电子科技大学英才实验学院

标 准 实 验 报 告

（实验）课程名称 汇编语言

电子科技大学教务处制表

电 子 科 技 大 学

实　验　报　告

学生姓名：魏祥翰　　学 号：2021270903005

指导教师：王华

实验地点：清水河校区主楼 A2 区　　实验时间：2022.05

一、实验室名称：国家级计算机实验教学示范中心

二、实验项目名称：

1、汇编源程序的上机调试操作基础训练

2、分支程序与循环程序设计

3、数制、码制和子程序的编程与调试

4、串操作指令及其应用程序的设计与调试运行

三、实验学时：16

四、实验原理：（包括知识点，电路图，流程图）

1．8086 汇编生成可执行程序的过程

1）编辑汇编源代码，生成汇编源文件*.asm

2）汇编，生成目标文件*.obj

3）连接，生成可执行文件*.exe

4）调试运行程序，主要用到以下 DEBUG 命令

① D 命令：查看指定内存范围的数据

② G 命令：连续运行指定范围的程序指令段

③ T 命令：执行指定的一条或多条指令，以分解出的基本指令为单位；

④ P 命令：执行指定的一条者多条指令，以助记符指令为单位；

⑤ Q 命令：退出 DEBUG

2．8086 汇编编程环境

由于 win7（64 位）操作系统不支持 16 位程序，所以需要安装 DOS 虚拟机 DOSBOX；

下载 DOS 汇编的编译器和连接器工具包 masm5，解压保存在电脑硬盘，比如：F:/MASM5；

下载 16 位调试工具 DEBUG.EXE 和编辑器 EDIT.EXE，并保存到上述工具包目录下。

3．需要用到的 DOS 功能调用

① （INT 21H）1 号功能调用：键盘输入一个字符并回显

【示例】
```
mov    ah,1     ;功能调用号写入 ah
int    21h    ;将输入键对应的 ASCII 码存入 AL
```

② （INT 21H）2 号功能调用：显示输出一个字符

【示例】
```
mov ah,2   ;功能调用号写入 ah
mov dl,'a'  ;将待显示字符对应的 ASCII 码存入 DL
int 21h      ;运行该中断后，显示小写字母 a
```

③ （INT 21H）9 号功能调用：显示一个字符串

【示例】
```
...    ... ...
mess db 'error! please input again: ',0ah,0dh,'$';预装字符串
...    ...    ...
mov ax,data
mov ds,ax       ;DS 重定位，取串所在段的段地址
error：    mov dx,offset mess    ;DX 取字符串在段内的偏移地址
```

```
mov ah,9        ;9 为该中断的功能调用号
int 21h         ;调用中断显示该字符串
```

④ （int 21h）的 10 号中断服务程序：输入一个字符串

【示例】
```
DATA SEGMENT
BUF    DB     25，？，25 dup（0）; 最多输入 25 个字符
...
MOV AX,DATA
MOV DS,AX
MOV DX,OFFSET BUF
MOV AH,10
INT 21H;
```

⑤ （int 21h）的 4ch 号中断服务程序：返回 DOS

【示例】
```
mov ah,4ch
int 21h
```

4．需要重点运用的指令

① 数据比较指令 CMP：

执行两操作数相减，影响标志位 OF、SF、ZF、PF 和 CF。也就是通过标志位的状态进行两数大小的比较。结果不影响目的操作数。

两数比较结果 A－B		CF	ZF	SF	OF
1．A=B		0	1	0	0
2．无符号数	A＜B（below）	1	0	-	-
	A＞B（above）	0	0	-	-
3．带符号数	A＞B（greater）	-	0	0	0
		-	0	1	1
	A＜B（less）	-	0	0	1
		-	0	1	0

② 条件转移指令：

- 对于无符号数比较，对应的条件转移指令为 JB(JNB)或者 JA(JNA);
- 对于带符号比较，对应的条件转移指令为 JG(JNG)或者 JL(JNL);

③ 组合十进制数加法调整指令 DAA：

- 调整在 AL 中进行，且该指令执行后，将重新影响除 OF 外的其他标志。
- 调整的方式分四种情况：

1）当 AF=0 且 CF=0 时，只对＞9 的部分作+6 处理（如果低四位处理后使得高四位＞9，对高四位继续+6 调整）；

2）当 AF=0 且 CF=1 时，只对高四位作+6 处理（不论高四位是否＞9）；

3）当 AF=1 且 CF=0 时，只对低四位作+6 处理（不论低四位是否＞9，且如果使得高四位＞9，对高四位继续+6 调整）；

4）当 AF=1 且 CF=1 时，对高低四位均作+6 处理（不论高低四位是否＞9）；

④ 串传送指令 MOVSB：

【功能】把位于 DS 段的由 SI 指向的存储单元中的字节或字传送到位于 ES 段由 DI 指向的存储单元中，并修改 SI 和 DI，以指向下一个元素。

⑤ 重复前缀 REP：

【功能】控制跟在其后的基本字符串操作指令，使之重复执行。

【注】该指令不能单独使用，且该指令不影响标志。

⑥ 存字符串指令 STOSB：

【功能】把 AL 或 AX 中的数存到 ES 段由 DI 所指的内存单元，并且自动修改地址指针，（DI）←（DI）＋1。该指令可以与 REP 前缀配合使用，实现在一串内存单元中填入某一相同的数。

5．涉及到的编码

① ASCII（American Standard Code for Information Interchange）

- ASCII 码为美国信息交换标准代码；

- ASCII 共定义了 256 个代码(0-255)；
- 0-127 是标准 ASCII 编码；
- 128-255 是扩展 ASCII 编码；
- 编号 0-32 为控制字符，如：换行、回车、退格等；
- 编号 33-127 为可打印字符，如：数字、字母等。

字符	对应的 ASCII 码
0~9	30H~39H
A~Z	41H~5AH
a-z	61H~7AH
换行	0AH
回车	0DH
ESC 键	1BH
空格	20H

② BCD 码（Binary-Coded Decimal）：

- 就是二进制码的十进制数， 也称二-十进制代码。
- BCD 码有两种表示形式：

1）组合 BCD 码：用 4 个二进制位表示一个十进制位

2）非组合 BCD 码：用 8 个二进制位表示一个十进制位

6．涉及到的主要伪指令

① Segment&ends：段定义伪指令

【示例】 data segment ;定义一个名为 data 的数据段

… … …

data ends ;定义的数据段结束

② Assume：说明段与段寄存器之间的联系

【示例】assume cs:code,ds:data　;指定存放各段段基址的段寄存器

③　PROC/ENDP：过程定义伪指令

【示例】 lop4 proc　;定义一个叫 lop4 的子过程

… … …

Ret　;调用返回指令

lop4 endp　;子过程结束

④　DUP：定义重复数据伪指令

【示例】 array1　db　10　dup(?);定义 10 个连续 byte 字长的数据？

array2　dw　5　dup(1,2,?);定义 5 个连续三数据数组（1,2,?），每个数据都是 word 字长

array3　db　10　dup(10 dup(?));dup 可以嵌套使用

⑤　Offset：取有效地址伪指令

【示例】mov bx,offset shuru；将 shuru 的有效/偏移地址送入 bx

7．影响程序执行顺序的指令

①　Call：调用子程序命令

【示例】call lop4;调用子程序（子过程）lop4

②　Int：调用软中断命令

【示例】int 21h;调用 DOS 功能调用

③　Loop：循环命令

【示例】 mov　cx, 20 ;设置循环次数，必须用 cx 存放

lop:　… … … ;循环体入口为 lop

Loop lop ;如果 cx≠0 则循环，反之执行后面的指令

8．32 位汇编与 16 位汇编的主要区别（涉及保护模式与实模式的主要区别）

①　存储方式及内存寻址方式不同：

保护模式下没有强调段的概念，在 flat 模式下，所有的内存空间都可以作为一个段的存储空间；在实模式中，程序以段的方式进行存储，一个段最大的内存空间大小是 64KB。

保护模式下，内存的物理地址对程序员是透明的，段寄存器不再是内存地址的一部分，实模式下，内存地址是由段寄存器提供的段地址和偏移地址共同构成的，程序员可以对内存数据随意进行修改，是十分不安全的。

② 寄存器数目和存储数据宽度不同：

32 位汇编能使用的寄存器更多，数据位宽均为 32 位，16 位汇编中能使用的寄存器只有不到 20 个，数据位宽为 16 位。

③ 对硬件的中断处理不同：

32 位汇编中调用系统中断控制硬件，需要用链接库函数 IRVINE，16 位汇编中可以直接调用 DOS 中断。

④ 软件开发平台不同：

本次实验中，32 位汇编推荐使用 VS（实验室安装的 2010 版本），16 位汇编需要在虚拟机中运行，用到的工具软件是 masm5.

五、实验目的：

1. 掌握 16 位 DEBUG 的基本命令及其功能
2. 学习数据传送和算术运算指令的用法
3. 熟悉 8086 汇编语言程序操作的过程。
4. 了解 VS 平台下进行 win32 汇编的环境配置方法和调试方法。
5. 1.掌握分支程序的结构、设计与调试方法。
6. 掌握循环程序的设计与调试方法。
7. 了解 DOS 功能调用和 Irvine 链接库的使用方法。
8. 掌握不同进制之间转换的方法。

9. 掌握 DOS 功能调用和链接库函数调用进行数据显示的方法。

10. 掌握子程序的编程与调试方法。

11. 熟悉串操作指令的基本格式和使用方法。

12. 掌握常用的串操作程序的设计方法。

13. 熟悉串操作程序的调试运行过程。

六、实验内容：

1. 学会 16 位汇编程序的调试与运行；
2. 编写 8086 汇编程序计算以下表达式:

Z=(5X+2Y-7)/2

设 X、Y 的值放在字节变量 VARX、VARY 中，结果存放在字节单元 VARZ 中。

3. 选作：在 VS 平台中，正确设置环境参数，编写 32 位汇编程序计算上面表达式 的值，结果通过寄存器窗口和内存窗口进行查看。
4. 用 8086 汇编完成下列题目中任意一题

- 根据输入的字母进行大小写转换（若是输入小写字母则转换成为大写字母，反之），若输入的不是字母则提示出错并重新输入。
- 分别统计下列 20 个数中小于零、大于等于零且小于等于 5、大于 5 的数据个数，分别存入字节单元 RES1、RES2 和 RES3 中。

BUF DB -1, 20, 3, 30, -5, 15, 100, -54, 0, 4, 78, 99, -12, 32, 3, 23, -7, 24, 60,-51

5. 选作：用 win32 汇编完成上述题目中任意一题
6. 用 8086 汇编完成下列题目中任意一题

- 从键盘输入两个 2 位十进制正数，对这两个数相加，结果以十六进制形式显示在屏幕上。
- 从键盘输入两个 2 位十进制正数转换成组合 BCD 码，对这两个数相加，结果以十进制形式显示在屏幕上。

7. 选作：用 win32 汇编完成上述题目中任意一题
8. 用 8086 汇编完成任意一题

- 编制一程序，从键盘输入两个长度不同的字符串，设字符串长度小于 25 个字符。要求在屏幕上以右边对齐的形式显示出来。
- 编制一程序，利用 PC 的定时计数器获取两个字随机数，作为两个地址，进行数据块的复制，要求保证数据块的完整性。

9. 选作：用 win32 汇编完成上面第一题

七、实验器材（设备、元器件）：

1. Hardware：XiaoXin Air 14+ ACN 2021（AMD Ryzen 5 5600U@2.30 GHz/16G RAM/512G ROM）
2. OS：Windows11 （64 位）
3. Software：Dosbox，masm5，debug.exe，edit.exe，notepad++

八、实验步骤：（编辑调试的过程）

实验一

1．安装 dosbox 虚拟机；

2．解压 masm5.zip ，将文件夹中的所有内容存于工作目录 C:\Users\vv123\Desktop\AssemblyLanguage；

3．将 debug.exe，edit.exe 存到 masm5 文件夹中；

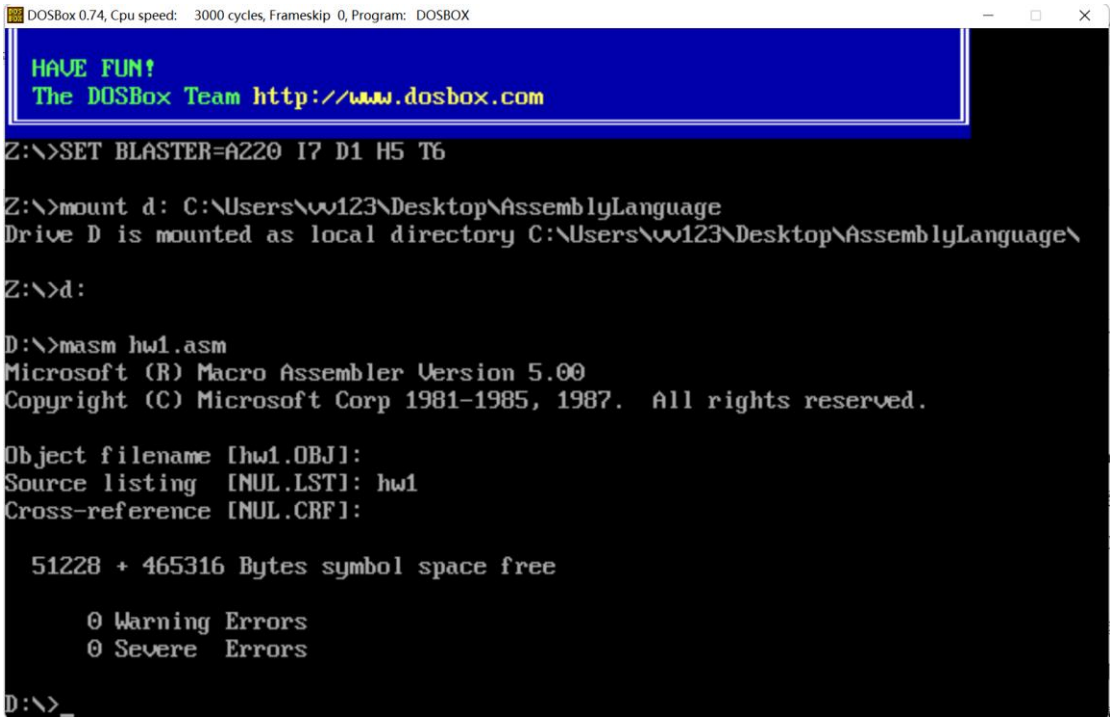
4．修改配置文件，将工作目录挂载为 D:\

5．打开文本编辑器（这里使用 notepad++），编写 16 位汇编程序，命名为 hw1.asm（实验 2、3、4 依次命名为 hw2、hw3、hw4，不再赘述）

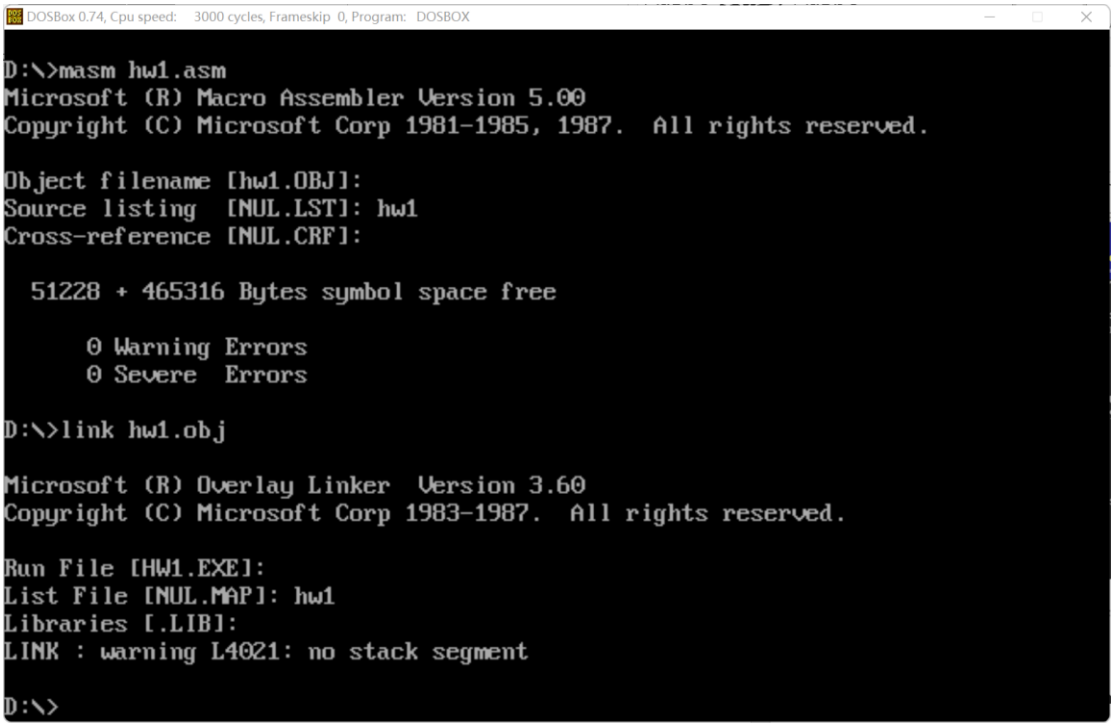
```
data segment
    varx db 5
    vary db 6
    varz db ?
    a db 5
    b db 2
    c db 7
    d db 2
data ends

code segment
    assume cs:code,ds:data
    start:
    mov ax,data
    mov ds,ax
    mov al,varx
    mov bl,vary
    mov cl,a
    mul cl        ;varx *= 5 -> AX(AL)
    add bl,bl     ;vary += vary
    add al,bl     ;varx += vary
    sub al,c      ;varx -= 7
    shr al,1        ;varx >>= 1
    mov varz,al
    mov ah,4ch
    int 21h
code ends

end start

```

注：本程序定义 X=5,Y=6,计算 Z=(5X+2Y-7)/2 的预期结果为 Z=15

6．在 dosbox 中进入工作目录，输入汇编命令：masm hw1.asm；

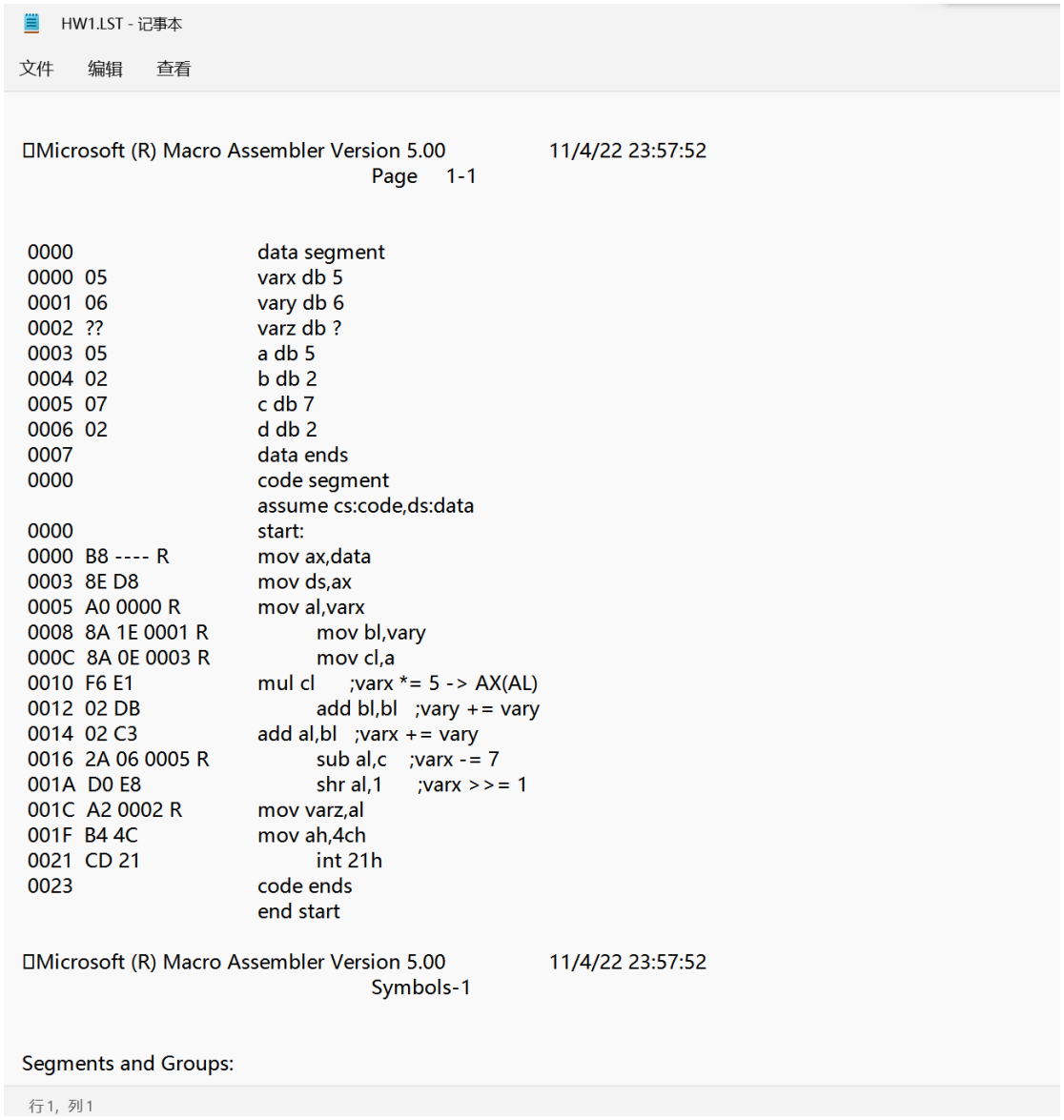


7．继续输入链接命令：link hw1.obj；



8．输入命令 debug hw1.exe，进入调试阶段：

由于实验一没有涉及到显示器输出，因此采用断点运行的方式进行调试。断点在 hw1.lst 文件中方便查找，所以先打开该文件：


HW1.LST - 记事本

文件 编辑 查看

Microsoft (R) Macro Assembler Version 5.00 11/4/22 23:57:52
Page 1-1

```
0000                    data segment
0000  05                varx db 5
0001  06                vary db 6
0002  ??                varz db ?
0003  05                a db 5
0004  02                b db 2
0005  07                c db 7
0006  02                d db 2
0007                    data ends
0000                    code segment
                        assume cs:code,ds:data
0000                    start:
0000  B8 ---- R         mov ax,data
0003  8E D8             mov ds,ax
0005  A0 0000 R         mov al,varx
0008  8A 1E 0001 R            mov bl,vary
000C  8A 0E 0003 R            mov cl,a
0010  F6 E1             mul cl      ;varx *= 5 -> AX(AL)
0012  02 DB                   add bl,bl   ;vary += vary
0014  02 C3             add al,bl   ;varx += vary
0016  2A 06 0005 R            sub al,c    ;varx -= 7
001A  D0 E8                   shr al,1     ;varx >>= 1
001C  A2 0002 R         mov varz,al
001F  B4 4C             mov ah,4ch
0021  CD 21                   int 21h
0023                    code ends
                        end start
```

Microsoft (R) Macro Assembler Version 5.00 11/4/22 23:57:52
Symbols-1

Segments and Groups:

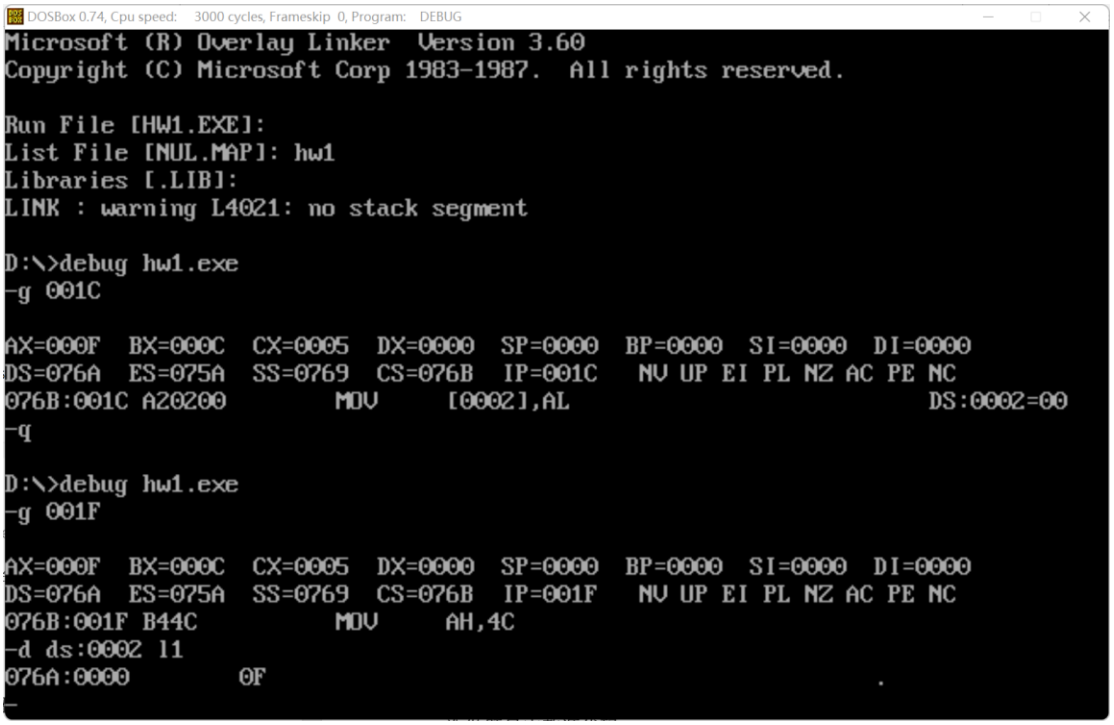
行 1，列 1

由此可见，可以用的断点是：001F

9. 在 dosbox 中继续调试，输入命令 g 001F

10. 运行到断点后，查看内存单元数据，输入命令 d ds:0002 l1

九、实验数据及结果分析：（实验运行结果介绍或者截图，对不同的结果进行分析）



【结论】ds:0002 即为 Z 的值,程序计算得到 Z=0F,十六进制数 0F 转换为十进制为 15，与预期结果一致

此后的实验中，编译、链接等过程与实验一相同，不再重复

实验二

选做题目：

2. 分别统计下列**20**个数中小于零、大于等于零且小于等于**5**、大于**5**的数据个数，分别存入字节单元**RES1**、**RES2**和**RES3**中。

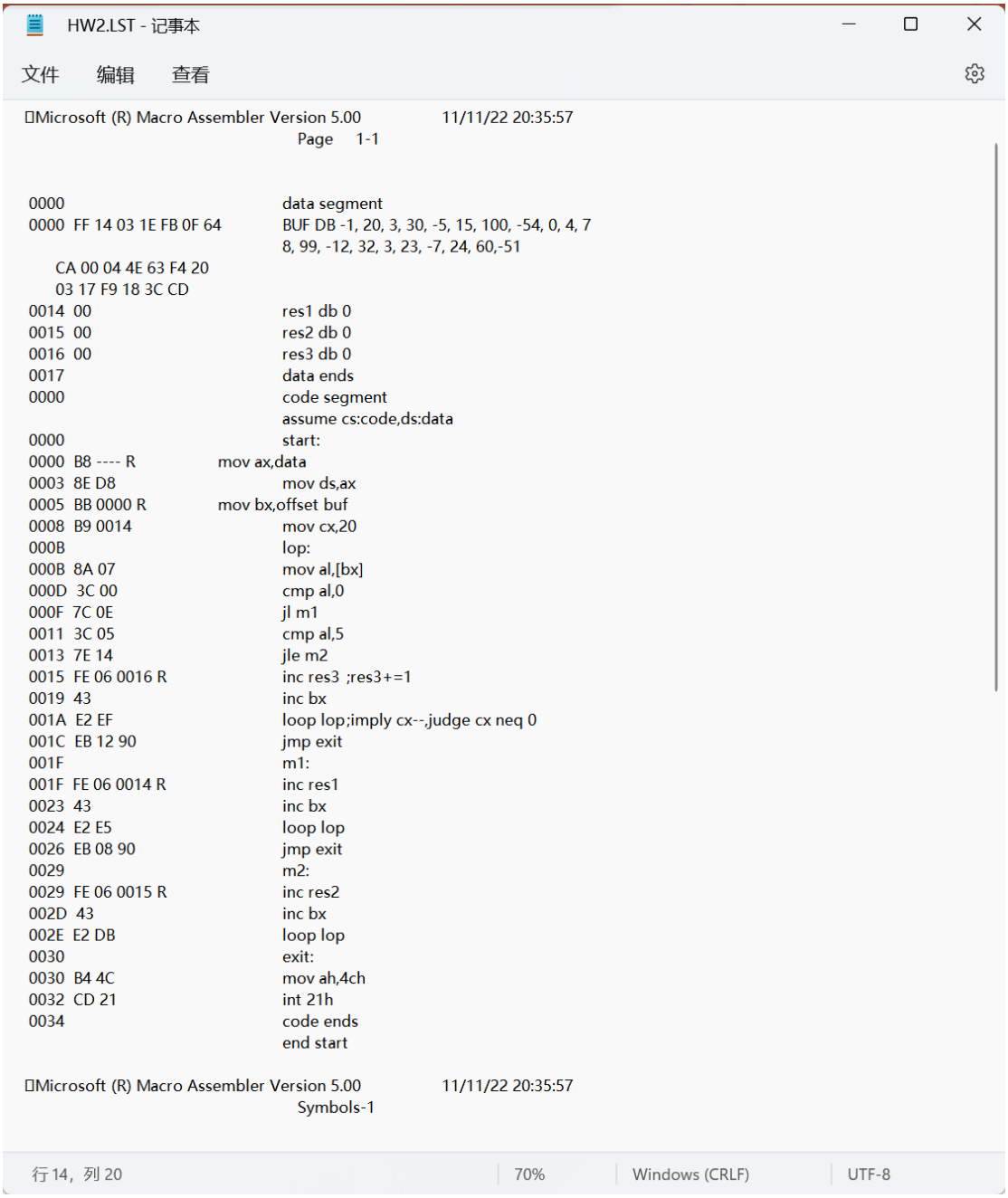
BUF DB -1, 20, 3, 30, -5, 15, 100, -54, 0, 4, 78, 99, -12, 32, 3, 23, -7, 24, 60,-51

实验二完整源代码：

```
data segment
    res1 db 0
    res2 db 0
    res3 db 0
    BUF DB -1,20,3,30,-5,15,100,-54,0,4,78,99,-12,32,3,23,-7,24,60,-51
data ends

code segment
    assume cs:code,ds:data
start:
    mov ax,data
    mov ds,ax
    mov bx,offset buf
    mov cx,20
    lop:
        mov al,[bx]
        cmp al,0
        jl m1
        cmp al,5
        jle m2
        inc res3    ;res3+=1
        inc bx
    loop lop    ;imply cx--,judge cx neq 0
    jmp exit

    m1:
        inc res1
        inc bx
        loop lop
        jmp exit

    m2:
        inc res2
        inc bx
        loop lop

    exit:
    mov ah,4ch
    int 21h
code ends

end start
```

实验二的调试结果：

将代码命名为 HW2.asm,生成并查看 HW2.lst

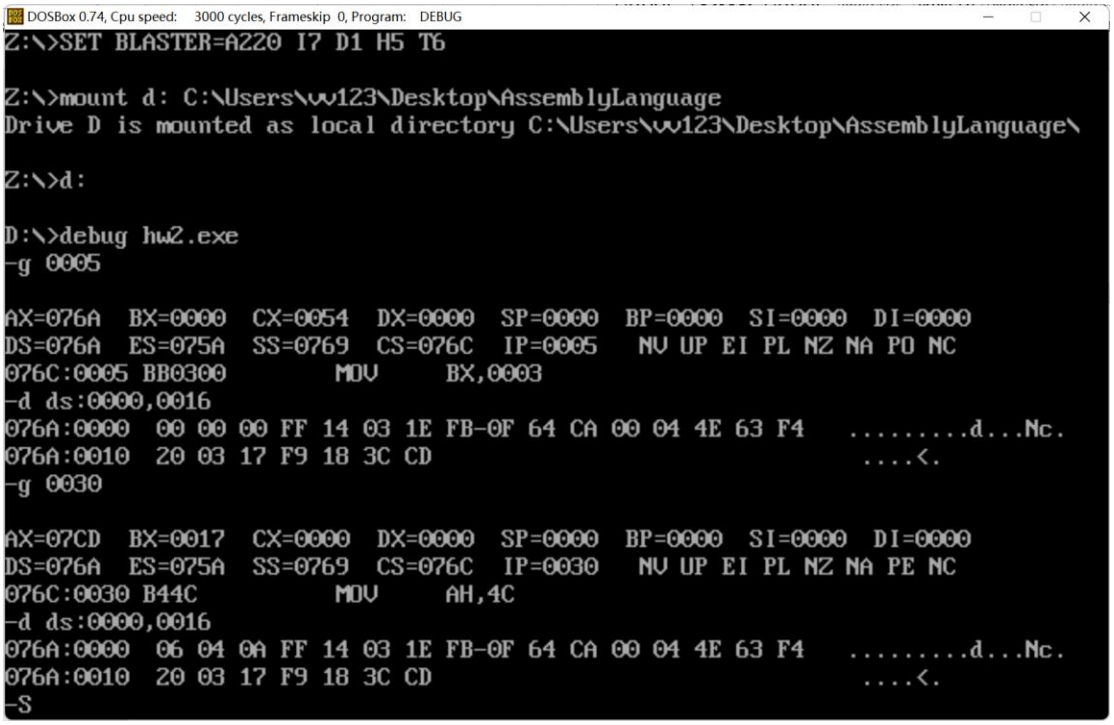


可见，断点设为 0005(完成 data 向 ds 的拷贝)和 0030(运行至 exit 之前)较为合适

输入 debug hw2.exe 开始调试：

输入-g 0005 跳转至 mov ds,ax 之后，此时输入-d ds:0000,0016 查看 data 中 23 个数据的值，可见前三个值 res1、res2、res3 均被初始化为 0

输入-g 0030 跳转至 exit 之前，程序完成了统计，此时此时输入-d ds:0000,0016 查看 data 中 23 个数据的值，可见 res1，res2，res3 的值分别变为 06，04，0A。



即程序统计出小于 0、大于等于 0 小于等于 5、大于 5 的数的个数分别为 6、4、10。

这与我们的预期完全相同。

实验三

选做题目：**从键盘输入两个 2 位十进制正数，对这两个数相加，结果以十六进制形式显示在屏幕上（显示完整算式）。**

实验三完整源代码：

大致思路：

不使用数据段，而是充分利用栈、push 和 pop 指令，将读入的数字经过十位*10+个位计算后分别存于寄存器 AX 和 BX 中；

编写一个子程序，利用循环左移操作，以十六进制形式输出寄存器 AX 中的值；

调用该子程序三次，分别输出(AX)、(BX)和(AX)+(BX),并在中间加上’+’和’=’

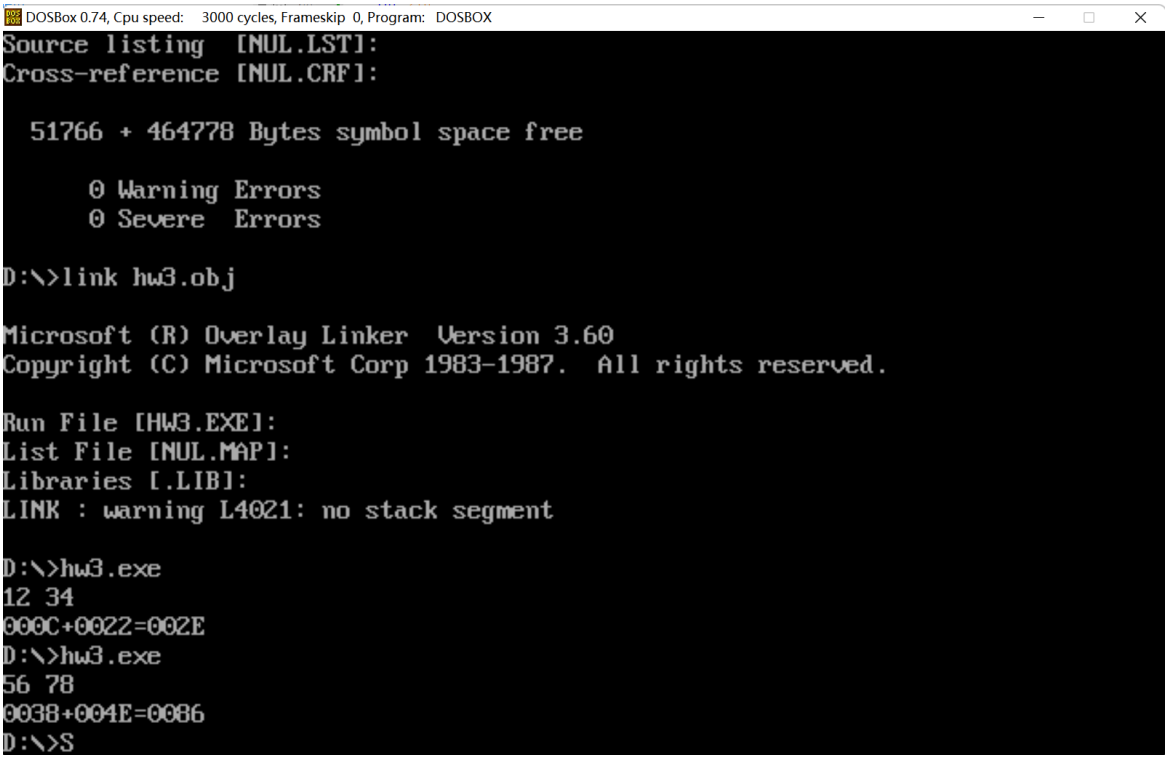
输出 BX 前将 AX 和 BX 借助 DX 交换，再调用子程序

详见代码注释

```
assume cs:code,ss:stack

stack segment
    dw 20 dup(0)
stack ends

code segment
start:
;处理第一个数
    ;处理第一个数的十位
    mov ah,1
    int 21h ;输入字符并将ascii码送入al
    sub al,30h;将ascii码转为数字
    ;将十位数字乘10，结果存于ax
    mov bl,10
    mul bl
    push ax;ax暂存在栈中
    ;处理第一个数的个位
    mov ah,1
    int 21h
    sub al,30h;同上，al得到个位的值
    mov ah,0;将ah置为0，则ax为个位的值
    pop bx  ;将十位的值pop到bx
    add ax,bx;ax得到第一个数的完整值
    push ax;第一个数push到栈中

;读入一个空格(不做处理)
    mov ah,1
    int 21h

;处理第二个数
    mov ah,1
    int 21h
    sub al,30h
    mov bl,10
    mul bl
    push ax
    mov ah,1
    int 21h
    sub al,30h
    mov ah,0
    pop bx  ;以上均与上一段相同
    add ax,bx;ax得到第二个数的完整值
    pop bx;第一个数pop到bx中

    ;swap(ax,bx)
    mov dx,ax    ;int tmp = ax
    mov ax,bx    ;bx = ax
    mov bx,dx    ;ax = tmp
;至此，ax、bx分别存放第一个数和第二个数

    ;输出换行,利用栈保存和恢复ax的值
    push ax
    mov dl,0dh
    mov ah,2
    int 21h
    mov dl,0ah
    mov ah,2
    int 21h
    pop ax
    ;输出第一个数(ax)
    call wrhax;调用下面写的子程序输出ax

    ;输出加号'+'
    push ax
    mov ah,2
    mov dl,'+'
    int 21h
    pop ax

    ;输出第二个数(bx)
    ;首先swap(ax,bx)
    mov dx,ax    ;int tmp = ax
    mov ax,bx    ;bx = ax
    mov bx,dx    ;ax = tmp
    call wrhax;调用下面写的子程序输出ax
    ;再次swap(ax,bx)，复原(其实可省略)
    mov dx,ax    ;int tmp = ax
    mov ax,bx    ;bx = ax
    mov bx,dx    ;ax = tmp

    ;输出等号'='
    push ax
    mov ah,2
    mov dl,'='
    int 21h
    pop ax

    ;相加并输出结果
    add ax,bx
    call wrhax;调用下面写的子程序输出ax


    mov ah,4ch
    int 21h;退出程序

;定义子程序：以16进制形式输出AX中的值
disp_ch proc
    push dx
    push ax
    mov  ah,02h
    mov  dl,al
    int  21h
    pop  ax
    pop  dx
    ret
disp_ch     endp

wrhax   PROC
    push    ax
    push    dx
    mov dx,ax
    mov ch, 4
L1: mov cl, 4
    rol dx, cl
    mov al,dl
    and al,0FH
    add al,30h
    cmp al,3ah
    jl printit
    add al,7h
printit:
    call disp_ch
    dec ch
    jnz L1
    pop dx
    pop ax
    ret
wrhax ENDP

code ends
end start
```

实验三的调试结果：

(1) 直接运行结果：



如图，输入 12 34 后，程序换行并输出 000C+0022=002E

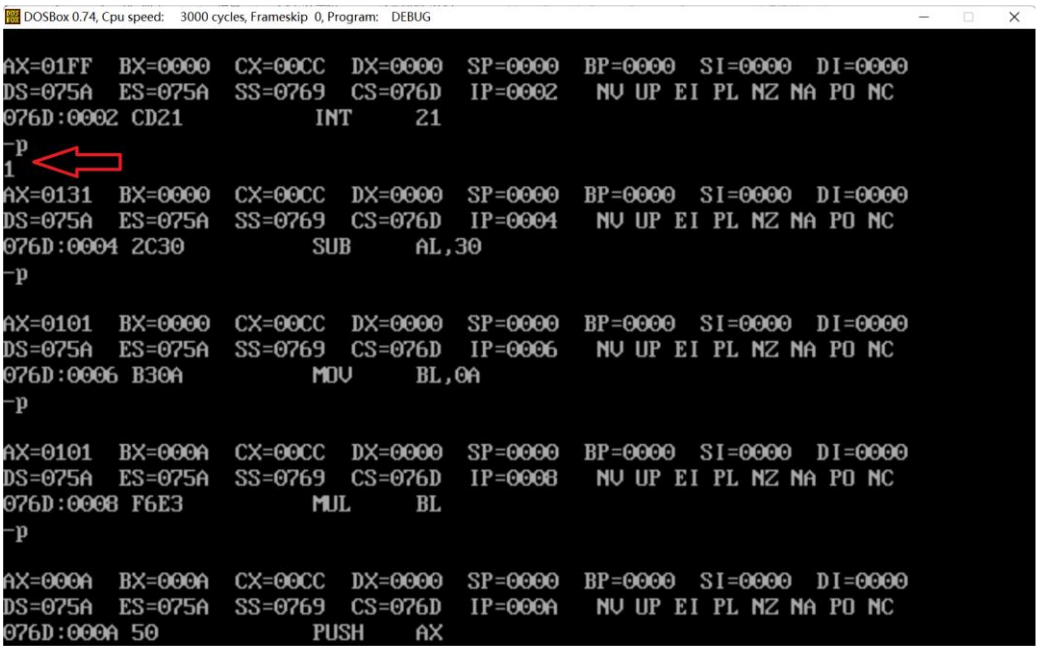
输出 56 78 后，程序换行并输出 0038+004E=0086

算式和结果均正确。

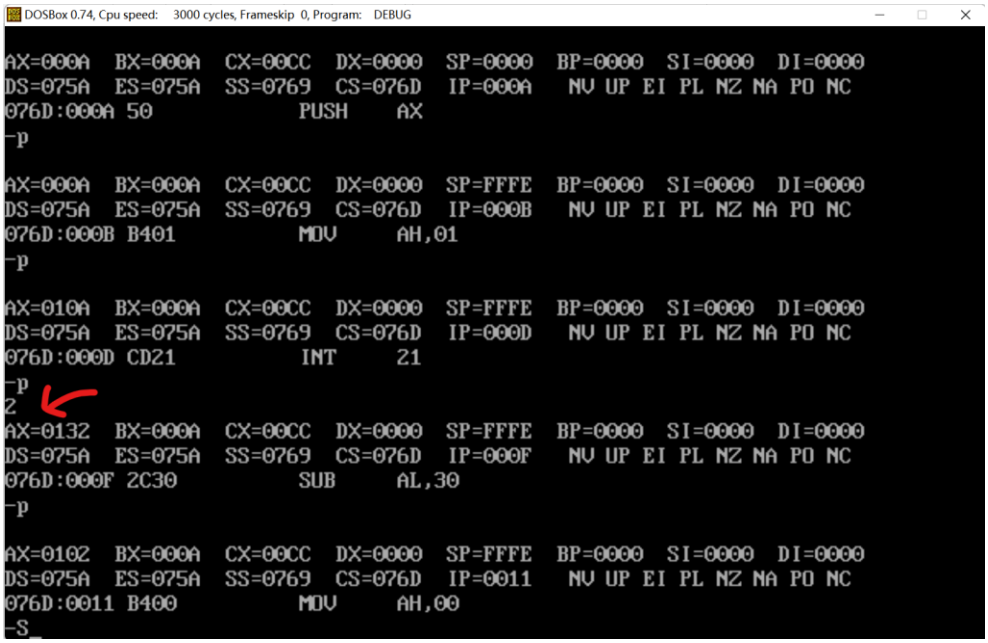
(2) Debug 调试：

由于本程序执行一开始就要输入数字，因此不设断点

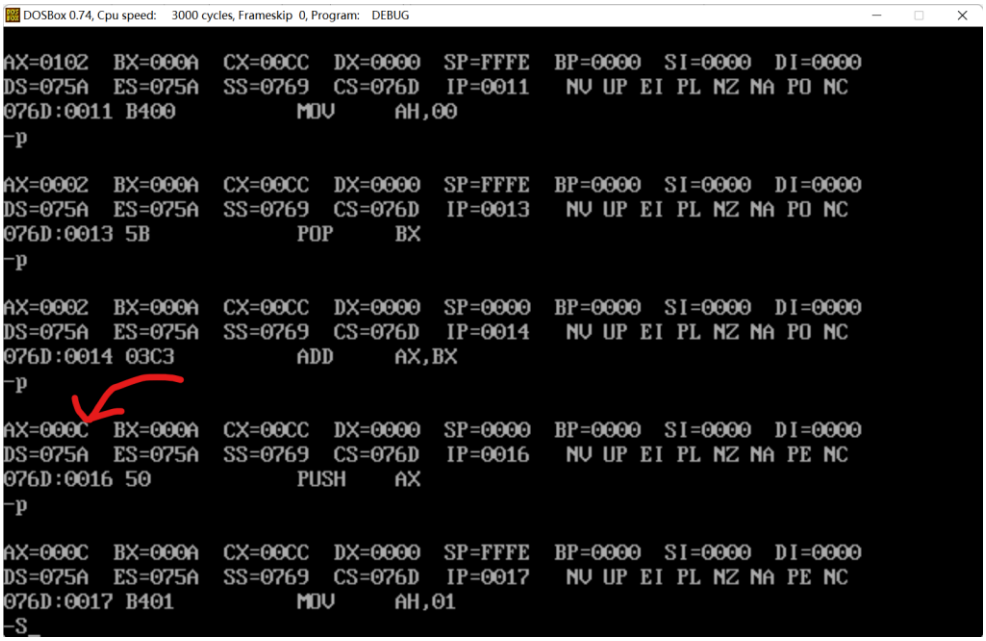
以输入 12 为例：



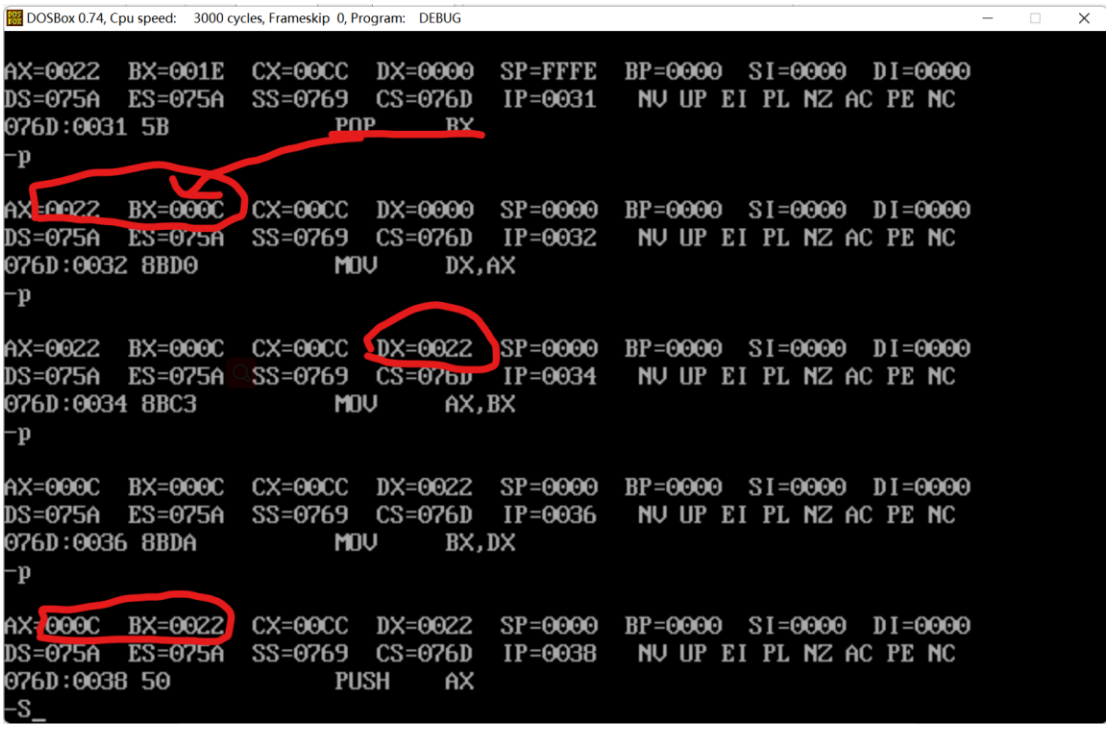
从键盘读入 1 后，’1’的 ascii 码存于 AL 中，将其减 30,并将 AH 置零，然后使用 mul 10 使得 AX 变为的值变为 000A，就得到了十位的值，将其入栈



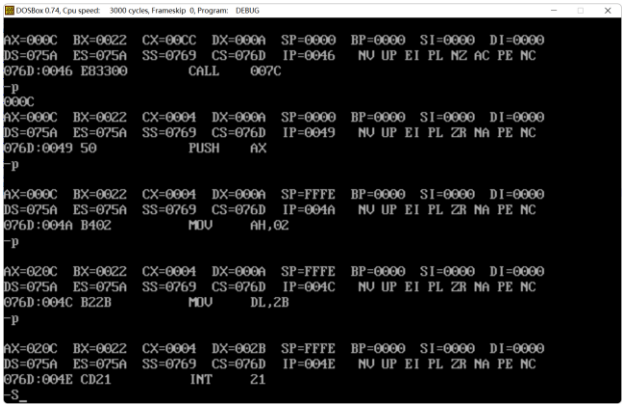
从键盘读入 2 后，以相同的方式使 AX=0002,得到各位的值

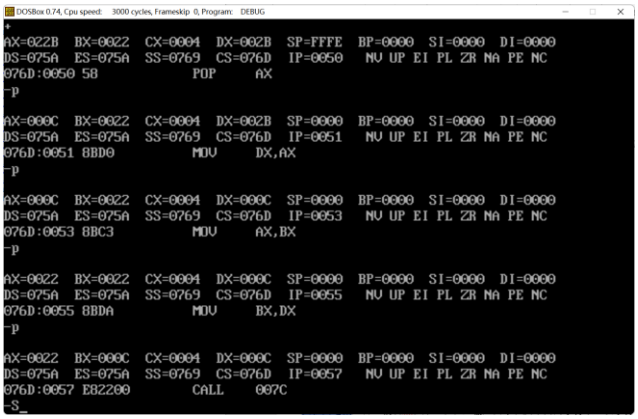


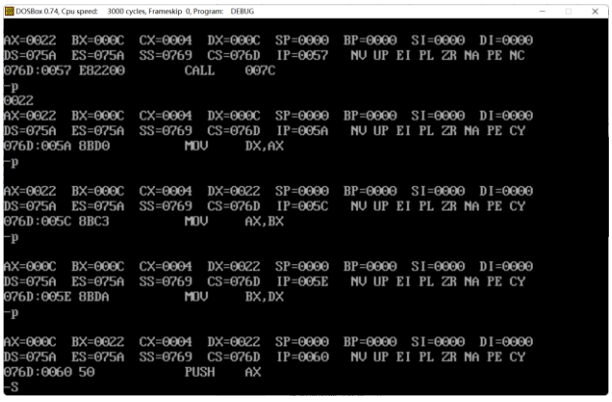
将栈中的十位的值 pop 到 BX，执行 add ax,bx，此时 AX 变为 000C，即十进制数 12。
此时将 AX 入栈，重复上述过程读入后两个数字 34，将 AX 变为 0022。

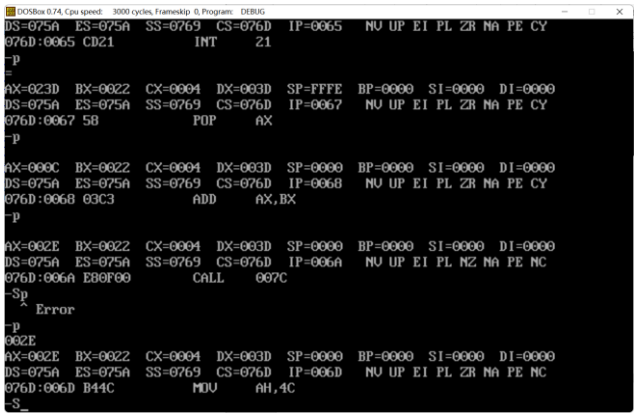


此时将栈中第一个数的值 pop 给 BX，此时 AX=0022=34，BX=000C=12
借助 DX 交换 AX 和 BX 的值后，AX=000C=12,BX=0022=34，符合我们的预期









此后五次调用 007C 处的子程序，分别向终端输出了 000C,+,0022,=,002E,符合我们的预期。

实验四选做题目：**编制一程序，从键盘输入两个长度不同的字符串，设字符串长度小于 25 个字符。要求在屏幕上以右边对齐的形式显示出来。**

实验四完整源代码：

```asm
assume cs:code,ds:data,es:data,ss:stack

data segment
    n = 25
    tip1 db 0ah,0dh,'Input string:','$'
    tip2 db 0ah,0dh,'After alignment:',0ah,0dh,'$'
    str1 db n,0,n dup(' '),0ah,0dh,'$'
    str2 db n,0,n dup(' '),0ah,0dh,'$'
data ends

stack segment
    dw 20h dup(0)
stack ends

code segment
start:
    mov ax,data
    mov ds,ax
    mov es,ax

    ;循环输入两字符串
    mov cx,2
    lea bx,str1
lop:
    lea dx,tip1;显示提示信息
    mov ah,09h
    int 21h
    mov dx,bx;读取字符串
    mov ah,0ah
    int 21h
    lea bx,str2;读取第二个
    loop lop

    ;对齐处理
    lea bx,str1+2
    call aligh
    lea bx,str2+2
    call aligh

    ;输出对齐后字符串
    lea dx,tip2
    mov ah,09h
    int 21h
    lea dx,str1+2
    mov ah,09h
    int 21h
    lea dx,str2+2
    mov ah,09h
    int 21h

    mov ah,4ch
    int 21h;退出

    ;定义子程序：将字符串右对齐
aligh proc
    xor ch,ch
    mov cl,-1[bx]
    mov si,cx
    add si,bx
    dec si
    mov di,bx
    add di,n-1
    std
    rep movsb
    mov cx,n
    sub cl,-1[bx]
    mov al,' '
    rep stosb
    ret
aligh endp

code ends
end start
```

实验四的调试结果：

直接运行 hw4.exe，键盘依次输入 abcdef 和 123

```
D:\>hw4.exe

Input string:abcdef
Input string:123
After alignment:
                    abcdef
                       123
```

可见字符串长度<n 时，程序正确实现实现了右对齐输出

如果字符串的长度=n=25，会发现**无法换行**，也无法继续输入字符

```
D:\>hw4.exe

Input string:0123456789012345678901234_
```

因此实际上字符串长度最大只能达到 24，效果如下

```
D:\>hw4.exe

Input string:01234567890123456789012345
Input string:abcdefghijklmnopqrstuvwx
After alignment:
 01234567890123456789012345
 abcdefghijklmnopqrstuvwx
```

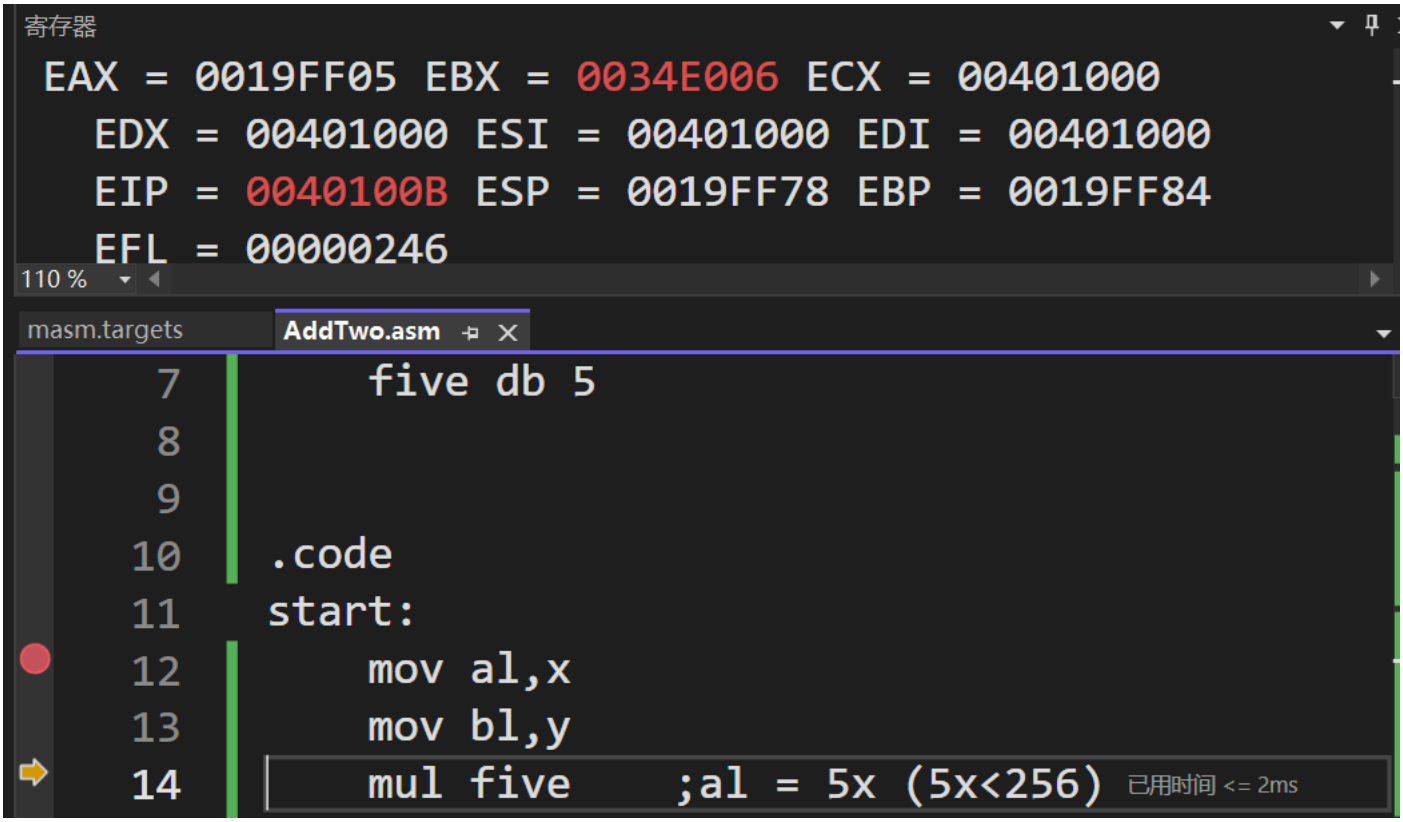
选做题目：**（必须完成一个 32 位汇编程序，该部分算作平时成绩）**

选做题目完整源代码：

```
include irvine32.inc

.data
    x db 5
    y db 6
    z db ?
    five db 5


.code
start:
    mov al,x
    mov bl,y
    mul five    ;al = 5x (5x<256)
    add bl,bl   ;bl = 2y
    add al,bl   ;al = 5x + 2y
    sub al,7    ;al = 5x + 2y - 7
    shr al,1    ;al >>= 1
    mov z,al    ;z = (5x + 2y - 7) / 2
    exit 已用时间 <= 3ms

end start
```

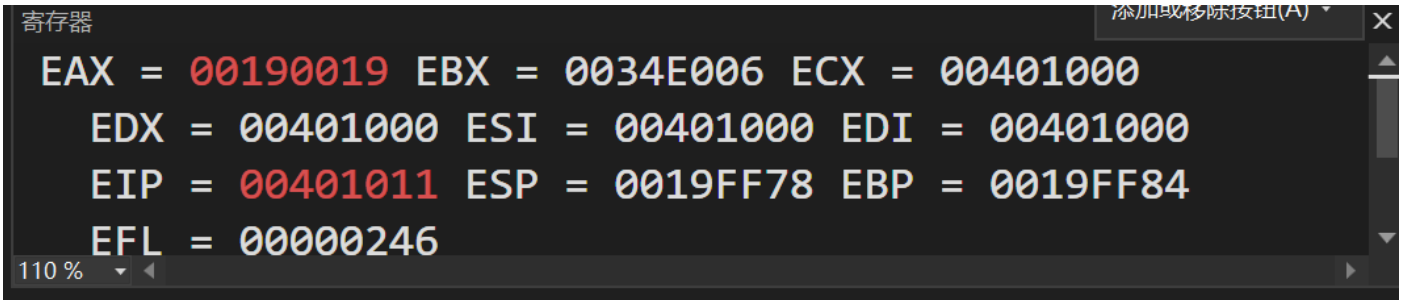
选做题目调试结果：

在 start 处设定断点，单步运行，到 mov bl,y 后，可见 al=05,bl=06

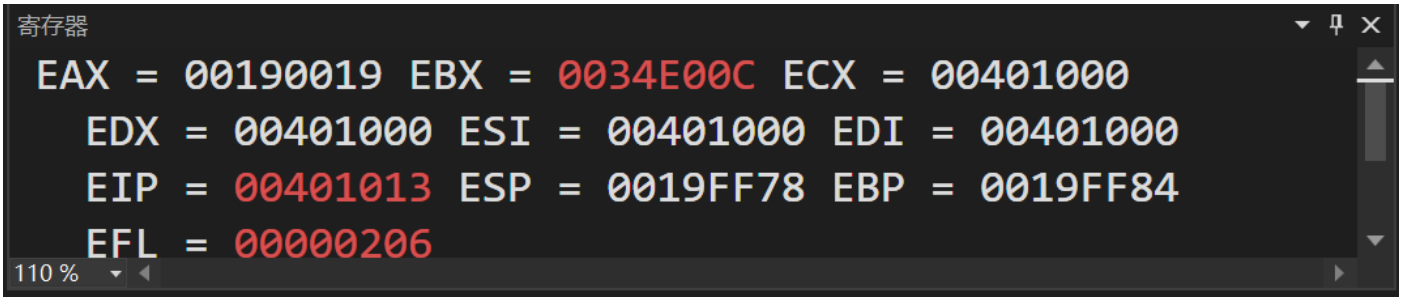


此时可以从监视窗口中看到变量的值 x=5,y=6,z=0

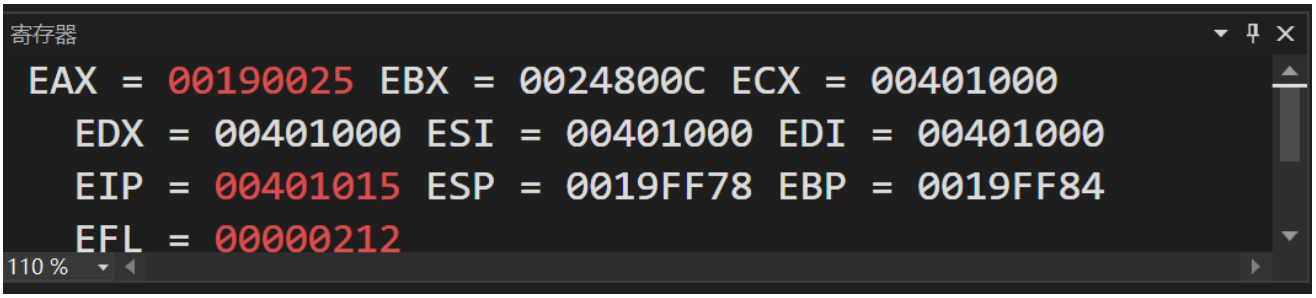
执行 mul five 后，EAX 后四位存放 5*5 的结果 19H=25D



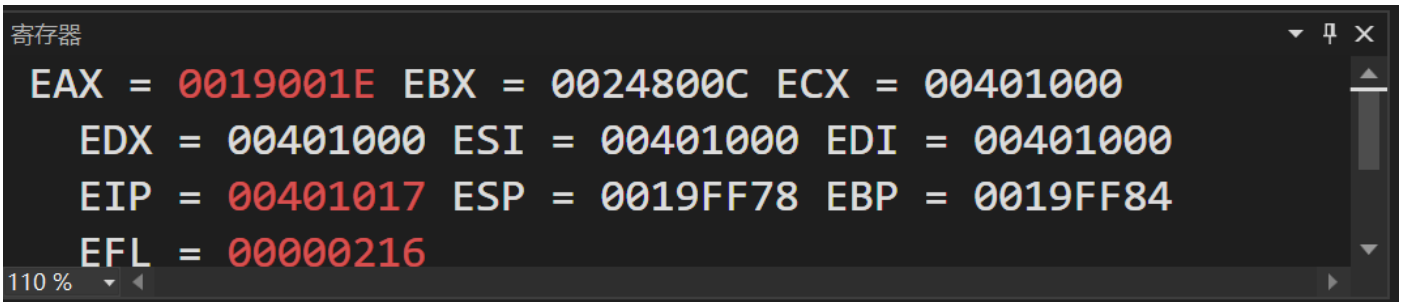
执行 add bl,bl 后，bl 从 06H 变成 0CH



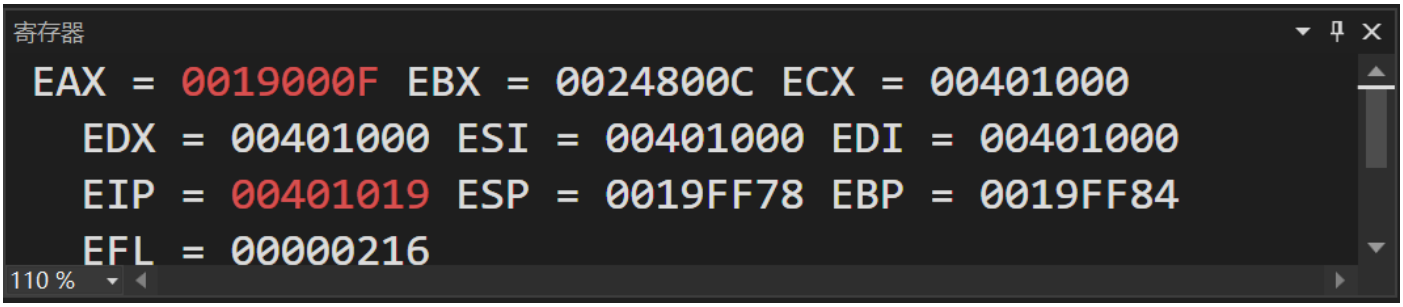
执行 add al,bl 后，al 从 19H 变成 19H+0CH=25H



执行 sub al,7 后，al 从 25H 变成 1EH



执行 shr al,1 后，al 从 1EH=111010B 变成 09H=11101B=15D



最后 mov z,al，在监视窗口中看到 z=15。结果 z=(5x+2y-7)/2，符合预期。

课后思考题：

1、请说明汇编语言的命名规则有哪些？

标识符的命名规则：

(1)字符个数：1~31 个；

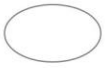
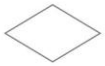
(2)标识符的第一个字符必须是字母、？、@或下划线_中的一个。

(3)从第二个字符开始，可以是字母、数字、？、@或下划线_；

(4)不能使用属于系统专用的保留字。主要有寄存器名称（DS、IP），指令助记符（DIV、POP）,伪指令（SEGMENT、DW），表达式中的运算符（GE、EQ），属性操作符（PTR、OFFSET、SEG）。

2、请说明流程图的基本元素有哪些？

程序流程图一般包含三种基本元素：加工处理步骤、逻辑条件和控制流方向。常用的元素如下

元素	名称	描述
	连接线	显示进程的操作顺序，是带方向的线段，用线段加箭头表示，可以在线上添加备注文字。
	终端	流程的开始或结束。用椭圆表示。
	过程	指流程或节点，表示一个动作或页面状态。用矩形表示。
	判断	表示逻辑判断，用以确定采用哪种路径，通常的操作行为是（Yes）/否（No）。用菱形表示。
	输入或输出	数据进行输入和输出的过程，用平行四边形表示。
	子流程	流程中一部分图形的逻辑关系集合起来形成一个子流程，用具有双重垂直边缘的矩形表示，便于主流程的简化。
	解释说明	对已有元素的解释说明。通常使用带有箭头指向的虚线矩形框表示。

3、请阐述 equ 和=有什么区别？

EQU 伪指令把一个符号名称与**一个整数表达式或一个任意文本**连接起来，它有 3 种格式：

```
name EQU expression
name EQU symbol
name EQU <text>
```

第一种格式中，expression 必须是一个有效整数表达式。第二种格式中，symbol 是一个已存在的符号名称，已经用 = 或 EQU 定义过了。第三种格式中，任何文本都可以出现在<...>内。当汇编器在程序后面遇到 name 时，它就用整数值或文本来代替符号。

在定义非整数值时，EQU 非常有用。比如，可以使用 EQU 定义实数常量：

```
PI EQU <3.1416>
```

【示例 1】下面的例子将一个符号与一个字符串连接起来，然后用该符号定义一个变量：

```
pressKey EQU <"Press any key to continue...", 0>
.data
prompt BYTE pressKey
```

【示例 2】假设想定义一个符号来计算一个 10 x 10 整数矩阵的元素个数。现在用两种不同的方法来进行符号定义，一种用整数表达式，一种用文本。然后把两个符号都用于数据定义：

```
matrix1 EQU 10 * 10
matrix2 EQU <10 * 10>
.data
M1 WORD matrix1
M2 WORD matrix2
```

汇编器将为 M1 和 M2 生成不同的数据定义。计算 matrix1 中的整数表达式，并将其赋给 M1。而 matrix2 中的文本则直接复制到 M2 的数据定义中：

```
M1 WORD 100
M2 WORD 10 * 10
```

与 = 伪指令不同，在同一源代码文件中，用 EQU 定义的符号不能被重新定义。这个限制可以防止现有符号在无意中被赋予新值。

等号伪指令（equal-sign directive）把一个符号名称与一个**整数表达式**连接起来，其语法如下：

name = expression

通常，表达式是一个 32 位的整数值。当程序进行汇编时，在汇编器预处理阶段，所有出现的 name 都会被替换为 expression。假设下面的语句出现在一个源代码文件开始的位置：

COUNT = 500

然后，假设在其后 10 行的位置有如下语句：

mov eax, COUNT

那么，当汇编文件时，MASM 将扫描这个源文件，并生成相应的代码行：

mov eax, 500

用“=”定义的符号，在同一程序内可以被重新定义。

4. 请阐述条件跳转指令和无条件跳转指令在跳转范围上有什么区别？

无条件跳转指令 jmp 既可以只修改 IP，也可以同时修改 CS 和 IP。

Jmp 有很多形式：

段内短转移(IP 变化-128~127)：jmp short [标号]

段内近转移(IP 变化-32768~32767)：jmp near ptr [标号]

段内转移只修改 IP，不修改 CS 的值

段间转移：jmp far ptr [标号] 会同时修改 CS 和 IP

寄存器跳转：jmp [16 位寄存器]　　属于段内转移

单字内存跳转：jmp word ptr [内存单元地址] 属于段内转移

双字内存跳转：jmp dword ptr [内存单元地址] 属于段间转移

jmp指令小结

jmp指令格式	示例
jmp 标号	– 段间转移（远转移）：jmp far ptr 标号 – 段内短转移：jmp short 标号；8位的位移 – 段内近转移：jmp near ptr 标号 ；16位的位移
jmp 寄存器	– jmp bx；16位的位移
jmp 内存单元(表示跳转到的地址)	– 段内转移：jmp word ptr 内存单元地址 ;jmp word ptr [bx] – 段间转移：jmp dword ptr 内存单元地址 ;jmp dword ptr [bx]

在源程序中，不允许使用“jmp 2000:0100”的转移指令实现段间转移
- 这是在 Debug 中使用的汇编指令，汇编编译器并不认识
- 如果在源程序中使用，编译时也会报错。

而所有条件跳转指令都是短转移，只修改 ip，所能变化的 ip 范围为(-128~127)。

条件转移指令

套路：cmp oper1, oper2 ;或者其他影响标志寄存器的指令
jxxx 标号

根据单个标志位转移的指令

指令	含义	测试条件
je/jz	相等/结果为0	ZF=1
jne/jnz	不等/结果不为0	ZF=0
js	结果为负	SF=1
jns	结果非负	SF=0
jo	结果溢出	OF=1
jno	结果溢出	OF=0
jp	奇偶位为1	PF=1
jnp	奇偶位不为1	PF=0
jb/jnae/jc	低于/不高于等于/有借位	CF=1
jnb/jae/jnc	不低于/高于等于/无借位	CF=0

根据无符号数比较结果进行转移的指令

指令	含义	测试条件
jb/jnae/jc	低于则转移	CF=1
jnb/jae/jnc	低于则转移	CF=0
jna/jbe	不高于则转移	CF=1或ZF=1
ja/jnbe	高于则转移	CF=0且ZF=0

根据有符号数比较结果进行转移的指令

指令	含义	测试条件
jl/jnge	小于则转移	SF = 1且OF=0
jnl/jge	不小于转移	SF = 0且OF = 0
jle/jng	小于等于则转移	SF = 0 或 OF = 1
jnle/jg	不小于等于则转移	SF = 1且OF = 1

j-Jump e-Equal n-Not b-Below a-Above L-less g-Greater
s-Sign C-carry p-Parity o-Overflow z-Zero

5.关于对 assume 的理解（test.asm 源代码随后给出）

1）当 assume 没有指定任何一个段，会如何？

因为没有指定代码段，无法通过编译。

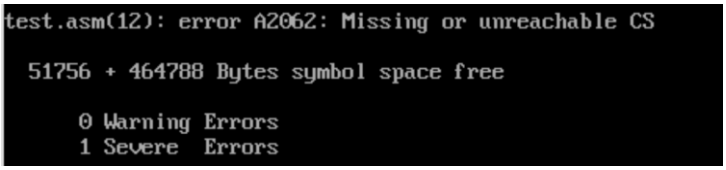
2）当 assume 有声明但是没有给 ds 和 es 赋值，会如何？

DS 和 ES 为默认值，可能无法实现程序预期的功能
使用-r 命令观察发现，默认情况下 ES 似乎会与 DS 相同，都指向第一个数据段

```
D:\>debug test.exe
-r
AX=FFFF  BX=0000  CX=0048  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=076C  IP=0001   NV UP EI PL NZ NA PO NC
076C:0001 B86A07        MOV     AX,076A
```

3）给出不同情况下的运行结果，并分析产生不同结果的原因。

assume 没有指定任何一个段，无法通过编译



assume 只声明代码段，没有给 ds 和 es 赋值，输出 113。因为 DS 和 ES 都默认指向 data1

保留 assume cs:code;,ds:data1,es:data2，输出 123。因为 ds、es 分别与 data1 和 data2 关联。

6.请阐述中断和子程序调用有哪些区别？

1、 两过程定义与作用

子程序是微机基本程序结构中的 1 种，基本程序结构包括顺序（简单）、分支（判断）、循环、子程序和查表等 5 种。

子程序是一组可以公用的指令序列，只要给出子程序的入口地址就能从主程序转入子程序。子程序在功能上具有相对的独立性，在执行主程序的过程中往往被多次调用，甚至被不同的程序所调用。一般微机首先执行主程序，碰到调用指令就转去执行子程序，子程序执行完后，返回指令就返回主程序断点（即调用指令的下一条指令），继续执行没有处理完的主程序，这一过程叫做（主程序）调用子程序过程。

子程序结构可简化程序，防止重复书写错误，并可节省内存空间。计算机中经常把常用的各种通用的程序段编成子程序，提供给用户使用。用户在自己编写的程序中，只要会调用这些子程序，就可大大简化用户编程的困难。

中断是计算机中央处理单元 CPU 与外设 I/O 交换数据的一种方式，除此方式外，还有无条件、条件（查询）、存贮器直接存取 DMA 和 I/O 通道等四种方式。由于无条件不可靠，条件效率低，DMA 和 I/O 通道两方式硬件复杂，而中断方式 CPU 效率高，因此一般大多采用中断方式。中断概念是当计算机正在执行某一（主）程序时，收到一中断请求，如果中断响应条件成立，计算机就把正在执行的程序暂停一下，去响应处理这一请求，执行中断服务程序，处理完服务程序后，中断返回指令使计算机返回原来还没有执行完的程序断点处继续执行，这一过程称为中断过程。有了中断，计算机才能具有并行处理，实时处理和故障处理等重要功能。

2、两过程的联系与区别

联系：两者都需要保护断点（即下一条指令地址）、跳至子程序或中断服务程序、保护现场、子程序或中断处理、恢复现场、恢复断点（即返回主程序）。两者都可实现嵌套，即正在执行的子程序再调另一子程序或正在处理的中断程序又被另一新中断请求所中断，嵌套可为多级。

区别：两者的根本区别主要表现在服务时间与服务对象不一样上。首先，调用子程序过程发生的时间是已知和固定的，即在主程序中的调用指令（CALL）执行时发生主程序调用子程序，调用指令所在位置是已知和固定的。而中断过程发生的时间一般的随机的，CPU 在执行某一主程序时收到中断源提出的中断申请时，就发生中断过程，而中断申请一般由硬件电路产生，申请提出时间是随机的（软中断发生时间是固定的），也可以说，调用子程序是程序设计者事先安排的，而执行中断服务程序是由系统工作环境随机决定的；其次，子程序完全为主程序服务的，两者属于主从关系，主程序需要子程序时就去调用子程序，并把调用结果带回主程序继续执行。而中断服务程序与主程序两者一般是无关的，不存在谁为谁服务的问题，两者是平行关系；第三，主程序调用子程序过程完全属于软件处理过程，不需要专门的硬件电路，而中断处理系统是一个软、硬件结合系统，需要专门的硬件电路才能完全中断处理的过程；第四，子程序嵌套可实现若干级，嵌套的最多级数由计算机内存开辟的堆栈大小限制，而中断嵌套级数主要由中断优先级数来决定，一般优先级数不会很大。

十、实验结论：（联系理论知识进行说明）

本课程完成了汇编源程序的上机调试操作基础训练，分支程序与循环程序设计数制、码制和子程序的编程与调

试和串操作指令及其应用程序的设计与调试运行四次实验。实验结果均达到了预期的目标。

十一、总结及心得体会：

通过学习汇编语言，我对计算机底层的运行机制有了更为深刻的理解。汇编语言虽然复杂，但在编写和调试的过程中，对数据在寄存器和内存之间的流动过程一清二楚，能够与硬件进行充分交互。汇编语言为充分发挥有限的硬件资源而设计，其中的一些思想对程序设计很有启发意义，例如标志寄存器的设计，与自己之前在算法竞赛中接触的一种高精度算法有异曲同工之妙。总而言之，汇编语言是计算机专业的“内功”，果然名不虚传。

十二、对本实验过程及方法、手段的改进建议：

本课程的课时较少，可能不太容易在课上把所有原理讲授清楚，可以给同学们提供一些线上的学习资料，以便预习复习。我在最近在通过该视频汇编语言程序设计_贺利坚主讲_哔哩哔哩_bilibili 学习 8086 汇编，感觉讲得很好，值得推荐。

报告评分：

指导教师签字：