第 3 章　8086/8088 指令系统与汇编语言程序设计

【本章的内容提要】

本章主要的内容是 8086 的指令系统，8086/8088 宏汇编语言和程序设计。具体包括以下几个部分：

(1) 操作数的寻址方式；

(2) 指令的形式、功能和应用；

(3) 8086/8088 汇编语言源程序的结构；

(4) 常量、变量和表达式以及常用伪指令；

(5) 常用的 DOS 系统功能调用。

【本章的学习重点】

本章应重点掌握和理解的知识：

(1) 重点掌握存储器操作数的寻址方式；

(2) 重点掌握指令的形式、功能，并学会一个功能用多种不同的指令组合完成；

(3) 掌握经典例题，并能在理解的基础上，经过适当的组合完成其延伸；

(4) 掌握 8086/8088 汇编语言源程序的基本结构；

(5) 借助汇编语言程序设计，培养逻辑思维的能力。

3.1　8086/8088CPU 的寄存器组与寻址方式

汇编语言与计算机硬件密切相关。因此，了解计算机系统的基本结构及相关知识是学习汇编语言程序设计的基础。本节从汇编语言程序设计的角度，介绍 8086/8088CPU 的寄存器组以及 8086/8088 的寻址方式。

3.1.1　8086/8088CPU 的寄存器组

8086/8088CPU 的寄存器组包括 8 位和 16 位寄存器，这些寄存器可分为三类：通用寄存器(General Purpose Register)、段寄存器(Segment Register)和专用寄存器(Special Purpose Register)，如图 3-1 所示。

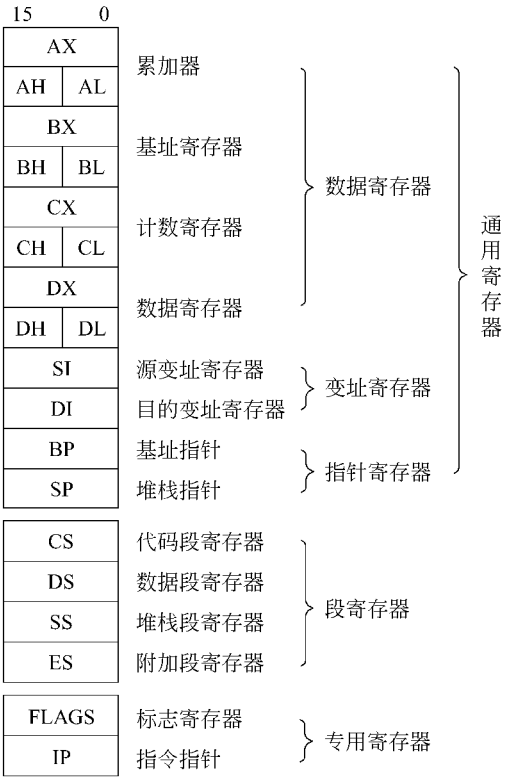

图 3-1 8086/8088CPU 的寄存器组

1. 通用寄存器

8086/8088CPU 的通用寄存器包括 8 个 8 位的通用寄存器：AL、AH、BL、BH、CL、CH、DL、DH。8 个 16 位的通用寄存器：AX、BX、CX、DX、SI、DI、BP、SP。

其中，AL 与 AH、BL 与 BH、CL 与 CH、DL 与 DH 分别对应于 AX、BX、CX 和 DX 的低 8 位与高 8 位。存取 8 位寄存器时，相应 16 位寄存器的其他位不受影响。例如，AX 是一个 16 位通用寄存器，低 8 位可以用 AL 访问，高 8 位可以用 AH 访问，修改 AL 会影响 AX(即 AX 的低 8 位有变化)，但不影响 AH(即 AX 的高 8 位无变化)。

通用寄存器可以作为指令的操作数，用于存储经常访问的数据。下面对这些寄存器作一简要说明。

- 累加器 AX(Accumulator)

  AX 与 AL 分别被称为 16 位和 8 位累加器。通常，以 AX 或 AL 作为指令的操作数，可以使指令的机器代码更短、执行速度更快。

- 基址寄存器 BX(Base)

  BX 常用来表示内存地址。

- 计数寄存器 CX(Count)

  许多指令以 CX 或 CL 作为计数器。例如，移位指令以 CL 表示移位次数，循环指令以 CX 作为隐含的循环次数。

- 数据寄存器 DX(Data)
  在乘法与除法运算中,用 DX 与 AX 的组合来存放一个双字数。此外,在 I/O 指令中,DX 也用来表示端口地址。
- 源变址寄存器 SI(Source Index)
  在串操作指令中,以 SI 表示源数据串的地址。
- 目的变址寄存器 DI(Destination Index)
  在串操作指令中,以 DI 表示目的数据串的地址。
- 基址指针 BP(Base Pointer)
  BP 常用来表示内存地址,默认情况下指向堆栈段中的存储单元。
- 堆栈指针 SP(Stack Pointer)
  SP 用来指示堆栈段中的栈顶地址。在设计程序时,一般不选择 SP 作为算术或逻辑运算的操作数。

2. 段寄存器

8086/8088CPU 包括 4 个 16 位的段寄存器：CS、DS、ES 和 SS,分别被称为代码段(Code Segment)寄存器、数据段(Data Segment)寄存器、附加段(Extra Segment)寄存器和堆栈段(Stack Segment)寄存器。

3. 专用寄存器

8086/8088 有两个 16 位的专用寄存器：指令指针 IP(Instruction Pointer)和标志寄存器 FLAGS(Flags Register)。这两个寄存器不能作为指令的操作数。

在 8086/8088 程序的执行过程中,CS 与 IP 共同指出要执行的下一条指令的地址,FLAGS 用来反映指令的执行结果或控制指令的执行方式。

3.1.2　8086/8088CPU 的寻址方式

8086/8088 指令由操作码和操作数两部分构成。

为了表示指令的操作数,8086/8088CPU 提供了多种形式,以指出操作数或操作数的地址,这就是寻址方式(Addressing Mode)。8086/8088 指令的操作数主要包括以下 3 类。

(1) 立即数。

操作数以常量的形式出现在指令中,称为立即数。立即数只能作为指令的源操作数。

(2) 寄存器操作数。

指令要操作的数据存放在寄存器中,在指令中给出寄存器名。

(3) 内存操作数。

指令要操作的数据存放在内存单元中,在指令中给出内存地址。

因此,8086/8088 的寻址方式也分为 3 类：立即寻址(Immediate Addressing)、寄存器寻址(Register Addressing)和内存寻址。内存寻址又分为：直接寻址、寄存器间接寻址、寄存器相对寻址、基址变址寻址和相对基址变址寻址。

1. 立即寻址

所谓立即寻址,是指以 8 位或 16 位立即数作为指令的操作数,并直接出现在指令中,

紧跟在操作码之后，作为指令的一部分放在代码段中。

【例 3-1】 立即寻址举例。

```
MOV        AL, 80H                ; 源操作数为立即数 80H
MOV        DX, 0B9AH              ; 源操作数为立即数 0B9A H
```

立即寻址只能用于源操作数，不能用于目的操作数，且只能为整数。

2. 寄存器寻址

所谓寄存器寻址，是指以寄存器的值作为操作数，寄存器名由指令指出。指令中可使用的寄存器包括 8 位或 16 位通用寄存器以及段寄存器。

【例 3-2】 寄存器寻址举例。

```
DEC  CL                           ; 将 CL 的内容减 1
MOV  DX,AX                        ; 将 AX 中的值送入 DX 中
```

对于寄存器寻址方式，由于操作数在 CPU 内部的寄存器中，不需要访问内存，因而执行速度较快。另外，若使用累加器 AX 或 AL 作为目的操作数，指令的机器代码通常会更短、执行速度更快。

3. 内存寻址

由于 CPU 中的寄存器数目有限，程序不可能把所有参加运算的数据都放在寄存器中，同时也不能满足程序员组织大量数据的需要。因此，在大多数情况下，操作数一般存放在内存中。

在内存寻址方式中，操作数是某个内存单元的值，在指令中给出内存单元的偏移地址（也称为有效地址 EA），段地址通常隐含在某个段寄存器中。

（1）直接寻址

所谓直接寻址，是指操作数的有效地址 EA，直接包含在指令中，放在操作码之后。

【例 3-3】 直接寻址举例。

```
MOV  AX,[2000H]
```

设(DS)＝3000H，则物理地址＝30000H＋2000H＝32000H。内存的存储情况如图 3-2 所示，则(AX)＝5068H。

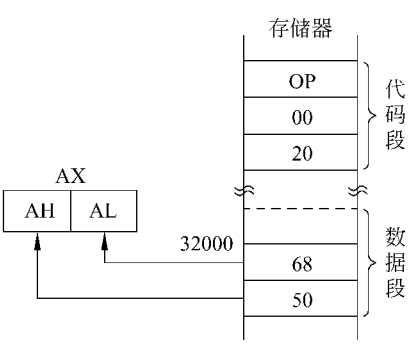

图 3-2 直接寻址示意图

在例 3-3 中，没有用前缀指明操作数在哪一段，则默认在数据段中。若要将附加段的 2000H 及 2001H 两单元内容取出送给 AX，则需指明。例如：

```
MOV AX,ES:[2000H]
```

在汇编语言中，可用符号地址代替数值地址。例如：

```
MOV AX,[BUFFER1]
```

其中 BUFFER1 为存放操作数的符号地址。当使

用符号地址时，中括号可以省略，如：

```
MOV AX,BUFFER1
```

与上式等价。

(2) 寄存器间接寻址

利用这种寻址方式时，操作数在存储器中，而操作数有效地址由 BX、BP、SI 和 DI 中的一个指出，若未用前缀指明，默认关系为：对 BX、SI、DI 默认在 DS 段中，对 BP 默认在 SS 段中。

【例 3-4】 寄存器间接寻址举例。

```
MOV  AX,[SI]
```

设(DS)＝5000H，(SI)＝3000H，该指令的作用是将 53000H 和 53001H 两存储单元中的内容传送到 AX。

(3) 寄存器相对寻址

寄存器相对寻址又称为变址寻址，利用寄存器间接寻址时，允许在指令中指定一个 8 位或 16 位的位移量，这样有效地址 EA 的构成为：

$$EA=\begin{Bmatrix}[BX]\\[BP]\\[SI]\\[DI]\end{Bmatrix}+\begin{Bmatrix}8\text{ 位位移量}\\16\text{ 位位移量}\end{Bmatrix}$$

【例 3-5】 寄存器相对寻址举例。

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

设(SS)＝6000H，(BP)＝2000H，COUNT＝1040H，物理地址为：63040H。内存的存储情况如图 3-3 所示，则(AX)＝6658H。

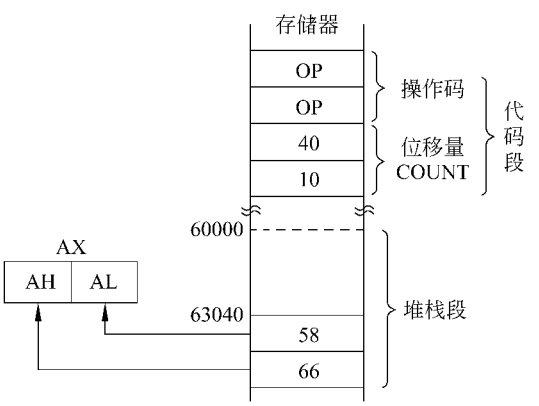

图 3-3　寄存器相对寻址示意图

(4) 基址加变址寻址

操作数的有效地址是一个基址寄存器内容加上一个变址寄存器内容，若未用前缀指

明，默认关系为：对 BX、SI、DI 默认在 DS 段中，对 BP 默认在 SS 段中。

【例 3-6】 基址加变址寻址举例。

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

设(DS)＝5500H，(BX)＝0256H，(SI)＝3094H，物理地址为：582EAH。内存的存储情况如图 3-4 所示，则(AX)＝2A08H。

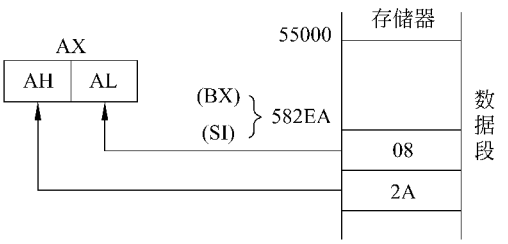

图 3-4 基址加变址寻址示意图

注意：基址寄存器只能是 BX 或 BP，变址寄存器只能是 SI 或 DI。有效地址 EA 为：

EA＝基址寄存器＋变址寄存器

(5) 相对的基址加变址寻址

在基址加变址寻址的基础上加上一个 8 位或 16 位的位移量，就构成了相对的基址加变址寻址。若未用前缀指明，默认关系为：对 BX、SI、DI 默认在 DS 段中，对 BP 默认在 SS 段中。

【例 3-7】 相对的基址加变址寻址举例。

```
MOV AX,[MASK+BX+DI]
```

设(DS)＝5000H，(BX)＝1256H，(DI)＝1500H，MASK＝1F34H，物理地址为：5468AH。内存的存储情况如图 3-5 所示，则(AX)＝2A08H。

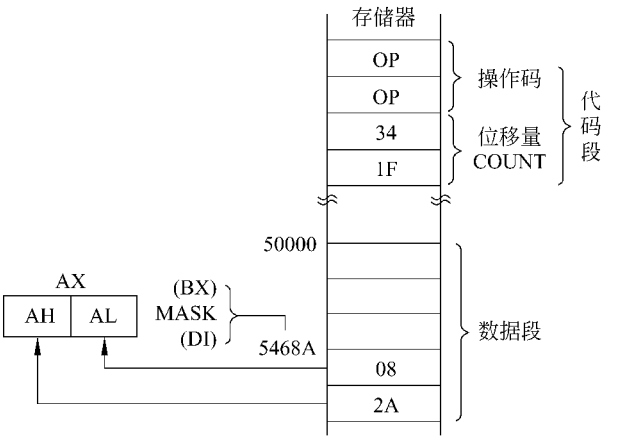

图 3-5 相对的基址加变址寻址示意图

下面对8086/8088寻址方式做几点说明。

① 在Microsoft宏汇编MASM(Microsoft Macro Assembler)中,内存操作数可以采用多种书写形式。例如:

```
MOV    AX, [BX+DI]              ; 等价于 MOV  AX, [BX][DI]
MOV    AX, BUF[BP+DI]           ; 等价于 MOV  AX, BUF[BP][DI]
MOV    AX, [BUF+BP+DI]          ; 等价于 MOV  AX, [BUF][BP][DI]
```

② 在源程序被汇编后,凡是出现在指令中的变量名,均由具体的偏移地址所取代。例如,设BUF是字类型的变量,在段内的偏移地址为100H,则指令:

```
MOV  AX, BUF
```

等价于:

```
MOV  AX, [100H]
```

这种数值地址通常只在调试程序(如DEBUG)中使用。在源程序中,若使用数值地址,通常要给出段超越前缀,例如:

```
MOV  AX, DS:[100H]
```

③ 在寄存器相对寻址与相对基址变址寻址方式中,位移量可以是符号常量或变量,汇编后为一个常数,若是变量,则取其偏移地址。例如:

```
MOV  AX, COUNT[SI]            ; 设 COUNT 是已定义的符号常量,相当于常数
                              ; 源操作数有效地址 EA=SI+COUNT
MOV  Al, BUF[BX][DI]          ; 设 BUF 是已定义的字节类型变量
                              ; 源操作数有效地址 EA=BX+DI+BUF 的偏移地址
```

④ 在指令的操作数部分,凡是能用变量的地方也可使用下列形式:

```
变量名+数值表达式
变量名-数值表达式
```

例如,设BUF是已定义的字类型变量,则:

```
MOV  AX, BUF+8                ; 源操作数有效地址=BUF 的偏移地址+8
MOV  AX, BUF[BX+8]            ; 源操作数有效地址=BUF 的偏移地址+BX+8
```

3.2　8086/8088的指令系统

掌握指令系统是汇编语言程序设计的基础。本节介绍8086/8088CPU的指令系统,主要包括指令的格式、功能以及对CF、OF、SF和ZF这4个标志位的影响,并结合具体实例,讲述指令的使用方法。其中,中断指令将在后续章节介绍。

3.2.1　8086/8088的指令格式

8086/8088指令由操作码和操作数两部分构成。操作码指出指令要执行哪种操作,

如：加、减、比较等。操作数指出指令要操作的数据对象。8086/8088 指令操作数部分分为 0 个、1 个或 2 个操作数等。

8086/8088 汇编语言指令的一般形式为：

```
[标号:]  指令助记符  操作数  [;注释]
```

说明如下。

- 标号代表该行指令的起始地址。
- 指令助记符表示指令的名称，它是指令功能的英文缩写。
- 若指令中包含多个操作数，则操作数之间用逗号分隔。通常，将存放操作结果的操作数称为目的操作数，一般作为指令的第 1 个操作数；将指令执行后保持不变的操作数称为源操作数，一般作为指令的第 2 个操作数。
- 注释以分号开始，用来说明程序功能。可以单独占一行，也可以放在一条指令的后面。
- 用中括号括起来的部分为选择项。

例如：最常用的数据传送指令 MOV 的格式如下。

```
MOV    dest, src
```

此条指令的作用是将源操作数的值传送给目的操作数。一般常用缩写 src 表示源操作数，dest 表示目的操作数。

8086/8088 指令可分成如下 6 类：

- 数据传送指令；
- 算术运算指令；
- 逻辑运算指令与移位指令；
- 串操作指令；
- 控制转移指令；
- 处理器控制指令。

3.2.2 数据传送指令

数据传送指令是汇编语言中最简单、最常用的一类指令，主要包括 MOV、XCHG、PUSH、POP 与 LEA 等。在这一类指令中，除了 POPF 和 SAHF 外，均不影响标志位。

数据传送指令可分为如下 4 类：

- 通用数据传送指令；
- 地址传送指令；
- 标志传送指令；
- 累加器专用(IO 端口数据)传送指令。

1. 通用数据传送指令 MOV(Move)

一般形式：

```
MOV  dest, src
```

功能：将源操作数 src 的值送给目的操作数 dest，src 的值不变。

说明：MOV 指令对标志位无影响。传送的是字节还是字取决于指令中涉及的寄存器是 8 位还是 16 位。具体来说可实现以下功能。

(1) MOV　mem/reg1，mem/reg2(寄存器之间，寄存器和存储单元之间数据传送)

【例 3-8】 寄存器和存储单元之间数据传送。

```
MOV  CL,AL                    ; 寄存器之间 8 位的数据传送
MOV  AX,DX                    ; 寄存器之间 16 位的数据传送
MOV  [SI],CX                  ; 寄存器和存储单元之间 16 位的数据传送
MOV  DL,[BX+COUNT]            ; 寄存器和存储单元之间 8 位的数据传送
```

(2) MOV　reg，data(立即数送寄存器)

【例 3-9】 将立即数传送给寄存器。

```
MOV  CL,08H                   ; 8 位的立即数传送寄存器
MOV  DX,234H                  ; 16 位的立即数传送寄存器
```

(3) MOV　mem，data(立即数送存储单元)

【例 3-10】 将立即数传送给存储单元。

```
MOV  BYTE PTR [BX],0FH        ; 8 位的立即数传送存储单元
MOV  WORD PTR [DI],234H       ; 16 位的立即数传送存储单元
```

(4) MOV　ac/mem，mem/ac(存储单元和累加器之间数据传送)

【例 3-11】 存储单元和累加器之间数据传送。

```
MOV   [BX],AL                 ; 存储单元和累加器之间 8 位的数据传送
MOV  AX,[DI]                  ; 存储单元和累加器之间 16 位的数据传送
```

(5) MOV　segreg，mem/reg(段寄存器和存储单元/寄存器之间数据传送)

【例 3-12】 段寄存器和存储单元/寄存器之间数据传送。

```
MOV  DS,AX                    ; 累加器传送段寄存器
MOV  [DI],DS                  ; 段寄存器传送存储单元
```

注意：使用 MOV 指令时，两个操作数不允许做如下搭配。

① 两个操作数同时为存储单元，即：

```
MOV  mem, mem
```

② 两个操作数同时为段寄存器，即：

```
MOV  seg, seg
```

③ 把立即数直接传送给段寄存器，即：

```
MOV  seg,立即数
```

④ CS 不能作为目的操作数。

⑤ 在两个操作数中，必须至少有其中一个的类型是确定的，而且要类型匹配，即同时是字节类型或字类型。

若不能确定操作数的类型，则需要用 BYTE PTR 或 WORD PTR 明确指出是字节或字类型。关于 PTR 的详细介绍见 3.5.3 节。

【例 3-13】 错误的 MOV 指令。设 BUF 是已定义的字节类型的变量。

```
MOV    DX, CL                    ; 类型不匹配
MOV    CS, DX                    ; CS 不能作为目的操作数
MOV    ES, 3100H                 ; 不允许立即数送段寄存器
MOV    [BX][SI], [DI]            ; 不允许内存操作数之间传送
MOV    DS, CS                    ; 不允许段寄存器之间传送
MOV    DX, BUF                   ; 类型不匹配
MOV    [BX][DI], 2               ; 无法确定操作数的类型
```

思考题：上述几种不能传送的解决办法是什么？

2. 交换指令 XCHG(Exchange)

一般形式：

```
XCHG  reg/mem, reg/mem
```

功能：将源操作数(字或字节)与目的操作数交换。

说明：XCHG 指令对标志位无影响。要求两个操作数的类型必须匹配，且两操作数中必须有一个在寄存器中，不允许使用段寄存器，不允许使用立即数。

【例 3-14】 正确地使用 XCHG 指令示例。

```
XCHG  AL, AH
XCHG  BX, DX
XCHG   DL, [BX][SI]
XCHG  [BP][SI], DX
```

【例 3-15】 错误地使用 XCHG 指令示例。

```
XCHG    DS, AX                  ; XCHG 指令的操作数不能是段寄存器
XCHG    AX, DL                  ; 类型不匹配
XCHG    [SI], [DI]              ; 不允许两个内存操作数之间交换
```

3. 地址传送指令

地址传送指令包括 LEA(Load Effective Address，装入有效地址)、LDS 与 LES。其中，LDS 和 LES 指令可用来给 1 个段寄存器和 1 个 16 位通用寄存器同时赋值。本书只介绍一条 LEA 指令。

一般形式：

```
LEA  reg,mem
```

功能：将指定存储单元的 16 位偏址送指定寄存器。