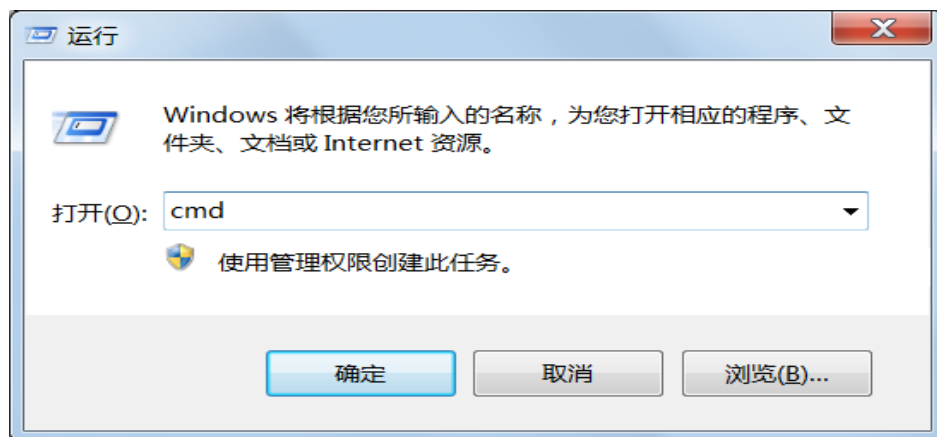


上机指导

2019.06

80x86 汇编语言程序设计的上机练习和编程需要在 DOS 环境下进行，对于 32 位的 Windows7 直接使用下列方式进入 DOS 环境：

单击【开始】菜单，选择【运行】，输入 cmd，进入 DOS 环境，如下图：



或者单击【开始】菜单，选择【所有程序】，选择【附件】，选择【命令提示符】，进入 DOS 环境。如下图：



如果是 64 位的 Windows7 或者 Windows8/10，则需先下载并安装 DOSBox，用以模拟 DOS 环境，请参照“Win8 下 DOSBox 的使用”。

建议将编程所需要的程序都建立在 masm 目录（文件夹）下，如本例是放在 E:\MASM 下。注意在 32 位 WIN7 下应使用系统本身提供的 DEBUG，而在 DOSBox 下则应选择一个 32 位的 DEBUG。

本课程的上机编程主要使用 DEBUG、EDIT、MASM、LINK 这几个程序。在 DOS 下进入 MASM 目录可见：

```
命令提示符
E:\>cd masm
E:\masm>dir
Volume in drive E is ???
Volume Serial Number is EA2D-CA3C

Directory of E:\masm

2018/05/30  21:37    <DIR>          .
2018/05/30  21:37    <DIR>          ..
2016/07/06  21:21        28,427 CREF.EXE
2016/07/06  21:21         413 EDIT.COM
2016/07/06  21:21       17,898 EDIT.HLP
2016/07/06  21:21       49,661 LIB.EXE
2016/07/06  21:21       65,475 LINK.EXE
2016/07/06  21:21      110,703 MASM.EXE
2016/07/06  21:21      194,309 QBASIC.EXE
2016/07/06  21:21     130,881 QBASIC.HLP
2016/07/06  21:21        132 QBASIC.INI
2016/07/06  21:21       4,964 SHOW.EXE
                10 File(s)      602,863 bytes
                2 Dir(s)  151,515,635,712 bytes free

E:\masm>
```

或在 DOXBox 下:

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
Z:\>MOUNT C E:\MASM
Drive C is mounted as local directory E:\MASM\

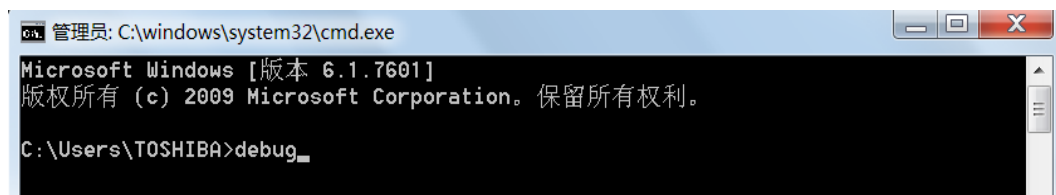
Z:\>C:
C:\>DIR
Directory of C:\.
.                <DIR>          30-05-2018 17:41
..               <DIR>          01-01-1980 00:00
CREF      EXE          28,427 06-07-2016 21:21
DEBUG     EXE          20,634 14-04-2018 21:38
EDIT      COM           413 06-07-2016 21:21
EDIT      HLP          17,898 06-07-2016 21:21
LIB       EXE          49,661 06-07-2016 21:21
LINK      EXE          65,475 06-07-2016 21:21
MASM      EXE         110,703 06-07-2016 21:21
QBASIC    EXE          194,309 06-07-2016 21:21
QBASIC    HLP          130,881 06-07-2016 21:21
QBASIC    INI           132 06-07-2016 21:21
SHOW      EXE           4,964 06-07-2016 21:21
    11 File(s)      623,497 Bytes.
    2 Dir(s)      262,111,744 Bytes free.

C:\>
```

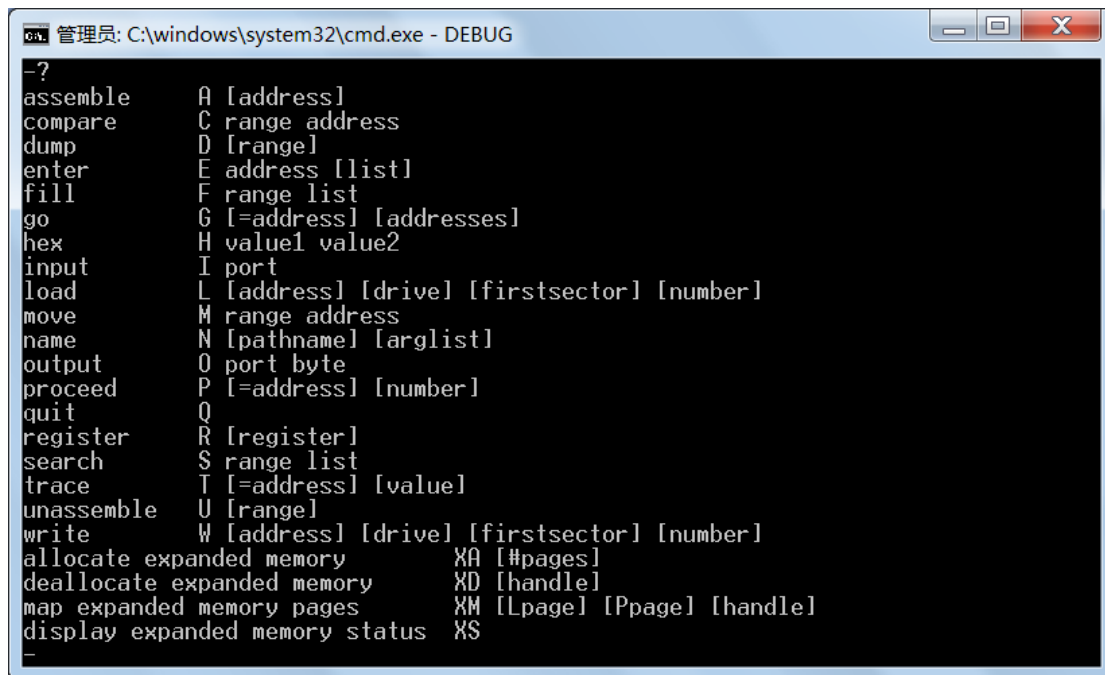
下面以 Windows7 的命令提示符 DOS 环境为例介绍 DEBUG 的使用,以及 EDIT(编辑)、MASM(汇编)、LINK(连接)和程序运行的全过程。DOSBox 环境下的操作类似。

一、DEBUG 的常用命令

在 DOS 提示符后面键入 DEBUG 并回车确认,即进入 DEBUG 操作窗口,如下图:



进入 DEBUG 后，在 DEBUG 提示符“-”后键入?，可以显示 DEBUG 的所有命令及其格式，如下图：



下面介绍常用的 DEBUG 命令：

特别提醒：在 DEBUG 下的显示的数据都是十六进制数且省去了后缀 H，若要在 DEBUG 下键入数据时也只能是十六进制数且不带后缀 H。命令或者数据的字符的大小写一样。

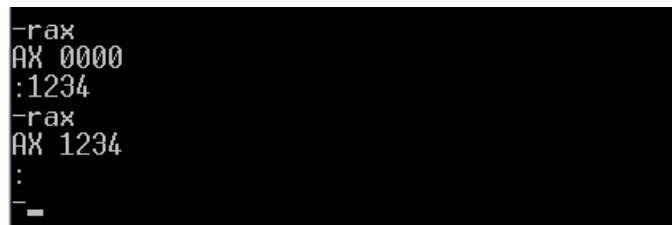
（1）查看和修改寄存器内容的命令 R

命令格式：

①-R[寄存器名]

②-RF

操作说明：格式①用于查看和修改指定寄存器的内容。[]表示寄存器名是可选项，下同。R 命令后跟寄存器名，则显示该寄存器的内容，在“:”后可以键入 1~4 位十六进制数，按回车键修改完成。若不需要修改其内容，可直接按回车键。如下图：



若省略寄存器名，则显示所有寄存器的内容和 8 个标志位的状态（前两行），以及当前 CS:IP 所指的机器指令代码及其反汇编（后一行），如下图：

```
-r
AX=1234 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13A3 ES=13A3 SS=13A3 CS=13A3 IP=0100 NV UP EI PL NZ NA PO NC
13A3:0100 0000 ADD [BX+SI],AL DS:0000=CD
-
```

格式②用于查看和修改标志位的状态。若不需要修改任何标志位，可直接按回车键，如下图：

```
-rf
NV UP EI PL NZ NA PO NC -
-
```

若需要修改一个或多个标志位，可以键入其相反的值。键入时各标志之间可以没有空格且无关顺序，修改 1~8 个标志都可以，修改后按回车键。下图中先修改了 CF、ZF、SF、OF、DF 共 5 个标志位，键入时无关原来的顺序且没有空格，修改后用 RF 查看确认，接着再次修改了 CF、OF、SF 和 IF，这次键入时有留空格的，修改后再次用 RF 查看确认。

```
-rf
NV UP EI PL NZ NA PO NC -cyzrngovdn
-rf
OV DN EI NG ZR NA PO CY -nc nv pldi
-rf
NV DN DI PL ZR NA PO NC -
-
```

标志位和在 DEBUG 下的状态符号的对照关系下表：

标志位		1	0
OF	溢出标志（是/否）	OV	NV
DF	方向标志（增/减）	DN	UP
IF	中断标志（允许/关闭）	EI	DI
SF	符号标志（负/正）	NG	PL
ZF	零标志（是/否）	ZR	NZ
AF	辅助进位（有/无）	AC	NA
PF	奇偶标志（偶/奇）	PE	PO
CF	进位标志（是/否）	CY	NC

（2）显示内存单元内容的命令 D

命令格式：

①-D[地址]

②-D [地址范围]

操作说明：地址的格式为“段地址:偏移地址”两个部分组成，也可以只有偏移地址，这时的段地址默认为当前的 DS 的内容。

格式①用于显示从指定地址开始的 128 个字节单元的内容，每一行的左边显示内存单元的地址，中间显示的是 16 个存储单元的内容，最右边区域显示的是这一行 16 个单元所对应的 ASCII 码，非可显示标准 ASCII 码则用“.”代替。若不指定地址，则显示当前数据段、当前偏移地址开始的 126 个字节单元的内容。下图中，第一次用 D 命令指定地址，显示从当前段地址、偏移地址为 1000H 开始的 128 个字节单元的内容；第二次用 D 命令，没有指定地址，则从当前的地址处开始显示 128 个单元的内容。

```

-d1000
13A3:1000  41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50  ABCDEFGHIJKLMNOP
13A3:1010  51 52 53 54 55 56 57 58-59 5A 20 20 30 31 33 34  QRSTUVWXYZ 0134
13A3:1020  35 36 37 38 39 3A 2B 3C-3D 3C 3E 3F 40 00 00 00  56789: +<=<>?@...
13A3:1030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  56789: +<=<>?@...
13A3:1040  61 62 63 64 65 66 67 68-69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
13A3:1050  71 72 73 74 75 76 77 78-79 7A 20 20 20 7E 00 00  qrstuvwxyz ~..
13A3:1060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  56789: +<=<>?@...
13A3:1070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  56789: +<=<>?@...
-d
13A3:1080  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  56789: +<=<>?@...
13A3:1090  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  56789: +<=<>?@...
13A3:10A0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  56789: +<=<>?@...
13A3:10B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  56789: +<=<>?@...
13A3:10C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  56789: +<=<>?@...
13A3:10D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  56789: +<=<>?@...
13A3:10E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  56789: +<=<>?@...
13A3:10F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00  56789: +<=<>?@...

```

格式②用于显示指定地址范围内的内存单元。

这里的地址范围有两种形式：

形式 1：起始地址 结束地址

形式 2：起始地址 L 长度

例如：上图中显示的从 1000H 开始的 128 个字节的内容，同

-D DS:1000 107F

效果是一样的。

同样，下面两条命令都是显示当前数据段偏移地址从 1000H 到 1030H 的单元内容，如下图。

-D 1000 1030

-D 1000 L31

注意地址从 1000H 到 1030H 的字节单元共有 31H 个，不是 30H。

```

-D1000 1030
13A3:1000  41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50  ABCDEFGHIJKLMNOP
13A3:1010  51 52 53 54 55 56 57 58-59 5A 20 20 30 31 33 34  QRSTUVWXYZ 0134
13A3:1020  35 36 37 38 39 3A 2B 3C-3D 3C 3E 3F 40 00 00 00  56789: +<=<>?@...
13A3:1030  00
-D1000 L31
13A3:1000  41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50  ABCDEFGHIJKLMNOP
13A3:1010  51 52 53 54 55 56 57 58-59 5A 20 20 30 31 33 34  QRSTUVWXYZ 0134
13A3:1020  35 36 37 38 39 3A 2B 3C-3D 3C 3E 3F 40 00 00 00  56789: +<=<>?@...
13A3:1030  00

```

（3）输入汇编指令的命令 A

命令格式：

①-A[地址]

操作说明：从指定地址开始输入汇编指令，按回车后即把该指令汇编成了机器码，顺序存放在指定地址开始的单元中。该命令常用的格式是：

-A100 ;指定从代码段的偏移地址 100H 开始输入汇编指令并汇编。

如果不指定地址会从当前地址处继续输入指令和汇编，如下例：

```

-a100
13A8:0100 MOV AX,1234
13A8:0103 MOV BX,5678
13A8:0106
-A
13A8:0106 ADD AX,BX
13A8:0108 INT 3
13A8:0109

```

结束输入，需退出输入时，直接按回车键即可。

这时用 D 命令可以看到机器码已经存放在内存中，如下图：

```
-d13a8:100
13A8:0100 B8 34 12 BB 78 56 01 D8-CC 00 00 00 00 00 00 00 .4...xV.....
13A8:0110 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
13A8:0120 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
```

图中可以看到，B8 34 12 是指令 MOV AX,1234H 的机器码，BB 78 56 是 MOV BX,5678H 的机器码，D8 01 是 ADD AX,BX 的机器码，CC 是 INT 3 的机器码。但是这样看到的机器码很难理解，所以下面介绍反汇编 U 命令。

（4）反汇编命令 U

命令格式：

①-U[地址]

②-U [地址范围]

操作说明：格式①是从指定地址开始，反汇编 32 个字节。若省略地址，则从上一个 U 命令反汇编的最后一条指令的下一条指令的地址作为起始地址开始汇编；若之前没有用过 U 命令，则以由 DEBUG 初始化的段寄存器作为段地址，以 100H 作为偏移地址开始汇编。如下图：

```
-u100
13A8:0100 B83412      MOV     AX,1234
13A8:0103 BB7856      MOV     BX,5678
13A8:0106 01D8        ADD     AX,BX
13A8:0108 CC          INT     3
13A8:0109 0000        ADD     [BX+SI],AL
13A8:010B 0000        ADD     [BX+SI],AL
13A8:010D 0000        ADD     [BX+SI],AL
13A8:010F 0000        ADD     [BX+SI],AL
13A8:0111 0000        ADD     [BX+SI],AL
13A8:0113 0000        ADD     [BX+SI],AL
13A8:0115 0000        ADD     [BX+SI],AL
13A8:0117 0000        ADD     [BX+SI],AL
13A8:0119 0000        ADD     [BX+SI],AL
13A8:011B 0000        ADD     [BX+SI],AL
13A8:011D 0000        ADD     [BX+SI],AL
13A8:011F 0000        ADD     [BX+SI],AL
```

格式②可以对指定范围的内存单元进行反汇编，范围同样可以是：

形式 1：起始地址 结束地址

形式 2：起始地址 L 长度

例如：

-U100 108

-U100 L9

这两条命令的效果是一样的，如下图：

```
-U100 108
13A8:0100 B83412      MOV     AX,1234
13A8:0103 BB7856      MOV     BX,5678
13A8:0106 01D8        ADD     AX,BX
13A8:0108 CC          INT     3
-U100 L9
13A8:0100 B83412      MOV     AX,1234
13A8:0103 BB7856      MOV     BX,5678
13A8:0106 01D8        ADD     AX,BX
13A8:0108 CC          INT     3
```

(5) 执行命令 G

命令格式:

①-G[断点地址]

②-G [=起始地址] [断点地址]

操作说明: 格式①是程序从当前 IP 所指向的指令开始执行, 执行到断点地址停下了, 并显示 CPU 各寄存器的内容和标志位的状态, 以及下一条待执行指令的地址、机器码及对应的汇编指令。若省略断点地址, 则连续执行整个程序。

格式②是程序从指定地址开始执行, 运行到断点地址停下来, 并显示 CPU 各寄存器的内容和标志位的状态, 以及下一条待执行指令的地址、机器码及对应的汇编指令。格式②是在 DEBUG 下运行程序的常用命令格式。例如:

-G109

-G=100 109

这两条命令的效果是一样的, 如下图:

```
-r
AX=68AC BX=5678 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13A3 ES=13A3 SS=13A3 CS=13A8 IP=0100 NV UP EI PL NZ NA PE NC
13A8:0100 B83412 MOV AX,1234
-g109

AX=68AC BX=5678 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13A3 ES=13A3 SS=13A3 CS=13A8 IP=0108 NV UP EI PL NZ NA PE NC
13A8:0108 CC INT 3
-g=100 109

AX=68AC BX=5678 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13A3 ES=13A3 SS=13A3 CS=13A8 IP=0108 NV UP EI PL NZ NA PE NC
13A8:0108 CC INT 3
```

需要注意的是, 上图中一开始的 IP=100H, 所以命令“-G109”就等同于“-G=100 109”, 执行的结果是一样的。如果不清楚当时的 IP 的值, 那么命令“-G109”是不可靠的。

从上图中也可以看到, 程序执行后, AX=68ACH, 完成了两数的相加。

通常调试程序时需要跟踪或单步执行指令, 所以下面介绍 T 命令。

(6) 追踪(单步执行)命令 T

命令格式:

①-T[=地址]

②-T [=地址] [值]

操作说明: 格式①是从当前 CS:IP 开始执行到指定地址时停下, 若省略地址时则从当前 CS:IP 开始单步执行, 每一次 T 命令执行一条指令。停下时显示 CPU 各寄存器的内容和标志位的状态, 以及下一条待执行指令的地址、机器码及对应的汇编指令。

格式②可以对多条指令进行跟踪, 即连续执行由值所指定的几条指令, 每次显示 CPU 各寄存器的内容和标志位的状态, 以及下一条指令。该格式在调试循环程序和子程序时特别有用。

例如: 在内存 13A8:0100H 到 108H 的单元中有一两数相加的指令段, 用反汇编命令 U 可见, 如下图。执行“-T=100”, 实际上是从 100H 起执行一条指令, 并显示寄存器的内容及标志位状态, 以后每次执行“-T”都执行一条指令, 显示一次当前寄存器的内容和下一条指令, 从而实现单步追踪指令的执行。

```

-u100 108
13A8:0100 B83412      MOV     AX,1234
13A8:0103 BB7856      MOV     BX,5678
13A8:0106 01D8        ADD     AX,BX
13A8:0108 CC          INT     3
-t=100

AX=1234 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13A3 ES=13A3 SS=13A3 CS=13A8 IP=0103  NV UP EI PL NZ NA PE NC
13A8:0103 BB7856      MOV     BX,5678
-t

AX=1234 BX=5678 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13A3 ES=13A3 SS=13A3 CS=13A8 IP=0106  NV UP EI PL NZ NA PE NC
13A8:0106 01D8        ADD     AX,BX
-t

AX=68AC BX=5678 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13A3 ES=13A3 SS=13A3 CS=13A8 IP=0108  NV UP EI PL NZ NA PE NC
13A8:0108 CC          INT     3
-

```

如果同样的程序段，执行的是“-T=100 3”，追踪执行到 100H 单元所指的指令，显示当前寄存器的内容和下一条指令，接着继续以单步方式再执行 3 条指令。如下图：

```

-u100 108
13A8:0100 B83412      MOV     AX,1234
13A8:0103 BB7856      MOV     BX,5678
13A8:0106 01D8        ADD     AX,BX
13A8:0108 CC          INT     3
-t=100 3

AX=1234 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13A3 ES=13A3 SS=13A3 CS=13A8 IP=0103  NV UP EI PL NZ NA PE NC
13A8:0103 BB7856      MOV     BX,5678

AX=1234 BX=5678 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13A3 ES=13A3 SS=13A3 CS=13A8 IP=0106  NV UP EI PL NZ NA PE NC
13A8:0106 01D8        ADD     AX,BX

AX=68AC BX=5678 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=13A3 ES=13A3 SS=13A3 CS=13A8 IP=0108  NV UP EI PL NZ NA PE NC
13A8:0108 CC          INT     3
-

```

(7) 退出 DEBUG 命令 Q

命令格式：

-Q

操作说明：退出 DEBUG 程序，返回 DOS。

二、汇编语言程序上机过程与程序的跟踪调试

汇编语言程序从源程序编写到生成可执行文件需经过编辑、汇编、连接三个步骤，可执行文件（.EXE）可以直接运行，也可以在 DEBUG 下运行和调试。

汇编程序 MASM 是用于汇编的系统软件，这里我们使用 Microsoft 公司推出的宏汇编程序 MASM（Micro Assembler）5.10 版本。

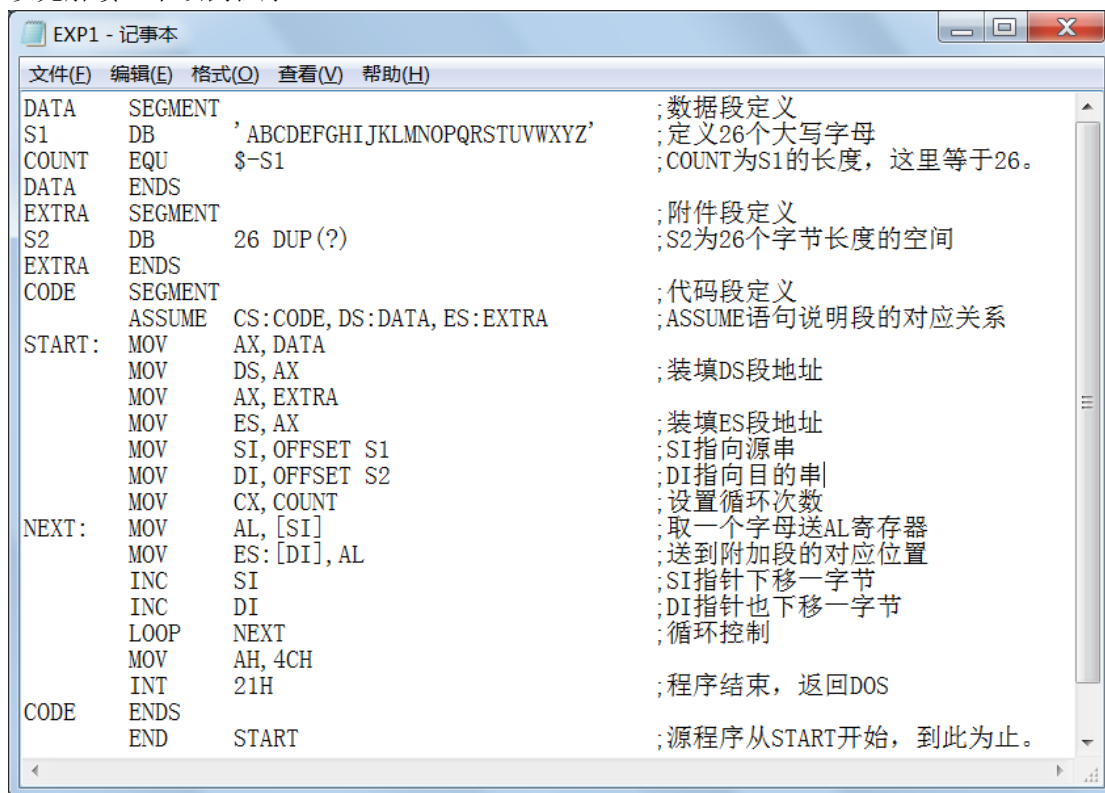
下面我们通过一个两数相加的例子来说明汇编语言程序的上机过程。假设上机所需要的程序，包括 EDIT、MASM、LINK 等以及部分例题都已存在于系统的 E 盘的 MASM 子目录下。

（课堂上我们演示时都用的是 D 盘，由于本人现在的电脑没有 D 盘可用，故以 E 盘为例。）

一、输入（编辑）源程序

输入和编辑源程序可以使用文本编辑器，如 Edit、记事本、Ultraedit 等，以及高级语言的编辑环境，编辑源程序，编辑结束时一定要保存成.asm 为扩展名的源程序。

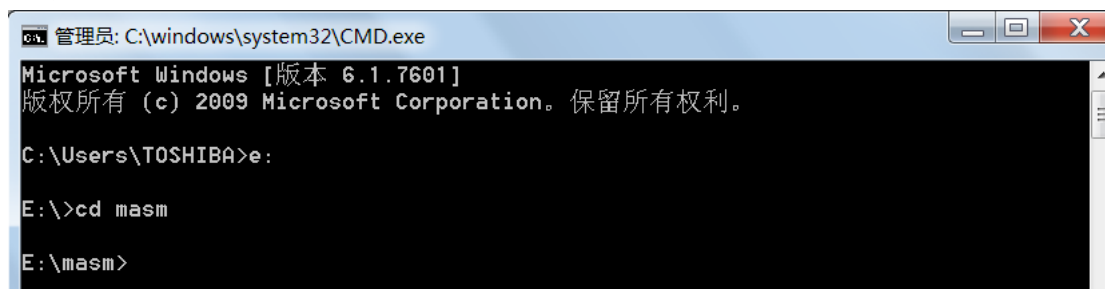
通常使用记事本来编辑源程序的好处是可以加中文注释，如下图，我们这里通过中文注释可以先解读一下该例程序。



```
EXP1 - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
DATA SEGMENT ;数据段定义
S1 DB 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' ;定义26个大写字母
COUNT EQU $-S1 ;COUNT为S1的长度，这里等于26。
DATA ENDS
EXTRA SEGMENT ;附件段定义
S2 DB 26 DUP(?) ;S2为26个字节长度的空间
EXTRA ENDS
CODE SEGMENT ;代码段定义
ASSUME CS:CODE, DS:DATA, ES:EXTRA ;ASSUME语句说明段的对应关系
START: MOV AX, DATA ;装填DS段地址
MOV DS, AX
MOV AX, EXTRA ;装填ES段地址
MOV ES, AX
MOV SI, OFFSET S1 ;SI指向源串
MOV DI, OFFSET S2 ;DI指向目的串
MOV CX, COUNT ;设置循环次数
NEXT: MOV AL, [SI] ;取一个字母送AL寄存器
MOV ES: [DI], AL ;送到附加段的对应位置
INC SI ;SI指针下移一字节
INC DI ;DI指针也下移一字节
LOOP NEXT ;循环控制
MOV AH, 4CH ;程序结束，返回DOS
INT 21H
CODE ENDS
END START ;源程序从START开始，到此为止。
```

通常情况下使用 Edit 来编辑源程序。过程如下：

（1）进入 DOS 环境，键入“e:”，回车，切换到 E 盘，键入“cd masm”，回车，进入 MASM 文件夹，如下图：



```
管理员: C:\windows\system32\CMD.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

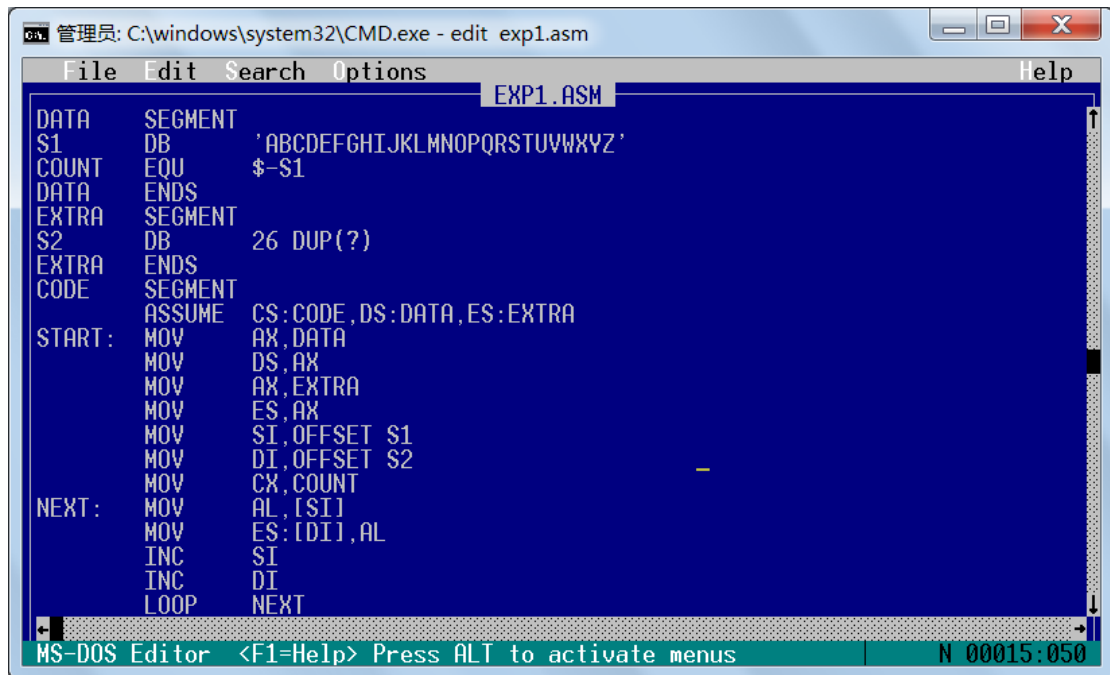
C:\Users\TOSHIBA>e:

E:\>cd masm

E:\masm>
```

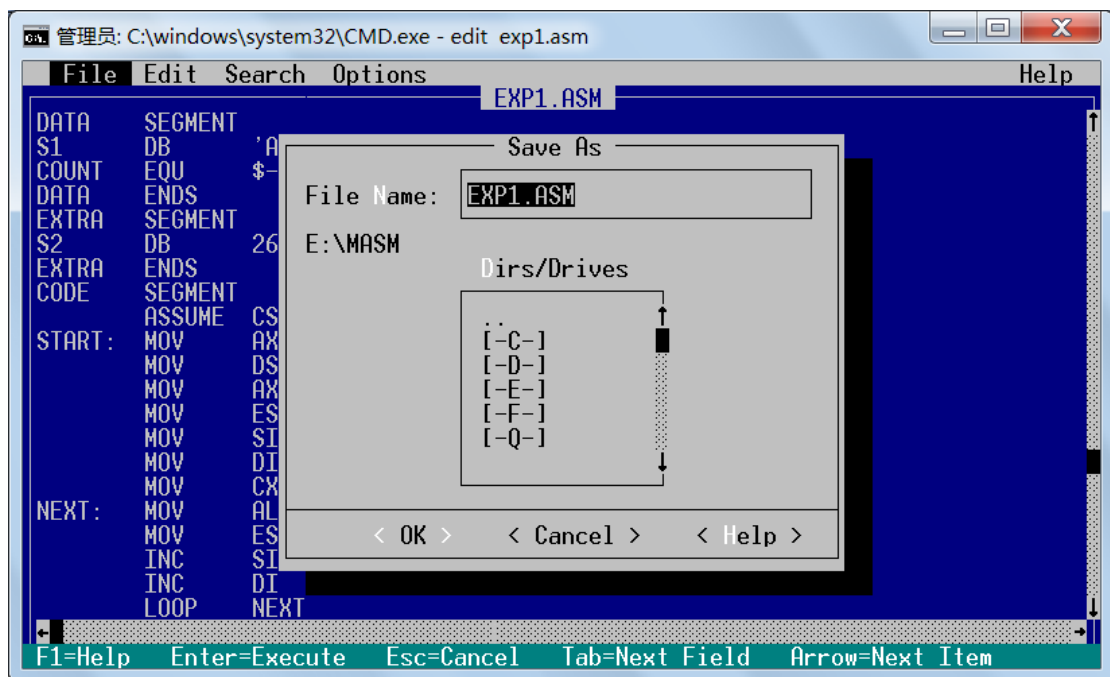
（2）输入“edit exp1.asm”，回车，进入 Edit 全屏幕编辑界面。由于 EXP1.ASM 已经存在，所以直接显示在窗口，可对其继续编辑。如果是新文件，需逐句输入、编辑。如下图，图中

只显示源程序的部分，可通过移动光标或滚动条来编辑整个源程序。



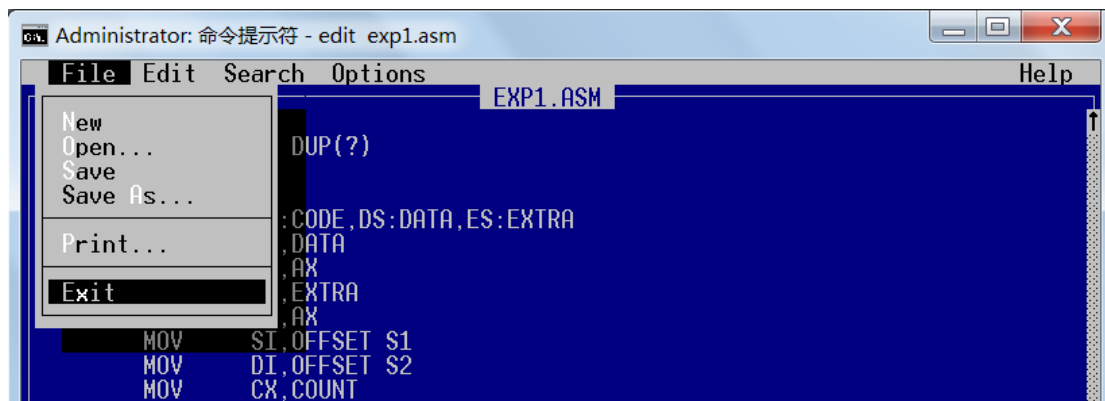
```
DATA SEGMENT
S1 DB 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
COUNT EQU $-S1
DATA ENDS
EXTRA SEGMENT
S2 DB 26 DUP(?)
EXTRA ENDS
CODE SEGMENT
ASSUME CS:CODE, DS:DATA, ES:EXTRA
START: MOV AX, DATA
        MOV DS, AX
        MOV AX, EXTRA
        MOV ES, AX
        MOV SI, OFFSET S1
        MOV DI, OFFSET S2
        MOV CX, COUNT
NEXT:  MOV AL, [SI]
        MOV ES:[DI], AL
        INC SI
        INC DI
        LOOP NEXT
```

(3) 点击“File”菜单，选择“Save”或“Save As”保存文件。注意文件的扩展名一定要是“.ASM”，这里保存的文件名是EXP1.ASM。

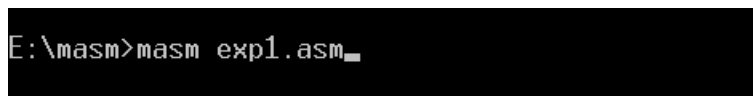


2. 汇编

(1) 点击“File”菜单，选择“Exit”，退出编辑窗口，返回DOS。



(2) 在 DOS 提示符下键入 “ masm exp1.asm”，对汇编源程序进行汇编。如下图：



汇编语言源程序的汇编过程是利用汇编程序（MASM）对已经编辑好的源程序（.ASM）进行汇编，将源程序文件中的汇编指令逐条翻译成机器指令，并完成源程序中伪指令所指出的各种操作。

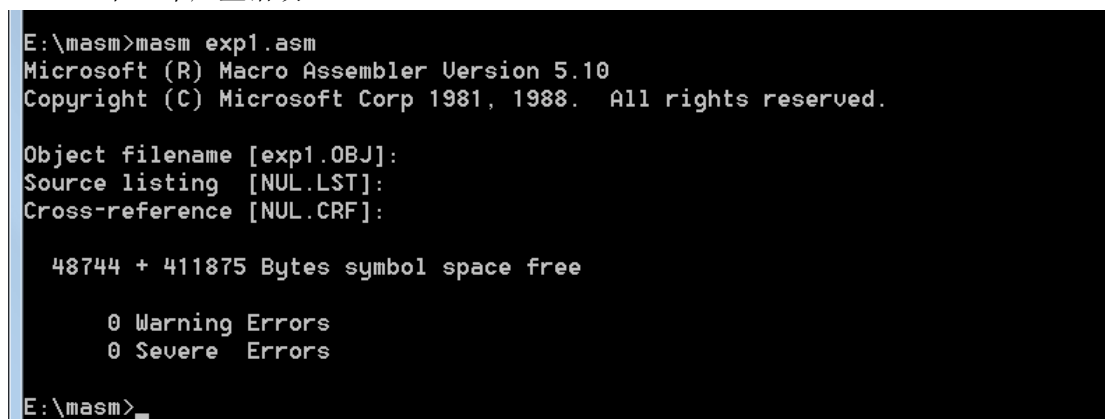
汇编程序将对源程序进行两遍扫描，若程序文件中存在语法错误，则结束汇编，给出出错信息。

(3) 在汇编过程中，根据屏幕提示要回答 3 项内容：

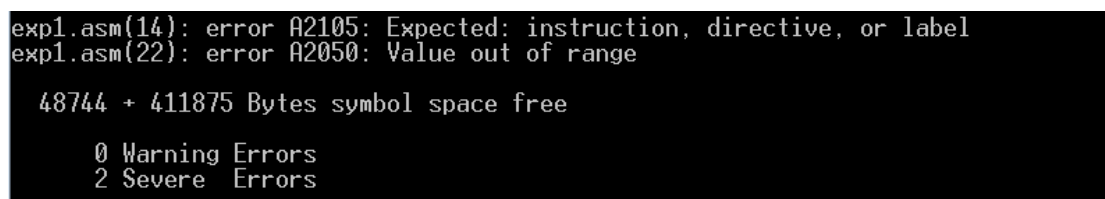
①Object filename [Filename.OBJ]：提示用户汇编将生成的二进制目标文件（OBJ 文件）。一般直接按回车表示同意生成该文件和系统给文件的命名（与源文件同名、扩展名为.OBJ，例中为 EXP1.OBJ）。用户也可以重新输入其它文件名。

②Source listing [NUL.LST]：提示是否要生成列表文件。一般直接回车，表示不需要。

③Cross-reference [NUL.CRF]：提示是否要生成交叉引用文件。一般直接回车，表示不需要。若源文件没有语法错误，则生成目标文件，如下图。注意图中的表示：0 个警告错误（0 warning Errors）和 0 个严重错误（0 Severe Errors）。



若程序有错，则显示提示和错误的行号，如下图。这时，汇编没有通过，需要重新编辑修改源文件，然后再重新汇编，直到没有错误，通过汇编。



3. 连接

(1) 经过汇编生成目标文件后，在 DOS 提示符下，键入“link exp1.obj”，回车，连接目标文件产生可执行的 EXE 文件。

(2) 在连接过程中，根据屏幕提示要回答 3 项内容：

①Run File [Filename.EXE]：提示用户将生成可执行文件（EXE 文件）。一般直接按回车表示同意生成该文件和系统给文件的命名（与目标文件同名、扩展名为.EXE，例中为 EXP1.EXE）。用户也可以重新输入其它文件名。

②List File [NUL.MAP]：提示是否要生成映像文件。一般直接回车，表示不需要。

③Libraries [.LIB]：提示是否要引入库文件。一般直接回车，表示不需要引入库文件。

回答完这 3 个问题，连接生成可执行文件。如下图：

```
E:\masm>link exp1

Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

Run File [EXP1.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment

E:\masm>
```

连接程序给出的“无堆栈段的警告（warning L4021: no stack segment）”不影响程序的运行。

4. 运行调试

(1) 在 DOS 提示符下，输入“Filename”或“Filename.EXE”（这里为 EXP1 或 EXP1.EXE），即可运行程序，如下图：

```
E:\masm>exp1.exe

E:\masm>_
```

因为本例的程序没有向屏幕直接输出，所以屏幕上未显示如何结果。但是屏幕可以显示程序已正常返回 DOS，说明接下来需要使用调试工具 DEBUG 来查看程序的运行结果。

(2) 在 DOS 提示符下，输入“debug Filename.exe”（本例为 debug exp1.exe），进入 DEBUG，如下图：

```
E:\masm>debug exp1.exe

_
```

(3) 在程序运行之前，先来看看程序加载时，PSP 和各寄存器的初始状态。输入“r”，可以看到各寄存器的值，如下图：

```
-r
AX=0000 BX=0000 CX=0060 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=1405 ES=1405 SS=1415 CS=1419 IP=0000 NV UP EI PL NZ NA PO NC
1419:0000 B81514 MOV AX,1415
```

图中，各寄存器的初值如下：

①CS 指向代码段，是代码段的段地址。IP 是代码段第一条指令的偏移地址。

②DS 和 ES 初始指向 PSP 的段地址，

③本例程序中没有设置堆栈段，如果有堆栈段，SS:SP 指向堆栈顶。

④BX 和 CX 是加载的程序的字节长度，BX 为高字节，CX 为低字节，这里为 60H。本程序的数据段长度 20H（按照分段的规定，一次最少分配 16 个字节单元，以使下一段可以从 xxx:0000H 的地方开始，所以尽管 26 个字母占 26 个字节单元，但是数据段实际分配了

32 个字节单元,即 20H),附件段长度也是 20H,在下面的反汇编时看出代码段长度也是 20H,这样数据段、附件段和代码段加起来程序长度就是 60H。

(4) 在提示符“-”下输入“u0”,进行反汇编,如下图:

```
-u0
1419:0000 B81514      MOV     AX,1415
1419:0003 8ED8          MOV     DS,AX
1419:0005 B81714      MOV     AX,1417
1419:0008 8EC0          MOV     ES,AX
1419:000A BE0000      MOV     SI,0000
1419:000D BF0000      MOV     DI,0000
1419:0010 B91A00      MOV     CX,001A
1419:0013 8A04          MOV     AL,[SI]
1419:0015 26           ES:
1419:0016 8805          MOV     [DI],AL
1419:0018 46           INC     SI
1419:0019 47           INC     DI
1419:001A E2F7          LOOP    0013
1419:001C B44C          MOV     AH,4C
1419:001E CD21          INT     21
-
```

从图中可以看到机器指令与汇编指令的对照显示。如果与源程序比较,这里看到的程序与源程序有相当的一致性又有差别。

代码段开始于 1419:0000,到 1419:001F 结束,共 32 个字节单元,即代码段长度为 20H。

在反汇编程序中,前面几句语句要看清楚,它告诉了你所要调试的程序的数据段以及附加段的段地址。(刚才加载程序后,初始情况下 DS 和 ES 为 1405H,是指向 PSP 的。)现在程序将把 1415H 赋给 DS,1417H 赋给 ES,说明该程序的数据段在 1415:0 处开始,附加段从 1417:0 处开始。使用显示内存命令 D 可以查看数据段和附加段的内容:

(5) 在提示符“-”下输入“d1415:0”,可以看到数据段中的 26 个大写字母,如下图:

```
-d1415:0
1415:0000 41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50  ABCDEFGHIJKLMNOP
1415:0010 51 52 53 54 55 56 57 58-59 5A 00 00 00 00 00 00  QRSTUVWXYZ.....
1415:0020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
1415:0030 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
1415:0040 B8 15 14 8E D8 B8 17 14-8E C0 BE 00 00 BF 00 00 00  .....
-
```

又在提示符“-”下输入“d1417:0”,可以看到附加段在程序没有执行之前是“空”的,如下图:

```
-d1417:0
1417:0000 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
1417:0010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
-
```

(6) 在提示符“-”下输入“g=0”,运行程序,显示“Program terminated normally”,程序正常结束,返回 DEBUG。如下图:

```
-g=0
Program terminated normally
-
```

接下来需要查看程序运行结果是否正常。

(7) 在提示符“-”下输入“d1417:0”,可以看到附加段中有了传送过来的 26 个大写字母,如下图:

```
-d1417:0
1417:0000 41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50  ABCDEFGHIJKLMNOP
1417:0010 51 52 53 54 55 56 57 58-59 5A 00 00 00 00 00 00  QRSTUVWXYZ.....
-
```

如果数据段和附加段一起看,也可以看到程序的运行结果,如下图:

```

-d1415:0
1415:0000 41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50 ABCDEFGHIJKLMNOP
1415:0010 51 52 53 54 55 56 57 58-59 5A 00 00 00 00 00 00 QRSTUVWXYZ.....
1415:0020 41 42 43 44 45 46 47 48-49 4A 4B 4C 4D 4E 4F 50 ABCDEFGHIJKLMNOP
1415:0030 51 52 53 54 55 56 57 58-59 5A 00 00 00 00 00 00 QRSTUVWXYZ.....
1415:0040 B8 15 14 8E D8 B8 17 14-8E C0 BE 00 00 BF 00 00 .....

```

这里应注意的是，1415:20 和 1417:0 是同一个地址。

(8) 如果需要单步执行调试程序，可以在提示符“-”下首先输入“t=0”，以后输入“t”，或者“t 值”，可以查看程序的每一步或几步运行之后的寄存器的结果。如下图：

```

-t=0
AX=1415 BX=0000 CX=001A DX=0000 SP=0000 BP=0000 SI=0001 DI=0000
DS=1415 ES=1417 SS=1415 CS=1419 IP=0003 NV UP EI PL NZ NA PO NC
1419:0003 8ED8 MOV DS,AX
-t
AX=1415 BX=0000 CX=001A DX=0000 SP=0000 BP=0000 SI=0001 DI=0000
DS=1415 ES=1417 SS=1415 CS=1419 IP=0005 NV UP EI PL NZ NA PO NC
1419:0005 B81714 MOV AX,1417
-t 3
AX=1417 BX=0000 CX=001A DX=0000 SP=0000 BP=0000 SI=0001 DI=0000
DS=1415 ES=1417 SS=1415 CS=1419 IP=0008 NV UP EI PL NZ NA PO NC
1419:0008 8EC0 MOV ES,AX
AX=1417 BX=0000 CX=001A DX=0000 SP=0000 BP=0000 SI=0001 DI=0000
DS=1415 ES=1417 SS=1415 CS=1419 IP=000A NV UP EI PL NZ NA PO NC
1419:000A BE0000 MOV SI,0000
AX=1417 BX=0000 CX=001A DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=1415 ES=1417 SS=1415 CS=1419 IP=000D NV UP EI PL NZ NA PO NC
1419:000D BF0000 MOV DI,0000
-

```

当运行到 INT 21H 时，输入“p”可以跳过进入 DOS 子程序，直接得到这个子程序的结果。如下图：

```

-t
AX=4C5A BX=0000 CX=0000 DX=0000 SP=0000 BP=0000 SI=001A DI=001A
DS=1415 ES=1417 SS=1415 CS=1419 IP=001E NV UP EI PL NZ NA PO NC
1419:001E CD21 INT 21
-p
Program terminated normally
-

```

P 命令和 t 命令的不同是：t 命令会跟踪进入子程序，由于 INT 21H 是 DOS 功能调用，那就会进入陌生的系统子程序运行。而 p 命令不会跟踪进入子程序，而是直接运行完该子程序。执行完该返回 DOS 的子程序后，程序正常结束，显示“Program terminated normally”，返回 DEBUG。

在该例中，也可以在运行到 LOOP 指令时，使用 p 命令把 LOOP 指令控制的循环一次执行完，如下图：

```

-t
AX=1441 BX=0000 CX=001A DX=0000 SP=0000 BP=0000 SI=0001 DI=0001
DS=1415 ES=1417 SS=1415 CS=1419 IP=001A NV UP EI PL NZ NA PO NC
1419:001A E2F7 LOOP 0013
-p
AX=145A BX=0000 CX=0000 DX=0000 SP=0000 BP=0000 SI=001A DI=001A
DS=1415 ES=1417 SS=1415 CS=1419 IP=001C NV UP EI PL NZ NA PO NC
1419:001C B44C MOV AH,4C

```

(9) 程序调试结束，输入“q”，退出 DEBUG，返回 DOS。

5. 如果通过调试程序没有得到正确的结果，并且发现错误所在，需要重新编辑修改源程序、再经过汇编、连接、运行，如果运行不能直接在屏幕上显示结果，需进入 **DEBUG** 运行查看结果。