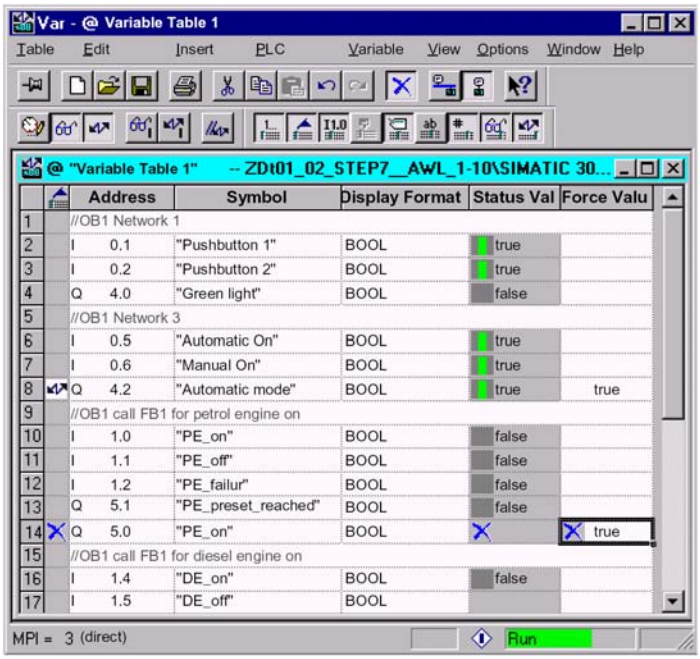
完整变量表的示例

下图所示是一个显示下述各栏的变量表：地址（Address）、符号（Symbol），显示格式（Monitor Format），监视值（Monitor Value）和修改值（Modify Value）



关于插入符号的注意

- 对需要修改的变量输入地址或符号，既可以在“符号”栏输入符号，也可以在“地址”栏输入地址。输入的条目自动写入正确的栏中。如果相应的符号已在符号表中定义，则符号栏或地址栏会自动填入。
- 只能输入已在符号表中定义过的符号。
- 符号必须与符号表中所定义的相同。
- 符号名中含有特殊字符则必须用引号括起来（例如，“Motor.off”，“Motor+Off”，“Motor-off”）。
- 要在符号表中定义新符号，可使用菜单命令**Options > Symbol Table**，也可以从符号表中复制符号然后粘贴到变量表中。

语法检查

在变量表中输入变量时，在每行的结束都会执行语法检查。任何不正确的输入都会被标为红色。如果将光标放在红色的行上，可以显示错误的原因。按F1可得到关于错误纠正的提示。

注意

如果用键盘而不用鼠标编辑变量表，应始能“Brief Information When Using the Keyboard”(使用键盘显示简短信息)特性。

如果需要，可以通过菜单命令**Option>Customize**修改变量表的设置，然后选择“General”标签栏。

最大限制

在变量表中每行最多可有255个字符。回车进入第二行是不可能的。一个变量表最多可有1024行，这是最大限制。

20.4.2 在变量表中插入一个连续的地址范围

1. 打开一个变量表。
2. 将光标指向地址列中的空闲位置，从该地址后将插入连续的地址范围。
3. 选择菜单命令**Insert > Range of Variables**。此时将出现“插入变量的范围”对话框。
4. 在“From Address”栏中输入起始地址。
5. 在“Number”栏中输入输入要插入列的数量。
6. 从显示清单中选择所需的显示格式。
7. 点击“OK”按钮。

此时在变量表中已经插入多个变量。

20.4.3 插入修改值

修改值作为注释

如果需要变量的“修改值”无效，可以使用菜单命令**Variable > Modify Value as Comment**。一个变量的修改值之前的注释符号“// ”表示它已经无效。注释符号“// ”插在“修改值”的前面，也可以代替调用菜单命令。通过再次调用菜单命令**Variable > Modify Value as Comment** 或删除注释符号，可以取消“修改值”的无效性。

20.4.4 定时器输入的上限

注意，下面是输入定时器的上限：

例如：W#16#3999（BCD格式的最大值）

举例

地址	监视格式	输入	修改值显示	解释
T 1	SIMATIC_TIME	137	S5TIME#130MS	转换为毫秒
MW4	SIMATIC_TIME	137	S5TIME#890MS	可用BCD格式表达
MW4	HEX	137	W#16#0089	可用BCD格式表达
MW6	HEX	157	W#16#009D	不能用BCD格式表达，所以监视格式不能选择为SIMATIC_TIME

注意

- 可以使用毫秒输入定时时间，但输入值会被作一些改变以适应时间格式。时间格式的大小要根据输入的时间值而定（137变成130ms，7ms被舍去）。
- 如果修改数据类型是WORD的变量，如IW1，被转换为BCD格式。但是，不是所有的位图都是有效的BCD码。如果将一个数据类型是WORD的变量表达为SIMATIC-TIME，则应用程序会自动转向缺省设置（这里是：HEX，请查看选择监视格式，缺省命令（“view”视窗菜单）），这样数值就可以被显示了。

用于SIMATIC-TIME格式变量的BCD码

使用类型为SIMATIC_TIME的变量输入数值时要用BCD码。16个位具有以下含义：

| 0 0 x x | h h h h | t t t t | u u u u |

位15和14　总是零

位13和12　（标为xx）为位0至位11设置乘数：

00=>乘数为10毫秒

01=>乘数为100毫秒

10=>乘数为1秒

11=>乘数为10秒

位11至8为百位（hhhh）

位7至4为十位（tttt）

位3至0为个位（uuuu）

20.4.5 计数器输入的上限

注意下面计数器输入的上限：

计数器的上限是：C#999

W#16#0999（BCD格式的最大值）

例如：

地址	监视格式	输入	修改值显示	解释
C1	COUNTER	137	C#137	转换
MW4	COUNTER	137	C#89	可用BCD格式表达
MW4	HEX	137	W#16#0089	可用BCD格式表达
MW6	HEX	157	W#16#009D	不可以表达为BCD格式，不能选择COUNTER格式显示

注意

- 如果为计数器输入一个十进制数但却没有输入标识符C#，这个值会自动转为BCD格式（137变为C#137）。
- 如果修改数据类型是WORD的变量，如IW1，被转换为BCD格式。但是，不是所有的位图都是有效的BCD码。如果，数据类型为WORD的变量值无法表达为COUNTER，则应用程序会自动转向缺省格式（这里为：HEX，请查看选择监视格式，缺省命令（view 视窗菜单）），这样数值就可以被显示了。

20.4.6 插入注释行

注释行以标识符“// ”开始。

如果需要使一行或多行变量表无效（作为一个注释行），可以使用菜单命令**Edit > Row not Effect** 或工具栏中相应的符号  。

20.4.7 举例

20.4.7.1 在变量表中输入地址示例

允许的地址：	数据类型：	举例（英语助记符）：
Input | Output | Bit memory	BOOL	I 1.0 | Q 1.7 | M 10.1
Input | Output | Bit memory	BYTE	IB 1 | QB 10 | MB 100
Input | Output | Bit memory	WORD	IW 1 | QW 10 | MW 100
Input | Output | Bit memory	DWORD	ID 1 | QD 10 | MD 100
I/O （Input | Output）	BYTE	PIB 0 | PQB 1
I/O （Input | Output）	WORD	PIW 0 | PQW 1
I/O （Input | Output）	DWORD	PID 0 | PQD 1

允许的地址：	数据类型：	举例（英语助记符）：
Timers	TIMER	T 1
Counters	COUNTER	C 1
Data block	BOOL	DB1.DBX 1.0
Data block	BYTE	DB1.DBB 1
Data block	WORD	DB1.DBW 1
Data block	DWORD	DB1.DBD 1

注意：

“DB0...”不允许使用，因为它已被内部占用。

在强制数值窗口：

- 对S7-300模板进行强制时，只有输入、输出和I/O（输出）是可以的。
- 对S7-400模板作强制时，只有输入、输出、位存储和I/O（输入/输出）是可以的。

20.4.7.2 输入连续地址范围的示例

打开一个变量表并用菜单命令 **Insert > Range of Variables** 调用“Insert Range of Variables（插入变量范围）”对话框。

作为对话框的条目，下列各行标志位存储区被插入到变量表中：

- From address（开始地址）：M 3.0
- Number（数量）： 10
- Display format（显示格式）：BIN

地址	显示格式
M 3.0	BIN
M 3.1	BIN
M 3.2	BIN
M 3.3	BIN
M 3.4	BIN
M 3.5	BIN
M 3.6	BIN
M 3.7	BIN
M 4.0	BIN
M 4.1	BIN

注意示例中“地址”栏中从第八行以后字节地址改变。

20.4.7.3 输入修改值和强制值的示例

位地址

可能的位地址	允许修改/强制值
I1.0	true
M1.7	false
Q10.7	0
DB1.DBX1.1	1
I1.1	2#0
M1.6	2#1

字节地址

可能的字节地址	允许修改/强制值
IB 1	2#00110011
MB 12	B#16#1F
MB 14	1F
QB 10	‘a’
DB1.DBB 1	10
PQB 2	-12

字地址

可能的字地址	允许修改/强制值
IW 1	2#0011001100110011
MW 12	w#16#ABCD
MW 14	ABCD
QW 10	B#(12,34)
DB1.DBW 1	‘ab’
PQW 2	-12345
MW3	12345
MW5	s5t#12s340ms
MW7	0.3s or 0,3s
MW9	c#123
MW11	d#1990-12-31

双字地址

可能的双字地址	允许修改/强制值
ID 1	2#00110011001100110011001100110011
MD 0	23e4
MD 4	2
QD 10	dw#16#abcdef10
QD 12	ABCDEF10
DB1.DBD 1	b#(12,34,56,78)

可能的双字地址	允许修改/强制值
PQD 2	‘abcd’
MD 8	l#-12
MD 12	l#12
MD 16	-123456789
MD 20	123456789
MD 24	t#12s345ms
MD 28	tod#1:2:34.567
MD 32	p#e0.0

定时器

可能的“Timer”类型地址	允许修改/强制值	解释
T 1	0	转为毫秒
T 12	20	转为毫秒
T 14	12345	转为毫秒
T 16	s5t#12s340ms	
T 18	3	转为1s300ms
T 20	3s	转为1s300ms

修改定时器只影响其数值不影响状态。这意味着定时器T1的时间值可设为零，但却不会改变A　T1的逻辑操作结果。

输入字符5t、s5time，不区分大小写。

计数器

可能的“Counter”类型地址	允许修改/强制值
C 1	0
C 14	20
C 16	c#123

修改计数器只影响其数值不影响其状态。这意味着可将计数器C1的数值修改为零，却不会影响A　C1的逻辑操作结果。

20.5　建立与 CPU 的连接

为了监视或修改变量表（VAT）中输入的变量，必须与相应的CPU建立连接。可以将每个变量表与不同的CPU建立链接。

显示在线连接

如果在线连接存在，变量表窗口标题栏中的“ONLINE（在线）”项会显示该情况。状态栏将显示操作状态“RUN”、“STOP”、“DISCONNECTED”或“CONNECTED”，这取决于CPU。

建立与CPU在线连接

如果与所需要的CPU没有建立在线连接，使用菜单命令PLC > Connect To > …来定义与所需CPU的连接，以便进行变量的监视或修改。

中断与CPU的在线连接

使用菜单命令PLC > Disconnect，可以中断变量表和CPU的连接。

注意

如果用菜单命令Table > New，生成了一个未命名的变量表，它将自动连接到最后组态的CPU上（变量表中带有连接信息）。

20.6 监视变量

20.6.1 监视变量介绍

可用以下方法监视变量：

- 用菜单命令Variable > Monitor，激活监视功能。所选变量的数值依据所设定的触发点和触发频率显示在变量表中。如果设置触发频率为“Every Cycle（每一次扫描）”，可以用菜单命令Variable > Monitor，将监视功能切换成无效。
- 可以使用菜单命令Variable > Update Monitor Values，对所选变量的数值作一次立即刷新。所选变量的当前数值则显示在变量表中。

用ESC键放弃“监视”

如果在监视功能激活的状态下按ESC键，该功能则不经询问就退出。

20.6.2 设定变量表监控触发

可以在编程器上显示用户程序中每个变量在程序处理过程中的某一特定点（触发点）的当前数值，以便对它进行监视。

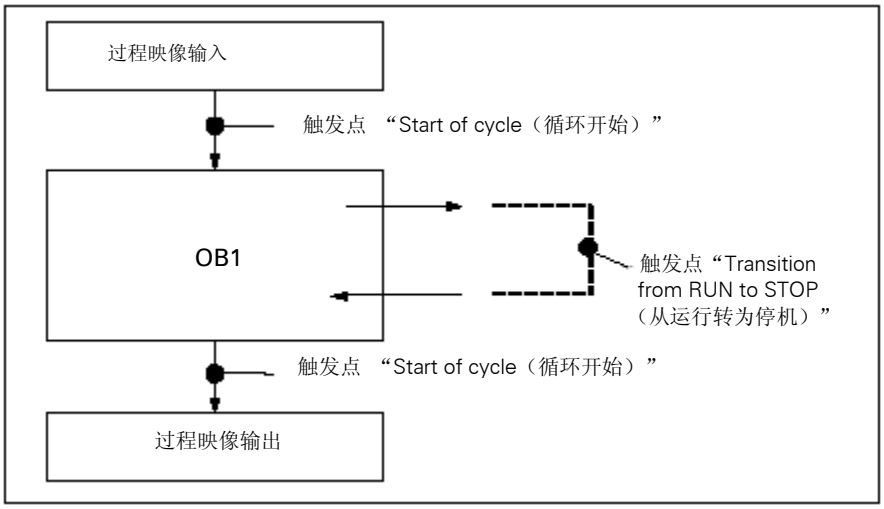
当选中一个触发点时，就决定了监视的变量在哪个时间点的数值被显示出来。

可以用菜单命令Variable > Trigger，设置触发点和触发频率。

触发	可能的设置
触发点	扫描开始 扫描结束 从RUN转换到STOP
触发频率	一次 每个扫描周期

触发点

下图所示为触发点的位置。



在“Status Value”栏中显示修改值，将监视的触发点设置在“Start of cycle”，将修改的触发点设置在“End of cycle”。

立即触发

可以用菜单命令**Variable > Update Monitor Values**，刷新所选变量的数值。这个命令即为“立即触发”，不参照用户程中的任何点，尽可能快地执行。这些功能主要用于CPU处于停机模式下的监视和修改。

触发频率

下表显示了触发频率对变量监视的影响：

	触发频率：一次	触发频率：每一个扫描
监视变量	依据触发点的设置，只触发一次	在定义的触发点上监视 当测试一个块时，可以准确地追踪处理的过程

20.7 修改变量

20.7.1 修该变量的介绍

可以利用下述方法进行变量修改：

- 用菜单命令Variable > Modify激活修改功能。在变量表中，根据触发点和触发频率的设定而对所选变量作的数值修改将应用到用户程序中。如果设置触发频率为“Every Cycle（每一扫描）”，可以用菜单命令Variable > Modify，将监视功能切换成无效。
- 用菜单命令Variable > Activate Modify Values，对所选变量的数据作一次立即刷新。强制功能和使能外设输出（PQ）提供一些其它功能的可能性。

进行修改时应注意下面的情况：

- 只能对在变量表中可以看到的地址进行修改。
  如果缩小变量表的可视区域，有一些也许已修改了的值将不能看到。如果将变量表可视区域扩大，则可看到一些未修改的地址。
- 如果变量被修改则不能撤销（比如，用Edit > Undo）。

危险

在程序运行中修改变量值，如果功能或程序中出错可能导致对人身或财产的损害。在执行“修改”功能前，要确认不会有危险情况出现。

用ESC键放弃“修改”

如果在“Modifying（修改）”功能进行过程中使用ESC键，该功能无询问退出。

20.7.2 设置变量表触发功能

可以在程序运行中的某一个特定点（触发点）给用户程序中的变量赋予固定值（一次或每一次扫描）。

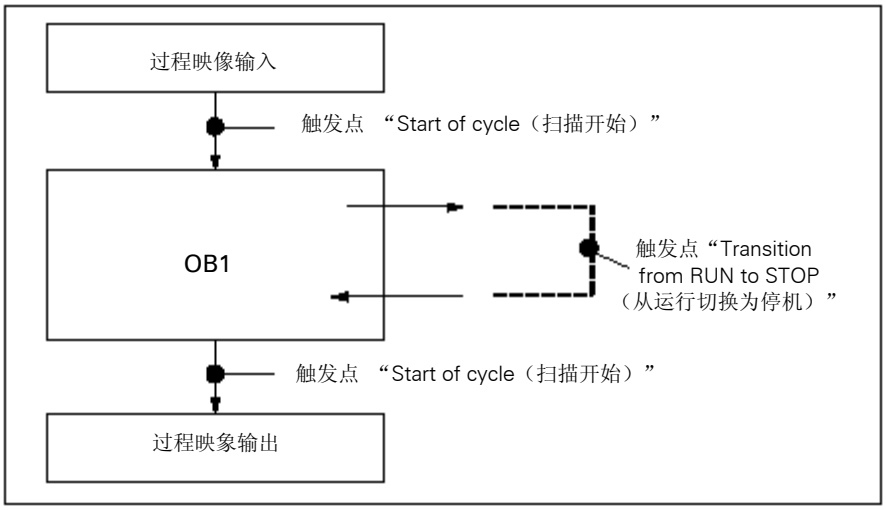
当选中一个触发点时，就决定了监视的变量在哪个时间点的数值被显示出来。

可以用菜单命令Variable > Trigger，设置触发点和触发频率。

触发	可能的设置
触发点	扫描开始 扫描结束 从运行转为停机
触发频率	一次 每一次扫描

触发点

下图所示为触发点的位置。



所示为触发点的位置：

- 修改输入只适用于触发点“Start of cycle”（对应于用户程序OB1的开始），因为在修改后将更新过程映象输入区。
- 修改输出只适用于触发点“End of cycle”（对应于用户程序OB1的结束），否则用户程序将覆盖过程映像输出区。

在“Status Value”栏中显示修改值，将监视的触发点设置在“Start of cycle”，将修改的触发点设置在“End of cycle”。

当变量修改时，下列情况适用不同的触发点：

- 如果触发频率设置为“Once”，而所选变量不能被修改时，会显示信息。
- 如果触发频率设置为“Every cycle”则无信息出现。

立即触发

可以使用菜单命令**Variable > Activate Modify Values**，对所选变量的数值进行修改。这个命令即为“立即触发”，不参照用户程中的任何点，尽可能快地执行。这个功能主要用于在CPU停机模式下修改。

触发频率

下表所示为触发频率的设定对变量修改的影响：

	触发频率：一次	触发频率：每一次扫描周期
修改变量	激活一次，与触发点无关，可以将数值赋给变量一次	在定义的触发点进行修改 通过赋予固定数值，可以为用户程序模拟某一情形，用这个功能可调已编好的功能

20.8 强制变量

20.8.1 强制变量时的安全措施

注意人员伤害和财产损失

注意，当使用“强制”功能时，任何不正确的操作都可能会：

- 危及人员的生命或健康，或者
- 造成设备或整个工厂的损失

小心

- 在开始强制功能之前必须检查确保同一时间在同一CPU上没有其他人在执行该功能。一个强制作业只能用菜单命令Variable > Stop Forcing来删除或终止。关闭强制数值窗口或退出 “Monitoring and Modifying Variables”应用程序并不能删除强制作业。
- 强制功能不能被取消（例如，用Edit > Undo）。
- 请阅读有关信息，了解强制和修改之间的差别。
- 如果一个CPU不支持强制功能，则在变量菜单中与强制有关的所有菜单命令都不能激活。

如果用菜单命令Variable > Enable Peripheral Output，使输出禁用失效，所有被强制的输出模板输出它们的强制值。

20.8.2 强制变量的介绍

可以给用户程序的每个变量赋予固定值，即使CPU中正在执行的用户程序也不能够改变或覆盖变量赋予的值。实现这一功能的要求是CPU支技该功能（如，S7-400的CPU）。通过将固定值赋给变量这一功能，可以为用户程序设置特定的情形并用该方法对已编程的功能进行测试。

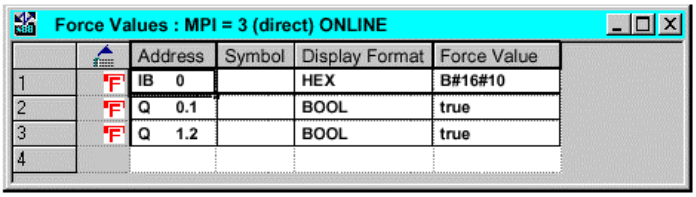
“强制数值（ForceValues）”窗口

只有当“强制数值”窗口处于激活状态，才能选择用于强制的菜单命令。

要显示这个窗口可选择菜单命令Variable > Display Force Values。

在CPU中打开一个“强制数值”窗口，被强制的变量和它们各自的强制值一起显示在窗口中。

强制数值窗口示例



当前在线连接的名字显示在**标题栏**中。

状态栏中显示的是从CPU中读出的强制作业的日期和时间。

如果没有激活的强制作业，该窗口是空的。

在“强制数值”窗口中**变量**不同的**显示**方法有以下含义：

显示	含　义
黑体	该变量在CPU中已被赋予固定值
正常	该变量正在被编辑
灰色	模板的变量在机架上不存在/未插入或者变量地址错误，显示错误信息

使用变量表中可强制的地址

如果在强制变量窗口中从变量表中输入一个变量，选择变量表及所需的变量，然后调用菜单命令Variable > Force values打开强制值窗口。将可以强制的变量输入到强制值窗口中。

使用CPU中的强制作业或建立新的强制作业

如果“强制数值”窗口是打开并且激活，则有另外的信息显示：

- 如果确认，该窗口中的改变将会被CPU中已存在的强制作业覆盖。用菜单命令Edit > Undo，可以重新存储前一窗口的内容。
- 如果取消，当前窗口中的内容就保留下来。
  然后可以用菜单命令Table > Save As，将“强制数值”窗口的内容存为一个变量表，或者选择菜单命令Variable > Force：这样就将当前窗口的内容写到CPU中作为一个新的强制作业。

变量的监视和修改只能在变量表中进行，不能在“强制数值”窗口。

删除强制数值

调用菜单命令Variable > Display Force Values打开强制数值窗口。然后调用菜单命令Variable > Delete Force从所选的CPU中删除强制值。

存储强制数值窗口

可以将强制数值窗口中的内容存到一个变量表中。使用菜单命令Insert > Variable Table，可以在一个强制数值窗口中重新插入已存储的内容。

在强制数值窗口中关于符号的提示

除非从其它的没有符号的应用程序中打开“监视和修改变量”应用程序，否则最后激活的窗口中的符号会被输入。

如果无法输入符号名，则“Symbol（符号）”这一栏被隐藏了。这种情况下，菜单命令 Option > Symlol Table没有激活。

20.8.3 强制和修改变量的区别

下表总结了强制和修改的区别：

特点/功能	S7-400(包 括 CPU 318-2DP)强制功能	S7-300(不包括CPU 318-2DP)强制功能	修改
位存储(M)	Yes	-	Yes
定时器和计数器(T、C)	-	-	Yes
数据块(DB)	-	-	Yes
外设输入(PIB、PIW、PID)	Yes	-	-
外设输出(PQB、PQW、PQD)	Yes	-	Yes
输入和输出(I、Q)	Yes	Yes	Yes
用户程序可覆盖修改/强制值	-	Yes	Yes
无中断替换强制数值	Yes	Yes	-
当应用程序退出时变量仍保持其数值	Yes	Yes	-
与CPU的连接断开后变量仍保持其数据	Yes	Yes	-
允许的寻址错误： 如：IW1 修改/强制值：1 IW1 修改/强制值：0	-	-	最后修改的成为效值
设置触发	总是立即触发	总是立即触发	一次或每次扫描周期
功能只影响激活窗口中可视区域的变量	影响所有强制值	影响所有强制值	Yes

注意

- “使能外设输出”时，在相应输出模板上外设输出的强制数值有效；而外设输出的修改值却无效。
- 使用强制功能，变量总是带有强制的数值。这些数值可以在用户程序的读取访问过程中被读取，对数值所有形式的写访问都无效。
- 使用永久修改功能时，对程序的读访问是有效的并可保持到下一触发点。

21 测试程序状态

通过显示程序状态（RLO，状态位）或为每条指令显示相应寄存器内容的方法对程序进行测试。可以在“Customize”对话框中定义在“LAD/FBD”画面中的显示范围。在“LAD/STL/FBD：Programming Blocks”窗口中，使用菜单命令**Options > Customize**打开这个对话框。

警告

在过程运行时测试程序，如功能或程序出错，会对人员或财产造成严重损害。

在开始测试功能之前，确认不会有危险情况出现。

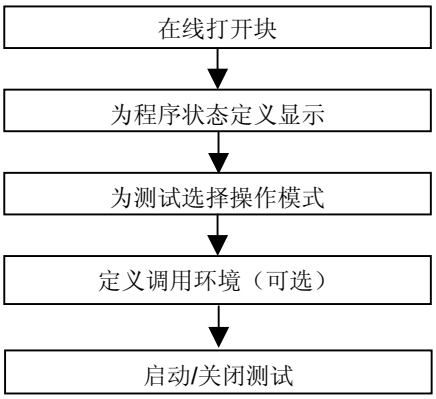
要求

显示程序状态，必须满足下列要求：

- 存储没有错误的程序，并且下载到CPU。
- CPU运行并且用户程序执行。

监视程序状态的基本步骤

建议不要调用整个程序进行调试，而应一个块一个块地调用并单独调试。应该从调用分层嵌套中最外层的块开始，例如，在OB1中调用它们，通过监视和修改变量功能生成测试的环境。



设置断点，并在单步模式下执行程序，必须设置为测试操作模式（见菜单命令**Debug > Operation**）。这些测试功能在过程操作模式下是不可能的。

21.1 程序状态显示

程序状态的显示是循环刷新的。它从所选择的程序段开始。

在LAD和FBD编辑器中的预设颜色代码

- 状态满足：绿色实线
- 状态不满足：兰色点线
- 状态未知：黑色实线

在“LAD/FBD”标签中，使用菜单命令**Options > Customize**，可改变线型及颜色的预置。

元素的状态

- 触点的状态是：
  - 如果该地址有“1”值则满足。
  - 如果该地址有“0”值则不满足。
  - 如果该地址的值不知道则为未知。
- 带有使能输出（ENO）的元素状态对应于以ENO输出值为地址的触点的状态。
- 带有Q输出的元素状态对应于该地址值的触点状态。
- 如果BR位在调用功能后被置位，则调用（CALL）的状态满足。
- 如果跳转被执行则跳转指令的状态满足，即意味着跳转条件满足。
- 带有使能输出（ENO）的元素，如果使能输出未被连接则该元素显示为黑色。

线的状态

- 线的状态如果未知或没有完全运行则是黑色的。
- 在能流开始处线的状态总是满足的（“1”）。
- 并行分支开始处线的状态总是满足的（“1”）。
- 如果一个元素和它前面的线的状态都满足则该元素后面的线的状态满足。
- 如果NOT指令前面的线的状态不满足（相反）则NOT指令后面的线状态满足。
- 在下列情况下，许多线交出点后面的线状态可以满足：
  - 与**之前**至少有一条线的状态被满足。
  - 分支前的线的状态满足。

参数状态

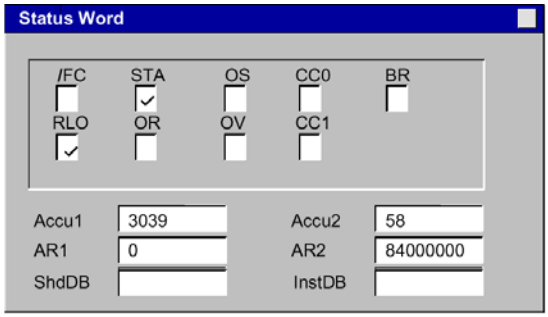
- **黑体类型**的参数值是当前值。
- 细体类型的参数值来自前一个扫描；该程序区在当前扫描循环中未被处理。

21.2 在单步模式/断点下进行测试应了解的信息

在单步模式下进行测试，可以做以下事情：

- 一条指令一条指令地执行程序（在单步模式下）
- 设置断点

“在单步模式下测试”的功能并不是在所有的可编程控制器上都能实现（参考相关的可编程控制器的资料）。



要求

- 必须对测试模式进行设置。在过程操作模式下单步测试模式是不可能的（见菜单命令Debug > Operation）。
- 只有使用STL编程时，才可以使用单步执行模式测试。使用LAD或FBD生成的块，必须用菜单命令View > STL改变为STL编程语言。
- 该块不能被保护。
- 块必须在线打开。
- 已打开的块不能在编辑器中进行修改。

断点的数量

断点的数量依据下列情况而变化：

- 已设置的断点数量
- 变量状态运行的数目
- 程序状态运行的数目

参考可编程控制器相关资料，查看是否支持单步执行模式。

在“Debug（调试）”菜单中，可以找到菜单命令用来设置、激活或删除断点，也可以用断点栏中的快捷键选择这些菜单命令，使用菜单命令View > Breakpoint Bar可以显示断点栏。

允许的测试功能

- 监视/修改变量

- 模板信息
- 操作模式

危险（Danger）

在HOLD（保持）模式下注意整个系统风险。

21.3 在 HOLD（保持）模式应该了解的信息

如果程序遇到断点，可编程控制器进入HOLD操作模式。

HOLD模式下LED的显示

- LED RUN 闪烁
- LED STOP 亮

在HOLD模式下程序的处理

- 在HOLD模式下不处理S7指令码，这意味着没有优先级被进一步处理。
- 所有定时器被冻结：
  - 不处理任何定时器单元
  - 所有监控时间停止
  - 时间控制的基本时钟被停止
- 实时时钟继续运行
- 为安全原因，在HOLD模式下输出总是禁止的（“输出禁止”）。

HOLD模式下电源掉电后的性能

- 装有后备电池的可编程控制器在HOLD模式下，在电源故障并恢复时将进入STOP模式并保持在STOP模式。CPU不执行自动重启动(暖启动)。在STOP模式下可以决定过程如何继续（例如，设置/清除断点，执行一个手动再启动）。
- 没有后备电池的可编程控制器不具有“记忆”功能，所以当电源恢复后执行一个自动暖启动，而不考虑以前的操作模式。

21.4 数据块的状态

版本5以上的STEP 7都可以在数据视窗中在线监控数据块。监控功能可以由在线数据块也可以由离线数据块激活。在这两种情况下，都可以显示可编程控制器中数据块内容。

在启动程序状态之前，不能修改数据块。如果在线数据和离线数据块之间有结构差异（声明），可以根据需要将离线数据块直接下载到可编程控制器中。

数据块必须位于“data view（数据视窗）”中，以使在线数值可以显示在“Actual Value（实际数值）”栏中。只有显示窗口中可以看到的数据块部分才能刷新。在状态激活时，不能切换为声明视窗。

在更新过程中，在状态栏中将显示一个绿框，并显示操作模式。

数值分别以各自的数据类型显示；其格式不能修改。

在结束程序状态后，“Actual Value（实际数值）”栏将显示程序状态之前的有效内容。不能将刷新的在线数值上传至离线数据块。

更新数据类型：

在共享DB和实例数据块的所有声明（in/out/in-out/stat）中，可以更新所有基本数据类型。

有些数据类型不能更新。当程序状态激活时，在“Actual Value（实际数值）”栏中包含没有更新的数据区域将显示为灰色背景。

- 复合数据类型DATE_AND_TIME和STRING不能更新。
- 在复合数据类型ARRAY、STRUCT、UDT、FB和SFB中，只有那些基本数据类型元素才能被更新。
- 在一个背景数据块的INOUT声明中，只显示复合数据类型指针，不显示数据类型的元素本身。指针不能更新。
- 参数类型不能更新。

21.5 为程序状态设置显示

可以设置程序状态的显示语言为STL、FBD或LAD。

按如下步骤进行显示设置：

1．选择菜单命令**Options > Customize**。

2．在“Customize”对话框中，选择STL或LAD/FBD。

3．为程序测试选择所需的选项，在状态域中可显示下面对象：

激　活	显　示
状态位	状态位；状态字的第2位
RLO	状态字的第1位；显示逻辑运算或算术比较的结果
标准状态	累加器1的内容
地址寄存器1/2	各自带有间接寻址的指针地址
Accu2	累加器2的内容
DB寄存器1/2	数据块寄存器的内容，第1个和/或第2个打开的数据块的内容

激　活	显　示
间接	间接存储器参考；指针参考(地址)，无地址内容参考；只能对存储器间接寻址，而不能对寄存器间接寻址。 如果在语句中有相应的指令，则显示定时器字或计数器字的内容。
状态字	状态字的所有的状态位

21.6　设置测试模式

步骤：

1. 使用菜单名令Debug > Operation显示设置的测试环境。
2. 选择所需的运行模式。可以选择测试运行模式或处理运行模式。

运行模式	说　明
测试运行	可以不受限制地使用所有测试功能。 会明显增加CPU扫描循环时间。
处理运行	测试功能程序状态是受限制的，以保证测试占用扫描循环最小的负荷。 • 不允许条件调用 • 编写的循环状态显示在返回点被禁止 • 不允许HOLD测试功能和单步执行

注意

当通过向CPU赋值参数来设置运行模式时，只能通过更改参数来改变运行模式，另外也可以在显示的对话框中改变运行模式。

22 使用模拟软件（可选软件包）进行测试

22.1 使用模拟程序 S7 PLCSIM（可选软件包）进行测试

可以使用可选软件包PLC Simulation（模拟PLC），在计算机或编程器（如PG740）上模拟可编程控制器的运行并测试程序。由于该模拟功能可完全由STEP 7软件实现，不需要任何S7硬件（CPU或信号模板）。使用模拟的S7 CPU，可以对S7-300和S7-400 CPU作程序测试和故障诊断。

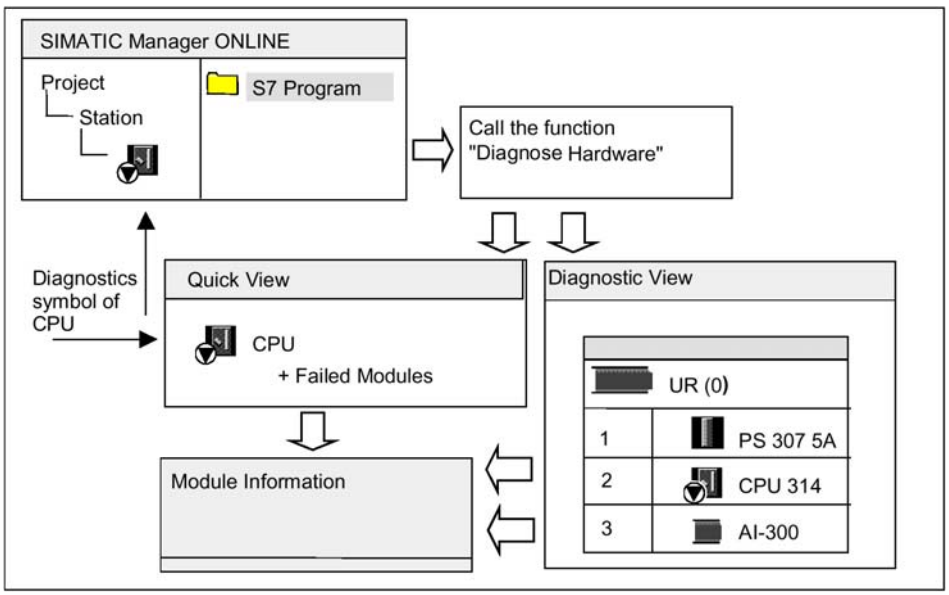
模拟软件为监视和修改各种用户程序中使用的参数提供了一个简单的用户界面（如，切换输入开关接通和断开）。程序由模拟的CPU处理时，还可以在STEP 7中使用各种应用程序。例如，可以用变量表监视和修改变量。

23 诊断

23.1 硬件诊断和故障排除

可以通过观察诊断符号来判断一个模板是否有诊断信息。诊断符号可显示相应模板的状态，对于CPU，还可显示操作模式。

诊断符号可显示在项目在线窗口中，以及当调用“Diagnose Hardware（诊断硬件）”功能时显示在快速视窗（缺省设置）或诊断视窗中。更详细的诊断信息显示在“Module Information（模板信息）”应用程序中，可以通过双击快速视窗或诊断视窗中的诊断符号起动该应用程序。



如何进行故障定位

1. 用菜单命令**View > Online**，打开项目的在线窗口。
2. 打开所有的站，以便看到其中组态的可编程模板。
3. 查看哪个CPU显示指示错误或故障诊断符号。可以使用F1键调用对诊断符号进行解释的在线帮助。
4. 选择要检查的站。
5. 选择菜单命令**PLC > Diagnostics/Settings > Module Information**，显示该站中CPU的模板信息。
6. 选择菜单命令**PLC > Diagnostics/Settings > Diagnose Hardware**，显示CPU的“quick view”（快速视窗）”及本站中有故障的模板。快速视窗的显示被设作缺省

设置（菜单命令Option > Customize，“View”标签）。

7. 在快速视窗中选择故障模板。
8. 点击“Module Information（模板信息）”按钮，以获得该模板的诊断信息。
9. 点击快速视窗中的“Open Station Online”按钮可显示诊断视窗。诊断视窗中包含了该站中按插槽顺序排列的所有模板。
10. 双击诊断视窗中的模板以显示其模板信息。用这种方式，还可以得到那些没有故障因而没有显示在快速视窗中的模板信息。

没有必要执行上述全部的步骤，当得到所需的诊断信息后就可以停止诊断工作。

23.2 在线视窗中的诊断符号

诊断符号显示在项目在线窗口和硬件组态界面中在线窗口。

诊断符号便于检查故障。只需看一眼模块符号，就知道有没有诊断信息。如果没有出现故障，则在模板类型上不显示诊断符号。

如果模板有诊断信息，则会在模板的符号上增加一个诊断符号，或显示的模板符号对比度降低。

模板的诊断符号(例：FM / CPU)

符　号	模　式
	预置组态与实际组态不匹配：被组态的模板不存在，或者插入了不同类型的模板
	故障：模板出现故障 可能的原因：诊断中断，I/O访问错误，或检查到故障LED
	不能进行诊断。原因：无在线连接或该模块不支持模板诊断功能（如：电源或子模板）

操作模式的诊断符号(例如：CPU)

符　号	模　式
	起动（STARTUP）
	停机（STOP）
	停机（STOP） 在多CPU操作模式下由另一个CPU触发的停机

符　号	模　式
	运行（RUN）
	保持（HOLD）

强制的诊断符号

符　号	含　义
	在该模板上有变量被强制，即在模板的用户程序中有变量被赋予一个固定值，该数据值不能被程序改变。 强制符号还可以与其它符号组合在一起显示（这里是与运行（RUN）模式符号一起显示）。

更新诊断符号的显示

必须激活相应的窗口。

- 按F5，或
- 在窗口中选择菜单命令**View > Update**。

23.3　硬件诊断：快速视窗

23.3.1　访问快速视窗功能

快速视窗可以快速进行“Diagnosing Hardware（硬件诊断）”，但其显示的信息没有硬件组态中诊断视窗所显示的信息详细。当调用“Diagnos Hardware”功能时，快速视窗作为缺省设置显示。

显示快速视窗

在SIMATIC Manager中使用菜单命令**PLC > Diagnostics/Settings > Diagnose Hardware**调用该功能。

可以在下列情况下使用菜单命令：

- 如果在一个在线项目窗口中选中了一个模板或一个S7/M7程序。
- 如果在“Accessible Nodes（可访问站点）”窗口中选中了一个站（“MPI=…”）并且该选项是一个CPU。

从显示的组态列表中，可以选择需要显示其模板信息的模板。

23.3.2 快速视窗中的信息功能

下列信息会显示在快速视窗中：

- 在线连接到CPU的数据
- CPU的诊断符号
- 被CPU检测出故障的诊断符号（例如，诊断中断、I/O访问错误）
- 模板类型和地址（机架、槽、带有站号的DP主站系统）

快速视窗中的其它诊断选项

- **显示模板**信息

通过点击“Module Information（模板信息）”按钮调用该对话框。对话框依据所选模板的诊断能力显示详细的诊断信息。可以通过CPU的诊断信息显示诊断缓冲区中的各个条目。

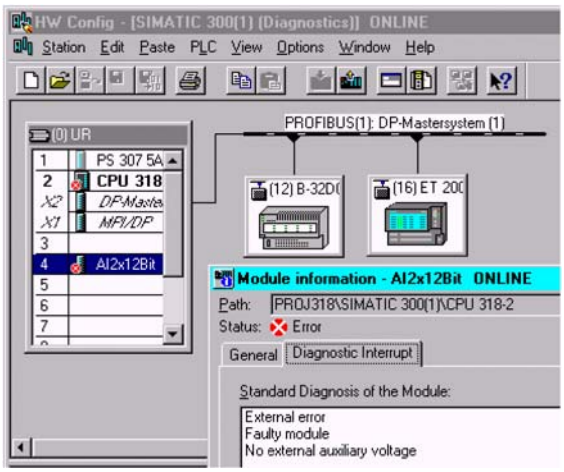
- **显示诊断视窗**

使用“Open Station Online（打开在线站点）”按钮可以打开一个对话框，与快速视窗相比，该对话框中包含整个站的图形概述以及组态信息。它主要显示“CPU/Faulty Modules（CPU/故障模板）”列表中被高亮选中的模板信息。

23.4 硬件诊断：诊断视窗

23.4.1 调用诊断视窗

使用这种方法可以为机架上的所有模板打开“Module Information”对话框。诊断视窗（组态列表）显示在机架层中站的实际结构，以及DP站及其模板。



注意

- 如果离线的组态表已经打开，可以使用菜单命令**Station > Open Online**，打开组态列表的在线视窗。
- 根据模板的诊断能力，在“Module Information”对话框中可以显示不同的信息标签。
- 在“Accessible Nodes”窗口中，只能显示有站地址（MPI、PROFIBUS、以太网地址）的模板。

从SIMATIC Manager的在线项目视窗中调用

1. 在SIMATIC Manager的项目视窗中，使用菜单命令**View > Online**建立与PLC的在线连接。
2. 选择一个站并双击打开。
3. 然后打开其中的“Hardware”对象。打开诊断视窗。

可以选择一个模板，使用菜单命令**PLC > Diagnostics/Settings > Module Information**调用其模板信息。

从SIMATIC Manager的离线项目视窗中调用

执行以下步骤：

1. 在SIMATIC Manager的项目视窗中选择一个站并双击打开。
2. 然后打开其中的“Hardware”对象，打开组态列表。
3. 选择菜单命令**Station > Open Online**。
4. 模板（如，CPU）所决定的站点组态及硬件组态的诊断视窗就打开了。模板的状态由符号来指示。各种符号的含义可参考在线帮助。故障模板以及已经组态但找不到的模板分别列在不同的对话框中。在这个对话框中可以直接去找选中的模板（“Go To”键）。
5. 双击模板的符号，带有标签（依模板类型而不同）的对话框可以给出有关模板状态的详细分析。

从SIMATIC Manager的“Accessible Nodes”窗口中调用

执行以下步骤：

1. 使用菜单命令**PLC > Display Accessible Nodes**，在SIMATIC Manager中打开“Accessible Nodes”窗口。
2. 在“Accessible Node”窗口选择一个站。
3. 选择菜单命令**PLC > Diagnostics/Settings > Diagnose Hardware**。

注意

在“Accessible Nodes”窗口中，只能显示有站地址（MPI、PROFIBUS、以太网地址）的模板。

23.4.2 诊断视窗中的信息功能

与快速视窗相比，诊断视窗显示已在线连接的整个站的组态信息。包括：

- 机架配置
- 所有组态模板的诊断符号，从这里可以读出每个模板的状态以及CPU模板的操作模式。
- 模板类型、订货号和地址细节以及组态注释。

诊断视窗中其他的诊断选项

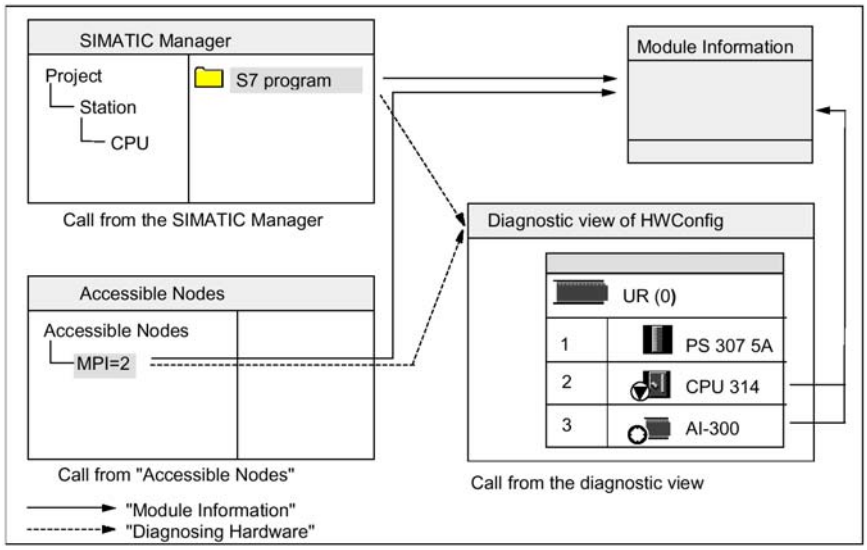
双击一个模板，可以显示该模板的操作模式。

23.5 模板信息

23.5.1 显示模板信息的选项

可以从不同的开始点显示“Module Information（模板信息）”对话框。下列步骤是调用模块信息经常使用的方法：

- 在SIMATIC Manager中从项目的“在线”或“离线”视窗调用。
- 在SIMATIC Manager的“Accessible Nodes（可访问节点）”窗口中调用。
- 在硬件组态的诊断视窗中调用。



为了要显示**模板及其站地址的**状态信息，需要对可编程控制器进行在线连接。可通过项目的在线视窗或通过“可访问站点”窗口建立连接。

23.5.2 模板信息功能

在“Module Information”对话框中，不同的标签对应不同的模板信息功能。在激活状态下显示时，只显示那些与所选模板相关的标签。

功能/标签	信　息	用　途
概述	所选模板的标识数据；比如，订货号、版本号、状态、机架中的插槽	可将所插模板的在线信息与所组态模板的数据进行比较
诊断缓冲区	诊断缓冲区中事件总览以及所选中事件的详细信息	查找引起CPU进入STOP模式的原因，并在选中的模板上评估导致绍停机原因的事件 使用诊断缓冲区还可在以后对系统错误进行分析，查找停机原因并对出现的每个诊断事件进行追踪和分类
诊断中断	所选模板的诊断数据	评估模板故障的原因
DP从站诊断	所选DP从站（参照EN50170）的诊断数据	评估DP从站中的故障原因
存储器	存储能力。所选的CPU或M7功能模板的工作存储器和装载存储器以及具有保持功能的存储器的当前使用情况	在新块或扩展块传送到CPU之前，可检查CPU/功能模板的装载存储器中是否有足够的空间，或者需要压缩存储器内容
扫描循环时间	所选CPU或M7功能模板最长、最短和最近一次的循环扫描时间	用于检查所组态的最小循环时间、最大循环时间和当前循环时间
时间系统	当前时间、操作小时数以及同步时钟的信息（同步周期）	可显示并设置模板的时间和日期，检查时间同步
性能数据	所选模板（CPU/FM）的地址区和可使用的块	在生成用户程序之前或之中检查CPU是否满足执行用户程序的要求。例如，装载存储区的大小或过程映像的大小
块(可在“Performance Data”标签中打开)	显示所选模板所有可用的块类型，包括OB、SFB、SFC列表，可将这些块用于该模板	检查用户程序中可包含或调用哪些可在所选CPU上运行的标准块
通讯	传送速率，通讯连接概述，通讯负载以及所选模板在通讯总线上最大的信息容量	用来决定CPU或M7 FM的连接数量和种类，以及正在使用的数量
堆栈	**堆栈**标签：只能在STOP模式或HOLD模式下调用 显示所选模板的B（块）堆栈。然后还可以显示I（中断）堆栈、L（局域）堆栈以及嵌套堆栈。可以跳转到中断块的故障点。	可判明引起停机的原因并对块进行修改

显示附加信息

对每个标签可显示以下信息：

- 到所选模板的在线路经。
- 相应CPU的操作模式（例如：运行、停机）。
- 所选模板的状态（例如：出错、OK）。

- 所选模板的操作模式（例如：运行、停机），如果模板具有操作模式（如CP342-5）。

如果从“Accessible Nodes”窗口中打开非CPU模板的模板信息中，则不能显示CPU本身的操作模式和所选模板的状态。

同时显示多个模板

可以同时显示多个模板的模板信息。要做到这一点，必须改变各个模板的背景，选择另一个模板，然后调用模板信息，“Module Information（模板信息）”对话框就会显示出来。每个模板只能打开一个对话框。

刷新模板的信息显示

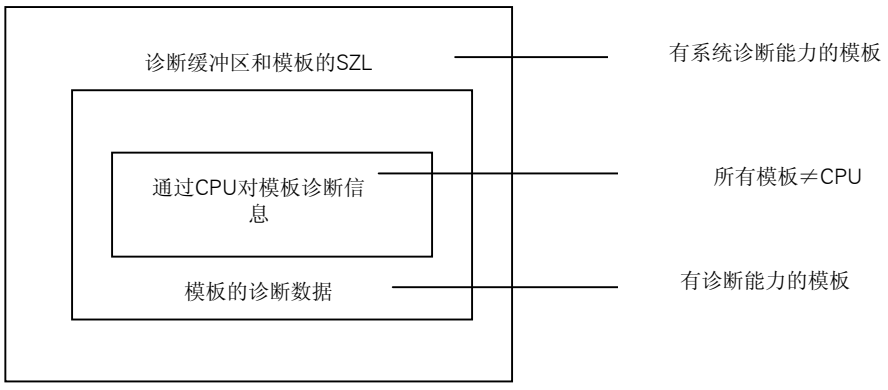
在“Module Information”对话框中，每次切换标签时，都可以重新读取模板数据。但是，当一页显示之后，其内容不再刷新。如果点击“Updete（刷新）”按钮，可以在不改变标签的情况下从模板中读取数据。

23.5.3 模板类型所决定的模板信息范围

可以评估和显示的信息范围取决于：

- 所选模板，以及
- 从哪个视窗调用模板信息
  当从组态列表的在线视窗或项目窗口调用时可得到全范围的信息。
  从“Accessible Nodes（可访问站点）”窗口调用只能得到有限范围内的信息。

根据信息的范围，模板可分为几类：“有系统诊断能力”、“有诊断能力”或“无诊断能力”。下图所示这些类别：



- 有系统诊断能力的模板，如模板FM 351和FM 354
- 模拟信号模板大多具有诊断能力
- 数字信号模板大多没有诊断能力

显示的标签

下表显示了“模板信息”对话框中每种类型的模板都显示哪些特性标签。

标签	CPU或M7 FM	有系统诊断能力的模板	有诊断能力的模板	无诊断能力的模板	DP从站
常规	有	有	有	有	有
诊断缓冲区	有	有	---	---	---
诊断中断	---	有	有	---	有
存储器	有	---	---	---	---
扫描循环时间	有	---	---	---	---
系统时间	有	---	---	---	---
性能数据	有	---	---	---	---
堆栈	有	---	---	---	---
通讯	有	---	---	---	---
DP从站诊断	---	---	---	---	有
H状态[1)]	有	---	---	---	---
1） 只有H系统的CPU					

除了标签特性页的信息之外，有操作模式的模板还会显示操作模式。当从在线的组态列表中打开对话框时，会从CPU角度来显示模板的状态（例如，OK、故障、模板不存在）。

23.5.4 显示Y-Link后面链接的PA现场设备和DP从站的模板状态

对于STEP 7 V5.1 SP3，在DP/PA链接后(IM 157)可以评估DP从站和PA现场设备的模板状态。

它将影响下面的组态：

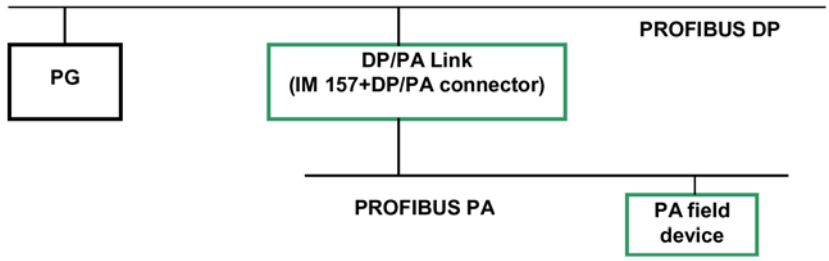
- 带DP/PA连接器的IM 157，用于连接PROFIBUS-PA
- IM 157作为一个冗余的接口模板，用于连接非冗余PROFIBUS- DP(“Y-link”)

在该组态中，编程器通过DP/PA链接连接到相同的PROFIBUS子网。

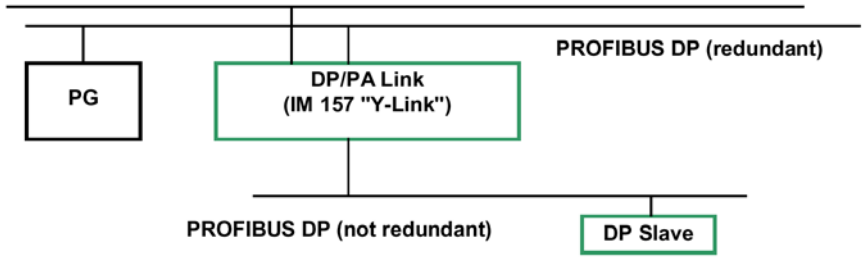
此外，还有一个选项，通过该选项，PG连接到一个工业以太网，通过S7-400站路由到一个连接的PROFIBUS子网上。

下图显示了该项设置的前提条件。

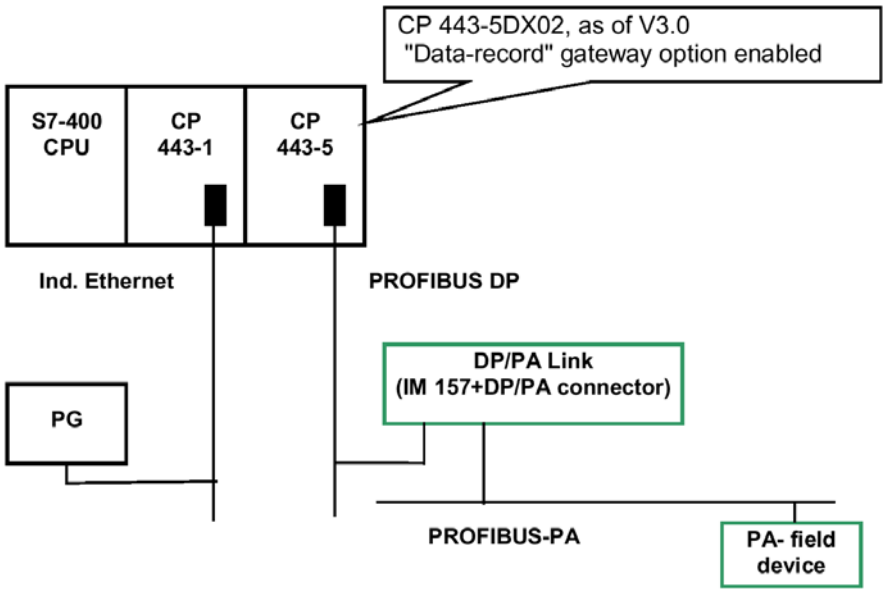
带DP/PA连接器的IM 157，用于连接PROFIBUS-PA



作为Y-link的IM 157



在工业以太网中的PG



23.6 在停机模式下进行诊断

23.6.1 判定停机原因的基本步骤

要判断CPU为什么进入“停机”模式，可按如下步骤进行：

1. 选择已进入停机的CPU。
2. 选择菜单命令**PLC > Diagnostics/Settings > Module Information**。
3. 选择“Diagnostic Buffer（诊断缓冲区）”标签。
4. 可以从诊断缓冲区中最后一项判定停机的原因。

如果出现编程错误：

1. 进入条目“STOP because programming error OB not loaded”意味着，CPU已查到一个编程错误，试图启动这个（不存在的）OB块去处理编程错误。前一条指出了实际的编程错误。
2. 选择与编程错误相关的信息。
3. 点击“Open Block”按钮。
4. 选择“Stacks（堆栈）”标签。

23.6.2 停机模式下堆栈的内容

通过评估诊断缓冲区和堆栈中的内容，你可以判定用户程序执行过程中引起故障的原因。

例如，如果由于编程错误或停机指令使CPU进入停机状态。则可用“I Stack（中断堆栈）”“L Stacks（局域堆栈）”和“Nesting Stack（嵌套堆栈）”按钮显示这些堆栈中的内容。堆栈内容给出了哪个块中的哪条指令引起CPU进入停机的信息。

B堆栈内容

B堆栈或称作块堆栈，列出了所有停机前已经被调用但未完全处理的块。

I堆栈内容

当点击“I stack（中断堆栈）”的按钮时，将显示中断点的数据。I堆栈，或称中断堆栈包含着中断时有效的数据或状态，例如：

- 累加器内容和寄存器内容
- 打开的数据块及其大小
- 状态字的内容
- 优先级（嵌套层次）

- 中断的块
- 中断后程序将继续处理的块

L堆栈内容

对于B堆栈中所列出的每个块，都可以通过选择该块并点击“L Stack（局域堆栈）”按钮显示相应的局域数据。

L堆栈，或称作局域数据堆栈，包含中断时用户程序正在处理的块的局域数据。

解释和评估所显示的局域数据需要更深入的系统知识。显示的第一部分的数据相应于块中的临时变量。

嵌套堆栈内容

当点击“Nesting Stack（嵌套堆栈）”按钮时，显示嵌套堆栈在断点处的内容。

嵌套堆栈是逻辑操作A(、AN(、O(、ON(、X(和XN(使用的存储区。

只有当中断时有括号操作仍在打开，该按钮才激活。

23.7 检查扫描循环时间以避免时间错误

在模板信息的“Scan Cycle Time（循环扫描时间）”标签中可以给出有关用户程序扫描循环时间的信息。

如果最长的循环时间接近所组态的最大扫描循环监控时间，就会存在由于循环时间的波动引起时间错误的危险。可通过延长用户程序的最大循环时间（监控时间）来避免这种危险。

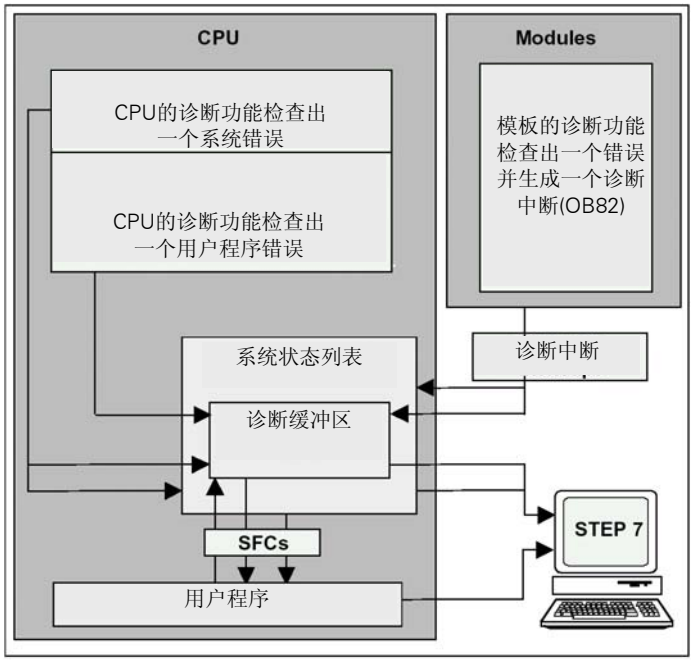
如果循环长度短于所组态的最小循环时间，则CPU/FM会自动将循环周期延长至所组态的最小循环时间。对于CPU，在这个延长时间内将处理背景OB（OB90）（如果它已下装）。

设置扫描循环时间

当组态硬件时，可以设置最大和最小循环时间。为此，双击组态列表离线视窗中的CPU/FM定义它的特性。可以在“Cycle/Clock Memory（循环周期/时钟存储器）”标签中输入适当的值。

23.8 诊断信息流向

下图所示为在SIMATIC S7中诊断信息的流向



显示诊断信息

可以在用户程序中使用SFC 51 RDSYSST读出诊断内容，或者用STEP 7以无格式文本显示诊断信息。它们可以提供以下信息：

- 错误出现的位置和时间。
- 该输入项所属的诊断事件的类型（用户定义的诊断事件、同步/异步错误、操作模式改变）。

生成过程控制组消息

CPU向诊断缓存区中写入标准诊断事件和扩展诊断事件。如果满足下列条件，它还为标准诊断事件生成一个过程控制组消息：

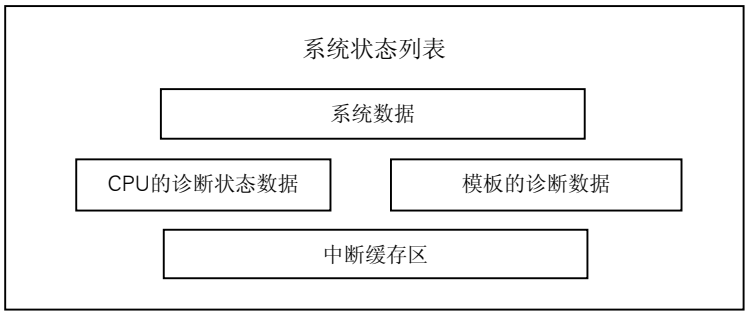
- 已指定将在STEP 7中生成过程控制消息。
- 至少有一个过程控制消息的显示单元已在CPU上登录。
- 一个过程控制组消息生成的条件是当前没有相应级别的过程控制组消息（有七个级别）。
- 每个级别可生成一个过程控制组消息。

23.8.1 系统状态列表SSL

系统状态列表（SSL）描述可编程控制器的当前状态。它提供了组态、当前参数赋值、当前CPU的状态和次序以及所属模板的信息。

系统状态列表中的数据只能读取不能修改。它是一个实际的列表，只当有请求时才会生成。

使用系统状态列表能够显示的信息可分为四个区域。



读取系统状态列表

有两种读取系统状态列表的方法：

- 间接通过编程器中STEP 7的菜单命令（例如，存储器组态、静态CPU数据、诊断缓冲、状态显示）。
- 直接在用户程序中通过系统功能SFC51 RDSYSST，输入所需部分系统状态列表的编号（参见块的帮助）。

系统状态列表的系统数据

系统数据是CPU内部数据或已分配的性能数据。下表展示可以显示的信息主题（部分系统状态列表）：

标　题	信　息
模板标识	模板的订货号、类型标识及版本
CPU特性	CPU的系统时间、系统特性（如：多CPU）以及语言描述
存储区	模板的存储器组态（工作存储器的大小）
系统存储区	模板的系统存储器（例如：存储位、定时器、计数器的数量、存储器的类型）
块类型	模板中存在哪些块（OB、DB、SDB、FC、FB），某一类型块的最大数量，以及某一类块的大小
中断及错误的赋值	中断/错误与OB之间的赋值关系
中断状态	中断处理/生成中断的当前状态
优先级的状态	由参数设定哪个OB块被执行、哪个优先级被禁止
运行模式和模式转换	可选择哪种运行模式，最后一次运行模式的变化，当前的运行模式

CPU中的诊断状态数据

诊断状态数据描述了由系统诊断所监视的组件当前状态。下表展示可以显示的信息主题（部分系统状态列表）：

标　题	信　息
通讯状态数据	当前在系统中设置的所有通讯功能
诊断模板	在CPU中登录了的有诊断能力的模板
OB的启动信息列表	有关CPU中OB的启动信息
启动事件列表	OB的启动事件及优先级
模板状态信息	所有已插入的、有故障的或生成硬件中断的模块的状态信息

模板的诊断数据

除CPU之外，其它模板也具有诊断能力（SM、CP、FM），它们的数据被存入系统状态列表中。下表所示为能够显示的信息主题（部分系统状态列表）：

标　题	信　息
模板诊断信息	模板起始地址、内部/外部故障、通道故障、参数错误（4字节）
模板诊断数据	某个特定模板的所有诊断数据

23.8.2　传送自己生成的诊断报文

通过系统功能SFC 52 WRUSMSG可以扩展SIMATIC S7的标准系统诊断：

- 在诊断缓存区中输入自己生成的诊断信息（例如，有关用户程序的执行信息）。
- 传送用户定义的诊断消息到已登录的站点（如PG、OP或TD监视设备）。

用户定义的诊断事件

诊断事件可以被分为事件级别1至F。用户定义的诊断事件属于事件级别8至B。可按如下所示分为两组：

- 事件级别8和9包括那些有固定编号和预定文本的消息，可根据编号调用。
- 事件级别A和B包括那些可由用户分配一个号码（A000至A0FF，B000至B0FF）和文本的消息。

向站点传送诊断消息

除了可以在诊断缓存区存入用户定义的条目，还可以用SFC 52 WRUSMSG将用户定义的诊断消息传送到已登录的显示设备上。当调用SFC 52，SEND=1时，诊断消息被写入发送缓冲区并自动发送到在CPU上登录的站点。

如果无法传送消息（比如，因为没有登录的显示设备或由于传送缓冲区已满），则用户定义的诊断事件仍然输入到诊断缓存区中。

生成一个带确认的消息

如果确认了一个用户定义的诊断事件并且想要记录这个确认，可按如下进行：

- 当事件到来时，事件状态将1写入一个BOOL类型的变量，当事件离开时事件状态将0写到这个变量。
- 然后使用SFB33 ALARM监视这个变量。

23.8.3 诊断功能

系统诊断将对PLC中发生的错误进行检测、评估和报告。为此，每个CPU和每个有系统诊断能力的模板（如FM354）都有一个诊断缓冲区，在这个缓冲区内诊断事件的详细信息按它们出现的顺序存入。

诊断事件

下列事项将作为诊断事件显示，例如：

- 模板上的内部和外部错误
- CPU中的系统错误
- 操作模式改变（如，从RUN到STOP）
- 用户程序中的错误
- 插/拔模板
- 用系统功能SFC52输入的用户报文

在存储器全清后诊断缓存区中的内容仍然保留。使用诊断缓冲区还可在后期对系统错误进行分析，查找停机原因并对出现的每个诊断事件分类。

获取诊断数据

不必为获取系统诊断的诊断数据而编写程序。这是一个可自动运行的标准特性。SIMATIC S7提供各种诊断功能。一些诊断功能是集成在CPU上的，另一些由模板提供（SM、CP和FM）。

显示故障

内部和外部模板故障会显示在模板的前面板上。在S7硬件手册中描述了有关LED的显示内容以及对如何对其评估的描述。在S7-300中，内部和外部故障作为一个组错误显示。

CPU能够识别系统错误和用户程序错误并将诊断信息存入到系统状态列表和诊断缓存区中。这些诊断信息可以由编程器读出。

具有诊断能力的信号模板和功能模板可检测内部和外部模板错误并产生一个诊断中断，可通过一个中断OB对其响应。

23.9 处理错误的程序

当CPU检测到程序处理过程中的错误（同步错误）和可编程控制器中的错误（异步错误）时，CPU会调用适当的组织块（OB）处理错误：

错　误	错误OB
I/O冗余错误	OB 70
CPU冗余错误	OB 72
时间错误	OB 80
电源错误	OB 81
诊断中断	OB 82
插/拔模板中断	OB 83
CPU硬件故障	OB 84
优先级错误	OB 85
机架故障或分布式I/O的站故障	OB 86
通讯错误	OB 87
编程错误	OB 121
I/O访问错误	OB 122

如果相应的OB不存在，CPU进入STOP模式(OB70、OB72、OB81、OB87除外)。此外，可在OB中调用指令以便对这种错误作出响应。这意味着可以减小或根除错误影响。

基本步骤

生成并打开OB

1. 显示CPU的模板信息。
2. 选择“Performance Data（性能数据）”标签。
3. 根据所显示的列表，确定需要编程的OB是否能在该CPU上执行。
4. 在程序的“Block（软件块）”文件夹中插入并打开该OB块。
5. 输入故障处理程序。
6. 将OB下载到可编程控制器。

处理故障的编程措施

1. 评估OB块的局部数据以判断故障的确切原因。局部数据中的变量OB8xFLTID和OB12xSWFLT中包含错误代码。其含义描述在《系统和标准功能参考手册》里。
2. 转向响应故障的程序段。

在标头为“用SFC51 （RDSYSST）作模板诊断的示例”的关于系统和标准功能的在线帮助中，能够找到处理诊断中断的示例。

在相应的块帮助中可以找有关OB、SFB、SFC的更详细的信息。

23.9.1 评估输出参数RET_VAL

系统功能用输出参数RET_VAL（返回值）指示CPU是否正确地执行了SFC功能。

返回值中的错误信息

返回值是整数类型（INT）。整数的符号位指示该整数是正还是负。返回值与数值“0”之间的关系用以指示在功能执行过程中是否有错误出现（见下表）：

- 如果功能执行时出现错误，返回值小于“0”。整数的符号位是“1”。
- 如果功能执行时没有错误，返回值大于等于“0”。整数的符号位是“0”。

由CPU处理SFC	返回值	整数的符号
错误出现	小于“0”	负（符号位是“1”）
没有错误	大于等于“0”	正（符号位是“0”）

对故障信息的反应

如果在SFC执行时出现错误，SFC在返回值（RET_VAL）中提供一个错误代码。

作以下区分：

- 所有SFC都可以输出的通用错误代码
- 依据SFC的特定功能而输出的特定错误代码

传送功能值

有些SFC还用输出参数RET_VAL来传送功能值，例如SFC64 TIMETCK用RET_VAL传送它已读出的系统时间。

在SFB/SFC帮助中可以找到一些有关输出程序SFB/SFC更详细的信息。

23.9.2 当检测到错误时故障OB的响应

可检测的错误

系统程序可以检测以下错误：

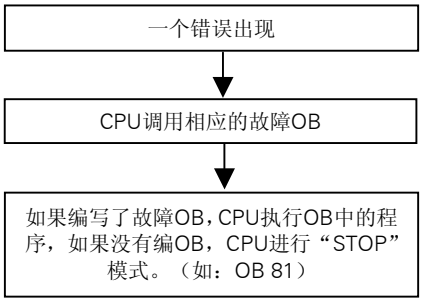
- 不正确的CPU功能
- 系统程序执行中的错误
- 用户程序中的错误
- I/O中的错误

根据错误类型的不同，CPU被设定为STOP模式或调用一个故障OB。

编程响应

可以编写程序响应不同类型的错误，并决定CPU的反应方式。为某个特定的错误而编制的

程序可以保存在一个故障OB中。如果故障OB块被调用，程序就会被执行。



故障OB

按如下所示区别同步错误和异步错误：

- 同步错误可以分配给一个MC7指令（例如，对一个已移走的信号模板使用装载指令）。
- 异步错误可分配给一个优先级或给整个可编程控制器（例如循环超时）。

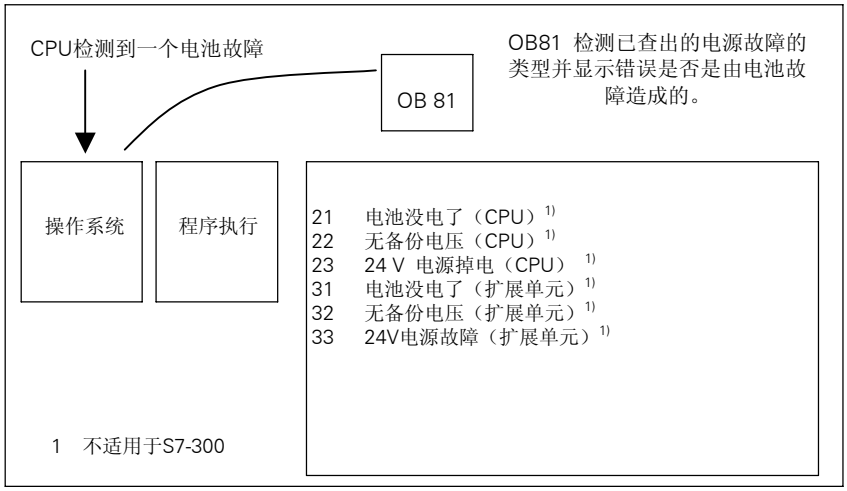
下表所示为可能出现的错误类型。参考《S7-300可编程控制器，硬件及安装手册》或《S7-400，M7-400可编程控制器，硬件及安装手册》，可找到有关CPU是否支持这些特定的OB的信息。

错误级别	错误类型	OB	优先级
冗余	I/O冗余错误(只在H CPU中)	OB 70	25
	CPU冗余错误(只在H CPU中)	OB 72	28
异步的故障	时间错误	OB 80	26
	电源错误	OB 81	（或者28如果故障OB在启动程序中调用了）
	诊断中断	OB 82	启动程序
	插/拔模板中断	OB 83	
	CPU硬件故障	OB 84	
	程序顺序错误	OB 85	
	机架故障	OB 86	
	通讯错误	OB 87	
同步	编程错误	OB 121	引起错误的OB的优先级
	I/O访问错误	OB 122	

故障OB81使用举例

使用错误OB的局域数据（启动信息），可以判断已出现的错误的类型。

例如，如果CPU检测到一个电池故障，操作系统则调用OB81（见图）。



可以编写一个程序来判定由OB81的调用而触发的事件代码。也可以编写程序作出响应，如，激活与操作站上的指示灯相连的输出。

故障OB81的局域数据

下表所示为在OB81的临时变量声明表中进行声明的临时变量（启动信息）。

符号“Battery error（BOOL）”必须被标识作为输出（例如，Q4.0），这样程序的其它部分就能够访问这些数据了。

声明	名　称	类型	描　述
TEMP	OB81EVCLASS	BYTE	错误级别/错误标识39xx
TEMP	OB81FLTID	BYTE	错误代码： b#16#21 = CPU中至少有一个备份电池没电了1) b#16#22 = 在CPU中无备份电压 b#16#23 = CPU中24-V电源掉电1) b#16#31 = 至少有一个扩展机架上的一个备份电池没电了1) b#16#32 = 在一个扩展机架上没有备份电压1) b#16#33 = 一个扩展机架24V电源掉电1)
TEMP	OB81PRIORITY	BYTE	优先级=26/28
TEMP	OB81OBNUMBR	BYTE	81 = OB81
TEMP	OB81RESERVED1	BYTE	保留
TEMP	OB81RESERVED2	BYTE	保留
TEMP	OB81MDLADDR	INT	保留

声明	名　称	类型	描　述
TEMP	OB81RESERVED3	BYTE	只与错误代码B#16#31，B#16#32，B#16#33相关
TEMP	OB81RESERVED4	BYTE	
TEMP	OB81RESERVED5	BYTE	
TEMP	OB81RESERVED6	BYTE	
TEMP	OB81DATETIME	DATEAND TIME	启动OB的日期和时间
1）=不适用于S7-300			

故障OB81程序举例

在STL程序举例中说明了如何在OB81中读错误代码。

程序结构如下：

- OB81中的错误代码（OB81 FLTID）被读出并与事件“电池没电”（B#16#3921）的数值作比较。
- 如果错误代码符合“（battery exhausted）电池没电”的代码，程序则跳到标号为Berr的指令并激活输出batteryerror。
- 如果错误代码与“（battery exhausted）电池没电”的代码不符，程序则将错误代码与“电池故障”的代码作比较。
- 如果错误代码符合“电池故障”代码，程序跳转到标号“Berr”，并激活输出“batteryerror”。否则该块结束。

```
    AWL                    说明
L     B#16#21              // 比较事件代码“电池没电”
                           // （B#16#21）
L     #OB81_FLT_ID         // 与OB81的错误代码。
==I                        // 如果相同（电池没电），
                           // 跳转到Berr。
JC Berr
L     B#16#22              // 比较事件代码“电池故障”
                           //（b#16#22）
==I                        // 与OB81的错误代码。
JC BF                      // 如果不相同，跳转到 Berr。
BEU                        // 无电池故障信息
Berr：L    B#16#39         // 与下一事件ID进行比较
L     #OB81_EV_CLASS       // OB81的错误代码。
==I                        // 如果电池故障或电池没电找到，
S     batteryerror         // 设置输出“battery error”。
                           //（符号表变量）
L     B#16#38              // 比较ID，以确定OB81
==I                        // 错误代码事件。
```

```
R       batteryerror                    // 复位输出“battery error”，当错误固定时。
```

在相应的块帮助中可以找有关OB、SFB、SFC的更详细的信息以及事件ID解释。

23.9.3 为故障诊断插入替代值

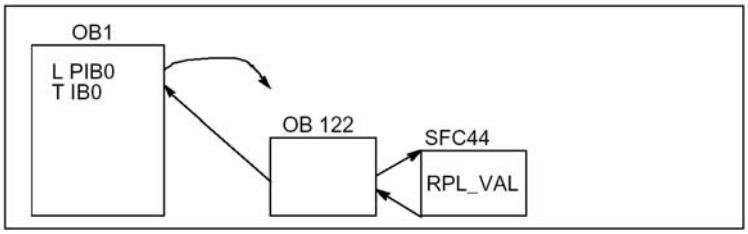
对于某种类型的故障（如，受断线影响的输入信号），可以为由于故障而无法使用的数值提供一个替代位。可用以下两种方法来提供替代值：

- 用STEP 7为可组态的输出模板分配替代值。无法得到赋值参数的输出模板用缺省替代值0。
- 用SFC44 RPLVAL，可以在故障OB中编写替代值（只适用于输入模板）。

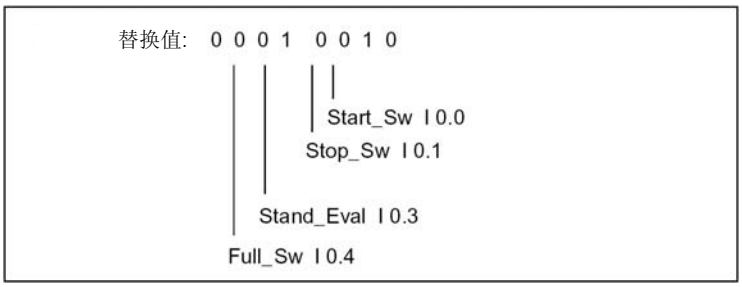
  对于所有引起同步错误的装载指令，可以在故障OB中为累加器内容指定一个替代值。

替代数值程序举例

在以下示例程序中，在SFC44 RPLVAL中有一个可用的替代值。下图说明了CPU是如何在检测到一个输入模板没有反应时调用OB122的。



在这个示例中，下图所示的替代值在程序中被输入，这样程序就可以用可行的数值继续操作。



如果一个输入模板有故障，执行指令L PIB0就会产生一个同步错误并启动OB122。作为标准，这个装载指令读得数值0。然而，用SFC44，可以为过程定义任何合适的值。SFC用指定的替代值替换累加器中的内容。

以下示例程序可写在OB122中。下表所示为在OB122的变量声明表中声明的临时变量（启动信息）。

声明	名称	类型	描述
TEMP	OB122EVCLASS	BYTE	错误级别/错误ID29xx
TEMP	OB122SWFLT	BYTE	错误代码： 16#42， 16#43， 16#44 1)， 16#45 1)
TEMP	OB122PRIORITY	BYTE	优先级=错误出现的OB的优先级
TEMP	OB122OBNUMBR	BYTE	122 = OB122
TEMP	OB122BLKTYPE	BYTE	错误出现的块的类型
TEMP	OB122MEMAREA	BYTE	存储区域和访问类型
TEMP	OB122MEMADDR	WORD	出错的存储器地址
TEMP	OB122BLKNUM	WORD	出错的块的号码
TEMP	OB122PRGADDR	WORD	出错指令的相对地址
TEMP	OB122DATETIME	DATEANDTIME	启动OB的日期和时间
TEMP	错误	INT	存储SFC44的错误代码
1）不适用于S7-300			

```
                STL                          说明
                L     B#16#2942              为应答读I/O时出现的时间错误，将OB122的事件码与事件码
                L     #OB122SWFLT            （B#16#2942）比较， 如果一样跳到“Aerr”。
                ==I                          为寻址错误（向一个不存在的模板作写操作）将OB122的事件代
                JC    Aerr                   码与事件代码（B#16#2943）作比较。如果一样跳到“STOP”。
                L     B#16#2943              标号“Aerr”：将DW#16#2912（二进制10010）传送到SFC44
                <> I                         （REPL_VAL）。SFC 44将该值装载到累加器1（替代由OB122
                JC Stop                      调用而触发的数值）。SFC的错误代码存储在#Error。

Aerr：          CALL ”REPL_VAL"
                VAL ：= DW#16#2912           将#Error与0比较（如果相同则OB122执行时没出错）。
                RETVAL ：= #Error            如果没有错结束块。
                L     #Erro
                L     0                      “Stop”标号：调用SFC46“STP”，
                ==I                          将CPU转为STOP模式。
                BEC

Stop：          CALL “STP"                   “Stop”标号：调用SFC46“STP”，
                                              将CPU转为STOP模式。
```

23.9.4 I/O冗余错误（OB70）

说明

对于H CPU的操作系统，如果PROFIBUS DP上出现冗余丢失（如，DP主站总线故障或DP从站接口模板故障），或者，如果带有转换I/O的DP从站变为激活的DP主站时，系统调用OB70。

在OB70中编程

必须使用STEP 7在用户程序中生成一个OB70。在生成的块中编写要在OB70中执行的程序，并作为用户程序的一部分下载到CPU。

例如，可以为如下目的使用OB70：

- 要评估OB70中的开始信息并判定哪个条件触发了I/O冗余丢失。
- 用SFC51RDSYSST判定系统的状态。（SZLID=B#16#71）。

如果出现I/O冗余错误且OB70没有编程，CPU不停机。

如果下载了OB70并且H系统不在冗余模式，则OB70在两个CPU中都被处理。H系统保持在冗余模式。

在相应的块帮助中可以找有关OB、SFB、SFC的更详细的信息。

23.9.5 CPU冗余错误（OB72）

说明

如果出现以下事件之一，H CPU的操作系统调用OB72：

- CPU冗余丢失
- 比较错误（如RAM、PIQ）
- 主从切断
- 同步错误
- SYNC子模块出错
- 刷新过程失败
- OB72被所有的处于RUN模式或起动事件后的STARTUP模式的CPU执行

在OB72中编程

必须使用STEP 7在用户程序中生成一个OB72。在生成的块中编写将由OB72执行的程序并将它作为用户程序的一部分下载到CPU。

例如，可以为以下目的使用OB72：

- 要评估OB72中的起动信息和判定哪个事件触发了CPU冗余的丢失。
- 用SFC51RDSYSST判定系统的状态。（SZLID=B#16#71）。

- 要为系统对CPU的冗余丢失作出特定的反应。

如果出现CPU冗余错误，并且OB72没有编程，CPU不会转为STOP模式。

在相应的块帮助中，可以找有关OB、SFB、SFC的更详细的信息。

23.9.6 时间错误（OB80）

说明

当有时间错误出现时CPU的操作系统调用OB80。时间错误包括以下，如：

- 超过最大循环时间
- 通过向前修改时间而跳过日时钟中断
- 处理优先级时延迟太多

在OB80中编程

必须使用STEP 7在用户程序中生成一个OB80。在生成的块中编写要在OB80中执行的程序并将它作为用户程序的一部分下载到CPU。

例如，可为以下目的使用OB80：

- 要评估OB80中的起动信息和判定哪个日时钟中断被跳过。
- 通过使用SFC29 CANTINT，可以取消已被跳过的日时钟中断而不再执行它，只有与新的时间相关的日时钟中断全被执行。

如果没有在OB80中取消跳过的日时钟中断，则第一个跳过的日时钟中断被执行，所有其它的被忽略。

如果没有编程OB80，当CPU检测到一个时间错误时转为STOP模式。

在相应的块帮助中可以找有关OB、SFB、SFC的更详细的信息。

23.9.7 电源故障（OB81）

说明

如果在CPU中或在扩展单元中有以下故障出现，CPU的操作系统调用OB81

- 24V电压
- 电池
- 完全备份

当问题消除时OB也会被调用（OB在事件到来和离开时被调用）。

在OB81中编程

必须使用STEP 7在用户程序中生成一个OB81。在生成的块中编写要在OB81中

执行的程序并将它作为用户程序的一部分下载到CPU。

例如，可以为如下目的编写OB81：

- 要评估OB81的起动信息以及判定出现哪个电源故障。
- 要查找有电源故障的机架的号码。
- 要激活操作站上的灯指示维护人员应该换电池了。

与其它的异步故障OB相反，如果没有编写OB81，CPU检测到电源故障时不会转为STOP模式。但是这个错误会存入诊断缓冲区并且前面板上相应的LED灯亮，指示错误。

在相应的块帮助中可以找到有关OB、SFB、SFC的更详细的信息。

23.9.8 诊断中断（OB82）

说明

对于一个有诊断能力的模板，如果使能了它的诊断中断，当它检测到错误，以及错误消除时CPU的操作系统会调用OB82。（该OB在事件到来和离去时都会被调用）。

在OB82中编程

必须使用STEP 7在用户程序中生成一个OB82。在生成的块中编写要在OB82中执行的程序，并将它作为用户程序的一部分下载到CPU中。

例如，可为以下目的使用OB82：

- 要评估OB82的启动信息。
- 要获得与已出现的错误有关的更确切的诊断信息。

当一个诊断中断被触发时，有问题的模板自动地在诊断中断OB的启动信息和诊断缓冲区中存入4个字节的诊断数据及其起始地址，提供了错误何时出现以及出现在哪个模板上的信息。

在OB82中编写合适的程序，可以进一步地评估模板的诊断数据（哪个通道出错，出现的是哪种错误）。使用SFC51 RDSYSST可以读出模板的诊断数据，用SFC 52WRUSRMSG可以将这些信息存入诊断缓冲区。也可以发送一个用户定义的诊断消息到监控设备。

如果没有编写OB82，当诊断中断被触发时CPU转为STOP模式。

在相应的块帮助中可以找有关OB、SFB、SFC的更详细的信息。

23.9.9 插/拔模块中断（OB83）

说明

S7-400 CPU以大约1秒的间隔监视着中央机架和扩展机架上现有的模板。

在上电后，CPU检测由STEP 7生成的组态列表中所列的所有模板是否都实际地插入了。如

果所有模板都有，这个实际组态被存储并用作对模板进行循环监控的参考值。在每一扫描循环周期内，新检测到的实际组态与原来的实际组态作比较。如果发现两个组态有差异，则发出插/拔模板中断信号，并且将信息存入诊断缓冲区和系统状态列表。在RUN模式下，启动插/拔模板中断OB。

注意

电源模板、CPU和IM不能在RUN模式下移开。

在移走和插入模板两个操作之间应至少相隔两秒，以便让CPU检测到移走的或插入的模板。

对新插入的模板进行参数赋值

如果一个模板在RUN模式下插入，CPU会检测新模板的类型与原来模板是否匹配。如果匹配，对该模板赋予参数。将缺省参数或用STEP 7分配的参数传送到模板中。

在OB83中编程

必须使用STEP 7在用户程序中生成一个OB83。在生成的块中编写要在OB83中执行的程序，并将它作为用户程序的一部分下载到CPU中。

例如，可以为以下目的使用OB83：

- 要评估OB83的启动信息。
- 用系统功能SFC55至59对新插入的模板赋值参数。

如果没有编写OB83，当一个插/拔模板中断出现时，CPU从RUN转为STOP。

在相应的块帮助中可以找到有关OB、SFB、SFC的更详细的信息。

23.9.10 CPU硬件故障（OB84）

说明

当CPU检测到连至MPI网络的接口故障、连至通讯总线的接口故障或连至分布式I/O网卡的接口故障时操作系统调用OB84；例如，在总线上检测到一个不正确的信号电平。故障消除时也会调用该OB块（事件到来和离开时都调用该OB）。

在OB84中编程

必须使用STEP 7在用户程序中生成一个OB84。在生成的块中编写要在OB84中执行的程序并将它作为用户程序的一部分下载到CPU。

可以为以下目的使用OB84：

- 要评估OB84的起动信息。
- 用系统功能SFC52 WRUSMSG发送消息至诊断缓存区。

如果OB84没有编程，当检测到CPU硬件错误时CPU转为STOP模式。

在相应的块帮助中可以找到有关OB、SFB、SFC的更详细的信息。

23.9.11 编程顺序错误（OB85）

说明

CPU的操作系统调用OB85：

- 当一个中断OB的启动事件存在，但该OB块由于没有下载到CPU而不能被执行时。
- 当访问一个系统功能块的背景数据块时出错。
- 当刷新过程映像区时出错（模板不存在或出故障）。

在OB85中编程

必须使用STEP 7在用户程序中生成一个OB85。在生成的块中编写要在OB85中执行的程序并将它作为用户程序的一部分下载到CPU。

例如，可以为以下目的使用OB85：

- 要评估OB85的启动信息和判定哪个模板损坏或没插入（指定模板的起始地址）。
- 用SFC49 LGCGADR查找相关模板所在的槽。

如果没有编写OB85，当优先级错误被检测到时CPU转为STOP模式。

在相应的块帮助中可以找到有关OB、SFB、SFC的更详细的信息。

23.9.12 机架故障（OB86）

说明

当检测到下列故障时，CPU的操作系统调用OB86：

- 中央扩展机架故障（不适用S7-300找不到IM或IM损坏，或者），例如连接电缆断线、机架上的分布电源故障等。
- PROFIBUS-DP 主站、从站故障或PROFINET 网络中IO元件或IO系统故障。

故障消除时也会调用该OB块（事件到来和离开时都调用该OB）。

在OB86中编程

必须使用STEP 7在用户程序中生成一个OB86。在生成的块中编写要在OB86中执行的程序，并将它作为用户程序的一部分下载到CPU中。

例如，可以为以下目的使用OB86：

- 要评估OB86的启动信息和判定哪个机架损坏或找不到。
- 用系统功能SFC 52 WRUSNSG将消息存入诊断缓冲区，并发送消息到监视设备上。

如果没有编写OB86，当检测到机架故障时CPU转为8TOP模式。

在相应的块帮助中可以找到有关OB、SFB、SFC的更详细的信息。

23.9.13 通讯错误（OB87）

说明

当使用通讯功能块或全局数据通讯进行数据交换时出现通讯错误，CPU的操作系统调用OB87，例如：

- 当接收全局数据时，检测到不正确的帧标识。
- 用于全局数据状态信息的数据块不存在或太短出。

在OB87中编程

必须使用STEP 7在用户程序中生成一个OB87。在生成块中编写要在OB87中执行的程序，并将它作为用户程序的一部分下载到CPU。

例如，可以为以下目的使用OB87：

- 要评估OB87的启动信息。
- 如果找不到用于全局数据通讯状态信息的数据块，要生成该数据块。

如果OB87没有编程，当检测到通讯错误时CPU转为STOP模式。

在相应的块帮助中可以找有关OB、SFB、SFC的更详细的信息。

23.9.14 编程错误（OB121）

说明

当出现编程错误时，CPU的操作系统调用OB121，例如：

- 寻址的定时器不存在。
- 调用的块未下载。

在OB121中编程

必须使用STEP 7在用户程序中生成一个OB121。在生成的块中编写要在OB121中执行的程序，并将它作为用户程序的一部分下载到CPU中。

例如，可以为以下目的使用OB121：

- 要评估OB121的启动信息。
- 要在消息数据块中存入故障原因。

如果不编写OB121，当检测到编程错误时CPU转为STOP模式。

在相应的块帮助中可以找到有关OB、SFB、SFC的更详细的信息。

23.9.15 I/O访问错误（OB122）

说明

当STEP 7指令访问一个信号模板的输入或输出时，而在最近的一次暖起动中没有分配这样的模板，CPU的操作系统会调用OB122，例如：

- 直接访问I/O出错（模板损坏或找不到）
- 访问一个CPU不能识别的I/O地址

在OB8122中编程

必须使用STEP 7在用户程序中生成一个OB8122。在生成的块中编写要在OB122中执行的程序，并将它作为用户程序的一部分下载到CPU中。

可以为以下目的使用OB122：

- 要评估OB122的启动信息。
- 要调用系统功能SFC44为输入模板提供一个替代值以便使程序可以用这个有意义的、随过程变化的数值继续执行。

如果不编写OB122，当检测到I/O访问错误时CPU转为STOP模式。

在相应的块帮助中可以找到有关OB、SFB、SFC的更详细的信息。

24 打印与归档

24.1 打印项目文档

自动化任务的控制程序编制完成后，便可以利用集成于STEP 7中的打印功能将所有重要数据打印出来，作为项目的文献资料。

项目中可打印的部分

可以从SIMATIC Manager直接打印选定对象的内容，也可以打开相关的对象再起动打印过程。

一个项目中的下列部分可通过SIMATIC Manager直接打印出来：

- 对象的树形图（项目或库的结构）
- 对象清单（一个对象文件夹的目录）
- 对象内容
- 信息

通过打开相关的对象，一个项目中的下列部分可以打印出来：

- 以梯形图、语句表或功能图表述的程序模块，也能以其它语言表述（选件）
- 带有符号名称和绝对地址的符号表
- 组态列表，包括可编程控制器中模块的排列及模块参数
- 诊断缓冲器区的内容
- 变量表，包括监控格式及监控值和更改值
- 参考数据，例如：交叉表、赋值表、程序结构、未使用地址表、无符号地址表
- 全局数据表
- 带有模块状态的模块信息
- 操作员相关文本（用户文本和文本库）
- 可选软件包的文档，例如其它编程语言

DOCPRO选件软件包

可以使用选件软件包DOCPRO，生成、编辑和打印标准化的接线手册。该软件包可生成符合DIN和ANSI标准的设备文档。

24.1.1 打印的基本步骤

打印的步骤如下：

1. 打开相关的对象，在屏幕上显示要打印的信息。
2. 在应用窗口上使用菜单命令**File > Print** 打开“打印”对话框。根据使用的应用程序，菜单条上的第一项可能不是“File”，而是应用程序要处理的对象，如“Symbol Table（符号表）”。
3. 在打印对话框中，如有必要可以修改打印选项（打印机，打印范围，份数等），然后关闭对话框。

有些对话框有“打印”按钮，例如“模块信息”对话框。点击该按钮即可打印对话框的内容。

不需要打开程序模块。可在SIMATIC Manager中使用菜单命令 **File > Print** 直接打印。

24.1.2 打印功能

打印对象所具有的附加功能见下表：

打印对象	菜单命令	功能	功能	功能
		打印预览	页面设定 “纸张大小”	打印设定 “页眉和页脚”
程序模块、STL源文件	File > *	•	•	•
模板信息		—	•	•
全局数据表	GD Table> *	•	•	•
组态列表	Station > *	•	•	•
对象、对象文件夹	File > *	—	•	•
参考数据	Reference Data > *	•	•	•
符号表	Symbol Table > *	•	•	•
变量表	Table> *	—	•	•
连接表	网络> *	•	•	•
操作员相关文本（用户文本，文本库）	Texts> *	•	•	•
* ： *号是通配符，表示菜单命令中的相关功能（例如，打印预览或页面设定）				

对于每个单独的打印对象的逐步引导说明，可以在“How to Print（如何打印）”之下找到。

打印预览

利用“打印预览”功能，可以显示需要打印文档的页面布局。

注意

文档最终的打印格式在打印预览中不能显示。

设定页面格式以及页眉和页脚

使用菜单命令File > Page Setup，可以对要打印的所有文档设定纸张大小（例如A4、A5、信封）以及打印方向（横向、纵向）。可以对当前段或整个文档分别设置。

调整文档的页面布局使其符合打印纸的格式。若文档过宽，右侧的页边会打印到后续页上。

如果选择带有页边的页面格式（如A4带页边），则打印出来的文献在页的左侧留出一个页边，可以穿孔和装订。

选择“Labeling Fields(标签区)”，可以为整个打印文档或打印当前段设定页眉和页脚。

24.1.3 关于打印对象树形图的特别说明

在“打印对象清单”对话框中，除了对象清单之外，还可以通过选中“树形图”选项而打印出对象的树形图。

如果在“打印范围”中选择“全部”，则可以打印出整个树形图。若选择“选择”，则可以打印出被选中对象以下的树形图。

注意

对话框中的设定仅适用于打印清单和树形图，而不适应于打印对象的内容；打印内容时在相应的应用中有相关的设定。

24.2 项目和库的归档

可以将项目或库以压缩的形式存储在一个归档文件中。这种压缩存储过程可以在硬盘上进行也可以在一种便携的数据载体上进行（例如，用软盘）。

归档程序

在STEP 7中，使用归档功能可以对需选程序进行归档。归档应用程序ARJ和PKZIP 4.0作为一个组成部分已包含在STEP 7软件包内。在...\Step7\S7bin\路径下可以找到这些程序及其使用说明。

如果使用下列归档程序，则应选择下列的版本（或最高版本）：

- PKZip Commandline V4.0（包含在STEP 7中）
- WinZip 版本6.0以上
- JAR 版本1.02以上

- ARJ V2.4.1a（只适合于可恢复的归档，包含在STEP 7中）
- ARJ32 V3.x（只适合于可恢复的归档）
- LHArc　版本2.13以上（只适合于可恢复的归档）

归档程序

STEP 7 V5.2只支持PKZip 4.0、JAR和WinZip。但上面列出的其它程序可以支持对其恢复。

如果在更早版本的STEP 7 中，只能使用ARJ32 V3.x创建归档文件，恢复归档文件时只能使用相同的程序。

使用PKZIP V4.0创建归档程序时，网络驱动器所需的时间比本地驱动器所需的时间要长些。

24.2.1　使用保存/归档

另存为（Save As）

使用另存功能，可以创建项目副本并以其它名称保存。

可以利用此功能：

- 生成备份副本。
- 复制一个存在的项目以便修改后用于其它目的。

生成副本的最快方法是，选择“Save as”选项并且不重新安排对话框。项目目录中的整个文件结构都不作任何检查地被复制，然后以其它名称保存。

数据介质必须有足够的空间来存储备份副本。不要尝试将项目保存到磁盘，因为磁盘通常没有足够的可用空间。要往磁盘上传输项目数据，使用“归档”功能。

带有重新排列任务的保存将花费较长的时间，当对象不能被复制和保存时会显示出一条信息。引起的原因可能是一个对象所需的选项软件包或数据不完全。

归档

可以将项目或库以压缩的形式存储在一个归档文件中。这种压缩存储过程可以在硬盘上进行，也可以在一种便携的数据载体上进行（例如，用软盘）。

将项目转移到软盘只能用归档文件的形式。若一个项目过大，则在选用归档程序时注意其应具有生成多张软盘上连续归档文件的功能。

被以压缩形式存入一个归档文件的项目或库不能被编辑修改。若希望对它们进行编辑，就必须将数据解压缩，即恢复项目或库。

24.2.2　对归档的要求

为了将一个项目或一个库进行归档，必须满足以下要求：

- 必须在系统中安装归档程序，归档程序与STEP 7的连接参见在线帮助中的“归档/恢复

的步骤”。

- 所有与项目有关的数据无一例外地必须在项目目录之内或项目子目录之内。当使用C语言开发环境时，有可能将数据存储在其它位置。则归档文件中不包括这些数据。
- STEP 7 V5.2只支持归档程序Pkzip 4.0，JAR。但是在恢复时也支持ARJ和LHArc程序。

24.2.3 归档/恢复的步骤

可以对项目或库进行归档和恢复，使用的菜单命令为File > Archive或File > Retrieve。

注意

不能对压缩到归档文件中的项目或库进行编辑。如果希望再次编辑它们，必须提取数据，即恢复项目或库。

当进行恢复操作时，项目或库会自动恢复到项目/库清单之中。

设定目标目录

可在SIMATIC Manager中使用菜单命令Options > Customize设定目标目录。

在该对话框的“Archive”中，可以选择或取消“恢复时检查目标路径”选项。

如果绍选项无效，那么在同一个对话框的“通用”标签中为“项目存储位置”和“库存储位置”而设置的路径将用来作为恢复的目标路径。

将一个归档文件复制到软盘

可以将一个项目/库进行归档，然后将归档文件复制到软盘上。也可以在“Archive（归档）”对话框中选择软盘驱动器作为目标路径。

25 使用M7可编程控制系统

25.1 M7 系统程序

具有标准PC机结构的M7-300/M7-400自动控制微机，在SIMATIC自动化平台上，使自由编程功能得到扩展。可以使用高级语言例如C语言或图形化编程语言CFC（连续功能图）来编制SIMATIC M7的用户程序。

为了生成程序，除STEP 7之外，你还需要用于M7-300/400的系统软件M7-SYS RT和用于M7程序的开发环境（Pro C/C++或CFC）。

基本步骤

当使用SIMATIC M7创建一个自动控制方案时，需要完成一系列基本任务。下表指出了对于大多数项目都要执行的任务，并将其定义为基本步骤。下表还给出了在本手册或其它手册中可供参考的相关章节。

步 骤	说 明
设计自动化控制方案	M7专用； 参考：M7-SYS RT编程手册
启动STEP 7	同S7
创建项目结构 设置站点 建立硬件组态	同S7
建立通讯连接组态	同S7
定义符号表	同S7
生成C语言或CFC用户程序	M7专用； 参考：ProC/C++
配置操作系统 在M7-300/M7-400中安装操作系统 向M7下载硬件组态和用户程序	M7专用； 参考：M7-SYS RT用户手册
检测并调试用户程序	ProC/C++
运行监控和M7诊断	同S7，但是没有用户自定义的诊断
打印与归档	同S7

在M7中有什么不同？

对于M7-300/M7-400，STEP 7不支持下列功能：

- 多CPU运算—若干个CPU的同步操作
- 变量强置

- 全局数据通讯
- 用户自定义的诊断

M7可编程控制系统的管理

STEP 7为使用M7可编程控制系统提供如下特别的支持：

- 在M7-300/M7-400中安装操作系统
- 通过编辑系统文件对操作系统进行配置
- 向M7-300/M7-400装载用户程序
- 软件升级

为了访问M7可编程控制系统的管理器，在包含有M7 CPU或FM模块工作站的项目中，选中M7程序文件夹，选择菜单命令：

PLC > Manage M7 System

在M7-SYS RT用户手册和在线帮助中有详细的说明。

25.2 用于 M7 编程的选装软件

M7选装软件

STEP 7提供了所需的如下基本功能：

- 项目的生成与管理
- 对硬件进行配置和参数设定
- 配置网络和连接
- 符号数据管理

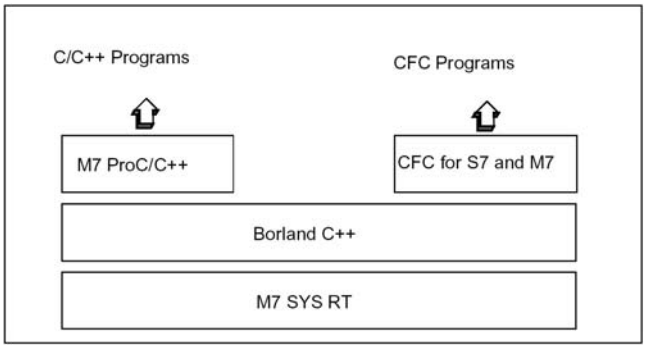
不论是使用SIMATIC S7还是SIMATIC M7，都能提供这些功能。创建M7应用程序时，除了STEP 7之外，还需要M7选装软件。

软件	内 容
M7-SYS RT	• M7 RMOS32操作系统 • M7-API系统库 • 对MPI的支持
用于S7和M7的CFC	CFC（连续功能图）编程软件
软件	**内 容**
M7-ProC/C++	• 用于在STEP 7中实现Borland开发环境的连接 • 符号的编辑与生成 • xdb386高级语言调试工具的推理规则
Borland C++	Borland C/C++开发环境

结合M7选装软件，STEP 7还可以支持下列附加功能：

- 通过多点接口（MPI）向M7可编程控制系统下装数据
- 查询有关M7可编程控制系统的信息
- 在M7可编程控制系统上进行特别设定和对M7进行复位

下图显示了M7编程对M7选装软件的依赖性。



小结

为生成...	所需的M7选装软件...
C/C++程序	• M7-SYS RT • M7-ProC/C++ • Borland C++
CFC程序	• M7-SYS RT • 用于S7和M7的CFC • Borland C++

不同软件提供的支持

为M7应用所特需的工具，一部分集成于STEP 7之中，另一部分是M7的选装软件。

下表指明了不同软件包支持的功能：

软件	提供的支持
STEP 7	• 安装M7操作系统 • 管理M7可编程控制系统 • 下载、起动和删除M7程序 • 显示状态和诊断数据 • CPU复位
M7-SYS RT	M7操作系统和M7系统软件应用帮助实现以下功能： • 程序处理的控制 • 内存和资源的管理 • 对计算机硬件和SIMATIC硬件的访问 • 处理中断 • 诊断 • 状态监控 • 通讯
M7-ProC/C++	• 集成代码的生成(将Borland的开发环境集成于STEP 7 中) • 将项目符号连接到源代码 • 集成的调试功能
Borland C++	• 生成C和C++程序
用于S7和M7的CFC	• 生成、检测和调试CFC程序 • 起动并运行CFC程序

25.3 M7-300/M7-400 操作系统

对于用高级语言C和C++生成的应用程序，操作系统的作用是非常重要的。对于这些应用，操作系统要承担下列任务：

- 访问硬件
- 管理资源
- 系统集成
- 与系统中其它部件进行通讯

为了完成自动控制任务，SIMATIC M7自动控制计算机使用M7 RMOS32（实时多任务操作系统）。M7 RMOS32经过扩展后包含了一个调用接口，M7 API（应用程序接口）将其集成到SIMATIC系统之中。

实时操作系统M7 RMOS32是用于实时和多任务控制方案的，在时间准则上使用32位处理。对于M7模块它可以有如下列配置：

- M7 RMOS32
- M7 RMOS32带MS-DOS

M7可编程控制系统的操作系统配置取决于所用的M7模块：

操作系统配置	模块/主内存	PROFIBUS-DP和TCP/IP有/无	安装在海量存储器上
M7 RMOS32	FM 356-4 / 4 MB FM 356-4 / 8 MB CPU 388-4 / 8 MB FM 456-4 / 16 MB CPU 488-3 / 16 MB CPU 486-3 / 16 MB	无 有 有 有 有 有	存储卡≥4MB或在硬盘上
M7 RMOS32带 MS-DOS	FM 356-4 / 8 MB CPU 388-4 / 8 MB FM 456-4 / 16 MB CPU 488-3 / 16 MB CPU 486-3 / 16 MB	无 无 有 有 有	存储卡≥4MB或在硬盘上

26 提示与技巧

26.1 更换组态列表中的模板

如果使用HW Config修正一个站的组态，并且需更换一个新模板，应如下进行：

1. 使用拖放功能，将模板从Hardware Catalog（硬件目录）窗口中拖到旧模板上。
2. 放下新模板。应尽可能地使新模板使用已插模板的参数。

该方法比删除旧模板、插入新模板并对其赋值参数的速度快。

可以使用菜单命令**Options > Settings**（“Enable Module Swapping（使能模板交换）”），在HW Config中关闭或打开该功能。

26.2 具有大量网络站的项目

如果逐个组态所有站，并使用菜单命令**Options > Configure Network**调用NetPro，以便组态连接，站将自动放置在网络视图中，其缺点是必须根据拓扑条件安排站和子网。

如果项目中包括大量的网络站，并且需要在这些站之间组态连接，则应在网络视图中组态系统结构，并且保存概览。

1. 在SIMATIC Manager中创建新项目（菜单命令**File > New**）。
2. 启动NetPro（菜单命令**Options > Configure Network**）。
3. 在NetPro中一个站一个站地创建：
   - 使用拖放功能，从Catalog窗口中拖出站。
   - 双击站点，启动HW Config。
   - 使用拖放功能，在HW Config中放置具有通讯功能的模块（CPU、CP、FM、IF模块）。
   - 如果需要网络连接这些模板，双击组态表中的相应行，创建新的子网，并网络连接接口。
   - 保存组态，并切换到NetPro。
   - 在NetPro中，定位站和子网（使用鼠标移动对象，直至到达所需位置）。
4. 在NetPro中组态连接，根据需要校正网络连接。

26.3 重新排列

若在使用STEP 7过程中，出现一些无法解释的问题，通常可以通过对项目或库的数据库进行重新排列予以解决。

选择菜单命令**File > Rearrange**可进行重新排列。该功能可消除由于某些内容被删除后而产生的数据存储不连续，即使得项目/库所需的存储器量减少。

该功能对项目或者库的数据存储进行优化，其方法类似硬盘碎片整理程序对硬盘的文件存储进行优化。

重新排列处理的时间长短取决于要移动的数据量大小，很可能要花些时间。因此该功能不能自动执行（例如，当关闭一个项目时无法执行该功能），而应由用户在打算重新排列项目或库时进行调用。

要求

项目和库需要进行重新排列时，必须保证其中不能有正在被其它用户编辑的对象，因为这时的存取权会被封锁。

26.4 如何在多个网络中编辑符号

可以用LAD/STL/FBD程序编辑器查看和编辑多个网络中的符号。

1. 点击选择一个网络名(例如“Network 1”)
2. 按住CTRL键可同时选择其它网络
3. 右击调用文本菜单命令**编辑符号**

可以使用快捷键CTRL+A选择一个程序块中的所有网络，然后选中网络名。

26.5 用变量表进行测试

监视和修改变量表中的变量时，请注意以下编辑技巧：

- 可以在“符号”栏输入符号和地址，也可以在“地址”栏输入符号和地址。然后输入便会自动地在正确的栏中写入。
- 需要显示修改值，应将“Monitoring（监视）”触发点设在“Beginning of Scan Cycle（扫描循环开始）”，将“Modifying（修改）”触发点设在“End of Scan Cycle（扫描循环结束）”。

- 如果将光标放在红色的行上，可以显示一个简短信息来说明错误的原因。按F1键，可获得消除这些错误的建议。
- 只能输入已在符号表中定义过的符号。必须完全按照符号表中的定义来输入符号。有特殊字符的符号名必须用引号括起来（例如，“Motor.Off”，“Motor+Off”，“Motor-Off”）。
- 可以在“Online（在线）”标签（“Customize（自定义）”对话框）中关闭警告。
- 改变连接时无需事先断开连接。
- 监控变量时，可以定义监控触发器。
- 通过选择行，并执行“Force（强制）”功能，可以修改所选变量。只有高亮显示的变量才能修改。
- 无确认退出：
  在“Monitoring（监视）”、“Modifying（修改）”、“Release PQ（释放PQ）”时，如果按下ESC键，将不会询问是否需要退出就会中止“Monitoring（监视）”和“Modifying（修改）”。
- 输入一个“Contiguous Address Range（连续的地址范围）”：
  使用菜单命令**Insert > Range of Variable**。
- 显示和隐藏栏：
  使用以下菜单命令可以显示和隐藏单个栏：
  符号：**View > Symbol**
  符号注释：**View > Symbol Comment**
  状态值的格式：**View > Display Format**
  变量的状态值：**View > Status Value**
  变量的修改值：**View > Modify Value**
- 同时修改表中几行的显示格式：
  1 按住鼠标左键在所需要的表中拖动，选中那些要改变显示格式的区域。
  2 使用菜单命令**View > Select Display Format**，选择输出格式。只有那些表中允许改变格式的行才能改变格式。
- 通过F1键显示输入示例：
  - 如果将光标放在地址栏并按F1键，可以获得地址输入的示例。
  - 如果将光标放在修改值栏并按F1键，可以获得修改/强制值输入的示例。

26.6 用程序编辑器修改变量

在程序编辑器中，可以将按钮设置为二进制输入和存储器位，这样可以通过点击鼠标快速、方便地修改这些地址。

需求

- 在符号表中，已经通过菜单命令 Special Object Properties > Control at Contact对所要修改的地址分配了属性。
- 已经在LAD/STL/FBD程序编辑器的“General”表中选择了“Control at Contact”选项(菜单命令Options > Customize)。
- 已经选择了菜单命令Debug > Monitor。

触发条件是“permanent/at the cycle start（永久/周期启动）”。

只要持续按下按钮，就可以一直监视所输入的实际变量。也可以通过多重选择(CTRL键)修改多个输入。

按下按钮将对位存储器或不能修改的输入状态置1。当通过文本菜单或在变量表中输入明确的要求，或如果用STEP 7程序对地址复位，才能将其置为0。

对于非负输入（常开）或位存储器，按下按钮将使修改值为“1”；对于负的输入（常闭）或位存储器，则修改值为“0”。

使用WinCC时的注意事项

如果已经通过操作员控制和变量监视在WinCC中启动程序编辑器，只有WinCC的控制选项是允许的。否则，如果操作员已经具有WinCC的“维护权”，两个修改选项都可以使用。

编辑“Controlat Contact”属性。

26.7 虚拟工作存储器

引起STEP 7出问题的另一个原因有可能是虚拟工作存储器不够。

为了使用STEP 7，应该调整虚拟存储器的设定。调整的步骤如下：

1. 打开“控制面板”，利用命令**Start > Settings > Control Panel**并双击“System（系统）”图标。对于XP，通过**START > Desktop > Properties > Advanced > System Performance > Settings**打开。
2. 在Windows2000下，选择“Advanced”并点击“System Properties Options（系统性能选项）”按钮。在Windows XP/Server 2003下，在“System Settings（系统设定）”对话框中选择“Advanced”。
3. 点击“Change（修改）”按钮。
4. 在“最小”处输入至少40MB，在“最大”处至少150MB。

注意

对于在硬盘（默认为C：）上且为动态的虚拟存储器，必须确保为子目录TMP或TEMP提供足够的存储器量（大约20至30MB）

- 如果S7项目与虚拟存储器设定在同一分区内，则应保证有大约两倍于S7项目大小的存储器空间。
- S7项目存于其它分区，此项要求无关。

A 附录

A.1 操作模式

A.1.1 操作模式和模式转换

操作模式

操作模式描述了 CPU 在某个特定的时间点的状态。在编程启动、测试控制器和故障诊断时了解 CPU 的操作模式是有用的。

S7-300 和 S7-400 可采取以下操作模式：

- STOP (停机)
- STARTUP (启动)
- RUN (运行)
- HOLD (保持)

在 STOP 模式，CPU 检查所有组态模板或由缺省地址设置的模板是否实际存在，并且将 I/O 设置为预定义的初始状态。在 STOP 模式下用户程序不执行。

在 STARTUP 模式下，要区别启动类型“warm restart(暖启动)”“cold restart(冷启动)”和“hot restart(热启动)”：

- 在暖启动中，程序处理从头开始，使用系统数据和用户地址区的初始设置(非记忆的定时器、计数器和位存储被复位)。
- 在冷启动中，读入过程映像输入表并且STEP 7用户程序从OB1的第一条指令开始处理(也适用于暖启动)。
  - 所有由SFC在工作存储器中生成的数据块都被删除；保留下来的数据块具有来自装载存储器的预设值。
  - 过程映像区和所有定时器、计数器及位存储被复位，无论它们是否是可记忆的。
- 在热启动中，程序从中断的断点处继续运行(定时器、计数器和位存储不复位)。热启动只在S7-400 CPU上是可能的。

在 RUN 模式下，CPU 执行用户程序，更新输入和输出，处理中断和过程故障信息服务。

在 HOLD 模式，用户程序的执行被暂停，可以单步地测试用户程序。只有当使用编程器进行测试时才有可能处于 HOLD 模式。

在所有这些模式中，CPU 可以通过多点接口(MPI)进行通讯。

其它操作模式

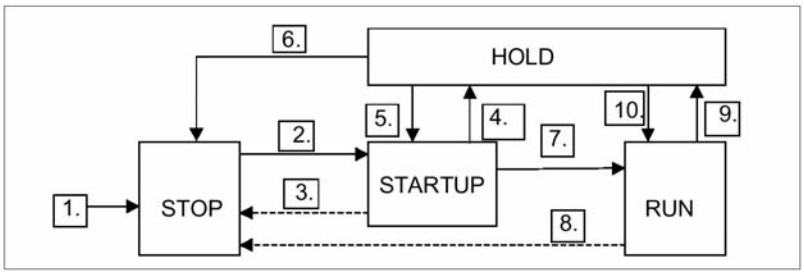
如果 CPU 尚未作好操作准备，它可以处于以下几种模式：

- off，即电源关断。
- 故障，即有故障出现。

要检测 CPU 是否有故障，将 CPU 切换到 STOP，关断电源再通电。如果 CPU 启动，打开诊断缓冲区并对问题进行分析。如果 CPU 不启动，则需要换新的了。

操作模式转换

下图所示，S7-300 和 S7-400 CPU 的操作模式及模式转换。



下表所示的操作模式转换的条件。

转换	描 述
1.	在接通电源后，CPU处于STOP模式
2.	CPU转为STARTUP模式： • 用钥匙开关或编程器将CPU转为RUN或RUN-P后 • 由通电自动触发起动之后 • 如果执行了RESUME或START通讯功能 在以上的后两种情况下，钥匙开关必须在RUN或RUN-P
3.	CPU转回STOP模式，当： • 在启动过程中检到错误 • 由钥匙开关或由编程器将CPU转为STOP • 在启动OB中执行了停机命令 • 执行了STOP通讯功能
4.	在启动程序中遇到断点，CPU转为HOLD模式
5.	在启动程序中设置了断点并且执行了"EXIT HOLD"命令（测试功能），CPU转为STARTUP模式
6.	CPU转为STOP模式，当： • 用钥匙开关或通过编程器将CPU转为STOP • 执行了STOP通讯命令
7.	如果启动成功，CPU转为RUN

转换	描 述
8.	CPU转回STOP模式当： • 在RUN模式下检测到错误且相应的OB块没有装载 • 用钥匙开关或编程器将CPU转为STOP • 在用户程序中执行了STOP命令 • 执行了STOP通讯功能
9.	当用户程序中遇到断点，CPU转为HOLD模式
10.	当设置了断点且执行了"EXIT HOLD"命令时，CPU转为RUN模式

操作模式优先级

如果同时有多个模式转换请求，则高优先级的操作模式被选中。例如，模式选择开关设为 RUN，试图在编程器上将 CPU 设为 STOP，因为这一模式具有高优先级，所以 CPU 转为 STOP。

优先级	模式
最高	STOP
	HOLD
	STARTUP
最低	RUN

A.1.2 STOP 模式

在 STOP 模式下用户程序不执行。所有输出都设为适当的值以保证控制过程处于安全状态。CPU 作以下检查：

- 是否有硬件问题(例如，模板故障)?
- 是否将缺省设置用于CPU或是否有参数设置?
- 编程的启动特性的条件是否满足?
- 系统软件是否有问题?

在 STOP 模式下，CPU 还可以接收全局数据并可进行被动的单向通讯，实现这些通讯需要对已组态的连接使用通讯 SFB 并且对没有组态的连接使用通讯 SFC。

存储器复位

CPU 存储器可在 STOP 模式下被复位。可以用钥匙开关(MRES)或通过编程器(例如，在下载用户程序前)手动复位存储器。

复位 CPU 的存储器使 CPU 回到它的初始状态：

- 在工作存储器和RAM装载存储器中所有的用户程序以及所有的地址区都被清除。
- 系统参数及CPU和模板的参数被复位为缺省设置。而复位前设置的MPI参数仍保留。
- 如果插入了存储卡(Flash EPROM)，CPU将存储卡中的用户程序复制到工作存储区(如

果存储卡中有适当的组态数据，则所复制的数据包括CPU和模板参数)。

诊断缓存区、MPI 参数、时间以及 CPU 运行时间计数器数器都不复位。

A.1.3 STARTUP 模式

在 CPU 启动用户程序处理之前，必须先执行起动程序。通过在起动程序中编写起动 OB，可以为循环程序指定初始设置。

有三种起动类型：暖起动、冷起动和热起动。只有 S7-400 CPU 有热起动。这必须用 STEP 7 直接在 CPU 参数中设置。

STARTUP 模式的特性如下：

- 处理起动OB中的程序(OB100暖起动，OB101热起动，OB102冷起动)。
- 不能执行时间驱动OB或过程驱动程序。
- 定时器被更新。
- 运行时间计数器数器起动运行。
- 禁止信号模板的数字输出(可通过直接访问置位)。

暖起动

除非系统要求存储器复位，否则暖起动总是允许的。在以下情形之后，暖起动是唯一可能的选项：

- 存储器复位
- 在CPU位于STOP模式时下载用户程序
- I堆栈/B截栈溢出
- 终止暖起动(由于掉电或改变模式选择开关的设置)
- 在热起动前的中断超过了选择的时间限制。

手动暖起动

以下可以触发手动暖起动：

- 模式选择开关
  (CRST/WRST 开关——如果有——必须设置为 CRST)
- 编程器上相应的命令或通讯功能。
  (如果模式选择开关设为 RUN 或 RUN-P)

自动暖起动

自动暖起动在上电后可由以下情形触发：

- 电源掉电时CPU不在STOP模式。

- 模式选择开关设为RUN或RUN-P。
- 上电后没有编程自动热起动。
- 在暖起动过程中CPU被电源掉电中断(与编程的起动类型无关)。

CRST/WRST 对自动暖起动没有影响。

没有后备电池的自动暖起动

如果 CPU 在没有后备电池的情况下运行(免维护操作是必要的)，在电源接通后或电源从关电到上电后，CPU 存储器自动复位并执行暖起动。用户程序必须装载到一个 Flash(闪存)EPROM(存储卡)。

热起动

CPU 处于 RUN 模式掉电，随着电源恢复，S7-400 CPU 运行完整的初始化例行程序，然后自动执行一个热起动。在热启动过程中，用户程序从它被中断的地方继续执行。掉电前没能执行完的用户程序部分就是剩余循环。剩余循环也可以包含时间驱动和中断驱动程序部分。

只有在 STOP 模式下没有进行用户程序的修改(如，重新装载修改过的块)以及没有其它理由作暖起动时才允许作热起动。手动和自动热起动都可以。

手动热起动

只有当 CPU 中设置了适当的参数并且由以下原因造成停机时才有可能执行手动热起动。

- 模式选择开关从RUN转化STOP。
- 用户编写的STOP程序或在调用了未装载的OB后停机。
- 通过编程器或通讯功能的命令引起的STOP模式。

手动热起动可由以下事件触发。

- 模式选择开关
  CRST/WRST 必须设为 WRST。
- 编程器或通讯功能中相应的命令(模式选择开关设为RUN或RUN-P)
- 当CPU参数中设置了手动热起动。

自动热起动

在以下情形下电源上电可触发自动热起动：

- 电源掉电时CPU不在STOP或HOLD模式。
- 模式选择开关设为RUN或RUN-P。
- 在CPU参数中设置了上电后执行自动热起动参数。

CRST/WRST 开关对自动热起动没有影响。

电源掉电后的可保持数据区

S7-300 和 S7-400 对电源掉电再上电的反映不同。

S7-300 CPU(除 CPU318 以外)只能暖起动。使用 STEP 7 可以指定位存储器、定时器、计

数器和数据块区域具有保持功能以避免由于电源掉电而丢失数据。当重新上电时，执行连同存储器在内的自动暖起动。

S7-400 对重新上电的反应依据参数设置而定，可以是暖起动(跟随保持或非保持上电)，也可以是热起动(只能是跟随可保持上电)。

下表所示是 S7-300 和 S7-400 CPU 在暖起动、冷起动或热起动过程中可保留的数据。

X	表示数据可保持
VC	表示逻辑块保留在EPROM中，所有过载的逻辑块都将丢失
VX	表示只有在EPROM中的数据块保留下来，可保持数据取自NV-RAM（在RAM中装载或生成的数据块丢失）
O	表示数据被复位或被擦除(DB块中的内容)
V	表示数据被设为来自EPROM存储器的初始值
-	由于没有NV-RAM所以不可能

下表所示在工作存储器（EPROM 和 RAM 装载存储器）中保留的数据。

EPROM（存储卡或内部集成的）									
有后备电池的CPU					没有后备电池的CPU				
数据	装载存储器中的块	工作存储器中的DB	位存储器 定时器 计数器	位存储器 定时器 计数器	装载存储器中的块	工作存储器中的DB	工作存储器中的DB	位存储器 定时器计数器	位存储器 定时器计数器
			（定义为可保持）	（定义为非保持）		（定义为可保持）	（定义为非保持）	（定义为可保持）	（定义为非保持）
在S7-300上暖起动	X	X	X	0	VC	VX	V	X	0
在S7-400上暖起动	X	X	X	0	VC	-	V	0	0
在S7-300上冷起动	X	0	0	0	VC	V	V	0	0
在S7-400上冷起动	X	0	0	0	VC	——	V	0	0
在S7-400上热起动	X	X	X	X		只允许暖起动			

起动的特性

下表所示为 CPU 起动过程中所作的操作：

按顺序执行的操作	暖起动	冷起动	热起动
清除I堆栈/B堆栈	X	X	0
清除非保持的位存储器、定时器、计数器	X	0	0
清除所有位存储器、定时器、计数器	0	X	0
清除过程映像输出表	X	X	可选
清除数字信号模板的输出	X	X	可选
放弃硬件中断	X	X	0
放弃延时中断	X	X	0
放弃诊断中断	X	X	X
更新系统状态列表（SZL）	X	X	X
评估模板参数并传至模板或传送缺省数值	X	X	X
执行相关的起动OB	X	X	X
执行剩余循环（由于掉电，部分用户程序没执行）	0	0	X
更新过程映像输入表	X	X	X
转换为RUN后使能数字输出（取消OD信号）	X	X	X
X 表示执行 0 表示不执行			

起动失败

如果在起动过程中出现错误，则起动失败并且 CPU 转为或保持 STOP 模式。

失败的暖起动必须再重复进行。起动失败后，暖起动和热起动都是可以执行的。

在以下情形下，起动(暖起或热起)不执行或失败：

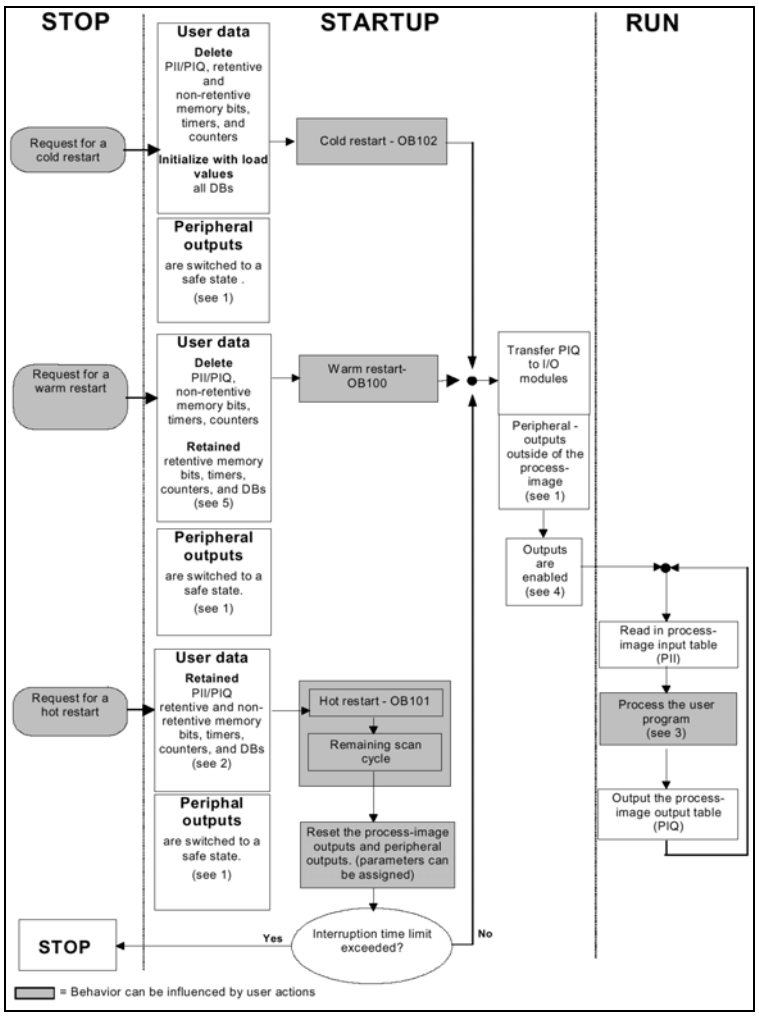
- CPU的钥匙开关设为STOP
- 请求存储器复位
- 插入的存储卡中带有非STEP 7程序(如STEP 5)
- 在单处理器模式下插入多个CPU
- CPU不能识别用户程序中的OB，或该OB已被禁用。
- 上电后，如果CPU发现用STEP 7生成的组态列表中所列的模板实际上没有全部插上(参数的预置值和实际值不同)。
- 如果评估模板参数时出错。

在以下情形下热起动不执行或被失败：

- CPU存储器被复位(存储器复位后只能暖起动)。
- 超过中断时间限制(这一时间是指包括剩余循环在内的起动OB被执行与现有RUN模式之间的时间)。
- 模板组态已改变(例如更换模板)。
- 参数赋值只允许暖起动。
- 在CPU处于STOP模式下装载、删除或修改块。

操作顺序

下图所示为 CPU 在 STARTUP 和 RUN 期间的操作：



CPU 在 STARTUP 和 RUN 期间的操作要点：

1. 通过 I/O 模板将硬件中的所有外设输出切换到安全状态(缺省值=0)。不管用户程序是否使用了过程映像区内的输出还是使用了过程映像区外的输出。

   如果使用的信号模板具有替换值功能，则可以对输出特性进行赋值，例如将输出设置为保留上次有效值。

2. 有必要处理剩余循环。
3. 当中断 OB 第一次调用后，当前的过程映像输入表对这些中断 OB 依然有效。
4. 在用户程序的首次循环中，本地外设输出以及分布式外设输出的状态通过下列步骤决定：

- 使用可进行参数赋值的输出模板，使能替换值输出或保持上次有效值。

- 对于热起动：激活 CPU 的起动参数“Reset ourputs during hot restart”，以便使其输出一个 0（对应于缺省设置）。

- 在起动 OB 中预置输出（OB100，OB101，OB102）。

5. 在没有后备的 S7-300 系统中，只能保持那些设置为保持功能的 DB 区。

A.1.4 RUN 模式

在 RUN(运行)模式，CPU 执行循环、时间驱动和中断驱动程序如下：

- 读入过程映像输入表。
- 执行用户程序。
- 输出过程映像输出表。

只有在 RUN 模式下才能使用全局数据通讯(全局数据表)，为组态连接使用通讯 SFB 以及为非组态连接使用通讯 SFC 在 CPU 之间进行主动的数据交换。

下表所示为在不同的操作模式下可能的数据交换：

通讯类型	CPU1的模式	数据交换的方向	CPU2的模式
全局数据通讯	RUN	←→	RUN
	RUN	→	STOP/HOLD
	STOP	←	RUN
	STOP	X	STOP
	HOLD	X	STOP/HOLD
单向通讯	RUN	→	RUN
调用通讯SFB	RUN	→	STOP/HOLD
调用通讯SFB的双向通讯	RUN	←→	RUN
单向通讯	RUN	→	RUN
调用通讯SFC	RUN	→	STOP/HOLD
调用通讯SFC的双向通讯	RUN	←→	RUN
←→表示　数据交换可双向进行 →表示　数据交只能单向进行 X表示　数据交换不可能			

A.1.5 HOLD 模式

HOLD(保持)模式是一种特殊模式。它只用于起动过程中或 RUN 模式下的测试目的。HOLD 模式意味着：

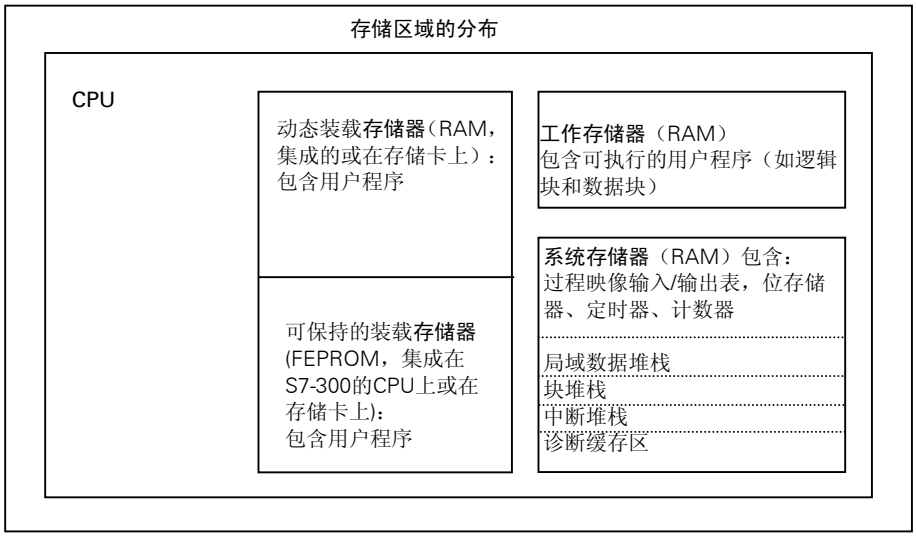
- 所有定时器被冻结：不处理定时器和CPU运行时间计数器，监控时间停止，时间驱动层的基本时钟脉冲停止。
- 实时时钟运行。
- 输出未使能，但是用于测试目的可直接使能。
- 输入和输出可被置位和复位。
- 如果在HOLD模式下，有后备电池的CPU出现掉电，当电源恢复时CPU转为STOP但不执行一个自动热起动或暖起动。没有后备电池的CPU当电源恢复时执行一个自动暖起动。
- 可以接收全局数据，可以为组态连接调用通讯SFB的被动单向通讯以及为非组态连接调用通讯SFC的被动单向通讯(见表中RUN模式)。

A.2 S7 CPU 的存储区域

A.2.1 存储区域的分布

S7 CPU 的存储器可以分为三个区域(见下图)：

- 装载存储器用于存储用户程序，不包括符号地址赋值或注释(这些保留在编程器的存储器上)。装载存储器可以是RAM或EPROM。
- 没有被标为起动所需的块只能保存在装载存储器中。
- 工作存储器(集成的RAM)包含与运行用户程序相关的那部分S7程序。程序只在工作存储器和系统存储器区域内被执行。
- 系统存储器(RAM)内包含由每个CPU为用户程序提供的存储器组件，如过程映像输入和输出表、位存储器、定时器和计数器。系统存储器还包含块堆栈和中断堆栈。
- 除了上述区域外，CPU的系统存储器还提供临时存储器(局域数据堆栈)，它包含块被调用时的临时数据。只有该块激活时这些数据才保持有效。



A.2.2 装载存储器和工作存储器

当从编程器上下载用户程序到 CPU 时，只有逻辑块和数据块能够被装载到 CPU 的装载存储器和工作存储器。

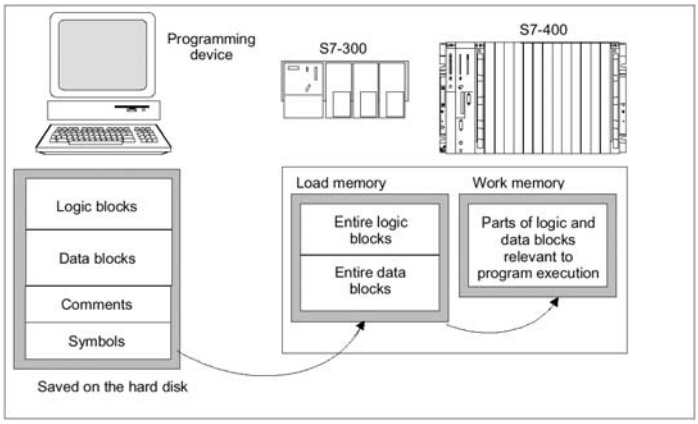
符号地址赋值(符号表)和块注释保留在编程器上。

用户程序的划分

为确保用户程序的快速执行以及对无法扩展的工作存储器减轻无必要的负担，只有与程序执行相关的块才被装载到工作存储器。

程序运行不需要的那些块(如块首)保留在装载存储器中。

下图所示为程序被装载到 CPU 的存储器中。



注意

在用户程序中使用系统功能(如 SFC22 CREAT_DB)生成的数据块全部存储在 CPU 的工作存储器中。

有些 CPU 对工作存储器中的程序和数据区分开管理。这些区域的大小和分配显示在 CPU 的模板信息的“Memory(存储器)”标签中。

将数据块标识为“与执行无关”

作为 STL 程序的一部分，用源文件编写的数据块可以被标识为“与执行无关”(关键字 UNLINKED)。这意味着当它们被下载到 CPU 时，这些 DB 块只能保存在装载存储器中。如果有必要的话，可以使用 SFC 20BLKMOV 将这些块的内容复制到工作存储器。

这种方法节省了工作存储器的空间。可扩展的装载存储器被用作缓存器(例如，一个混合剂的公式：只有用于下一批的公式被装载到工作存储器)。

装载存储器结构

装载存储器可以用存储卡进行扩展。要查找装载存储器的最大容量，可参考“S7-300 可编程控制器，硬件和安装手册”以及“S7-400，MF-400 可编程控制器模板规范参考手册”。

在 S7-300 的 CPU 中装载存储器不仅有集成的 RAM 部分还可以有集成的 EPROM 部分。数据块区域可以通过用 STEP 7 进行参数赋值声明为可保持(见 S7-300CPU 可保持存储区域)。

在 S7-400 的 CPU 上，必须使用一个存储卡(RAM 或 EPROM)来扩展装载存储器。集成的装载存储器是一个 RAM 存储器，主要用于块的重新装载和修改。新型的 S7-400 CPU，可以插入附加的工作存储器。

装载存储器在 RAM 区和 EPROM 区的性能

根据选择了 RAM 或 EPROM 存储器来扩展装载存储器，装载存储器在下载、重新装载或存储器复位过程中可能会有不同的反应。

下表所示为不同的装载方式：

存储器类型	装载方式	装载类型
RAM	装载和删除单个块	PG-CPU连接
	下载和删除一个完整的S7程序	PG-CPU连接
	重新装载单个块	PG-CPU连接
集成的（只有S7-300）或外插的EPROM	下载整个S7程序	PG-CPU连接
外插EPROM	下载整个S7程序	上传EPROM到PG以及在CPU上插入存储卡下载EPROM到CPU

当复位 CPU 存储器(MRES)或者移走 CPU 或 RAM 存储卡时，会丢失存储在 RAM 中的程序。

存储在 EPROM 存储卡上的程序不会被 CUP 存储器复位擦除，即使没有后备电池数据也能保持(传送、备份拷贝)。

A.2.3 系统存储器

A.2.3.1 使用系统存储器区域

S7 CPU 的系统存储器被划分为地址区域(见下表)。在用户程序中使用指令可以在相应的地址区内直接对数据进行寻址。

地址区域	通过以下单位访问	S7记号（IEC）	描述
过程映像输入表	输入（位）	I	在扫描循环开始处，CPU读取输入模板的输入并在该区域内记录数值。
	输入字节	IB	
	输入字	IW	
	输入双字	ID	
过程映像输出表	输出（位）	Q	在扫描循环过程中，程序计算输出值并将它们放在该区域。在扫描循环结束处，CPU将这些计算输出值发送到输出模板。
	输出字节	QB	
	输出字	QW	
	输出双字	QD	
位存储器	存储（位）	M	该区域为用户程序中的中间计算结果提供存储。
	存储字节	MB	
	存储字	MW	
	存储双字	MD	
定时器	定时器（T）	T	该区域为定时器提供存储。
计数器	Counter(C)	C	该区域为计数器提供存储。
数据块	数据块，用"OPN DB"打开	DB	数据块中包含程序需要的信息。它们可以被定义为由所有逻辑块通用（共享DB）或分配给某个特定的FB或SFB（背景DB）
	数据位	DBX	
	数据字节	DBB	
	数据字	DBW	
	数据双字	DBD	
	数据块，用"OPN DI"打开	DI	
	数据位	DIX	
	数据字节	DIB	
	数据字	DIW	
	数据双字	DID	
局域数据	局域数据值	L	该区域包含块被执行时的临时数据。L堆栈还为传送块参数以及记录来自梯形逻辑段的中间结果提供存储器。
	局域数据字节	LB	
	局域数据字	LW	
	局域数据双字	LD	
外设（I/O）区域：输入	外设输入字节	PIB	外设输入和输出区域允许直接访问中央的和分布式的输入和输出模板（DP）。
	外设输入字	PIW	
	外设输入双字	PID	
外设（I/O）区域：输出	外设输出字节	PQB	
	外设输出字	PQW	
	外设输出双字	PQD	

参考以下 CPU 手册或指令集，查找 CPU 上能够使用的地址区域的信息：

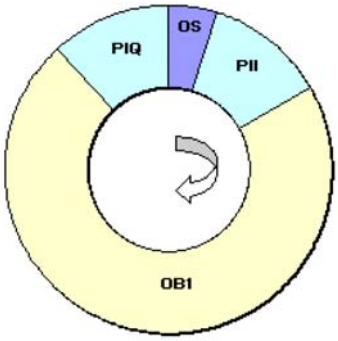
- “S7-300可编程控制器，硬件和安装”手册
- “S7-400，M7-400可编程控制器，模板规范”参考手册。
- “S7-300可编程控制器，指令列表”
- “S7-400可编程控制器，参考指南”

A.2.3.2 过程映像输入/输出表

在用户程序中访问输入(I)和输出(Q)地址区时，程序并不直接访问数字信号模板上的信号状态而是访问 CPU 的系统存储器中的存储区和分布式 I/O。这一存储区域就是过程映像区。

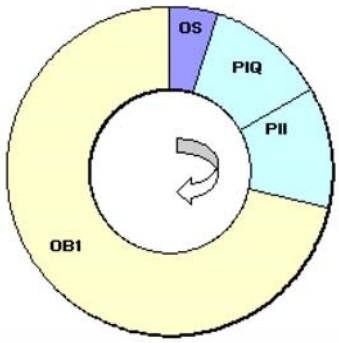
更新过程映像

下图所示为 10/98 以前 CPU 的一个扫描循环周期内过程处理的步骤：



操作系统的内部任务之一是将输入状态读入到过程映像输入表(PII)，一旦该步完成，用户程序将执行所调用的所有块。循环结束时，将过程映像输出表(PIQ)写入模板的输出区。操作系统将单独控制过程映像输入表的读入和过程映像输出表的输出。

下图所示为 10/98 以后的 CPU 的一个扫描循环周期内过程处理的步骤：



操作系统的内部任务之一是将过程映像输出表(PIQ)写入模板的输出区，并将输入状态读入到过程映像输入表(PII)，一旦该步完成，用户程序将执行所调用的所有块。操作系统将单独控制过程映像输入表的读入和过程映像输出表的输出。

过程映像区的优点

与直接访问输入/输出模板相比，访问过程映像的优点在于：在一个程序循环中 CPU 有一个一致的过程信号映像。如果在程序执行过程中输入模板上的信号状态变化，过程映像中的信号状态保持不变，直到下一个循环过程映像才再次更新。

由于过程映像区位于 CPU 内部，访问过程映像所需的时间比直接访问信号模板要少得多。

更新过程映像区的部分区域

除了操作系统对过程映像输入表(PII)和过程映像输出表(PIQ)自动更新外，S7-400 CPU 可以最多对 15 个过程映像的输入区域进行参数赋值。 (见“S7-400 可编程控制器，硬件和安装手册”以及“S7-400，M7-400 可编程控制器模板技术规范参考手册”)。这意味着当需要时，可以更新过程映像表的部分区域，而不依赖过程象表的循环更新。

STEP 7 对过程映像区分配的每个输入/输出地址不再属于 OB1 过程映像输入/输出表。只能用 OB1 过程映像区和所有的过程映像区对输入/输出地址分配一次。

使用 STEP 7 分配地址时可以定义过程映像区分区(在过程映像区中列出了模板的输入/输出地址)。可以通过用户程序，也可以用 SFC 更新过程映像区。

例外：系统不能在同步循环中断 OB 中更新过程映像区分区，即使它们连接到 OB61 至 OB64。

提示

在 S7-300 CPU 上，未使用的输入和输出过程映像表可用作附加的位存储区。使用这一选项的程序则不能运行在 S7-400 的 CPU 上(4/99 以前的 CPU)。

对于 S7-400 CPU

- 作为位存储区的过程映像区不能位于所赋值的“过程映像区大小”内。
- 或者不能位于系统或SFC26/27所能更新的过程映像区内。

使用 SFC 更新过程映像区

通过使用下列 SFC，用户程序可以更新整个过程映像区或过程映像区分区：

- 要求：不能更新有问题的过程映像。
- SFC26 UPDAT_PI：更新过程映像输入表。
- SFC27 UPDAT_PO：更新过程映像输出表。

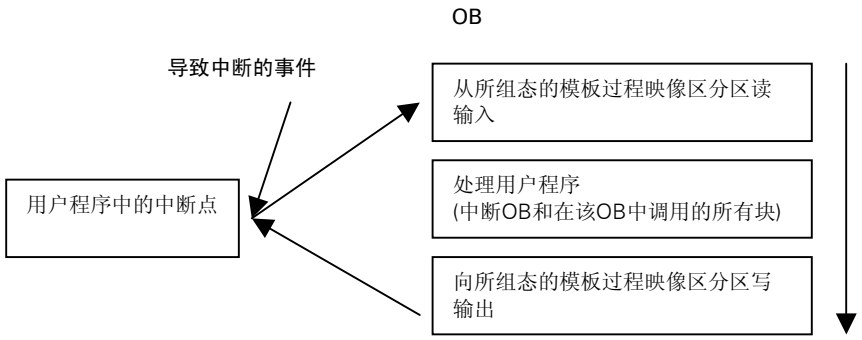
过程映像区分区的系统更新

与在 OB1 处理前或处理后周期更新整个过程映像相似，可以通过调用一个 OB 让系统自动更新过程映像区分区。对于一些特殊的 CPU，可以将该功能作为参数进行赋值。

在运行期间，可以自动地更新所分配的过程映像区分区：

- 在OB处理前，过程映像区分输入
- 在OB处理后，过程映像区分区输出

可以对 CPU 连同 OB 的优先级进行参数赋值，用来指示哪个过程映像区分区分配给哪个 OB。



更新过程映像期间发生 I/O 访问错误(PZF)

在更新过程映像期间，CPU 家族(S7-300 和 400)对一个错误的缺省响应有以下不同：

- S7-300：诊断缓冲区没有相应条目，相应的输入字节复位为0并在故障排除前始终为0。
- S7-400：诊断缓冲区有相应的条目，启动OB85对每个相应的过程映像进行I/O访问。每次访问过程映像时对有故障的输入字节复位为0。

对于 4/99 以后的 CPU，可以对 I/O 访问错误的响应重新进行参数赋值：

- 生成一个诊断缓冲区的条目，并对I/O访问错误启动OB85(在调用OB85前，有故障的输入字节复位为0，在故障排除前系统不会对该字节进行改写)
- 产生S7-300的默认响应(不调用OB85，有故障的输入字节复位为0，在故障排除前系统不会对该字节进行改写)
- 产生S7-400的默认响应(每次访问均调用OB85，对过程映像访问时对有故障的输入字节复位为0)

OB85 的起动频率

除了对已进行参数赋值（到来/离去，或每次 I/O 访问）的 PZF 进行响应外，一个模板的地址空间同时也决定了 OB85 的起动频率：

对于一个地址空间最多为双字的模板，OB85 起动一次。例如：对于一个最大为 32 个输入或输出的数字量模板，或一个 2 通道的模拟量模板。

对于有大量地址空间的模板，根据双字命令访问的频率起动 OB85。例如 4 通道模拟量模板起动 2 次 OB85。

A.2.3.3 局域数据堆栈

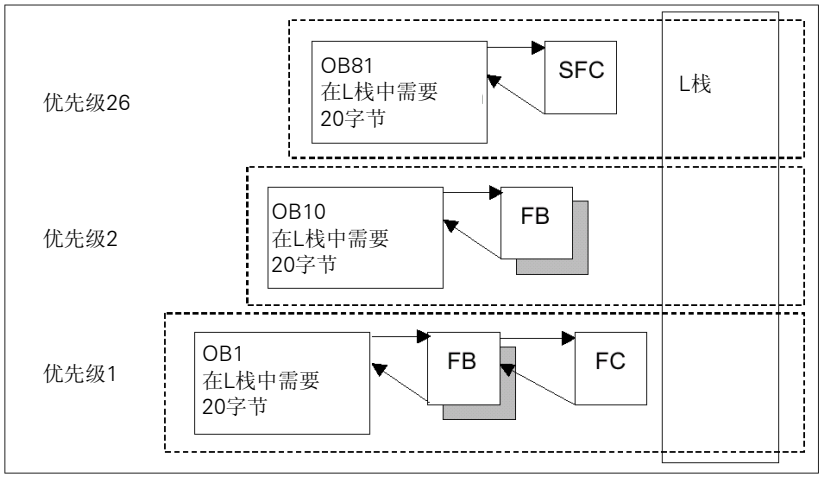
L 堆栈存储以下内容：

- 块的局域数据的临时变量
- 组织块的起动信息
- 传递参数的信息
- 在梯形逻辑程序中的逻辑中间结果

当编写组织块时，可以声明临时变量(TEMP)，这些临时变量只在该块执行时是有效的，然后就被覆盖了。在首次访问局域数据堆栈前，这些局域数据必须被初始化。除此之外，每个组织块还需要 20 个字节的局域数据存储其起动信息。

CPU 对当前正在执行的块的临时变量(局域数据)所用的存储器数量是有限制的。这个存储区域的大小、局域数据堆栈的大小要依据 CPU 而定。局域数据堆栈在各优先级中平等划分(缺省设置)。这意味着每个优先级有自己的局域数据区域，因此，保证了高优先级及其 OB 也有用于它们的局域数据的空间。

下图所示，在一个 OB1 被 OB10 中断，OB10 又被 OB81 中断的例子中说明不同优先级在 L 堆栈中局域数据的分配。



注意

OB 及其相关块的所有临时变量(TEMP)都存储在 L 堆栈。如果在执行块时有太多层嵌套，L 堆栈可能会溢出。

如果超过了一个程序所允许使用的 L 堆栈的大小，S7 的 CPU 转为 STOP 模式。

测试程序中的 L 堆栈(临时变量)。

同步错误 OB 所需的局域数据必须考虑在内。

为优先级分配局域数据

不是每个优先级都需要同等数量的局域数据堆栈的存储区域。通过在 STEP 7 中进行参数赋值，可以为 S7-400 CPU 和 CPU318 的各个优先级分配不同大小的局域数据区域。不需要的优先级可以不选择，对 S7-400 CPU 和 CUP318 来说，其它优先级的存储区域则增加了。未激活的 OB 在程序执行过程中被忽略，从而节省循环时间。

对于其它的 S7-300 CPU，每个优先级分配有固定数量的局域数据(256 字节)，不能够改变。

A.2.3.4 中断堆栈

如果程序的执行被更高优先级的 OB 中断，操作系统将当前累加器和地址寄存器中的内容以及打开的数据块的号码和大小保存在中断堆栈中。

一旦新 OB 执行完了，操作系统从中断堆栈装载信息，从被中断的块的断点处继续执行。

当 CPU 在 STOP 模式时，可以在编程器上用 STEP 7 显示 I 堆栈。可以发现 CPU 转为 STOP 模式的原因。

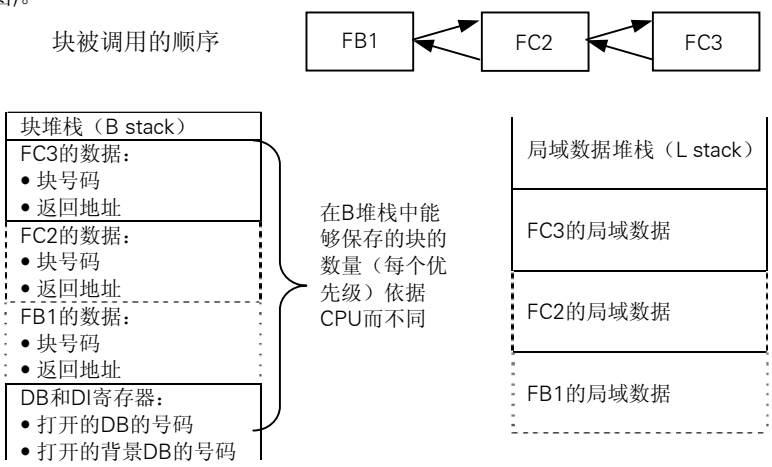
A.2.3.5 块堆栈

如果一个块的处理被另外一个块的调用或更高的优先级(中断/故障服务)中断，B 堆栈存储以下数据：

- 被中断块的号码、类型(OB、FB、FC、SFB、SFC)和返回地址。
- 在被中断块中打开的数据块的号码(从DB和DI寄存器)。

利用这些数据，用户程序在被中断后则可继续执行。

如果 CPU 在 STOP 模式，可以在编程器上用 STEP 7 显示 B 堆栈。B 堆栈中列出所有在 CPU 转为 STOP 模式时已被调用但还未完全执行的块。这些块按照处理启动的顺序排列(见下图)。



数据块寄存器

有两个数据块寄存器。它们包含打开的数据块的号码：

- DB寄存器中包含打开的共享数据块的号码
- DI寄存器中包含打开的背景数据块的号码

A.2.3.6 诊断缓冲器

诊断缓冲器按照其发生的顺序显示诊断消息。第一条是最新发生的事件。诊断缓冲器中条目的数量取决于所使用的模板及当前的运行模式。

诊断事件包括：

- 一个模板的故障
- 接线错误
- CPU中的系统故障
- CPU的模式转换
- 用户程序中的错误
- 用户定义的诊断事件(通过系统功能SFC52实现)

A.2.3.7 评估诊断缓存器

系统状态列表的一部分就是诊断缓冲区，这里包括更多的关于系统诊断事件和用户定义的诊断事件的信息，并按其出现的顺序排列。当系统诊断事件出现时写入诊断缓冲区的信息与传送至相应组织块的启动信息是一致的。

无法清除诊断缓存区中的各项条目及其内容，即使存储器全清之后它们也会保留下来。

诊断缓存区提供以下可能：

- 如果CPU转为STOP模式，可以评估最后的引起停机的事件并确定起因的位置。
- 能够更快地检测故障的原因，增加了系统的有效性。
- 可以评估和优化动态的系统响应。

组织诊断缓存区

诊断缓冲区被设计用作一个环式缓冲器，其最多输入项的数量依各模板而定。这意味着当达到最大输入项数时，下一个诊断缓冲事件使最早的一项被删除。所有输入项向后移一个置位。即最新的输入总是诊断缓冲区的第一项。对于 S7-300 CPU314，可能的输入项数是 100：

显示的诊断缓存区中输入条目依模板及其当前操作模式而定。有些 CPU 可以设置其诊断缓存区的长度。

诊断缓存内容

诊断缓存中上部列表框中包含所有出现的诊断事件并有以下信息：

- 输入条目的序列号(最新的输入项是1号)
- 诊断事件的时间和日期：如果模板有集成的时钟则显示该模板的时间和日期。保持缓存区中的时间数据有效，为模板设置时间和日期并定期检查是非常重要的。
- 对诊断事件的简短描述。

在诊断缓存下部的文本框中，显示在上部窗口列表中选中事件的所有附加信息。这些信息包括：

- 事件号
- 事件描述
- 由诊断事件引起的模式转换
- 引起该条目进入缓存区的错误在块中所处位置的索引(块类型、块号码、相对地址)
- 到来或离去的事件状态
- 事件特定的附加信息

用“Help on Event(事件帮助)”按钮可以显示在上部列表框中选中事件的附加信息。

在系统块和系统功能中的参考帮助中可以得到事件 ID 的信息。

保存内容到一个文本文件

在“Module Information(模板信息)”对话框的“Diagnostic Buffer(诊断缓存区)”标签中使用“Save As(另存为)”按钮，可以将诊断缓存区中的内容存为 ASCII 文本。

显示诊断缓存区

通过“Module Information”对话框中的“Diagnostic Buffer”标签，或在程序中使用系统功能 SFC5I RDSYSST，可以在编程器上显示诊断缓存区的内容。

停机前的最后输入项

可以指定将 RUN 转为 STOP 前最后一个诊断缓存条目送到一个登录过的监视设备上(如 PG、OP、TD)，以便快速锁定和纠正引起 STOP 模式的错误。

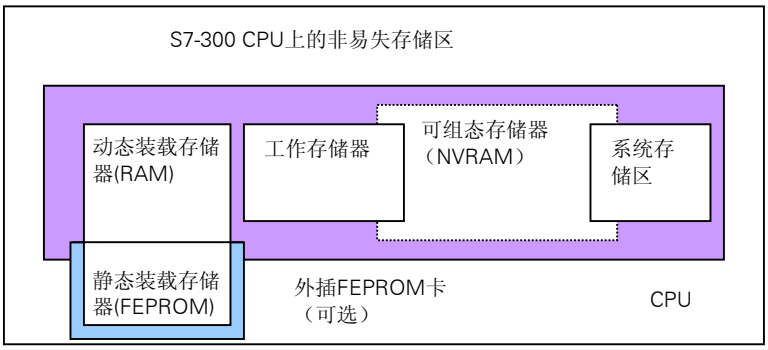
A.2.3.8 S7-300 CPU 上的保持存储区

如果出现电源掉电或 CPU 存储器复位(MRES)，S7-300 的存储器(动态装载存储器(RAM)、工作存储器和系统存储器)被复位并且以前保存在这些区域的所有数据全部丢失。对 S7-300 CPU，可以用以下方法保护程序及其数据：

- 使用后备电池保护存在装载存储器、工作存储器以及部分系统存储器中的全部数据。
- 可以将程序存在EPROM(既可以是存储卡也可以是集成在CPU上的，参考“S7-300可编程控制器，硬件和安装”手册)。
- 依据CPU的不同，可以存储一定数量的数据在非易失区域NVRAM。

使用 NVRAM

S7-300 CPU 在 NVRAM(非易失 RAM)中提供一个区域(见下图)。如果已将程序存储到装载存储的 EPROM 中，可以通过组态 CPU 保存一定的数据(如果出现掉电或当 CPU 从 STOP 转为 RUN)。



要实现这一功能，设置 CPU 使以下数据保存在非易失 RAM 中：

- DB中包含的数据(如果已将程序保存在装载存储器的EPROM中，这样做才有用)
- 定时器和计数器的值
- 位存储的数据

在每个 CPU 上，可以存储一定数量的定时器、计数器和存储位，还可以指定一定数量的字节用以存储 DB 中包含的数据。

CPU 的 MPI 地址存储在 NVRAM 中。这就确保了 CPU 在掉电或存储器复位后可以进行通讯。

使用后备电池保护数据

使用后备电池，装载存储器和工作存储器在掉电期间是可以保持的：如果组态 CPU，使定时器、计数器和位存储器保存在 NVRAM 中，则不论是否使用了后备电池，这些信息都可以保持。

组态 NVRAM 的数据

当用 STEP 7 组态 CPU 时，可以决定哪个存储区域是可保持的。可以被组态到 NVRAM 中的存储器的数量根据所使用的 CPU 而定。所备份数量不能比 CPU 所指定的数目多。

A.2.3.9 S7-400 CPU 上的可保持存储区域

无后备电池的操作

如果在没有后备电池的情况下操作系统，当电源掉电出现时或复位 CPU 存储器(MRES)时，S7-400 CPU 的存储器(动态装载存储器(RAM)、工作存储器和系统存储器)被复位，这些区域中所包含的数据全部丢失。

没有后备电池，只能作暖起动并且没有可保持区域。电源掉电后，只有 MPI 参数(如 CPU 的 MPI 地址)被保留下来。这意味着在电源掉电或存储器复位后仍保持有通讯的能力。

有后备电池的操作

如果使用电池备份存储器：

- 电源掉电后，当CPU再起动时，所有RAM区域的全部内容都被保留。
- 在暖起动过程中，位存储器、定时器和计数器的地址区被清除。数据块的内容被保留。
- 除了被设定为非保持的位存储器、定时器和计数器之外，RAM工作存储器中的内容也都保留。

组态保持数据区域

可以声明一定数量的存储位、定时器和计数器具有保持功能(数量根据所使用的 CPU 而定)。在暖起动过程中，当使用后备电池时，这些数据也可保留。

当用 STEP 7 参数赋值时，可以定义哪些位存储器、定时器和计数器在暖起动时要保留。只能备份 CPU 所允许的数量的数据。

更多的关于定义可保持存储器区域的信息，可参考“S7-400，M7-400 可编程控制器，模板技术规范”参考手册。

A.2.3.10 在工作存储器中的可组态存储器对象

有些 CPU 可以在 HWConfig(硬件组态)中设置诸如局域或诊断缓存区对象的大小。例如，缺省值，在其它地方的工作存储器就可以获得一个较大的部分。这些 CPU 的设置可以在 CPU 模块属性“Memory(存储器)”标签中显示(“Details(详细数据)”按钮)。

在修改存储器组态并下载到可编程控制器以后，为使修改生效，必须执行一个冷起动。

A.3 数据类型和参数类型

A.3.1 介绍数据类型和参数类型

用户程序中的所有数据必须标有一个数据类型。可用的数据类型有以下几种：

- STEP 7提供的基本数据类型
- 通过组合基本数据类型而生成的复合数据类型
- 用来定义传送给FB或FC参数的参数类型

一般信息

语句表、梯形逻辑和功能块图指令使用特定大小的数据对象。位逻辑指令使用位。例如，装载和传送指令(STL)以及 move(移动)指令(LAD 和 FBD)使用字节、字和双字。

一个位是一个二进制数字“0”或“1”。一个字节由八个位组成，一个字十六位，一个双字三十二位。

数学指令也使用字节、字或双字。在这些字节、字或双字的地址中可以用各种格式编码，比如整数和浮点数。

当使用符号寻址时，定义符号并且要为这些符号指定一个数据类型(见下表)。不同的数据类型有不同的格式选项和数字表示法。

本章只描述了数字和常数的若干写法。下表列出一些数字和常数的格式不再作详细的解释。

格式	所占位的多少	数值表示法
Hexadecimal	8,16 和 32	B#16#,W#16#,and DW#16#
Binary	8,16 和 32	2#
IEC date	16	D#
IEC time	32	T#
Time of day	32	TOD#
Character	8	'A'

A.3.2 基本数据类型

每个基本数据类型有个规定的长度。下表列出了基本数据类型。

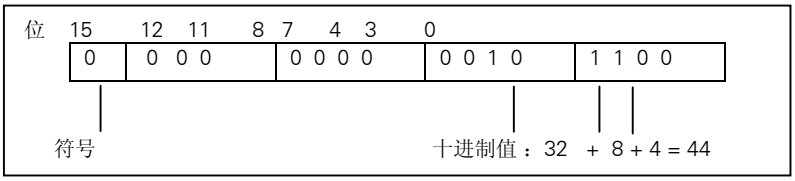
类型和描述	所 占位数	格式选项	范围及数值表示法（最低值至最高值）	示例
BOOL(位)	1	布尔文本	TRUE/FALSE	TRUE
BYTE (字节)	8	十六进制数	B 16#0 to B 16#FF	L B#16#10 L byte#16#10
WORD (字)	16	二进制数 十六进制数 BCD 无符号的十进制数	2#0 to 2#1111_1111_1111_1111 W#16#0 to W#16#FFFF C#0 to C#999 B#(0,0)to B#(255,255)	L 2#0001_0000_0000_0000 L W#16#1000 L word16#1000 L C#998 L B#(10,20) L byte(10,20)
DWORD (双字)	32	二进制数 十六进制数 BCD 无符号的十进制数	2#0 to 2#1111_1111_1111_1111 1111_1111_1111_1111 DW#16#0000_0000 to DW#16#FFFF_FFFF B#(0,0,0,0)to B#(255,255,255,255)	2#1000_0001_0001_1000_ 1011_1011_0111_1111 L DW#16#00A2_1234 L dword#16#00A2_1234 L B#(1,14,100,120) L byte#(1,14,100,120)
INT (整数)	16	有符号的十进制数	-32768 to 32767	L 1
DINT (整数，32位)	32	有符号的十进制数	L#-2147483648 to L#2147483647	L L#1
REAL (浮点数)	32	IEEE 浮点数	上限:±3.402823e+38 下限:±1.175495e-38	L 1.234567e+13
S5TIME (SIMATIC 时间)	16	S7 时间，每步10ms (缺省值)	S5T#0H_0M_0S_10MS to S5T#2H_46M_30S_0MS and S5T#0H_0M_0S_0MS	L S5T#0H_1M_0S_0MS L S5TIME#0H_1H_1M_0S_0MS
TIME (IEC时间)	32	IEC时间，每步1ms，带符号整数	-T#24D_20H_31M_23S_648MS to T#24D_20H_31M_23S_647MS	L T#0D_1H_1M_0S_0MS L TIME#0D_1H_1M_0S_0MS
DATE (IEC日期)	16	IEC 日期，每步 1天	D#1990-1-1 to D#2168-12-31	L D#1996-3-15 L DATE#1996-3-15
TIME_OF_ DAY (时间)	32	时间每步 1 ms	TOD#0:0:0.0 to TOD#23:59:59.999	L TOD#1:10:3.3 L TIME_OF_DAY#1:10:3.3
CHAR (字符)	8	ASCII 字符	‘A’,’B’etc.	L ‘E’

A.3.2.1 数据类型 INT 的格式(16 位整数)

一个整数有一个符号指示它是一个正整数还是一个负整数。一个整数(16 位)在存储器中所占空间为一个字。下表所示为一个整数的范围(16 位)。

格式	范围
整数(16位)	-32768至+32767

下图为整数+44 的二进制码的表示。

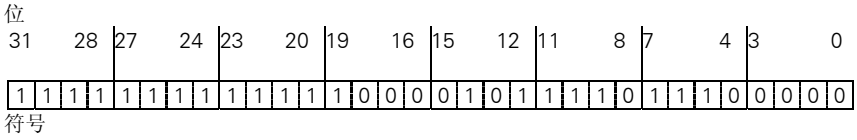


A.3.2.2 数据类型 DINT 的格式(32 位整数)

一个整数有一个符号指示它是一个正整数还是一个负整数。一个双整数占用的存储空间为两个字，下表所示为一个双整数的范围。

格式	范围
整数(32位)	-2.147,483,648至+2,147,483,647

下图所示为整数-500,000 的二进制码的表示。在二进制系统中，整数的负值形式表示为这个整数的二维补码。可以通过对一个整数的各位取反然后加 1 得到二维补码。



A.3.2.3 数据类型 REAL 的格式(浮点数)

浮点数格式的表达通式为“数值=m*E 的 b 次幂”。基数“b”和指数“E”是整数；尾数“m”是有理数。

这种数值表达方式的优点是在有限的空间内即可以表示一个非常大的数也可以表示一个非常小的数。使用有限位数的尾数和指数，可以覆盖一个较宽范围内的数字。

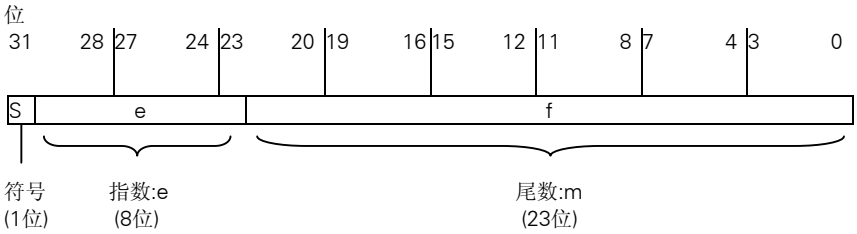
缺点则在于对计算的精确度的限制。例如，当求两个数的和时，必须通过移动尾数(因此浮动小数点)使指数对阶，因为只有具有相同指数的数值可以相加。

STEP 7 中的浮点数格式

STEP 7 中的浮点数符合二进制浮点运算 IEEE 标准中 ANSI/IEEE 标准 754-1985 有关基本格式、单宽的描述。它们包括以下部分：

- 符号S
- 指数e=E+bias(偏移)，通过一个常数(偏移=+127) 增加
- 尾数m的小数部分。尾数的整数部分不和余数存在一起，因为在有效数值范围内它总是等于1。

这三个部分一共占用一个双字(32 位)：



下图所示为一个浮点数格式各个位数值。

浮点数的组成元素	位号	数值
符号S	31	
指数e	30	2的7次幂
…	…	…
指数e	24	2的1次幂
指数e	23	2的0次幂
尾数m	22	2的-1次幂
…	…	…
尾数m	1	2的-22次幂
尾数m	0	2的-23次幂

使用三个组成部分 **s**，**e** 和 **m**，这种形式的数值可以由以下公式定义：

数值=1.**m***2 的(e bias)次幂，这里：

- e：1≤e≤254
- Bias：bias=127。这意味着指数不再另外需要一个符号。
- S：正数S=0，负数S=1。

浮点数的数值范围

使用以上所示的浮点数格式，有以下结果：

- 最小的浮点数=1.0*2的(1-127)次幂=1.0*2的(-126)次幂=1.175495E-38
- 最大的浮点数=2-2的(-23)次幂*2的(254-127)次幂=2-2的(-23)次幂*2的(+127)次幂=3.402823E+38

零值表示为 e=m=0；e=255 和 m=0 表示“infinite(无穷大)”。

格式	范围[1)]
符合ANSI/IEEE标准的浮点数	-3.402 823E+38至-1.175 495E-38和0以及 +1.175 495E-38至3.402 823E+38

下图所示为使用了非有效范围内的浮点数的指令结果的状态字中各位的信号状态：

无效的结果范围	CC1	CC0	OV	OS
-1.175494E-38<结果<-1.401298E-45(负数)下溢	0	0	1	1
+1.40298E-45<结果<+1.175494E-38(正数)下溢	0	0	1	1
结果<-3.402823E+38(负值)上溢	0	1	1	1
结果>3.402823E+38(正值)上溢	1	0	1	1
不是有效的浮点数或者无效指令（输入值在有效范围之外）	1	1	1	1

使用数字操作时请注意：

例如，当试图对-2 求平方根时，会得到“非有效的浮点数”的结果。所以，在开始基于某个结果的基础上要继续计算前，总是要首先评估状态位。

修改变量时请注意：

例如，将一个浮点数数值存储在一个存储双字时，可以以位的形式对这些数作修改，可是，并非所有的位模式都是一个有效数字。

浮点数计算的精确度

注意

涉及包含非常大和非常小的一长串数值的计算时，可能会产生不精确的结果

在 STEP 7 中浮点数可以精确到 6 个十进制位。因此，当你输入浮点数常数时，最多只能指定 6 位。

提示

计算的精确度为 6 位意味着，例如，如果数值 1 大于数值 2*10 的 y 次幂，这里 y>6 则数值 1+数值 2=数值 1

100 000 000+1=100 000 000

浮点数格式的数字示例

下图所示为以十进制值值表示的浮点数格式：

- 10.0
- P(3.141593)
- 2的平方根(P2=1.414214)

第一个示例中的数值 10.0 的浮点数格式(十六进制表示：4120 0000)如下：

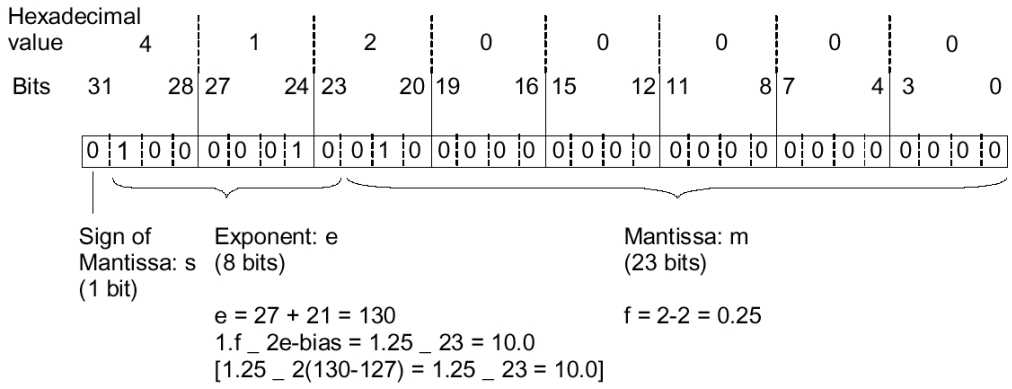
e=2 的 1 次幂+2 的 7 次幂=2+128=130

m=2 的(-2)次幂=0.25

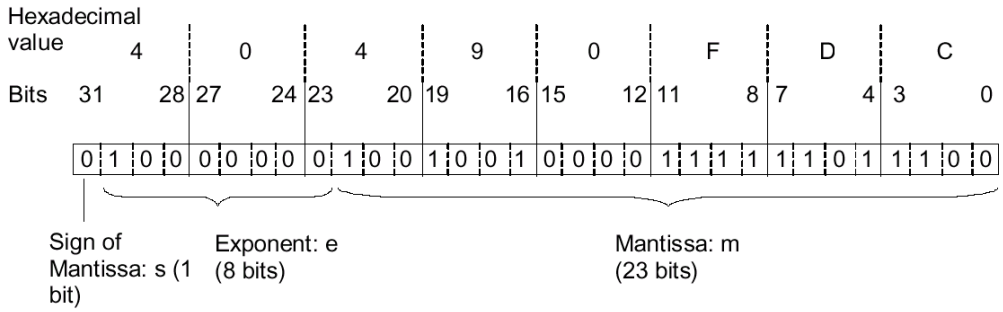
这样就可得到：

(1+m)*2 的(e-bias)次幂=1.25*2 的(130-127)次幂=1.25×2 的 3 次幂=10.0

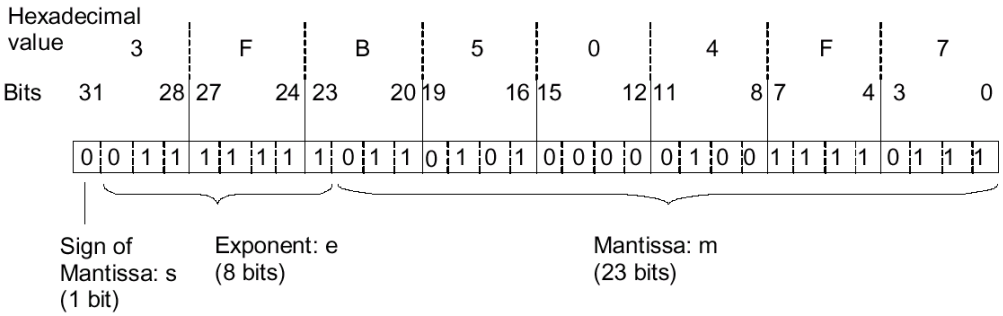
Decimal value 10.0



Decimal value 3.141593



Decimal value 1.414214



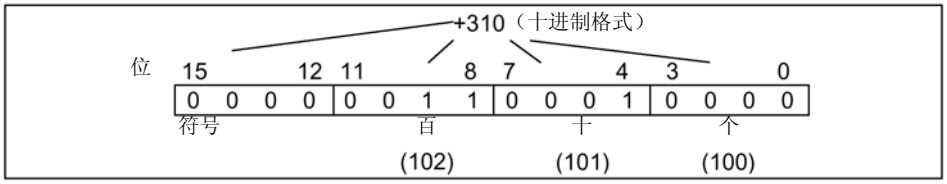
A.3.2.4 数据类型为字和双字的二进制编码的十进制数的格式

二进制编码的十进制数(BCD)格式是用一组二进制数(位)来表达一个十进制数。每 4 个位一组表示一个带符号的十进制数或十进制数的符号。几个 4 位组形成一个字(16 位)或双字(32 位)。最高有效位的 4 位指示数值和符号(1111 指示减号，0000 指示加号)使用 BCD 编码地址的命令只评估最高位(字的 15 位，双字的 31 位)。下表所示为这两种 BCD 码的格式和范围。

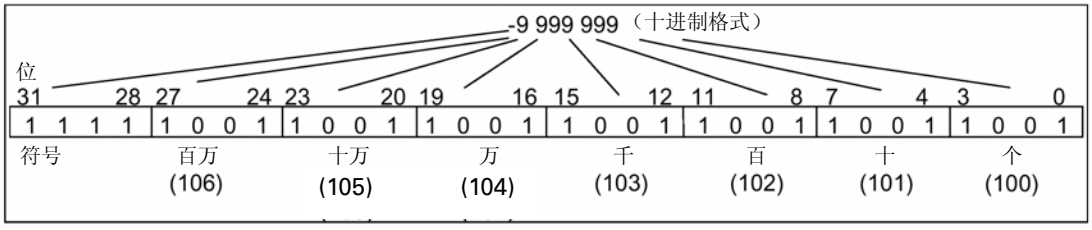
格式	范围
字(Word) (16位，3位BCD码及符号)	-999至+999
双字(Double word) (32位，7位BCD码及符号	-9 999 999至+9 999 999

下图是二进制编码的十进制数的示例：

- 字格式

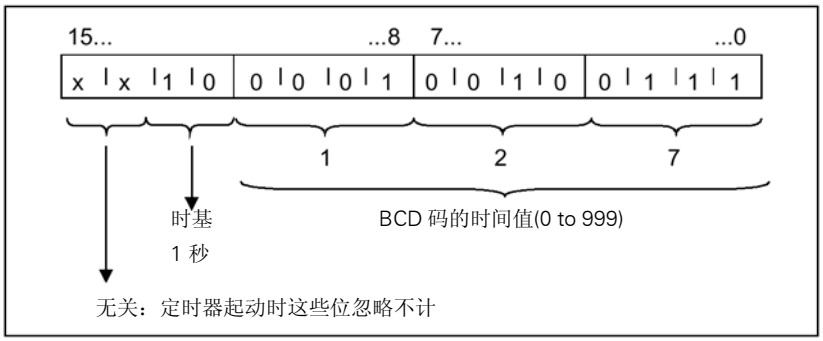


- 双字格式



A.3.2.5 数据类型 S5TIME(时间长度)的格式

当使用 S5TIME 数据类型输入时间长度时，输入以二进制编码的十进制格式存储。下图所示的时间值为 127，时基为 1s 的时间地址中的内容。



当使用 S5TIME 时，要输入一个 0-999 范围内的时间值并指明时基(见下表)。时基指示定时器以什么样的间隔一个单位一个单位地递减时间值，直至达到 0。

S5TIME 的时基

时基	时基的二进制码
10 ms	00
100 ms	01
1 s	10
10 s	11

可以使用下列语法格式预装一个时间值：

- L[1)]W#16#wxyz
  - 这里 w=时基(即时间间隔或分辨率)
  - 这里 xyz=二进制编码的十进制格式的时间值
- L[1)] S5T#aH_bbM_ccS_dddMS
  - 这里 a=小时，bb=分钟，cc=秒，dd=毫秒
  - 时基自动选择，时间值按其所取时基取整为下一个较小的数。

可以输入的最大时间值是 9，990 秒，或 2H_46M_30S。

[1)] =L 只能用 STL 编写。

A.3.3 复合数据类型

复合数据类型可以定义大于 32 位的数据记录或包括其它数据类型的数据记录。STEP 7 允许以下复合数据类型：

- DATE_AND_TIME（日期和时间)
- STRING（字符串)
- ARRAY（数组)
- STRUCT（结构体)
- UDT（用户定义的数据类型)
- FB和SFB

下表描述了复合数据类型。可以在逻辑块的变量声明中或数据块中定义结构体和数组。

数据类型	描述
DATE_AND_TIME DT	定义一个64位(8个字节)的区域。这种数据类型以二进制编码的十进制格式存储：
STRING	缺省定义一个最多可有254个字符（数据类型CHAR）的组。存储字符串的标准区域长256字节。这是存储254个字符及2字节首部所需要的空间，可以定义需要存储在该字符串中的字符的数量来减少一个字符串所需求的存储器。（如：string[9]'Siemens"）
ARRAY	定义同一数据类型（基本的或复合的）的多维组。例 如："ARRAY[1..2,1..3]OF INT"定义了一个2×3格式的整数数组。可以使用下标（"[2,2]"）访问存储在一个数组中的数据。可以在一个数组中最多定义6维。下标可以是任意整数(-32768到+32767)
STRUCT	定义一组任意的数据类型的组合。例如，可以定义一个由结构体组成的数组或一个由结构体和数组组成的结构体
UDT	当生成数据块或在变量声明中声明变量时，可以简化大量数据的构造和数据类型的输入。在STEP 7中，可以组合复合和基本数据类型以生成"用户定义的"数据类型。UDT有它们自己的名字并可以多次使用
FB,SFB	决定背景数据块的结构并允许在一个背景DB中传送几个FB调用的背景数据

结构体数据类型存储时与字地址的限制一致(WORD 排列)。

A.3.3.1 数据类型 DATE_AND_TIME 的格式

当使用 DATE_AND_TIME 数据类型(DT)输入日期和时间时，输入以二进制编码的十进制格式存在 8 个字节中。DATE_AND_TIME 数据类型范围如下：

DT#1990-1-1-0:0:0.0 至 DT#2089-12-31-23:59:59.999

下列所示为 1993 年 12 月 25 日星期四上午 8 点 01 分 1.23 秒的日期和时间语法，可有以下两种格式：

- DATE_AND_TIME#1993-12-25-8:01:1.23
- DT#1993-12-25-8:01:1.23

以下特殊的 IEC(国际电工技术委员会)标准功能可以对 DATE_AND_TIME 数据类型数据进行槽：

- 将某一天的日期和时间转换为DATE_AND_TIME数据格式： FC3：D_TOD_DT
- 从DATE_AND_TIME格式中提取日期： FC6：DT_DATE
- 从DATE_AND_TIME格式中提取星期： FC7：DT_DAY
- 从DATE_AND_TIME格式中提取日时钟： FC8：DT_TOD

下表所示为包含日期和时间格式字节中的内容，该日期和时间包含以下信息：星期四，12 月 25 日，1993，早晨 8:01 又 1.23 秒。

字节	内容	举例
0	年	B#16#93
1	月	B#16#12
2	日	B#16#25
3	小时	B#16#08
4	分	B#16#01
5	秒	B#16#01
6	毫秒的高两位	B#16#23
7 (4MSB)	毫秒的低两位	B#16#0
7 (4LSB)	星期 1=周日 2=周一 … 7=周六	B#16#5

数据类型 DATE_AND_TIME 允许的范围是：

- 最小：DT#1990-1-1-0:0:0.0
- 最大：DT#2089-12-31-23:59:59.999

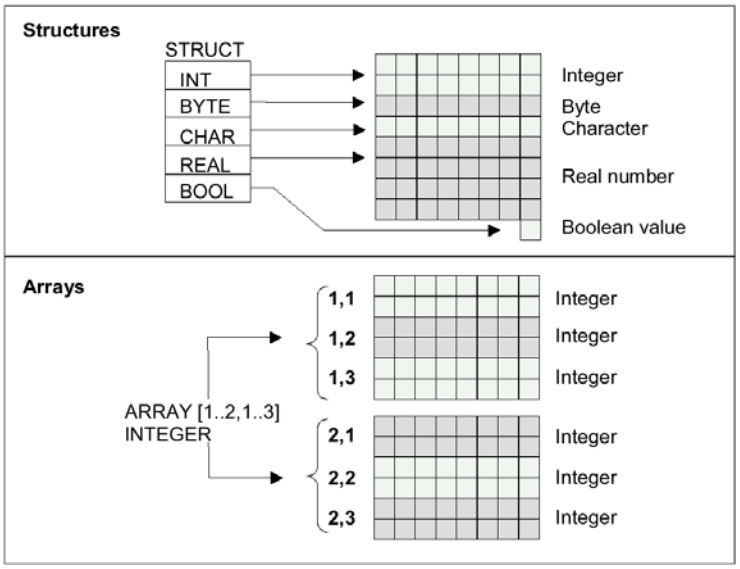
	可能的数据范围	BCD码
年	1990-1999 2000-2089	90h-99h 00h-89h
月	1-12	01h-12h
日	1-31	01h-31h
小时	00-23	00h-23h
分	00-59	00h-59h
秒	00-59	00h-59h
毫秒	00-999	000h-999h
星期	Sunday-Saturday	1h-7h

A.3.3.2 使用复合数据类型

可以通过组合基本和复合数据类型生成以下复合数据类型，从而生成新的数据类型：

- 数组(数据类型ARRAY)：一个数组是组合一组同一类型的数据形成一个单元。
- 结构体(数据类型STRUCT)：一个结构体组合不同的数据类型形成一个单元。
- 字符串(数据类型STRING)：一个字符串是义了一个最多有254字符(数据类型CHAR)的一维数组。字符串的长度必须与块的形参和实参相匹配。
- 日期和时间(数据类型DATE_AND_TIME)：日期和时间数据类型存储年、月、日、小时、分、秒、毫秒和星期。

下图所示说明数组和结构体是如何在一个区域内构造数据类型并存储信息的。可以在一个 DB 或一个 FB、OB 或 FC 的变量声明中定义一个数组或一个结构体。



A.3.3.3 使用数组访问数据

数组

一个数组组合一组同一类型的数据(基本类型或复合类型)形成一个单元。可以生成一个包含数组的数组，当定义一个数组时，必须作以下各项：

- 给数组指定一个名字。
- 用关键字ARRAY声明一个数组。
- 用下标指定数组的大小，在数组中指定每一维(最多6维)的第一个和最后一个数字。在一个方括号中输入下标，每一维之间用逗号隔开，该维的第一个数和最后一个数之间是两个点号。例如，下列下标定义了一个三维数组：

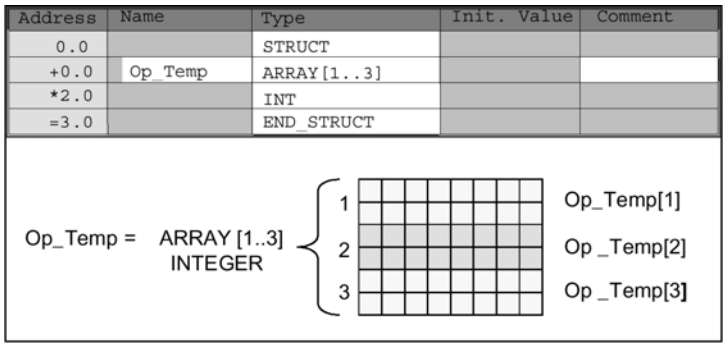
[1..5，-2..3，30..32]

- 指定数组中包含的数据的数据类型。

示例：1

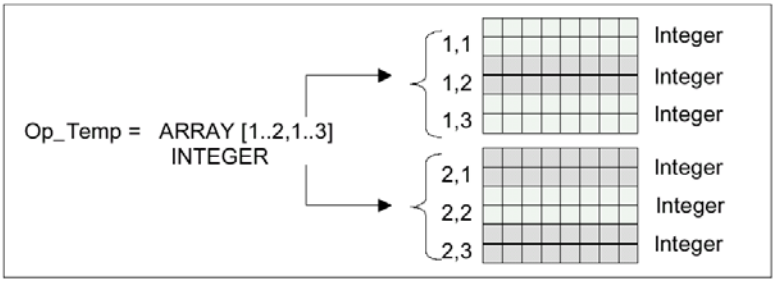
下图所示是一个有三个整数的数组。使用下标可以访问存储在数组中的数据。下标就是方括号中的数字。例如，第二个整数的下标是 Op_temp[2]。

下标可以是包括负值在内的任意整数(-32768 到+32767)。下图中的数组也可以定义为 ARRAY[-1..1]。第一个整数的下标则为 Op_temp[-1]，第二个是 Op_temp[0]第三个整数则是 Op_temp[1]。



示例 2

一个数组还可以描述多维组的数据类型。下图所示为整数(integer)的二维数组。



可以使用下标访问多维数组的数据。在本例中，第一个整数是 Op_Temp[1，1]，第三个是 Op_Temp[1，3]，第四个是 Op_Temp[2，1]，第六个是 Op_Temp[2，3]。

一个数组最多可以定义到六维。例如，可以按如下所示将变量 Op_Temp 定义为六维数组：

ARRAY[1..3，1..2，1..3，1..4，1..3，1..4]

这个数组的第一个元素的下标是 Op_Temp[1，1，1，1，1，1]。最后一个元素的下标为 Op_Temp[3，2，3，4，3，4]。

生成数组

在 DB 块中或变量声明表中可以定义数组。在创建数组时要指定关键字(ARRAY)及其后方括号中的大小，如下所示：

[低限值..高限值]

在多维数组中还要指定其它的高限和低限值并用逗号隔开各维。下图所示为生成一个 2×3 数组的声明。

Address	Name	Type	Init. Value	Comment
0.0		STRUCT		
+0.0	Heat_2x3	ARRAY[1..2,1..3]		
*2.0		INT		
=6.0		END_STRUCT		

为数组输入初始值

当生成数组时可以为每个数组元素赋予一个初始值。STEP 7 提供两种输入初始值的方法：

- 单个值的输入：可以为每个数组元素指定一个对于数组数据类型是有效的数值。要按元素的顺序指定数值：[1，1]。记住每个元素必须用逗号彼此隔开。
- 指定重复系数：使顺序的元素具有相同的初值，可以为这些元素指定元素个数(重复系数)和初始值。输入重复系数的格式为X(y)，这里X是重复系数，y是要重复的数值。

如果使用上图中的数组声明，可以按如下所示为所有的六个元素指定初始值：17，23，-45，556，3342，0。也可以通过指定 6(10)将所有六个元素的初始值设为 10。还可以为前两个元素指定特定的值而将其余的四个元素指定为 0，如：17，23，4(0)。

访问数组中的数据

可以通过数组中特定元素的下标访问数组中的数据。下标和符号名一起使用。

例如：上图中声明的数组从 DB20(motor)的第一个字节开始，可以用如下地址访问数组的第二个元素：

Motor.Heat_2×3[1，2]。

使用数组作为参数

可以将数组作为参数进行传送。如果一个参数在变量声明中被声明为数组，则必须传送整个数组(不是单个元素)。而数组中的一个元素可以在调用块时赋值给一个参数，只要数组元素与参数的数据类型相应。

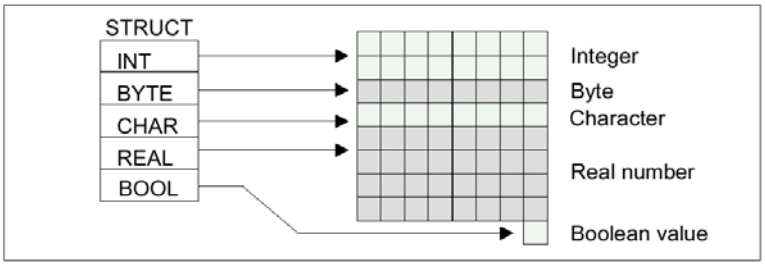
如果使用数组作为参数，数组无需与参数同名(它们甚至不需要名字)。而两个数组(形参和实参)的结构必须相同。例如，一个格式为 2×3 的整数数组，如果一个块的形参为 2×3 的整数数组，那么赋值的实参也必须为 2×3 的整数数组。

A.3.3.4 用结构体访问数据

结构体

结构体是组合各种数据类型(基本和复合数据类型，包括数组和结构体)形成一个单元。可以组合数据以适合过程控制。因此，可将参数作为一个数据单元来传送而不是单个元素。下图

所示为一个结构体，它包括一个整数、一个字节、一个字符、一个浮点数和一个布尔值。



一个结构体最多可有八层嵌套(例如，一个结构体包括包含数组的结构体)。

生成结构体

当在 DB 中或在一个逻辑块的变量声明中定义结构体。

下图是一个结构体(Stack_1)的声明，该结构体包括以下元素：一个整数(用于存储总量)，一个字节(用于存储原始数据)，一个字符(用于存储控制码)，一个浮点数(用于存储温度)以及一个布尔存储位(用于结束信号)。

Address	Name	Type	Init,Value	Comment
0.0	Stack_1	STRUCT		
+0.0	Amount	INT	100	
+2.0	Original_data	BYTE		
+4.0	Control_code	CHAR		
+6.0	Temperature	REAL	120	
+8.1	End	BOOL	FALSE	
=10.0		END_STRUCT		

为一个结构体分配初始值

如果要给一个结构体的每个元素分配初始值，需要指定一个该数据类型的有效值以及元素的名字。例如，分配以下的初始值(给上图中声明的结构体)：

```
Amount          =    100
Original_data   =    B#(0)
Control_Code    =    `C'
Temperature     =    120
End             =    False
```

在结构体中存储和访问数据

可以访问结构体中的单个元素。可以使用符号地址(如，Stack_1.Temperature)。也可以指定元素所在的绝对地址(例如，如果 Stack_1 位于 DB20 的起始字节 0，amount 的绝对地址是 DB20.DBW0，temperature 的绝对地址是 DB20.DBD6)。

用结构体作为参数

可以传送作为参数的结构体。如果在变量声明中一个参数被声明为 STRUCT(结构体)，必

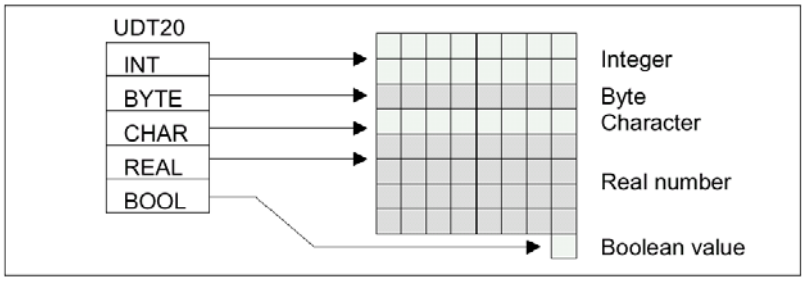
须为它传送一个具有相同元素的结构体。而一个结构体中的元素也可以被赋值给所用块的参数，只要结构体中的元素与参数的数据类型相符。

如果使用结构体作为参数，两个结构体(形参和实参)必须具有相同的元素，也就是说相同的数据类型按相同的顺序排列。

A.3.3.5 使用用户定义的数据类型访问数据

用户定义的数据类型

用户定义的数据类型(UDT)可以包括基本的和复合的数据类型。可以为 UDT 指定一个名字并且可以不止一次地使用它们。下图说明一个用户定义的数据类型的结构，它包括一个整数、一个字节、一个字符、一个浮点数和一个布尔值。



不用单个地或作为一个结构体输入所有的数据类型，只需要指定“UDT20”作为一个数据类型，STEP 7 自动地分配相应的存储空间。

生成一个用户定义的数据类型

用 STEP 7 可以定义 UDT。下图所示是一个 UDT，它包含以下元素：

一个整数(用于存储总量)、一个字节(用于存储原始数据)、一个字符(用于存储控制码)、一个浮点数(用于存储温度)以及一个布尔存储位(用于结束信号)。你可以在符号表中给 UDT 赋予一个符号名(如，process data)。

Address	Name	Type	Init,Value	Comment
0.0	Stack_1	STRUCT		
+0.0	Amount	INT	100	
+2.0	Original_data	BYTE		
+4.0	Control_code	CHAR		
+6.0	Temperature	REAL	120	
+8.1	End	BOOL	FALSE	
=10.0		END_STRUCT		

一旦生成了一个 UDT，可以用 UDT 作为一个数据类型，例如，在一个 DB(或在一个 FB 的变量声明)中一个变量声明数据类型 UDT200。

下图所示是一个含有变量 process_data_1 的 DB，该变量的数据类型是 UDT200。只需要指定 UDT200 和 process_data_1。当编译该 DB 时，显示为斜体的数组被生成。

Address	Name	Type	Init,Value	Comment
0.0		STRUCT		
+~~6~~10.0	Process_data_1	UDT200		
=10~~6~~.0		END_STRUCT		

为用户定义的数据类型分配初值

如果要给用户定义的数据类型的每个元素分配初值，需要指定一个该数据类型的有效值以及元素的名字。例如，可以分配以下初值(给上图中声明的用户定义的数据类型)：

Amount	=	100
Original_data	=	B#16#0)
Control_code	=	`C'
Temperature	=	1.2000000+002
End	=	False

如果声明了一个变量为 UDT，这个变量的初值就是生成 UDT 时指定的数值。

存储和访问用户定义的数据类型

可以访问一个 UDT 的每一个元素。可以使用符号地址(如，Stack_1. Temperature) 。也可以指定元素所在的绝对地址(例如，如果 Stack_1 位于 DB20 的起始字节 0，则 amount 的绝对地址是 DB20.DBWO 而 temperature 是 DB20.DBD6)。

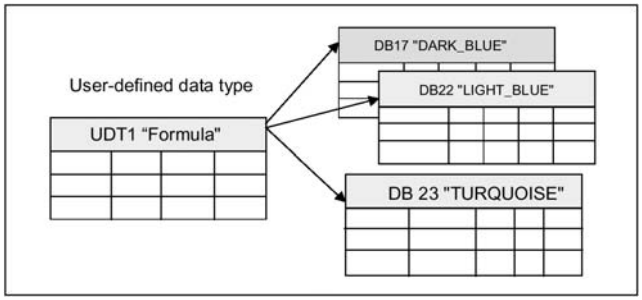
使用用户定义的数据类型作为参数

可以传送作为参数的 UDT 变量。如果一个参数在变量声明中声明为 UDT，必传送一个具有相同结构的 UDT。而当调用一个块的时候，UDT 的一个元素也可以赋值给一个参数，只要 UDT 的元素与参数的数据类型相符。

使用选定的 UDT 的 DB 的优点

使用一个一次生成的 UDT，可以生成许多具有相同数据结构的数据块。然后可以使用这些数据块为特定的任务存入不同的实际值。

例如，为一个配方(如调和颜色)构造一个 UDT，可以将这个 UDT 指定给几个 DB，每个包含不同数量。



数据块的结构由指定给定的 UDT 决定。

A.3.4 参数类型

除了基本和复合数据类型之外，还可以为块之间传递的形参定义参数类型。STEP 7 可以识别以下参数类型：

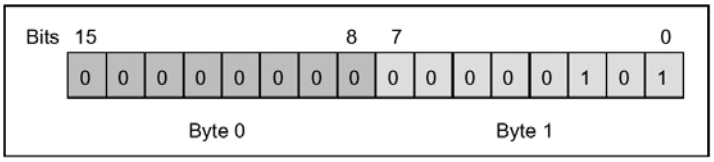
- TIMER(定时器)或COUNTER(计数器)：这里指定块执行时将使用一个特定的定时器或计数器。如果将TIMER或COUNTER参数类型作为形参，则相应的实参也必须是一个定时器或一个计数器，换句话说，输入一个后面跟着正整数的“T”或“C”。
- BLOCK(块)：指定一个特定的块用作输入或输出。由参数的声明决定使用的块的类型(FB、FC、DB等)。如果为BLOCK参数类型的形参提供数值，要指定一个块地址作为实参。例如“FC101”(当使用绝对地址时)或“Value”(使用符号地址)。
- POINTER(指针)：指向一个变量的地址。指针包含一个地址而不是一个数值。当为一个PONITER参数类型的形参提供数值时，必须指定一个地址作为实际参数。在STEP 7中，可以用指针的格式来指定一个指针，也可以简单地用一个地址来指定(如，M50.0)。以指针的格式寻址从M50.0开始的数据可以写为：P#M50.0
- ANY：这一类型用于实参的数据类型不知道或可以是任何数据类型的情形。要了解关于ANY参数类型的更多的信息，可参考“参数类型ANY的格式”以及“使用参数类型ANY”。

参数类型也可用于用户定义的数据类型(UDT)。有关 UDT 的更多的信息参考”使用用户定义的数据类型访问数据。”

参数	容量	描述
TIMER	2字节	指示在被调用的逻辑块中由程序使用的定时器 格式： T1
COUNTER	2字节	指示在被调用的逻辑块中由程序使用的计数器 格式： C10
BLOCK_FB BLOCK_FC BLOCK_DB BLOCK_SDB	2字节	指示在被调用的逻辑块中由程序使用的块 格式： FC101 DB42
POINTER	6字节	标识地址 Format: P#M50.0
ANY	10字节	用于当前参数的数据类型是未知的 格式： P#M50.0 BYTE 10 数据类型是ANY格式 P#M100.0 WORD 5 L#1COUNTER 10 参数类型是ANY格式

A.3.4.1 参数类型 BLOCK、COUNTER、TIMER 的格式

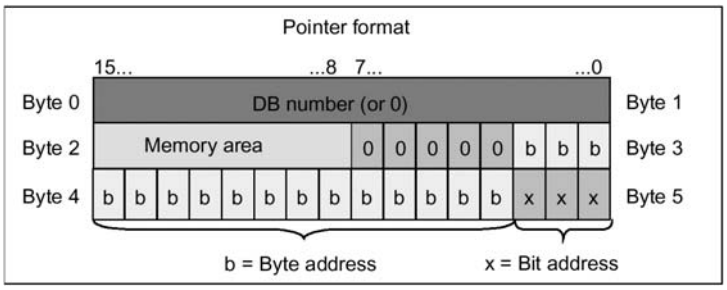
STEP 7 将参数类型 BLOCK、COUNTER 和 TIMER 作为二进制数存在一个字中(32 位)。下图所示为这些参数类型的格式。



允许使用的块、定时器和计数器的数量根据 S7 CPU 的类型而定。在“S7-300 可编程控制器、硬件和安装手册”或“S7-400，M7-400 可编程控制器，硬件和安装手册”的数据页中能够找到更多关于 CPU 能够使用的定时器、计数器的数目以及可用块的最大数目的信息。

A.3.4.2 参数类型 POINTER 的格式

下图所示为该类型的数据在每一个字节中的存储。



参数类型 POINTER 存储以下信息：

- DB号码(如果数据未存储在DB块中则为0)
- CPU中的存储区域(下表所示为参数类型POINTER的存储区域的十六进制代码)

十六进制代码	存储区域	描述
B#16#81	I	输入区域
B#16#82	Q	输出区域
B#16#83	M	位存储区域
B#16#84	DB	数据块
B#16#85	DI	背景数据块
B#16#86	L	局域数据(L堆栈)
B#16#87	V	上一个块调用使用的局域数据

- 数据的地址(字节.位的格式)

STEP 7 提供指针格式：P#memory_area byte.bit_address(P#存储区域 字节.位地址)(如果形参被声明为参数类型 POINTER，只需指出存储区域和地址。STEP 7 会自动将输入转换为指针的格式)。下面的例子说明如何为从 M50.0 开始的数据输入参数类型 POINTER：

- P#M50.0
- M50.0(如果形参声明的POINTER)。

A.3.4.3 使用参数类型 POINTER

指针用于指向一个地址。这种寻址方式的优势在于可以在程序执行过程中动态地修改语句的地址。

指针用于存储器间接寻址

用存储器间接寻址的程序语句由一个指令，一个地址标识和一个偏移量组成。(该偏移量必须在方括号中给出)。

双字格式的指针举例：

L	P#8.7	将指针数值装入累加器1
T	MD2	将指针传入MD2
A	I[MD2]	查询输入位I8.7的信号状态并且
=	Q[MD2]	将信号状态赋给输出位Q8.7

指针用于区域内部和区域交叉寻址

用这种寻址类型的程序指令由一个指令和以下部分组成：地址标识、地址寄存器标识符、偏移量。地址寄存器(AR1/2)和偏移量必须在方括号内一起被指定。

区域内部寻址示例

指针中不包含存储区域的指示：

L	P#8.7	将指针数值装入累加器1
LAR1		将指针从累加器1装入AR1
A	I[AR1,P#0.0]	查询输入位I8.7的信号状态并且
=	Q[AR1,P#1.1]	将信号状态赋给输出位Q10.0

偏移量 0.0 不产生任何影响，输出 10.0 由 8.7(ARI)加偏移量 1.1 计算得出。结果为 10.0 而不是 9.8，见指针格式。

区域交叉寻址示例

在区域交叉寻址中指针指示存储区域(在示例中为 I 和 Q)。

L	P#I8.7	装载指针值和区域标识到累加器1
LAR1		装载存储区域I和地址8.7到AR1
L	P#Q8.7	装载指针值和区域标识到累加器1
LAR2		装载存储区域Q和地址8.7到AR2
A	[AR1,P#0.0]	查询输入位I8.7的信号状态并且
=	[AR1,P#1.1]	将信号状态赋给输出位Q8.7

偏移量 0.0 不产生影响。输出 10.0 由 8.7(AR2)加偏移量 1.1 计算得出。结果是 10.0 而不是 9.8，见指针格式。

A.3.4.4 用于修改指针的块

使用示例块 FC3“Routing Pointers”可以修改一个指针的位地址或字节地址。当该 FC 被调用时需要修改的指针传送给变量“Pointer”(可以使用双字格式的区域内部和区域交叉指针)。

用参数“Bit-Byte”可以修改指针的位或字节地址(0：位地址，1：字节地址)。变量“Inc-Value”(整数格式)指定要从地址内容中加上或减去的数值。也可以指定负值来减小地址。

改变位地址，会对字节地址产生进位(或在减少情况下)，例如：

- P#M5.3，Bit_Byte=0，Inc_Value=6 => P#M6.1或
- P#M5.3，Bit_Byte=0，Inc_Value=-6 => P#M4.5

指针的区域信息不受该功能影响。

FC 截取指针的上溢/下溢。这种情况下指针不变，输出变量“RET_VAL”(可以错误处理)设为“1”(直至下一个 FC3 的正确处理)。这里：

- 1.位地址被选中并且Inc_Value>7，或<-7
- 2.位或字节地址被选中并且修改会造成一个“负”字节地址
- 3.位或字节地址被选中，修改会造成一个非法的过大的字节地址。

修改指针的 STL 示例块

```
FUNCTION FC 3:BOOL
TITLE=Routing Pointers
//FC3 可用于修改指针
AUTHOR：AUTICS1
FAMILY:INDADDR
NAME:ADDRPOINT
VERSION:0.0

VAR_INPUT
```

```
    Bit_Byte:BOOL://0:位地址 1：字节地址
    Inc_Value:INT；//增量(如果数值为负=>减小/如果数值为正=>增加)
END_VAR

VAR_IN_OUT
    pointer:DWORD;//要修改的指针
END_VAR
VAR_TEMP
    inc_Value1:INT;//中间值增量
    pointer1:DWORD；//中间值指针
    Int_Value:DWORD;//辅助变量
END_VAR
BEGIN
NETWORK
TITLE=
//该块截取对指针区域信息的修改
//或自动转为负指针
    SET   ;//置 RLO 为 1 并且
    R     #RET_VAL;//复位溢出
    L     #Pointer;//为临时变量提供数值
    T     #Pointer1;//中间值指针
    L     #Inc_Value;//为临时变量提供数值
    T     #Inc_Value1;//中间值增量
    A     #Bit_Byte;//如果=1,字节地址指令
    JC    Byte;//跳转到字节地址的计算
    L     7;//如增量值>7
    L     #Inc_Value1;
    <I    ;
    S     #RET_VAL;//置位 RET_VAL 并且
    JC    End;//跳到 End
    L     -7;//如果增量值<-7
    <I    ;
    S     #RET_VAL;//置位 RET_VAL 并且
    JC    End;//跳转到 End
    A     L     1.3；//如果该值的位 4=1(Inc_Value 为负)
    JC    neg;//则跳转到位地址的减法计算
    L     #Pointer1;//L 装载指针地址信息
    L     #Inc_Value1;//加上增量
    +D    ；
    JU    test；//跳转到对负结果的测试
neg：     L     #Pointer1;//装载指针地址
    L     #Inc_Value1;//装载增量
    NEG1 ;//对负值取反
```

```
    -D      ；//减去增量值
    JU      test;//跳转到测试
Byte:       L      0;//开始字节地址修改
    L       #Inc_Value1;//如果增量>=0 则
    <I      ;
    JC      pos;//跳转到加法，否则
    L       #Pointer1;//装载指针地址信息
    L       #Inc_Value1;//装载增量
    NEGI；//对负值取反
    SLD   3;//将增量向左移 3 位
    -D      ；//减去该数值
    JU      test;//并跳转到测试
pos:        SLD  3;//增量左移三位
    L       #Pointer1;//装载指针地址信息
    +D      ;//加增量
test: T     #Int_Value;//将计算结果传至 Int_Value.
    A       L 7.3;//如果字节地址无效(太大或为负)，
    S       #RET_VAL;//则置位 RET_VAL
    JC      End;//并跳转到 End，
    L       #Int_Value;//否则传送结果
    T       #Pointer;//到指针
End:        NOP  0;
END_FUNCTION
```

A.3.4.5 参数类型 ANY 的格式

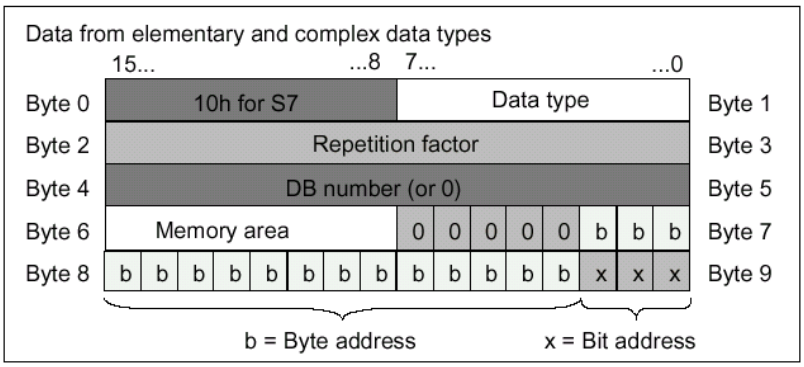
STEP 7 用 10 个字节(80 个位)来存储参数类型 ANY。当构造 ANY 参数类型时，由于被调用的块要评估该参数的整个内容，必须确保所有的 80 个位都被占用。例如，如果在字节 4 指定一个 DB 号码，还必须在字节 6 直接指定存储区域。

STEP 7 对基本数据和复合数据类型的管理与参数类型的数据不同。

ANY 格式的数据类型

对基本和复合数据类型，STEP 7 存储以下数据：

- 数据类型
- 重复系数
- DB号码
- 保存信息的存储区域
- 数据的起始地址



重复系数指明由参数类型 ANY 要传送的所指数据类型的数量。这意味着可以指定一个数据区域，并且用数组和结构体与参数类型 ANY 连接。STEP 7 将数组和结构体作为以字节为单位的数来识别(在重复系数的帮助下)。例如，如果要传送 10 个字，必须输入 20(字节)作为重复系数。

地址以字节.位的格式存储，其中字节地址存在字节 7 的 0 位至 2 位，字节 8 的 0 位至 7 位以及字节 9 的 3 位至 7 位。位地址存储在字节 9 的 0 位至 2 位。

对于类型为 NIL 的空指针，从字节 1 开始的所有字节都赋值为 0。

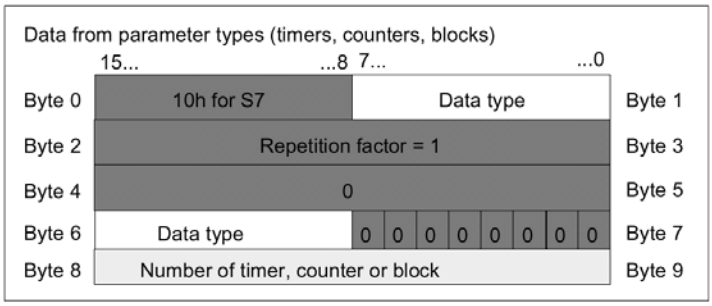
下表所示为参数类型 ANY 数据类型与 CPU 存储器的编码。

十六进制码	数据类型4码	描述
B#16#00	NIL	Null pointer空指针
B#16#01	BOOL	Bits位
B#16#02	BYTE	Bytes(8 bits)字节（8位）
B#16#03	CHAR	Characters(8 bits)字符（8位）
B#16#04	WORD	Words(16 bits)字（16位）
B#16#05	INT	Integers(16 bits)整数（16位）
B#16#06	DWORD	Words(32 bits)字（32位）
B#16#07	DINT	Double integers(32 bits)双整数（32位）
B#16#08	REAL	浮点数（32位）
B#16#09	DATE	Date日期
B#16#0A	TIME_OF_DAY(TOD)	Time of day日时钟
B#16#0B	TIME	Time时间
B#16#0C	S5TIME	Data type S5TIME数据类型S5TIME
B#16#0E	DATE_AND_TIME(DT)	Date and time(64 bits)日期和时间（64位）
B#16#13	STRING	String字符串

十六进制码	数据类型4码	描述
B#16#81	I	输入区
B#16#82	Q	输出区
B#16#83	M	位存储区
B#16#84	DB	数据块
B#16#85	DI	背景数据块
B#16#86	L	局部数据(L栈)
B#16#87	V	上一个块调用使用的局域数据

ANY 格式的参数类型

STEP 7 为参数类型存储数据类型和参数地址。重复系位总是 1。字节 4，5 和 7 总是 0，字节 8 和 9 指示定时器、计数器或块的号码。



下表所示为参数类型 ANY 的数据类型代码。

数据类型的代码		
十六进制代码	数据类型	描述
B#16#17	BLOCK_FB	FB号码
B#16#18	BLOCK_FC	FC号码
B#16#19	BLOCK_DB	DB号码
B#16#1A	BLOCK_SDB	SDB号码
B#16#1C	COUNTER	计数器号码
B#16#1D	TIMER	定时器号码

A.3.4.6 使用参数类型 ANY

可以为块定义一个形参使之适合任何数据类型的实参。当块被调用时其提供的实参的数据类型是未知的或可变的(以及允许是任意数据类型)时候，这种功能特别有用。在块的变量声明中，可以声明参数为数据类型 ANY。然后在 STEP 7 中可以指定任意数据类型的一个实参。

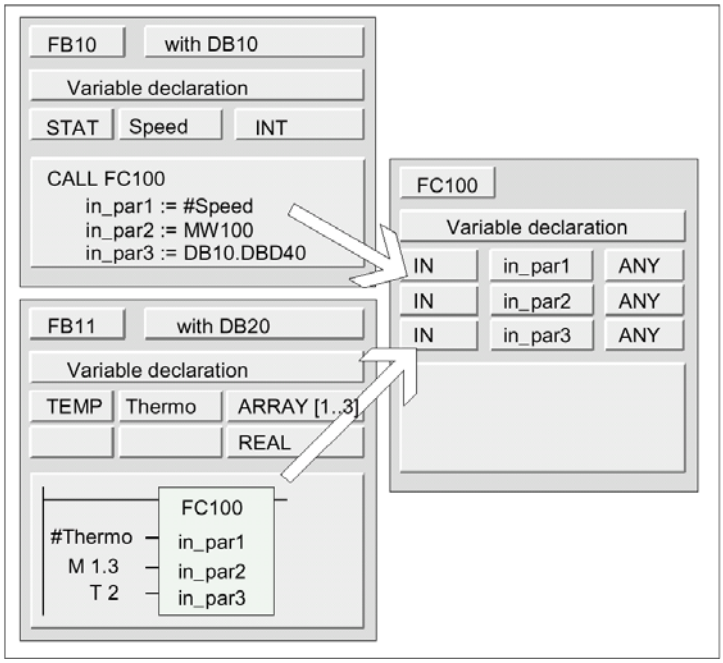
STEP 7 为 ANY 数据类型的变量分配 80 个位的存储区。如果将一个实参赋值给这个形参，STEP 7 会将实参的起始地址、数据类型和长度编码在这 80 个位中。被调用的块分析为这个 ANY 参数所存储的 80 个位的数据，获得进一步处理所需的信息。

分配一个实参给一个 ANY 参数

如果一个参数数据类型为 ANY，则可以分配一个任意数据类型的实参给这个形参。在 STEP 7 中，可以指定以下数据类型作为实参：

- 基本数据类型：可以指定这个实参的绝对地址或符号名。
- 复合数据类型：对于复合数据类型(如数组和结构体)的数据要指定符号名。
- 定时器、计数器和块：指定号码(例如，T1，C20或FB6)。

下图说明数据是如何传送给一个带有数据类型为 ANY 参数的 FC。



在本例中 FC100 有三个参数(in_parl,in_par2 以及 in_par3)被声明为 ANY 数据类型。

- 当FB10调用FC100时，FB10传送一个整数(静态变量speed)，一个字(MW100)以及一个双字到DB10(DB10.DBD40)。
- 当FB11调用FC100时，FB11传送一个实数数组(临时变量“Thermo”)，一个布尔值(M1.3)，以及一个定时器(T2)。

为一个 ANY 参数指定一个数据区域

不仅可以给一个 ANY 参数指定一个单个地址(例如，MW100)，而且还可以指定一个数据区域。如果想指定一个区域作为实参，使用下列常数格式指定要传送的数据的量：

P#　　区域标识 字节.位　　数据类型　重复系数

对于数据类型，可以以常数的格式指定所有的基本数据类型和数据类型 DATE_AND_TIME。如果数据类型不是 BOOL，位地址必须指定 0(x.0)。下表是对三个格式示例的说明，这些格式用于指定传送给一个 ANY 参数的存储区域。

实参	描述
P#M50.0 BYTE 10	在字节存储区域指定10个字节： MB50至MB59
P#DB10.DBX5.0 S5TIME 3	指定三个位于DB10中的数据类型为 S5TIME的数据单位：DB字节5至DB字节10
P#Q10.0 BOOL 4	在输出区域指定4位：Q10.0至Q10.3

使用参数类型 ANY 的示例

下例说明如何使用参数类型 ANY 和系统功能 SFC20 BLKMOV 复制一个 10 个字节的存储区域。

```
STL                              解释
FUNCTION FC10:VOID
VAR_TEMP
   Source : ANY;
   Target : ANY;
END_VAR
BEGIN
LAR1   P#Source;                 将 ANY 指针的起始地址装入 AR1。

L   B#16#10;                     装载语法标识(ID)并且
T   LB[ARI,P#0.0];               传送给 ANY 指针

L   B#16#02;                     装载数据类型字节并且
T   LB[AR1,P#1.0];               传送给 ANY 指针

L   10;                          装载 10 个字节并且
T   LW[AR1,P#2.0];               传送给 ANY 指针

L   22;                          源是 DB22，DBB11
T   LW[AR1,P#4.0];
L   P#DBX11.0;
T   LD[AR1,P#6.0];

LAR1   P#Target;                 装载 ANY 指针的起始地址到 AR1

L   B#16#10;                     装载语法标识(ID)并且
T   LB[AR1,P#0.0];               传送给 ANY 指针

L   B#16#02;                     装载数据类型字节并且
T   LB[AR1,P#1.0];               传送给 ANY 指针

L   10;                          装载 10 个字节并且
T   LW[AR1,P#2.0];               传送给 ANY 指针

L   33;                          目标是 DB33，DBB202
T   LW[AR1,P#4.0];
L   P#DBX202.0;
T   LD[AR1,P#6.0];

CALL SFC 20(                     调用系统功能 BLKMOV
   SRC  BLK:=Source,
   RET_VAL:=MW12,                评估 BR 位和 MW12
   DSTBLK:=Target
   );
END_FUNCTION
```

A.3.4.7 为逻辑块的局域数据指定数据类型

使用 STEP 7，在块变量声明中，分配给局域数据的数据类型(基本和复合数据类型以及参数类型) 有限定的。

OB 局域数据的有效数据类型

下表说明为一个 OB 声明局域数据的限定(一)。由于不可能调用一个 OB，所以 OB 没有参数(输入、输出或入/出)。由于 OB 没有背景 DB，所以不能为 OB 声明任何静态变量。OB 的临时变量的数据类型可以是基本的或复合的数据类型以及数据类型 ANY。

有效的赋值用符号 • 表示。

声明类型	基本数据类型	复合数据类型	参数类型	参数类型	参数类型	参数类型	参数类型
			TIMER	COUNTER	BLOCK	POINTER	ANY
Input	—	—	—	—	—	—	—
Output	—	—	—	—	—	—	—
In/out	—	—	—	—	—	—	—
Static	—	—	—	—	—	—	—
Temporary	•(1)	•(1)	—	—	—	—	•(1)

1）位于OB的L堆栈中。

FB 局域数据的有效数据类型

下表说明为一个 FB 声明局域数据的限定(一)。由于使用背景 DB，为 FB 声明局域数据所受的限定很少。当声明输入参数时没有任何限定；对于一个输出参数不能声明任何参数类型，而对于入/出参数所允许的参数类型只有 POINTER 和 ANY。可以声明临时变量为 ANY 数据类型，其它的所有参数类型都是非法的。

有效的赋值用符号 • 表示。

声明类型	基本数据类型	复合数据类型	参数类型	参数类型	参数类型	参数类型	参数类型
			TIMER	COUNTER	BLOCK	POINTER	ANY
Input	•	•	•	•	•	•	•
Output	•	•	—	—	—	—	—
In/out	•	•(1), (3)	—	—	—	•	•
Static	•	•	—	—	—	—	—
Temporary	• (2)	•(2)	—	—	—	—	•(2)

（1）在背景数据块中存为参考(48位指针)
（2）位于FB的L推栈中。
（3）字符串只能定义为却省长度。

FC 局域数据的有效数据类型

下表说明为一个 FC 声明局域数据的限定(一)。由于 FC 没有背景 DB，所以它没有静态变量。对于 FC 的输入、输出和入 / 出参数只有参数类型 POINTER 和 ANY 是允许的。也可以声明临时变量为 ANY 参数类型。

有效的赋值用符号 • 表示。

声明类型	基本数据类型	复合数据类型	参数类型	参数类型	参数类型	参数类型	参数类型
			TIMER	COUNTER	BLOCK	POINTER	ANY
Input	•	•(2)	•	•	•	•	•
Output	•	•(2)	—	—	—	•	•
In/out	•	•(2)	—	—	—	•	•
Static	—	—	—	—	—	—	—
Temporary	•(1)	•(1)	—	—	—	—	•(1)

（1）位于FC的L的堆栈

（2）字符串只能定义为却省长度。

A.3.4.8 程序块间传送参数时允许的数据类型

程序块之间参数传送的规则

当给形参赋实参时，可以指定一个绝对地址\一个符号名或一个常数。STEP 7 为不同的参数限定有效的赋值。例如，输出和输入/输出参数不能被指定为常数(因为输出或入/出参数的目的就是要改变它的数值)。这些限定特别地应用于复合数据类型的参数，对它们既不能指定绝对地址也不能是常数。

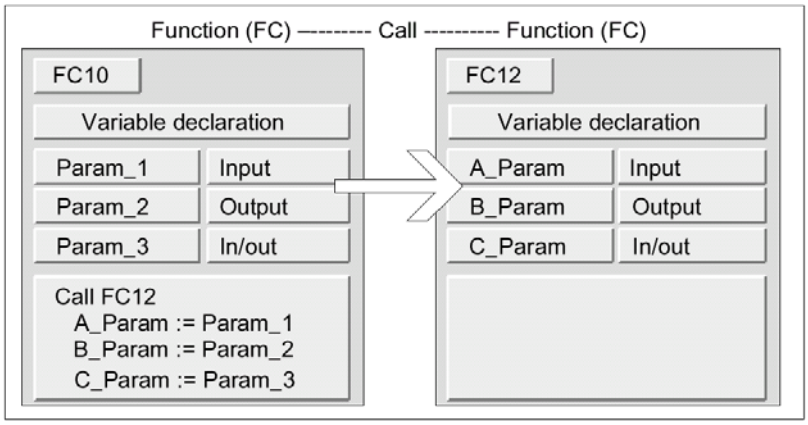
下表说明对涉及给形参赋实参的数据类型的限定。

有效的赋值使用符号 • 表示。

基本数据类型				
声明类型	绝对地址	符号名（在符号表中）	临时局域符号	常数
Input	•	•	•	•
Output	•	•	•	—
In/out	•	•	•	—
复合数据类型				
声明类型	绝对地址	DB元素的符号名	临时局域符号	常数
Input	—	•	•	—
Output	—	•	•	—
In/out	—	•	•	—

功能（FC）调功能（FC）时的有效数据类型

可以将一个调用 FC 的形参赋值给一个被调用的 FC 的形参。下图说明将 FC10 的形参作为实参赋值给 FC12 的形参。



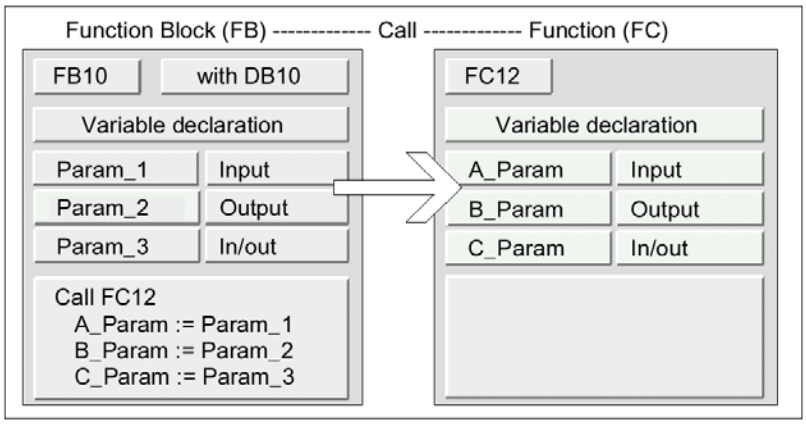
STEP 7 对于将一个 FC 的形参作为实参赋值给另一个 FC 的形参作了限定。例如，不能将复合数据类的参数或一个参数类型作为实参赋值。

下表所示为一个 FC 调用另一个 FC 时允许的数据类型(•)。

声明类型	基本数据类型	复合数据类型	参数类型	参数类型	参数类型	参数类型	参数类型
			TIMER	COUNTER	BLOCK	POINTER	ANY
Input→Input	•	—	—	—	—	—	—
Input→Output	—	—	—	—	—	—	—
In/out→In/out	—	—	—	—	—	—	—
Output→Input	—	—	—	—	—	—	—
Output→Output	•	—	—	—	—	—	—
Output→In/out	—	—	—	—	—	—	—
In/out→Input	•	—	—	—	—	—	—
In/out→Output	•	—	—	—	—	—	—
In/out→In/out	•	—	—	—	—	—	—

功能块（FB）调用功能（FC）时的有效数据类型

可以将一个调用 FB 的形参赋值给一个被调用的 FC 的形参。下图说明将 FB10 的形参作为实参赋值给 FC12 的形参。

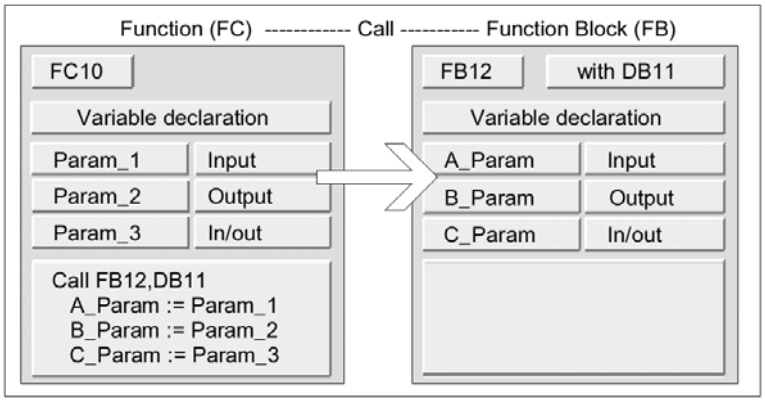


STEP 7 对于将一个 FB 的形参赋值给一个 FC 的形参作了限定。例如，不能将参数类型的参数作为实参赋值。下表所示为一个 FB 调用一个 FC 时允许的数据类型(•)。

声明类型	基本数据类型	复合数据类型	参数类型	参数类型	参数类型	参数类型	参数类型
			TIMER	COUNTER	BLOCK	POINTER	ANY
Input→Input	•	•	—	—	—	—	—
Input→Output	—	—	—	—	—	—	—
Input→In/out	—	—	—	—	—	—	—
Output→Input	—	—	—	—	—	—	—
Output→Output	•	•	—	—	—	—	—
Output→In/out	—	•	—	—	—	—	—
In/out→Input	•	—	—	—	—	—	—
In/out→Output	•	—	—	—	—	—	—
In/out→In/out	•	—	—	—	—	—	—

功能（FC）调用功能块时的有效数据类型

可以将一个调用 FC 的形参赋值给一个被调用的 FB 的形参。下图说明 FC10 的形参作为实参赋值给 FB12 的形参。



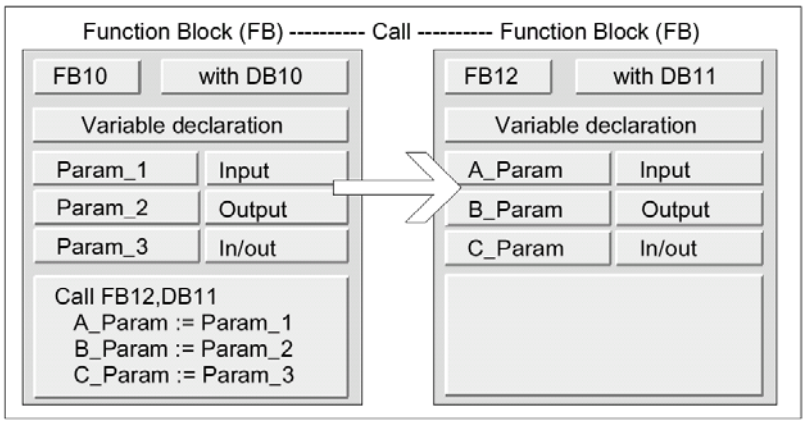
STEP 7 对将一个 FC 的形参赋值给一个 FB 的形参作了限定。例如，不能将复合数据类型的参数作为实参赋值。但是，可以将参数类型为 TIMER、COUNTER 或 BLCCK 的输入参数赋值给被调用 FB 的输入参数。

下表所示为一个 FC 调用一个 FB 时允许的数据类型(•)。

声明类型	基本数据类型	复合数据类型	参数类型	参数类型	参数类型	参数类型	参数类型
			TIMER	COUNTER	BLOCK	POINTER	ANY
Input→Input	•	—	•	•	•	—	—
Input→Output	—	—	—	—	—	—	—
Input→In/out	—	—	—	—	—	—	—
Output→Input	—	—	—	—	—	—	—
Output→Output	•	—	—	—	—	—	—
Output→In/out	—	—	—	—	—	—	—
In/out→Input	•	—	—	—	—	—	—
In/out→Output	•	—	—	—	—	—	—
In/out→In/out	•	—	—	—	—	—	—

功能块调用功能块时的有效数据类型

可以将一个调用 FB 的形参赋值给一个被调用的 FB。下图说明 FB10 的形参作为实参赋值给 FB12 的形参。



STEP 7 对于将一个 FB 的形参赋值给另一个 FB 的形参作了限定。例如，不能用复合数据类型的输入和输出参数作为实参赋值给被调用的 FB 的输入和参数。但是可以将参数类型为 TIMER、COUNTER 或 BLOCK 的输入参数赋值给被调用的 FB 的输入参数。

下表所示为一个 FB 调用另一个 FB 时允许的数据类型(•)

声明类型	基本数据类型	复合数据类型	参数类型	参数类型	参数类型	参数类型	参数类型
			TIMER	COUNTER	BLOCK	POINTER	ANY
Input→Input	•	•	•	•	•	—	—
Input→Output	—	—	—	—	—	—	—
Input→In/out	—	—	—	—	—	—	—
Output→Input	—	—	—	—	—	—	—
Output→Output	•	•	—	—	—	—	—
Output→In/out	—	—	—	—	—	—	—
In/out→Input	•	—	—	—	—	—	—
In/out→Output	•	—	—	—	—	—	—
In/out→In/out	•	—	—	—	—	—	—

A.3.4.9 向功能块的 IN_OUT 参数作传送

当复合数据类型被传送给功能块(FB)的 IN_OUT 参数时，变量的地址被传送(由参考调用)。

当基本数据类型被传送给一个功能块的 IN_OUT 参数时，在起动功能块之前数值被复制到背景数据块中，当功能块结束后又从背景数据复制回来。这意味着基本数据类型的 IN_OUT 变量可以被一个数值初始化，但是，在调用中不能用一个常数作为实参代替 IN_OUT 变量，因为不能对一个常数作写操作。

数据类型为 STRUCT 或 ARRAY 的变量不能被初始化，因为这种情况在背景数据块中只有一个地址。

A.4 使用旧项目工作

A.4.1 转换版本 1 的项目

对于使用 STEP 7 版本 1 生成的项目可以再利用。如需这样，必须将版本 1 的项目转换为版本 2 的项目。

版本 1 的项目中以下组件可以保留：

- 带有程序的项目结构
- 块
- STL源文件
- 符号表

硬件组态不会被转换。不能复制这个项目中的程序组件到其它的项目。还可以在新项目中增加一个站并对它组态和参数赋值。

一旦完成版本 2 的转换，就可以在一个对话框中决定现在是否要把这个版本 2 的项目转换为当前版本下 STEP 7 的项目。

提示

各个块仍保留版本 1 块作为它们的属性。版本 1 中生成的程序不会修改，因此不能被用来连接多重背景。

如果想在转换过来的块中声明多重背景，首先使用“LAD/STL/FBD：Programming BLocks”应用程序将转换过来的块生成 STL 源文件，然后再编译源文件，生成新的块替代原有的块。

编程多重背景是 STEP 7 版本 2 的新特性，用以生成功能块(FB)。如果想在版本 2 的项目中按原来的方式继续使用在版本 1 中生成的功能块，不必转换它们。

步骤

要转换版本 1 的项目，按如下进行

1. 选择菜单命令 File > Open Version 1 Project
2. 在出现的对话框中选择要转换为版本 2 的项目，可以通过扩展名*.s7a 来识别版本 1 的项目。
3. 在下一个对话框中，为版本 1 的项目转换后的新项目输入名字。

A.4.2 转换版本 2 项目

在 STEP 7 中也可以用菜单命令 **File > Open** 打开版本 2 的项目。

版本 2 的项目/库可以被转换(迁移)到你当前的 STEP 7 版本。使用菜单命令 **File > Save As** 并选中选项”Rearrage before saving(存储前重新整理)”，该项则被存为当前 STEP 7 版本下的一个项目。

可以编辑来自旧的 STEP 7 版本的项目和库并保持它们的格式，并且在“Save Project As”对话框中选择旧的 STEP 7 版本作为它们保存的文件类型。例如，要编辑 STEP 7 版本 2.1 的对象，这里选择“Project 2.x”或“library 2.x”（不可能将 5.1 以上的版本存储为版本 2，可以参考编辑版本 2 的项目和库）。

文件类型标志

	STEP 7 V3	从STEP 7 V4开始
当前版本的文件类型	Project 3.x Library 3.x	Project Library
旧版本的文件类型	Project 2.x Library 2.x	Project 2.x Library 2.x

这意味着只能访问某一范围内的 STEP 7 旧版本的功能。但是仍可以继续用旧的 STEP 7 版本管理这些项目和库。

提示

从版本 3 更新到版本 4 及更高的版本只涉及名字的变化：格式仍保持一致。因此在 STEP 7 V4 中不存在文件类型“Project3.x”。

步骤

将版本 2 的项目转换为当前 STEP 7 版本的格式可按如下进行：

1. 在 File 菜单中为该项目执行“Save As”命令以及“Rearrange before saving”选项。
2. 在“Save Project As”对话框中选择文件类型“Project”并点击“Save”按钮。

要转换版本 2 的项目到当前的 STEP 7 版本同时要保持它们的格式，可按如下进行：

1. 如果有必要执行以上步骤 1。
2. 在“Save Praject As”对话框中选择旧的 STEP 7 版本的文件类型并点击“Save”按钮。

A.4.3 关于 STEP 7 V2.1 项目使用 GD 通讯的提示

- 如果要将一个使用全局数据的STEP 7 V2.1的项目转换为STEP 7 V5，首先在STEP 7 V2.1的项目中用STEP 7V5.0打开GD表。以前所组态的通讯数据通过GD通讯应用程序自动转换为新的结构。
- 当存档一个STEP 7 V2.1项目时，如果项目中包含文件名长于八个字符的文件时，旧程序(ARJ，PKZIP…)会发出一个错误信息。如果被编辑的STEP 7 V2.1项目中的MPI网络具有长于8个字符的ID(标识)时，也会出现这条错误信息。在首次启动组态全局数据通信之前，在有全局数据的STEP 7 V2.1项目中，为MPI网络编辑一个不超过8个字符长的名字。
- 如果要更改一个STEP 7 V2.1项目名，必须在GD表的标头栏(CPU)中通过重新选择适当的CPU来重新赋值。如果恢复旧的项目名，该赋值关系会被再次显示。

A.4.4 DP 从站丢失 GSD 文件或 GSD 文件有故障

如果用 STEP 7 V5.1 处理以前组态的站，可能会丢失 DP 从站的 GSD 文件包，或不能编译 GSD 文件（例如由于 GSD 文件中有语法错误）

在这种情况下，STEP 7 生成一个“虚假”的从站来代表所组态的从站，例如当一个站复制到编程器中或打开项目执行更多的处理。该“虚假”从站只能做有限的处理，不能更改从站结构（DP 标识符）以及从站参数。从站原有组态被保留，也可以删除整个从站。

重新对 DP 从站组态并赋值参数

如果要对 DP 从站进行重新组态和参数赋值，必须从制造商申请最新的 GSD 文件并通过 Options>Install New GSD 进行安装。

装入正确文件后，用它来代表 DP 从站。此时 DP 从站包含了所需数据，并可以对其进行处理。

A.5 示例程序

A.5.1 示例项目和示例程序

安装 CD 中包含许多示例项目。在 SIMATIC Manager 的“Open”对话框中可以找到这些示例项目。当安装选项软件包时可能还会有其它示例项目。有关这些示例项目的信息请参见选项软件包中的资料。

实例和示例项目	包括在CD中	在本章中已描述	OB1中的说明
“Zen01_01_STEP7_*” to “Zen01_06_STEP7_*”项目（使用入门和练习）	•	单独的手册	•
“Zen01_11_STEP7_DezP”项目（PROFIBUS DP组态实例）	•	-	-
“Zen01_08_STEP7_Blending”项目（工业混料实例）	•	•	-
“Zen01_09_STEP7-Zebra”项目（斑马线交通信号控制）	•		•
“Zen01_10_STEP7_COM-SFB”项目（两个S7-400 CPU间的数据交换）	•		•
“ZXX01_14_Hsystem_S7400H”项目（容错系统的起动项目）	•	单独的手册	•
“ZXX01_15_Hsystem_RED_IO”项目(带冗余I/O设备的容错系统的起动项目）	•	单独的手册	•
“Zen01_11_STEP7_COM_SFC1” 和 ”Zen01_11_STEP7_COM_SFC12”项目(为非组态的连接用通讯SFC交换数据）	•		•
项目 “ZEn01_13_STEP7_PID-Temp” (使用FB 58 和FB59进行温度控制的示例)	•		•
处理日时钟中断实例		•	
处理时间延迟中断实例		•	
屏蔽和中断屏蔽同步错误的实例		•	
禁止和使能中断及异步错误的实例		•	
中断和异步错误的延迟处理实例		•	

这些实例的侧重点不是要教会一种控制特定过程所需的编程样式或专家知识，这些实例只不过用来说明设计一个程序应遵循的步骤。

删除和安装提供的示例项目

提供的示例项目在SIMATIC管理器中可以被删除，然后还可以再安装。要安装示例项目，必须起动STEP 7 V5.0的安装程序，示例项目也可以被选则稍后安装。可以将提供的示例程序及用户生成的示例程序使用菜单命令“Save As”存储为用户项目。

提示

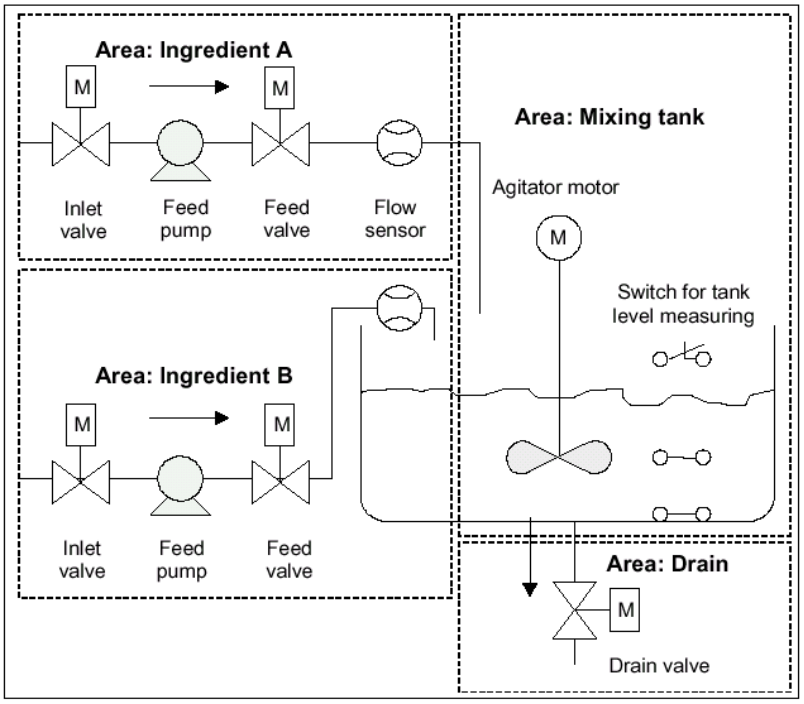
当安装 STEP 7 时，除非指定否则会复制提供的示例项目。如果已对提供的示例项目作了编辑，则这些修改过的项目在 STEP 7 重新安装时会被原版覆盖。为此，应该在作任何修改前复制提供的示例项目，然后在备份的项目上编辑。

A.5.2 工业搅拌过程的示例程序

本示例程序使用的关于控制一个工业搅拌过程的信息可能已在本手册的第一部中读过。

任务

两种配料(配料 A 和配料 B)在一个混合罐中由搅拌器混合在一起。混合后的产品通过一个排料阀排出该罐。下图为示例过程的框图。



过程中各个部分的描述

手册的第一部分包括了如何将示例过程分割为功能区域和单个任务的描述。下面将说明各个区域。

配料 A 区和配料 B 区：

- 每种配料的管道都配备有一个入口和一个进料阀以及进料泵。

- 进料管还有流量传感器。
- 当罐的液面传感器指示罐满时，进料泵的接通必须被锁定。
- 当排料阀打开时进料泵的启动必须被锁定。
- 在起动进料泵后最开始的1秒钟内必须打开入口阀和进料阀。
- 在进料泵停止后(来自流量传感器的信号)阀门必须立即被关闭以防止配料从泵中泄露。
- 进料泵的启动与一个时间监控功能相结合，换句话说，在泵启动后的7秒之内，流量传感器会报告溢出。
- 当进料泵运行时，如果流量传感器没有流量信号，进料泵必须尽可能快地断开。
- 进料泵启动的次数必须进行计数。(维护间隔)

混合罐区：

- 当罐的液面传感器指示“液面低于最低限”或排料阀打开时，搅拌电机的启动必须被锁定。
- 搅拌电机在达到额定速度时要发出一个响应信号。如果在电机启动后10秒内还未接收到该信号，则电机必须被断开。
- 必须对搅拌电机的启动次数进行计数(维护间隔)。
- 在混合罐中必须安装三个传感器；
  - 罐装满：一个常闭触点。当达到罐的最高液面时，该触点断开。
  - 罐中液面高于最低限：一个常开触点。如果达到最低限，该触点关闭。
  - 罐非空：一个常开触点，如果罐不空，该触点闭合。

排料区：

- 罐内产品的排出由一个螺线管阀门控制。
- 这个螺线管阀门由操作员控制，但是最迟在“罐空”信号产生时，该阀必须被关闭。
- 打开排料阀必须被锁定，当：
  - 搅拌电机在工作时
  - 罐空时

操作员站

要让一个操作员启动、停止和监控过程，就需要一个操作站。操作站有以下配备：

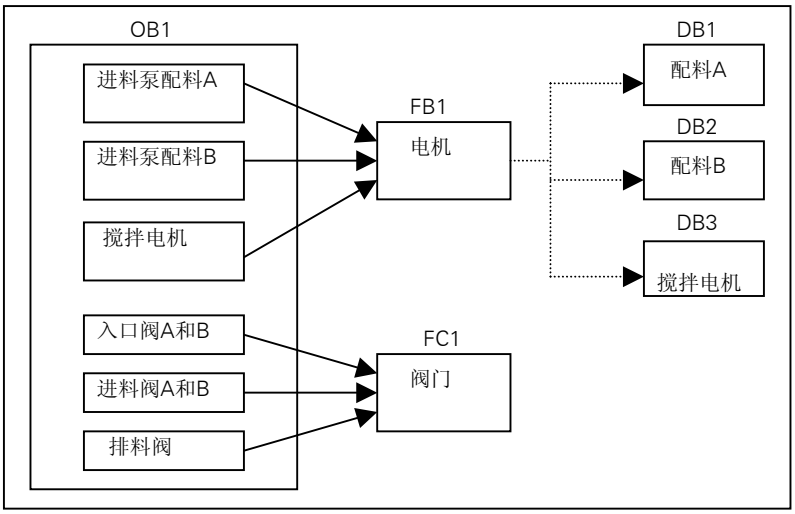
- 用于控制过程中最重要阶段的开关。使用“reset maintenance display(复位维护显示)”开关，可以关掉电机的维护显示灯，复位相应的计数器使维护间隔为0。
- 指示过程状态的显示灯。
- 紧急停机开关。

A.5.2.1 定义逻辑块

可以将用户程序分布到不同的块中并建立块调用的分层结构来结构化程序。

块调用的分层结构

下图所示为结构化编程的块的分层调用结构。



- OB1：与CPU操作系统的接口，包含主要程序。在OB1中调用块FB1和FC1并传送控制过程所需的特定参数。
- FB1：配料A的进料泵、配料B的进料泵和搅拌电机的控制由于要求一致(接通、断开、计数应用程序等)，所以可通过同一个功能块进行控制。
- 背景DB1-3：用于控制配料A、配料B的进料泵和搅拌电机的实参及静态数据各不相同，因此分别存储在与FB1相关的三个背景DB中。
- FC1：配料A和B的入口阀和进料阀以及排料阀也共同使用一个逻辑块。由于只需编辑“打开和闭合”功能，一个FC就足够了。

A.5.2.2 指定符号名

定义符号名

在示例程序中使用了符号，这些符号必须用 STEP 7 在符号表中定义。下表所示为所用程序组件的符号名及绝对地址。

进料泵、搅拌电机和入口阀的符号地址			
符号名	地址	数据类型	说明
Feed_pump_A_start	I0.0	BOOL	启动配料A的进料泵
Feed_pump_A_stop	I0.1	BOOL	停止配料A的进料泵
Flow_A	I0.2	BOOL	配料A流动
Inlet_valve_A	Q4.0	BOOL	启动配料A的入口阀
Feed_valve_A	Q4.1	BOOL	启动配料A的进料阀
Feed_pump_A_on	Q4.2	BOOL	“配料A的进料泵运行”指示灯
Feed_pump_A_off	Q4.3	BOOL	“配料A的进泵未运行”指示灯
Feed_pump_A	Q4.4	BOOL	配料A的进料泵启动
Feed_pump_A_fault	Q4.5	BOOL	“进料泵A故障”指示灯
Feed_pump_A_maint	Q4.6	BOOL	“进料泵A维护”指示灯
Feed_pump_B_start	I0.3	BOOL	启动配料B的进料泵
Feed_pump_B_stop	I0.4	BOOL	停止配料B的进料泵
Flow_B	I0.5	BOOL	配料B流动
Inlet_valve_B	Q5.0	BOOL	启动配料B的入口阀
Feed_valve_B	Q5.1	BOOL	启动配料B的进料阀
Feed_pump_B_on	Q5.2	BOOL	“配料B进料泵运行”指示灯
Feed_pump_B_off	Q5.3	BOOL	“配料B的进料泵未运行”指示灯
Feed_pump_B	Q5.4	BOOL	启动配料B的进料泵
Feed_pump_B_fault	Q5.5	BOOL	“进料泵B故障”指示灯
Feed_pump_B_maint	Q5.6	BOOL	“进料泵B维护”指示灯
Agitator_running	I1.0	BOOL	搅拌电机的响应信号
Agitator_start	I1.1	BOOL	搅拌启动按扭
Agitator_stop	I1.2	BOOL	搅拌停止按扭
Agitator	Q8.0	BOOL	启动搅拌器
Agitator_on	Q8.1	BOOL	“搅拌器运行”指示灯
Agitator_off	Q8.2	BOOL	“搅拌器未运行”指示灯
Agitator_fault	Q8.3	BOOL	“搅拌电机故障”指示灯
Agitator_maint	Q8.4	BOOL	“搅拌电机维护”指示灯

传感器和罐液面显示的符号地址			
符号名	地址	数据类型	说明
Tank_below_max	I1.3	BOOL	“混合罐未满”传感器
Tank_above_min	I1.4	BOOL	“混合罐液面高于最低限”传感器
Tank_not_empty	I1.5	BOOL	“混合罐非空”传感器
Tank_max_disp	Q9.0	BOOL	“混合罐满”指示灯
Tank_min_disp	Q9.1	BOOL	“混合罐液面低于低限”指示灯
Tank_empty_disp	Q9.2	BOOL	“混合罐空”指示灯

排料阀的符号地址			
符号名	地址	数据类型	说明
Drain_open	I0.6	BOOL	打开排料阀的按钮
Drain_closed	I0.7	BOOL	关闭排料阀的按钮
Drain	Q9.5	BOOL	启动排料阀
Drain_open_disp	Q9.6	BOOL	“排料阀打开”指示灯
Drain_closed_disp	Q9.7	BOOL	“排料阀关闭”指示灯

其它编程组件的符号地址			
符号名	地址	数据类型	说明
EMER_STOP_off	I1.6	BOOL	紧急停机开关
Reset_maint	I1.7	BOOL	复位所有电机上的维护指示灯的开关
Motor_block	FB1	FB1	控制泵和电机的FB
Valve_block	FC1	FC1	控制阀门的FC
DB_feed_pump_A	DB1	FB1	控制送料泵A的背景DB
DB_feed_pump_B	DB2	FB1	控制送料泵B的背景DB
DB_agitator	DB3	FB1	控制搅拌电机的背景DB

A.5.2.3 生成电机的 FB

该 FB 的要求是什么?

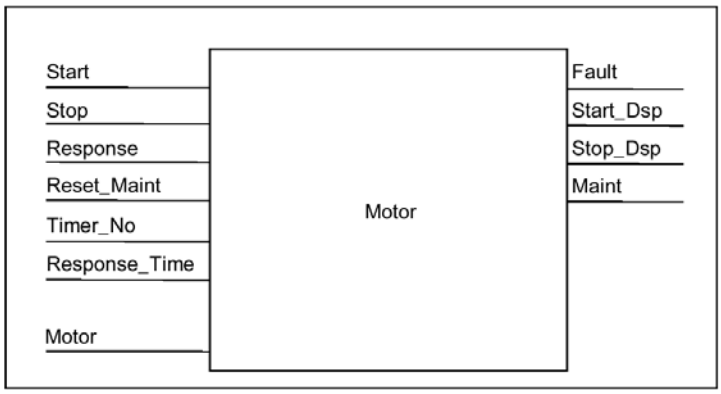
电机的 FB 包括以下逻辑功能：

- 有启动和停止输入。
- 允许设备操作的一系列互锁(泵和搅拌电机)。互锁状态存储在OB1的临时局域数据(L推栈)中(“Motor_enable”，“Valve-enable”)，并且当电机的FB被处理时与启动和停止输入进行逻辑组合。
- 来自设备的返馈必须在一个特定的时间内出现，否则就假定有故障或错误出现。该功能则停下电机。
- 时间点和响应时间或错误/故障循环持续时间都必须被指定。

- 如果启动按钮被按下并且电机被使能，设备自行接通并运行直至按下停机按钮。
- 当设备接通时，一个定时器启动运行，如果在定时器的时间到达之前未接到来自设备的响应信号，则停机。

指定输入和输出

下图所示是电机通用 FB 的输入和输出。



定义 FB 的参数

如果为电机(用于控制泵和电机)使用多重背景的 FB，必须为输入和输出定义通用参数名。

用于示例过程中的电机的 FB 需要以下各项；

- 它必须有来自操作站的信号可以停止或启动电机和泵。
- 它需要来自电机和泵的响应信号以指示电机在运行。
- 它必须计算从发出启动电机的信号到接收到响应信号的时间。如果在这一时间内没有收到响应信号则电机必须关断。
- 它必须能接通或断开操作站上的指示灯。
- 它提供一个信号启动电机。

这些要求可以被定义为 FB 的输入和输出。下表所示为我们的示例过程中电机的 FB 的参数。

参数名	输入	输出	入/出
Start	n		
Stop	n		
Response	n		
Reset_maint	n		
Timer_No	n		
Response_Time	n		
Fault		n	
Start_Dsp		n	
Stop_Dsp		n	
Maint		n	
Motor			n

为电机的 FB 声明变量

必须为电机的 FB 声明输入、输出和入/出参数。

地址	声明	变量名称	类型	初始值
0.0	IN	Start	BOOL	FALSE
0.1	IN	Stop	BOOL	FALSE
0.2	IN	Response	BOOL	FALSE
0.3	IN	Reset_Manit	BOOL	FALSE
2.0	IN	Timer_No	TIMER	
4.0	IN	Response_Time	S5TIME	S5T#0MS
6.0	OUT	Fault	BOOL	FALSE
6.1	OUT	Start_Dsp	BOOL	FALSE
6.2	OUT	Stop_Dsp	BOOL	FALSE
6.3	OUT	Maint	BOOL	FALSE
8.0	IN_OUT	Motor	BOOL	FALSE
10.0	STAT	Time_bin	WORD	W#16#0
12.0	STAT	Time_BCD	WORD	W#16#0
14.0	STAT	Starts	INT	0
16.0	STAT	Start_Edge	BOOL	FALSE

FB 的输入、输出、输入输/出和静态变量存储在指定的背景 DB 中，临时变量存储在 L 堆栈中。

为电机编程 FB

在 STEP 7 中，每一个被调用的块一定要在调用它的块之前生成。因此在示例程序中必须在 OB1 之前生成电机的 FB。

使用 STL 编程语言的 FB1 程序部分如下：

Network 1 启动/停止和锁存

```
      A(
      O     #Start
      O     #Motor
      )
      AN    #Stop
      =     #Motor
```

Network 2 启动监控

```
      A     #Motor
      L     #Response_Time
      SD    #Timer_No
      AN    #Motor
      R     #Timer_No
      L     #Timer_No
      T     #Timer_bin
      LC    #Timer_No
      T     #Timer_BCD
```

```
      A     #Timer_No
      AN    #Response
      S     #Fault
      R     #Motor
Network 3 启动指示灯和故障复位
      A     #Response
      =     #Start_Dsp
      R     #Fault
Network 4 断开指示灯
      AN    #Response
      =     #Stop_Dsp
Network 5 启动计数
      A     #Motor
      FP    #Start_Edge
      JCN   lab1
      L     #Starts
      +     1
      T     #Starts
      lab1: NOP 0
Network 6 维护指示灯
      L     #Starts
      L     50
      >=I
      =     #Maint
Network 7 复位累计启动次数的计数器
      A     #Reset_Maint
      A     #Maint
      JCN   END
      L     0
      T     #Starts
      END:NOP 0
```

生成背景数据块

生成三个数据块并依次打打开。在“New Data Block(新数据块)”对话框中选择选项“Data block referencing a funtion block(参照一个功能块的数据块)”，在“Reference(参考)”列表框中选择“FB1”。则数据块被指定为 FB1 的背景数据块。

A.5.2.4 为阀门生成 FC

该 FC 的要求是什么?

这个入口和进料阀以及排料阀的功能包含以下逻辑功能：

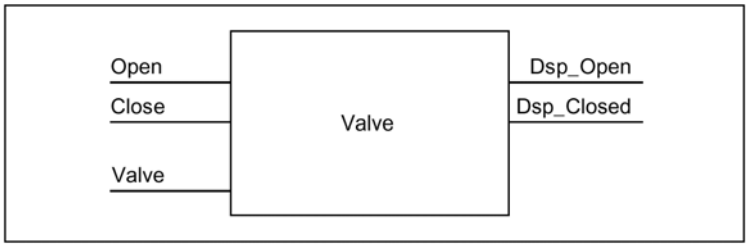
- 一个用于打开阀门的输入一个用于关闭门的输入。
- 互锁允许阀门被打开。互锁状态存储在OB1的临时局域数据(L堆栈)中(“Valve_enable”)并且在阀门的FC被处理时与打开和关闭的输入信号进行逻辑组合。

下表所示为必须传送给 FC 的参数。

阀门的参数	Input	Output	In/Out
Open	✓		
Close	✓		
Dsp_Open		✓	
Dsp_Closed		✓	
Valve			✓

指定输入和输出

下图所示为阀门的通用 FC 的输入和输出。调用 FB 为电机传送输入参数，调用 FC 为阀门返回输出参数。



为阀门的 FC 声明变量

就象为电机的 FB 一样，必须为阀门的 FC 声明输入、输出和输入/输出参数(见下面的变量声明表)

地址	声明	变量名称	类型	初始值
0.0	IN	Open	BOOL	FALSE
0.1	IN	Close	BOOL	FALSE
2.0	OUT	Dsp_Open	BOOL	FALSE
2.1	OUT	Dsp_Closed	BOOL	FALSE
4.0	IN_OUT	Valve	BOOL	FALSE

在 FC 中，临时变量存储在 L 堆栈。输入、输出和输入/输出变量则作为指针存储于调用 FC 的逻辑块中。L 堆栈中另外的存储空间(在临时变量之后)用于存储这些变量。

为阀门编程 FC

由于被调用的块必须在调用块之前生成，所以阀门的 FC1 功能必须在 OB1 之前生成。

使用 STL 编程语言的 FC1 程序部分如下：

```
Network 1 打开/关闭和锁存
      A(
```

```
      O     #Open
      O     #Valve
      )
      AN    #Close
      =     #Valve
Netword 2 显示“阀门打开”
      A     #Valve
      =     #Dsp_Open
Network 3 显示“阀门关闭”
      AN    #Valve
      =     #Dsp_Closed
```

A.5.2.5 生成 OB1

OB1 决定示例程序的结构。OB1 中也包含要传送给各个功能的参数，例如：

- 为进料泵和搅拌电机而编制的STL程序段中为电机FB提供起动(“Start”)、停止(“Stop”)，响应(“Respense”)以及复位维护显示(“Reset_Maint”)的输入参数。PLC的每一个循环周期都会处理这个电机的FB。
- 如果电机的FB被处理，则输入Timer_No和Respense_Time指示所使用的定时器功能以及要测量的时间。
- 因为阀门的FC和电机的FB是在OB1中调用的，所以它们在每个程序循环都会被可编程控制器处理。

程序为处理进料泵和搅拌电机的控制任务，使用电机的 FB 时分别配备了不同的背景 DB。

为 OB1 声明变量

OB1 的变量声明表如下所示。前 20 个字节包含 OB1 的启动信息，不能够被修改。

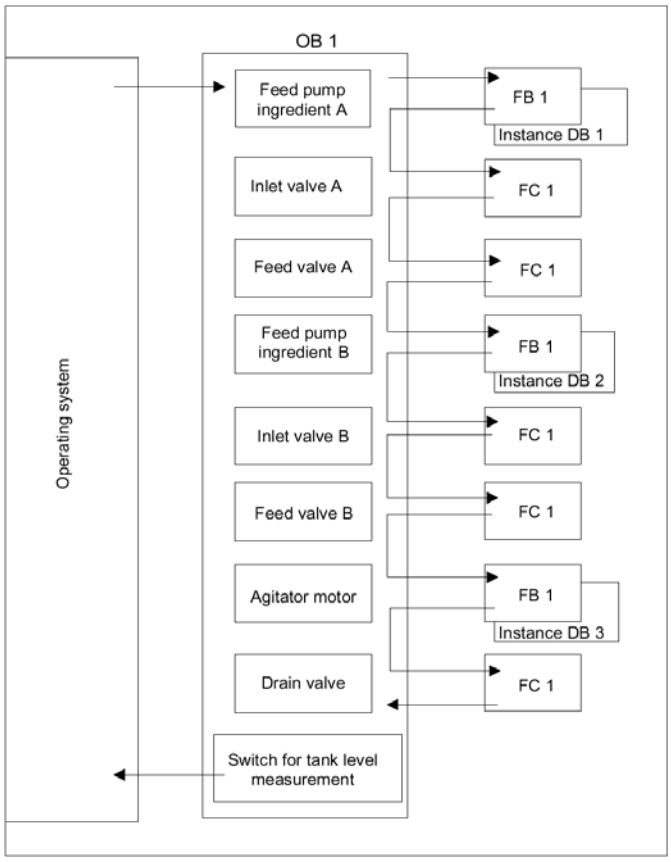
地址	变量声明	变量名	类型
0.0	TEMP	OB1_EV_CLASS	BYTE
1.0	TEMP	OB1_SCAN1	BYTE
2.0	TEMP	OB1_PRIORITY	BYTE
3.0	TEMP	OB1_OB_NUMBER	BYTE
4.0	TEMP	OB1_RESERVED_1	BYTE
5.0	TEMP	OB1_RESERVED_2	BYTE
6.0	TEMP	OB1_PREV_CYCLE	INT
8.0	TEMP	OB1_MIN_CYCLE	INT
10.0	TEMP	OB1_MAX_CYCLE	INT
12.0	TEMP	OB1_DATE_TIME	DATE_AND_TIME
20.0	TEMP	Enable_motor	BOOL
20.1	TEMP	Enable_valve	BOOL
20.2	TEMP	Start_fulfilled	BOOL

地址	变量声明	变量名	类型
20.3	TEMP	Stop_fulfilled	BOOL
20.4	TEMP	Inlet_valve_A_open	BOOL
20.5	TEMP	Inlet_valve_A_closed	BOOL
20.6	TEMP	Feed_valve_A_open	BOOL
20.7	TEMP	Feed_valve_A_closed	BOOL
21.0	TEMP	Inlet_valve_B_open	BOOL
21.1	TEMP	Inlet_valve_B_closed	BOOL
21.2	TEMP	Feed_valve_B_open	BOOL
21.3	TEMP	Feed_valve_B_closed	BOOL
21.4	TEMP	Open_drain	BOOL
21.5	TEMP	Close_drain	BOOL
21.6	TEMP	Valve_closed_fulfilled	BOOL

生成 OB1 程序

在 STEP 7 中，每个被调用的块必须在调用它的块之前生成。在示例程序中，必须在编程 OB1 之前生成电机的 FB 和阀门的 FC。FB1 和 FC1 在 OB1 中不止一次地被调用；FB1 的调用使用了不同的背景 DB：

参见下页框图。



STL 编程语言的 OBI 程序部分如下所示：

```
Network1 进料泵 A 的互锁
    A     “EMEP_STOP_off”
    A     “Tank_below_max”
    AN    “Driain”
    =     #Enable_Motor
Network2 为配料 A 调用电机的 FB
    A                “Feed_pump_A_start”
    A                #Enable_Motor
    =                #Start_Fulfilled
    A(
    O                “Feed_Pump_A_stop”
    ON               #Enable_Motor
    )
    =                #Stop_Fulfilled
    CALL “Motor_block”,“DB_feed_pump_A”
         Start   :=Start_Fulfilled
         Stop    :=Stop_Fulfilled
         Response        := “Flow_A”
         Reset_Maint     := “Reset_maint”
         Timer_No        :=T12
         Reponse_Time    :=S5T#7S
         Fault   := “Feed_pump_A_fault”
         Start_Dsp   := “Feed_pump_A_on”
         Stop_Dsp    := “Feed_pump_A_off”
         Maint := “Feed_pump_A_maint”
         Motor := “Feed_pump_A”
Network 3 延迟配料 A 的阀使能
    A     “Feed_pump_A”
    L     S5T#1S
    SD    T       13
    AN    “Feed_pump_A”
    R     T       13
    A     T       13
    =     #Enable_Valve
Network 4 配料 A 的入口阀控制
    AN    “Flow_A”
    AN    “Feed_pump_A”
    =             #Close_Valve_Fulfilled
    CALL  “Valve_baock”
          Open       :=#Enable_Valve
          Close      :=#Close_Valve_Fulfilled
          Dsp_Open       :=#Inlet_Valve_A_Open
```

```
          Dsp_Closed     :=#Inlet_Valve_A_Closed
          Valve          :=“Inet_Valve_A”
Network 5 配料 A 的进料阀控制
    AN    “Flow_A”
    AN    “Feed_pump_A”
    =     #Close_Valve_Fulfilled
    CALL  “Valve_block”
          Open           :=#Enable_Valeve
          Close          :=#Close_Valve_Fulfilled
          Dsp_Open       :=#Feed_Valve_A_Open
          Dsp_Closed:=#Feed_Valve_A_Closed
          Valve  :=“Feed_Valve_A”
Network 6 进料泵 B 的互锁
    A     “EMER_STOP_off”
    A     “Tank_below_max”
    AN    “Drain”
    =     “Enable_Motor”
Network 7 为配料 B 调用电机的 FB
    A                “Feed_pump_B_start”
    A                #Enable_Motor
    =                #Start_Fulfilled
    A(
    O                “Feed_pump_B_stop”
    ON               #Enable_Motor”
    )
    =          #Stop_Fulfilled
    CALL             “Motor_block”,“DB_feed_pump_B”
          Star     :=#Start_Fulfilled
          Stop     :=#Stop_Fullfilled
          Response    :=“Flow_B”
          Reset_Maint      :=“Reset_maint”
          Timer_No    :T14
          Reponse_Time    :=S5T#7S
          Fault    :=“Feed_pump_B_fault”
          Start_Dsp    :=“Feed_pump_B_on”
          Stop_Dsp    :=“Feed_pump_B_off”
          Maint :=“Feed_pump_B_mait”
          Motor :=“Feed_pump_B”
Network 8 延迟配料 B 的阀使能
    A     “Feed_pump_B”
    L     S5T#1S
    SD    T        15
    AN    “Feed_pump_B”
```

```
    R     T      15
    A     T      15
    =     #Enable_Valve
Network 9  配料 B 的入口阀控制
    AN    “Flow_B”
    AN    “Feed_pump_B”
    =     #Close_Valve_Fulfilled
    CALL  “Valve-block”
          Open  :=#Enable_Valve
          Close  :=#Close_Valve_Fulfilled
          Dsp_Open  :=#Inlet_Valve_B_Open
          Dsp_Closed:=#Inlet_Valve_B_Closed
          Valve  := “Inlet_Valve_B”
Network 10  配料 B 的进料阀控制
    AN    “Flow_B”
    AN    “Feed_pump_B”
    =     #Close_Valve_Fulfilled
    CALL  “Valve_block”
          Open  :=#Enable_Valve
          Close  :=#Close_Valve_Fulfilled
          Dsp_Open  :=#Feed_Valve_B_Open
          Dsp_Closed:=#Feed_Valve_B_Closed
          Valve  := “Feed_Valve_B”
Network 11  搅拌器的互锁
    A     “EMER_STOP_off”
    A     “Tabj_above_min”
    AN    “Drain”
    =     #Enable_Motor
Network 12  为搅拌器调用电机的 FB
    A     “Agitator_Start”
    A     #Enable_Motor
    =     #Start_Fulfilled
    A(
    O     “Agitator_stop”
    ON    #Engable_Motor
    )
    =     #Stop_Fulfilled
    CALL  “Motor_block”,“DB_Agitator”
          Start   :=#Start_Fulfilled
          Stop   :=#Stop_Fullfilled
          Response   := “Agitator_running”
          Reset_Maint      := “Reset_maint”
          Timer_No   :=T16
```

```
          Reponse_Time   :=S5T#10S
          Fault   :=“Agitator fault”
          Start_Dsp   :=“Agitator on”
          Stop_Dsp   :=“Agitator_off”
          Maint :=“Agitator_maint”
          Motor :=“Agitator”
Network 13 排料阀的互锁
    A     “EMER_STOP_off”
    A     “Tank_not_empty”
    AN    “Agitator”
    =     “Enable_Valve”
Network 14 排料阀控制
    A     “Drain_open”
    A     #Enable_Valve
    =     #Open_Drain
    A(
    O     “Drain_closed”
    ON    #Enable_Valve
    )
    =     #Close_Drain
    CALL  “Valve_baock”
          Open  :=#Open_Drain
          Close  :=#Close_Drain
          Dsp_Open  :=“Drain_open_disp”
          Dsp_Closed       :=“Drain_closed_disp”
          Valve  :=“Drain”
Network 15Tank level display 罐液面显示
    AN    “Tank_below_max”
    =     “Tank_max disp”
    AN    “Tank_above_min”
    =     “Tank_min_disp”
    AN    “Tank_not-empty”
    =     “Tank_empty_disp”
```

A.5.3 处理日时钟中断举例

“日时钟中断” 的用户程序的结构

FC 12

OB 10

OB1 和 OB80

A.5.3.1 “日时钟中断”的用户程序的结构

任务

输出 Q4.0 要在周一的上午 5：00 至周五的下午 8：00 被置位，从周五的下午 8：00 至周一的上午 5：00 被复位。

转换为用户程序

下表所示为每个块的子任务

块	分任务
OB1	调用功能FC12
FC12	视输出Q4.0的状态、日时钟中断的状态及输入I0.0和I0.1而定 • 指定起始时间 • 设置日时钟中断 • 激活日时钟中断 • CAN_TINT
OB10	视当前的星期而定 • 指定起始时间 • 置位或复位输出Q4.0 • 设置下一个日时钟中断 • 激活下一个日时钟中断
OB80	置位输出Q4.1 在位存储区域保存OB80的起动信息

使用的地址

下表所示为使用的共享地址。临时局域变量在各个块的声明区内声明。

地址	含义
I0.0	使能“设置日时钟中断”和“激活日时钟中断”的输入
I0.1	取消一个日时钟中断的输入
Q4.0	由日时钟中断OB（OB10）置位/复位的输出
Q4.1	由时间错误（OB80）置位的输出
MW16	日时钟中断的状态（SFC31”QRY_TINT”）
MB100至MB107	OB10(只有日时钟)的起动事件信息存储区
MB110至MB129	OB80(时间错误)的坡动事件信息存储区
MW200	SFC 28”SET_TINT”的RET_VAL(返回值)
MB202	SFC的二进制结果（状态位BR）缓存区
MW204	SFC 30”ACT_TINT”的RET_VAL
MW208	SFC 31”QRY_TINT”的RET_VAL

使用的系统功能和功能

在编程示例中使用了以下系统功能：

- SFC28“SET_TINT”：设置日时钟中断
- SFC29“CAN_TINT”：取消日时钟中断
- SFC30“ACT_TINT”：激活日时钟中断
- SFC31“QRY_TINT”：查询日时钟中断
- SFCT“D_TOD_DT”：组合DATE和TIME_OF_DAY为DT。

A.5.3.2 FC12

声明部分

以下是在 FC12 的声明部分声明的临时局域变量：

Variable Name	DateType	Declaration	Comment
IN_TIME	TIME_OF_DAY	TEMP	起动时间
IN_DATE	DATE	TEMP	起动日期
OUT_TIME_DATE	DATE_AND_TIME	TEMP	转换后的起动日期/时间
OK_MEMORY	BOOL	TEMP	使能日时钟中断设置

STL 程序部分

使用 STL 编写的用户程序 FC12 如下：

STL(FC12)	解释
Network 1	SFC QRY_TINT
CALL SFC 31	查询日时钟中断的状态
OB_NO := 10	
RET_VAL:= MW 208	
STATUS := MW 16	
Network 2	依据 Q4.0 指定起始时间（在变量
AN Q 4.0	#IN_DATE 和#IN_TIME 中）
JC mond	起始日期是星期五
L D#1995-1-27	
T #IN_DATE	
L TOD#20:0:0.0	
T #IN_TIME	
JU CNVRT	
mond:L D#1995-1-23	
T #IN_DATE	起始日期是星期一

L TOD#5:0:0.0 T #IN_TIME cnvrt:NOP 0 Network 3: CALL FC 3 IN1 := #IN_DATE In2 := #IN_TIME RET_VAL:= #OUT_TIME_DATE Network 4: A I 0.0 AN M 17.2 A M 17.4 = #OK_MEMROY Network 5: A #OK_MEMORY JNB m001 CALL SFC 28 OB_NO := 10 SDT := #OUT_TIME_DATE PERIOD := W#16#1201 RET_VAL := MW 200 m001 A BR = M 202.3 Network 6: A #OK_MEMMORY JNB m002 CALL SFC 30 OB_N := 10 RET_VAL := MW 204 m002 A BR = M202.4 Network 7: A I0.1 JNB M003 Call SFC29 OB_NO:=10 RET_VAL:=MW210 m003: A BR = M202.5	从 DATE 和 TIME_OF_DAY 转换格式为 DATE_AND_TIME(用于设置日时钟中断) 设置日时钟中断的要求是否全部满足？（使能输入置位、日时钟中断未激活以及日时钟中断 OB 已装载）如果满足，设置日时钟中断… …并且激活日时钟中断 如果取消日时钟中断的输入置位了，则取消日钟中断

A.5.3.3 OB10

声明部分

与 OB10 的缺省声明部分不同，以下临时局域变量要进行声明：

- 用于整个起动事件信息的结构体(STARTINFO)
- 在STARTINFO结构体内用于时间的结构体(T_STMP)
- 其它临时局域变量WDAY，IN_DATE、IN_TIME和OUT_TIME_DATE

变量名	数据类型	变量声明	注释
STARTINFO	STRUCT	TEMP	OB10的整个起动事件信息声明为结构体
E_ID	WORD	TEMP	事件ID
PR_CLASS	BYTE	TEMP	优先级
OB_NO	BYTE	TEMP	OB号码
RESERVED_1	BYTE	TEMP	保留
RESERVED_2	BYTE	TEMP	保留
PERIOD	WORD	TEMP	日时钟中断的周期
RESERVED_3	DWORD	TEMP	保留
T_STMP	STRUCT	TEMP	用于日时钟明细数据的结构体
YEAR	BYTE	TEMP	
MONTH	BYTE	TEMP	
DAY	BYTE	TEMP	
HOUR	BYTE	TEMP	
MINUTES	BYTE	TEMP	
SECONDS	BYTE	TEMP	
MSEC_WDAY	WORD	TEMP	
	END_STRUCT	TEMP	
	END_STRUCT	TEMP	
WDAY	INT	TEMP	星期
IN_DATE	DATE	TEMP	FC3的输入变量 （日期转换格式）
IN_TIME	TIME_OF_DAY	TEMP	FC3的输入变量 （时间转换格式）
OUT_TIME_DATE	DATE_AND_TIME	TEMP	FC3的输出变量和SFC28的输入变量

STL 程序部分

使用 STL 编写的用户程序 OB10 如下：

STL(OB10)	解释
Network 1	
L #STARTINFO.7_STMP.MSEC_WDAY	选择星期
L W#16#F	
AW	
T #WDAY	并存储
Network 2:	
L #WDAY	如果不是周一,则指定周一,5.00AM为下一个
L 2	起动时间并复位输出Q4.0
<>I	
JC mond	
Network 3:	
L D#1995-1-27	否则，如果是周一，指定周五，8.00pm (20.00)
T #IN_DATE	作为下一个起动时间并置位输出Q4.0
L TOD#20:0:0.0	
T #IN_TIME	
SET	
= Q 4.0	
JU cnvrt	
mond: L D#1995-1-23	
T #IN_DATE	
L TOD#5:0:0.0	
T #IN_TIME	
CLR	
= Q 4.0	启动指定的时间
	转换指定的起始时间为DATE_AND_TIME格
cnvrt: NOP 0	式(用于SFC28)
Network 4:	
CALL FC 3	
IN1 := #IN_DATE	
IN2 := #IN_TIME	
RET_VAL := #OUT_TIME_DATE	
Network 5:	
CALL SFC 28	
OB_NO := 10	
SDT := #OUT_TIME_DATE	
PERIOD := W#16#1201	
RET_VAL := MW 200	
A BR	
= M 202.1	
Network 6:	
CALL SFC 30	激活日时钟中断
OB_NO := 10	
RET_VAL := MW 204	

A BR = M 202.2 Network 7: CALL SFC 20 SRCBLK := #STARTINFO.T_STMP RET_VAL := MW 206 DSTBLK := P#M 100.0 BYTE 8	块传送:将OB10的起动事件信息中的日时钟存储到存储区域MB100至MB107

A.5.3.4 OB1 和 OB80

由于在本例中没有评估 OB1(用于循环程序的 OB)的起动事件信息，所以只显示 OB80 的起动事件信息。

OB1 程序部分

使用 STL 编写的用户程序 OB1 如下：

STL(OB1)	解释
CALL FC12	调用功能FC12

OB80 声明部分

与 OB80 的缺省声明部分不同，以下临时局域变量要进行声明：

- 用于整个起动事件信息的结构体(STARTINFO)
- 在STARTINFO结构体内的用于时间的结构体(T_STMP)

变量名	数据类型	变量声明	注释
STARTINFO	STRUCT	TEMP	OB80的整个起动事件信息声明为结构体
E_ID	WORD	TEMP	事件ID
PR_CLASS	BYTE	TEMP	优先级
OB_NO	BYTE	TEMP	OB号码
RESERVED_1	BYTE	TEMP	保留
RESERVED_2	BYTE	TEMP	保留
A1_INFO	WORD	TEMP	关于造成错误的事件的附加信息
A2_INFO	DWORD	TEMP	关于事件优先级和故障OB号码的附加信息
T_STMP	STRUCT	TEMP	用于日时钟明细数据的结构体
YEAR	BYTE	TEMP	
MONTH	BYTE	TEMP	
DAY	BYTE	TEMP	
HOUR	BYTE	TEMP	
MINUTES	BYTE	TEMP	

变量名	数据类型	变量声明	注释
SECONDS	BYTE	TEMP	
MSEC_WDAY	WORD	TEMP	
	END_STRUCT	TEMP	
	END_STRUCT	TEMP	

OB80 程序部分

使用 STL 编写的用户程序 OB10 如下，如果出现时间错误，操作系统会调用 OB80：

STL(OB80)	解释
Network 1 AN Q 4.1 S Q 4.1 CALL SFC 20 SRCBLK := #STARTINFO RET_VAL := MW 210 DSTBLK := P#M 110.0 BYTE 20	如果时间错误出现，置位Q4.1 块传送：将整个起动事件信息存储到存储区域MB110至MB129

A.5.4 处理时间延迟中断的示例

“时间延迟中断”的用户程序结构

OB20、OB1

A.5.4.1 “时间延迟中断”用户程序的结构

任务

当输入 I0.0 置位，输出 Q4.0 应在 10 秒后被置位。每次输入 I0.0 置位，延迟时间都将被启动。

时间延迟中断的起动时间(秒和毫秒)应作为用户定义的 ID 出现在时间延迟中断 OB(OB20)的起动事件信息中。

如果在这 10 秒钟内 I0.1 被置位，则组织块 OB20 不应被调用；这意味着输出 Q4.0 不应被置位。

当输入 I0.2 置位时，输出 Q4.0 应被复位。

转换为用户程序

下表为每个块的子任务。

块	分任务
OB1	读当前时间并为起动时间延迟中断作准备 根据输入I0.0的边沿变化，起动时间延迟中断；根据时间延迟中断的状态以及输入I0.1的边沿变化，取消时间延迟中断；要据输入I0.2的状态，复位输出Q4.0
OB20	置位输出Q4.0 读并且准备当前时间 将起动事件信息存到位存储区域

使用的地址

下表所示为使用的共享地址，临时局域变量在各个块的声明区域内声明。

地址	含义
I0.0	使能“起动时间延迟中断”的输入
I0.1	取消一个时间延迟中间的输入
I0.2	复位输出Q4.0的输入
Q4.0	由时间延迟中断OB(OB20)置位的输入
MB1	用作边沿标志及SFC的二进制结果（状态位BR）缓存
MW4	时间延迟中断的状态（SFC34”QRY_TINT”）
MD10	来自OB1起动事件信息的秒和毫秒的BCD码
MW100	SFC32”SRT_DINT”的RET_VAL（返回值）
MW102	SFC34”QRY_DINT”的RET_VAL
MW104	SFC33”CAN_DINT”的RET_VAL
MW106	SFC20”BLKMOV”的RET_VAL
MB120-MB139	OB20的起动事件信息的存储区
MD140	来自OB20的起动事件信息的秒和毫秒的BCD码
MW144	来自OB1的起动事件信息的秒和毫秒的BCD码；取自OB20的起动事件信息（用户指定IDSIGN）

使用的系统功能

以下是在“时间延迟中断”的用户程序中使用的 SFC：

- SFC32“SRT_DINT”：启动时间延迟中断
- SFC33“CAN_DINT”：取消时间延迟中断
- SFC34“QRY_DINT”：查询时间延迟中断的状态

A.5.4.2 OB20

声明部分

与 OB20 的缺省声明部分不同，下列临时局域变量要进行声明：

- 用于整个起动事件信息的结构体(STARTINFO)
- 在STARTINFO结构体中用于时间的结构体(T_STMP)

变量名	数据类型	变量声明	注释
STARTINFO	STRUCT	TEMP	OB20的起动信息
E_ID	WORD	TEMP	事件ID
PC_NO	BYTE	TEMP	优先级
OB_NO	BYTE	TEMP	OB号码
D_ID1	BYTE	TEMP	数据ID1
D_ID2	BYTE	TEMP	数据ID2
SIGN	WORD	TEMP	用户指定的ID
DTIME	TIME	TEMP	时间延迟中断的起动时间
T_STMP	STRUCT	TEMP	用于日时钟明细数据的结构体（时间印记）
YEAR	BYTE	TEMP	
MONTH	BYTE	TEMP	
DAY	BYTE	TEMP	
HOUR	BYTE	TEMP	
MINUTES	BYTE	TEMP	
SECONDS	BYTE	TEMP	
MSEC_WDAY	WORD	TEMP	
	END_STRUCT	TEMP	
	END_STRUCT	TEMP	

程序部分

使用 STL 编写的用户程序 OB20 如下：

STL(OB20)	解释
Network 1	
SET	无条件置位输出Q4.0
= Q 4.0	
Network 2:	
L QW 4	立即启动输出字
T PQW 4	
Network 3:	
L #STARTINFO.T_STMP.SECONDS	从起动事件信息中读秒
T MW 140	
L #STARTINFO.T_STMP.MSEC_WDAY	从起动事件信息中读毫秒和星期
T MW 142	

L MD 140 SRD 4 T MD 140	去掉星期并将毫秒重新写回（现在MW142中为BCD码）
Network 4: L #STARTINFO.SIGN T MW 144	从起动事件信息中读时间延迟中断的起动时间（=调用SFC32）
Network 5: CALL SFC 20 SRCBLK := STARTINFO RET_VAL := MW 106 DSTBLK := P#M 120.0 Byte 20	将起动事件信息复制到存储区域（MB120至MB139）

A.5.4.3 OB1

声明部分

与 OB1 中缺省声明部分不同，以下临时局域变量要进行声明：

- 用于整个起动事件信息的结构体(STARTINFO)
- 在STARTINFO结构体中用于时间的结构体(T_STMP)

变量名	数据类型	变量声明	注释
STARTINFO	STRUCT	TEMP	OB1的起动信息
E_ID	WORD	TEMP	事件ID
PC_NO	BYTE	TEMP	优先级
OB_NO	BYTE	TEMP	OB号码
D_ID1	BYTE	TEMP	数据ID1
D_ID2	BYTE	TEMP	数据ID2
CUR_CYC	INT	TEMP	当前循环时间
MIN_CYC	INT	TEMP	最小循环时间
MAX_CYC	INT	TEMP	最大循环时间
T_STMP	STRUCT	TEMP	用于日时钟明细数据的结构体（时间印记）
YEAR	BYTE	TEMP	
MONTH	BYTE	TEMP	
DAY	BYTE	TEMP	
HOUR	BYTE	TEMP	
MINUTES	BYTE	TEMP	
SECONDS	BYTE	TEMP	
MSEC_WDAY	WORD	TEMP	
	END_STRUCT	TEMP	
	END_STRUCT	TEMP	

程序部分

使用 STL 编写的用户程序 OB10 如下：

STL(OB1)	解释
Network 1:	
L #STARTINFO.T_STMP.SECONDS	从起动事件信息中读秒值
T MW 10	
L #STARTINFO.T_STMP.MSEC_WDAY	从起动事件信息中读毫秒和星期
T MW 12	去掉星期并重新写回毫米（现为BCD码在MW12
L MD 10	中）
SRD 4	
T MD 10	
Network 2:	输入I0.0是否有上升沿？
A I 0.0	
FP M 1.0	
= M 1.1	
Network 3:	
A M 1.1	如果有，起动时间延迟中断
JNB m001	（时间延迟中断的起始时间赋给参数SIGN）
CALL SFC 32	
OB_NO := 20	
DTME := T#10S	
SIGN := MW 12	
RET_VAL :=MW 100	
m001: NOP 0	
Network 4:	查询时间延迟中断的状态
CALL SFC 34	(SFC QRY_DINT)
OB_NO := 20	
RET_VAL := MW 102	
STATUS := MW 4	输入I0.1有上升沿吗？
Network 5:	
A I 0.1	
FP M 1.3	
= M 1.4	
Network 6:	…并且时间延迟中断已被激活吗
A M 1.4	（时间延迟中断状态的位2）？
A M 5.2	则取消时间延迟中断
JNB m002	
CALL SFC 13	
OB_NO := 20	
RET_VAL :=104	
m002: NOP 0	
A I 0.2	用输入I0.2复位输出Q4.0
R Q 4.0	

A.5.4.4 屏蔽和中断屏蔽同步错误的示例

以下用户程序的示例说明如何屏蔽和中断屏蔽同步错误。使用 SFC36“MSK_FLT”，以下错误可以在编程错误筛选器中被屏蔽：

- 作读操作时的区域长度错误
- 作写操作时的区域长度错误

再次调用 SFC36“MSK_FLT”也可以屏蔽一个访问区域：

- 作写操作时I/O访问错误

使用 SFC38“READ-ERR”可查询被屏蔽的同步错误。用 SFC37“DMSK-FLT”“写操作时 I/O 访问错误”的屏蔽可以被取消。

程序部分

在下面的 OB1 中可以看到使用语句表编程的用户程序示例。

STL(Network 1)	解释
AN M 255.0	非保持存储位M255.0(只有第一次运行时=0)
JNB m001 CALL SFC 36	SFC 36 MSK_FLT(屏蔽同步错误) 位2=位3=1(BLFL和BLFS被屏蔽)
PRGFLT_SET_MASK :=DW#16#C	所有位=0(没有访问错误被屏蔽)
ACCFLT_SET_MASK :=DW#16#0	返回值
RET_VAL :=MW 100	输出当前编程错误筛选器到MD10
PRGFLT_MASKED :=MD 10	输出当前编程错误筛选器到MD14
ACCFLT_MASKED :=MD14	如果屏蔽成功置位M255.0
m001: A BR S M 255.0	

STL(Network 2)	解释
CALL SFC 36	SFC 36 MSK_FLT(屏蔽同步错误)
PRGFLT_SET_MASK :=DW#16#0	所有位=0(没有更进一步的编程错误被屏蔽)
ACCFLT_SET_MASK :=DW#16#8	位3=1(写操作访问错误被屏蔽)
RET_VAL :=MW 102	返回值
PRGFLT_MASKED :=MD 20	输出当前编程错误筛选器到MD20
ACCFLT_MASKED :=MD 24	输出当前编程错误筛选器到MD24

STL(Network 3)	解释
AN M 27.3 BEC	如果写访问错误(ACCFLT_MASKED中的位3)没有屏蔽则块结束

STL(Network 4)	解释
L B#16#0 T PQB 16	 向PQB16作写访问(用数值0)

STL(Network 5)	解释
CALL SFC 38 PRGFLT_QUERY :=DW#16#0 ACCFLT_QUERY :=DW#16#8 RET_VAL :=MW 104 PRGFLT_CLR :=MD 30 ACCFLT_CLR :=MD 34 A BR A M 37.3 NOT = M 0.0	SFC 38 READ_ERR(查询同步错误) 所有位=0(没有编程错误被查询) 位3=1(查询写访问错误) 返回值 输出当前编程错误筛选器到MD 30 输出当前编程错误筛选器到MD 34 无错误出现并且在检查写访问错误 RLO取反 如果PQB16存在则M0.0=1

STL(Network 6)	解释
L B#16#0 T PQB 17	向PQB17作写访问(用数值0)

STL(Network 7)	解释
CALL SFC 38 PRGFLT_QUERY :=DW#16#0 ACCFLT_QUERY :=DW#16#8 RET_VAL :=MW 104 PRGFLT_CLR :=MD 30 ACCFLT_CLR :=MD 34 A BR A M 37.3 NOT = M 0.0	SFC 38 READ_ERR(查询同步错误) 所有位=0(没有编程错误被查询) 位3=1(查询写访问错误) 返回值 输出当前编程错误筛选器到MD 30 输出当前编程错误筛选器到MD 34 无错误出现并且在检查写访问错误 RLO取反 如果PQB17存在则M0.1=1

STL(Network 8)	解释
L B#16#0 T PQB 18	向PQB18作写访问(用数值0)

STL(Network 9)	解释
CALL SFC 38	SFC 38 READ_ERR(查询同步错误)
PRGFLT_QUERY :=DW#16#0	所有位=0(没有编程错误被查询) 位3=1(查询写访问错误)
ACCFLT_QUERY :=DW#16#8	返回值
RET_VAL :=MW 104	
PRGFLT_CLR :=MD 30	输出当前编程错误筛选器到MD 30
ACCFLT_CLR :=MD 34	输出当前编程错误筛选器到MD 34
A BR	无错误出现并且在检查写访问错误
A M 37.3	RLQ取反
NOT	
= M 0.2	如果PQB18存在则M0.2=1

STL(Network 10)	解释
L B#16#0	向PQB19作写访问(用数值0)
T PQB 19	

STL(Network 11)	解释
CALL SFC 38	SFC 38 READ_ERR(查询同步错误)
PRGFLT_QUERY :=DW#16#0	所有位=0(没有编程错误被查询) 位3=1(查询写访问错误)
ACCFLT_QUERY :=DW#16#8	返回值
RET_VAL :=MW 104	
PRGFLT_CLR :=MD 30	输出当前编程错误筛选器到MD 30
ACCFLT_CLR :=MD 34	输出当前编程错误筛选器到MD 34
A BR	无错误出现并且在检查写访问错误
A M 37.3	RLQ取反
NOT	
= M 0.3	如果PQB19存在则M0.3=1

STL(Network 12)	解释
CALL SFC 37 PRGFLT_QUERY :=DW#16#0 ACCFLT_QUERY :=DW#16#8 RET_VAL :=MW 102 PRGFLT_CLR :=MD 20 ACCFLT_CLR :=MD 24	SFC 37 READ_ERR(中断屏蔽同步错误) 所有位=0(没有进一步的编程错误被屏蔽) 位3=1(屏蔽写访问错误) 返回值 输出当前编程错误筛选器到MD 20 输出当前编程错误筛选器到MD 24

STL(Network 13)	解释
A M 27.3 BEC	如果写访问错误(ACCFLT_MASKED位3)未被中断屏蔽则块结束

STL(Network 14)	解释
A M 0.0 JNB m002 L IB 0 T PQB 16 m002: NOP 0	如果存在，传送IB0到PQB16

STL(Network 15)	解释
A M 0.1 JNB m003 L IB 1 T PQB 17 m003: NOP 0	如果存在，传送IB1到PQB17

STL(Network 16)	解释
A M 0.2 JNB m004 L IB 2 T PQB 18 m004: NOP 0	如果存在，传送IB2到PQB18

STL(Network 17)	解释
A M 0.3 JNB m005 L IB 3 T PQB 19 m005: NOP 0	如果存在，传送IB3到PQB19

A.5.4.5 禁止和使能中断及异步错误的示例(SFC39 和 SFC40)

在这个用户程序的示例中，假设某个程序部分不能被中断中断。为这部分程序使用 SFC39 “DIS_IRT”禁止 OB35(循环中断)的调用，以后可以用 SFC40 “EN-IRT”再重新使能。

SFC39 和 SFC40 在 OB1 中调用：

STL(OB1)	解释
A M 0.0 S M 90.1 A M 0.1 S M 90.0 : :	可以被中断的程序部分：
CALL SFC 39 MODE :=B#16#2 OB_NO :=35 RET_VAL :=MW 100 : : L PIW 100 T MW 200 L MW 90 T MW 92 : :	不能被中断的程序部分： 禁止和放弃中断 模式2:禁止单个中断OB 禁止OB35
CALL SFC 40 MODE :=B#26#2 OB_NO :=35 RET_VAL :=MW 102	使能中断 模式2：使能单个中断OB 使能OB35
A M 10.0 S M 190.1 A M 10.1 S M 190.0 : :	可以被中断的程序部分

A.5.4.6 中断和异步错误的延迟处理示例(SFC41 和 SFC42)

在本例的用户程序中，假定某部分程序不能被中断中断。为这部分程序使用 SFC41 “DIS_AIRT”使中断延迟，以后再用 SFC42 “EN_AIRT”使能。

SFC41 和 SFC42 在 OB1 中调用：

STL(OB1)	解释
A M 0.0 S M 90.1 A M 0.1 S M 90.0 : :	可以被中断的程序部分：
CALL SFC 41 RET_VAL :=MW 100 L PIW 100 T MW 200 L MW 90 T MW 92 : :	不能被中断的程序部分： 禁止和延迟中断
CALL SFC 42 RET_VAL :=MW 102 L MW 100 DEC 1 L MW 102 <>I JC err	使能中断 设置中断禁止的次数在返回值中 该数值在中断使能后应与中断禁止前有相同的值（这里为“0”）
A M 10.0 S M 190.1 A M 10.1 S M 190.0 : : BEU	可以被中断的程序部分
err: L MW 102 T QW 12	显示已设置的中断禁止的次数

A.6 访问过程数据区和外设数据区

A.6.1 访问过程数据区

CPU 既可以通过使用过程映像表间接访问中央或分布式数字输入/输出模板的输入和输出，也可以通过背板/P 总线直接访问。

CPU 通过背板/P 总线直接访问中央输入和输出和分布式模拟输入/输出模板。也可以为模拟量模块分配过程映像区地址。

模板询址

可按如下所示，在用 STEP 7 组态模板时，将用户程序中使用的地址分配给模板：

- 对于中央的I/O模板：在组态表中布置机架并将模板分配到各个槽。
- 对于带有分布式I/O(PROFIBUS-DP)的站：在组态表的“master system(在系统)”中安排DP从站，为其设置PROFIBUS地址并将模板分配到各槽。

通过组态模板，不再需要用开关在每个模板上设置地址。作为组态的结果，编程器把数据发送到 CPU 使得 CPU 能够识别分配给它的模板。

外设 I/O 寻址

输入和输出有不同的地址区域。这意味着外设区域的地址不仅要包括访问类型字节、字，而且还要有输入的标识符 I 和输出的标识符 Q。

下图所示为可用的外设地址区域。

地址区域	访问单元	S7标识符（IEC）
外设（I/O）区域：输入	外设输入字节 外设输入字 外设输入双字	PIB PIW PID
外设（I/O）区域：输出	外设输出字节 外设输出字 外设输出双字	PQB PQW PQD

要查找在各个模板上可以使用哪些地址区域，可参考以下手册：

- “S7-300可编程控制器，硬件和安装”手册
- “S7-300，M7-300可编程控制器，模板技术规范参考手册
- “S7-400，M7-400可编程控制器，模板技术规范参考手册

模板起始地址

模板起始地址是该模板的最低字节地址。它代表该模板的用户数据区域的起始地址，在许多情况下用来代表整个模板。

例如，在硬件中断、诊断中断、插入/移出模板中断以及电源故障中断中都要在相应的组织块的启动信息中输入模板的起始地址，用以识别引起中断的模板。

A.6.2 寻址外设数据区

外设数据区域可被分为以下几部分：

- 用户数据以及
- 诊断和参数数据

两个区域都有输入区域(只读)和输出区域(只写)。

用户数据

用户数据可以用字节地址(对数字信号模板)或字地址(对模拟量模板)来寻址其输入或输出区域。用户数据可以用装载和传送命令、通过功能(接作员接口访问)来访问，或者通过传送过程映像区。用户数据可以是以下各项：

- 来自信号模板的数字和模拟输入/输出信号
- 来自功能模板的控制和状态信息
- 来自通讯模板的点到点和总线连接信息(只有S7-300)

当传送用户数据时，最高可传送 4 个字节(DP 标准从站除外，见设置操作性能)。如果使用“传送双字”指令，四个连续且不可修改(一致的)字节被传送。如果你分别用四个“传送输入字节”指令，则硬件中断 OB 可能会插在指令之间以及插在向同一地址传送数据之间，这样 4 个字节后来的内容在它们被全部传送之前可能会被改变。

诊断和参数数据

模板的诊断和参数数据不能够被单独寻址但总可以以一套完整的数据被传送，这意味着总可以传送一致的诊断和参数数据。

对诊断和参数数据的访问可以使用模板的起始地址和数据记录号(DS)。数据记录可分为输入和输出数据记录。输入数据记录只能读，输出数据记录只能写。可以用系统功能或通讯功能(用户接口)访问数据记录。下表所示为数据记录与诊断和参数数据之间的关系。

数据	说明
诊断数据	如果该模板有诊断能力，则可通过读数据记录0和1获得模板的诊断数据
参数数据	如果该模板可组态，则可通过向数据记录0和1作写操作而将参数传送到模板

访问数据记录

可以使用模板的数据记录中的信息给可组态模板重新赋值参数，并且可以读有诊断能力的模板的诊断信息。

下表所示为哪些系统功能可以用来访问数据记录

SFC	目的
给模板赋值参数	
SFC55 WR_PARM	传送可修改参数（数据记录1）到寻址的信号模板
SFC56 WR_DPARM	从SDB100至129传送参数（数组0或1）到寻址的信号模板
SFC57 PARM_MOD	从SDB100至129传送参数（数组0或1）到寻址的信号模板
SFC58 WR_REC	传送任意数据到寻址的信号模板
读出诊断信息	
SFC59 RD_REC	读诊数据数据

注意：

如果一个 DPV1 从站被组态为使用 GSD 文件（GSD Rev.3），并且 DP 主站的 DP 接口设置为“S7 兼容通讯”，用户程序通过 SFC58/59 或 SFB53/52 不能访问 I/O 模板的数据纪录。原因是此时 DP 主站对插槽询址错误（组态的插槽+3）。

解决方法：将 DP 主站的接口设置到“DPV1”。

寻址 S5 模板

可以按如下所述访问 S5 模板：

- 使用接口模板IM463-2将一个S7-400与SIMATIC S5扩展机架相连
- 将装在适配盒中的S5模板插在S7-400的中央机架上。

如何在 SIMATIC S7 中寻址 S5 模板的解释在“S7-400，M7-400 可编程控制器，硬件和安装”手册中或随适配盒提供的说明中。

A.7 设置操作性

本章将解释如何通过设置系统参数或使用系统功能(SFC)修改 S7-300 和 S7-400 可编程控制器的某些特性。

可以在 STEP 7 的在线帮助中以及下列手册中找到关于模板参数的更详细信息：

- “S7-300可编程控制器，硬件和安装”手册
- “S7-300，M7-300可编程控制器，模板技术规范”参考手册
- “S7-400，M7-400可编程控制器、模板技术规范”参考手册

在“S7-300 和 S7-400 系统软件，系统和标准功能”参考手册中可以找到关于 SFC 的所有内容。

寻址 DP 标准从站

如果与 DP 标准从站之间的数据交换大于 4 个字节，则必须使用用于这种数据交换的特殊的 SFC。

一些新的 CPU 中通过 I/O 地址区交换大于 4 个字节的连续数据时，不再需要调用 SFC14/SFC15（参考对分布式 I/O 连续数据读写的描述）。

SFC	目的
给模板赋值参数	
SFC 15 DPWR_DAT	传送任意数据到寻址的信号模板
读出诊断信息	
SFC 13 DPNRM_DG	读诊断信息（异步读访问）
SFC 14 DPRD_DAT	读连续的数据（长度为3个字节或大于4个字节）

当一个 DP 诊断帧到达时，一个有 4 个字节诊断数据触发 CPU 诊断中断。可以调用 SFC13 DPNRM_DG 读出这 4 个字节。

A.7.1 改变模板的性能和特性

缺省设置

- 出厂时，所有S7可编程控制器的可组态模板都有适合与标准应用的缺省设置。用这些缺省值，可以立即使用模板而无需作任何设置。这些缺省值的解释在以下手册的模板说明中：
- “S7-300可编程控制器，硬件和安装”手册
- “S7-300、M7-300可编程控制器，模板技术规范”参考手册
- “S7-400，M7-400可编程控制器，模板技术规范”参考手册

哪些模板可以赋值参数?

可以修改模板的性能和特性，使它们适应当前的要求和工厂情况。可组态的模板是 CPU、FM、CP 以及一些模拟量输入/输出模板和数字量输入模板。

可组态模板有的有后备电池，有的没有后备电池。

没有后备电池的模板在任何掉电之后都必须重新提供数据。这些模板的参数保存在 CPU 的可保持存储区域中(由 CPU 作间接的参数赋值)。

设置和装载参数

可以用 STEP 7 设置模板参数。当存储这些参数时，STEP 7 生成对象“系统数据块”，它与用户程序一起下载到 CPU，当 CPU 启动时被传送到模板上。

可以作哪些设置?

模板参数被分为参数块。在“S7-300 可编程控制器，硬件和安装”手册以及“S7-400，M7-400 可编程控制器，模板技术规范”参考手册中解释了哪些参数块在哪些 CPU 上有效的。

参数块举例：

- 启动性能
- 循环
- MPI
- 诊断
- 可保持数据
- 时钟存储
- 中断处理

- 集成I/O(只有S7-300)
- 保护等级
- 局域数据
- 实时时钟
- 异步错误

用 SFC 作参数赋值

除了用 STEP 7 作参数赋值外，还可以用 S7 程序的系统功能修改模板参数。下表说明哪些 SFC 传送哪些模板参数。

SFC	目的
SFC 55 WR_PARM	传送可修改的参数（数据记录1）到寻址的信号模板
SFC 56 WR_DPAPM	从相应的SDB传送参数（数据记录0或1）到寻址的信号模板
SFC 57 PARM_MOD	从相应的SDB传送所有参数（数据记录0或1）到寻址的信号模板
SFC 58 WR_REC	传送任意数据记录到寻址的信号模板

在“S7-300 和 S7-400 系统软件，系统和标准功能”参考手册中有关于系统功能的详细描述。

以下手册中解释了哪些模板参数可以被动态修改：

- “S7-300可编程控制器，硬件和安装”手册
- “S7-300，M7-300可编程控制器，模板技术规范”参考手册
- “S7-400，M7-400可编程控制器，模板技术规范”参考手册

A.7.2 离线更新模块或子模块的固件版本（操作系统）

按下列步骤向存储器卡传送更新文件：

1. 建立一个新的路径
2. 从更新软盘上将 UPD 文件拷贝到该目录
3. 选择菜单命令 **PLC > Update Operating System**
4. 在出现的对话框中选择 UPD 文件的目录
5. 选择任何一个 UPD 文件
6. 点击“OK”，退出对话框

将存储器卡插入 PLC 中。

执行操作系统的更新

1. 关掉 CPU 的电源(PS 模板)
2. 将准备好的装有操作系统更新文件的存储器卡插入 PLC 中
3. CPU 上电。从存储器卡中将操作系统传送到内部 FLASH-EPROM 中，操作期间，CPU

上的所有指示灯点亮

4. 大约两分钟后，操作系统完成更新。CPU 上的 STOP 指示灯通过慢闪来指示更新的完成(系统请求存储器复位)
5. 关掉电源，插入想要运行的存储器卡
6. 接通电源，CPU 将自动执行存储器复位，之后 CPU 就可以运行了。
7. 更新其它模块和子模块的固件版本。

A.7.3 使用时钟功能

所有的 S7-300/-400CPU 都配备有一个时钟(实时时钟或软件时钟)。这个时钟可在可编程控制器中，即能作主时钟也能作具有被外步同步的从时钟。日时钟中断和 CPU 运行时间计数器数器需要这个时钟。

时间格式

时钟总是指示时间(最小精度 1 秒)、日期和星期。有些 CPU 还可以指示毫秒(参考“S7-300 可编程控制器，硬件和安装”手册以及“S7-400，M7-400 可编程控制器模板技术规范”参考手册)。

设置和读时间

可以在用户程序中通过调用 SFC0 SET_CLK 设置 CPU 时间和日期时钟，或者在编程器上使用菜单选项启动时钟。使用 SFC1 READ_CLK 或编程器上的菜单选项可以读 CPU 当前的日期和时间。

为时钟赋值参数

如果在一个网络中不止一个模板配有时钟，必须用 STEP 7 设置参数指定当时间同步时哪个 CPU 功能为主，哪个为从。当设置参数时，还要决定时间是通过通讯总线被同步还是通过多点接口，以及对时间作自动同步的间隔。

同步时间

为确保网络中的所有模板的时间是一样的，从时钟则由系统程序以一个定期的(可选)间隔对它同步，也可以调用系统功能 SFC48 SFC_RTCB 将主时钟的日期和时间传送给从时钟。

使用一个 CPU 运行时间计数器数器

CPU 运行时间计数器对所连接的设备的操作时间或者 CPU 的总的运行时间进行计数。

在 STOP 模式，运行时间计数器数器停止，它的计数值在存储器复位后仍可保持。在暖起动期间，CPU 运行时间计数器必须被用户程序重新启动；在热起动期间，如果它以前就已被起动了，现在则可自动运行。

可以用 SFC2 SET_RTM 给 CPU 运行时间计数器设置一个初始值。可以用 SFC 3

CTRL_RTM 启动或停止 CPU 运行时间计数器。可以用 SFC4 READ_RTM 读当前总的操作小时数和计数器的状态(“停止”或“计数”)。

一个 CPU 最多可有八个 CPU 运行时间计数器。号码从 0 开始。

A.7.4 使用时钟存储器和定时器

时钟存储器

时钟存储器信号存储在位存储器的一个字节中，每个位的状态按照 1：1 的比例周期性地为 1 和为 0。当使用 STEP 7 为时钟存储器赋值参数时，可以选择在 CPU 上使用哪个位存储器字节。

用途

可以在用户程序中使用时钟存储器字节，例如，启动一个闪烁灯或触发一个周期性的操作(如，测量一个实际值)。

可能的频率

时钟字节的每一位被赋予一个频率。下表为赋值关系

时钟存储字节的位	7	6	5	4	3	2	1	0
周期(s)	2.0	1.6	1.0	0.8	0.5	0.4	0.2	0.1
频率(Hz)	0.5	0.625	1	1.25	2	2.5	5	10

提示

时钟存储字节与 CPU 循环不同步，也就是说，在一个长的周期内，时钟存储字节的状态可能会改变若干次。

定时器

定时器是系统存储器的一个存储区域。可以在用户程序中指定一个定时器功能(如延迟接通定时器)。可用的定时器的数据依据 CPU 而定。

提示

- 如果在用户程序中使用的定时器的数量超过了CPU所允许的，则出现同步错误且触发OB121启动。
- 在S7-300上(CPU318除外)，只能在OB1和OB100中同时启动并更新定时器，在其它OB中定时器只能被启动。