第8章　地址寻址

S7-300、400 系列 PLC 地址寻址分为直接地址寻址和间接地址寻址两大类，直接地址寻址通过指令直接对地址进行访问，地址通常是一个常数，不可以改变；间接地址寻址时地址存储于地址指针中，地址是一个变量，程序执行时才能确定实际的地址。在各自寻址的方式下又进行细的划分，不同的寻址方式如图 8-1 所示。

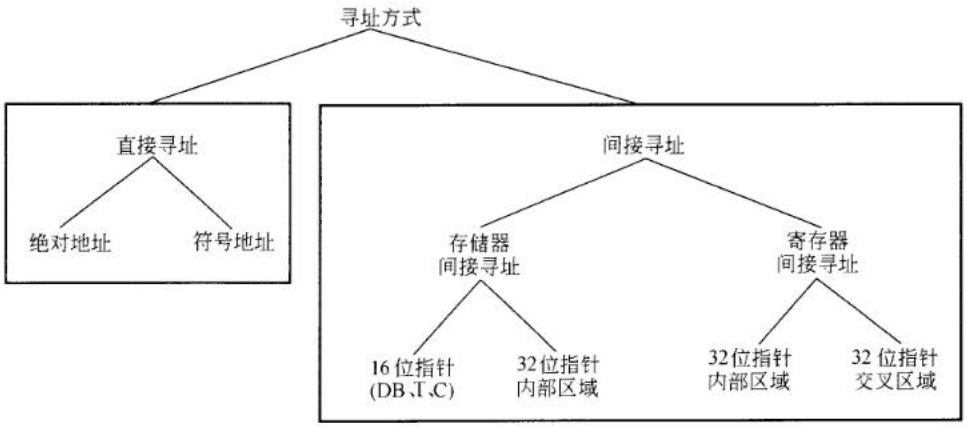


图 8-1　寻址方式

8.1　绝对地址寻址与符号地址寻址

绝对地址寻址是指存储单元地址可以直接包含在指令中，指令通过地址标识符号直接对变量进行读写访问，在 STEP7 中，I/O 信号、标志位存储区、定时器、计数器、程序块、数据块等都可以通过绝对地址进行访问，地址区及直接访问举例见表 8-1 所示。

表 8-1　直接地址寻址类型

地址区符号及访问长度	说　明	举　例
I/IB/IW/ID	过程映像区输入	I1.0、IB2、IW4、ID12
Q/QB/QW/QD	过程映像区输出	Q3.2、QB12、QW20、QD40
PIB/PIW/PID	外设输入（或立即读）	PIB256、PIW300、PID400
PQB/PQW/PQD	外设输出（或立即写）	PQB256、PQW288、PQD300
M/MB/MW/MD	标志位存储区	M4.0、MB3、MW12、MD42
L/LB/LW/LD	区域数据	L2.2、LB1、LW20、LW42
T	定时器	T1
C	计数器	C1

（续）

地址区符号及访问长度	说　明	举　例
FC/FB/SFC/SFB	程序块	FC1、SFC67
DBX/DBB/DBW/DBD	数据块（使用 OPN DB）*	DBX12.0、DBB20、DBW40、DBD100
DIX/DIB/DIW/DID	数据块（使用 OPN DI）*	DIX12.0、DIB20、DIW40、DID100

* DB 块的访问也可以直接带有 DB 号，例如 DB1.DBX20.0。

I、Q、M、L 有位寻址、字节寻址、字寻址、双字寻址几种方式。PI 类型和 PQ 类型只有字节寻址、字寻址、双字寻址几种方式。

符号地址寻址是指为每个绝对地址分配一个符号名称，便于识别和记忆，增强用户程序的可读性，便于设备的调试，例如用符号“Motor_Start”来代替绝对地址 I1.0。STEP7 中符号寻址分为全局符号（在符号表中定义）和区域符号（在程序块的通信接口及临时变量、静态变量中定义），全局符号在整个用户程序范围内有效，局部符号只能在一个程序块内部使用。

8.2 间接寻址

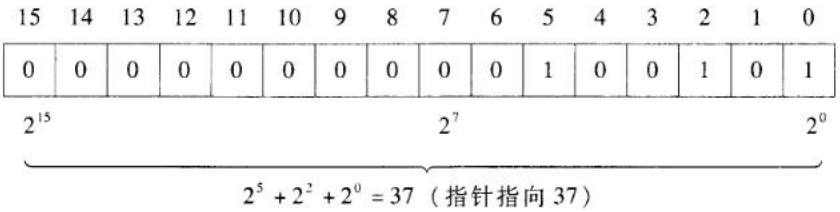
间接寻址分为存储区间接寻址和寄存器间接寻址，前者寻址的地址指针存储于存储器中，如 M、L 区等，后者存储于 CPU 的地址寄存器 AR1 或 AR2 中。

8.2.1 存储器间接寻址

存储于存储器中的地址指针分为 16 位地址指针和 32 位地址指针，允许存储地址指针的存储器为 M（标志位）、L（区域数据）及数据块（DB 或 DI），下面分别介绍两种地址指针的使用。

1. 16 位地址指针

16 位地址指针用于定时器、计数器、程序块（DB、FC、FB）的寻址，16 位指针被看作是一个无符号整数（取值范围 0 ~ 65535），它指向定时器（T）、计数器（C）、数据块（DB、DI）或程序块（FB、FC）的号，16 位指针的格式如下：



所有的定时器、计数器及程序块都可以使用间接寻址访问，访问时需要使用 T、C、DB、DI、FB、FC 等区域标识符，寻址的指针存储于一个 16 位的字中，地址寻址表示格式为：区域标识符［16 位地址指针］，例如打开一个 DB 块表示为：

使用16位地址指针访问一个定时器和计数器的示例程序如下：

```
L       11                  //将11传送到累加器1中
T       MW      20          //将累加器1中的数值传送到MW20中
A       I       2.1         //如果I2.1为1，将预置值10s装载到T11中
L       S5T#10S
SE      T [MW20]
L       MW      20
L       1
+I
T       MW      22          //MW20再加1
A       I       2.2         //如果I2.2为1，C12向上计数一次
CU      C [MW22]
```

从上面的示例程序中，可以看出地址指针存储于MW20中，可以使用普通指令对MW20进行操作，MW20中存储数值的变化直接影响定器、计数器访问的地址。

数据块可以使用DB打开，也可以使用DI打开，如果地址指针为0，CPU不会报错，使用16位地址指针访问数据块的示例程序如下：

```
L       20
T       MW      40
OPN     DB [MW40]           //打开DB20
L       1
+I
T       MW      42
OPN     DI [MW42]           //打开DI21
```

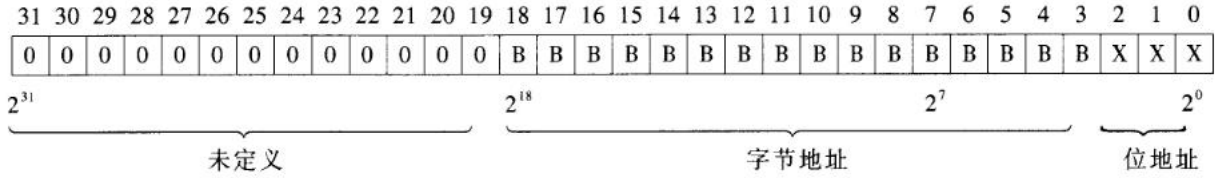
使用16位地址指针调用程序块（FB、FC）时，只能使用UC（无条件调用）或CC（有条件调用），而不能使用CALL指令，函数块中不能带有任何接口参数或静态变量，使用指针调用函数及函数块的示例程序如下：

```
L       12
T       LW      20
UC      FC [LW20]           //无条件调用FC12
L       13
T       MW      20
A       I       2.3
CC      FB [MW20]           //如果I2.3为1，调用FB13
```

FC12和FB13不能带有形参，这是由CC和UC调用指令决定的。

2. 32位地址指针

32位地址指针用于I、Q、M、L数据块等存储器中位、字节、字及双字的寻址，32位的地址指针可以使用一个双字表示，第0位～第2位作为寻址操作的位地址，第3位～第18位作为寻址操作的字节地址，第19位～第31位没有定义，32位地址指针的格式如下：



访问时需要使用地址存储器标识符及 32 位地址指针，地址寻址表示格式为：
地址存储器标识符［32 位地址指针］，例如指针存储于 LD20 中，装载 M 存储器一个字节表示为：

32 位地址指针也可以使用常数表示，例如装载 32 位指针常数 L P#40.3（P = 指针，字节地址 = 40，位地址 = 3）。32 位地址指针数据与双整数可以相互转换，由于指针指到一个位地址上，地址指针最小变化为一个位，相邻两个位的指针转换为双整数相差 1，例如 P#0.0 转换双整数为 L#0，P#0.1 转换双整数为 L#1，每一个字节地址加 1，相应转换的整数值加 8 的倍数，例如 P#2.0 转换双整数为 L#16，P#3.1 转换双整数为 L#25，在指针寻址时，可以使用指针的格式，也可以使用整数格式进行运算。使用 32 位地址指针寻址参考下面的示例程序：

```
OPN     DB      1       //打开 DB1
OPN     DI      3       //打开 DB3，最多可以同时打开两个 DB 块
L       4               //装载 4 到累加器 1 中
SLD     3               //累加器 1 中数值左移 3 位
T       MD      20      //将逻辑操作结果传送到 MD20 中，MD20 包含地址
                          指针为 P#4.0
L       P#20.0          //将地址指针 P#20.0 装载到 MD24 中
T       MD      24
L       320             //320 转换指针为 P#40.0 并装载到 MD28 中
T       MD      28
L       DBW［MD20］     //装载 DB1.DBW4
L       DBW［MD24］     //装载 DB1.DBW20
+I                      //相加
L       DIW［MD28］     //装载 DB3.DBW40
-I                      //相减
T       DIW     2       //将运算结果传送到 DB3.DBW2 中
```

使用 32 位地址指针寻址数据块地址时，数据块必须先打开，然后才能寻址，数据块寻址方法参考下面的示例程序，如果直接使用指令，则对完整数据格式地址（例如地址 DB1.DBB［MD100］）进行间接寻址被视为非法。

使用 LOOP 指令和 32 位地址指针可以进行循环操作，假设一个编程应用：一个字变量（MW2）与一个数组（假设存储于 DB1 中，包含 100 个元素为字的数组）存储的值相比较，

如果数值相同，则指出第一个相同数值存储在 DB 块中的位置（数组中的位置）。使用通常的编程方法，需要逐字进行比较，因而程序量比较大，如果实际需要与很多个数值比较，一个小型的 CPU 将无法完成控制任务，使用 LOOP 指令和地址指针相结合的方法，可以轻松地解决上述问题，参考下面的示例程序：

```
      L     0                    //初始化 MW100 和 MD4
      T     MW    100
      T     MD    4
      OPN   DB    1              //打开 DB1
      L     100                  //循环操作的次数，100 次
next: T     MW    100            //将循环 100 次装载到 MW100 中，固定格式
      L     MW    2              //进行比较的数值存储于 MW2
      L     DBW [MD4]            //与 DB 块中存储的值进行比较，开始地址为 DBW0
      =I                         //如果数值相等跳到 m1
      JC    m1
      L     MD    4              //将地址指针加 2（每个相邻的字地址相差 2）
      L     P#2.0
      +D
      T     MD    4
      L     MW    100            //次数减 1，跳回 next，如果 MW100 等于 0，跳出循
                                   环操作 LOOP 指令，LOOP 指令固定格式
      LOOP  next
m1: FP    M     10.0           //如果数值相等，记录 MD4 存储的指针，转换为数
                                 组的位置［（地址值/P#2.0）+1］值，并存储于
                                 MD8 中
      JCN   m2
      L     MD    4
      L     P#2.0
      /D
      +     1
      T     MD    8
m2: NOP   0
```

上述示例程序中编程量减少，但是程序执行时间并没有减少。32 位地址指针指向一个位地址，在编程应用中注意 P#1.0 不等于整数 1，否则 CPU 会因为无法处理而停机，参考下面错误的程序：

```
      OPN   DB    1
      L     20
      T     MD    20             //MD20 装载的地址指针为 P#2.4
      L     11
      T     DBB [MD20]           //指针指向 P#2.4，相当于 L DBB2.4，CPU 因无法
```

识别地址而停机

编程小技巧：

如果对字节进行操作，则指针转换为双整数最小变化率必须为 8（指针为 P#1.0）的倍数；如果对字进行操作，则指针转换为双整数值最小变化率必须为 16（指针为 P#2.0）的倍数；如果对双字、浮点变量进行操作，则指针转换为双整数值最小变化率必须为 32（指针为 P#4.0）的倍数，对指向字与双字的地址指针，这样的要求避免数据间的冲突，例如 DBW［MD2］，MD2 为 16 的倍数时，按照 DBW2、DBW4、DBW6 寻址，如果为 8 的倍数，按照 DBW1、DBW2、DBW3 寻址，则造成数据冲突。

8.2.2　寄存器间接寻址

与存储器间接寻址不同，寄存器间接寻址使用 CPU 内部集成的两个 32 位地址寄存器 AR1、AR2 存储地址指针。使用地址寄存器指令可以对 AR1、AR2 地址寄存器进行操作，指令如下所示：

LAR1　　：将 ACCU1 存储的地址指针写入 AR1。
LAR1 <D>　　：将指明的地址指针写入 AR1，例如 LAR1 P#20.0 或 LAR1 MD20。
LAR1 AR2　　：将 AR2 的内容写入 AR1。
LAR2　　：将 ACCU1 存储的地址指针写入 AR2。
LAR2 <D>　　：将指明的地址指针写入 AR2，与 LAR1 <D> 使用方式相同。
TAR1　　：将 AR1 存储的地址指针传输给 ACCU1。
TAR1 <D>　　：将 AR1 存储的地址指针传输给指明的变量中。
TAR1 AR2　　：将 AR1 存储的地址指针传输给 AR2。
TAR2　　：将 AR2 存储的地址指针传输给 ACCU1。
TAR2 <D>　　：将 AR2 存储的地址指针传输给指明的变量中。
CAR　　：交换 AR1 和 AR2 的内容。

寄存器间接寻址分为 32 位内部区域指针和 32 位交叉区域指针。

1. 32 位内部区域指针

32 位内部区域指针用于 I、Q、M、L 数据块等存储器中位、字节、字及双字的寻址，与 32 位存储器指针的使用相同，不同之处在于指针存储的位置不同。32 位内部区域地址指针的格式如下：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	X	X	X

2^{31}　　2^{18}　　2^{7}　　2^{0}

未定义　　字节地址　　位地址

第 0 位～第 2 位作为寻址操作的位地址，第 3 位～第 18 位作为寻址操作的字节地址，第 19 位～第 30 位没有定义，第 31 位为内部区域与交叉区域指针标识，0 表示内部区域指针，1 表示交叉区域指针。

32 位内部区域指针地址寻址表示格式为：地址存储器标识符［地址寄存器，地址偏移常量］，例如装载 M 存储器一个字节表示为：

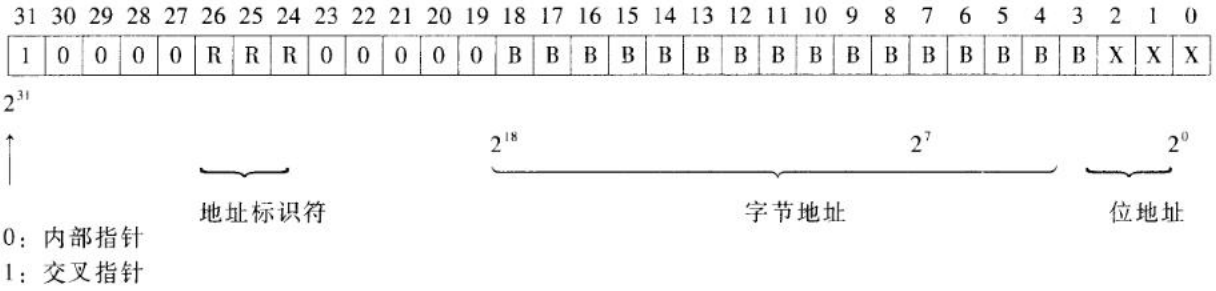
```
L       M       B       [AR1/2,   P#10.0]
↑       ↑       ↑       ↑         ↑
装载指令  存储区域  访问宽度  地址寄存器  地址偏移常量
```

指针指向地址 = 地址寄存器存储地址 + 地址偏移常量，如果 AR1 装载的地址为 P#8.0，实际装载的地址为 MB18。32 位内部区域指针的使用方法参考下面的示例程序：

```
OPN     DB      1               //打开 DB1。
LAR1    P#10.0                  //将指针 P#10.0 装载到地址寄存器 1 中
L       DBW [AR1, P#12.0]       //将 DBW22 装载到累加器 1 中
LAR1    MD      20              //将存储于 MD20 中的指针装载到地址寄存器 1
                                  中
L       DBW [AR1, P#0.0]        //将 DBW 装载到累加器 1 中，地址存储于 MD20
                                  中
+I
LAR2    P#40.0                  //将指针 P#40.0 装载到地址寄存器 2 中
T       DBW [AR2, P#0.0]        //运算结果传送到 DBW40 中
```

2. 32 位交叉区域指针

32 位交叉区域指针与 32 位内部指针相比，地址指针中带有存储区域标识符，如 I、Q、M 等，32 位交叉区域地址指针的格式如下：



第 0 位 ~ 第 2 位作为寻址操作的位地址，第 3 位 ~ 第 18 位作为寻址操作的字节地址，第 24 位 ~ 第 26 位为地址标识符，表示的地址区域如下：

000 表示没有地址区，例如 P#12.0；

001 表示输入地址区 I，例如 P#I12.0；

010 表示输出地址区 Q，例如 P#Q12.0；

011 表示标志位地址区 M，例如 P#M12.0；

100 表示数据块（DB）中的数据，例如 P#DB1.DBX12.0

101 表示数据块（DI）中的数据，例如 P#DI1.DIX12.0

110 表示区域地址区 L，例如 P#L12.0；

111 表示调用程序块的区域地址区 V，例如 P#V12.0；

第 31 位为内部区域与交叉区域指针标识，0 表示内部区域指针，1 表示交叉区域指针。使用交叉区域指针的表示方法（例如装载 M 存储器一个字节）为：

```
LAR1/2  P#M20.0             //装载地址指针 P#M20.0 到 AR1 或 AR2。
```

```
L          B        [AR1/2,    P#10.0]
↑          ↑          ↑          ↑
装载指令  访问宽度  地址寄存器  地址偏移常量
```

指针指向地址 = 地址寄存器存储地址 + 地址偏移常量，上例中实际装载的地址为 MB30。如果访问一个位信号，则没有访问宽度。32 位交叉区域指针的使用方法参考下面的示例程序：

```
LAR1    P#M 20.0          //将指针 P#M20.0 装载到地址寄存器 1 中
A       [AR1, P#1.1]      //M21.1 "与" 操作
=       Q       1.2       //如果 M21.1 为 1，则输出 Q1.2 为 1
L       P#I 40.0          //将指针 P#I40.0 装载到累加器 1 中
LAR2                      //将累加器 1 中存储的地址指针装载到地址寄存器 2
                            中
L       W [AR2, P#0.0]    //装载 IW40.0 到累加器 1 中
T       MW      60        //将累加器 1 中存储的数值传送到 MW60 中
```

3. 地址寄存器 AR1、AR2 的监控

存储于存储器的地址指针可以直接通过变量监控表监控，存储于 AR1、AR2 中的地址指针则需要使用 STL 程序 "Monitor" 进行监控，监控之前必须在程序块中的菜单进行设置，点击菜单 "Options" → "Customize" → "STL" 界面中，选择 "Address Register1" 和 "Address Register2" 选项，在程序监控中可以显示 AR1、AR2 的内容，如图 8-2 所示。

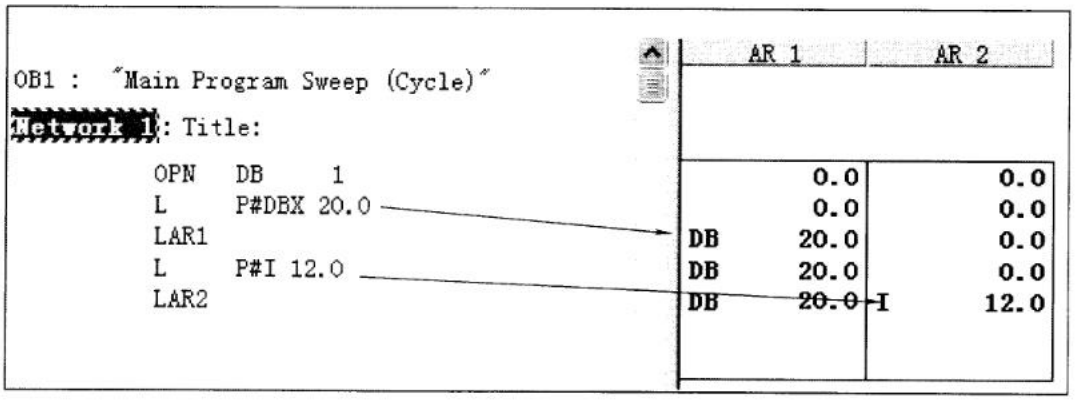


图 8-2　监控 AR1、AR2

4. 地址寄存器 AR1、AR2 的限制

存储区间接寻址和寄存器间接寻址具有相同的功能，在一些特殊条件下，使用地址寄存器 AR1、AR2 会有下列的限制：

1）在形参的传递中，STEP7 使用地址寄存器 AR1 访问函数（FC）接口及函数块（FB）"INOUT" 接口中定义的复合类型参数，如 ARRAY、STRUCT、DATE_AND_TIME 等，AR1 和 DB 块寄存器中的内容将被覆盖，例如在 FC1 中 "IN" 接口中定义一个数组变量，在 OB1 中调用，使用 OB1 的 L 区域数据进行赋值，调用关系如图 8-3 所示。

如果在 FC1 中访问数组变量的元素，如 ARR_TEST [1]，地址寄存器 AR1 及 DB 块寄

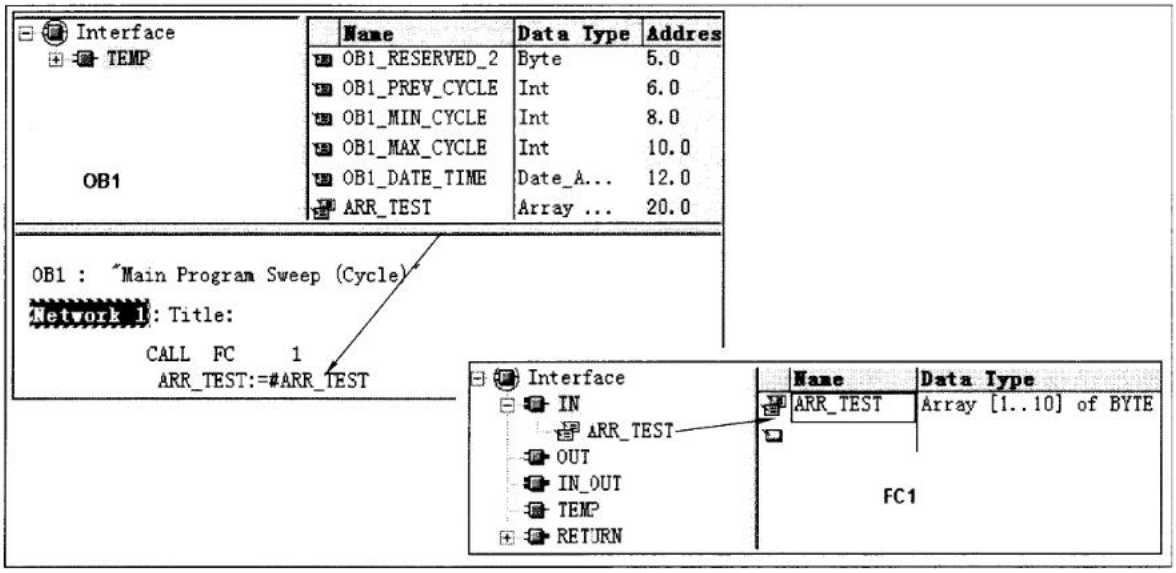


图 8-3 调用关系

存器会发生变化，示例程序如图 8-4 所示。

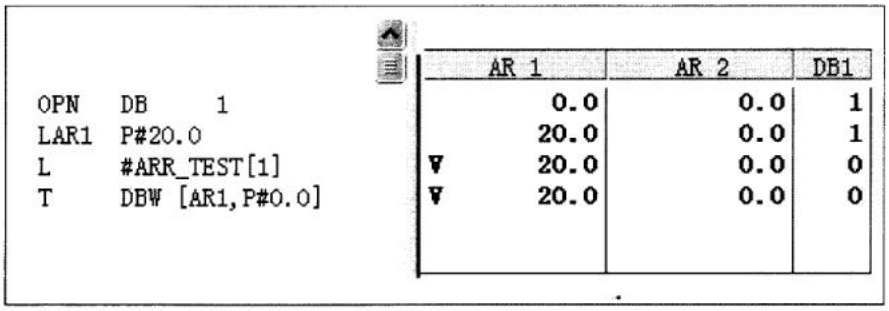


图 8-4 访问 FC 中复合数据类型数据

图 8-4 示例程序中前两条语句中打开 DB1，并将 P#20.0 装载到 AR1 中，在第三条语句访问数组的一个元素后，AR1 存储的地址指针变为 P#V20.0（指向 OB1 中实参 ARR_TEST 的地址，参考图 8-3），DB 块寄存器中存储的值为 0，程序下传到 CPU 后将出现故障报警。将程序的执行次序进行修改，CPU 即可运行，修改后的程序如下：

```
L     #ARR_TEST [1]     //装载形参变量 ARR_TEST [1] 到累加器 1 中
OPN   DB      1         //打开 OB1
LAR1  P#20.0            //将 P#20.0 装载到地址寄存器 AR1 中
T     DBW [AR1, P#0.0]  //将累加器 1 中的值传送到 DB1.DBW20 中
```

执行次序修改后可能会造成不必要的错误，访问复合变量时建议使用存储区间接寻址。访问 FC、FB 区域变量 L，不会覆盖 AR1、DB 块寄存器。

2）AR2 和 DI 寄存器分别包含 FB 背景数据块的块号及在背景数据中偏移地址（多重背景数据块），在 FB 中，使用 AR2 和 DI 寄存器将会覆盖系统存储的内容。如果必须在 FB 中使用 AR2 和 DI 寄存器，建议使用下面的方法处理 AR2 和 DI 寄存器，首先保存 AR2 和 DI 寄存器中的数据，程序如下：

```
TAR2    MD      100        //将 AR2 的数据存储于 MD100 中
L       DINO               //将背景数据块 DB 的块号存储于 MW104 中
T       MW      104
```

然后编写用户程序，可以对 AR2 和 DI 寄存器进行操作，但是在程序中不能访问 FB 参数或静态变量。使用完成后恢复 AR2 和 DI 寄存器的系统值，程序如下：

```
LAR2    MD      100        //将 MD100 中存储的地址指针装载到 AR2 中
OPN     DI [MW104]         //打开 DI 数据块
```

3）P# < 地址 > 可以直接指向布尔型变量，例如 P#M20.0、P#DBX12.0 等，也可以指向程序块中接口参数区、区域数据区 L 及静态变量，例如 P##PARA（参数）指向 PARA 参数的开始地址。将指向函数 FC 的 IN、OUT、INOUT 类型参数的指针装载到地址寄存器中不可以直接操作，例如：

```
LAR1    P##PARA（参数）    //非法指令
```

必须先装载到累加器中，然后再装载到地址寄存器中，例如：

```
L       P##PARA（参数）    //将地址指针装载到累加器 1
LAR1/2                     //将累加器 1 中存储的地址指针装载到地址寄存器中
```

8.3　程序块参数—POINTER 与 ANY 数据类型指针

STEP7 中除提供 16 位、32 位存储器地址指针和 32 位寄存器地址指针外，在程序块 FC、FB 的接口参数中还提供 48 位（“POINTER”数据类型）地址指针和 80 位（“ANY”数据类型）地址指针，16 位和 32 位地址指针可以直接装载到存储器或地址寄存器中，从而可以直接在程序块中使用进行间接寻址，“POINTER”和“ANY”数据类型指针由于大于 32 位而不能装载到存储器或寄存器中，所以不能在程序块中直接使用，必须进行拆分使用。这两种地址指针专用于函数（FC）及函数块（FB）接口参数的传递，例如调用函数赋值形参时，对实参的完全寻址。

8.3.1　POINTER 数据类型指针

POINTER 数据类型指针用于向被调用函数（FC）及函数块（FB）传递复合数据类型（如 ARRAY、STRUCT 及 DT 等）的实参。在被调用的函数（FC）及函数块（FB）内部可以间接访问实参的存储器。

POINTER 指针占用 48 位地址空间，数据格式如下：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DB 块号（或 0）															
数据区								0	0	0	0	0	B	B	B
B	B	B	B	B	B	B	B	B	B	B	B	B	X	X	X

B = 字节地址　　X = 位地址

POINTER 指针前 16 位的数值表示数据块（DB 或 DI）的块号，如果指针没有指向一个

DB 块，则数值为 0，POINTER 指针可以指向的数据区见表 8-2。

表 8-2 POINTER 指针数据区表示的地址区

16 进制代码	数 据 区	简 单 描 述
B#16#81	I	过程映像输入区
B#16#82	Q	过程映像输出区
B#16#83	M	标志位
B#16#84	DB	数据块
B#16#85	DI	背景数据块
B#16#86	L	区域数据区
B#16#87	V	上一级赋值的区域数据

与区域交叉指针相比，POINTER 类型指针可以直接指向一个数据块中的变量，例如 P#DB1. DBX0. 0。

调用 FB、FC 时，对 POINTER 指针数据类型的形参进行赋值时，可以选择指针显示方式为直接赋值，例如：

```
P#DB2. DBX12. 0          //指向 DB2. DBX12. 0
P#M12. 1                 //指向 M12. 1
```

也可以选择使用地址声明或符号名（不使用符号 P#）的方式进行赋值，例如：

```
DB2. DBX12. 0            //指向 DB2. DBX12. 0
M12. 1                   //指向 M12. 1
```

在被调用的 FC、FB 中，需要对 POINTER 指针数据类型形参拆分，以便读出实参的地址，下面以示例的方式介绍 POINTER 指针的使用，例如编写一个计算功能的函数（FC3），在输入参数“In_Data”输入首地址，在输入参数“NO”输入变量（浮点格式，以首地址开始，地址连续即每隔 4 个字节为一个浮点变量）的个数，在输出参数“OUT_VAL”输出几个变量的平均值。OB1 中调用函数 FC3 的程序如下：

```
CALL      FC      3      //调用函数 3
In_Data:  = P#M100. 0    //输入的首地址
NO        :  = 4         //变量的个数
OUT_VAL:  = MD20         //计算结果
```

完成的计算功能相当于 MD20: =（MD100 + MD104 + MD108 + MD112）/4。在函数（FC3）的接口参数中定义输入、输出变量及临时变量，见表 8-3 所示。

表 8-3 FC3 接口参数

数 据 接 口	名 称	数 据 类 型	地 址
IN	In_Data	Pointer	
IN	NO	INT	
OUT	OUT_VAL	REAL	
TEMP	BLOCK_NO	INT	0. 0
TEMP	NO_TEMP	INT	2. 0
TEMP	ADD_TEMP	REAL	4. 0

FC3 中的示例程序如下：

```
      L     0                    //初始化临时变量#ADD_TEMP
      T     #ADD_TEMP
      L     P##In_Data           //指向存储地址指针 P#M100.0 的首地址，并装载到
      LAR1                         地址寄存器 AR1 中
      L     0                    //判断 OB1 中赋值的地址指针是否为数据块（参考
      L     W [AR1, P#0.0]         POINTER 的数据格式）
      =I
      JC    M1
      T     #BLOCK_NO
      OPN   DB [#BLOCK_NO]       //如果是 DB 块，打开指定的 DB 块
M1:   L     D [AR1, P#2.0]       //找出需要计算数据区的开始地址，POINTER 数据
                                   中，后 4 个字节包含内部交叉指针，将 P#M100.0
                                   装载到 AR1 中
      LAR1
      L     0
      L     #NO                  //如果输入变量个数为 0，结束 FC3 的执行。如果
      =I                           不等于 0 作为循环执行的次数（NO_TEMP）
      JC    END
NO:   T     #NO_TEMP             //如果不为 0，作为循环执行的次数。循环执行加运
                                   算，本例中循环执行的次数为 4
      L     D [AR1, P#0.0]       //装载 MD100 到累加器 1 中
      L     #ADD_TEMP            //与临时变量#ADD_TEMP 相加后，将计算结果再存
                                   储于#ADD_TEMP 中
      +R
      T     #ADD_TEMP
      +AR1  P#4.0                //地址寄存器加 4，下一次与 MD104 相加
      L     #NO_TEMP             //LOOP 指令固定格式
      LOOP  NO                   //跳回“NO”循环执行，执行完定义在变量#NO_
                                   TEMP 的次数后，自动跳出循环程序
      L     #ADD_TEMP            //求平均值，装载运算结果到累加器 1 中
      L     #NO
      DTR                        //将变量个数转变为浮点值便于运算
      /R
      T     #OUT_VAL             //输出运算结果
END:  NOP   0
```

注意：

在 OB1 中调用 FC3 时，如果需要对指针类型的“In_Data”参数赋值为变量（指向的地址区为变量），可以在 OB1 中使用 POINTER 指针变量预先赋值（在临时变量或 DB 块中定

义 POINTER 数据类型变量），根据 POINTER 指针的数据格式，通过改变 POINTER 指针组成部分的值而改变赋值的地址指针，达到将输入参数“In_Data”作为变量的目的。

8.3.2 ANY 数据类型指针

ANY 数据类型指针主要用于为系统函数（SFC）及系统函数块（SFB）分配参数。用户程序中也可以使用 ANY 数据类型指针，作为程序块的接口参数传递数据。

ANY 数据类型指针由数据类型、数据长度、DB 块号、存储器数据开始地址组成，占用 80 位地址空间，数据格式如下：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
10H（保留）								数据类型							
数据长度															
DB 块号（或 0）															
数据区								0	0	0	0	0	B	B	B
B	B	B	B	B	B	B	B	B	B	B	B	B	X	X	X

B = 字节地址　　X = 位地址

ANY 指针使用的数据类型见表 8-4。

表 8-4 ANY 指针使用的数据类型

数据类型代码		
十六进制代码	数 据 类 型	简 单 描 述
B#16#00	NIL	空
B#16#01	BOOL	位
B#16#02	BYTE	8 位字节
B#16#03	CHAR	8 位字符
B#16#04	WORD	16 位字
B#16#05	INT	16 位整形
B#16#06	DWORD	32 位双字
B#16#07	DINT	32 位双整形
B#16#08	REAL	32 位浮点
B#16#09	DATE	IEC 日期
B#16#0A	TIME_OF_DAY（TOD）	24 小时时间
B#16#0B	TIME	IEC 时间
B#16#0C	S5TIME	SIMATIC 时间
B#16#0E	DATE_AND_TIME（DT）	时钟
B#16#13	STRING	字符串
B#16#17	BLOCK_FB	FB 号
B#16#18	BLOCK_FC	FC 号
B#16#19	BLOCK_DB	DB 号
B#16#1A	BLOCK_SDB	SDB 号
B#16#1C	COUNTER	计数器
B#16#1D	TIMER	定时器

指针中的数据长度表示指向一个数据区域，例如指向整个数组、结构体等；如果 ANY 指针没有指向一个 DB 块，DB 块号将为 0；ANY 指针的数据区与 POINTER 指针数据区定义相同（参考表 8-2）。

与 POINTER 指针相比，ANY 类型指针可以表示一段数据区域，例如 P#DB1. DBX0. 0 BYTE10，表示指向 DB1. DBB0 ~ DB1. DBB9。调用 FB、FC 时，对 ANY 指针数据类型的形参进行赋值时，可以选择指针显示方式为直接赋值，例如：

```
P#DB2. DBX12. 0   WORD   22          //指向从 DB2. DBW12 开始 22 个字
P#M12. 1   BOOL   10                 //指向从 M12. 1 开始 10 个位信号
```

也可以选择使用地址声明或符号名（不使用符号 P#）的方式进行赋值，例如：

```
DB2. DBW12                           //指向 DB2. DBW12 一个字，数据长度为 1
M12. 1                               //指向 M12. 1 一个位信号，数据长度为 1
```

使用地址声明或符号名只能指向一个变量。下面以示例的方式介绍 ANY 指针的使用，实现与 POINTER 指针示例相同的功能。编写一个计算功能的函数（FC13），输入参数“In_Data”为一个数组变量，如果数组元素为浮点数，输出所有元素的平均值“OUT_VAL”，如果数组元素为其他数据类型，不执行计算功能。OB1 中调用函数（FC13）的程序如下：

```
CALL    FC       13                  //调用函数 13
In_Data:  = P#DB1.  DBX0. 0 REAL 8   //输入数据区从 DB1. DBD0 开始 8 个浮点值
OUT_VAL:  = MD20                     //计算结果
```

完成的计算功能相当于 MD20: = (DB1. DBD0 +.. +.. + DB1. DBD28)/8。在函数（FC13）的接口参数中定义输入、输出变量及临时变量见表 8-5 所示。

表 8-5 FC13 接口参数

数据接口	名　称	数据类型	地　址
IN	In_Data	ANY	
OUT	OUT_VAL	REAL	
TEMP	DATA_LEN	INT	0. 0
TEMP	BLOCK_NO	INT	2. 0
TEMP	ADD_TEMP	REAL	4. 0
TEMP	DATA_NO	INT	8. 0

FC13 中的示例程序如下：

```
L       0                    //初始化临时变量#ADD_TEMP
T       #ADD_TEMP

L       P##In_Date           //指向存储地址指针 In_Date 首地址，并装载到
LAR1                           地址寄存器 AR1 中
L       B [AR1, P#1. 0]      //如果数据类型不是 REAL，跳转到 END
L       B#16#8
< >I
JC      END

L       0
```

```
      L     W [AR1, P#4.0]   //判断 OB1 中赋值的地址指针是否为数据块（参考
      =I                       ANY 的数据格式）
      JC    M1
      T     #BLOCK_NO
      OPN   DB [#BLOCK_NO]   //如果是 DB 块，打开指定的 DB 块

M1:   L     W [AR1, P#2.0]   //判断 ANY 指针中数据长度，本例中为 REAL 变量的
      T     #DATA_LEN          个数
      L     D [AR1, P#6.0]   //找出需要计算数据区的开始地址，本例中为
                               DB1.DBX0.0
      LAR1
      L     #DATA_LEN
NO:   T     #DATA_NO         //循环执行加运算，本例中循环执行的次数为 8
      L     D [AR1, P#0.0]   //装载 DB1.DBD0 到累加器 1 中
      L     #ADD_TEMP        //与临时变量#ADD_TEMP 相加后，将计算结果再存储
      +R                       #ADD_TEMP 中
      T     #ADD_TEMP
      +AR1  P#4.0            //地址寄存器加 4，地址偏移量
      L     #DATA_NO         //LOOP 指令固定格式
      LOOP  NO               //跳回“NO”循环执行，执行完定义在变量#NO_
                               TEMP 的次数后，自动跳出循环程序
      L     #ADD_TEMP        //求平均值，装载运算结果到累加器 1 中
      L     #DATA_LEN
      DTR                    //将变量个数转变为浮点值
      /R
      T     #OUT_VAL         //输出运算结果
END:  NOP   0
```

注意：

在 OB1 中调用 FC13 时，如果需要对指针类型的“In_Data”参数赋值为变量（指向的地址区为变量），可以在 OB1 中使用 ANY 指针变量预先赋值（在临时变量或 DB 块中定义 ANY 数据类型变量），根据 ANY 指针的数据格式，通过改变 ANY 指针组成部分的值而改变赋值的地址指针，达到将输入参数“In_Data”作为变量的目的。

8.4 FB 在多重数据块中的寻址

在一些编程应用中，一些功能计算或包含部分工艺的函数或函数块往往由开发人员编写，例如编写 FB 块，通常情况下不会考虑在程序中的调用次序。工程人员在程序中调用这些已经开发完成的 FB，实现整个的控制任务。在一个 FB 中，将实现固定功能的 FB 块作为静态变量使用，生成多重背景数据块 DB，每个实现固定功能 FB 块的接口参数都会占用多

重背景数据块 DB 的空间，如图 8-5 所示。

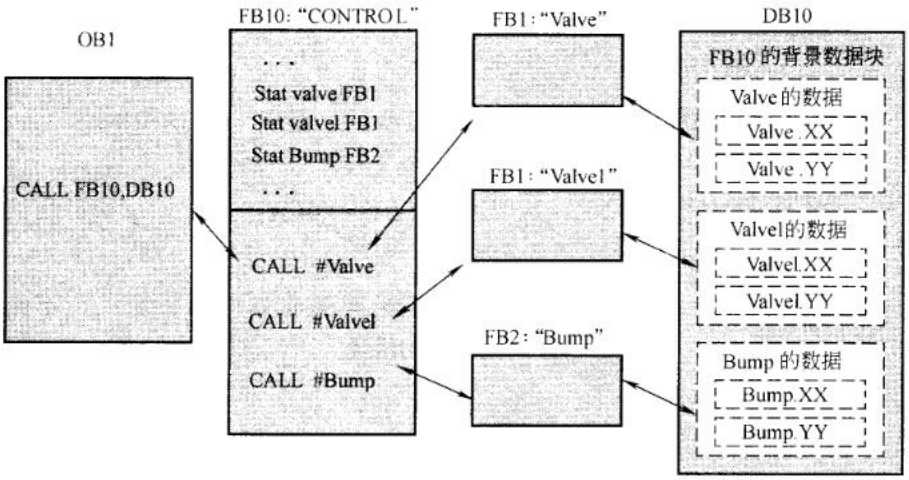


图 8-5　生成多重背景数据块（DB）

在 FB10 中，将具有独立控制功能的 FB1、FB2 作为静态变量调用，FB10 在 OB1 中调用生成背景数据块 DB、DB10，在背景数据块中包含 FB10、FB1、FB2 的接口参数，每个 FB 块接口参数在 DB10 中的位置与在 FB10 静态变量中的次序有关。

如果在 FB1、FB2 中使用 POINTER 或 ANY 类型地址指针进行拆分时，如果不考虑在多重背景数据块 DB 中的位置，将会造成错误，例如在 FB1 中定义输入接口参数 FB1_POS，数据类型为 POINTER，FB1 中的程序如下：

```
L       P##FB1_POS          //指向存储地址指针 FB1_POS 首地址
LAR1                        //存储于地址寄存器 1 中
L       D [AR1, P#2.0]      //装载实参赋值的地址指针，并传送到 MD20 中
T       MD      20
```

同样在 FB2 中定义输入接口参数 FB2_POS，数据类型为 POINTER，在 FB2 中的程序如下：

```
L       P##FB2_POS          //指向存储地址指针 FB2_POS 首地址
LAR1                        //存储于地址寄存器 1 中
L       D [AR1, P#2.0]      //装载实参赋值的地址指针，并传送到 MD24 中
T       MD      24
```

在 FB10 中，将 FB1、FB2 作为静态变量使用，见表 8-6 所示。

表 8-6　FB10 的接口参数

数据接口	名　称	数据类型	地　址
STAT	FB1_POS	FB1	0.0
STAT	FB2_POS	FB2	6.0

FB10 的程序如下：

```
CALL    #FB1_POS            //调用 FB1，赋值地址指针 P#M100.0
FB1_POS: =P#M 100.0
```

```
CALL    #FB2_POS            //调用 FB2，赋值地址指针 P#M120.0
FB2_POS: =P#M 120.0
```

在 OB1 中，调用 FB10，并生成 DB10，程序如下：

```
CALL    FB      10,     DB10        //调用 FB10，生成 DB10
```

希望将 P#100.0 赋值到 MD20 中，将 P#120.0 赋值到 MD24 中，但程序执行后 MD20 和 MD24 存储的地址指针同为 P#M100.0，这是由于 FB1 中的指令 L P##FB1_POS 与 FB2 中的指令 L P##FB2_POS 同时指向多重背景数据块 DB10 中 FB1 接口数据区的首地址 DB10.DBX0.0，以地址 DB10.DBX0.0 开始的 POINTER 指针变量存储的地址指针为 P#M100.0；FB2 接口数据区在 DB10 的首地址为 DB10.DBX6.0，以地址 DB10.DBX6.0 开始的 POINTER 指针变量存储的地址指针为 P#M120.0。在 FB1、FB2 程序中，没有考虑接口数据区在多重背景数据块的偏移地址而造成地址冲突。系统将每一个 FB 块接口数据区在多重背景数据块中的偏移地址自动存储于地址寄存器 AR2 中，利用 TAR2 功能可以实现地址偏移。在 FB1 中实现地址自动偏移的示例程序如下：

```
TAR2                        //将偏移地址传送到累加器 1 中
L       DW#16#FFFFFF        //过滤地址区，如将 P#M20.0 变为 P#20.0
AD
L       P##FB1_POS          //将偏移地址与 FB1_POS 首地址相加，并装载到
                              AR1 中
+D
LAR1                        //得到 FB1 在多重背景数据块 DB 中的首地址
L       D [AR1, P#2.0]      //将 P#M100.0 装载到 MD20 中
T       MD      20
```

在 FB2 中，按上述的方法编写程序，将 P#M120.0 装载到 MD20 中，这样 FB1、FB2 在 FB10 中作为参数的次序没有限制。

提示：

使用间接寻址编程灵活、节省程序空间，但是不容易入门。在实际编程应用中，可选择比较简单、容易上手、易于监控的编程方法。