第 3 章　80x86 寻址方式和指令系统

【本章导学】

本章主要介绍了 8086 CPU 的寻址方式和指令系统，它们是汇编语言程序设计的基础，是全书的重点。本章还介绍了调试工具 DEBUG，它不仅是汇编语言程序设计中常用的调试工具，也是理解和掌握指令的有力工具。学习本章时，要明确各种寻址方式的差别和特点，掌握有效地址和物理地址的计算方法；掌握各条指令的功能、对标志位的影响和在使用上的一些特殊限制；理解和掌握 DEBUG 中各种命令的功能和用法，并要善于通过 DEBUG 命令来进行学习。本章通过实例介绍了一些常用指令的用法，但由于初学者刚开始接触这些指令，与学习高级语言相比，可能会有一些困难和不适应，特别是对指令中寄存器的缺省使用方法及其用途，是初学者感到难以理解和记忆的内容，所以需要通过大量例题和习题来练习，并且在第 4 章中的程序设计中灵活运用这些内容之后，才能加以逐步理解和掌握。

3.1　指令格式及操作数类型

计算机通过执行程序来完成指定的任务，而程序是由完成一个完整任务的一系列有序指令组成的。指令是计算机能够识别和执行的操作命令，每条指令都明确规定了计算机必须完成的一套操作以及对哪一组操作数进行操作。每种计算机都有一套能反映计算机全部功能的指令，称为计算机的指令系统，不同系列的微处理器，有不同的指令系统。指令系统是计算机硬件和软件之间的桥梁，是汇编语言程序设计的基础。

3.1.1　指令格式

指令格式是指令的编码格式，其体现了指令系统的概貌。汇编语言指令的格式如下。

[标号:] [前缀] 操作码 [操作数] [;注释]

其由 4 部分组成，方括号中的内容为可选部分。

1. 标号

标号是一条指令语句的符号地址，在汇编源程序中，只有在需要转向一条指令语句时，才为该指令语句设置标号，以便在转移指令或子程序调用指令中直接引用这个标号。标号和后面的操作码之间必须用冒号“:”分隔。

2. 前缀及操作码

操作码也称指令码或助记符，用来指示计算机要执行的具体操作（如传送、运算、移位、跳转等），通常用一些意义相近的英文缩写（即助记符）来表示。操作码是所有指令中必不可少的部分，在一些特殊指令中，有时要在操作码前面加上前缀，以便和操作码配合实现某些附加操作。

3. 操作数

操作数是指令执行过程中参与指令操作的对象，它的表现形式比较复杂，可以是操作数本身，也可以是操作数地址或是地址的一部分，还可以是指向操作数地址的指针或其他有关操作数据的信息。根据指令的不同，在指令中可以不含操作数，即无操作数；也可以只含有一个操作数，即单操作数；或者是含有两个操作数，即双操作数。当是双操作数时，操作数中间必须用逗号“,”分隔，并且称逗号左边的操作数为目的操作数，逗号右边的操作数为源操作数。操作数与操作码之间必须用空格分隔。

4. 注释

注释是对有关指令及程序功能的标注性说明，以增加程序的可读性；用户可根据自己的需要来添加注释，但这并不影响程序的执行，但要注意注释与操作数之间必须用分号“;”分隔。

说明：读者要正确理解指令格式中方括号“[]”所表示的可选内容的含义。其中“注释”这个可选项完全取决于用户，是真正的可选；但对于“标号”或“操作数”则是出于指令的需要，对于需要的指令就一定要写，对于不需要的指令就不必写了。

3.1.2 操作数类型

操作数按其存放的地方，可分为立即操作数、寄存器操作数和存储器操作数三种类型。

1. 立即操作数

立即操作数是指具有固定数值的操作数（即常数），它具体可以是一个字节、字或双字。在指令书写中，立即操作数作为指令代码的一部分出现在指令中，通常作为源操作数使用，书写形式可以是二进制、十进制或十六进制，也可以是一个可求出确定值的表达式；存放时，该操作数跟随指令操作码一起被存放在指令区，故又称为指令区操作数。

2. 寄存器操作数

寄存器操作数事先存放在某寄存器（CPU 的通用寄存器、专用寄存器或段寄存器）中，在指令执行时只要知道寄存器名就可以寻找到操作数。在双操作数指令中，寄存器操作数既可以作为源操作数使用，又可以作为目的操作数使用。

3. 存储器操作数

存储器操作数是指放在存储单元中的数据，在指令中，只要知道存放操作数的存储单元的地址就可以寻找到该操作数。

存储器操作数可以是字节、字、双字等，分别存放在 1 个、连续 2 个或连续 4 个存储单元中。书写指令时为了方便，存储器操作数一般采用偏移地址（也称有效地址 EA）形式，段地址以隐含方式给出。

说明：存储器操作数有两种表示形式：①若用数字形式表示某存储单元的物理地址为 M，该地址中的内容为“N”，则有（M）＝N，即用圆括号将地址括起来表示该地址的内容，这种形式通常在文字描述中使用；②带方括号“[]”的操作数表示存储器操作数，括号中的内容为存储单元的偏移地址，这种形式通常在编程中使用。

3.2 寻址方式

在汇编指令中，可以直接给出操作数的值或者是存放操作数的地址，CPU 根据指令中的地址信息可求出存放操作数的有效地址，对存放在有效地址中的操作数进行操作。指令中关于如何求出操作数有效地址的方法称为操作数的寻址方式。下面介绍 80x86 常用的寻址方式。

3.2.1 立即寻址

立即寻址方式所提供的操作数直接包含在指令中，此操作数紧跟在操作码后面，与操作码一起被放在代码段区域中。立即寻址方式中的操作数只能是源操作数，主要用来给寄存器或存储单元赋初值。

对于 16 位微机系统，立即数可以是一个 8 位或 16 位数据。若是 16 位，则其低 8 位字节存放在相邻两个存储单元的低地址单元中，高 8 位字节存放在高地址单元中。例如：

```
MOV  AL,0EH              ; 将 8 位立即数 0EH 传送到 AL 寄存器中
MOV  AX,3102H            ; 将 16 位立即数 3102H 传送到 AX 寄存器中
```

这两条指令的指令码在内存中的存放格式及指令执行过程如图 3.1 所示。

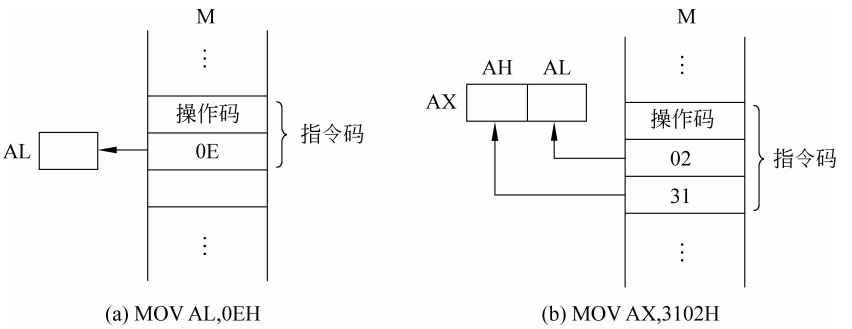

图 3.1 立即寻址示意图

对于 32 位微处理器，立即数还可以是 32 位操作数。例如：

```
MOV  EAX,23451023H       ; 将 32 位立即数 23451023H 传送到 EAX 寄存器中
```

3.2.2 寄存器寻址

寄存器寻址的操作数存放在 CPU 的某个寄存器中，在指令中写出指定的寄存器名即可。对于 8 位操作数，可用的寄存器是 AL、AH、BL、BH、CL、CH、DL 和 DH；对于 16 位的操作数，一般使用 AX、BX、CX、DX、SI、DI、SP、BP 和段寄存器；对于 32 位的操作数，可用的寄存器有 EAX、EBX、ECX、EDX、ESI、EDI、ESP 和 EBP。例如：

```
MOV  AX, CX              ; 将 CX 中的内容传送到 AX 中
MOV  EAX, ECX            ; 将 ECX 中的内容传送到 EAX 中
```

相对于其他寻址方式，因寄存器寻址的操作就在 CPU 内部进行，不需要访问存储器，因而其执行速度最快。

3.2.3 存储器寻址

在存储器寻址方式中，操作数存放在存储单元中，CPU 要访问存储器中的操作数，必须先计算出存放该操作数的存储单元的物理地址，然后才能对指定的存储单元进行读/写操作。

由于 80x86 系列 CPU 对存储器采用分段管理，若在指令中给出任何存储单元的物理地址则比较困难，所以在编程时都是采用逻辑地址，在指令中直接或间接地给出存放操作数的有效地址，并用方括号“[]”括起来，以达到访问操作数的目的，以下介绍几种具体的存储器寻址方式。

1. 直接寻址

在直接寻址方式中，指令中的操作数部分直接给出了操作数的有效地址(16 位的偏移地址)，且该地址与操作码一起被放在代码段中。通常，直接寻址方式的操作数放在存储器的数据段中，这是一种默认方式，当然也可以使用段超越。例如：

```
MOV  AX,[2000H]        ; 将偏移地址 2000H 与 DS 中的段地址所形成的物理地址字单元中的内
                       ; 容传送到 AX 寄存器中
MOV  AX,ES:[2000H]     ; 将偏移地址 2000H 与 ES 中的段地址所形成的物理地址字单元中的内
                       ; 容传送到 AX 寄存器中
```

存储器操作数本身并不能代表数据的类型，需要通过另一个寄存器操作数的类型或其他方式来确定。因此对于以上两例，由于目的操作数 AX 为字类型，所以作为存储器操作数的源操作数也应为字类型。假设 DS＝3000H，则在执行“MOV AX,[2000H]”指令后，AX 寄存器中的内容变为 3102H。指令的寻址及执行过程如图 3.2 所示。

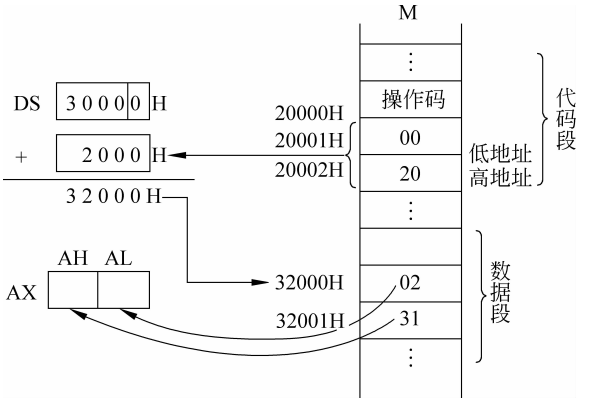

图 3.2 “MOV AX,[2000H]”指令的寻址及执行过程

在汇编语言编程中，常用符号来代替数值以表示操作数的偏移地址，上例中若用 BUF 代替地址 2000H，则“MOV AX,[2000H]”指令可写成：

```
MOV  AX,BUF
```

其中 BUF 为存放操作数的存储单元的符号地址，该符号须在汇编源程序的开始处予以定义，具体参见 4.2.2 节中的详细介绍。

2. 寄存器间接寻址

在寄存器间接寻址方式中，操作数的有效地址在指令指明的寄存器中，即寄存器的内容为操作数的偏移地址，而操作数存放在存储器中。

对于 16 位的 CPU，偏移地址存放在 SI、DI、BX、BP 中。如果指令中指定的寄存器是 BX、SI、DI，则操作数默认放在数据段中，操作数的物理地址是由数据段寄存器 DS 的内容乘以 16 后，加上 BX、SI 或 DI 中的偏移地址而形成的；如果指定的寄存器是 BP，则默认操作数在当前堆栈段中，操作数的物理地址由 SS 的内容乘以 16 后与 BP 中的偏移地址形成。

【例 3.1】 请分析指令“MOV AX,[BX]”的寻址情况。

执行“MOV AX,[BX]”指令，将以 DS 的内容为段地址，以 BX 中的内容为偏移地址的数据段中的相应字单元内容送入 AX。若 DS＝2000H，BX＝1000H，存储单元(21000H)＝3456H，则执行该指令后，AX＝3456H。

【例 3.2】 请分析指令“MOV AX,[BP]”的寻址情况。

若 SS＝3000H，BP＝1000H，则执行“MOV AX,[BP]”指令后，AX 的内容为 3320H。图 3.3 所示为该指令的寻址及执行结果。

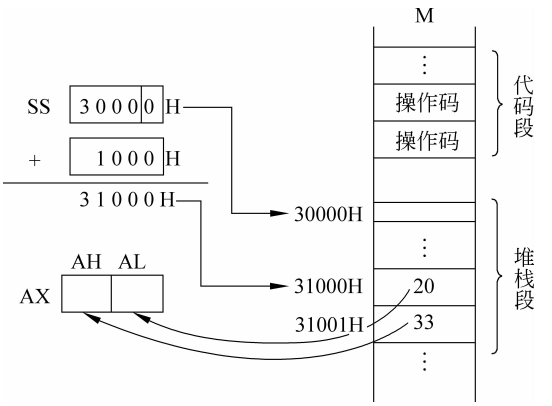

图 3.3 “MOV AX,[BP]”指令的寻址及执行结果

16 位的寄存器间接寻址允许在指令中指定段超越前缀来存取其他段中的操作数。例如：

```
MOV AX,ES:[BX]     ; 将以 ES 的内容为段地址，以 BX 的内容为偏移地址的附加段中的相应字单元
                   ; 内容送入 AX 寄存器中
```

3. 寄存器相对寻址

寄存器相对寻址又叫变址寻址，其特点是操作数的有效地址由一个基址(BP、BX)或变址(SI、DI)寄存器的内容加上指令中指定的 8 位或 16 位偏移量形成，偏移量用补码表示。即

$$EA=\begin{Bmatrix}BX\\BP\\SI\\DI\end{Bmatrix}+\begin{pmatrix}8\text{ 位}\\16\text{ 位}\end{pmatrix}\text{偏移量}$$

例如：

```
MOV AX,COUNT [BX]      ; 将段地址为 DS,有效地址为 BX+COUNT 所形成的字存储单元中的内容送
                       ; 至 AX 寄存器中
```

寄存器相对寻址方式允许段超越，在默认段超越前缀时，BX、SI、DI 默认的段寄存器为 DS，BP 默认的段寄存器为 SS。例如：

```
MOV SS: STR[ SI ] , AX     ;"SS"是段超越前缀，即 SS 为当前段寄存器
```

寄存器相对寻址方式的操作数在汇编语言编程中可以采用多种书写形式，以下是与“MOV AX,COUNT [BX]”指令等价的另外几种书写形式。

```
MOV AX,[BX+COUNT]
MOV AX,[COUNT+BX]
MOV AX,[ BX] COUNT
```

【例 3.3】 假设 DS＝3000H，BX＝1000H，COUNT＝4000H，物理地址(35000H)字单元中的内容为 1990H，则在执行“MOV AX,COUNT [BX]”指令后，AX＝1990H。图 3.4 所示为该指令的寻址及执行结果。

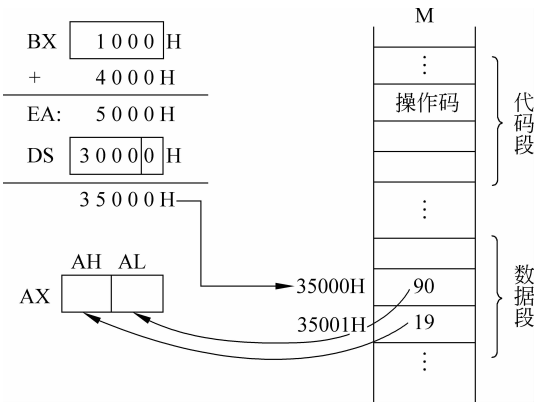

图 3.4 “MOV AX,COUNT[BX]”指令的寻址及执行结果

4. 基址变址寻址

在基址变址寻址方式中，操作数的有效地址由一个基址寄存器(BX、BP)的内容和一个变址寄存器(SI、DI)的内容之和形成，两个寄存器均由指令指出。即

$$EA=BX+\begin{pmatrix}SI\\DI\end{pmatrix}\text{或者是 }EA=BP+\begin{pmatrix}SI\\DI\end{pmatrix}$$

基址变址寻址的操作数段地址分配与寄存器相对寻址方式相同，也是使用默认段地址或使用段超越前缀来指定段地址；操作数的书写形式也有多种。

【例 3.4】 请分析指令“MOV AX,[BX][DI]”的执行情况。设 DS＝2000H，BX＝8000H，DI＝1000H，物理地址(29000H)的字单元内容为 5678H。

分析：对于指令“MOV AX,[BX][DI]”，源操作数的有效地址为 BX＋DI＝9000H，与段地址 DS＝2000H 所形成的物理地址为 29000H，执行该指令，将(29000H)字单元中的内容送给 AX。所以，AX＝5678H。

指令“MOV AX,[BX][DI]”也可以写成“MOV AX,[BX+DI]”或“MOV AX,[DI][BX]”或“MOV AX,[DI+BX]”的形式。

基址变址寻址方式适用于处理数组和表格,在编程时,可将首地址放在基址寄存器中,而用变址寄存器来访问数组或表格中的各个元素。

5. 基址变址相对寻址

在基址变址相对寻址方式中,操作数的有效地址 EA 是由一个基址寄存器的内容与一个变址寄存器的内容以及一个在指令中指定的 8 位或 16 位偏移量三者之和形成的,即

$$EA=\begin{pmatrix}BX\\BP\end{pmatrix}+\begin{pmatrix}SI\\DI\end{pmatrix}+\begin{pmatrix}8\text{ 位}\\16\text{ 位}\end{pmatrix}\text{偏移量}$$

与基址变址寻址方式相同,基址变址相对寻址方式的操作数在汇编语言编程中可采用多种等价形式;也允许段超越,在缺省段超越前缀时,BX 默认的段寄存器为 DS,BP 默认的段寄存器为 SS。例如以下指令:

```
MOV DX ,disp [BX][SI]
MOV DX ,disp [BX+SI]
MOV DX,[BX+SI+disp]
MOV DX,[BX+SI]disp
MOV DX,[BX] disp [SI]
```

这 5 种不同形式指令的寻址含义一致,都是将 DS 段寄存器与有效地址 BX+SI+disp 所形成的物理地址所对应的字存储单元中的内容传送到 DX 寄存器中。

【例 3.5】 设 DS=3000H,BX=2000H,SI=1000H,偏移量 disp=0250H,物理地址=DS×16+BX+SI+disp=30000H+2000H+1000H+0250H=33250H 的字单元内容为 3132H,则执行指令“MOV DX,disp [BX][SI]”后,DX 中的内容为 3132H。图 3.5 所示为该指令的寻址及执行结果。

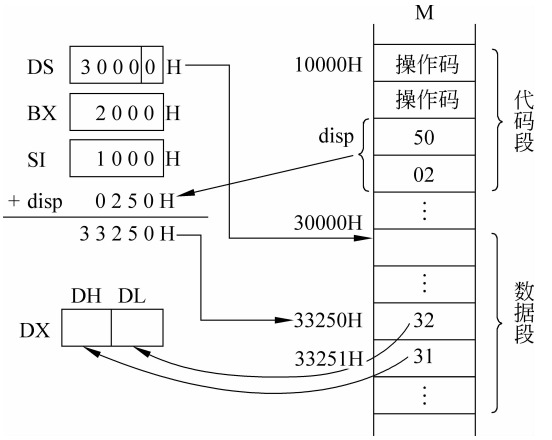

图 3.5 “MOV DX,disp[BX][SI]”指令的寻址及执行过程示意图

3.2.4 32 位存储器寻址

在 32 位微机系统中,除了支持 16 位微机系统的 7 种寻址方式外,还提供了一种更加灵

活、方便，但也更加复杂的存储器寻址方式，从而使内存地址的寻址范围得到了进一步扩大。

在用 16 位寄存器来访问存储单元时，只能使用基址寄存器(BX、BP)和变址寄存器(SI、DI)来作为有效地址的一部分，但在用 32 位寄存器寻址时，不存在这样的限制，所有 32 位寄存器都可以是有效地址的一个组成部分。

当用 32 位有效地址进行寻址时，存储地址的偏移量可分为三个部分。这三个部分分别是：一个 32 位基址寄存器，一个可乘以 1、2、4 或 8 的 32 位变址寄存器，一个 8/32 位的偏移量，并且这三个部分还可以进行任意组合，省去其中之一或之二。其中的 32 位基址寄存器可以是 EAX、EBX、ECX、EDX、ESI、EDI、EBP 和 ESP；32 位变址寄存器可以是 EAX、EBX、ECX、EDX、ESI、EDI 和 EBP。32 位寻址的有效地址计算公式可以归纳为

$$EA=\begin{pmatrix}无\\EAX\\EBX\\ECX\\EDX\\ESI\\EDI\\EBP\\ESP\end{pmatrix}+\begin{pmatrix}无\\EAX\\EBX\\ECX\\EDX\\ESI\\EDI\\EBP\end{pmatrix}\times\begin{pmatrix}1\\2\\4\\8\end{pmatrix}+\begin{pmatrix}无\\8\text{ 位}\\32\text{ 位}\end{pmatrix}$$

基址寄存器　变址寄存器　比例因子　偏移量

由于 32 位存储器寻址方式能使用所有的通用寄存器，所以，和该有效地址相组合的段寄存器也就有了新的规定，具体如下。

(1) 有效地址中寄存器的书写顺序决定了该寄存器是基址寄存器，还是变址寄存器。例如：

[EBX+EBP]中的 EBX 是基址寄存器，EBP 是变址寄存器，而[EBP+EBX]中的 EBP 是基址寄存器，EBX 是变址寄存器。

(2) 默认段寄存器的选用取决于基址寄存器，当基址寄存器是 EBP 或 ESP 时，默认的段寄存器是 SS，否则，默认的段寄存器是 DS。

(3) 在指令中，如果使用段前缀的方式，则表示显式段寄存器优先。

以下列举的是 32 位存储器寻址指令及其存储器操作数所引用的段寄存器例子。

指令举例	访问存储单元所用的段寄存器

```
MOV  AX, [123456H]                    ; 默认段寄存器 DS
MOV  EAX, [EBX+EBP]                   ; 默认段寄存器 DS
MOV  EBX, [EBP+EBX]                   ; 默认段寄存器 SS
MOV  EBX, [EAX+100H]                  ; 默认段寄存器 DS
MOV  EDX, ES:[EAX * 4+200H]           ; 显式段寄存器 ES
MOV  [ESP+EDX * 2], AX                ; 默认段寄存器 SS
MOV  EBX, GS:[EAX+ EDX * 2+300H]      ; 显式段寄存器 GS
```

3.3 调试工具 DEBUG

DEBUG 是专为汇编语言提供的一种调试工具，用户通过其所提供的命令可以直接查看和修改存储单元及寄存器的内容；可以装入、存储程序并通过单步执行、设置断点等方式方便地运行程序，为汇编程序员提供了非常有效的程序调试手段。不仅如此，对汇编语言初学者来说，DEBUG 也是练习使用汇编指令的一种有效工具，初学者可以直接在 DEBUG 环境下执行汇编指令。所以，本节提前介绍 DEBUG 工具，以使读者能更有效地学习后面的指令系统。

3.3.1 DEBUG 的启动

在 DOS 和 Windows 操作系统中，DEBUG 都以外部命令文件（DEBUG. EXE）的形式提供给用户，因此，启动 DEBUG 常可用以下两种方法。

1）采用鼠标方式启动

方法：在本机上找到 DEBUG. EXE 文件（如果没有，须先复制），双击其图标即可，如图 3.6 所示。

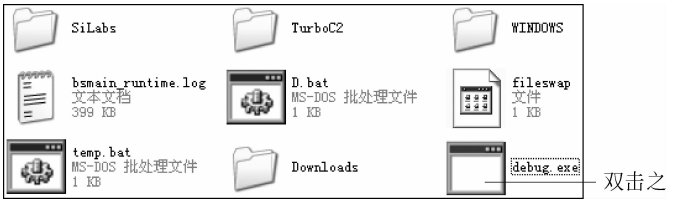

图 3.6 用鼠标方式启动 DEBUG

2）采用 DOS 命令方式启动

采用 DOS 命令方式启动的操作步骤如下。

（1）进入 DOS 环境。在 Windows 桌面下单击“开始”菜单，在弹出的菜单中单击“运行”命令，再在“运行”窗口中，输入 cmd（或 command）命令，如图 3.7 所示 。

（2）进入到 DEBUG. EXE 所在的文件夹，输入 DEBUG 命令并按 Enter 键。

启动 DEBUG 后的提示符为小短线“—”，如图 3.8 所示。

3.3.2 DEBUG 的常用命令

1. DEBUG 命令格式

DEBUG 的命令格式为

命令字母［参数］

每条命令以单个字母的命令符开头，后面是命令的操作参数，所有命令都遵从以下规定。

（1）命令及参数的输入不分大小写。

（2）命令中使用的参数均为十六进制数，输入时不需加“H”。

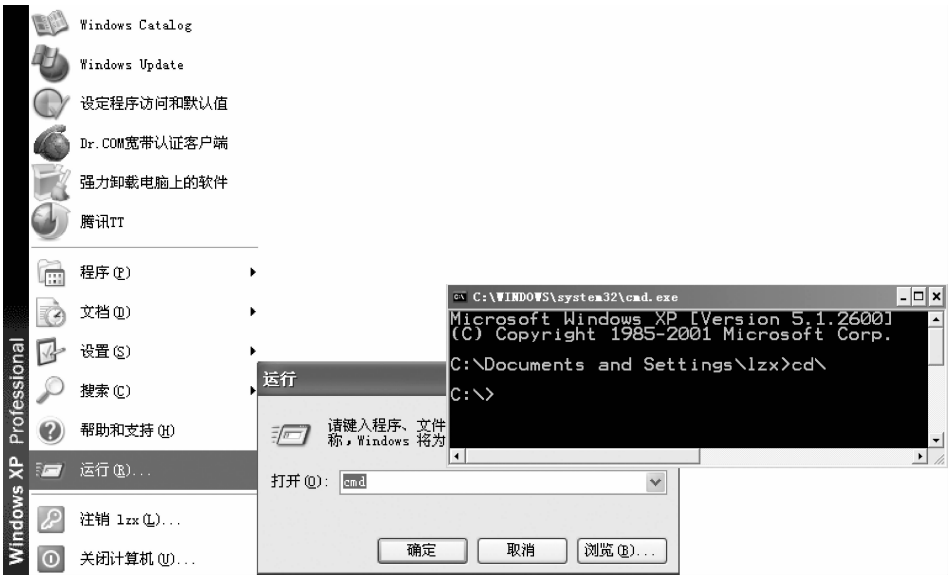

图 3.7　进入 DOS 命令方式

图 3.8　DEBUG 环境

(3) 命令和参数之间可用空格分隔(也可以不用空格),但参数和参数之间,须用空格或逗号分隔,具体是空格还是逗号,与参数有关。

(4) 命令的结束符是回车键 Enter,Ctrl+Break 键可中止命令的执行。

(5) 命令执行时,如果不符合 DEBUG 规则,则会提示"error" 错误信息。

(6) 参数既可以表示地址也可以表示地址范围。当为地址时只能用逻辑地址形式表示,具体有以下三种表示形式。

① 完整的逻辑地址(段地址:偏移地址)形式。例如:"D 0400:2500","D DS:04","D CS:100"。

② 只写出偏移地址的形式。采用这种形式时 DEBUG 认为输入的是偏移地址,段地址采用默认的段寄存器,不同的命令采用的默认段寄存器不同。若是针对汇编指令所操作的命令,则 DEBUG 默认的段寄存器为 CS;若是针对数据所操作的命令,则 DEBUG 默认的段寄存器为 DS。例如:

```
—D 2505                    ; 2505H 为偏移地址,默认段值在 DS 中
—A 100                     ; 0100H 为偏移地址,默认段值在 CS 中
```

③ 既不写段地址也不写偏移地址的形式。这种形式的段地址采用默认的段寄存器,偏移地址采用当前值。例如:

```
—A                         ; 默认的是当前的 CS:IP 值
```