础性实验**” DEBUG的加载及其常用命令（D、E、F、R、G、A、T、U、N、L、W、Q）的使用情况。

“二、**加强性实验**”

（1）程序：

MOV AX,1000

MOV [1200],AX

MOV BX,1202

MOV BYTE PTR[BX],20

MOV DL,39

INC BX

MOV [BX],DL

DEC DL

MOV SI,3

MOV [BX+SI],DL

MOV [BX+SI+1],DL

MOV WORD PTR[BX+SI+2],1234

MOV AX,1000

MOV [1200],AX

MOV BX,1202

MOV BYTE PTR[BX],20

MOV DL,39

INC BX

MOV [BX],DL

DEC DL

MOV SI,3

MOV [BX+SI],DL

MOV [BX+SI+1],DL

MOV WORD PTR[BX+SI+2],1234

MOV AL,06

SHL AL,1

MOV BL,AL

MOV CL,2

SHL AL,CL

ADD,AL,BL

mov si,1000

mov di,2000

mov cx,0f

lop:mov al,[si]

mov [di],al

inc si

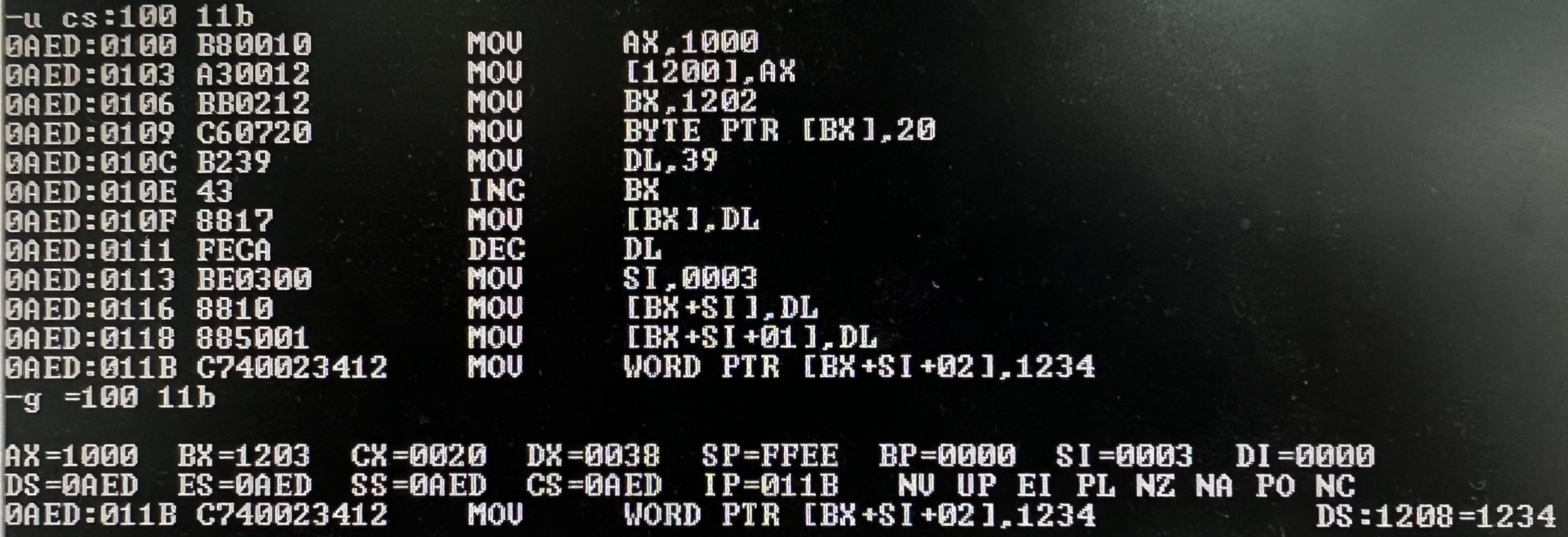
inc di

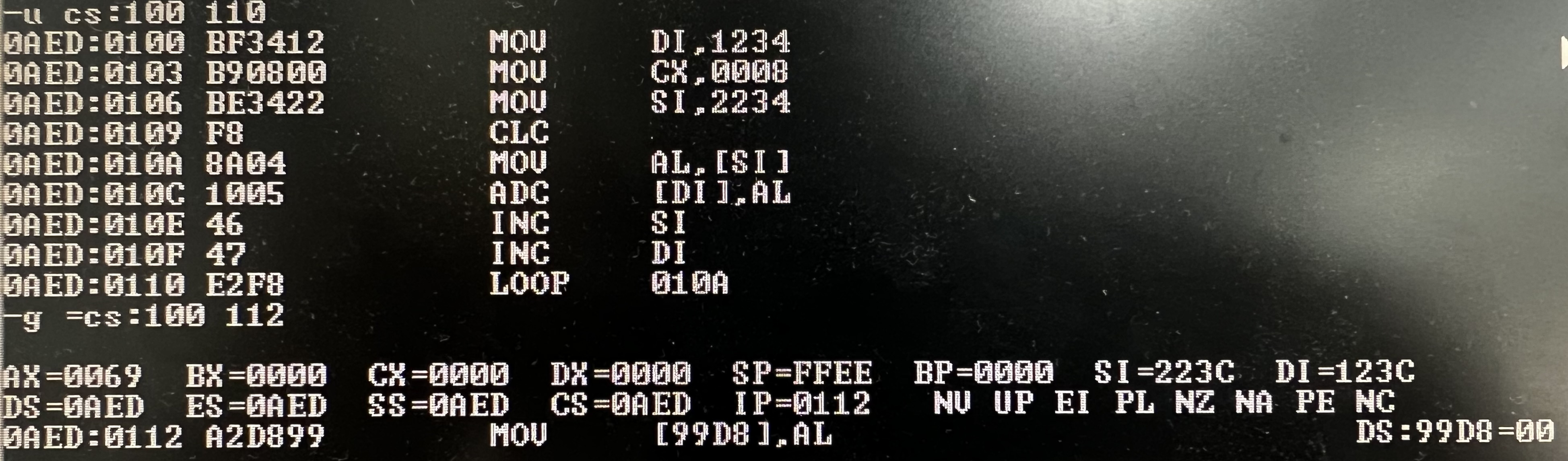
loop lop

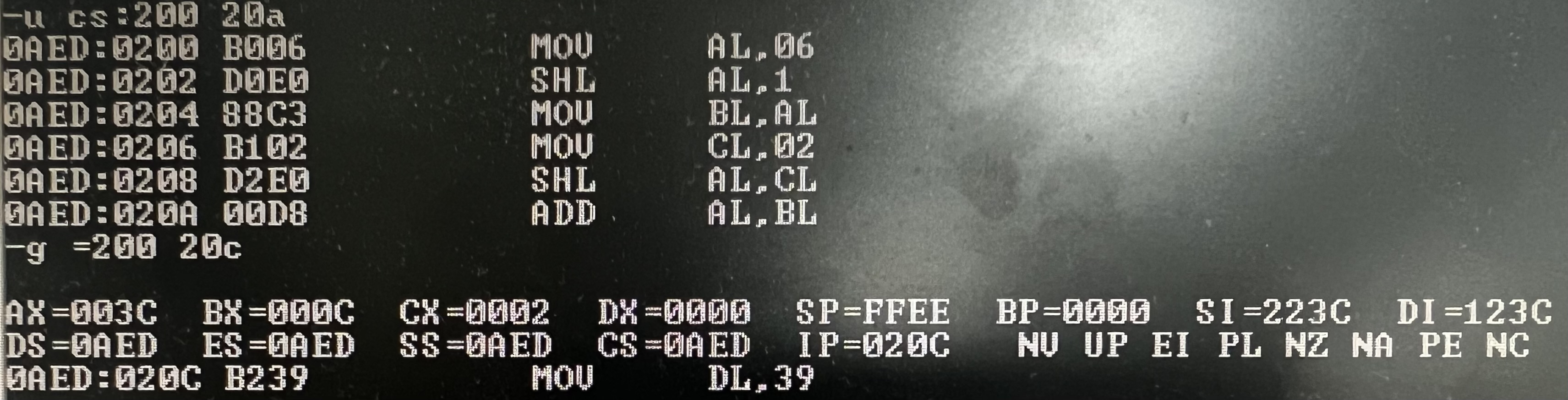
dec cx

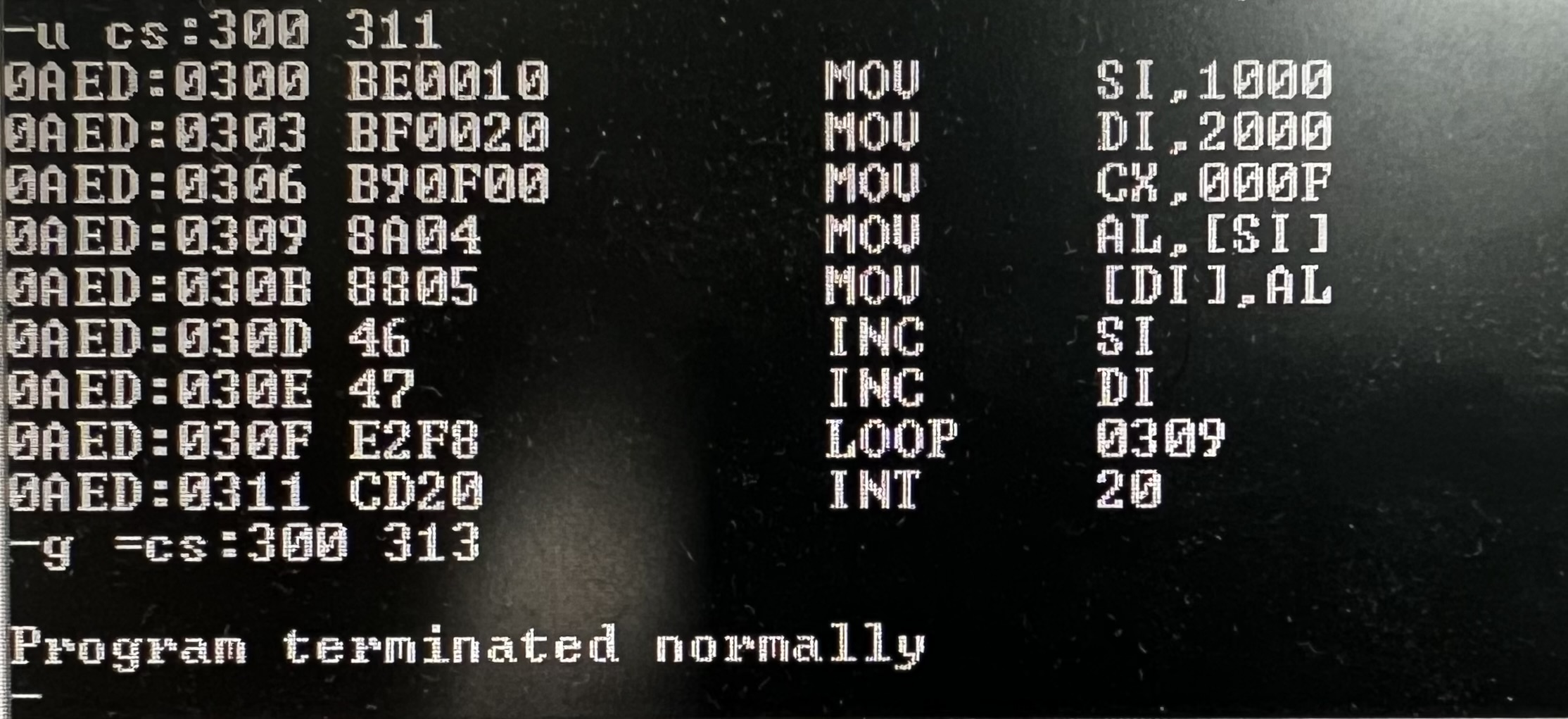
int 20

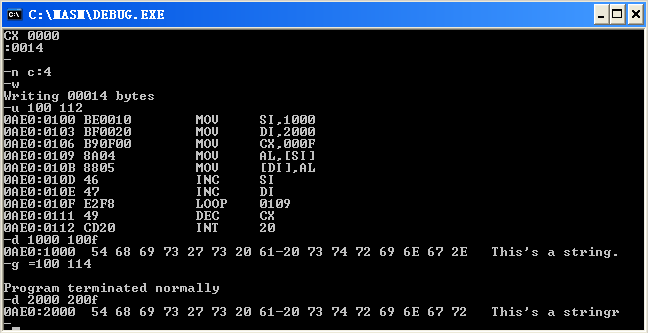
（2）实验结果（截图）：











（3）“思考题”：

1.

指令：**INT 3**

**无符号数：204**

**有符号数：-52**

**进入DEBUG环境，输入命令D [地址]，例如：D 100，来查看内存地址100H处的内容。**

**根据查看到的内存内容，将CCH转换为相应的二进制值。**

**使用适当的指令系统文档或教材，查找二进制值对应的指令编码。**

**解析指令编码，确定CCH表示的是什么指令及其相应的操作。**

**2.**

**指令"MOV [BX], AX"使用了基址寻址（Base Addressing）方式来寻址操作数"[BX]"。**

**在DEBUG环境中执行指令。输入"T"命令进行单步执行，或输入"G"命令进行整体执行，使得该指令被执行。**

**执行完指令后，可以使用"D"命令来查看内存中的结果。输入"D [BX]"命令，DEBUG将会显示以BX寄存器中的值为基址的内存内容**

**3.**

**在DEBUG提示符"-"后发出不带参数的命令"G"是合法的，它将继续执行程序，从上一次的执行点继续执行。这可以在不更改程序执行地址的情况下，继续程序的正常执行。**

**然而，在某些情况下，可以省略使用"="给出执行的首地址：**

**当程序的执行点已经被设置为一个地址（通过调试命令或先前的执行），并且你想继续执行程序时，可以直接使用命令"G"，而不必指定新的执行地址。**

**在执行过程中，当遇到断点（通过调试命令设置的停止点）时，可以直接使用命令"G"来继续执行程序，而不必再次指定执行的首地址。**

**第二次实验**

（对应《汇编语言实验教程》 **实验二** “一、基础性实验”、“二、加强性实验”）

首先，在PC机使用MASM5.0环境，**练习并熟练掌握**“一、**基础性实验**”的全部操作内容。

“一、**基础性实验**”

（1）程序：

DATA SEGMENT

DD1 DB 35H，78H，85H

DD2 DB ?

DATA ENDS

CODE SEGMENT

MAIN PROC FAR

ASSUME CS：CODE，DS：DATA

START： PUSH DS

SUB AX，AX

PUSH AX

MOV AX，DATA

MOV DS，AX

MOV AL，DD1

CMP AL，DD1+1

JA AAA1

MOV AL，DD1+1

AAA1： CMP AL，DD1+2

JA AAA2

MOV AL，DD1+2

AAA2： MOV DD2，AL

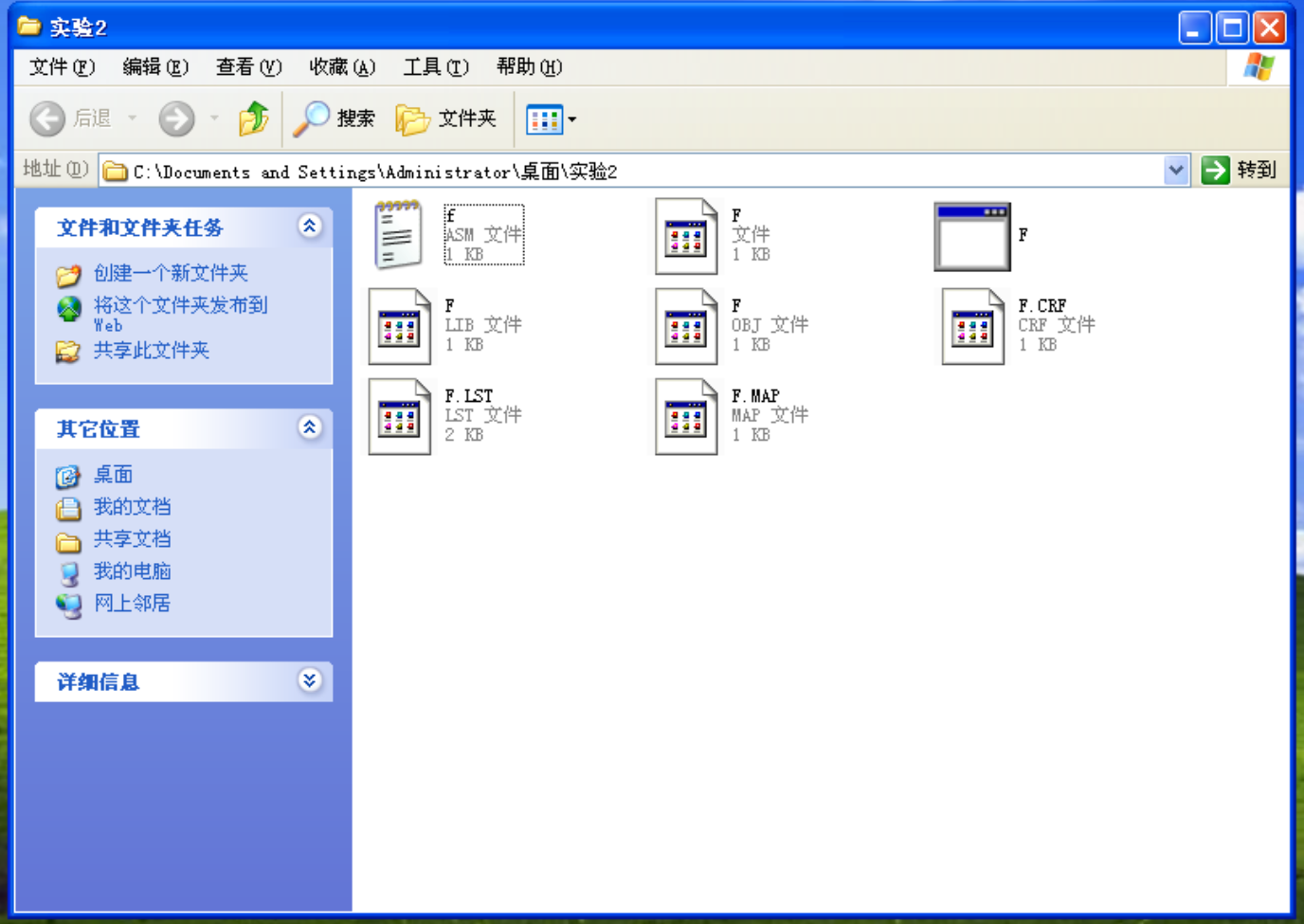
RET

MAIN ENDP

CODE ENDS

END START

（2）实验结果（截图）：



（3）“思考题”：

**1.**

**在汇编语言程序中，语句END后的标号是一个可选的程序标号，用于标识程序的结束点。它并不会被汇编器或处理器用于执行或寻址，主要有两个作用：**

**1.标识程序结束点：END语句后的标号可以用于标识程序的结束点，表示代码的执行已经到达了程序的最后。这对于程序员和阅读代码的人来说是一个清晰的标识，以指示程序的逻辑结构。**

**2.提供跳转目标：标号可以被其他指令或指令中的跳转指令引用。例如，程序中的某个地方需要跳转到程序的结束处，可以使用该标号作为跳转的目标。**

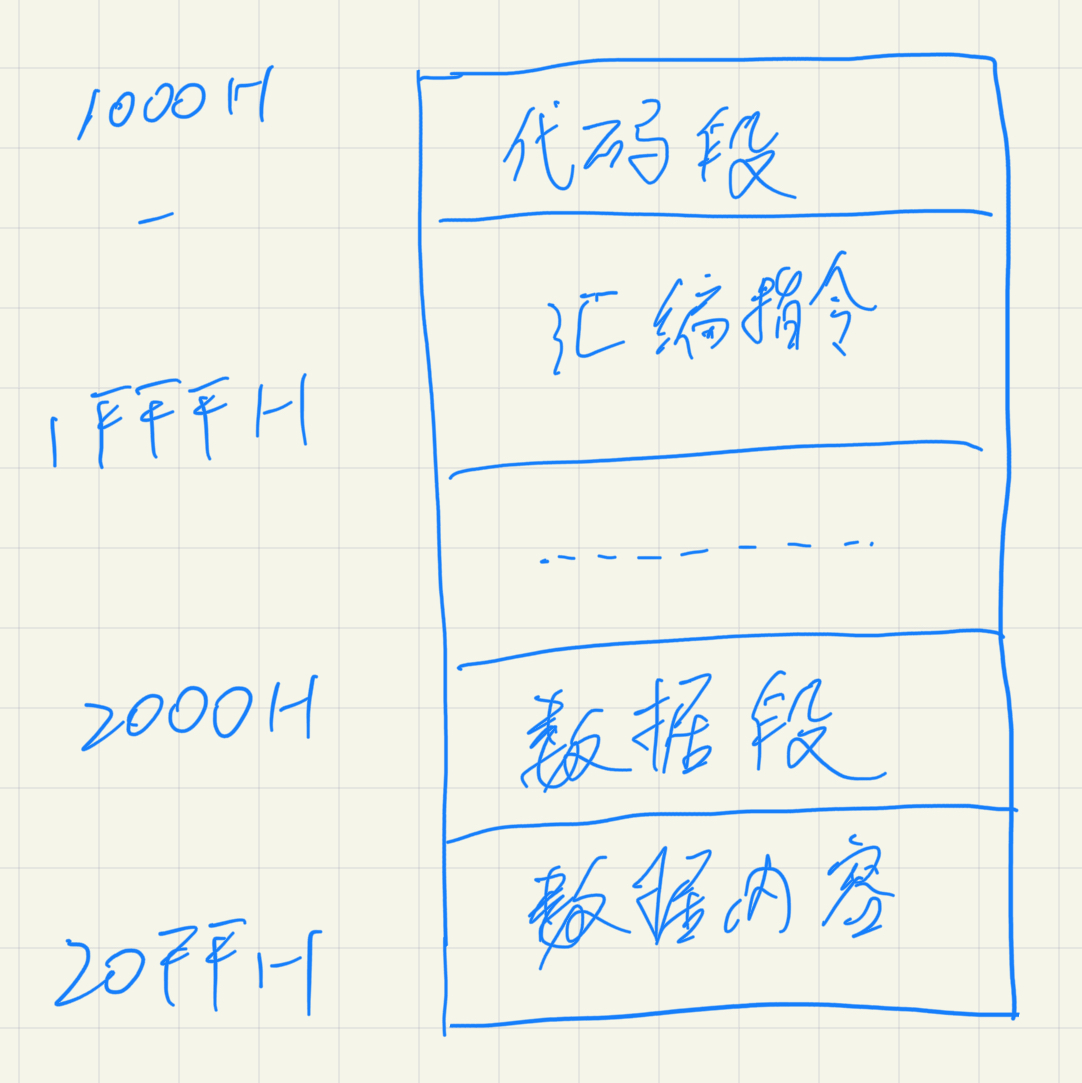
**2.**

**一个段的大小不一定是固定的64KB。在x86架构的实模式下，一个段的大小可以达到最大的64KB（即16位地址空间的最大大小），但在保护模式下，一个段的大小可以远远超过64KB。**

**1.使用"L"命令加载程序：在DEBUG提示符下输入"L program.com"（假设程序名为program.com），以加载程序到内存中。**

**2.使用"D"命令查看内存内容：输入"D"命令来查看内存中的内容。你可以通过指定内存地址范围来查看程序在内存中的存放情况。例如，输入"D 1000H"可以查看地址为1000H开始的内存内容。**

**3.使用"U"命令将机器代码转换为汇编指令：输入"U"命令可以将内存中的机器代码转换为汇编指令。例如，输入"U 1000H"可以将地址为1000H开始的机器代码转换为汇编指令，以便更好地理解程序的逻辑。**



“二、**加强性实验**”

（1）程序：

DATA SEGMENT

NOTIC DB "Please input the word !",0AH,0DH

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

START: MOV AX,DATA

MOV DS,AX

MOV CX,19H

LEA BX,[NOTIC]

AA0: MOV DL,[BX] ;输出提示

MOV AH,2

INT 21H

INC BX

LOOP AA0

AA1: MOV AH,1 ;读入回显

INT 21H

CMP AL,1BH ;是ESC 跳转

JZ AA3 ;判断是小写就转换 否则不变

CMP AL,61H ;判断是不是小写

JS AA2

CMP AL,7AH

JNS AA2

SUB AL,20H ;转换成大写

AA2: MOV DL,AL

MOV AH,2

INT 21H

LOOP AA1

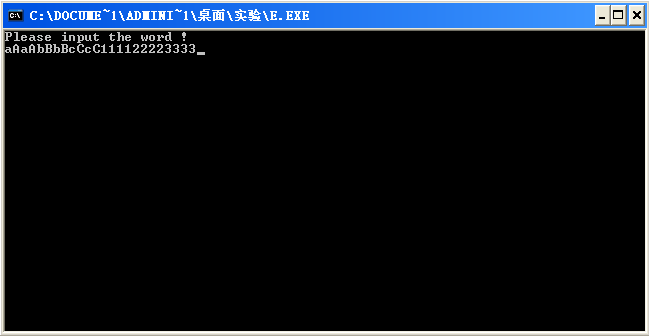
AA3: MOV AH,4CH

INT 21H

CODE ENDS

END START

（2）实验结果（截图）：



（3）“思考题”：

**1.**

**如果在源程序中省略了 INT 21H 的 'H'，会导致语法错误。指令 INT 21H 是用于调用 DOS 功能的中断指令，其中的 'H' 表示使用十六进制数作为参数。省略 'H' 将使指令不完整，编译器无法正确解析该指令，从而导致编译错误。**

**2.**

**将 INT 21H 4CH 功能改为 INT 20H 是不可行的。INT 21H 是用于调用 DOS 功能的中断指令，而 INT 20H 是用于终止程序的中断指令。这两个中断具有完全不同的功能。使用 INT 20H 中断将导致程序被终止，并且不会执行后续的指令。因此，将 INT 21H 4CH 改为 INT 20H 是不适当的，它会导致程序意外终止。**

（对应《汇编语言实验教程》 **实验三** “一、基础性实验”、“二、加强性实验”）

“一、**基础性实验**”

（1）程序：

DSEG SEGMENT PARA PUBLIC 'DSEG'; 定义数据段DSEG，用于存储数据

M DB 1H,2H,3H,4H,5H,6H,7H,8H,9H,10H,-1H,-2H,-3H,-4H,-5H,-6H,-7H,-8H,-9H,-10H, ;填写实际需要分组的数值

P DB 20 DUP(0)

N DB 20 DUP(1)

DSEG ENDS ; 数据段DSEG定义结束

CODE SEGMENT ; 定义代码段CODE

ASSUME CS:CODE,DS:DSEG ; 设置代码段和数据段的段寄存器

START:

MOV AX,DSEG ; 将数据段DSEG的段地址加载到寄存器AX中

MOV DS,AX ; 将DS寄存器设置为数据段DSEG的段地址

LEA SI,P ; 将正数数组P的首地址加载到寄存器SI中

LEA DI,N ; 将负数数组N的首地址加载到寄存器DI中

LEA BX,M ; 将输入数组M的首地址加载到寄存器BX中

XOR AX,AX ; 将AX寄存器清零

XOR DX,DX ; 将DX寄存器清零

MOV CX,20 ; 将CX寄存器设置为20，即循环次数为20

L1: MOV AL,[BX] ; 将BX寄存器指向的内存地址中的值加载到AL寄存器中

TEST AL,80H ; 将AL寄存器与80H进行逻辑与运算，检查最高位是否为1

JZ L2 ; 如果最高位为0，则跳转到标签L2

MOV [DI],AL ; 将AL寄存器的值存储到DI寄存器指向的内存地址中，即放入负数数组N

INC BX ; 增加BX的值，指向下一个输入数组的元素

INC DI ; 增加DI的值，指向下一个负数数组的位置

INC DH ; 增加DH的值，即负数个数增加

JMP L3 ; 跳转到标签L3

L2: MOV [SI],AL ; 将AL寄存器的值存储到SI寄存器指向的内存地址中，即放入正数数组P

INC BX ; 增加BX的值，指向下一个输入数组的元素

INC SI ; 增加SI的值，指向下一个正数数组的位置

INC DL ; 增加DL的值，即正数个数增加

L3: LOOP L1 ; 循环，继续执行标签L1处的指令，直到CX寄存器的值为0

MOV CX,2 ; 将CX寄存器设置为2，用于后续输出操作

L5: MOV BL,DL ; 将DL寄存器的值复制到BL寄存器中，然后每4位输出

SHR DL,1 ; 将DL寄存器右移1位，相当于除以2

SHR DL,1 ; 将DL寄存器右移1位

SHR DL,1 ; 将DL寄存器右移1位

SHR DL,1 ; 将DL寄存器右移1位

AND DL,0FH ; 将DL寄存器与0FH进行逻辑与运算，保留低4位的值

CMP DL,10 ; 将DL寄存器的值与10进行比较

JB L4 ; 如果DL的值小于10，则跳转到标签L4

ADD DL,7 ; 如果DL的值大于等于10，加7进行调整后输出

L4: ADD DL,30H ; 将DL的值加上30H，即将数字转换为ASCII码值进行输出

MOV AH,2 ; 将AH寄存器设置为2，表示进行字符输出

INT 21H ; 调用21H中断，实现字符输出

MOV DL,BL ; 将BL寄存器的值复制到DL寄存器中

AND DL,0FH ; 将DL寄存器与0FH进行逻辑与运算，保留低4位的值

CMP DL,10 ; 将DL寄存器的值与10进行比较

JB L6 ; 如果DL的值小于10，则跳转到标签L6

ADD DL,7 ; 如果DL的值大于等于10，加7进行调整后输出

L6: ADD DL,30H ; 将DL的值加上30H，即将数字转换为ASCII码值进行输出

MOV AH,2 ; 将AH寄存器设置为2，表示进行字符输出

INT 21H ; 调用21H中断，实现字符输出

MOV DL,0AH ; 将DL寄存器设置为换行符的ASCII码值

MOV AH,2 ; 将AH寄存器设置为2，表示进行字符输出

INT 21H ; 调用21H中断，实现字符输出

MOV DL,0DH ; 将DL寄存器设置为回车符的ASCII码值

MOV AH,2 ; 将AH寄存器设置为2，表示进行字符输出

INT 21H ; 调用21H中断，实现字符输出

MOV DL,DH ; 将DH的值复制到DL寄存器中，即负数个数进行输出

LOOP L5 ; 循环，继续执行标签L5处的指令，直到CX寄存器的值为0

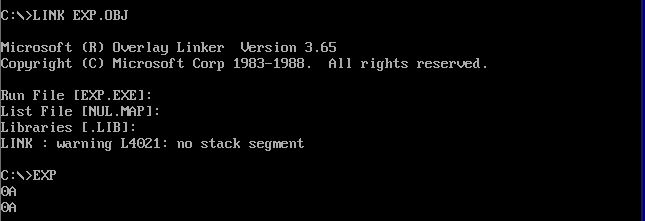
MOV AH,4CH ; 将AH寄存器设置为4CH，表示程序退出

INT 21H ; 调用21H中断，实现程序退出

CODE ENDS ; 代码段CODE定义结束

END START ; 程序结束

（2）实验结果（截图）：



（3）“思考题”：

**在给定的示例程序中，条件转移使用了CMP指令来比较寄存器的值。但是，并不一定要使用CMP指令来实现条件转移，还可以使用其他指令来达到相同的效果。一个常见的替代方案是使用测试位指令（TEST）和条件跳转指令（JZ、JNZ、JB、JBE等）来实现条件转移。这些指令可以通过逻辑运算和位测试来确定条件是否满足，并根据结果进行跳转。下面是一个使用TEST和JZ指令替代CMP指令的示例代码片段：**

L1: MOV AL, [BX]

TEST AL, 80H

JZ L2

MOV [DI], AL

INC BX

INC DI

INC DH

JMP L3

L2: MOV [SI], AL

INC BX

INC SI

INC DL

L3: LOOP L1

**在上述代码中，我们用TEST指令测试AL寄存器与80H的逻辑与结果，并根据测试结果使用JZ指令进行条件跳转。如果AL的最高位为0，表示正数，执行L2的逻辑；如果最高位为1，表示负数，执行L1中的逻辑。**

“二、**加强性实验**”

（1）程序：

DATA SEGMENT

DIGIT DB "DIGIT",0AH,0DH ; 存储"数字"字符串，0AH和0DH是换行符

LETTER DB "LETTER",0AH,0DH ; 存储"字母"字符串，0AH和0DH是换行符

OTHER DB "OTHER",0AH,0DH ; 存储"其他"字符串，0AH和0DH是换行符

DATA ENDS

CODE SEGMENT ; 代码段开始

ASSUME CS:CODE,DS:DATA ; 设置代码段寄存器CS和数据段寄存器DS的默认值为CODE和DATA

START: MOV AX,DATA ; 将DATA段的起始地址加载到AX寄存器

MOV DS,AX ; 将AX中的值移动到DS寄存器，将DS设置为DATA段的段地址

MOV AH,07 ; 设置AH寄存器为07H，表示输入字符但不回显

INT 21H ; 调用21H中断，等待用户输入字符，结果存储在AL寄存器

CMP AL,30H ; 将AL寄存器的值与ASCII码为30H的字符('0')进行比较

JB AA1 ; 如果小于'0'，跳转到标号AA1

CMP AL,39H ; 将AL寄存器的值与ASCII码为39H的字符('9')进行比较

JA AA1 ; 如果大于'9'，跳转到标号AA1

;非数字就跳转到AA1

;是数字就执行下面的代码

MOV CX,7 ; 将CX寄存器设置为7，用于循环显示"数字"

LEA SI,DIGIT ; 将DIGIT字符串的偏移地址加载到SI寄存器

AA0: MOV DL,[SI] ; 将SI指向的内存位置中的值移动到DL寄存器，即获取字符串中的一个字符

MOV AH,2 ; 设置AH寄存器为2，表示在屏幕上显示一个字符

INT 21H ; 调用21H中断，显示DL寄存器中的字符

INC SI ; 递增SI寄存器的值，移动到下一个字符

LOOP AA0 ; 循环执行AA0标号处的代码，循环次数为CX寄存器的值，即显示"数字"字符串的所有字符

JMP BB ; 跳转到标号BB，跳过后续的判断和显示代码

;非数字，看看是不是大写字母

AA1: CMP AL,41H ; 将AL寄存器的值与ASCII码为41H的字符('A')进行比较

JB AA3 ; 如果小于'A'，跳转到标号AA3

CMP AL,5AH ; 将AL寄存器的值与ASCII码为5AH的字符('Z')进行比较

JA AA2 ; 如果大于'Z'，跳转到标号AA2

LEA SI,LETTER ; 将LETTER字符串的偏移地址加载到SI寄存器

MOV CX,8 ; 将CX寄存器设置为8，用于循环显示"字母"

AA4: MOV DL,[SI] ; 将SI指向的内存位置中的值移动到DL寄存器，即获取字符串中的一个字符

MOV AH,2 ; 设置AH寄存器为2，表示在屏幕上显示一个字符

INT 21H ; 调用21H中断，显示DL寄存器中的字符

INC SI ; 递增SI寄存器的值，移动到下一个字符

LOOP AA4 ; 循环执行AA4标号处的代码，循环次数为CX寄存器的值，即显示"字母"字符串的所有字符

JMP BB ; 跳转到标号BB，跳过后续的判断和显示代码

;非数字且非大写字母看看是不是小写字母

AA2: CMP AL,61H ; 将AL寄存器的值与ASCII码为61H的字符('a')进行比较

JB AA3 ; 如果小于'a'，跳转到标号AA3

CMP AL,7AH ; 将AL寄存器的值与ASCII码为7AH的字符('z')进行比较

JA AA3 ; 如果大于'z'，跳转到标号AA3

LEA SI,LETTER ; 将LETTER字符串的偏移地址加载到SI寄存器

MOV CX,8 ; 将CX寄存器设置为8，用于循环显示"字母"

AA5: MOV DL,[SI] ; 将SI指向的内存位置中的值移动到DL寄存器，即获取字符串中的一个字符

MOV AH,2 ; 设置AH寄存器为2，表示在屏幕上显示一个字符

INT 21H ; 调用21H中断，显示DL寄存器中的字符

INC SI ; 递增SI寄存器的值，移动到下一个字符

LOOP AA5 ; 循环执行AA5标号处的代码，循环次数为CX寄存器的值，即显示"字母"字符串的所有字符

JMP BB ; 跳转到标号BB，跳过后续的判断和显示代码

AA3: LEA SI,OTHER ; 将OTHER字符串的偏移地址加载到SI寄存器

MOV CX,7 ; 将CX寄存器设置为7，用于循环显示"其他"

AA6: MOV DL,[SI] ; 将SI指向的内存位置中的值移动到DL寄存器，即获取字符串中的一个字符

MOV AH,2 ; 设置AH寄存器为2，表示在屏幕上显示一个字符

INT 21H ; 调用21H中断，显示DL寄存器中的字符

INC SI ; 递增SI寄存器的值，移动到下一个字符

LOOP AA6 ; 循环执行AA6标号处的代码，循环次数为CX寄存器的值，即显示"其他"字符串的所有字符

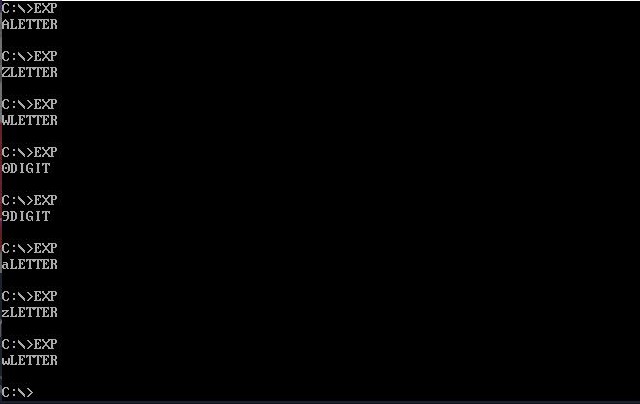
BB: MOV AH,4CH ; 设置AH寄存器为4CH，表示程序的正常退出

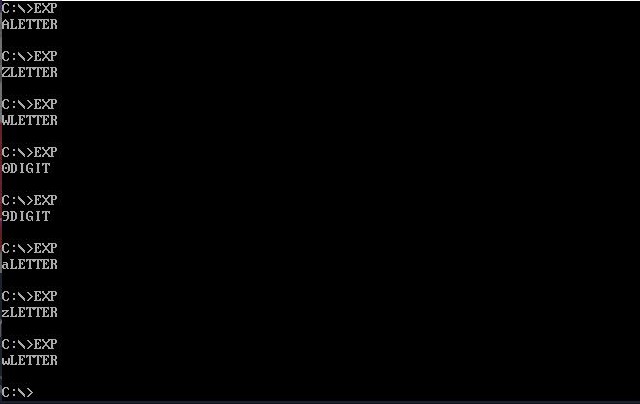
INT 21H ; 调用21H中断，结束程序的执行

CODE ENDS ; 代码段结束

END START ; 程序的入口点为START标号处，程序结束

（2）实验结果（截图）：





（3）“思考题”：

**若需要进一步区分大写字母和小写字母，可以在非数字且非大写字母的判断分支AA2处再添加一段判断代码。以下是修改后的程序：**

DATA SEGMENT

DIGIT DB "DIGIT",0AH,0DH ; 存储"数字"字符串，0AH和0DH是换行符

UPPERCASE DB "UPPERCASE",0AH,0DH ; 存储"大写字母"字符串，0AH和0DH是换行符

LOWERCASE DB "LOWERCASE",0AH,0DH ; 存储"小写字母"字符串，0AH和0DH是换行符

OTHER DB "OTHER",0AH,0DH ; 存储"其他"字符串，0AH和0DH是换行符

DATA ENDS

CODE SEGMENT ; 代码段开始

ASSUME CS:CODE,DS:DATA ; 设置代码段寄存器CS和数据段寄存器DS的默认值为CODE和DATA

START: MOV AX,DATA ; 将DATA段的起始地址加载到AX寄存器

MOV DS,AX ; 将AX中的值移动到DS寄存器，将DS设置为DATA段的段地址

MOV AH,07 ; 设置AH寄存器为07H，表示输入字符但不回显

INT 21H ; 调用21H中断，等待用户输入字符，结果存储在AL寄存器

CMP AL,30H ; 将AL寄存器的值与ASCII码为30H的字符('0')进行比较

JB AA1 ; 如果小于'0'，跳转到标号AA1

CMP AL,39H ; 将AL寄存器的值与ASCII码为39H的字符('9')进行比较

JA AA1 ; 如果大于'9'，跳转到标号AA1

; 是数字，显示"数字"

MOV CX,7 ; 将CX寄存器设置为7，用于循环显示"数字"

LEA SI,DIGIT ; 将DIGIT字符串的偏移地址加载到SI寄存器

AA0: MOV DL,[SI] ; 将SI指向的内存位置中的值移动到DL寄存器，即获取字符串中的一个字符

MOV AH,2 ; 设置AH寄存器为2，表示在屏幕上显示一个字符

INT 21H ; 调用21H中断，显示DL寄存器中的字符

INC SI ; 递增SI寄存器的值，移动到下一个字符

LOOP AA0 ; 循环执行AA0标号处的代码，循环次数为CX寄存器的值，即显示"数字"字符串的所有字符

JMP BB ; 跳转到标号BB，跳过后续的判断和显示代码

; 非数字，判断是否大写字母

AA1: CMP AL,41H ; 将AL寄存器的值与ASCII码为41H的字符('A')进行比较

JB AA2 ;

如果小于'A'，跳转到标号AA2

CMP AL,5AH ; 将AL寄存器的值与ASCII码为5AH的字符('Z')进行比较

JBE UPPERCASE ; 如果小于等于'Z'，跳转到标号UPPERCASE

JMP AA3 ; 否则，跳转到标号AA3

; 是大写字母，显示"大写字母"

UPPERCASE:

MOV CX,9 ; 将CX寄存器设置为9，用于循环显示"大写字母"

LEA SI,UPPERCASE ; 将UPPERCASE字符串的偏移地址加载到SI寄存器

AA4: MOV DL,[SI] ; 将SI指向的内存位置中的值移动到DL寄存器，即获取字符串中的一个字符

MOV AH,2 ; 设置AH寄存器为2，表示在屏幕上显示一个字符

INT 21H ; 调用21H中断，显示DL寄存器中的字符

INC SI ; 递增SI寄存器的值，移动到下一个字符

LOOP AA4 ; 循环执行AA4标号处的代码，循环次数为CX寄存器的值，即显示"大写字母"字符串的所有字符

JMP BB ; 跳转到标号BB，跳过后续的判断和显示代码

; 非数字且非大写字母，判断是否小写字母

AA2: CMP AL,61H ; 将AL寄存器的值与ASCII码为61H的字符('a')进行比较

JB AA3 ; 如果小于'a'，跳转到标号AA3

CMP AL,7AH ; 将AL寄存器的值与ASCII码为7AH的字符('z')进行比较

JBE LOWERCASE ; 如果小于等于'z'，跳转到标号LOWERCASE

JMP AA3 ; 否则，跳转到标号AA3

; 是小写字母，显示"小写字母"

LOWERCASE:

MOV CX,9 ; 将CX寄存器设置为9，用于循环显示"小写字母"

LEA SI,LOWERCASE ; 将LOWERCASE字符串的偏移地址加载到SI寄存器

AA5: MOV DL,[SI] ; 将SI指向的内存位置中的值移动到DL寄存器，即获取字符串中的一个字符

MOV AH,2 ; 设置AH寄存器为2，表示在屏幕上显示一个字符

INT 21H ; 调用21H中断，显示DL寄存器中的字符

INC SI ; 递增SI寄存器的值，移动到下一个字符

LOOP AA5 ; 循环执行AA5标号处的代码，循环次数为CX寄存器的值，即显示"小写字母"字符串的所有字符

JMP BB ; 跳转到标号BB，跳过后续的

判断和显示代码

; 非数字且非大写字母且非小写字母，显示"其他"

AA3: MOV CX,7 ; 将CX寄存器设置为7，用于循环显示"其他"

LEA SI,OTHER ; 将OTHER字符串的偏移地址加载到SI寄存器

AA6: MOV DL,[SI] ; 将SI指向的内存位置中的值移动到DL寄存器，即获取字符串中的一个字符

MOV AH,2 ; 设置AH寄存器为2，表示在屏幕上显示一个字符

INT 21H ; 调用21H中断，显示DL寄存器中的字符

INC SI ; 递增SI寄存器的值，移动到下一个字符

LOOP AA6 ; 循环执行AA6标号处的代码，循环次数为CX寄存器的值，即显示"其他"字符串的所有字符

BB: MOV AH,4CH ; 设置AH寄存器为4CH，表示程序的正常退出

INT 21H ; 调用21H中断，结束程序的执行

CODE ENDS ; 代码段结束

END START ; 程序的入口点为START标号处，程序结束

```

**在上述修改的程序中，新增了两个标号：`UPPERCASE`和`LOWERCASE`，分别用于显示大写字母和小写字母。根据判断条件，如果输入的字符在大写字母范围内，则跳转到标号`UPPERCASE`，执行显示大写字母的代码；如果输入的字符在小写字母范围内，则跳转到标号`LOWERCASE`，执行显示小写字母的代码。这样可以区分并显示不同类型的字母字符。**

**第三次实验**

（对应《汇编语言实验教程》 **实验四** “一、基础性实验”、“二、加强性实验”）

“一、**基础性实验**”

（1）程序：

DATA SEGMENT

COURSE1 DB 70H,88H,92H,90H,99H ; 第一门课成绩

DB 67H,77H,88H,76H,69H ; 第二门课成绩

DB 74H,87H,77H,74H,70H ; 第三门课成绩

DB 99H,97H,94H,98H,96H ; 第四门课成绩

NUM1 DW 5 DUP(0) ; 用于存储学生总分的数组

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

START: MOV AX,DATA ; 将数据段地址赋值给AX寄存器

MOV DS,AX ; 将AX寄存器的值赋给DS寄存器，设置数据段寄存器

LEA SI,COURSE1 ; 将COURSE1的偏移地址赋给SI寄存器

LEA DI, NUM1 ; 将NUM1的偏移地址赋给DI寄存器

SUB SI,5 ; 将SI寄存器的值减去5，以便循环

MOV CL,5 ; 设置外层循环计数器初值为5

AA1: MOV BX,SI ; 将SI的值赋给BX寄存器，形成某个学生第一门课成绩的地址减去5

SUB AX,AX ; 清空AX寄存器的值

MOV CH,4 ; 设置内层循环计数器初值为4

AA2: ADD BX,5 ; 将BX的值增加5，形成某个学生的1、2、...、4门课成绩的地址

ADD AL,[BX] ; 将[BX]内存中的值加到AL寄存器中，按BCD码求和

DAA ; 压缩BCD码并进行调整

ADC AH,0 ; 将进位加到AH寄存器中

DEC CH ; 内层循环计数器减1

JNZ AA2 ; 如果内层循环计数器不为0，跳转到AA2标签处继续循环

ADD [DI],AX ; 将AX寄存器中的值存储到DI所指向的内存地址（保存总分）

INC SI ; 将SI寄存器的值加1，形成下一个学生第一门课成绩的地址减去5

ADD DI,2 ; 将DI寄存器的值增加2，形成下一个学生的总分地址

DEC CL ; 外层循环计数器减1

JNZ AA1 ; 如果外层循环计数器不为0，跳转到AA1标签处继续循环

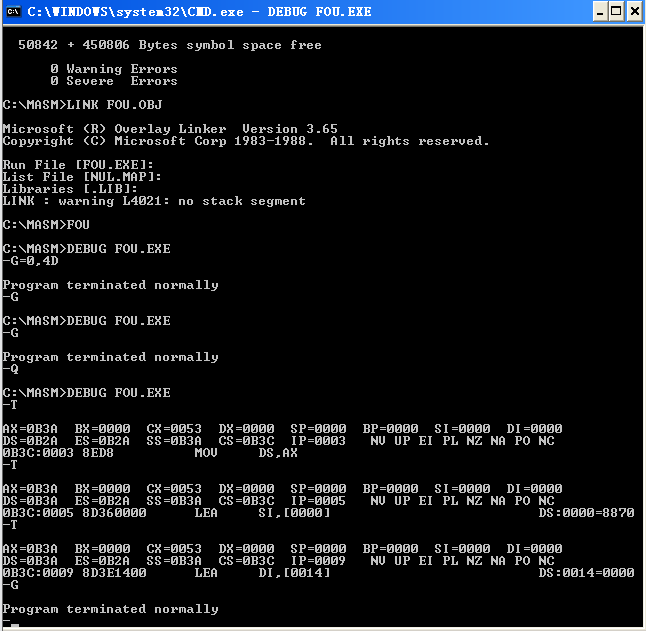
MOV AH,4CH ; 设置AH寄存器的值为4CH，表示程序终止

INT 21H ; 调用21H中断（DOS功能调用）

CODE ENDS

END START

（2）实验结果（截图）：



（3）“思考题”：

**1.**

**在示例代码中，`JMP`指令用于无条件跳转到指定的标号处。它可以改变程序的执行流程，使程序跳过一些指令或者重复执行一段代码。在代码中，`JMP`指令被用于控制程序的循环执行。具体来说，在示例代码中，`JMP`指令用于两个地方：**

**1. `JMP BB`：在判断和显示代码执行完毕后，通过`JMP BB`跳转到标号`BB`处，以跳过后续的判断和显示代码，直接执行程序的退出部分。**

**2. `JMP AA1`：在内层循环结束后，通过`JMP AA1`跳转到标号`AA1`处，以继续外层循环的执行，实现对所有学生的成绩求和的功能。**

**2.**

**错误：在emu8086上运行有bug，会导致第二个人的成绩计算错误。**

**排除：在masm上运行即可解决。**

“二、**加强性实验**”

（1）程序：

DATA SEGMENT

MATRIX1 DB 3H,4H,15H,8H ; 定义MATRIX1矩阵，包含4行4列的字节数据

DB 4H,5H,6H,17H

DB 8H,9H,3H,2H

DB 1H,1H,4H,6H

MATRIX2 DB 0F3H,0F9H,8H,0E6H ; 定义MATRIX2矩阵，包含1行4列的字节数据

UNIT DW 4 DUP(0) ; 定义UNIT数组，包含4个字的空间，初始值为0

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

START: MOV AX,DATA ; 将数据段地址赋值给AX寄存器

MOV DS,AX ; 将AX寄存器的值赋给DS寄存器，设置数据段寄存器

LEA SI,MATRIX1 ; 将MATRIX1的偏移地址赋给SI寄存器

LEA DI,MATRIX2 ; 将MATRIX2的偏移地址赋给DI寄存器

LEA BX,UNIT ; 将UNIT的偏移地址赋给BX寄存器

MOV CL,4H ; 设置外层循环计数器初值为4

AA0: SUB AX,AX ; 清空AX寄存器的值

MOV CH,4H ; 设置内层循环计数器初值为4

AA1: MOV DX,AX ; 将AX的值存储到DX寄存器

MOV AL,[DI] ; 将[DI]内存中的值加载到AL寄存器

MUL BYTE PTR[SI] ; 将AL与[SI]内存中的值相乘，结果存储在AX中

ADD AX,DX ; 将AX与DX相加，结果存储在AX中

INC SI ; 将SI寄存器的值加1，移动到下一个MATRIX1的元素

INC DI ; 将DI寄存器的值加1，移动到下一个MATRIX2的元素

DEC CH ; 内层循环计数器减1

JNZ AA1 ; 如果内层循环计数器不为0，跳转到AA1标签处继续循环

LEA DI,MATRIX2 ; 将MATRIX2的偏移地址赋给DI寄存器

MOV [BX],AX ; 将AX寄存器中的值存储到BX所指向的内存地址（保存计算结果）

INC BX ; 将BX寄存器的值加1，移动到下一个UNIT数组的元素

INC BX ; 再次将BX寄存器的值加1，移动到下一个UNIT数组的元素

DEC CL ; 外层循环计数器减1

JNZ AA0 ; 如果外层循环计数器不为0，跳转到AA0标签处继续循环

MOV AH,4CH ; 设置AH寄存器的值为4CH，表示程序终止

INT 21H ; 调用21H中断（DOS功能调用）

CODE ENDS

END START

（2）实验结果（截图）：



**第四次实验**

（对应《汇编语言实验教程》 **实验五** “一、基础性实验”、“二、加强性实验”）

“一、**基础性实验**”

（1）程序：

DATA SEGMENT

NUMBER1 DB 8 DUP(0) ; 被加数参数列表，8个字节大小，初始值为0

NUMBER2 DB 8 DUP(0) ; 加数参数列表，8个字节大小，初始值为0

NUMBER3 DB 9 DUP(0) ; 和参数列表，9个字节大小，初始值为0

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

MAIN PROC FAR ; 主程序定义为 FAR 类型

MOV AX,DATA

MOV DS,AX ; 将数据段的地址加载到DS寄存器中，以便访问数据段的变量

CALL SUB1 ; 调用SUB1非压缩BCD码接收子程序

POP CX ; 取被加数位数

LEA BX,NUMBER1 ; 取被加数参数表地址

AA3: POP AX ; 取被加数个位、十位、百位...

MOV [BX],AL ; 将被加数存入参数表

INC BX ; 形成下一个地址

LOOP AA3 ; 循环，直到被加数的所有位都被处理完毕

MOV AH,3 ; 查找当前光标位置

INT 10H

MOV AH,2 ; 设置光标位置

MOV DL,8

INT 10H

MOV DL,2BH ; 输出加号(+)

MOV AH,2

INT 21H

CALL SUB1 ; 调用SUB1非压缩BCD码接收子程序

POP CX ; 取加数位数

LEA BX,NUMBER2 ; 取加数参数表地址

AA4: POP AX ; 取加数个位、十位、百位...

MOV [BX],AL ; 将加数存入参数表

INC BX ; 形成下一个地址

LOOP AA4 ; 循环，直到加数的所有位都被处理完毕

MOV AH,3 ; 查找当前光标位置

INT 10H

MOV AH,2 ; 设置光标位置

MOV DL,17

INT 10H

MOV DL,3DH ; 输出等号(=)

MOV AH,2

INT 21H

LEA SI,NUMBER1 ; 取被加数参数表地址

LEA DI,NUMBER2 ; 取加数参数表地址

LEA BX,NUMBER3 ; 取和参数表地址

SUB CX,CX ; 将CX清零，用作计数器

MOV CX,8 ; 加位数计数器初值

AA5: MOV AL,[SI] ; 取被加数

ADC AL,[DI] ; 非压缩BCD码加法

AAA ; 调整和的结果

MOV [BX],AX ; 将和存到NUMBER3中

INC SI ; 形成下一个地址

INC DI ; 形成下一个地址

INC BX ; 形成下一个地址

LOOP AA5 ; 循环，直到加法运算的所有位都被处理完毕

ADC CL,CL ; 最高位送CL

MOV [BX],CL ; 存储最高位

LEA AX,NUMBER3+8 ; 取和参数表最高位地址

PUSH AX ; 向子程序提供和参数表最高位地址

CALL SUB2 ; 非压缩BCD码显示子程序

MOV CX,16

LEA BX,NUMBER1 ; 取被加数参数表地址

XOR AL,AL

QQQ2: MOV [BX],AL ; 将被加数参数表的值清零

INC BX

LOOP QQQ2

MOV AH,4CH ; 返回DOS

INT 21H

SUB1 PROC NEAR ; 非压缩BCD码数据输入子程序

POP BX ; 保存返回地址

SUB CX,CX ; 键入位数计数器清零

AA1: MOV AH,1

INT 21H

CMP AL,30H ; 判断是否小于0键，如果是，则返回主程序

JC AA2

CMP AL,3AH ; 判断是否大于9键，如果是，则返回主程序

JNC AA2

INC CX ; 输入数据位数

PUSH AX ; 非压缩BCD码压栈

JMP AA1 ; 继续接收输入的数据

AA2: PUSH CX ; 输入数据位数压栈

PUSH BX ; 返回地址压栈

RET ; 返回主函数

SUB1 ENDP

SUB2 PROC NEAR ; 非压缩BCD码显示子程序

POP AX ; 保存返回地址

POP BX ; 取和参数表最高位地址

PUSH AX ; 返回地址压栈

MOV CX,9

AA7: MOV AL,[BX] ; 消除头部的无效数字0

CMP AL,0

JNZ AA6

DEC CX

DEC BX

JMP AA7

AA6: MOV DL,[BX] ; 取和最高位、次高位...个位

ADD DL,30H ; 将数字转换为ASCII码

MOV AH,2

INT 21H

DEC BX

LOOP AA6

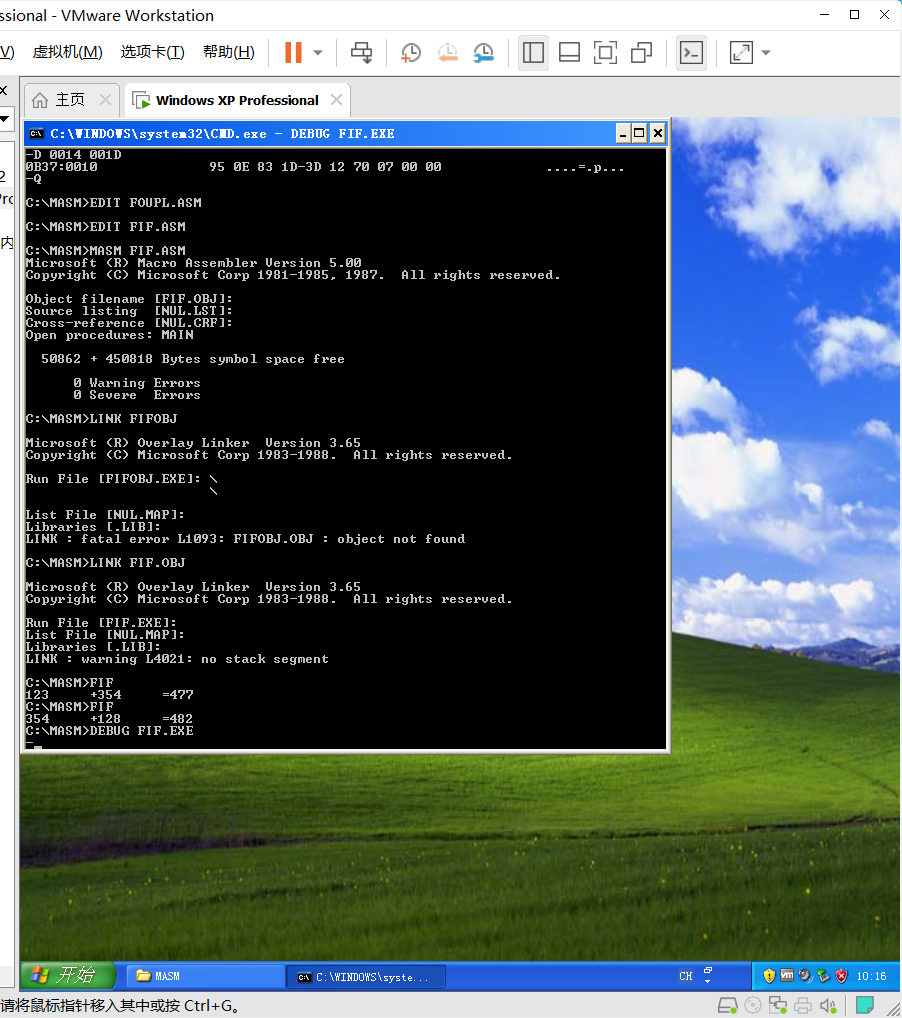
RET

SUB2 ENDP

CODE ENDS

END MAIN

（2）实验结果（截图）：



（3）“思考题”：

**1. 区别：**

**- `CALL` 指令：`CALL` 指令用于调用子程序（过程或函数），它会将当前的指令地址（返回地址）压入栈中，并跳转到指定的子程序开始执行。执行完子程序后，通过返回指令（如 `RET`）可以返回到调用指令的下一条指令继续执行。**

**- 无条件转移指令（如 `JMP`）：无条件转移指令用于无条件地跳转到指定的标号或地址，它不会将当前指令地址保存起来，也不会返回原来的位置。执行完跳转后，程序将继续执行跳转目标处的指令。**

**2. 应用：**

**- `CALL` 指令的主要应用是在程序中调用子程序，实现模块化编程和代码重用。它可以将程序分解为多个子程序，每个子程序负责完成特定的任务，通过 `CALL` 指令进行调用和返回。这样可以提高代码的可读性、可维护性和重用性。**

**- 无条件转移指令（如 `JMP`）主要应用于程序流程控制。它可以用于循环结构、条件判断、函数返回等。通过跳转到不同的标号或地址，可以改变程序的执行流程，实现不同的控制逻辑。无条件转移指令通常与条件判断指令（如 `CMP`、`JZ`、`JNZ` 等）结合使用，实现复杂的条件分支和循环控制。**

“二、**加强性实验**”

（1）程序：

DATA SEGMENT

ARRAY DB 13,24,92,42,25,46,75,81,53,10 ; 声明一个数组，包含了一组数字

MA DB ' MAX = ','$' ; 声明一个字符串常量，用于显示最大值

MI DB 0DH,0AH,' MIN = $' ; 声明一个字符串常量，用于显示最小值

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

MAIN PROC FAR ; 主程序定义为 FAR 类型

MOV AX,DATA ; 将数据段的地址加载到 AX 寄存器

MOV DS,AX ; 将 AX 寄存器中的值移动到 DS 寄存器，将数据段设置为 DS 寄存器的值

LEA DX,MA ; 将字符串 "MAX = " 的地址加载到 DX 寄存器

MOV AH,9 ; 设置 AH 寄存器的值为 9，表示要打印字符串

INT 21H ; 调用 DOS 中断 21H，显示字符串

LEA BX,ARRAY ; 将数组 ARRAY 的地址加载到 BX 寄存器

CALL MAX ; 调用 MAX 过程，查找最大值

CMP DL,10 ; 检查最大值是否为两位数

JC AA6 ; 如果最大值为两位数，跳转到标签 AA6

MOV DH,30H ; 将十位数的 ASCII 值加载到 DH 寄存器

AA5: INC DH ; 十位数加一

SUB DL,10 ; 对个位数进行模 10

CMP DL,10 ; 检查是否超过个位数的范围

JNC AA5 ; 如果超过范围，跳转到标签 AA5

PUSH DX ; 将最大值保存在堆栈中

MOV DL,DH ; 将十位数的值移动到 DL 寄存器，准备显示

MOV AH,2 ; 设置 AH 寄存器的值为 2，表示要打印字符

INT 21H ; 调用 DOS 中断 21H，显示十位数

POP DX ; 恢复最大值到 DX 寄存器

AA6: ADD DL,30H ; 将个位数的 ASCII 值加载到 DL 寄存器

MOV AH,2 ; 设置 AH 寄存器的值为 2，表示要打印字符

INT 21H ; 调用 DOS 中断 21H，显示个位数

LEA DX,MI ; 将字符串 "回车 MIN = " 的地址加载到 DX 寄存器

MOV AH,9 ; 设置 AH 寄存器的值为 9，表示要打印字符串

INT 21H ; 调用 DOS 中断 21H，显示字符串

LEA BX,ARRAY ; 将数组 ARRAY 的地址加载到 BX 寄存器

CALL MIN ; 调用 MIN 过程，查找最小值

CMP DL,10 ; 检查最小值是否为两位数

JC AA8 ; 如果最小值为两位数，跳转到标签 AA8

MOV DH,30H ; 将十位数的 ASCII 值加载到 DH 寄存器

AA7: INC DH ; 十位数加一

SUB DL,10 ; 对个位数进行模 10

CMP DL,10 ; 检查是否超过个位数的范围

JNC AA7 ; 如果超过范围，跳转到标签 AA7

PUSH DX ; 将最小值保存在堆栈中

MOV DL,DH ; 将十位数的值移动到 DL 寄存器，准备显示

MOV AH,2 ; 设置 AH 寄存器的值为 2，表示要打印字符

INT 21H ; 调用 DOS 中断 21H，显示十位数

POP DX ; 恢复最小值到 DX 寄存器

AA8: ADD DL,30H ; 将个位数的 ASCII 值加载到 DL 寄存器

MOV AH,2 ; 设置 AH 寄存器的值为 2，表示要打印字符

INT 21H ; 调用 DOS 中断 21H，显示个位数

MOV AH,4CH ; 设置 AH 寄存器的值为 4CH，表示要返回到 DOS

INT 21H ; 调用 DOS 中断 21H

MAX PROC NEAR ; 最大值查找子过程

MOV CX,10 ; 设置计数器的初始值为 10，用于查找循环

MOV DL,[BX] ; 将 BX 寄存器指向的内存位置的值移动到 DL 寄存器，初始化最大值

AA1: MOV DH,[BX] ; 将 BX 寄存器指向的内存位置的值移动到 DH 寄存器

CMP DL,DH ; 比较当前最大值和当前元素值

JC AA2 ; 如果当前元素值大于当前最大值，跳转到标签 AA2

INC BX ; 移动到下一个元素的位置

DEC CX ; 计数器减一

CMP CX,0 ; 检查计数器是否为零

JZ OUT1 ; 如果计数器为零，跳转到标签 OUT1

JMP AA1 ; 继续循环查找最大值

AA2: MOV DL,DH ; 更新最大值为当前元素值

INC BX ; 移动到下一个元素的位置

DEC CX ; 计数器减一

CMP CX,0 ; 检查计数器是否为零

JZ OUT1 ; 如果计数器为零，跳转到标签 OUT1

JMP AA1 ; 继续循环查找最大值

OUT1: RET ; 返回到主函数

MAX ENDP ; MAX 过程结束

MIN PROC NEAR ; 最小值查找子过程，查找过程与 MAX 过程类似

MOV CX,10 ; 设置计数器的初始值为 10，用于查找循环

MOV DL,[BX] ; 将 BX 寄存器指向的内存位置的值移动到 DL 寄存器，初始化最小值

AA3: MOV DH,[BX] ; 将 BX 寄存器指向的内存位置的值移动到 DH 寄存器

CMP DL,DH ; 比较当前最小值和当前元素值

JNC AA4 ; 如果当前元素值小于当前最小值，跳转到标签 AA4

INC BX ; 移动到下一个元素的位置

DEC CX ; 计数器减一

CMP CX,0 ; 检查计数器是否为零

JZ OUT2 ; 如果计数器为零，跳转到标签 OUT2

JMP AA3 ; 继续循环查找最小值

AA4: MOV DL,DH ; 更新最小值为当前元素值

INC BX ; 移动到下一个元素的位置

DEC CX ; 计数器减一

CMP CX,0 ; 检查计数器是否为零

JZ OUT2 ; 如果计数器为零，跳转到标签 OUT2

;要修改的地方，源代码给的是OUT1

JMP AA3 ; 继续循环查找最小值

OUT2: RET ; 返回到主函数

MIN ENDP ; MIN 过程结束

CODE ENDS

END MAIN

（2）实验结果（截图）：



（对应《汇编语言实验教程》 **实验六** “一、基础性实验”、“二、加强性实验”）

“一、**基础性实验**”

（1）程序：

DATA SEGMENT

ARRAY1 DW 5 DUP(5 DUP(212FH,5A5DH,1234H,7865H)) ;待查找数组

EOD DW 0 ;结束标记，用于计算数组个数

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

MAIN PROC FAR ;主函数定义为远程调用

MOV AX,DATA

MOV DS,AX ;将DS寄存器设置为数据段的段地址

LEA SI,ARRAY1 ;将数组的起始地址加载到SI寄存器

MOV CX,(EOD-ARRAY1)/2 ;计算数组个数，(EOD-ARRAY1)/2为数组元素个数，其中每个元素占两个字节

CALL SUB3 ;调用查找子程序 SUB3，查找数组中的最大元素及其位置

MOV DI,CX ;最大元素所在的段地址保存在DI寄存器中

MOV BX,DATA

CALL SUB4 ;调用显示子程序，显示最大元素的段地址

MOV DL,':'

MOV AH,2

INT 21H ;显示冒号字符

MOV BX,SI ;最大元素的段内偏移保存在BX寄存器中

CALL SUB4 ;调用显示子程序，显示最大元素的段内偏移

MOV DL,20H ;显示空格

MOV AH,2

INT 21H

MOV BX,DI ;传出数组中的最大元素的值保存在BX寄存器中

CALL SUB4 ;显示查找到的最大值

MOV AH,4CH

INT 21H ;程序结束，返回DOS

SUB3 PROC NEAR ;查找子程序，查找数组中最大元素及其位置

PUSH AX ;相关寄存器保护

PUSH DI

MOV DI,SI ;DI中保存最大值的段内地址

MOV AX,[SI] ;AX中保存最大值

AA1: CMP AX,[SI] ;比较当前元素与最大值

JNC AA2 ;如果当前元素大于等于最大值，跳转到AA2

MOV AX,[SI] ;更新最大值

MOV DI,SI ;更新最大值的段内地址

AA2: ADD SI,2 ;移动到下一个元素

LOOP AA1 ;循环直到处理完所有元素

MOV SI,DI ;最大元素段内地址通过SI传出

MOV CX,AX ;最大元素的值通过CX传出

POP DI ;恢复保护的寄存器

POP AX

RET ;返回调用子程序的位置

SUB3 ENDP

SUB4 PROC NEAR ;十六进制数显示子程序

MOV DL,BH ;处理BX中高8位

MOV CL,4

SHR DL,CL ;将高8位右移4位，获取高位的十六进制数

CALL SUB5 ;调用十六进制数显示子程序

MOV DL,BH

AND DL,0FH ;将BX高8位中的高位掩码为0，获取低位的十六进制数

CALL SUB5 ;调用十六进制数显示子程序

MOV DL,BL ;处理BX中低8位

SHR DL,CL ;将低8位右移4位，获取高位的十六进制数

CALL SUB5 ;调用十六进制数显示子程序

MOV DL,BL

AND DL,0FH ;将BX低8位中的高位掩码为0，获取低位的十六进制数

CALL SUB5 ;调用十六进制数显示子程序

RET ;返回调用子程序的位置

SUB4 ENDP

SUB5 PROC NEAR ;十六进制数显示子程序

OR DL,30H ;将DL中的数值转换为ASCII码表示的字符

CMP DL,3AH ;比较DL与ASCII码表示的字符':'

JC AA3 ;如果小于':'，跳转到AA3

ADD DL,7 ;否则将DL中的数值加上7，得到大写字母的ASCII码

AA3: MOV AH,2

INT 21H ;显示DL中的字符

RET ;返回调用子程序的位置

SUB5 ENDP

CODE ENDS

END MAIN ;程序结束

（2）实验结果（截图）：

1

“二、**加强性实验**”

（1）程序：

DATA SEGMENT

RESULT DW 0 ; 声明一个名为 RESULT 的字型变量，用于存储结果

DATA ENDS

CODE SEGMENT

ASSUME DS:DATA,CS:CODE

MAIN PROC FAR ; 主程序定义为 FAR 类型

MOV AX,DATA

MOV DS,AX ; 将数据段地址加载到 DS 寄存器

MOV CX,5 ; 设置循环次数为 5

CALL FACT ; 调用 FACT 函数计算结果

CMP CX,10000 ; 比较结果和 10000

JC BB1 ; 如果结果小于 10000，则跳转到标签 BB1

MOV DL,30H ; 将十进制数字 0 的 ASCII 值加载到 DL 寄存器

AA1: INC DL ; 万位计数

SUB CX,10000 ; 模 10000，获取千位及以下的数值

CMP CX,10000 ; 检查是否仍然大于等于 10000

JNC AA1 ; 如果仍然大于等于 10000，则继续循环

MOV AH,2 ; 设置 AH 寄存器的值为 2，表示要打印字符

INT 21H ; 调用 DOS 中断 21H，显示万位数值

BB1: CMP CX,1000 ; 比较结果和 1000

JC BB2 ; 如果结果小于 1000，则跳转到标签 BB2

MOV DL,30H ; 将十进制数字 0 的 ASCII 值加载到 DL 寄存器

AA2: INC DL ; 千位计数

SUB CX,1000 ; 模 1000，获取百位及以下的数值

CMP CX,1000 ; 检查是否仍然大于等于 1000

JNC AA2 ; 如果仍然大于等于 1000，则继续循环

MOV AH,2 ; 设置 AH 寄存器的值为 2，表示要打印字符

INT 21H ; 调用 DOS 中断 21H，显示千位数值

BB2: CMP CX,100 ; 比较结果和 100

JC BB3 ; 如果结果小于 100，则跳转到标签 BB3

MOV DL,30H ; 将十进制数字 0 的 ASCII 值加载到 DL 寄存器

AA3: INC DL ; 百位计数

SUB CX,100 ; 模 100，获取十位及以下的数值

CMP CX,100 ; 检查是否仍然大于等于 100

JNC AA3 ; 如果仍然大于等于 100，则继续循环

MOV AH,2 ; 设置 AH 寄存器的值为 2，表示要打印字符

INT 21H ; 调用 DOS 中断 21H，显示百位数值

BB3: CMP CX,10 ; 比较结果和 10

JC BB4 ; 如果结果小于 10，则跳转到标签 BB4

MOV DL,30H ; 将十进制数字 0 的 ASCII 值加载到 DL 寄存器

AA4: INC DL ; 十位计数

SUB CX,10 ; 模 10，获取个位数值

CMP CX,10 ; 检查是否仍然大于等于 10

JNC AA4 ; 如果仍然大于等于 10，则继续循环

MOV AH,2 ; 设置 AH 寄存器的值为 2，表示要打印字符

INT 21H ; 调用 DOS 中断 21H，显示十位数值

BB4: MOV DL,CL ; 将 CX 的值（即个位数值）加载到 DL 寄存器

ADD DL,30H ; 将个位数值转换为 ASCII 字符

MOV AH,2 ; 设置 AH 寄存器的值为 2，表示要打印字符

INT 21H ; 调用 DOS 中断 21H，显示个位数值

MOV AH,4CH ; 设置 AH 寄存器的值为 4CH，表示程序结束

INT 21H ; 调用 DOS 中断 21H，程序结束

FACT PROC NEAR ; FACT 子程序，用于计算 1!+2!+3!+4!+5!

AND CX,0FFH ; 将 CX 寄存器的高位清零，确保只使用低位的 8 位

MOV AX,0 ; 将 AX 寄存器的值设为 0，用于累加阶乘结果

LOP2: CALL FANG ; 调用 FANG 子程序，计算当前循环计数的阶乘，并将结果存入 DX

ADD AX,DX ; 将计算得到的阶乘结果累加到 AX 寄存器中

DEC CX ; 循环计数减一

CMP CX,0 ; 检查循环计数是否为 0

JNZ LOP2 ; 如果不为 0，则继续循环

MOV CX,AX ; 将 AX 寄存器中的累加结果存入 CX 寄存器，作为最终的结果

RET ; 返回到调用 FACT 子程序的位置

FACT ENDP ; FACT 子程序结束

FANG PROC NEAR ; FANG 子程序，用于计算阶乘

PUSH CX ; 保存 CX 寄存器的值

PUSH AX ; 保存 AX 寄存器的值

AND CX,0FFH ; 将 CX 寄存器的高位清零，确保只使用低位的 8 位

MOV AX,1 ; 将 AX 寄存器的值设为 1，用于计算阶乘

LOP1: MUL CX ; 将 CX 寄存器的值乘以 AX 寄存器的值，结果存入 AX

DEC CX ; 循环计数减一

CMP CX,0 ; 检查循环计数是否为 0

JNZ LOP1 ; 如果不为 0，则继续循环

MOV DX,AX ; 将 AX 寄存器中的结果存入 DX 寄存器，作为阶乘结果

POP AX ; 恢复 AX 寄存器的值

POP CX ; 恢复 CX 寄存器的值

RET ; 返回到调用 FANG 子程序的位置

FANG ENDP ; FANG 子程序结束

CODE ENDS

END MAIN ; 程序结束

（2）实验结果（截图）：

2

**第五次实验**

（对应《汇编语言实验教程》 **实验七** “一、基础性实验”、“二、加强性实验”）

“一、**基础性实验**”

（1）程序：

DATA SEGMENT

STRING DB 100 DUP(0) ;用来存放输入的字符串，最大长度为100字节

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

MAIN PROC FAR

MOV AX,DATA

MOV DS,AX ;将DS寄存器设置为数据段的段地址

LEA BX,STRING ;将字符串的起始地址加载到BX寄存器

MOV CL,0 ;小写字母数，初始化为0

AA1:

MOV AH,1 ;输入字符

INT 21H

CMP AL,0DH ;当输入回车时，计算小写字母个数并输出

JZ AA3

CMP AL,41H ;小于 'A' 的字符，退出

JC AA4

CMP AL,5BH ;大于 'Z' 的字符，退出

JNC AA2

MOV [BX],AL ;将输入的字符存入内存

INC BX ;移动到下一个存储位置

JMP AA1 ;继续输入字符

AA2:

CMP AL,61H ;小于 'a' 而大于 'Z' 的字符，退出

;这是a

JC AA4

CMP AL,7BH ;大于 'z' 的字符，退出

;这是123，122是z

JNC AA4

MOV [BX],AL ;将输入的字符存入内存

INC BX ;移动到下一个存储位置

INC CL ;小写字母数量加一

JMP AA1 ;继续输入字符

AA3:

PUSH CX ;保存小写字母的数量

MOV AH,3 ;获取当前光标位置

INT 10H

MOV AH,2 ;设置光标位置

LEA DI,STRING ;将字符串的起始地址加载到DI寄存器

SUB BX,DI ;计算光标列位置

MOV DL,BL ;将光标列位置保存到DL寄存器

INC DL ;光标向后移动一位

INT 10H

POP CX ;恢复小写字母的数量

MOV DL,CL

ADD DL,30H ;将小写字母数量从十进制转换成十六进制

CMP DL, 3AH ;检查是否大于9

JBE DisplayDigit ;小于等于9，直接显示

ADD DL, 7H ;大于9，加上7H转换成A-F的ASCII码

DisplayDigit:

MOV AH, 2 ;显示DL寄存器中的字符

INT 21H

AA4:

MOV AH, 4CH ;程序结束，返回DOS

INT 21H

CODE ENDS

END MAIN ;程序结束

（2）实验结果（截图）：



（3）“思考题”：

**错误：当小写字母的个数超过十六个时，不能正确输出。更改后的代码既可以正确输出小写字母的个数。  
原因是当统计输出时，当小写字母个数大于9时直接将其加上7，而没有考虑高位的情况。**

**排除方法：将程序（1）改为下述代码，黄色部分为替换的部分，用DL保存十位数，用DH保存个位数，分别输出DL和DH，就可以得到正确的结果。用这种方法可以统计0-99个字符。**

**更改代码：**

DATA SEGMENT

STRING DB 100 DUP(0) ;用来存放输入的字符串

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

MAIN PROC FAR

MOV AX,DATA

MOV DS,AX

LEA BX,STRING

MOV CL,0 ;小写字母数

AA1: MOV AH,1 ;输入字符串

INT 21H

CMP AL,0DH ;当输入回车时，计算小写字母个数并输出

JZ AA3

CMP AL,41H ;小于 A 的退出

JC AA4

CMP AL,5BH ;大于 Z 的

JNC AA2

MOV [BX],AL

INC BX

JMP AA1

AA2: CMP AL,61H ;小于 a 而大于 Z 的退出

JC AA4

CMP AL,7BH ;大于 z 的退出

JNC AA4

MOV [BX],AL ;将输入合法字符存入内存中

INC BX ;形成下一存放地址

INC CL ;小写字母数量加一

JMP AA1 ;继续输入字符

AA3: PUSH CX ;保存小写字母个数值

MOV AH,3 ;查找当前光标位置

INT 10H

MOV AH,2 ;设置光标位置

LEA DI,STRING

SUB BX,DI ;计算光标列位置

MOV DL,BL ;设置光标列位置

INC DL ;光标向后移一位

INT 10H

POP CX

MOV DH,CL

MOV DL,0

AA5:CMP DH,10

JS AA6

SUB DH,10

ADD DL,1

JMP AA5

AA6:ADD DL,30H ;

MOV AH,2

INT 21H

MOV DL,DH

ADD DL,30H

MOV AH,2

INT 21H

AA4: MOV AH,4CH

INT 21H

MAIN ENDP

CODE ENDS

END MAIN

“二、**加强性实验**”

（1）程序：

DATA SEGMENT

STRING1 DB 'ABCDEF' ;存放输入的字符串

STRING2 DB 'ABCDEFG'

YES DB 'YES$'

NO DB 'NO$'

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

MAIN PROC FAR

MOV AX,DATA ; 将数据段的首地址装入AX寄存器

MOV DS,AX ; 将DS寄存器设置为数据段的地址

MOV CX,STRING2-STRING1 ; 计算字符串长度，存入CX寄存器

MOV BX,YES-STRING2 ; 计算字符串长度，存入BX寄存器

CMP CX,BX ; 比较两个字符串的长度

JNZ AA2 ; 如果长度不相等，跳转到标号AA2处，输出"NO"

LEA BX,STRING1 ; 将STRING1的地址装入BX寄存器

LEA SI,STRING2 ; 将STRING2的地址装入SI寄存器

AA1: ; 循环开始标号

MOV AL,[BX] ; 将BX寄存器指向的内存位置的值装入AL寄存器，即取得字符串1的字符

MOV AH,[SI] ; 将SI寄存器指向的内存位置的值装入AH寄存器，即取得字符串2的字符

CMP AL,AH ; 比较字符串1和字符串2对应位置的字符

JNZ AA2 ; 如果不相等，跳转到标号AA2处，输出"NO"

INC BX ; 字符相等，将BX寄存器递增，指向字符串1的下一个字符

INC SI ; 字符相等，将SI寄存器递增，指向字符串2的下一个字符

LOOP AA1 ; 继续循环，直到CX寄存器的值为0，即字符串比较结束

MOV AH,9 ; 设置AH寄存器的值为9，表示打印字符串

LEA DX,YES ; 将YES的地址装入DX寄存器，准备打印"Yes"

INT 21H ; 调用21H中断，执行打印操作

MOV AH,4CH ; 设置AH寄存器的值为4CH，表示程序终止

INT 21H ; 调用21H中断，程序正常终止

AA2: ; 字符串长度不相等或字符比较失败时的标号

MOV AH,9 ; 设置AH寄存器的值为9，表示打印字符串

LEA DX,NO ; 将NO的地址装入DX寄存器，准备打印"No"

INT 21H ; 调用21H中断，执行打印操作

MOV AH,4CH ; 设置AH寄存器的值为4CH，表示程序终止

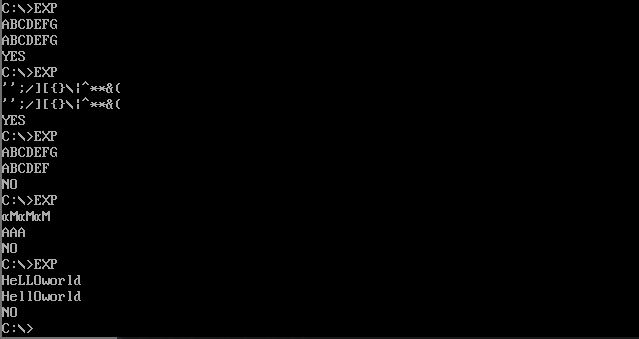
INT 21H ; 调用21H中断，程序正常终止

MAIN ENDP

CODE ENDS

END MAIN

（2）实验结果（截图）：

****

（3）“思考题”：

**编写程序，在一个字符串中查找匹配字符：**

DATA SEGMENT

STR DB 'Hello, World!',0 ; 要搜索的字符串

SEARCH\_CHAR DB 'o' ; 要搜索的字符

FOUND\_MSG DB 'Character found.' ; 匹配字符被找到的消息

NOT\_FOUND\_MSG DB 'Character not found.' ; 匹配字符未找到的消息

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA

START:

MOV AX, DATA ; 将数据段的起始地址加载到 AX 寄存器

MOV DS, AX ; 将 AX 中的值移动到 DS 寄存器，将 DS 设置为数据段的段地址

MOV SI, OFFSET STR ; 将字符串的偏移地址加载到 SI 寄存器

MOV AL, SEARCH\_CHAR ; 将要搜索的字符加载到 AL 寄存器

XOR CX, CX ; 将 CX 寄存器清零，用于计数匹配字符的次数

SEARCH\_LOOP:

MOV BL, [SI] ; 将 SI 指向的内存位置中的值加载到 BL 寄存器

CMP BL, AL ; 将 BL 寄存器的值与要搜索的字符比较

JE CHARACTER\_FOUND ; 如果相等，表示找到匹配字符，跳转到 CHARACTER\_FOUND 标签处

INC SI ; 递增 SI 寄存器的值，移动到下一个字符

CMP BL, 0 ; 检查是否到达字符串的结尾

JNE SEARCH\_LOOP ; 如果不是结尾，继续循环搜索

CHARACTER\_NOT\_FOUND:

MOV AH, 09H ; 设置 AH 寄存器为 09H，表示显示字符串

LEA DX, NOT\_FOUND\_MSG ; 将未找到匹配字符的消息的偏移地址加载到 DX 寄存器

INT 21H ; 调用 21H 中断，显示未找到匹配字符的消息

JMP END\_PROGRAM ; 跳转到 END\_PROGRAM 标签处，结束程序的执行

CHARACTER\_FOUND:

MOV AH, 09H ; 设置 AH 寄存器为 09H，表示显示字符串

LEA DX, FOUND\_MSG ; 将找到匹配字符的消息的偏移地址加载到 DX 寄存器

INT 21H ; 调用 21H 中断，显示找到匹配字符的消息

END\_PROGRAM:

MOV AH, 4CH ; 设置 AH 寄存器为 4CH，表示程序的正常退出

INT 21H ; 调用 21H 中断，结束程序的执行

CODE ENDS

END START

**第六次实验**

（对应《汇编语言实验教程》 **实验八** “一、基础性实验”、“二、加强性实验”）

“一、**基础性实验**”

（1）程序：

DATA SEGMENT

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA

START:

MOV AL, 'A' ; 要显示的字符，修改为字符"A"

MOV DX, 10H ; 显示个数为 16

MOV BL, 0F0H ; 初始化颜色，前景为黑色，背景为白色

A1:

PUSH DX

MOV AH, 9 ; 显示带有属性的字符

MOV BH, 0 ; 页号为 0

MOV CX, 1 ; 显示一个字符

INT 10H

MOV AH, 3 ; 读取光标位置

MOV BH, 0 ; 页数为 0

INT 10H

ADD DL, 1 ; 递增光标的列号

ADD DH, 1 ; 递增光标的行号

MOV AH, 2 ; 设置光标

INT 10H

ADD BL, 1 ; 字符属性加 1，改变前景颜色

POP DX

DEC DX ; 显示字符个数统计

JNZ A1

MOV AH, 4CH

INT 21H

CODE ENDS

END START

（2）实验结果（截图）：



（3）“思考题”：

**要将程序中的字符以斜线方式显示，应让控制列数的DH 和控制行数的DL都+1，以下是示例代码：**

DATA SEGMENT

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA

START:

MOV AL, 'A' ; 要显示的字符，修改为字符"A"

MOV DX, 10H ; 显示个数为 16

MOV BL, 0F0H ; 初始化颜色，前景为黑色，背景为白色

A1:

PUSH DX

MOV AH, 9 ; 显示带有属性的字符

MOV BH, 0 ; 页号为 0

MOV CX, 1 ; 显示一个字符

INT 10H

MOV AH, 3 ; 读取光标位置

MOV BH, 0 ; 页数为 0

INT 10H

ADD DL, 1 ; 递增光标的列号

ADD DH, 1 ; 递增光标的行号

MOV AH, 2 ; 设置光标

INT 10H

ADD BL, 1 ; 字符属性加 1，改变前景颜色

POP DX

DEC DX ; 显示字符个数统计

JNZ A1

MOV AH, 4CH

INT 21H

CODE ENDS

END START

“二、**加强性实验**”

（1）程序：

WR MACRO CHA,ATR,NUM ;宏定义,显示处理

MOV AH,2 ;设置光标位置

INT 10H

MOV AL,CHA ;要显示的字符送 AL

MOV CX,NUM ;要显示字符的个数送 CX

MOV BL,ATR ;要显示字符的颜色属性

MOV AH,9

INT 10H

ENDM

CODE SEGMENT

ASSUME CS:CODE

MAIN PROC FAR

CAR: MOV AH,0

MOV AL,3 ;设置屏幕显示模式为 80\*25 彩色字符方式

INT 10H

LOP1: MOV SI,0A0AH ;定义显示位置(10,10)

LOP2: MOV DX,SI

WR 0DBH,0CH,5 ;显示第一行

INC DH ;调整行坐标

SUB DL,2 ;调整列坐标

WR 0DBH,4,9 ;显示第二行

INC DH ;调整行坐标

INC DL ;调整列坐标

WR 09H,8EH,1 ;显示前轮

ADD DL,6 ;调整列坐标

WR 9,8EH,1 ;显示后轮

CALL DELAY ;调用延时

LOP3:

MOV DX,SI

WR 0,0,5 ;清除第一行

INC DH

SUB DL,2

WR 0,0,9 ;清除第二行

INC DH

WR 0,0,8 ;清除第三行

MOV AH,1 ;判断是否有按键

INT 16H

JZ CONU

MOV AH,4CH ;返回 DOS

INT 21H

JLOP2: JMP LOP2

CONU: INC SI

CMP SI,0A50H ;判断车是否跑到头

JB JLOP2

JMP LOP1

MAIN ENDP

DELAY PROC NEAR ;延时子程序

MOV CX,0FFFFH

LOOP LOP3 FFFFH

RET

DELAY ENDP

CODE ENDS

END MAIN

（2）实验结果（截图）：



（对应《汇编语言实验教程》 **实验九** “一、基础性实验”、“二、加强性实验”）

“一、**基础性实验**”

（1）程序：

DATA SEGMENT

STRING DB 'ERROR!DIVIDE BY ZERO!',0AH,0DH,'$'

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA

MAIN PROC FAR

LEA DX, INT0 ;中断服务程序偏移地址

MOV AX, CS

MOV DS, AX

MOV AL, 32 ;选用32号软中断

MOV AH, 25H ;调用设置中断向量中断

INT 21H

MOV AX, DATA

MOV DS, AX

MOV CX, 10 ;9/BL做10次

MOV BL, 6 ;BL初值设为6.

A1:

MOV AX, 9

DIV BL

ADD AL, 30H ;将十进制数转换成十六进制数

MOV DL, AL ;显示除法结果

MOV AH, 2

INT 21H ;显示字符

MOV DL, 0DH

MOV AH, 2

INT 21H

MOV DL, 0AH

MOV AH, 2

INT 21H

CMP CX,0

JZ A3

DEC BL

CMP BL, 0 ;当除数为0时

JZ A2

LOOP A1

A3:

MOV AH,4CH

INT 21H

A2:

INT 32

SUB CX,1

;MOV AH, 4CH

;INT 21

JMP A1

MAIN ENDP

INT0 PROC FAR ;中断服务程序

LEA DX, STRING

MOV AH, 9 ;调用字符串输出中断

INT 21H

MOV BL, 3 ;BL赋为3

IRET ;返回主程序

INT0 ENDP

CODE ENDS

END MAIN

CODE ENDS

END MAIN

（2）实验结果（截图）：



（3）“思考题”：

**1.**

**要实现当出现除数为0时显示错误提示并结束程序，需要对主程序进行修改，要修改以下部分：**

**1. 将标记为`A2`的代码块移动到除法操作之前的位置。**

...

CMP CX, 0

JZ A3

DEC BL

CMP BL, 0 ; 当除数为0时

JZ A2

LOOP A1

...

```

修改为：

...

CMP CX, 0

JZ A2

DEC BL

CMP BL, 0 ; 当除数为0时

JZ A3

LOOP A1

...

**这样，当除数BL为0时，程序将直接跳转到标记为`A3`的代码块，显示错误提示并结束程序。**

**2. 添加一个错误处理的代码块，标记为`A3`。**

A3:

LEA DX, STRING

MOV AH, 9 ; 调用字符串输出中断

INT 21H

MOV AH, 4CH

INT 21H

**这段代码将字符串"ERROR! DIVIDE BY ZERO!"输出，并通过`INT 21H`中断终止程序。**

**2.本实验没有错误，无需排除。**

“二、**加强性实验**”

（1）程序：

; 定义代码段

CSEG SEGMENT

ASSUME CS:CSEG

START PROC FAR

PUSH CS

POP DS

; 获取原 1CH 号的中断向量

MOV AX, 351CH

INT 21H

MOV CS:WORD PTR OLD1C, BX

MOV CS:WORD PTR OLD1C+2, ES ; 保存原 1CH 号的中断向量

; 设置新的 1CH 号的中断向量

MOV DX, OFFSET INT1C

MOV AX, 251CH

INT 21H

; 此后，每 55 毫秒就进入一次新的 1CH 号的中断处理程序

WAITN:

MOV AH, 1

INT 16H ; 查有无键按下

JZ WAITN ; 转等待键按下

MOV AH, 0

INT 16H ; 读键盘

LDS DX, CS:OLD1C

MOV AX, 251CH

INT 21H ; 恢复原 1CH 中断向量

MOV AH, 4CH

INT 21H ; 返回 DOS

START ENDP

; 定义数据空间

OLD1C DD ? ; 保存原中断向量

COUNT DW 0 ; 调用 1CH 中断程序的次数

HHH DB ?, ?, ':' ; "时"

MMM DB ?, ?, ':' ; "分"

SSS DB ?, ?, '$' ; "秒"

; 定义新的 1CH 号的中断处理程序

INT1C PROC FAR

CMP COUNT, 0 ; 调用次数为 "0" 时（1 秒到）

JZ NEXT

DEC COUNT ; 显示次数递减（第一次 18-1）

IRET

NEXT:

MOV COUNT, 18 ; 置显示次数初值

STI

PUSH DS ; 保护现场

PUSH ES

PUSH AX

PUSH BX

PUSH CX

PUSH DX

PUSH SI

PUSH DI

MOV AH, 2

INT 1AH ; 读实时时钟

MOV AL, CH ; 时送 AL

CALL TTASC ; 转换成 ASCII 码

MOV WORD PTR HHH, AX ; 保存时

MOV AL, CL ; 分转换

CALL TTASC

MOV WORD PTR MMM, AX

MOV AL, DH ; 秒转换

CALL TTASC

MOV WORD PTR SSS, AX

CALL CLS ; 清屏

MOV BH, 0

MOV DX, 0140H

MOV AH, 2

INT 10H ; 设置光标（1，65）

PUSH CS

POP DS

MOV DX, OFFSET HHH

MOV AH, 9

INT 21H ; 显示实时时钟

POP DI

POP SI

POP DX

POP CX

POP BX

POP AX

POP ES

POP DS

IRET ; 中断返回

INT1C ENDP

; 将 AL 中的 BCD 数据转换成 ASCII 码存入 AX 中

TTASC PROC

MOV AH, AL

AND AL, 0FH

SHR AH, 1

SHR AH, 1

SHR AH, 1

SHR AH, 1

ADD AX, 3030H

XCHG AH, AL

RET

TTASC ENDP

; 清屏子程序

CLS PROC

MOV AX, 0600H

MOV CX, 0

MOV DX, 184FH

MOV BH, 7

INT 10H

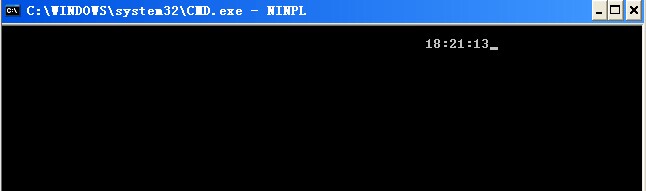
RET

CLS ENDP

CSEG ENDS

END START

（2）实验结果（截图）：



（3）“思考题”：

**1.要对程序进行修改，以读取年月日时间并实现计数1分钟更新显示时间，需要进行以下修改：**

**1. 在数据空间中添加存储年月日时间的变量：**

YEAR DW ?

MONTH DW ?

DAY DW ?

```

**2. 在`INT1C`中断处理程序的起始部分，读取年月日时间并将其存储到相应的变量中：**

MOV AH, 4 ; 调用DOS功能4h，读取年份

INT 1AH

MOV YEAR, CX

MOV AH, 2 ; 调用DOS功能2h，读取月份

INT 1AH

MOV MONTH, DL

MOV AH, 4 ; 调用DOS功能4h，读取日期

INT 1AH

MOV DAY, CX

```

3. 在`INT1C`中断处理程序的末尾部分，添加计数器和更新时间的代码：

INC WORD PTR COUNT ; 计数器加1

CMP COUNT, 60 ; 检查计数器是否达到1分钟

JNE NEXT

; 一分钟已过，更新显示时间

MOV COUNT, 0 ; 重置计数器

MOV BH, 0

MOV DX, 0140H

MOV AH, 2

INT 10H ; 设置光标位置（1，65）

MOV AH, 2 ; 显示时

MOV DL, HHH

INT 21H

MOV DL, ':' ; 显示冒号

INT 21H

MOV AH, 2 ; 显示分

MOV DL, MMM

INT 21H

MOV DL, ':' ; 显示冒号

INT 21H

MOV AH, 2 ; 显示秒

MOV DL, SSS

INT 21H

JMP NEXT

NEXT:

IRET

```

**这样，每当计数器达到60（1分钟）时，程序会更新显示的时间，并重置计数器。**

**2.本实验没有错误，无需排除。**

**总结实验的体会与收获：**

在之前对汇编语言的学习之前，我可能对这门强大而又难学的程序设计语言持怀疑态度。毕竟，现在有那么多高级编程语言可以选择学习，尤其是那些日益强大的面向对象语言。相比之下，汇编语言学起来更加困难，而且开发效率也相对较低。于是，我开始思考，既然有这么多高级语言可以使用，我们为什么还要学习汇编语言？汇编语言作为一门程序设计语言，存在的意义究竟在哪里呢？

然而，通过学习了汇编语言，我发现自己对于汇编语言的强大之处不再持怀疑态度。汇编语言是一门非常强大的语言，尽管以前我不愿意承认这一点。它的强大之处在于它的执行效率。与现阶段流行的面向对象语言和其他高级语言不同，汇编语言追求的是程序的执行效率，而不是开发效率。用汇编语言编写的程序，无论在程序大小还是执行时间上，都远远优于高级语言，即使是那些被公认为比较底层的C语言。

与其他程序设计语言相比，汇编语言操作的是更加底层的东西，它可以直接与硬件进行交互，而其他语言基本上无法实现这一功能。由于与硬件直接打交道，我们在编写程序时必须更加小心，否则很容易让程序失控。如果我们无法控制程序的正常逻辑，程序运行超出我们的控制范围后，我们也无法预测它将运行到何处。例如，我们本来只是要操作用户内存地址空间的数据，但有时候我们可能会疏忽，导致程序运行到内核内存地址空间，这样的程序肯定会崩溃。幸好，我们现在使用的是虚拟模式下的汇编语言编程，不需要过多考虑这些问题，最多重新启动资源管理器，否则电脑可能会多次崩溃。

与汇编语言相比，高级语言在这方面做得相对较好。毕竟，在编写高级语言代码时，我们不直接操作底层硬件，而是先与操作系统进行交互。至于如何操作硬件，那是操作系统的工作，这样我们在一定程度上能够减少错误，并且节省调试程序的时间，提高开发效率。

应用汇编语言的一个重要领域是系统编程和嵌入式系统开发。在这些领域中，汇编语言的优势得到了充分的发挥。由于系统编程和嵌入式系统需要直接与硬件进行交互，使用汇编语言可以更好地控制硬件资源和优化程序性能。例如，操作系统内核的开发通常需要使用汇编语言来编写关键部分，以实现任务切换、中断处理和设备驱动程序等功能。嵌入式系统开发中，由于资源有限，对程序性能和资源利用的要求更高，汇编语言可以提供更精确的控制和优化。

此外，对于对计算机体系结构和底层工作原理有深入理解的计算机科学专业学生或从事底层开发工作的程序员来说，学习汇编语言也是很有价值的。通过学习汇编语言，可以更好地理解计算机的工作原理、指令集架构和内存管理等概念。这对于理解和调试高级语言代码、优化性能以及进行底层故障排除都非常有帮助。

另外，虽然大部分软件开发工作可以使用高级语言完成，但在某些特定场景下，使用汇编语言可以获得更高的灵活性和效率。例如，对于某些密集计算、实时系统或特定硬件平台上的优化，使用汇编语言可以实现更高的性能。在图形编程、密码学和加密算法等领域，汇编语言的速度和精确性也往往是不可或缺的。

总结来说，尽管汇编语言的学习曲线较陡峭，而且在开发效率上相对较低，但它仍然具有重要的应用领域和独特的优势。通过学习汇编语言，可以更好地理解计算机底层原理，掌握系统编程和嵌入式系统开发，以及在某些特定场景下实现更高的性能和效率。因此，对于对计算机科学有深入追求的人或从事底层开发的专业人士来说，学习汇编语言仍然是非常有价值的。