第十四章

宏指令的使用

14.1. 宏的种类 1

14.2. 宏的使用 4

14.2.1. 建立宏 4

14.2.2. 打开和关闭宏 4

14.2.3. 设定宏名称 5

14.2.4. 删除宏 5

14.2.5. 储存和导出宏 5

14.2.6. 宏对话框的设定 6

14.3. 宏的编辑 8

14.3.1. 宏编辑器窗口 8

14.3.2. 宏指令属性工具窗口 10

14.4. 宏指令和范例 11

14.4.1. 宏的符号和专有名词 11

14.4.2. 数据搬移和设值 13

14.4.3. 算术运算 14

14.4.4. 逻辑运算 16

14.4.5. 计算 18

14.4.6. 数据转换 20

14.4.7. 条件操作 24

14.4.8. 流程控制 29

14.4.9. 定时器操作 33

14.4.10. 键盘操作 34

14.4.11. 配方操作 35

14.4.12. 通讯操作 36

14.4.13. 系统服务 37
14.4.14. 画面操作 38
14.4.15. 文件操作 39
14.4.16. 比较指令 46
14.4.17. 字符串操作 48
14.4.18. 运行操作 56
14.4.19. 打印操作 57
14.4.20. 声音操作 60

本章节说明如何编写宏来操作目标人机(触控屏)。一个宏包含一系列的宏指令，当宏在人机上执行时，就如同执行简单的计算机程序，像流程控制、数据转换、条件操作、序列运算等基本对象不容易完成的工作，都可使用宏来轻松完成。

注意：请勿使用宏来控制系统 I/O 动作，这可能导致非常严重且威胁生命的伤害。
注意：人机(触控屏)运行时，其操作系统所要执行的工作非常多。所以宏内的程序应尽量减短，以免造成人机运作不正常。
注意：所有宏都是独立且个别执行。当各个宏之间共享一般功能时，人机可能会产生冲突。当循环宏在更新地址的值，而事件宏也同时在使用这个地址。此时，如果事件宏在循环宏使用前更改地址，则循环宏的结果会是不正确的。

14.1. 宏的种类

■ 全局宏
全局宏是指在同一个项目中，能被所有人机应用所使用的宏。在项目中的多个人机应用，使用全局宏即可共享同一宏功能，因此不需保留相同的本地宏。所以全局宏所规划的宏程序，可以在整个项目中，被全部的所有画面宏和对象宏所使用的称之全局宏。所以顾名思议，另一种为本地宏就只能被该单独的人机所使用。

只有全局宏可设为保护模式，也就是须输入正确密码才能查看和编辑宏指令，所以说可以充分保护设计者的特殊程序或重要程序的知识产权，例如将一段 PID 运算法则或通讯程序利用全局宏设计，然后就可以在各种对象中呼叫调用，但其内容非设计者就无法修改。要设置全局宏的密码保护请在项目栏的信息和保护项目中设定。

注意：全局宏只能使用内部变量。

■ 本地宏
本地宏只能使用于宏所在的人机应用。

■ 子宏
子宏是其它宏可以使用 CALL 指令，来执行的宏。当宏执行遇到 CALL 指令时，将停止目前宏的执行，并执行子宏。子宏最后一个指令必须是 RET，RET 会结束子宏的执行并且回去执行原先的宏。RET 指令也可以放在任意的位置。当子宏结束并回到原宏时，会接着 CALL 指令的下一行指令，继续执行之后的操作。

藉由执行子宏的一般功能，可使宏模块化、可共享、易于阅读和维护。

在该软件中我们已经提供有许多常用的宏操作，大约分为以下几类：
■ 整个应用提供了四种系统宏：**启始宏**，**主宏**，**事件宏**，**时间宏**。
■ 画面提供了三种宏：**开宏**，**关宏**，**循环宏**。
■ 对象提供了三种宏： **壹宏、零宏、和物件宏**。

请在适当的时机和目的使用适当的宏，可以让您整个人机应用得到最好的运行效益。

执行时机	执行目的
开机，当人机应用打开时，只执行一次(初始化用)。	**启始宏** 当触控屏运行时，第一次启动触控屏画面工作程序应用时(通常指断电后第一次重新送电执行应用画面程序)，只执行一次启始宏内的全部程序指令，然后才显示第一个画面。所以如果启始宏内的程序太长则第一次显示画面会需要较长开机时间，一般而言启始宏是用来执行初始化动作。可使用启始宏来初始化人机应用的资料和设定，启始宏可以在人机应用一般设置的对话框中设定。
开机后，当人机应用开始执行时，不断地执行。	**主宏** 当触控屏运行时，就会不断地循环执行主宏内的程序指令，但每次将只执行最多 30 行的宏指令。而且不论触控屏当前所在画面为何，此主宏均时时刻刻会被执行。只是主宏的执行周期时间是无法非常精准的确定时间值，通常大约为每 100-1000 毫秒(msec)会执行一次。(宏程序运行时间依程序长短而定)。也就是说当人机应用执行时，主宏会循环不断地执行，人机应用会在主宏执行完毕或执行到 END 指令时，回到第一个指令重复执行。主宏可以在人机应用一般设置的对话框中设定。
当指定的触发位从 0 变到 1 时，触发执行宏一次。	**事件宏** 当触控屏运行时，当触发位信号有效(由关状态=0 切为开状态=1)时，触控屏将立刻执行事件触发宏内的全部指令一次。一个人机应用最多可设定四个事件宏。事件宏可以在人机应用一般设置的对话框中设定。
依所设间隔时间循环执行宏程序。	**时间宏** 当触控屏运行时，触控屏将固定以所设时间周期循环运行时间宏内的全部指令一次。所以为了使触控屏的动作能合理运行，请尽量简短时间宏的指令数量。时间宏的执行是周期性地重复执行。一个人机应用最多可设定四个时间宏。每个时间宏可设定不同的间隔时间(间隔时间 0.5 或 1 秒或每准点分钟或每准点小时)。时间宏可以在人机应用一般设置的对话框中设定。
打开此画面时，执行宏一次。	**画面开宏** 当画面打开时，会先执行此宏内所有程序一次。等画面开宏执行完毕后，画面才会出现。开宏可以在画面属性的对话框中设定。
显示此画面时，依所设间隔时间循环执行宏程序。	**画面循环宏** 当画面打开后，循环宏会循环不断地执行。也就是说，循环宏从第一个指令开始，直到宏关闭或执行到 END 指令时，再从第一个指令开始循环，当画面关闭时循环宏也会立刻关闭。循环宏可以在画面属性的对话框中设定。循环宏的程序指令间隔时间 0.1、0.2、1.0 sec.区间可设。
当关闭此画面时，执行宏丨次。	**画面关宏** 当画面要被关闭时，会执行此宏内所有全部程序一次，当关宏执行完毕后，然后画面才会关闭。关宏可以在画面属性的对话框中设定。

接下页

执行时机	执行目的
按钮为 1 时，执行宏一次。当按压或放开按钮使控制接点为 1 时。	**壹宏** 当按压或放开按钮使控制接点为 1 时，会执行壹宏内所有程序一次。当壹宏结束，位的设定才会执行。所以，壹宏的长度应该要尽可能缩短，才不会延迟位的执行。只有位按钮及切换开关可设定壹宏。壹宏可以在按钮的对话框中设定。
按钮为 0 时，执行宏一次。当按压或放开按钮使控制接点为 0 时。	**零宏** 当按压或放开按钮使控制接点为 0 时，会执行零宏内所有程序一次。当零宏结束，位的设定才会执行。所以，零宏的长度应该要尽可能缩短，才不会延迟位的执行。只有位按钮和切换开关可设定零宏。零宏可以在按钮的对话框中设定。
当对象执行操作时，执行特定宏一次。	**物件宏** 当对象执行特定操作时，会执行对象宏一次。在操作前或操作后执行宏，是依据该操作的种类。画面按钮、功能按钮和键盘按钮可设定物件宏。对象宏可以在对象的对话框中设定。 画面按钮宏：当执行换画面按钮时，执行该宏一次。 功能按钮宏：选择文件的功能按钮对象提供了一个宏。 键盘按钮宏：当执行键盘按钮时，执行宏。 进阶数值显示宏：当执行高级数值显示器时，可以执行显示宏。 进阶数值输出宏：当执行高级数值显示器输入操作时，可以执行输出宏。

14.2. 宏的使用

14.2.1. 建立宏

■ 建立一个新的宏

1) 在项目 > 全局宏，点选新增…按钮即可建立全局宏。或是在项目管理员 > 全局 > 全局宏中，点击鼠标右键后，从弹出的菜单中点选新增宏。

   在人机应用 > 宏，点选新增…按钮即可建立本地宏。或在项目管理员 > 人机应用 > 宏中，点选鼠标右键后，从弹出的菜单中点选新增宏。

2) 在新增宏对话框中，输入宏名称，然后按下回车键或用鼠标点击确定键，然后完成建立宏。

■ 导入宏

1) 在项目管理员中 > 全局 > 全局宏中，点击鼠标右键后，从弹出的菜单中点选导入宏…，即可导入现有的宏为全局宏。

   在项目管理员中 > 人机应用 > 宏中，点击鼠标右键后，从弹出的菜单中点选导入宏…，即可导入现有的宏为本地宏。

2) 从所需的*.mcr 或*.txt 文件中，点选并建立新的宏。如果想要导入的宏在不同的文件夹，请先点选该文件夹。

3) 用鼠标点击打开旧文件。所建立的宏就是原始建立的宏。

14.2.2. 打开和关闭宏

■ 打开现有的宏

在项目 > 全局宏 > 编辑中，点选打开所需的全局宏，或在项目管理员>全局中，用鼠标双击全局宏，或从物件对话框的宏设定中，选择下拉式清单中，在"----------全局----------"后的全局宏。

在人机应用 > 宏 > 编辑中，点选打开所需的本地宏，或在项目管理员>人机应用中，用鼠标双击本地宏，或从物件对话框的宏设定中，选择下拉式清单中，在"----------全局----------"前的本地宏。

■ 在宏编辑器窗口中，打开*.txt 或*.mcr 文件：

可利用拖放完成：

1) 点选现有的宏来打开宏编辑器窗口。

2) 拖曳.mcr 或.txt 文件至宏编辑器中后，放开鼠标即可。

注意：在宏编辑器窗口中的原宏将会被读入的宏所取代。

■ 关闭宏编辑器窗口：

要关闭单一窗口，选择该窗口并点击关闭钮。

要关闭所有窗口，选择窗口菜单中的 Windows…，在窗口对话框中，选择所有想要关闭的窗口，并按下关闭窗口钮。

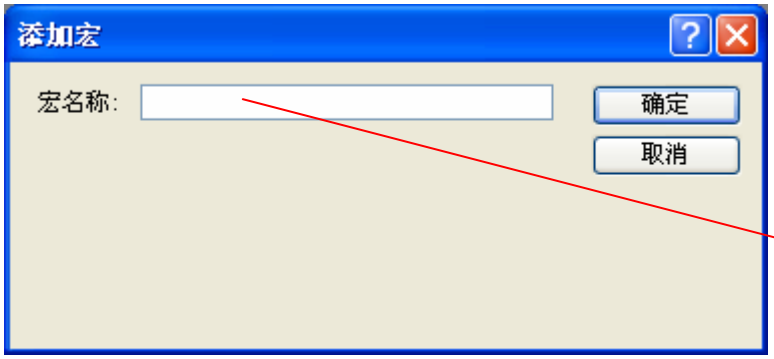
注意：当关闭宏编辑器窗口时，宏指令属性窗口会自动关闭。即使关闭宏编辑器窗口，宏的变更仍会储存，除非关闭该软件软件时没有储存变更。

■ 关闭宏指令属性窗口：

点击在宏指令属性窗口的关闭钮或勾选/取消在查看菜单中的宏指令属性，来关闭宏指令属性窗口。

14.2.3. 设定宏名称

当新增全局或本地宏，必须在以下的窗口中输入宏的名称。



在此输入宏的名称。最大的长度是256字符。宏的名称是没有大小写之分的。比如：TURN ON 和 turn on 是一样的名称。

当导入文件为宏时，默认的宏名称就是该文件的名称。在每一个人机应用中，本地宏的名称必须是唯一的，但可与全局宏的名称相同。

■ 从项目管理员中更改宏名称：

1) 点选欲更名的宏。
2) 点击鼠标右键后，弹出菜单，然后点选更名。
3) 输入新的宏名称后，按下回车键后完成。

14.2.4. 删除宏

■ 从项目管理员中删除宏：

1) 点选欲删除的宏。
2) 点击鼠标右键后，弹出菜单，然后点选删除。

■ 从菜单中删除宏

在项目菜单中，选取全局宏、删除的子菜单，点选欲删除的全局宏。在人机应用菜单中，选取本地宏、删除的子菜单，点选欲删除的本地宏。

注意：一次只能删除一个宏。如果要删除的宏已经被人机应用或对象使用中，会出现确认删除窗口。

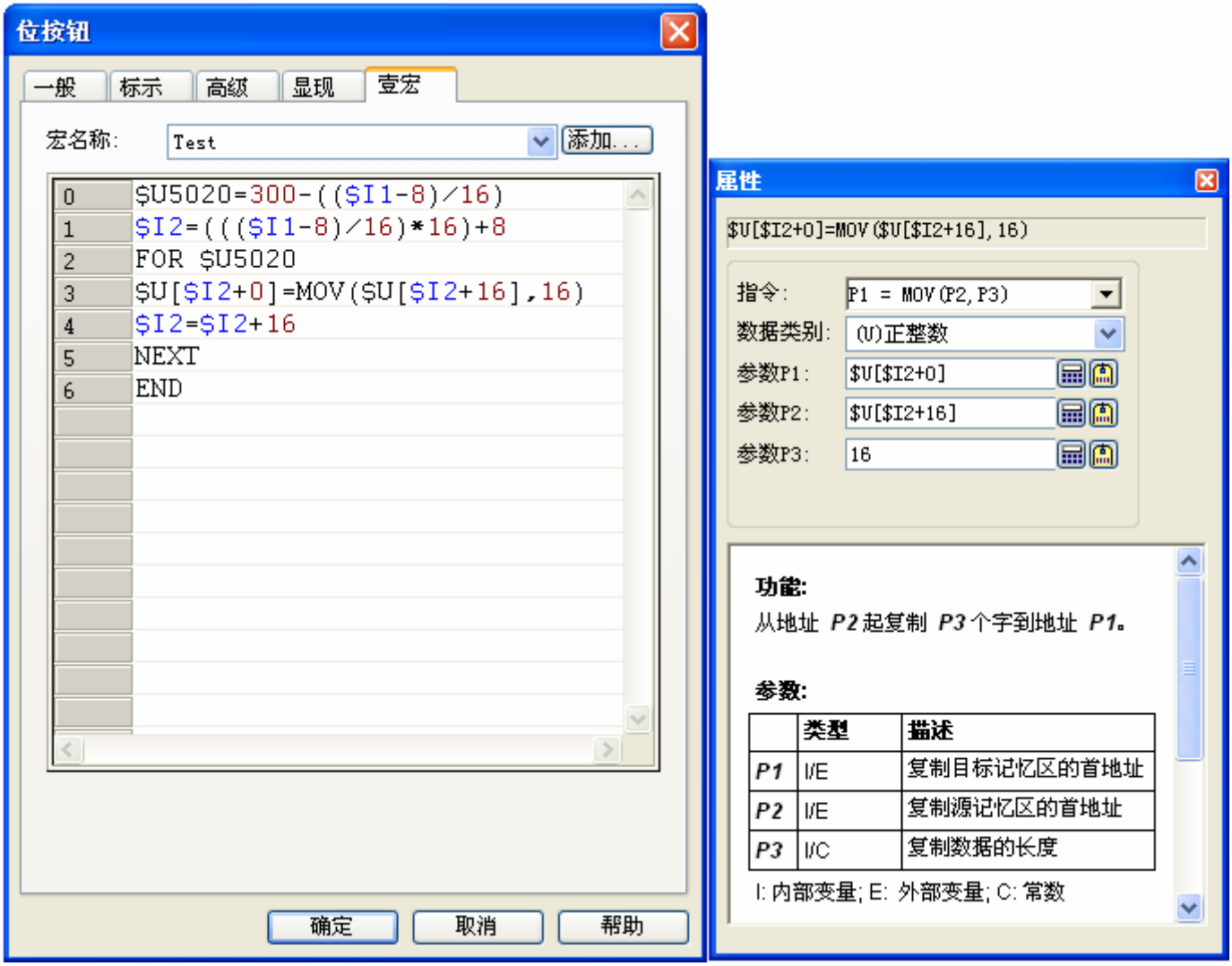
14.2.5. 储存和导出宏

如果想让宏在其它人机应用中使用，可将宏导出为.txt 或.mcr 文件，步骤如下：

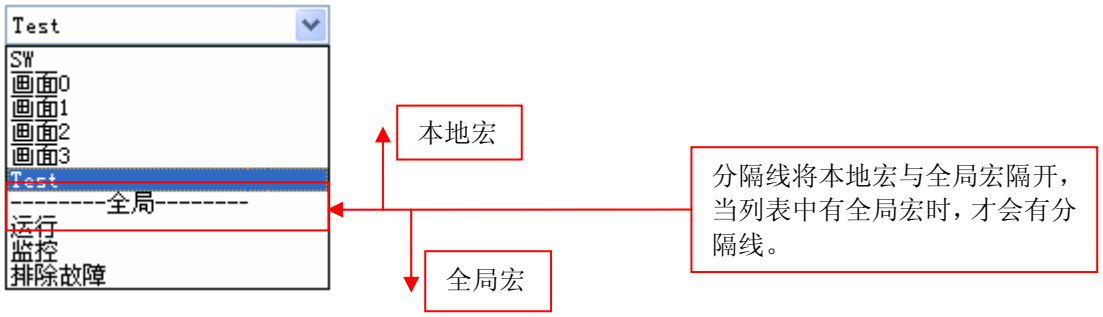
1) 点选欲导出的宏。
2) 点击鼠标右键后，弹出菜单，然后点选导出宏。
3) 如果要将宏存储在其它的文件夹，点选要储存的文件夹，然后用鼠标点击存盘。

14.2.6. 宏对话框的设定

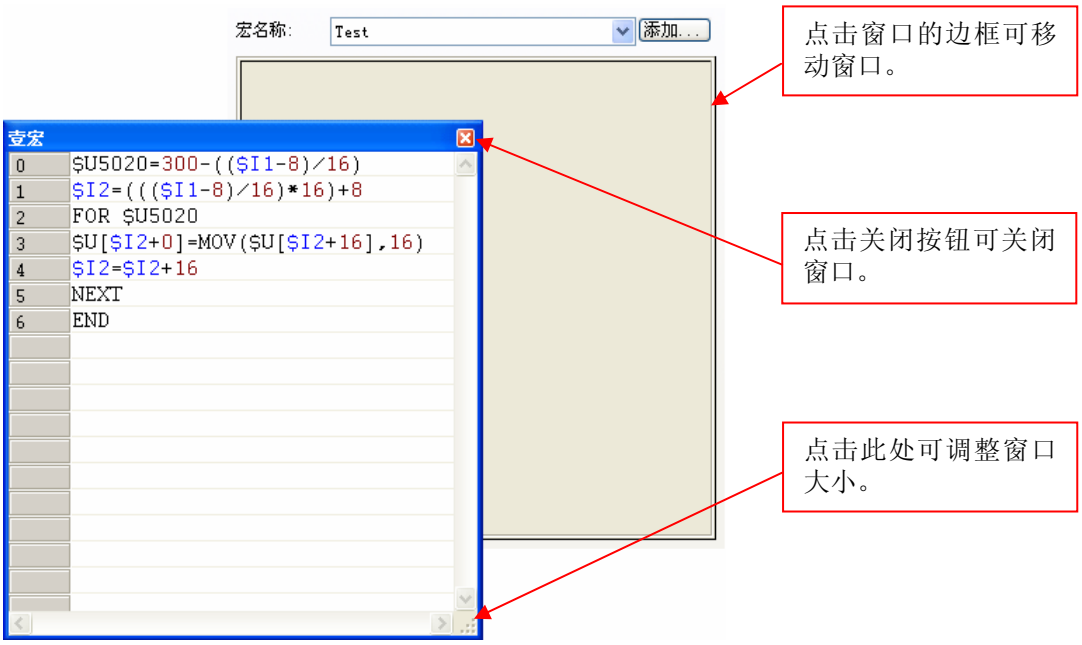
用户可以在有宏页面的对话框中，打开和编辑宏或建立一个新的宏。下面范例是位按钮的宏页面的范例。



下表为一般页面各项属性的说明：

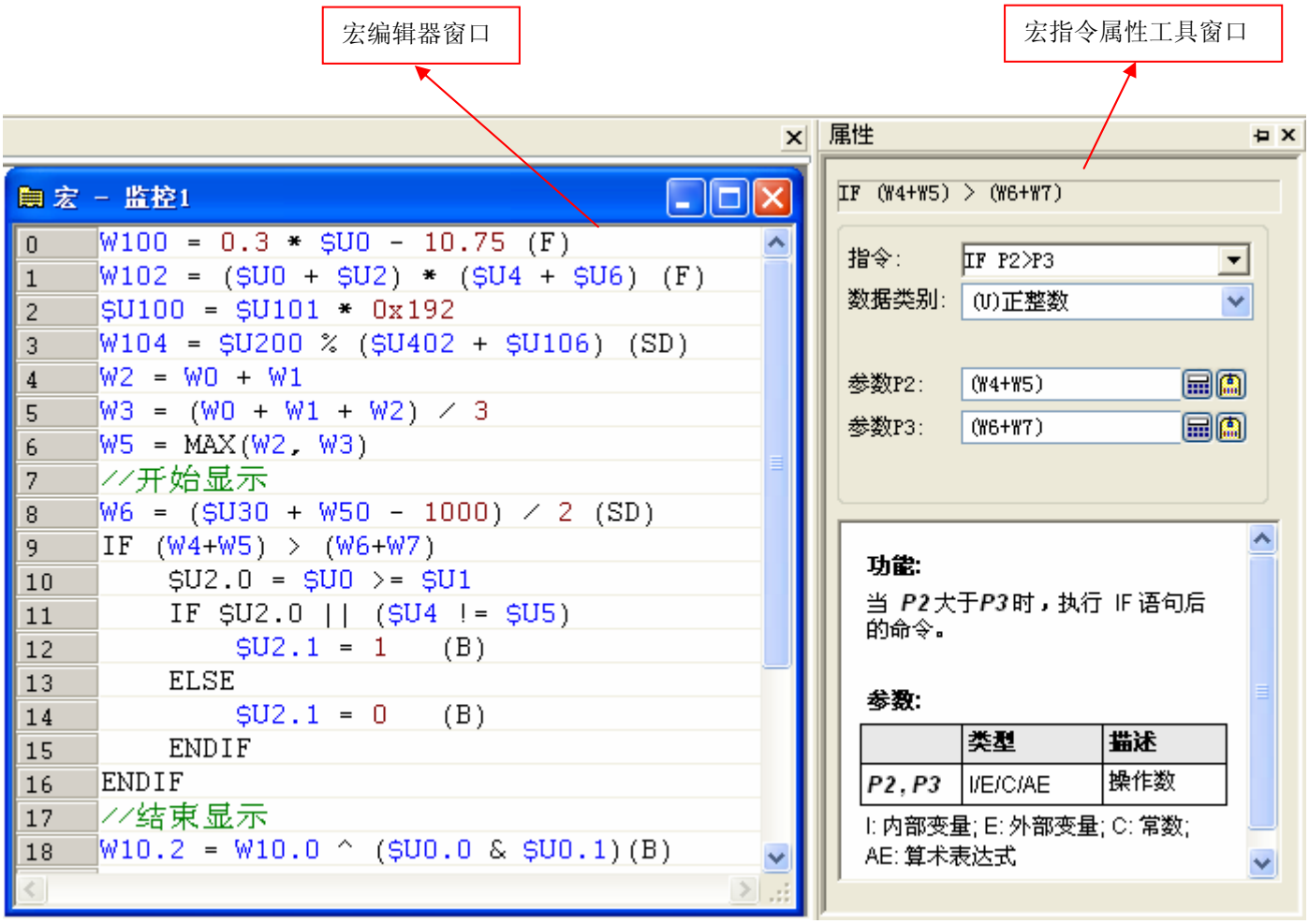
属性	说明
宏名称	从下拉式清单中选择现有的本地或全局宏。下面是范例： 
新增...	点选此处出现新增宏对话框，可建立新的宏。

接下页

属性	说明
宏编辑器窗口	在宏编辑器窗口中编写宏。参考 第 14.3.1 节，取得更详细的说明。如果编辑器窗口太小，你可以拖曳窗口来重设大小。要移动窗口，在窗口的边框上点击并按住鼠标左键不放，拖曳鼠标来移动窗口，当拖曳窗口时，该窗口会置于最上层，使你可以任意移动窗口。然后，在你想要放置窗口的位置，放开鼠标左键。点击在窗口右下角重设大小的卷标，可调整窗口大小。点击关闭按钮来关闭窗口。下面是宏编辑器窗口的范例： 
指令属性编辑	在属性对话框可设定宏指令属性编辑。参考 第 14.3.2 节，取得更详细的说明。宏属性对话框可以移动、重设大小。但必须先关闭宏编辑器窗口，才能关闭属性对话框。

14.3. 宏的编辑

在该软件中，编写宏的地方包含两个部分：宏编辑器窗口和宏指令属性工具窗口。
当你从项目管理员中打开宏时，将会看见以下的范例：



14.3.1. 宏编辑器窗口

宏编辑器是一种文字的编辑器，有语法标色和给予行编号。窗口左边每一行的编号，使指定宏的特定位置容易。不同的语法标色使宏的架构清楚，利用不同的颜色区分不同的语法。如：关键词是黑色、批注是绿色、地址是蓝色、常数是红色。

■ 编辑宏

用户可以使用菜单的指令、组合键和拖放三种操作方式，将选取的文字剪切、复制或粘贴。你也可以撤销或重做。
点击鼠标右键弹出编辑指令菜单。出现的编辑指令依箭头所指的位置而定。

宏编辑器可进行以下的编辑动作：

- 剪切、复制、粘贴和删除选取单行、多行或文字。
- 撤销或重做。
- 在同一个窗口或两个不同的窗口中，使用拖放来移动、复制所选的文字。

下表为支持的编辑指令。

菜单指令	组合键	说明
剪切	CTRL+X	从宏编辑器窗口将选的文字移除。
复制	CTRL+C	从宏编辑器窗口复制选取的文字。
粘贴	CTRL+V	将复制或剪切的文字粘贴宏窗口中。
删除	DELETE	直接删除文字，不放到剪贴板中。
撤销	CTRL+Z	回到上一个编辑动作。
重做	CTRL+Y	重做被撤销的编辑动作。
全选	CTRL+A	选取宏编辑器中所有的文字。

注意：所有的编辑指令需要先选取才能进行编辑操作。一些指令可以在光标所在的位置选取。

■ 宏的批注

批注是宏在执行时并不会被执行的补助说明。宏支持单行批注或区块批注。单行批注以双斜线开始(//)，到该行最后结束。下面是单行批注的范例：IF $U0.0 (B) // 按下按键

区块批注以开始竖线(/*)开始，到结束竖线(*/)结束。下面是区块批注的范例：

```
/* $N1001=WH2021
   $N1010=$N1001 */
```

■ 设定宏的常数

设定十六进制数，使用 h 或 H 做字尾。例如：12abH 和 3ABh 是有效的十六进制数。另外，也可使用 0x 或 0X 做前缀，例如：0x1278abc 和 0XFFFF0000 是有效的十六进制数。

设定二进制数，使用 b 或 B 做字尾，例如：001100111b 和 11110000B 是有效的二进制数。

十进制则输入要设定常数的数字即可。然而，当常数与有效的外部变量相同时，会产生混淆。例如：如果人机应用连接到一个 Modicon ModBus 从设备，则无法区分 40001 是常数还是控制器的地址。为避免这样情形产生，用以下的方法表示数字为常数：

1) 使用 K、k、D 或 d 作为整数的字尾，例如：-123K 和-123d 是有效的常数-123。
2) 使用 F 或 f 作为有小数数字的字尾，例如：-12.3F 或-12.3f 是有效的常数-12.3。

14.3.2. 宏指令属性工具窗口

宏指令属性工具窗口可帮助你快速、简单地增加或修改宏指令。

打开项目管理员或是菜单，打开宏后，宏属性工具窗口打开后为停靠窗口。停靠工具窗口会自动地显示、隐藏、以卷标页型式与其它工具窗口链接或停靠在窗口边缘或自由浮动。当宏编辑器打开时，你可以选择打开或关闭宏指令属性工具窗口，点选查看菜单中的宏指令属性即可。

如果你从对象的对话框打开宏，宏属性工具窗口会漂浮在宏编辑器的旁边。宏属性工具窗口可以任意移动，但无法关闭。

下表为宏属性工作窗口中，各种属性的说明。

项目属性		说明
指令		点击下拉式清单，出现宏指令选择对话框。从卷标页中浏览宏指令的关键词，然后，点选宏指令。指令的格式在对话框关闭后，将会出现在下拉式清单中。点选宏指令选择对话框外的任何区域，即可取消操作。
数据类别		从下拉式列表选择宏指令的数据类别。不同的宏指令支持不同的数据类别。数据类别如下：**(S)**整数，**(U)**正整数，**(SD)**32 位整数，**(UD)**32 位正整数，**(F)**浮点数，**(B)**位/接点。
参数	输入栏	数据类别是**(B)**时，输入位变量。 数据类别是**(U)/(S)**时，输入字变量或常数。 数据类别是**(UD)/(SD)/(F)**时，输入双字变量或常数。
		点击此图标，出现变量地址输入键盘，可设定地址。
		点击此图标，出现选取标签对话框，可选取卷标变量。
宏指令提示窗口		显示宏指令的操作、参数类别。

注意：在属性对话框中的任何修改都会改变在宏编辑器的宏指令。

14.4. 宏指令和范例

14.4.1. 宏的符号和专有名词

下面是宏指令和范例所使用的符号和专有名词。

■ **符号**

1) *P1, P2, P3, P4, P5*: 宏指令的参数。
2) I, E, C, A, CS, M, AE, CE: 指示宏指令可使用的参数种类。

缩写	参数种类
I	内部变量
E	外部变量
C	常数
A	ASCII 文字符串
CS	文字标记对应宏程序
M	子宏名称
AE	算术表达式
CE	比较表达式

3) U, S, UD, SD, F, B: 指出宏指令可支持的数据类别。

缩写	数据类别
U	16 位正整数
S	16 位正负整数
UD	32 位正整数
SD	32 位正负整数
F	32 位浮点数
B	位

■ **专有名词**

专有名词	定义
内部记忆区	在 HMI 的系统记忆区中，可被人机应用所存取的部分。例如：用户记忆区$U，非挥发性记忆区$N(有停电保持)，系统记忆区$S 和配方记忆区$R(有停电保持)，都是内部记忆区。另外$D 和$W 是专门提供给“高阶数值运算宏”的暂时性运算内存。
内部变量	对应到内部记忆区的地址或卷标。
内部位变量	内部位变量指的是在内部记忆区的一个位。为了阅读方便，在不会产生混淆的前提下，文章中我们通常用“内部变量”，而不用“内部位变量”。
内部字变量	内部字变量指的是在内部记忆区的一个字。这个变量同时还可代表双字、字节区块(字节数组)、字区块(字数组)和双字区块(双字数组)。 为了阅读方便，在不会产生混淆的前提下，文章中我们通常用“内部变量”，而不用“内部字变量”。
外部记忆区	在控制器的记忆区或是设备的可寻址的数据地址，人机应用可经由通信链接来存取的部份。

接下页

<table>
<tr><th>专有名词</th><th>定义</th></tr>
<tr><td>外部变量</td><td>在外部记忆区的变量数据地址或卷标。</td></tr>
<tr><td>外部位变量</td><td>外部位变量指的是在外部记忆区的一个位。为了阅读方便，在不会产生混淆的前提下，文章中我们通常用外部变量，而不用外部位变量。</td></tr>
<tr><td>外部字变量</td><td>外部字变量指的是在外部记忆区的一个字。如果相关地址存取单位是字，这个变量同时还可代表双字、字节区块(字节数组)、字区块(字数组)和双字区块(双字数组)。如果相关地址存取单位是双字，这个变量只可代表双字或长度是 4 的倍数的记忆区块。为了阅读方便，在不会产生混淆的前提下，文章中我们通常用外部变量，而不用外部字变量。</td></tr>
<tr><td>表达式</td><td>
<table>
<tr><th>种类</th><th>缩写</th><th>说明</th></tr>
<tr><td>算术表达式</td><td>AE</td><td>操作数和参数的顺序是用来计算参数的值。</td></tr>
<tr><td>比较表达式</td><td>CE</td><td>操作数和参数的顺序是用来比较参数的值。</td></tr>
</table>
本软件提供以下种类的操作数：
<table>
<tr><th>操作数</th><th>名称或意义</th><th>结合规律</th><th>使用表达式</th></tr>
<tr><td>( )</td><td>括号</td><td>由左至右</td><td>AE/CE</td></tr>
<tr><td>*</td><td>乘法</td><td>由左至右</td><td rowspan="8">AE</td></tr>
<tr><td>/</td><td>除法</td><td>由左至右</td></tr>
<tr><td>%</td><td>取余数</td><td>由左至右</td></tr>
<tr><td>+</td><td>加法</td><td>由左至右</td></tr>
<tr><td>-</td><td>减法</td><td>由左至右</td></tr>
<tr><td><<</td><td>左移</td><td>由左至右</td></tr>
<tr><td>>></td><td>右移</td><td>由左至右</td></tr>
<tr><td><</td><td>小于</td><td>由左至右</td><td rowspan="6">CE</td></tr>
<tr><td>></td><td>大于</td><td>由左至右</td></tr>
<tr><td><=</td><td>小于等于</td><td>由左至右</td></tr>
<tr><td>>=</td><td>大于等于</td><td>由左至右</td></tr>
<tr><td>==</td><td>等于</td><td>由左至右</td></tr>
<tr><td>!=</td><td>不等于</td><td>由左至右</td></tr>
<tr><td>&</td><td>相对位逻辑_与</td><td>由左至右</td><td rowspan="3">AE</td></tr>
<tr><td>^</td><td>相对位逻辑_互斥</td><td>由左至右</td></tr>
<tr><td>|</td><td>相对位逻辑_或</td><td>由左至右</td></tr>
<tr><td>&&</td><td>位逻辑_与</td><td>由左至右</td><td rowspan="2">CE</td></tr>
<tr><td>||</td><td>位逻辑_或</td><td>由左至右</td></tr>
<tr><td>=</td><td>设值</td><td>由右至左</td><td>AE/CE</td></tr>
</table>
注意：上表是依照操作数的优先执行顺序排序上到下(由高至低)。在表中，相同的表达式有相同的优先级。除非表达式有出现括号，否则依从由左至右的顺序执行运算。
</td></tr>
</table>

14.4.2. 数据搬移和设值

Assignment (=) →设值

指令格式	***P1 = P2***	数据类型	U/S/UD/SD/F/B
功能	将 ***P2*** 的值存储到 ***P1*** 中。		
P1 (I/E)	目标值。		
P2 (I/E/C/AE)	原始值。		
例 1	$U2 = 123.45 (F)　/* 将浮点数 123.45 指派给$U2(包括$U3)。 */		
例 2	$U100.f = 1 (B)　/* 将 On 信号指派给指定地址位。 */		
例 3	W60 = ($U30 + $W50 - 1000) / 2 (SD)　/* 算术表达式的运算结果指派给 W60。 */		
例 4	V0.0 = 2\M0 (B)　/* 将连接 2 上的 M0 的值赋给连接 1 上的 V0.0。*/		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

Logical NOT (= !) →位逻辑反相

指令格式	***P1 = ! P2***	数据类型	B
功能	对 ***P2*** 取反，并将运行结果保存到 ***P1***。		
P1 (I/E)	位逻辑运算结果。		
P2 (I/E)	运算位。		
例 1	$U2.3 = ! $U3.4 (B) /* 当$U3.4 为 1(On)时，则$U2.3 为 0(Off)。 */		
例 2	V0.0 = ! 2\M0 (B)　/*将连接 2 上的 M0 的值赋给连接 1 上的 V0.0。 */		

I: 内部变量; E: 外部变量

" " ASCII →设值

指令格式	***P1 = "P2"***
功能	将 ***P2*** 的文字符串转换为 ASCII 码复制到 ***P1*** 的目标位置。需要注意的是这个字符串是一个带空格中止符的字符串 00h，如果这个字符串的长度是 N，那么复制给 ***P1*** 的有 N+1 个字且最后一个字为 0。
P1 (I)	运算结果。
P2 (A)	被复制的字符串
例 1	$U60 = "TEST" /* $U60=7748 ("T"+"E")、$U61=1587 ("S"+"T")、$U62 的低字节值为 0　*/
例 2	$U20 = "ABCDE" /* 则 $U20=4241 H (BA)，$U21=4443 H (DC)，$U22=0045 H (E)。$U22 的高字节为空格字符串 0x00。 */

I: 内部变量;　A: ASCII 码字符串

MOV →区块搬移

指令格式	***P1* = MOV*(P2,P3)***	数据类型	U
功能	从地址 ***P2*** 起复制 ***P3*** 个字到地址 ***P1***。		
P1 (I/E)	复制目标记忆区的首地址。		
P2 (I/E)	复制来源记忆区的首地址。		
P3 (I/C)	复制数据的长度。		
例 1	$U100 = MOV($U200, 16) /* 从$U200 起复制 16 个字缓存器的数据给$U100。*/		
例 2	W60 = MOV($U200, $U2) /* 从$U200 起复制一个数据组给 W60，数据组的长度由$U2 决定。*/		
例 3	$U10 = MOV(2\D100, 16) /* 将 Link2 PLC 的 D100~D109 缓存器的值复制到触控屏内部记忆区 $U10~19。*/		

I: 内部变量; E: 外部变量; C: 常数

SETM →多笔设值

指令格式	***P1* = SETM*(P2,P3)***	数据类型	U
功能	将 ***P2*** 的值赋值给以 ***P1*** 为起始地址的 ***P3*** 个字中。		
P1 (I/E)	被赋值记忆区的起始地址。		
P2 (I/C)	被赋值的值或存储该值的地址。		
P3 (I/C)	被赋值记忆区的大小，单位：字。		
例 1	$U100 = SETM(0, 16) /* 将数值 0 设定给$U100～$U115。*/		
例 2	W60 = SETM($U200, $U2) /* 将$U200 的值赋给以 W60 为起始地址，$U2 为大小的记忆区中。*/		

I: 内部变量; E: 外部变量; C: 常数

14.4.3. 算术运算

Addition (+) → 加法

指令格式	***P1 = P2 + P3***	数据类型	U/S/UD/SD/F
功能	将 ***P2*** 加上 ***P3*** 的运算结果保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2,P3 (I/E/C/AE)	操作数。		
例 1	$U100 = $U101 + $U102 (U)		
例 2	W100 = 0.3*$U0 + 0.1*$U2 + 0.6*$U4 (F)		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

Subtraction (-) →减法

指令格式	***P1 = P2 - P3***	数据类型	U/S/UD/SD/F
功能	将 ***P2*** 减去以 ***P3*** 的运行结果保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2,P3 (I/E/C/AE)	操作数。		
例 1	$U100 = $U101 - $U102 (U)		
例 2	W100 = 0.3*$U0 - 10.75 (F)		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

Multiplication (*) →乘法

指令格式	***P1 = P2 * P3***	数据类型	U/S/UD/SD/F
功能	将 ***P2*** 乘以 ***P3*** 的运算结果保存到 ***P1***。		
P1 (I/E)	运算结果，如果运算结果产生了溢出，溢出的高位将被释放，低位则保存到 ***P1***。双字的乘法则 ***P1*** 是双字(double words)，***P2*** 单字(word)乘 ***P3*** 单字(word)的乘法则 ***P1*** 是单字(word)。		
P2,P3 (I/E/C/AE)	操作数。		
例 1	$U100 = $U102 * 0x192		
例 2	W100 = ($U0 + $U2) * ($U4 + $U6) (F)		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

Division (/) →`除法

指令格式	***P1 = P2 / P3***	数据类型	U/S/UD/SD/F
功能	将 ***P2*** 除以 ***P3*** 的运算结果保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2,P3 (I/E/C/AE)	操作数。		
例 1	$U100 = $U101 / $U102 (U)		
例 2	W100 = ($U0 + $U2) / ($U4 + $U6) (F)		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

Modulus (%) →取余数

指令格式	***P1 = P2 % P3***	数据类型	U/S/UD/SD
功能	将 ***P2*** 除以 ***P3*** 后的余数保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2,P3 (I/E/C/AE)	操作数。		
例 1	$U100 = $U30 % 16(U)		
例 2	W100 = $U200 % ($U402 + $U106) (SD)		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

14.4.4. 逻辑运算

Bitwise Inclusive OR (|) →相对位逻辑_或(OR)

指令格式	***P1 = P2 | P3***	数据类型	U/UD/B
功能	将 ***P2*** 和 ***P3*** 按位进行或运算，并将运行结果保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2,P3 (I/E/C)	操作数。		
例 1	$U100 = $U101 \| W60 (U) /* 当$U101 为 0000111100001111(B)，而 W60 为 1111000000001111(B)时，则$U100 为 1111111100001111(B)。 */		
例 2	W200 = W100 | 0xffff0000 (UD)		
例 3	B15 = $U1.2 \| B14 (B) /* 当$U1.2 为 0(Off)且 B14 也为 0(Off)时，则 B15 为 0(Off)其它情况为 1(On)。 */		

I: 内部变量; E: 外部变量; C: 常数

Bitwise AND (&) →相对位逻辑_与(AND)

指令格式	***P1 = P2 & P3***	数据类型	U/UD/B
功能	将 ***P2*** 和 ***P3*** 按位进行与运算，并将运行结果保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2,P3 (I/E/C)	操作数。		
例 1	$U100 = $U101 & W60 (U) /* 当$U101 为 0000111100001111(B)，而 W60 为 1111000000001111(B)时，则$U100 为 0000000000001111(B)。 */		
例 2	W200 = W100 & 0xffff0000 (UD)		
例 3	B15 = $U1.2 & B14 (B) /* 当$U1.2 为 1(On)且 B14 也为 1(On)时，则 B15 为 1(On)其它情况为 0(Off)。 */		

I: 内部变量; E: 外部变量; C: 常数

Bitwise Exclusive OR (^) →相对位逻辑_异或(XOR)

指令格式	***P1 = P2 ^ P3***	数据类型	U/UD/B
功能	将 ***P2*** 和 ***P3*** 按位异或，并将运行结果保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2,P3 (I/E/C)	操作数。		
例 1	$U100 = $U101 ^ W60 (U) /* 当$U101 为 0000111100001111(B)，而 W60 为 1111000000001111(B)时，则$U100 为 1111111100000000(B)。*/		
例 2	W200 = W100 ^ 0xffff0000 (UD)		
例 3	B15 = $U1.2 ^ B14 (B) /* 当$U1.2 为 1(On)且 B14 也为 1(On)时，则 B15 为 0(Off)，当$U1.2 为 1(On)而 B14 为 0(Off)时，则 B15 为 1(On)。*/		

I: 内部变量; E: 外部变量; C: 常数

Left Shift (<<) →左移

指令格式	***P1 = P2 << P3***	数据类型	U/UD
功能	将 ***P2*** 的值向左移动 ***P3*** 个位(bits)，并将运算结果保存到 ***P1***。这个运算操作只支持逻辑移动。		
P1 (I/E)	运算结果。		
P2 (I/E/C)	被移动的值。		
P3 (I/E/C)	被移动的位数(bits) 。		
例 1	$U100 = $U101 << 8 (U)		
例 2	W200 = W100 << $U10 (UD)		

I: 内部变量; E: 外部变量; C: 常数

Right Shift (>>) →右移

指令格式	***P1 = P2 >> P3***	数据类型	U/UD
功能	将 ***P2*** 的值向右移动 ***P3*** 个位(bits)，并将运算结果保存到 ***P1***。这个运算操作只支持逻辑移动。		
P1 (I/E)	运算结果。		
P2 (I/E/C)	被移动的值。		
P3 (I/E/C)	被移动的位数(bits) 。		
例 1	$U100 = $U101 >> 8 (U)		
例 2	W200 = W100 >> $U10 (UD)		

I: 内部变量; E: 外部变量; C: 常数

Logical AND (&&) →位逻辑_与

指令格式	***P1 = P2 && P3***	数据类型	B
功能	若位 ***P1*** 和位 ***P2*** 的值都为 1，那么将位 ***P3*** 置 1，否则清 0。		
P1 (I/E)	运算结果。		
P2,P3(I/E/C)	操作数。		
例 1	$U100.0 = $U101.0 && $U101.1 (B)		

I: 内部变量; E: 外部变量; C: 常数

Logical OR (||) →位逻辑_或

指令格式	***P1 = P2 || P3***	数据类型	B
功能	若位 ***P1*** 和位 ***P2*** 之中至少有一个的状态值为 1，那么将位 ***P3*** 置 1，否则清 0。		
P1 (I/E)	运算结果。		
P2,P3(I/E/C)	操作数。		
例 1	$U100.0 = $U101.0 || $U101.1 (B)		

I: 内部变量; E: 外部变量; C: 常数

14.4.5. 计算

MAX →两值比较取较大值

指令格式	***P1* = MAX*(P2,P3)***	数据类型	U/S/UD/SD/F
功能	在 ***P2*** 和 ***P3*** 两个数据中取较大值，并将该值保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2,P3(I/E/C)	操作数。		
例 1	$U100 = MAX(100, 200) /*将常数 200 保存到$U100。 */ $U100 = MAX($U10, $U11) /* 假设 $U10＝1、$U11＝5，则$U100＝5。 */		

I: 内部变量; E: 外部变量; C: 常数

MIN →两值比较取较小值

指令格式	***P1* = MIN*(P2,P3)***	数据类型	U/S/UD/SD/F
功能	在 ***P2*** 和 ***P3*** 两个数据中取较小值，并将该值保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2,P3(I/E/C)	操作数。		
例 1	$U100 = MIN(100, 200) /* $U100=100。*/ $U100 = MIN($U10, $U11) /* 假设 $U10＝1、$U11＝5，则$U100＝1。*/		

I: 内部变量; E: 外部变量; C: 常数

BMAX →数组取最大值

指令格式	***P1* = BMAX*(P2,P3)***	数据类型	U/S/UD/SD/F
功能	在以 ***P2*** 为起始地址，以 ***P3*** 为数据长度的一组数组数据中，查找最大值，并将该值保存到 ***P1***。		
P1 (I)	运算结果。		
P2 (I)	数据组（数组）的起始地址。		
P3 (I/C)	数据组（数组）的长度。		
例 1	$U100 = BMAX($U200, 16) (F) /* 以$U200 为起始地址，连续 16 个浮点数，查找最大值，并将该值保存到$U100。*/		

I: 内部变量; C: 常数

BMIN →数组取最小值

指令格式	***P1* = BMIN(*P2*,*P3*)**	数据类型	U/S/UD/SD/F
功能	在以 ***P2*** 为起始地址，以 ***P3*** 为数据长度的一组数组数据，查找最小值，并将该值保存到 ***P1***。		
P1 (I)	运算结果。		
P2 (I)	数据组（数组）的起始地址。		
P3 (I/C)	数据组（数组）的长度。		
例 1	$U100 = BMIN($U200, 60) (F) /* 以$U200 为起始地址的连续 60 个浮点数，查找最小值，并将该值保存到$U100。*/		

I: 内部变量; C: 常数

SUM →数组加总

指令格式	***P1* = SUM(*P2*,*P3*)**	数据类型	U/S/UD/SD/F
功能	计算出以 ***P2*** 为起始地址，以 ***P3*** 为数据长度的一组数据的总和，并将运算结果保存到 ***P1***。		
P1 (I)	运算结果。		
P2 (I)	数据组（数组）的起始地址。		
P3 (I/C)	数据组（数组）的长度。		
例 1	$U100 = SUM($U200, 16) (F) /* 计算出以$U200 起始地址的连续 16 个浮点数的总和，并将运算结果保存到$U100。*/		

I: 内部变量; C: 常数

XSUM →逻辑异或(XOR) 加总和

指令格式	***P1* = XSUM(*P2*,*P3*)**	数据类型	U/UD
功能	计算以 ***P2*** 为起始地址，以 ***P3*** 为数据长度的数组中所有数据的异或和，并将运行结果保存到 ***P1***。		
P1 (I)	运算结果。		
P2 (I)	数据组（数组）的起始地址。		
P3 (I/C)	数据组（数组）的长度。		
例 1	$U100 = XSUM($U200, 80) (UD) /* 计算从$U200 开始的 80 个 32 位正整数的异或加总和，并将运算结果保存到双字$U100。它的另一种表达方式为 $U100 = $U200 ^ $U202 ^ $U204 ^ $U206 ^ $U208 (UD)... ...。 */		
例 2	$U100 =1001B $U101 =1100B $U102 =0110B $U120 = XSUM($U100,3) /* $U120=0011B。*/		

I: 内部变量; C: 常数

SWAP →字节交换

指令格式	**SWAP*(P1,P2)***	数据类型	U
功能	以 ***P1*** 为起始地址，以 ***P2*** 为数据长度的整组数组数据中，每个字的高字节与低字节进行交换。		
P1 (I)	数据组（数组）的起始地址。		
P2 (I)	数据组（数组）的长度。		
例 1	SWAP($U200, 16)		
例 2	$U120=1111111100000000B $U121=1000000100000000B SWAP($U120, 2) /* $U120 的值等于 0000000011111111B，$U121 的值等于 0000000010000001B。*/		

I: 内部变量; C: 常数

14.4.6. 数据转换

BCD →二进制代码转换为十进制BCD码

指令格式	***P1* = BCD(*P2*)**	数据类型	U/UD
功能	将二进制数据 ***P2*** 转换成 BCD 码，并将转换结果保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2 (I/E/C)	被转换的二进制数。		
例 1	$U100 = BCD(0x1234) (U) /* $U100 的值等于 1234。*/		

I: 内部变量; E: 外部变量; C: 常数

BIN →十进制BCD码转换为二进制代码

指令格式	***P1* = BIN(*P2*)**	数据类型	U/UD
功能	将 BCD 码 ***P2*** 转换成二进制数据，并将转换结果保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2 (I/E/C)	被转换的 BCD 码。		
例 1	$U100 = BIN(1234) (U) /* $U100 的值等于 0x1234。*/		

I: 内部变量; E: 外部变量; C: 常数

DW →字传送至双字

指令格式	***P1*** **= DW(*P2*)**	数据类型	U/S
功能	将 16 位整数 ***P2*** 转换为 32 位整数，并将转换结果保存到 ***P1***，高字节补零。		
P1 (I/E)	运算结果。		
P2 (I/E/C)	被转换的 16 位整数。		
例 1	$U100 = DW(12345) (S) /* $U100 的值等于 12345，而$U101 的值等于 0。 */		
例 2	$U200 = DW(-12345) (S) /* $U200 的值等于-12345，而 $U201 的值等于 0xFFFF。 */		

I: 内部变量; E: 外部变量; C: 常数

W →双字取低字

指令格式	***P1*** **= W(*P2*)**	数据类型	UD/SD
功能	将 32 位 ***P2*** 转换成 16 位整数，并将转换结果保存到 ***P1***。运行可能会产生溢出。		
P1 (I/E)	运算结果。		
P2 (I/E/C)	被转换的 32 位整数。		
例 1	$U100 = W(0x12345678) (UD) /* $U100 的值等于 0x5678。 */		
例 2	$U200 = W(-12345) (SD) /* $U200 的值等于 -12345。 */		

I: 内部变量; E: 外部变量; C: 常数

B2W →高低字节转换到连续字存放

指令格式	***P1*** **= B2W(*P2*,P3)**	数据类型	U
功能	以 ***P2*** 为起始地址，***P3*** 为数据长度的字节数组，转换成 ***P3*** 个数据长度的字数组，并将转换结果保存到以 ***P1*** 为起始地址的数组数据中。所有字的高字节都补 0。		
P1 (I)	运算结果的起始地址。		
P2 (I)	被转换的字节数的起始地址。		
P3 (I/C)	数据组的长度。		
例 1	$U200 = 0x45FA $U201 = 0xEB29 $U100 = B2W($U200, 3) /* 将以$U200 为起始地址的连续 3 个字节转换成 3 个字，并将结果保存到以$U100 为起始地址的连续 3 个字中。$U100 = 0xFA，$U101 = 0x45，$U102 = 0x29。 */		

I: 内部变量; C: 常数

W2B →取字的低字节转换至另一字存放

指令格式	***P1* = W2B(*P2*,P3)**	数据类型	U
功能	将 ***P2*** 为起始地址，以 ***P3*** 为数据长度的字数组，转换成以 ***P3*** 为数据长度的字节，并将转换结果保存到以 ***P1*** 为起始地址的数组数据中。所有被转换的字数据内的高字节都被释放。数据组数组最大不能超过 256 个字。		
P1 (I)	运算结果存放的起始地址。		
P2 (I)	被转换的字数据组（数组）的起始地址。		
P3 (I/C)	数据组（数组）的长度。		
例 1	$U200 = 0x45FA $U201 = 0xEB29 $U202 = 0xC781 $U100 = W2B($U200, 3) /*将以$U200 为起始地址的连续 3 个字转换成 3 个字节(Byte)，并将结果保存到以$U100 为起始地址的连续 3 个字节中。$U100 的值等于 0x29FA，$U101 的低字节等于 0x81，$U101 的高字节等于 0x00。*/		

I: 内部变量; C: 常数

A2X →ASCII码转换成 16 位正整数

指令格式	***P1* = A2X(*P2*)**	数据类型	U
功能	将 4 个以 ***P2*** 为起始地址的 ASCII 文字转换为一个二进制的数，并将转换结果保存到 ***P1***。		
P1 (I)	转换结果。		
P2 (I)	被转换参数的起始地址。***P2*** 的 ASCII 文字需为 0-F 方为有效。		
例 1	$U20 = 49 // '1' $U21 = 50 // '2' $U22 = 69 // 'E' $U23 = 70 // 'F' $U100 = A2X($U20) /* $U100 的值等于 0x12EF。 */		

I: 内部变量

X2A →16 位正整数转换成 4 个ASCII

指令格式	***P1* = X2A(*P2*)**	数据类型	U
功能	将一个 16 位整数 ***P2*** 转换成 4 个连续的 ASCII 文字码，并将转换结果保存到 ***P1*** 为起始地址的数组中，高字节补零。		
P1 (I)	运算结果的起始字地址。		
P2 (I/C)	被转换的 16 位整数。		
例 1	$U10 = X2A(0x34AB) /* 从$U10 开始数据依次为：51('3'), 52('4'), 65('A'), 66('B')。 */		

I: 内部变量; C: 常数

W2F →字转换成浮点数

指令格式	***P1* = W2F(*P2*)**	数据类型	U/S
功能	将 16 位整数 ***P2*** 转换成浮点数，并将转换结果保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2 (I/E/C)	被转换的 16 位整数。		
例 1	$U200 = W2F($U10) (S)		

I: 内部变量; E: 外部变量; C: 常数

D2F →双字转换成浮点数

指令格式	***P1* = D2F(*P2*)**	数据类型	UD/SD
功能	将 32 位整数 ***P2*** 转换成浮点数，并将转换结果保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2 (I/E/C)	被转换的 32 位整数。		
例 1	$U200 = D2F($U10) (SD)		

I: 内部变量; E: 外部变量; C: 常数

F2W →浮点数转换成字

指令格式	***P1* = F2W(*P2*)**	数据类型	F
功能	将浮点数 ***P2*** 转换成 16 位整数，并将转换结果保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2 (I/E/C)	被转换的浮点数。		
例 1	$U200 = F2W($U10) (F)		

I: 内部变量; E: 外部变量; C: 常数

F2D →浮点数转换成双字

指令格式	***P1* = F2D(*P2*)**	数据类型	F
功能	将浮点数 ***P2*** 转换成 32 位整数，并将转换结果保存到 ***P1***。		
P1 (I/E)	运算结果。		
P2 (I/E/C)	被转换的浮点数。		
例 1	$U200 = F2D($U10) (F)		

I: 内部变量; E: 外部变量; C: 常数

EXTRACT_BIT →提取位状态

指令格式	***P1* = EXTRACT_BIT(*P2*,P3)**	数据类型	U/UD
功能	从 ***P2*** 中提取第 ***P3*** 位的状态，并将运算结果保存到 ***P1***。		
P1 (I)	运算结果。		
P2 (I)	提取位的所在字地址或双字地址。		
P3 (I/C)	提取的位地址，0-31 可以接受，设超过 32 等于回到位地址 0 开始算。		
例 1	$U2.0 = EXTRACT_BIT($U10, 31) (UD) /* 从双字$U10 中提取第 31 位的状态，并将运算结果保存到$U2.0。 */		

I: 内部变量; C: 常数

14.4.7. 条件操作

IF == →等于时执行

指令格式	**IF *P2* == *P3***	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 等于 ***P3*** 时，执行 IF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

IF != →不等于时执行

指令格式	**IF *P2* != *P3***	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 不等于 ***P3*** 时，执行 IF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

IF > →大于时执行

指令格式	**IF *P2* > *P3***	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 大于 ***P3*** 时，执行 IF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

IF >= →大于或等于时执行

指令格式	**IF *P2* >= *P3***	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 大于或等于 ***P3*** 时，执行 IF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

IF < →小于时执行

指令格式	**IF *P2 < P3***	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 小于 ***P3*** 时，执行 IF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

IF <= →小于或等于时执行

指令格式	**IF *P2 <= P3***	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 小于或等于 ***P3*** 时，执行 IF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

IF & →与条件成立时执行

指令格式	**IF *P2 & P3***	数据类型	U/UD
功能	当 ***P2*** 和 ***P3*** 相对位做「与」（AND）逻辑运算结果为非 0 时，执行 IF 后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

IF !& →非与条件成立时执行

指令格式	**IF *!(P2 & P3)***	数据类型	U/UD
功能	当 ***P2*** 和 ***P3*** 相对位做「与」（AND）逻辑运算结果为 0 时，执行 IF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

IF <bit> →条件成立时执行

指令格式	**IF *P2***	数据类型	B
功能	当判断条件 ***P2*** 成立(1/On)时，执行 IF 语句后的命令。		
P2 (I/E/CE)	判断条件。		

I: 内部变量; E: 外部变量; CE: 比较表达式

IF ! <bit> →条件不成立时执行

指令格式	**IF *!P2***	数据类型	B
功能	当判断条件 ***P2*** 不成立(0/Off)时，执行 IF 语句后的命令。		
P2 (I/E/CE)	判断条件。		

I: 内部变量; E: 外部变量; CE: 比较表达式

ELIF == →等于时执行

指令格式	ELIF *P2 == P3*	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 等于 ***P3*** 时，执行 ELIF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

ELIF != →不等于时执行

指令格式	ELIF *P2 != P3*	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 不等于 ***P3*** 时，执行 ELIF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

ELIF > →大于时执行

指令格式	ELIF *P2 > P3*	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 大于 ***P3*** 时，执行 ELIF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

ELIF >= →大于或等于时执行

指令格式	ELIF *P2 >= P3*	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 大于或等于 ***P3*** 时，执行 ELIF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

ELIF < →小于时执行

指令格式	ELIF *P2 < P3*	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 小于 ***P3*** 时，执行 ELIF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

ELIF <= →小于或等于时执行

指令格式	ELIF *P2 <= P3*	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 小于或等于 ***P3*** 时，执行 ELIF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

ELIF & →与条件成立时执行

指令格式	ELIF *P2 & P3*	数据类型	U/UD
功能	当 ***P2*** 和 ***P3*** 相对位做「与」（AND）逻辑运算结果为非 0 时，执行 ELIF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

ELIF !& →非与条件成立时执行

指令格式	ELIF *!(P2 & P3)*	数据类型	U/UD
功能	当 ***P2*** 和 ***P3*** 相对位做「与」（AND）逻辑运算结果为 0 时，执行 ELIF 语句后的命令。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

ELIF <bit> →条件成立时执行

指令格式	ELIF *P2*	数据类型	B
功能	当判断条件 ***P2*** 成立(1/On)时，执行 ELIF 语句后的命令。		
P2 (I/E/CE)	判断条件。		

I: 内部变量; E: 外部变量; CE: 比较表达式

ELIF ! <bit> →条件不成立时执行

指令格式	ELIF *!P2*	数据类型	B
功能	当判断条件 ***P2*** 不成立(0/Off)时，执行 ELIF 语句后的命令。		
P2 (I/E/CE)	判断条件。		

I: 内部变量; E: 外部变量; CE: 比较表达式

ELSE →前条件不成立时执行

指令格式	ELSE
功能	这个命令所指定的命令语句，要在其前面所有 IF 和 ELIF 条件都不成立的情况下才能被执行。这个命令不可单独执行。

ENDIF →条件结束

<table>
<tr><td>指令格式</td><td>ENDIF</td></tr>
<tr><td>功能</td><td>这个命令用来提示一段以 IF、ELIF、或者 ELSE 语句开始命令的结束。这个命令不可单独执行。</td></tr>
<tr><td>例</td><td>IF 命令结构:
<table>
<tr><th>命令与结构</th><th>描述</th></tr>
<tr><td>IF <condition>
…
ENDIF</td><td>当条件成立时，执行 IF 与 ENDIF 之间的命令，否则跳过这段命令。</td></tr>
<tr><td>IF <condition>
…
ELSE
…
ENDIF</td><td>当条件成立时，执行 IF 与 ELSE 之间的命令，否则执行 ELSE 与 ENDIF 之间的命令。</td></tr>
<tr><td>IF <condition_1>
…
ELIF <condition_2>
…
ELIF <condition_3>
.
.
ELIF <condition_N>
…
ENDIF</td><td>当条件 1 成立时，执行 IF 与第 1 个 ELIF 之间的命令，并跳过余下所有的命令，否则检查条件 2。当条件 2 成立时，执行第 1 个 ELIF 与第 2 个 ELIF 之间的命令，并跳过余下所有的命令，否则检查条件 3。依此类推。若所有条件都不成立，不执行这段语句的所有命令。</td></tr>
<tr><td>IF <condition_1>
…
ELIF <condition_2>
…
ELIF <condition_3>
.
.
ELIF <condition_N>
…
ELSE
…
ENDIF</td><td>当条件 1 成立时，执行 IF 与第 1 个 ELIF 之间的命令，并跳过余下所有的命令，否则检查条件 2。当条件 2 成立时，执行第 1 个 ELIF 与第 2 个 ELIF 之间的命令，并跳过余下所有的命令，否则检查条件 3。依此类推。若所有条件都不成立，执行 ELSE 与 ENDIF 之间的命令。</td></tr>
</table>
注意：最多只能嵌套 20 个 IF-命令结构</td></tr>
</table>

14.4.8. 流程控制

JMP →无条件跳转

指令格式	**JMP *P1***
功能	程序无条件跳转到卷标记号 ***P1*** 的地方。
P1 (CS)	跳转点的程序卷标记号。
例 1	IF $U10 == 0 JMP SKIP /* 跳过命令"$U20 = $U10 / 2" */ ENDIF $U20 = $U10 / 2 SKIP: $U10 = 1

CS: 字符串

<label> →卷标记号字符串

指令格式	***P1:***
功能	***P1*** 是标记程序跳转点位置的卷标记号。这个命令不可单独执行。
P1 (CS)	标记程序跳转点位置的卷标记号，后面必须加上符号":"。
例 1	IF $U10 == 0 JMP SKIP /* 跳过命令"$U20 = $U10 / 2" */ ENDIF $U20 = $U10 / 2 SKIP: $U10 = 1

CS: 字符串

JMP == →等于时跳转

指令格式	**JMP(*P1*,*P2* == *P3*)**	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 等于 ***P3*** 时，程序跳转到卷标记号 ***P1*** 的地方。		
P1 (CS)	跳转点的程序卷标记号。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式; CS: 字符串

JMP != →不等于时跳转

指令格式	**JMP(*P1*,*P2* != *P3*)**	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 不等于 ***P3*** 时，程序跳转到卷标记号 ***P1*** 的地方。		
P1 (CS)	跳转点的程序卷标记号。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式; CS: 字符串

JMP > →大于时跳转

指令格式	JMP(*P1*,*P2* > *P3*)	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 大于 ***P3*** 时，程序跳转到卷标记号 ***P1*** 的地方。		
P1 (CS)	跳转点的程序卷标记号。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式; CS: 字符串

JMP >= →大于或等于时跳转

指令格式	JMP(*P1*,*P2* >= *P3*)	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 大于或等于 ***P3*** 时，程序跳转到卷标记号 ***P1*** 的地方。		
P1 (CS)	跳转点的程序卷标记号。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式; CS: 字符串

JMP < →小于时跳转

指令格式	JMP(*P1*,*P2* < *P3*)	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 小于 ***P3*** 时，程序跳转到卷标记号 ***P1*** 的地方。		
P1 (CS)	跳转点的程序卷标记号。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式; CS: 字符串

JMP <= →小于或等于时跳转

指令格式	JMP(*P1*,*P2* <= *P3*)	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 小于或等于 ***P3*** 时，程序跳转到卷标记号 ***P1*** 的地方。		
P1 (CS)	跳转点的程序卷标记号。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式; CS: 字符串

JMP & →与结果为非 0 时跳转

指令格式	JMP(*P1*,*P2* & *P3*)	数据类型	U/UD
功能	当 ***P2*** 和 ***P3*** 相对位做「与」（AND）逻辑运算结果为非 0 时，程序跳转到卷标记号 ***P1*** 的地方。		
P1 (CS)	跳转点的程序卷标记号。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式; CS: 字符串

JMP !& →与结果为 0 时跳转

指令格式	**JMP(*P1,!(P2 & P3*))**	数据类型	U/UD
功能	当 ***P2*** 和 ***P3*** 相对位做「与」（AND）逻辑运算结果为 0 时，程序跳转到卷标记号 ***P1*** 的地方。		
P1 (CS)	跳转点的程序卷标记号。		
P2,P3 (I/E/C/AE)	操作数。		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式; CS: 字符串

JMP <bit> →条件成立时跳转

指令格式	**JMP(*P1,P2*)**	数据类型	B
功能	当判断条件 ***P2*** 成立(1/On)时，程序跳转到卷标记号 ***P1*** 的地方。		
P1 (CS)	跳转点的程序卷标记号。		
P2,P3 (I/E/CE)	判断条件。		

I: 内部变量; E: 外部变量; AE: 算术表达式; CS: 字符串

JMP ! <bit> →条件不成立时跳转

指令格式	**JMP(*P1,!P2*)**	数据类型	B
功能	当判断条件 ***P2*** 不成立(0/Off)时，程序跳转到卷标记号 ***P1*** 的地方。		
P1 (CS)	跳转点的程序卷标记号。		
P2,P3 (I/E/CE)	判断条件。		

I: 内部变量; E: 外部变量; AE: 算术表达式; CS: 字符串

CALL →调用子宏

指令格式	**CALL *P1***
功能	调用子宏 ***P1***。
P1	调用的子宏名。
例 1	CALL CommonFunction_01 /* 调用名称为 CommonFuncation_01 的宏当作子宏(Sub-macro)。*/

RET →从子宏返回

指令格式	**RET**
功能	从当前子宏回到原先的宏中。这个命令只能用在子宏中。

FOR →循环

指令格式	**FOR *P2***	数据类型	U
功能	执行 ***P1*** 次 FOR 循环的命令。一个 FOR 循环是从 FOR 命令开始，NEXT 命令结束。一共可以嵌入 20 个 FOR ..NEXT 循环。		
P1 (I/C)	FOR 循环的全部次数。		
例 1	FOR 10 $U100 = $U100 + 1 /* 这个命令将总共被执行 10 次 */ FOR 12 $U200 = $U200 + 1 /* 这个命令将总共被执行 120 次 */ NEXT NEXT		

I: 内部变量; C: 常数

NEXT →循环结束

指令格式	**NEXT**
功能	这个命令用了表示一个 FOR 循环的结束。这个命令不可单独执行。
例 1	$U1 = 10 $U2 = 12 FOR $U1 $U100 = $U100 + 1 /* 这个命令将总共被执行 10 次 */ FOR $U2 $U200 = $U200 + 1 /* 这个命令将总共被执行 120 次 */ NEXT NEXT

STOP →立即停止

指令格式	**STOP**
功能	立即停止运行当前宏。如果这个宏是在画面循环宏，那么它将会立即停止继续运行当前宏，这个宏只有在其关联的画面被再次重新打开的时后才会从头开始运行。 如果这个宏是在主宏，那么它将会立即停止继续运行当前宏，这个宏只有在触控屏的应用程序被重新启动的时后才会从新从头开始运行。 注意:这个命令不能使用在子宏中。

END →结束

指令格式	**END**
功能	表示宏或者当前循环的结束。它可以放在宏的任何位置，去终止当前宏的运行。如果这个宏是一个循环宏(例如主宏或是画面循环宏)，它可以终止当前的循环程序，并从宏的第一个指令重新开始新的循环。 这个命令不能被使用在被调用的子宏中。

14.4.9. 定时器操作

SET_T →启动定时器

<table>
<tr><td>指令格式</td><td colspan="2">SET_T(P1,P2)</td><td>数据类型</td><td>U</td></tr>
<tr><td>功能</td><td colspan="4">根据计时控制区 P2 来启动定时器 P1。</td></tr>
<tr><td>P1 (C)</td><td colspan="4">定时器编号。总共有 8 个可用的定时器，它们的编号为 0～7。</td></tr>
<tr><td>P2 (I)</td><td colspan="4">定时器所使用的计时控制区所占用的内存(字节数组)的起始地址。计时控制区的结构如下表所示：
<table>
<tr><th>地址</th><th>数据项</th><th>描述</th></tr>
<tr><td>0</td><td>运行模式</td><td>0：一次计时； 1：循环计时，标志位 0、1 交替</td></tr>
<tr><td>1</td><td>当前值</td><td>计时器当前值，每 100ms 加 1。</td></tr>
<tr><td>2</td><td>设定值</td><td>当计时器当前值等于设定值时，计时器将根据运行模式完成下列操作：
1) 当运行模式为一次计时(0)，将计时器标志位设为 1，当前值复位为 0，并停止计时。
2) 当运行模式为循环计时(1)，触发标志位，当前值复位为 0，并重新开始计时。</td></tr>
<tr><td>3</td><td>标志位</td><td>当当前值等于设定值时，标志位被设置为 0 或者 1。</td></tr>
</table>
定时器的运行需要单独使用定时器控制区，所以请不要使用控制区中的任何字作为他用，否则会误动作。一个定时器控制区需要 4 个字的空间大小。</td></tr>
<tr><td>例 1</td><td colspan="4">$U100 = 1 /* 运行模式为循环计时。 */
$U101 = 0 /* 初始化现在值为 0。 */
$U102 = 5 /* 计时时间设定为 0.5 秒(5*100 毫秒)。 */
$U103 = 0 /* 初始化标志位 0。*/
SET_T(3, $U100) /* 使用 3 号定时器让$U103.0 产生 1Hz 的方波。*/</td></tr>
</table>

I: 内部变量; C: 常数

STOP_T →停止定时器

指令格式	STOP_T(*P1*)	数据类型	U
功能	停止定时器 ***P1*** 的运行。		
P1 (C)	定时器编号。		
例 1	STOP_T(1) /* 停止 1 号定时器运行。*/		

C: 常数

WAIT_T →等待定时器

指令格式	**WAIT_T(*P1*)**	数据类型	U
功能	等待定时器 ***P1*** 计时完成。在计时完成之前这个宏指令之后的指令将不会执行。		
P1 (C)	定时器编号。		
例 1	$U100 = 0 /* 运行模式为一次计时。 */ $U101 = 0 /* 初始化现在值为 0。 */ $U102 = 5 /* 计时时间设定为 0.5 秒(5*100 毫秒)。*/ $U103 = 0 /* 初始化标志位 0。*/ SET_T(7, $U100) /* 启动 7 号定时器进行一个 0.5 秒的计时。*/ WAIT_T(7) /* 等待 0.5 秒 */		

C: 常数

14.4.10. 键盘操作

KB_MCR →接收或忽略当前键盘输入

指令格式	**KB_MCR(*P1*)**	数据类型	U
功能	接收或忽略当前键盘输入的字符/指令。这个命令只能使用在键盘按钮的宏中。当指定键盘按钮被按下时，执行该宏。你可以使用这个键盘按钮宏的指令来选择或忽略这个当前按钮的输入。		
P1 (I/C)	用来决定是否接收当前输入的指示值或该指示值的地址。当这个值等于 0 时，接收当前输入；非 0 则表示将忽略，不接受当前输入。		
例 1	KB_MCR(1) /* 忽略当前输入。*/		

I: 内部变量; C: 常数

KPD_TEXT →初始化键盘

指令格式	**KPD_TEXT(*P1*)**	数据类型	U
功能	初始化键盘显示器，并将字符串 ***P1*** 存入缓存。		
P1 (I)	包含一个零终止符 ASCII 码字符串的记忆区(或字节数组)，用来初始化键盘显示器和缓存。		
例 1	$U100 = "initial text" KPD_TEXT($U100) /* 初始化键盘显示器，并将字符串"initial text" 存入缓存。*/		

I: 内部变量

14.4.11. 配方操作

RB2ROM →配方资料保存到闪存

指令格式	***P1*** **= RB2ROM(*P2*)**	数据类型	U
功能	将 ***P2*** 配方区的数据保存到闪存，并将完成代码保存到 ***P1***。		
P1 (I)	运行后产生的完成代码。如果这个代码等于 0，表示操作成功；否则操作失败。		
P2 (I/C)	要被保存配方区的编号。配方属性中的“需要在闪存空间中保存数据备份”选项必须要选中。		
例 1	$U10 = RB2ROM(3) /* 将 3 号配方区的资料保存到闪存。*/		

I: 内部变量; C: 常数

ROM2RB →闪存中的数据读到配方区

指令格式	***P1*** **= ROM2RB(*P2*)**	数据类型	U
功能	从闪存中读取数据到 ***P2*** 配方区，并将完成代码保存到 ***P1***。		
P1 (I)	运行后产生的完成代码。如果这个代码等于 0，表示操作成功；否则操作失败。		
P2 (I/C)	恢复闪存数据的配方区的编号。配方属性中的“需要在闪存空间中保存数据备份”选项必须要选中。		
例 1	$U10 = ROM2RB(3) /* 将闪存内的资料回存到 3 号配方区的资料。*/		

I: 内部变量; C: 常数

REF_RCP_OBJ →更新配方对象

指令格式	**REF_RCP_OBJ(*P1*)**	数据类型	U
功能	刷新使用指定配方区的配方对象。被刷新的配方对象包括配方选择器和配方表。你可以在宏程序改变了配方区数据后， 使用该指令刷新相关配方对象的显示。		
P1 (I/C)	配方区号或是存储配方区号的地址。		
例 1	REF_RCP_OBJ(3) /* 刷新使用 3 号配方区的配方对象。*/		

I: 内部变量; C: 常数

14.4.12. 通讯操作

EN_LINK →打开或断开通讯联机

指令格式	EN_LINK(*P1*,*P2*,*P3*)	数据类型	U
功能	当 ***P3*** 等于 1 时，打开通讯联机编号 ***P1*** 或者是联机编号 ***P1*** 的次联机站号 ***P2***。当 ***P3*** 等于 0 时，断开指定的通讯联机或者次联机。		
P1 (I/C)	被打开或断开的通讯联机编号。		
P2 (I/C)	被打开或断开的次联机站号。如果指定的通讯联机没有次联机，这个参数将被忽略。如果指定的通讯联机有次联机，而你想让它经由画面的次连接对象来控制其自动打开或断开，请将该参数设为 0。		
P3 (I/C)	当该参数为 1 时，打开指定的通讯联机或次联机；当该参数为 0 时，断开指定的通讯联机或次联机。		
例 1	ENABLE_LINK(1, 20, 0) /* 断开联机编号 1 的次联机中站号 20 号的设备的联机。*/		

I: 内部变量; C: 常数

LINK_STS →通讯联机状态

指令格式	***P1*** = LINK_STS(***P2***,***P3***)	数据类型	U
功能	读取通讯联机编号 ***P2*** 或者通讯联机编号 ***P2*** 的站号 ***P3*** 的次联机的状态，并将读取结果保存到 ***P1***。		
P1 (I/C)	所获取的指定联机或次联机的状态值。所有状态以一个 16 位的数值来表示其对应的意义如下表所示：		
P2 (I/C)	被打开或断开的通讯联机编号。		
P3 (I/C)	被打开或断开的通讯次联机站号。如果指定的通讯联机没有次联机，这个参数将被忽略。		
例 1	$U100 = LINK_STS(2, 0) /* 获取联机编号 2 的状态，并将结果保存到$U100。*/		
例 2	$U12 = LINK_STS(1, 128) /* 获取联机编号 1 的次联机站号 128 的设备的联机状态，并将结果保存到 $U12。*/		

状态值	含义	状态值	含义
0	正常	14	设备繁忙
1	溢出错误	15	未知错误
2	中断错误	16	连接失效
3	奇偶校验错误	17	初始化失败
4	组帧错误	18	发送数据失败
5	无响应	19	读取数据失败
6	未知的响应	20	打开连接失败
7	超时	21	连接没有准备好
8	失效的 CTS	22	无效的次连接
9	和校验错误	23	无效的 COM 口
10	命令被拒绝	24	出错
11	无效的地址	255	无法确定状态
12	无效的范围	65535	获取状态失败
13	无效的请求		

I: 内部变量; C: 常数

14.4.13. 系统服务

GET_RTC →读取万年历时间

指令格式	GET_RTC(*P1*)	数据类型	U
功能	读取当前触控屏万年历时间(RTC)，并将所获取的时间保存到 ***P1***。		
P1 (I)	将所读取的 RTC 万年历时间数据存入此记忆区起始地址。一个 RTC 数据块需要 8 个字的空间大小。RTC 数据块的结构如下表所示： **数据项** / **数据类型/长度** / **地址范围** 秒 / 16 位无符号整数 / 0 分 / 16 位无符号整数 / 1 时 / 16 位无符号整数 / 2 RTC 调整 / 16 位有符号整数 / 3 日 / 16 位无符号整数 / 4 月 / 16 位无符号整数 / 5 年 / 16 位无符号整数 / 6 星期 / 16 位无符号整数 / 7 秒: 0~59; 分: 0~59; 时: 0~23; RTC 精度调整: -63~63; 日: 1~31; 月: 1~12; 年: 0(2000)~99(2099); 星期: 0(星期日)~6(星期六)		
例 1	GET_RTC($U100) /* 读取当前万年历时间。秒被保存到$U100，而星期被保存到$U107。*/		

I: 内部变量

SET_RTC →设置万年历时间

指令格式	SET_RTC(*P1*)	数据类型	U
功能	将 ***P1*** 内的时间数据设置为触控屏的万年历时间(RTC)。		
P1 (I)	用来设置RTC万年历时间数据块的记忆区起始地址。RTC数据块的结构详见 GET_RTC 说明。		
例 1	$U100 = 0 // 秒 $U101 = 30 // 分 $U102 = 8 // 时 $U103 = 0 // 精度调整 $U104 = 1 // 日 $U105 = 7 // 七月 $U106 = 10 // 年 2010 $U107 = 4 // 星期四 SET_RTC($U100) /* 将 2010 年 7 月 1 日 8:30:00 星期四设置为当前实时时间。*/		

I: 内部变量

SYS →系统服务

指令格式	**SYS(*P1*,*P2*,*P3*)**	数据类型	U
功能	根据参数 ***P2*** 和 ***P3*** 请求系统服务 ***P1***。这个指令是留给系统使用的。		
P1 (I)	系统服务代码。		
P2,P3 (I/C)	系统服务参数。		

I: 内部变量; C: 常数

14.4.14. 画面操作

OPEN_WS →打开窗口画面

指令格式	**OPEN_WS *P1***	数据类型	U
功能	打开由 ***P1*** 指定的窗口画面。本指令只能用在画面的循环宏中。		
P1 (I/C)	需要被打开的窗口画面编号。 如果指定的画面是普通画面或是菜单画面， 该指令不会打开指定的画面。只有在被打开的窗口画面关闭后，本指令之后的所有指令才会被执行。如果画面的循环宏正在等待关闭由本指令打开的窗口画面，这时该画面不能以任何方式关闭或切换。		

I: 内部变量; C: 常数

CLOSE_WS →关闭窗口画面

指令格式	**CLOSE_WS**
功能	关闭窗口画面并且中止正在执行的循环宏。本指令只能用在窗口画面的循环宏中。

14.4.15. 文件操作

FILE_IO →保存文件

<table>
<tr><td>指令格式</td><td colspan="2">P1 = FILE_IO(P2,P3)</td><td>数据类型</td><td>U</td></tr>
<tr><td>功能</td><td colspan="4">根据 P2 和 P3 所指定的形式以默认文件名保存文件，并将完成代码保存到 P1。</td></tr>
<tr><td>P1 (I)</td><td colspan="4">运行后产生的完成代码。如果这个代码等于 0，表示操作成功；不等于 0 表示操作失败。</td></tr>
<tr><td>P2,P3 (I/C)</td><td colspan="4">P2 指定操作类型。P3 指定数据源的编号。具体 P2 和 P3 设置如下表所示：
<table>
<tr><th>文件操作</th><th>P2</th><th>P3</th></tr>
<tr><td>保存历史数据 (.txt)</td><td>1</td><td>数据收集器编号 (0~15)</td></tr>
<tr><td>保存历史警报 (.txt)</td><td>2</td><td>0</td></tr>
<tr><td>保存警报计数(.txt)</td><td>3</td><td>0</td></tr>
<tr><td>保存配方数据 (.txt)</td><td>4</td><td>配方编号 (0~15)</td></tr>
<tr><td>保存配方数据 (.prd)</td><td>5</td><td>配方编号 (0~15)</td></tr>
<tr><td>保存画面为图片 (256-color .bmp)</td><td>6</td><td>画面编号 (1~7999)</td></tr>
<tr><td>保存画面为图片 (64K-color .bmp)</td><td>7</td><td>画面编号(1~7999)</td></tr>
<tr><td>保存操作记录 (.txt)</td><td>9</td><td>0</td></tr>
<tr><td>保存历史数据 (.ldf)</td><td>10</td><td>数据收集器编号 (0~15)</td></tr>
<tr><td>获取摄像头图像 (.bmp)</td><td>12</td><td>USB 摄像头编号 (0~3)</td></tr>
<tr><td>获取摄像头图像(.jpg)</td><td>13</td><td>USB 摄像头编号 (0~3)</td></tr>
</table>
默认文件名：
<table>
<tr><th>文件操作</th><th>文件名格式</th><th>说明</th></tr>
<tr><td>保存历史数据 (.txt)</td><td>DL<ID>_<Date>_<Time>.txt</td><td><ID>: 数据收集器编号</td></tr>
<tr><td>保存历史警报 (.txt)</td><td>AL_<Date>_<Time>.txt</td><td></td></tr>
<tr><td>保存警报计数(.txt)</td><td>AC_<Date>_<Time>.txt</td><td></td></tr>
<tr><td>保存配方数据 (.txt)</td><td>RB<ID>.txt</td><td><ID>: 配方编号</td></tr>
<tr><td>保存配方数据 (.prd)</td><td>RB<ID>.prd</td><td><ID>: 配方编号</td></tr>
<tr><td>保存画面为图片 (256-color .bmp)</td><td>S<ID>_<Date>_<Time>.bmp</td><td><ID>: 画面编号</td></tr>
<tr><td>保存画面为图片 (64K-color .bmp)</td><td>S<ID>_<Date>_<Time>.bmp</td><td><ID>: 画面编号</td></tr>
<tr><td>保存操作记录 (.txt)</td><td>OL_<Date>_<Time>.txt</td><td></td></tr>
<tr><td>保存历史数据 (.ldf)</td><td>DL<ID>_<Date>_<Time>.ldf</td><td><ID>: 数据收集器编号</td></tr>
<tr><td>获取摄像头图像 (.bmp)</td><td>CAM<ID>_<Date>_<Time>.bmp</td><td>USB 摄像头编号</td></tr>
<tr><td>获取摄像头图像(.jpg)</td><td>CAM<ID>_<Date>_<Time>.jpg</td><td>USB 摄像头编号</td></tr>
</table>
注意:
<Date>: 数据保存的日期。 <Time>: 数据保存的时间。
你可以在 General Setup 对话框内的“客制页面”自定义 <Date> 和 <Time>的格式。</td></tr>
</table>

I: 内部变量; C: 常数

FILE_IO_N →另存为新文件

<table>
<tr><td>指令格式</td><td colspan="2">P1 = FILE_IO_N(P2,P3,P4)</td><td>数据类型</td><td>U</td></tr>
<tr><td>功能</td><td colspan="4">根据 P2 和 P3 所指定的形式保存文件，并以 P4 为文件名。执行后并将完成代码保存到 P1。</td></tr>
<tr><td>P1 (I)</td><td colspan="4">运行后产生的完成代码。如果这个代码等于 0，表示操作成功；不等于 0 表示操作失败。</td></tr>
<tr><td>P2,P3 (I/C)</td><td colspan="4">P2 指定操作文件类型。P3 指定数据源的编号。具体 P2 和 P3 设置如下表所示：
<table>
<tr><th>文件操作</th><th>P2</th><th>P3</th></tr>
<tr><td>保存历史数据 (.txt)</td><td>31</td><td>数据收集器编号 (0~15)</td></tr>
<tr><td>保存历史警报 (.txt)</td><td>32</td><td>0</td></tr>
<tr><td>保存警报计数(.txt)</td><td>33</td><td>0</td></tr>
<tr><td>保存配方数据 (.txt)</td><td>34</td><td>配方编号 (0~15)</td></tr>
<tr><td>保存配方数据 (.prd)</td><td>35</td><td>配方编号 (0~15)</td></tr>
<tr><td>保存画面为图片 (256-color .bmp)</td><td>36</td><td>画面编号 (1~7999)</td></tr>
<tr><td>保存画面为图片 (64K-color .bmp)</td><td>37</td><td>画面编号(1~7999)</td></tr>
<tr><td>保存操作记录 (.txt)</td><td>39</td><td>0</td></tr>
<tr><td>保存历史数据 (.ldf)</td><td>40</td><td>数据收集器编号 (0~15)</td></tr>
<tr><td>获取摄像头图像 (.bmp)</td><td>42</td><td>USB 摄像头编号 (0~3)</td></tr>
<tr><td>获取摄像头图像(.jpg)</td><td>43</td><td>USB 摄像头编号 (0~3)</td></tr>
</table></td></tr>
<tr><td>P4 (I)</td><td colspan="4">用来存储自定义的文件名或完整的路径的字节数组。文件名及路径只能是由 ASCII 码组成的 Windows 有效路径。这个字符串必须包含零终止符，每个字符占用一个字节。它的最大长度为 127 个字节。所有文件的路径必须已经存在(例如 U 盘 SD 卡)，否则操作将会失败。</td></tr>
</table>

I: 内部变量; C: 常数

MKDIR →创建一个目录

指令格式	***P1* = MKDIR(*P2*)**
功能	根据 ***P2*** 所指定的名称创建一个目录。并将完成代码保存到 ***P1***。
P1 (I)	运行后产生的完成代码。如果这个代码等于 0，表示操作成功；不等于 0 表示操作失败。
P2 (I)	这个字节数组包含了新目录的名称。 这个名称必须是一个有效的目录名，包不包含路径均可，它只能由 ASCII 码字符组成。

I: 内部变量

OPEN_FILE →新建或打开文件

<table>
<tr><td>指令格式</td><td colspan="2">P1 = OPEN_FILE(P2,P3)</td><td>数据类型</td><td>U</td></tr>
<tr><td>功能</td><td colspan="4">新建或打开一个文件。</td></tr>
<tr><td>P1 (I)</td><td colspan="4">用来存储接收到的文件信息块的记忆区起始地址。
文件信息块的数据结构如下表所示：
<table>
<tr><th>数据项</th><th>数据类型</th><th>地址范围</th></tr>
<tr><td>文件指挥代码</td><td>32 位无符号整数</td><td>0 和 1</td></tr>
<tr><td>文件大小</td><td>32 位无符号整数</td><td>2 和 3</td></tr>
<tr><td>文件名</td><td>81 个元素的字节数组</td><td>4 到 44</td></tr>
</table>
当执行失败时，文件指挥代码等于零。当执行成功时此代码由系统自动产生一组非 0 的 32 位无符号整数。
一个新创建的文件大小等于零。
文件名是一个包含零终止符的字符串。它的最大允许范围是 80 字。它在文件被成功打开时被赋值。
一个文件信息块一共需要 45 个字的内存空间。</td></tr>
<tr><td>P2 (I)</td><td colspan="4">这个字节数组包含要被打开的文件名，包不包含路径均可，它只能由 ASCII 码字符组成，并包含一个零终止符。</td></tr>
<tr><td>P3 (I/C)</td><td colspan="4">指定文件打开方式
<table>
<tr><th>打开方式</th><th>值</th></tr>
<tr><td>读</td><td>0</td></tr>
<tr><td>写</td><td>1</td></tr>
<tr><td>添加</td><td>3</td></tr>
<tr><td>读 CSV 文件</td><td>5</td></tr>
</table></td></tr>
<tr><td>例 1</td><td colspan="4">$U10 = “test.txt”
$U100 = OPEN_FILE($U10, 0) /* 以只读方式打开文件 “test.txt” 。其中双字$100 存储文件句柄，双字$102 存储文件大小，字节数组$104~$109 存储文件名。*/</td></tr>
</table>

I: 内部变量; C: 常数

READ_FILE →读文件的数据

指令格式	***P1* = READ_FILE(*P2,P3,P4*)**	数据类型	U
功能	从文件 ***P2*** 读取 ***P4*** 个字节到内存 ***P3***，并将运行结果保存到 ***P1***。		
P1 (I)	实际读取到的字节数。如果操作失败，这个数字是 65535(0xFFFF) 。		
P2 (I)	被读文件的文件句柄。		
P3 (I)	存放从文件读取到的数据到此一内存首地址。		
P4 (I/C)	要读取文件的字节数。你可以设定的最大值是 32767(0x7FFF)。		
例 1	$U200 = READ_FILE($U100,$U150,20) /* 从$U100 所指定文件句柄的文件中读取 20 个字节，并将所读取的数据保存到以$U150 为起始地址的内存中。 */		

I: 内部变量; C: 常数

WRITE_FILE →将数据写入文件

指令格式	***P1* = WRITE_FILE(*P2,P3,P4*)**	数据类型	U
功能	将数据地址 ***P3*** 开始一共 ***P4*** 个字节写入文件 ***P2*** 中，并将完成代码保存到 ***P1***。		
P1 (I)	运行后产生的完成代码。如果这个代码等于 0，表示操作成功；不等于 0 表示操作失败。		
P2 (I)	文件的文件句柄。		
P3 (I)	源数据存储器首地址，源数据是指要被写入文件的数据(或字节数组)。		
P4 (I/C)	要被写入文件的字节数。		
例 1	$U200=WRITE_FILE($U100,$U150,30) /* 将数据记忆区$U150 开始的 30 个字节的数据写入由$U100 所指定的文件中。*/		

I: 内部变量; C: 常数

CLOSE_FILE →关闭已打开的文件

指令格式	***P1* = CLOSE_FILE(*P2,P3*)**	数据类型	U
功能	关闭一个已打开的文件 ***P2***，并将完成代码保存到 ***P1***。		
P1 (I)	运行后完成产生的代码。如果这个代码等于 0，表示操作成功；不等于 0 表示操作失败。		
P2 (I)	被关闭文件的文件名。		
例 1	$U200=CLOSE_FILE($U100) /* 关闭由$U100 指定文件名的文件。*/		

I: 内部变量

DELETE_FILE →删除文件

指令格式	***P1* = DELETE_FILE(*P2*)**	数据类型	U
功能	删除一个文件名为 ***P2*** 的文件，并将完成代码保存到 ***P1***。		
P1 (I)	运行后产生的完成代码。如果这个代码等于 0，表示操作成功；不等于 0 表示操作失败。		
P2 (I)	要被删除的文件名或是含完整路径的字节文字数据（数组）。这个字节（数组）必须是 ASCII 字符串且包含零终止符。		
例 1	$U10 = “test.txt” $U200 = DELETE_FILE($U10) /* 删除文件 “test.txt” */		

I: 内部变量

RENAME_FILE →重新命名文件

指令格式	***P1* = RENAME_FILE(*P2,P3*)**	数据类型	U
功能	使用新名称 ***P3*** 来重新命名文件 ***P2***，并将完成代码保存到 ***P1***。		
P1 (I)	运行后产生的完成代码。如果这个代码等于 0，表示操作成功；不等于 0 表示操作失败。		
P2 (I)	这个字节数组包含了要被重新命名的原文件名，包不包含路径均可，它只能由 ASCII 码字符组成，并包含一个零终止符。		
P3 (I)	这个字节数组包含了新文件名。它只能由 ASCII 字符组成，并包含一个零终止符。		
例 1	$U10 = “test.txt” $U50 = “new.txt” $U200 = RENAME_FILE($U10, $U50) /* 将文件 “test.txt” 重新命名为 “new.txt”。 */		

I: 内部变量

GET_VOL_INFO →读取的标签信息

<table>
<tr><td>指令格式</td><td colspan="2">P1 = GET_VOL_INFO(P2,P3)</td><td>数据类型</td><td>U</td></tr>
<tr><td>功能</td><td colspan="4">读取标签 P2 的信息，并将所获取的标签信息保存到 P3。而完成代码则保存到 P1。</td></tr>
<tr><td>P1 (I)</td><td colspan="4">运行后产生的完成代码。如果这个代码等于 0，表示操作成功；不等于 0 表示操作失败。</td></tr>
<tr><td>P2 (I/C)</td><td colspan="4">驱动器编号
<table>
<tr><th>编号</th><th>驱动器</th></tr>
<tr><td>0</td><td>当前驱动器</td></tr>
<tr><td>3</td><td>驱动器 C</td></tr>
<tr><td>4</td><td>驱动器 D</td></tr>
<tr><td>5</td><td>驱动器 E</td></tr>
</table></td></tr>
<tr><td>P3 (I)</td><td colspan="4">执行读取标签信息指令后将所得到的标签区块储存到这个起始地址。标签区块的结构如下所示：
<table>
<tr><th>数据项</th><th>数据类型/大小</th><th>地址范围</th></tr>
<tr><td>标签名</td><td>32 个元素的字节数组</td><td>0 到 15</td></tr>
<tr><td>标签大小</td><td>32 位无符号整数</td><td>16 和 17</td></tr>
<tr><td>可用空间</td><td>32 位无符号整数</td><td>18 和 19</td></tr>
<tr><td>驱动器编号</td><td>16 位无符号整数</td><td>20</td></tr>
</table>
标签名是一个包含零终止符的字符串。它的最大允许范围是 31 个字符。
标签大小和可用空间大小都是 2 个字。
一个卷标区块需要 21 个字的内存空间大小。</td></tr>
<tr><td>例 1</td><td colspan="4">$U100 = GET_VOL_INFO(0, $U0) /* 获取当前卷标信息。标签名储存到$U0~$U15 中；驱动器大小储存到$U16~$U17；驱动器可用空间大小储存到$U18~$U19；当前驱动器编号储存到$U20。 */</td></tr>
</table>

I: 内部变量; C: 常数

READ_CSV →读CSV文件并读取数值

指令格式	***P1* = READ_CSV(*P2,P3,P4*)**	数据类型	S/U/SD/UD/F
功能	读 CSV 文件 ***P2*** 中位置在 ***P3*** 行 ***P4*** 列的单元，并将读取的数值存到 ***P1***。		
P1 (I)	存放读取结果的字位置。替这个命令选取的数据类别必须与要读取字段的数据类别一致，否则可能造成执行失败。若执行失败，如数据类别不对或字段不存在，则都不会写值到 ***P1***. 要知道执行是否失败，须检查字$S522。当字$S522 的值不为 0, 就表示执行失败。		
P2 (I)	被读CSV文件的文件句柄。文件句柄是执行 OPEN_FILE产生的。被读文件必须是一个CSV文件，并且是以"读CSV文件 "方式打开。 分隔字符必须是TAB。		
P3 (I/C)	被读单元的行数。行数从 0 开始算起。		
P4 (I/C)	被读单元的列数。列数从 0 开始算起。		
例 1	U10 = "test.csv" $U100 = OPEN_FILE($U10,5) /*以"读取 CSV 文件"方式打开文件 "test.csv"*/ $U200 = READ_CSV($U100,2,3) (F) /*从行 2 列 3 单元读取一个浮点数并存放结果到字$U200 及 $U201*/		

I: 内部变量; C: 常数

READ_CSV_STR →读CSV文件并读取字符串

指令格式	***P1* = READ_CSV_STR (*P2,P3,P4*)**
功能	读 CSV 文件 ***P2*** 中位置在 ***P3*** 行 ***P4*** 列的单元，并将读取的字符串存到 P1。
P1 (I)	存放读取结果的字节数组位置。这个命令可读取字符串的 最大长度是 128。如果被读字符串长度超过 128，则会造成执行失败。若执行失败，如字符串太长或字段不存在，则都不会写值到 ***P1***. 要知道执行是否失败，须检查字$S522。当字$S522 的值不为 0, 就表示执行失败。
P2 (I)	被读CSV文件的文件句柄。文件句柄是执行 OPEN_FILE产生的。被读文件必须是一个CSV文件，并且是以"读CSV文件 "方式打开。 分隔字符必须是TAB。
P3 (I/C)	被读单元的行数。行数从 0 开始算起。
P4 (I/C)	被读单元的列数。列数从 0 开始算起。
例 1	$U10 = "test.csv" $U100 = OPEN_FILE($U10,5) /*以"读取 CSV 文件"方式打开文件 "test.csv"*/ $U200 = READ_CSV_STR($U100,2,6) /*从行 2 列 6 单元读取一个 字符串并存放结果到启始位置在$U200 的字节数组*/

I: 内部变量; C: 常数

14.4.16. 比较指令

== →等于

指令格式	***P1 = P2 == P3***	数据类型	U/S/UD/SD/F/B
功能	当 ***P2*** 等于 ***P3*** 时，将位 ***P1*** 置 1，否则清 0。		
P1 (I/E)	位(Bit)运算结果。		
P2,P3 (I/E/C/AE)	操作数。		
例 1	$U3.3 = ($U10 + $U20) == 25.75 (F)		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

!= →不等于

指令格式	***P1 = P2 != P3***	数据类型	U/S/UD/SD/F/B
功能	当 ***P2*** 不等于 ***P3*** 时，将位 ***P1*** 置 1，否则清 0。		
P1 (I/E)	位运算结果。		
P2,P3 (I/E/C/AE)	操作数。		
例 1	$U3.3 = ($U10 + $U20) != -700 (S)		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

> →大于

指令格式	***P1 = P2 > P3***	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 大于 ***P3*** 时，将位 ***P1*** 置 1，否则清 0。		
P1 (I/E)	位运算结果。		
P2,P3 (I/E/C/AE)	操作数。		
例 1	$U3.3 = ($U10 + $U20) > $U30 (UD)		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

>= →大于或等于

指令格式	***P1 = P2 >= P3***	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 大于或等于 ***P3*** 时，将位 ***P1*** 置 1，否则清 0。		
P1 (I/E)	位运算结果。		
P2,P3 (I/E/C/AE)	操作数。		
例 1	$U3.3 = ($U10 + $U20) >= 25.75 (F)		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

< →小于

指令格式	***P1 = P2 < P3***	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 小于 ***P3*** 时，将位 ***P1*** 置 1，否则清 0。		
P1 (I/E)	位运算结果。		
P2,P3 (I/E/C/AE)	操作数。		
例 1	$U3.3 = ($U10 + $U20) < 25.75 (F)		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

<= →小于或等于

指令格式	***P1 = P2 <= P3***	数据类型	U/S/UD/SD/F
功能	当 ***P2*** 小于或等于 ***P3*** 时，将位 ***P1*** 置 1，否则清 0。		
P1 (I/E)	位运算结果。		
P2,P3 (I/E/C/AE)	操作数。		
例 1	$U3.3 = ($U10 + $U20) <= 25.75 (F)		

I: 内部变量; E: 外部变量; C: 常数; AE: 算术表达式

14.4.17. 字符串操作

STRCPY →字符串复制

指令格式	**STRCPY(*P1, P2*)**
功能	将 ***P2*** 内的字符串复制到 ***P1***。
P1 (I)	接收 ***P2*** 字符串的字节数组。这个字节数组必须有足够的空间去存放 ASCII 字符串和零终止符。
P2 (I)	数据源，即被复制的包含零终止符的字节文字数据（数组）。
例 1	$U10 = “ABCDE” STRCPY($U20, $U10) 执行 STRCPY 命令后，字节数组$U10 所包含的文字“ABCDE” 转存为 ASCII 文字码到 $U20，具体记忆区分配如下： 字 | 高字节 | 低字节 $U20 | 'B' | 'A' $U21 | 'D' | 'C' $U22 | 0 | 'E'
例 2	$U10 = “12” STRCPY($U20, $U10) 执行 STRCPY 命令后，字节数据组（数组）$U10 所包含的文字“12” 转存为 ASCII 文字码到 $U20，具体记忆区分配如下： 字 | 高字节 | 低字节 $U20 | '2' | '1' $U21 | 未定义 | 0

I: 内部变量

STRCAT →文字符串添加

指令格式	**STRCAT(*P1, P2*)**
功能	将 ***P2*** 内的文字符串添加到 ***P1*** 里。
P1 (I)	字节数组，包含一个零终止符和被添加的 ***P2*** 字符串。 这个字节数组必须有足够的空间去存放字符串和零终止符。
P2 (I)	要添加到 ***P1*** 的字节数组，它是包含了一个零终止符的字符串。
例 1	$U10 = “ABC” $U20 = “12345” STRCAT($U10, $U20) /* 执行命令后，字节数组$U10 所包含的字符串为“ABC12345”。*/
例 2	$U100 = “C:\MyFolder\” $U130 = “Test” $U140 = “.txt“ STRCAT($U100, $U130) STRCAT($U100, $U140) /* 执行命令后，字节数组$U100 所包含的字符串为“C:\MyFolder\Test.txt”。*/

I: 内部变量

STRLEN →计算字符串长度

指令格式	***P1* = STRLEN(*P2*)**
功能	计算 ***P2*** 字符串的文字长度(零终止符不计入)，并将运算结果保存到 ***P1***。
P1 (I)	运行结果。
P2 (I)	包含零终止符字符串的字节数组。
例 1	$U10 = “ABC” $U20 = STRLEN($U10) /* 执行命令后，$U20 的值等于 3。*/

I: 内部变量

STRCMP →比较字符串

指令格式	***P1* = STRCMP(*P2*,*P3*)**
功能	按照字典顺序, 比较字符串 ***P2*** 和 ***P3*** (大小写敏感), 并将比较结果保存到 ***P1***。
P1 (I)	比较结果。 **值** / **描述** 0 / ***P2*** 等于 ***P3*** 1 / ***P2*** 大于 ***P3*** 0xFFFF / ***P2*** 小于 ***P3***
P2,P3 (I)	包含零终止符字符串的字节数组。
例 1	$U10 = “ABC” $U20 = “abc” $U30 = STRCMP($U10, $U20) /* 执行命令后，$U30 的值等于 0xFFFF。*/
例 2	$U10 = “XYZ” $U20 = “ABC” $U30 = STRCMP($U10, $U20) /* 执行命令后，$U30 的值等于 1。*/
例 3	$U10 = “ABC” $U20 = “ABC” $U30 = STRCMP($U10, $U20) /* 执行命令后，$U30 的值等于 0。*/

I: 内部变量

STRICMP →比较字符串(大小写不敏感)

<table>
<tr><td>指令格式</td><td>P1 = STRICMP(P2,P3)</td></tr>
<tr><td>功能</td><td>按照字典顺序, 以不区分字母大小写的方式比较字符串 P2 和 P3 , 并将比较结果保存到 P1。</td></tr>
<tr><td>P1 (I)</td><td>比较结果。
<table>
<tr><th>值</th><th>描述</th></tr>
<tr><td>0</td><td>P2 等于 P3</td></tr>
<tr><td>1</td><td>P2 大于 P3</td></tr>
<tr><td>0xFFFF</td><td>P2 小于 P3</td></tr>
</table></td></tr>
<tr><td>P2,P3 (I)</td><td>包含零终止符字符串的字节数组。</td></tr>
<tr><td>例 1</td><td>$U10 = “ABC”
$U20 = “abc”
$U30 = STRICMP($U10, $U20) /* 执行命令后，$U30 的值等于 0。*/</td></tr>
<tr><td>例 2</td><td>$U10 = “XYZ”
$U20 = “ABC”
$U30 = STRICMP($U10, $U20) /* 执行命令后，$U30 的值等于 1。*/</td></tr>
<tr><td>例 3</td><td>$U10 = “ABC”
$U20 = “XYZ”
$U30 = STRICMP($U10, $U20) /* 执行命令后，$U30 的值等于 0xFFFF。*/</td></tr>
</table>

I: 内部变量

STRNCMP →比较字符串的前n个字符

指令格式	***P1* = STRNCMP(*P2*,*P3*,*P4*)**
功能	按照字典顺序, 比较字符串 ***P2*** 和 ***P3*** 的前 ***P4*** 个字符(大小写敏感)。并将比较结果保存到 ***P1***。
P1 (I)	比较结果。 值 / 描述 0 / ***P2*** 的子串等于 ***P3*** 的子串 1 / ***P2*** 的子串大于 ***P3*** 的子串 0xFFFF / ***P2*** 的子串小于 ***P3*** 的子串 **注意：** 在对 ***P4*** 个字符比较时，遇到任意一个字符串的零终止符，比较结束。如果在比较结束时两字符子串相等，字符子串短的更小。 在 ASCII 码表中，字符代码从 91 到 96 ('[', '\', ']', '^', '_', and '`')的字符比任何字母都小。
P2,P3 (I)	包含零终止符字符串的字节数组。
P4(I/C)	需要比较的字符个数
例 1	$U10 = "XYZ" $U20 = "XYZAB" $U30 = STRNCMP($U10, $U20,4) /* 执行命令后，$U30 的值等于 0xFFFF。*/
例 2	$U10 = "ABZ" $U20 = "ABC" $U30 = STRNCMP($U10, $U20,2) /* 执行命令后，$U30 的值等于 0。*/
例 3	$U10 = "AXC" $U20 = "ABC" $U30 = STRNCMP($U10, $U20,3) /* 执行命令后，$U30 的值等于 1。*/

I: 内部变量; C: 常数

STRCHR →查找字符串中首次出现字符的位置

指令格式	***P1* = STRCHR(*P2*,*P3*)**
功能	查找字符串 ***P2*** 中首次出现字符 ***P3*** 的位置, 并将查找结果保存到 ***P1***。
P1 (I)	查找结果 。如果字符串 ***P2*** 中不包括 ***P3*** 字符,查找结果为 0xFFFF。否则为字符串 ***P2*** 中首次出现字符 ***P3*** 的位置。
P2 (I)	包含零终止符字符串的字节数组。
P3 (I/C)	字符的 ASCII 代码
例 1	$U10 = "The quick brown dog jumps over the lazy fox." $U20 = 0x72 /* 字符'r' 的 ASCII 代码 */ $U30 = STRCHR($U10, $U20) /* 执行命令后，$U30 的值等于 11。*/

I: 内部变量; C: 常数

NUM2STR →数值转换ASCII字符

指令格式	***P1*** **= NUM2STR(*P2*,*P3*)**	数据类型	U/UD
功能	将 ***P2*** 内的数值转换成 ***P3*** 个 ASCII 字符，并将转换结果保存到 ***P1***。		
P1 (I)	转换结果。		
P2 (I/C)	需转换的常数或是存储数值的地址。		
P3 (I/C)	指定转换字符数的确切数目。如果 ***P2*** 数值的位数小于 ***P3***，那么要在转换后的字符串前补上相应数量的零；如果 ***P2*** 数值的位数大于 ***P3***，那么超出的字符就要被释放；如果 ***P3*** 等于 0，那么转换后的字符串就根据实际长度保存到 ***P1***。		
例 1	$U120 = 123 $U100 = NUM2STR($U120, 0) (U) /* 执行命令后，字节数据$U100 所包含 ASCII 文字符串为“123”。 */		
例 2	$U120 = 1234567 (UD) $U100 = NUM2STR($U120, 10) (UD) /* 执行命令后，字节数据$U100 所包含 ASCII 文字符串为“0001234567”。*/		
例 3	$U120 = 1234567 (UD) $U100 = NUM2STR($U120, 5) (UD) /* 执行命令后，字节数据$U100 所包含 ASCII 文字符串为“34567”。*/		

I: 内部变量; C: 常数

TIME2STR →系统时间转换成字符串

指令格式	***P1*** **= TIME2STR(*P2*)**	数据类型	U
功能	将当前系统时间转换成字符串形式，字符串的格式由 ***P2*** 决定，并将转换结果保存到 ***P1***。		
P1 (I)	转换结果。		
P2 (I/C)	指定转换后的格式		
例 1	$U10 = TIME2STR(0) /* 假设当前系统时间是 12:30:59，执行该命令后，字节数组$U10~$U11 所包含的字符转存为“123059”。 */		

格式	***P2*** **的值**	备注
hhmmss	0	hh: 时(00~23); mm: 分(00~59) ; ss: 秒(00~59)
hhmm	1	hh, mm: 同上

I: 内部变量; C: 常数

DATE2STR →系统日期转换成字符串

<table>
<tr><td>指令格式</td><td colspan="2">P1 = DATE2STR(P2)</td><td>数据类型</td><td>U</td></tr>
<tr><td>功能</td><td colspan="4">将当前系统日期转换成 ASCII 文字符串形式，字符串的格式由 P2 决定，并将转换结果保存到 P1。</td></tr>
<tr><td>P1 (I)</td><td colspan="4">转换结果。</td></tr>
<tr><td>P2 (I/C)</td><td colspan="4">指定转换后的格式
<table>
<tr><th>格式</th><th>P2 的值</th><th>备注</th></tr>
<tr><td>YYMMDD</td><td>0</td><td>YY: 年 (00~99); MM: 月(01~12); DD: 日(01~31)</td></tr>
<tr><td>YYMM</td><td>1</td><td>YY, MM: 同上</td></tr>
<tr><td>YYMMMDD</td><td>2</td><td>YY: 年 (00~99); MMM: 月(JAN~DEC); DD: 日(01~31)</td></tr>
<tr><td>YYMMM</td><td>3</td><td>YY, MMM: 同上</td></tr>
</table></td></tr>
<tr><td>例 1</td><td colspan="4">$U10 = DATE2STR(0) /* 假设当前系统日期是 2008 年 12 月 7 日，执行该命令后，字节数组$U10 所包含的字符转存为“081207”。*/</td></tr>
<tr><td>例 2</td><td colspan="4">$U20 = DATE2STR(3) /*假设当前系统日期是 2008 年 12 月 31 日，执行该命令后，字节数组$U20 所包含的字符转存为“08DEC”。*/</td></tr>
</table>

I: 内部变量; C: 常数

TD2STR →系统时间和日期转换成字符串

<table>
<tr><td>指令格式</td><td colspan="2">P1 = TD2STR(P2)</td><td>数据类型</td><td>U</td></tr>
<tr><td>功能</td><td colspan="4">将当前系统时间和日期转换成字符串形式，字符串的格式由 P2 决定，并将转换结果保存到 P1。</td></tr>
<tr><td>P1 (I)</td><td colspan="4">转换结果。</td></tr>
<tr><td>P2 (I/C)</td><td colspan="4">指定转换后的格式：
<table>
<tr><th>格式</th><th>P2 的值</th><th>备注</th></tr>
<tr><td>YYMMDD_hhmmss</td><td>0</td><td>YY: 年(00~99); MM:月(01~12); DD: 日(01~31)
hh: 时(00~23); mm: 分(00~59) ; ss: 秒(00~59)</td></tr>
<tr><td>YYMMMDD_hhmmss</td><td>1</td><td>YY, DD, hh, mm, ss: 同上 MMM: 月(JAN~DEC)</td></tr>
<tr><td>YYMMDD_hhmm</td><td>2</td><td>YY, DD, hh, mm: 同上; MM: 月(01~12)</td></tr>
<tr><td>YYMMMDD_hhmm</td><td>3</td><td>YY, DD, hh, mm: 同上; MMM: 月(JAN~DEC)</td></tr>
</table></td></tr>
<tr><td>例 1</td><td colspan="4">$U10 = TD2STR(0) /* 假设当前系统日期是 2008 年 12 月 7 日， 系统时间是 15:18:30。执行该命令后，字节数组$U10 所包含的字符转存为“081207_151830”。*/</td></tr>
<tr><td>例 2</td><td colspan="4">$U20 = TD2STR(3) /* 假设当前系统日期是 2008 年 12 月 31 日，系统时间是 13:30:00。执行该命令后，字节数组$U20 所包含的字符转存为“08DEC31_1330”。*/</td></tr>
</table>

I: 内部变量; C: 常数

I2A →整数转换成ASCII文字符串

指令格式	***P1* = I2A(*P2,P3*)**	数据类型	U/S/UD/SD
功能	将保存在 ***P2*** 内的整数转换为字符串，并将转换结果保存到 ***P1***。该字符串的格式由 ***P3*** 指定。		
P1 (I)	存储运行结果的字节数组。它是一个含零终止符的字符串。		
P2 (I/C)	需转换的整数或是存储整数的地址。		
P3 (I/C)	指定字符串的小数字数。当 ***P3*** 等于 n 时，小数点插在整数右起第 n 位之前。当 ***P3*** 等于 0 时，没有小数点。		
例 1	$U120 = 123 $U100 = I2A($U120, 5, 0) /* 执行命令后，字节数组$U100 为 “123”. */		
例 2	$U120 = 1234567 (UD) $U100 = I2A($U120, 6, 2) (UD) /* 执行命令后，字节数组$U100 为 “12345.67”. */		
例 3	$U120 = -12345 (S) $U100 = I2A($U120, 5, 1) (UD) /* 执行命令后，字节数组$U100 为 “-1234.5”. */		

I: 内部变量; C: 常数

A2I →ASCII文字符串转换成整数

指令格式	***P1* = A2I(*P2,P3,P4*)**	数据类型	U/S/UD/SD
功能	将保存在 ***P2*** 的字符串转换成整数，并将运行结果保存到 ***P1***。		
P1 (I)	运行结果。如果转换时发生错误，结果等于 0。		
P2 (I)	需转换字符串的起始地址。		
P3 (I/C)	指定字符串的长度。***P3*** 可以等于 0。当 ***P3*** 等于 0 时，字符串必须是一个含零终止符的字符串。		
P4 (I/C)	指定字符串中需要转换的小数字数。		
例 1	$U120 = “123” $U100 = A2I($U120, 0, 1) /* 执行命令后，$U100 内的 16 位正整数值等于 1230。*/		
例 2	$U120 = “123” $U100 = A2I($U120, 0, 1) /* 执行命令后，$U100 内的 16 位正整数值等于 1230。*/		
例 3	$U120 = “123” $U100 = A2I($U120, 0, 1) /* 执行命令后，$U100 内的 16 位正整数值等于 1230。*/		

I: 内部变量; C: 常数

F2A→浮点数转换成ASCII文字符串

指令格式	***P1* = F2A(*P2,P3*)**	数据类型	F
功能	将保存在 ***P2*** 的浮点数转换成字符串，并将转换结果保存到 ***P1***。该字符串的格式由 ***P3*** 指定。		
P1 (I)	存储运行结果的字节数组。它是一个含零终止符的字符串。		
P2 (I/C)	需转换的浮点数或是存储浮点数的地址。		
P3 (I/C)	指定字符串包含的小数字数。		
例 1	$U120 = 123.45 (F) $U100 = F2A($U120, 2) /* 执行命令后，字节数组$U100 为 “123.45”. */		
例 2	$U120 = 567.89 (F) $U100 = F2A($U120, 1) (UD) /* 执行命令后，字节数组$U100 为 “567.8”. */		
例 3	$U120 = -1234 (S) $U100 = F2A($U120, 1) (UD) /* 执行命令后，字节数组$U100 为“-1234.0”. */		

I: 内部变数; C: 常数

A2F →ASCII文字符串转换成浮点数

指令格式	***P1* = A2F(*P2,P3*)**	数据类型	F
功能	将保存在 ***P2*** 的字符串转换成浮点数，并将转换结果保存到 ***P1***。		
P1 (I)	运行结果。如果转换时发生错误，结果等于 0。		
P2 (I)	需转换字符串的起始地址。		
P3 (I/C)	指定字符串的长度。***P3*** 可以等于 0。当 ***P3*** 等于 0 时，字符串必须是一个含零终止符的字符串。		
例 1	$U120 = “123.4” $U100 = A2F($U120, 0) /* 执行命令后，$U100 内的浮点数值为 123.4。*/		
例 2	$U120 = “1234567” $U100 = A2F($U120, 6) (UD) /* 执行命令后， $U100 内的浮点数值为 123456。*/		
例 3	$U120 = “-123.45” $U100 = A2F($U120, 0) (S) /* 执行命令后，$U100 内的浮点数值为 -123.45。*/		

I: 内部变数; C: 常数

14.4.18. 运行操作

RUN →运行

指令格式	**RUN(*P1*)**
功能	运行在同一计算机上的可执行程序 ***P1***。这个命令只能在图控软件上使用。
P1 (I/A)	需要运行的可执行程序的文件名。
例 1	RUN "ABC.exe" /*运行程序 ABC */
例 2	$U10 = "XYZ.bat" RUN($U10) /* 运行批处理文件 XYZ */

I: 内部变量; A: ASCII 码字符串

RUNW →运行并且等待

指令格式	***P1*** **= RUNW(*P2*)**
功能	运行在同一计算机上的可执行程序 ***P2***，并将运行结果保存到 ***P1***。需要注意的是这个命令之后的命令语句，需要等到该可执行程序结束运行后才会执行。这个命令只能在图控软件上使用。
P1 (I)	运行结果；如果这个代码等于 0，表示运行成功；否则运行失败。
P2 (I/A)	需要运行的可执行程序的文件名。
例 1	$U10 = RUNW "ABC.exe" /* 运行程序 ABC，并将运行结果保存到 $U10。*/ IF $U10 == 0 /* 如果运行结果为 0，那么运行批处理文件 XYZ。*/ $U20 = "XYZ.bat" $U11= RUNW($U20) /* 运行批处理文件 XYZ。 */ ENDIF

I: 内部变量; A: ASCII 码字符串

14.4.19. 打印操作

PRINT →打印

指令格式	***P1* = PRINT(*P2*,*P3*)**	数据类型	U
功能	将 ***P2*** 的数据依 ***P3*** 所指定的字节个数输出到打印机通讯接口，并将完成代码保存到 ***P1***。		
P1 (I)	运行后产生的完成代码。代码描述见下表： **代码** / **描述** 0 成功 1 打印机没有准备好 3 系统错误 4 打印机正忙 7 未指定打印机		
P2 (I)	源数据存储器首地址，源数据是指要被输出的数据(字节数)。注意如果要打印中文字需要先确认该台打印机是否内建中文字集库。		
P3 (I/C)	要被输出到打印机的字节数。		
例 1	$U10 = "This is a test." $U20 = PRINT($U10, 24) /*发送 This is a test.字符串到打印机。 */ $U10 = 10 $U20 = PRINT($U10, 1) /*发送换行字符到打印机。 */ $U10 = 12 $U20 = PRINT($U10, 1) /*发送换页字符到打印机。 */		
例 2	$U10 = 0x401b /* ESC, '@' */ $U20 = PRINT($U10, 2) /* 初始化 EPSON 打印机。 */		

I: 内部变数; C: 常数

PRINT_SCREEN →打印画面

<table>
<tr><td>指令格式</td><td colspan="2">P1 = PRINT_SCREEN(P2,P3)</td><td>数据类型</td><td>U</td></tr>
<tr><td>功能</td><td colspan="4">打印画面编号 P2 的画面内容到打印机，并将完成代码保存到 P1。</td></tr>
<tr><td>P1 (I)</td><td colspan="4">运行后产生的完成代码。代码描述见下表：
<table>
<tr><th>代码</th><th>描述</th></tr>
<tr><td>0</td><td>操作成功</td></tr>
<tr><td>1</td><td>打印机没有准备好</td></tr>
<tr><td>2</td><td>画面编号无效</td></tr>
<tr><td>3</td><td>系统错误</td></tr>
<tr><td>4</td><td>打印机正忙</td></tr>
<tr><td>5</td><td>系统正忙</td></tr>
<tr><td>6</td><td>非正当使用该指令(见注)</td></tr>
<tr><td>7</td><td>未指定打印机</td></tr>
</table>
注意：该指令只能用在主宏，事件宏，时间宏和循环宏中。</td></tr>
<tr><td>P2 (I/C)</td><td colspan="4">要打印的画面编号。 要打印的画面区域的指定是在每一画面的画面属性对话框来设置。</td></tr>
<tr><td>P3 (I/C)</td><td colspan="4">保留将来扩充功能使用，目前必须为零。</td></tr>
<tr><td>例 1</td><td colspan="4">$U0 = PRINT_SCREEN(28, 0) /* 打印画面编号 28 的画面。*/</td></tr>
</table>

I: 内部变数; C: 常数

BLANK →清空

指令格式	***P1* = BLANK (*P2*)**	数据类型	U
功能	清空打印缓冲区 ***P1***，也就是让打印缓冲区 ***P1*** 仅含有空白字符。		
P1 (I)	要清空的打印缓冲区。 打印缓冲区是一个字节数组。 在打印任何字符串到一个打印缓冲区之前，必须将它清空。		
P2 (I/C)	打印缓冲区的长度。长度的单位是字节。 举例而言， 一个长度为 40 的打印缓冲区，是一个含有 40 个字节的数组，它可以含有 40 个英文字符或 20 个中文字。		
例 1	BLANK($U100, 80) /*清空长度为 40，位置在$U100 的打印缓冲区。*/		

I: 内部变数; C: 常数

P2B →打印字符串

指令格式	***P1* = P2B (*P2, P3*)**	数据类型	U
功能	打印字符串 ***P1*** 到打印缓冲区 ***P2*** 的位置 ***P3***。		
P1 (I)	要打印的字符串。字符串是一个字节数组。要打印的字符串必须是以零结束的字符串。		
P2 (I)	接受字符串 ***P1*** 的打印缓冲区。打印缓冲区是一个字节数组。		
P3 (I/C)	在打印缓冲区中摆放字符串 ***P1*** 的字节位置。 字节位置是重从 0 开始算起。 举例而言，要打印字符串到打印缓冲区的起头位置，就要设 ***P3*** 为 0。		
例 1	BLANK($U100, 20) /*清空打印缓冲区*/ $U10 = “Weight:” P2B($U100, $U10, 0) /*打印字符串“Weight:” 到打印缓冲区的字节位置 0*/ $U10 = I2A(1234, 2) /*字节数组将含有字符串“12.34”*/ P2B($U100, $U10, 8) /*打印字符串“12.34” 到打印缓冲区的字节位置 8*/ $U10 = “kg” P2B($U100, $U10, 14) /*打印字符串“kg” 到打印缓冲区的字节位置 14*/ PRINT($U100, 20) /*打印字符串“Weight: 12.34 kg”到打印机*/		

I: 内部变数; C: 常数

P2B_R →打印字符串（靠右对齐）

指令格式	***P1* = P2B_R (*P2,P3*)**	数据类型	U
功能	打印字符串 ***P1*** 到打印缓冲区 ***P2*** 。打印位置 ***P3*** 是字符串在打印缓冲区内靠右对齐的位置。		
P1 (I)	要打印的字符串。字符串是一个字节数组。要打印的字符串必须是以零结束的字符串。		
P2 (I)	接受字符串 ***P1*** 的打印缓冲区。打印缓冲区是一个字节数组。		
P3 (I/C)	字符串靠右对齐的位置，也就是字符串的最后一个字符在打印缓冲区中要摆放的字节位置 。字节位置是从 0 开始算起。 举例而言，要打印长度为 6 的字符串到打印缓冲区的起头位置， 就要设 ***P3*** 为 5。		
例 1	BLANK($U100, 20) /*清空打印缓冲区*/ $U10 = “Weight:” P2B_R($U100, $U10, 6) /*打印字符串“Weight:” 到打印缓冲区并将字符串靠右对齐到字节位置 6*/ $U10 = I2A(1234, 2) /*字节数组将含有字符串“12.34”*/ P2B_R($U100, $U10, 12) /*打印字符串“12.34” 到打印缓冲区并将字符串靠右对齐到字节位置 12*/ $U10 = “kg” P2B_R($U100, $U10, 15) /*打印字符串“kg” 到打印缓冲区并将字符串靠右对齐到字节位置 15*/ PRINT($U100, 20) /*打印字符串“Weight: 12.34 kg”到打印机*/		

I: 内部变数; C: 常数

14.4.20. 声音操作

SOUND →声音

指令格式	**SOUND (*P1, P2, P3*)**	数据类型	U
功能	播放声音。		
P1 (I/C)	声音编号。 注: 声音与编号是在人机应用的声音表中定义。		
P2 (I/C)	播放次数。如果只播放一次，就设为 1。		
P3 (I/C)	连续播放的间隔时间。时间单位是 0.1 秒。如果不需间隔，就设为 0。		
例 1	SOUND(10, 5, 3) /* 播放编号 10 的声音 5 次，回放的间隔时间为 0.3 秒. */		

I: 内部变数; C: 常数

STOP_SOUND →停止声音

指令格式	**STOP_SOUND**
功能	停止播放当前声音。
例 1	STOP_SOUND /*停止播放当前声音. */