

XC Studio 编程手册

Version 1.0

前言

本手册编写的目的是为了更好的服务全国所有的客户，为客户提供更为全面的参考资料，我司致力于对产品的不断优化改进，让客户快速了解产品，产品手册也会持续更新。

安全提示

1. 注意事项

控制器集成度高，设计尺寸小巧轻便，易于安装，用户可以有效地利用空间。可以将控制器安装在面板或标准导轨上，并且可以选择水平或垂直安装方式。

作为安装布置系统中各种设备的基本规则，将控制器等低压逻辑型设备与热辐射、高压和电噪声隔离开，远离粉尘、腐蚀性气体、水、油、化学品等场所。

在面板上配置控制器的布局时，由于控制器长时间运行会产生发热现象，应考虑将控制器布置在较凉爽区域，少暴露在高温环境中会延长电子设备的使用寿命，温度过高的环境可能会导致控制器无法正常使用。

安装时还要考虑面板中设备的布线，避免将低压信号线和通信电缆铺设在具有交流动力线和高能量快速开关直流线的槽中。

四周留出足够的空隙以便控制器冷却和接线，控制器可通过自然对流冷却、为保证适当冷却，在设备上方和下方必须留出一定的空隙。

2. 警告

需要将控制器安装在外壳、控制柜或电控室内等不易触碰的地方，避免非操作人员接触。

机械运行时为了您的人身安全请勿靠近。

请勿对本产品进行分解、修理或改装。

在外部采用控制电路构成紧急停止电路、联锁电路、限制电路等于安全保护相关的电路。

安装或拆卸控制器时应先将控制系统的电源全部断开，避免发生触电或意外设备操作导致不必要的损失。

不遵守这些以上要求可能会导致人员重伤和财产损失，苏州新川智能装备有限公司不承担相应风险与责任。

3. 接线要求

使用螺丝将控制器固定，防止产品跌落或遭受异常冲击导致使用故障。

为保证通讯稳定，控制器通讯电缆应选用带屏蔽层的高性能电缆。

采用 24V 直流电源给控制器供电，另 IO 口需要单独供电，和控制器电源分开。

控制器网络的各个部件为了安全起见，确保控制器和相关设备的所有公共端和接地连接在同一个点接地，该点应该直接连接到系统的大地接地。确定接地点时，应考虑安全接地要求和保护性中断装置的正常运行。

完成所有接线工作后再给控制电路通电，请勿在带电的情况下操作。

所有线路连接应尽可能短，能减少干扰，保证通信质量。

版权说明

本手册版权归苏州新川智能装备有限公司所有，未经苏州新川智能装备有限公司书面许可，任何人不得翻印、翻译和抄袭本手册中的任何内容。本手册中的信息资料仅供参考。由于改进设计和功能等原因，苏州新川智能装备有限公司保留对本资料的最终解释权！内容如有更改，恕不另行通知！如需新版资料，请联系我司相关人员。



调试机器要注意安全！请务必在机器中设计有效的硬件或者机械安全保护装置，并在软件中加入出错处理程序，否则所造成的损失，苏州新川智能有限公司没有义务或责任对此负责。

第一章 产品简介

1.1. 产品概述

控制器实现了对机械传动部件的位置、速度、加速度等的实时控制，使其按照预期的轨迹与规定的运动参数完成相应的动作。

控制系统以处理器、检测机构、执行机构为核心，实现逻辑控制、位置控制、轨迹加工控制、机器人运动控制等。其中处理器通常是可编程控制器、单片机、或运动控制器，相当与系统的大脑，主要负责对接受到的信号进行逻辑处理，并给执行机构下发命令，协调系统的正常运转。检测机构通常由各种传感器构成，相当与系统的眼睛，目的是检测系统条件的变化并反馈给控制器，执行机构通常由伺服单元、阀门构成，相当于系统的双手，主要执行控制器下发的命令。

运动控制器是运动控制系统的核心部件，负责产生运动路径的控制指令，用于设备的逻辑控制，将运动参数分配给需要运动的轴，并对被控对象的外部环境变化及时做出响应。

通用运动控制器通常都提供一系列运动规划方法，基于对冲击、加速度和速度等这些可影响动态轨迹精度的量值加以限制，提供对运动控制过程的运动参数的设置和运动相关的指令，使其按预先规定的运动参数和规定的轨迹完成相应的动作。

1.2. 产品优点

新川控制系统能满足各行各业的机器人控制需求，支持 SCARA、DELTA、桁架手臂、六轴等。

运动控制产品支持直线、圆弧、空间圆弧、椭圆、螺旋等多种插补运动。支持速度前瞻、电子凸轮、电子齿轮、螺距补偿、固步跟踪、运动叠加、虚拟轴、精准输出、硬件位置锁存、位置比较输出、连续插补、运动暂停等功能；

支持 EtherCAT 工业以太网运动控制总线，在性能和稳定性方面均处于领先地位。

EtherCAT 总线周期最快达 100 微秒，同时还支持总线轴硬件位置锁存与位置比较输出。

新川还为控制产品提供了强大的 XCStudio 软件开发环境，操作简单易学。

控制器支持以太网、U 盘、CAN 总线、RS485 串口、RS232 串口等通讯接口，通过 CAN 总线或 EtherCAT 总线可以扩展模块，从而扩展输入输出点或脉冲运动轴数（CAN 总线两端需要并接 120 欧姆的电阻）。

1.3. 控制器主要功能描述

控制器的主要功能简介：

项目		描述
任务		以指定条件执行 I/O 刷新和用户程序的功能，支持多任务同时进行，互不干扰。
中断		支持三种类型的中断（外部中断、定时器中断、掉电中断）
监控窗口		监控变量常量、输入输出、轴坐标等
编程语言种类		BASIC
在线命令		在线命令栏输入指令参数发送给控制器立即执行
通讯接口	串口	232 串口和 485 串口，支持 MODBUS_RTU 协议和自定义通讯

	网口	通讯速度快，接线方便，支持 MODBUS_TCP 协议和自定义通讯
	U 盘	插入 U 盘，数据交互
	CAN 总线	连接 IO 扩展模块，控制器互联
	EtherCAT 总线	连接 EtherCAT 驱动器或 EtherCAT 扩展模块
数据类型	自定义数组	集中相同数据类型的元素，默认浮点型
	自定义变量	默认浮点型
	自定义常量	可以是布尔型，字符串型，时间型，日期型，整型等
	寄存器	自带四类寄存器，TABLE，MODBUS，VR，FLASH
常用运动控制功能	点位运动	点动
	插补运动	直线、圆弧、空间圆弧、椭圆、螺旋等多种插补，支持连续插补
	运动叠加	不同轴的运动叠加
	轨迹前瞻	根据前瞻参数，速度自行优化
	位置锁存	根据外部信号触发的发生记录轴的位置
	位置比较输出	到达比较点输出 OP 信号，连续比较，快速响应
	精准输出	OP 快速响应

1.4. 控制器应用场景

上下料机械手、紧螺钉机设备、分拣、码垛等等。

1.5. 控制器接口

支持以太网、USB、CAN、485、232 等通讯接口，通过 CAN 总线或 EtherCAT 总线可以连接相应扩展模块，从而扩展输入输出点数或脉冲轴数（CAN 总线接线两端需要并接 120 欧姆的电阻）

接口功能如下表：

标识	接口	个数	说明
RS232	232 串口	1 个	采用 MODBUS_RTU 协议
485	485 串口	1 个	采用 MODBUS_RTU 协议
CAN	CAN 总线	1 个	连接 CAN 扩展模块或控制器
ETHERNET	网口	1 个	采用 MODBUS_TCP 协议，通过交换机扩展接口个数
EtherCAT	EtherCAT 总线	1 个	连接 EtherCAT 驱动器或 EtherCAT 扩展模块
UDISK	U 盘	1 个	插入 U 盘设备
E +24V	主电源	1 个	24V 直流电源供电
IN	数字量输入	24 个	NPN 类型，内部 24V 供电
OUT	数字量输出	12 个	NPN 类型，内部 24V 供电
AD	模拟量输入	2 个	精度 12 位，0-10V
DA	模拟量输出	2 个	精度 12 位，0-10V

1.6. 控制器的使用

1. 准备工作

软件：安装 XCStudio 编程软件。

设备：控制器、计算机

连接线：控制器与计算机通讯的连接线，控制器与驱动器轴接口的连接线，IO 接口、电源接口等的连接线。

2. 程序设计

1. 系统架构设计

根据功能需求选择所需元件和连接线，熟悉与该功能相关的控制指令的使用方法，设计系统软件的整体构成，包括变量设计、任务设计、程序功能设计等。

2. 软件设定与编程

编程需要设置的参数：BASE 选择轴号、XACCEL 轴加速度，XDECEL 轴减速度等基础轴参数，再给轴发送运动指令。

3. 安装与接线

计算机与控制器接线：可以采用串口通信或网口通讯，串口通信连接控制器的 RS232 串口，网口通讯连接控制器的 EtherNET 网口。

4. 试运行

请确认接线无误后上电，将调试好的程序下载到控制器，开始试运行。

第二章 XC Studio 软件编程

2.1. 编程软件简介

XCStudio 编程软件支持 Basic 组态编程，提供所有标准程序语法、变量、数组、条件判断、循环及数学运算。此扩展的 Basic 指令和函数能提供广泛的运动控制功能，例如单轴运动、多轴的同步和插补运动，同时还有对数字和模拟 I/O 的控制。

支持以下功能：

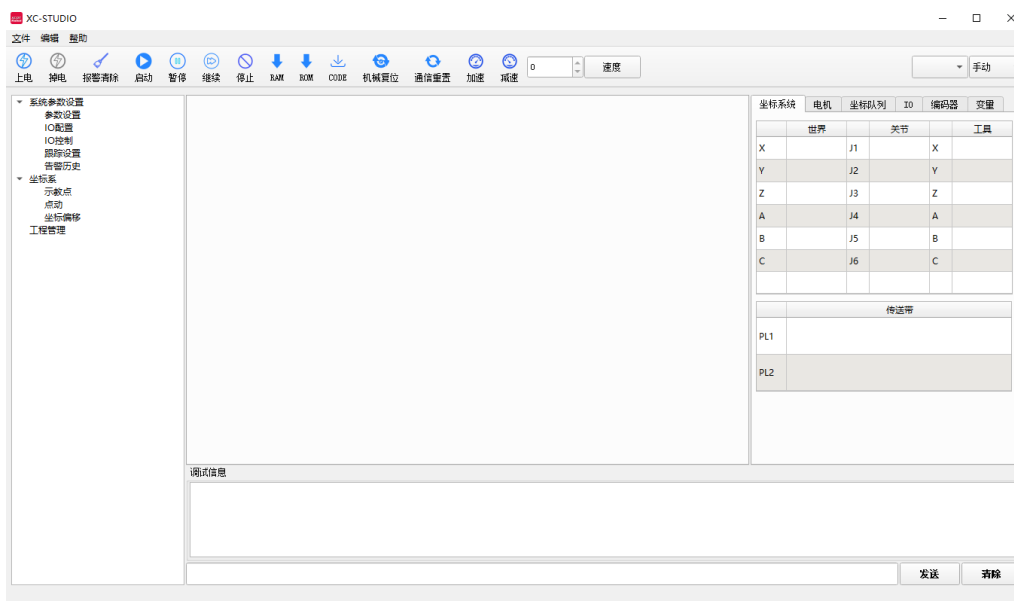
1. 自定义 SUB 过程，可以把一些通用的功能编写为自定义 SUB 过程，方便程序编写和修改。
2. G 代码形式的 SUB 过程，支持 G00、G01、G02、G03、G04、G90、G92 等常用指令。
3. 支持全局的变量（GLOBAL）、数组和 SUB 过程；文件模块变量、数组和 SUB 过程；以及局部变量（LOCAL）。
4. 中断程序（掉电中断、外部中断、定时器中断），例如掉电中断，通过掉电中断时保存数据，可以使得掉电的状态得到恢复。

具有实时多任务的特性，多个 Basic 程序可以同时构建并多任务实时运行，使得复杂的应用变得简单易行。

通过 PC 在线发送 Basic 命令也可以实现同样的效果，控制器内置的 Basic 程序和 PC 在线 Basic 命令可以同时多任务运行。

2.2. 新建工程

1. 新建项目：菜单栏“文件”→“新建项目”，或者 Ctrl+N。



2. 点击“新建项目”后弹出“另存为”界面，选择一个文件夹打开，输入文件名后保存项目，后缀为“.XCJ”。

3. 编辑程序：程序编写完成，点击保存文件，新建的 Basic 文件会自动保存到项目 XCJ 所在的文件夹下。

4. 下载程序：点击菜单栏按钮“RAM”或按钮“ROM”，下载成功命令和输出窗口会有提示，同时程序下载到控制器并自动运行，提示加载 100%则控制器程序启动完成。



RAM 下载掉电后程序不保存，ROM 下载掉电后程序保存。下载到 ROM 的程序下次连接上控制器之后程序会自动按照任务号运行。

2.3. 在线命令与输出

在线命令与输出窗口可以查询与输出控制器的各种参数、打印程序运行结果、打印程序错误信息，软件开发人员在程序中给出的打印输出函数（由?、PRINT、WARN、ERROR、TRACE 等命令输出）。

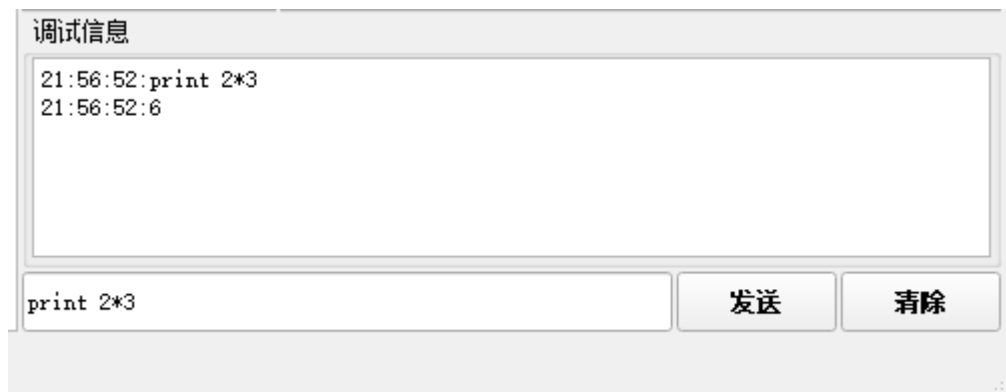
注意问号使用英文符号，中文符号输入无效。ERRSWITCH 为 TRACE、WARN、ERROR 指令的控制开关，不同的参数值对应不同的输出效果：

0: TRACE、WARN、ERROR 指令全部不输出

1: 只输出 ERROR 指令

2: 输出 WARN、ERROR 指令

在线命令与输出窗口如下所示，命令输入“print 2*3”窗口会打印计算结果。



修改变量的值。通过“在线命令”可以实现 VR 变量、TABLE 变量、MODBUS 变量、全局变量、系统设置。

第三章 X Basic 编程基础

本手册以 Basic 编程语言为例进行详细说明。

3.1. 编程基础知识

3.1.1. 程序

程序是由序列组成的，告诉计算机如何完成一个具体的任务。程序是软件开发人员根据用户需求开发的、用程序设计语言描述的适合计算机执行的指令（语句）序列。

X Basic 语法不区分大小写，程序指令的所有标点符号应为英文格式。

一个程序应该包括以下两方面的内容：

1. 对数据的描述。在程序中要指定数据的类型和数据的组织形式，即数据结构（参见：DIM、GLOBAL、CONST 等变量定义语句描述）。
2. 对操作的描述。即操作步骤，也就是算法，结合到运动控制就是运动与动作的工艺。

1. 常见程序结构

为编写算法，我们一般要用到以下几种程序结构描述方式：顺序、选择、循环、延时、等待和子程序调用，子程序调用参见下节。

1. 顺序

在没有条件和循环的情况下，程序总是从上往下运动。当设置自动运行时，文件缺省都是从文件开始顺序往下执行的。

功能块 1

功能块 2

如上，功能块 1 先执行，然后是功能块 2

BASIC 编程下，程序从上往下扫描一次。

PLC 编程下，程序从上往下周期扫描。

2. 选择

根据执行条件的不同，选择不同的语句执行。主要的选择语句有：IF THEN，ON GOTO，ON GOSUB 等。

例程 1：

```
DIM aa
```

```
aa=1
```

```
IF aa=0 THEN
```

```
    语句 1
```

```
ELSEIF aa=1 THEN
```

```
    语句 2
```

```
ELSE
    语句 3
ENDIF
END
```

例程 2:

```
DIM a
a=100
ON a>10 GOTO label1
a=1000
END      '主程序结束
```

```
label1:
PRINT a
END      'goto 跳转无法 return 返回。
```

3. 循环

程序重复执行，则称为循环。主要的循环语句有：FOR NEXT，WHILE WEND，REPEAT UNTIL 等。

例程 1:

```
DIM a
a=0
FOR i = 1 TO 10 STEP 1
    a = a+1
    PRINT a
NEXT
END
```

例程 2:

```
DIM a
a=0
WHILE IN(1)=OFF '直到输入 1 有效，退出循环
    a=a+1
    PRINT a
    DELAY(1000)
WEND
END
```

4. 延时

程序遇到 DELAY 延时语句，会停止对应时间后，再继续向下执行。

例程:

```
PRINT 1
DELAY(2000) '延时 2000ms
PRINT 2      '打印 1 延时 2000ms 后打印 2
END
```

5. 等待

程序遇到 WAIT 等待语句时，会停止在此行，直到 WAIT 条件满足，才继续向下执行。

例程

BASE(0,1)

MOVE(100,100)

WAIT IDLE '等待当前插补运动结束

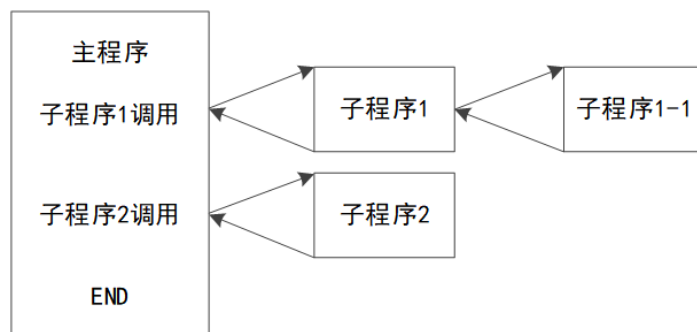
PRINT "运动完成"

程序除了在执行到 WAIT 等待语句，DELAY 延时语句时会阻塞，另外在扫描到运动指令时，如果轴的运动缓冲区满了，此时程序会停在当前运动指令行，直到当前运动完成，缓冲区空出一条，程序就会继续往下执行。缓冲区查看运动缓冲说明。

2. 子程序

编程过程中会经常应用到子程序（Basic 中使用 SUB 指令定义子程序），运用子程序可将编程模块化，各模块之间的关系尽可能简单，在功能上相对独立，相当于简化了主程序，使编程效率更高更易读，能有效地将一个较复杂的程序系统设计任务分解成许多易于控制和处理的子程序和子任务，便于开发和维护。

主程序和子程序的执行逻辑如下图：



SUB 子程序可作为一个子程序开启，运行完 END SUB 返回主程序，也可以使用任务指令 RUNTASK 开启，作为一个任务单独运行，作为而任务开启后与主程序无关，运行完成后子程序任务结束，不返回主程序。

主程序调用子程序嵌套最多 8 层。

子程序分为全局 SUB，文件模块 SUB，全局 SUB 可以应用在所有文件中，文件模块 SUB 只能用于当前文件，子程序还能用于传递参数以及返回参数。

示例：

SUB sub1() '定义过程 SUB1，只能在当前文件中使用

 ?1

 ...

END SUB '自定义 SUB 过程结束

GLOBAL SUB g_sub2() '定义全局过程 g_sub2，可以在任意文件中使用

 ?2

 ...

END SUB '自定义 SUB 过程结束

```

GLOBAL SUB g_sub3(para1,para2)      '定义全局过程 g_sub3, 传递两个参数
    ?para1,para2
    ...
    RETURN para1+para2              '函数返回参数相加
END SUB                             '自定义 SUB 过程结束

```

3.1.2. 数据相关

特别注意：所有系统自带的寄存器区间都是 0-3999，3999 以后的为系统内部参数用户不可写入数据，如任意改变内部参数造成的后果，新川智能装备有限公司不承担任何责任。具体如下：

MODBUS_BIT 、 MODBUS_REG 、 MODBUS_LONG 、 VR 、 VR_INT 、 MODBUS_IEEE 、
VRTABLE

1. 数据定义

1. 变量定义

变量是用户可以自定义的参数，变量用于暂时保存与外部设备的通信数据或任务内部处理需要的数据，换言之，它是用于保存带名称和数据类型等属性的数据，无需指定变量与存储器地址之间的分配。

变量定义指令：分为全局变量（GLOBAL）、文件模块变量（DIM）、局部变量（LOCAL）三种。

全局变量（GLOBAL）：可以在项目内的任意文件中使用；

文件模块变量（DIM）：只能在本程序文件内部使用；

局部变量（LOCAL）：主要用在 SUB 中，其他文件无法使用。

变量可以不经定义直接赋值，此时的变量默认为文件模块变量。

示例：

```

GLOBAL g_var2      '定义全局变量 g_var2
DIM VAR1           '定义文件变量 VAR1
SUB aaa()
    LOCAL v1       '定义局部变量 V1
    v1=100
END SUB

```

2. 常量定义

变量的值因代入该变量的数据而异。与之相对的固定不变的值为常数，常量的值一经定义不能再修改，只可读取。

CONST 定义常量，一次只能定义一个，且定义与赋值必须在一行。常量可定义为全局常量 GLOBAL CONST，全局常量可以在任意文件中使用，不存在 LOCAL CONST 的写法。常数与变量不同，不是保存在存储器中的信息，常见的常量有布尔型、字符串型、时间型、日期型、整型等。

示例：

```
CONST    MAX_VALUE = 100000      '定义文件常量
GLOBAL CONST    MAX_AXIS=6      '定义全局常量
```

3. 数组定义

数组指定是指将相同属性的数据集中后对其进行统一定义，并对数据个数进行指定。构成数组的各数据称为“元素”。

数组定义相关指令为 GLOBAL、DIM，不支持 LOCAL 定义。

数组定义时注意数组空间大小的指定，不能使用超出定义范围的空间，否则程序报错数组空间超限。

示例：

```
DIM array1(15)    '定义文件数组，可使用的数组空间编号为 0~14，共 15 个空间
GLOBAL array2(10)  '定义全局数组，可使用的数组空间编号为 0~9，共 10 个空间
```

2. 数据类型

在计算机内部，数据都是以二进制的形式存储和运算的，二进制数据中的位（bit）是计算机存储数据的最小单位。

一个二进制位只能表示 0 或 1 两种状态，要表示更多的信息，就要把多个位组合成一个整体，一般以 8 位二进制组成一个基本单位字节（Byte）。

字节是计算机数据处理的最基本单位，并主要以字节为单位解释信息。一般情况下，一个 ASCII 码占用一个字节，一个汉字国际码占用两个字节。计算机型号不同，其字长是不同的，常用的字长有 8、16、32 和 64 位。

单位转换：1Byte=8bit，1KB=1024B，1MB=1024KB，1GB=1024MB。

常见进制有二进制，八进制，十进制，十六进制。各种运动指令的参数默认为十进制数据。

名称	说明
位（Bit）	位为二进制数值之最基本单位，其状态非 1 即 0
位数（Nibble）	由连续的 4 个位所组成（如 bit3~bit0）一个位数可用以表示十进制数字 0~15 或十六进制 0~F
字节（Byte）	由连续之两个位数所组成（8 位，bit7~bit0）。可表示十进制数字 0~255 或十六进制的 00~FF
字符（Word）	由连续之两个字节所组成（16 位，bit15~bit0）可表示十进制数字 0~65535 或十六进制的 4 个位数值 0000~FFFF
双字符（Double Word）	由连续之两个字符所组成（32 位，bit31~bit0），可表示十进制数字 0~2 ³² -1 或十六进制的 8 个位数值 00000000~FFFFFFFF

数据类型是指对变量表示的值的形式和范围进行特定的规定。声明该变量时，数据类型的大小根据存储器内的数据范围大小而定，存储器内的数据范围越大，可表示的值的范围就越大。

基本数据类型	描述
布尔型	值为 0 或 1 的数据类型
整数型	值为整数的数据类型
实数型	值为实数的数据类型
日期型	以日期格式表示 DD: MM: YYYY
时间型	以时间格式表示 hh: mm: ss

字符串型	值为字符串的数据类型
------	------------

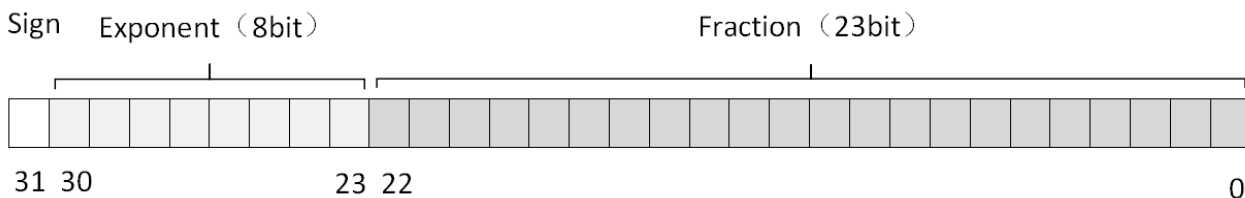
指令的输入或输出变量的数据类型由指令确定。

自定义变量的数据类型属于动态类型，将整数赋值给变量时，变量就是整型；将浮点数赋值给变量，变量就是浮点型。

自定义数组的数据类型分为单精度浮点数和双精度浮点数，参照下文浮点数相关说明。

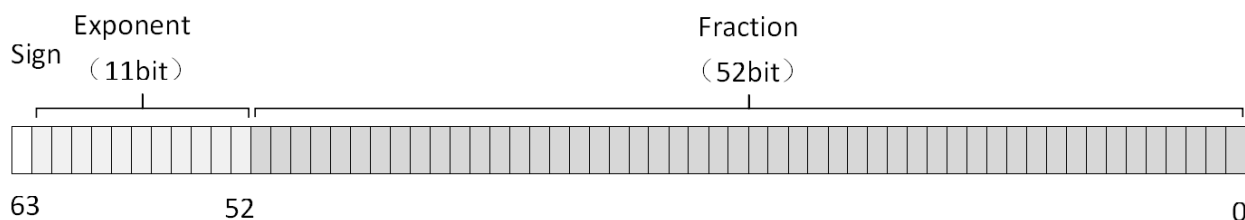
单精度浮点数 32 位，如下图。

数据格式属于单精度浮点数的有：VR、MODBUS_IEEE、TABLE 及自定义数组与变量（ZMC3 系列及之前的控制器）。



双精度浮点数 64 位，如下图。

数据格式属于双精度浮点数的有：TABLE 及自定义数组与变量（ZMC4 系列及之后的控制器）。



常用寄存器数据类型表：

寄存器类型	数据类型	取值范围
MODBUS_BIT	布尔型	0 或 1
MODBUS_REG	16 位整型	-32768 到 32767
MODBUS_LONG	32 位整型	-2147483648 到 2147483647
VR_INT		
MODBUS_IEEE	32 位浮点型	-3.4028235E+38 到 -1.401298E-45
VR		
TABLE ，自定义数组，变量	64 位浮点型	1.7E-308 到 1.7E+308
VRSTRING	字符	一个字符占一个 VR
MODBUS_STRING	字符	一个字符占 8 位

所有数据所需的存储器容量与各数据的总数据大小（容量值）不一致，原因在于，分配至存储器的数据的开头位置自动配置至各数据类型的“校准值（边界值）”倍数位置，各数据类型之间会产生空白。即使数据类型的种类相同，整体占用的数据大小仍会因数据类型的顺序而异。

3. 数据操作

不同类型数据之间的操作要注意数据类型，类型不匹配会产生下列问题：

1. 数据丢失：浮点型向整型转换时会丢失小数部分。

例程：

```
VR(0)=10.314
```

```
MODBUS_REG(0)=0
```

```
MODBUS_REG(0)=VR(0)
```

```
?MODBUS_REG(0)      '结果为 10
```

2. 强制转换：整型存储到浮点型寄存器后会变成浮点型，再使用整型操作数据可能会不正确。

3. 常见使用问题：获取日期时，不要使用单精度浮点型存储，因为日期格式是 8 位的，而单精度浮点数有效位只有 6 位，建议直接使用 32 位整型 MODBUS_LONG 来存储。

部分参数必须用字符串类型常量或变量，各种字符串可以通过“+”合并，对字符串的单个字节的操作需要利用数组来进行。

字符串相关指令列表：

字符串指令	描述
DIM	定义的数组可以直接作为字符串使用，每个元素表示一个字节
“ ”	双引号直接定义字符串常量
CHR	把 ASCII 转成一个字符串常量，此字符串只有一个字节
MODBUS_STRING	标准 MODBUS 协议定义字符串，每个 16 位寄存器存储 2 个字节
VRSTRING	VR 列表作为字符串使用，一个 VR 存储一个字节
±	操作符，两个字符串合并
VAL	数字字符串转换为数值
TOSTR	数值转换为字符串
STRCOMP	字符串比较函数
DMCPY	数组拷贝函数，可以用于拷贝字符串
HEX	返回为十六进制格式，只能用于打印
DATES	按 dd: mm: yyyy 格式返回 DATE 设置日期
DAYS	返回本日的星期英文名
TIMES	按 24 小时的格式 hh: mm: ss 返回当前时间

4. 掉电存储

机器人控制器具有掉电非易失保护寄存器 VR 和多个扇区存储 FLASH 块。

[ONPOWEROFF](#) 掉电中断可以用编写的程序记录掉电时的位置到 VR，系统重新上电时，使用程序将 VR 的数据恢复到当前位置，因为掉电执行时间非常短，建议只存几个数据。

使用 SETCOM 指令可以把 VR 与 MODBUS_REG 寄存器匹配起来，设置指令参数 variable，详细设置参见 [SETCOM](#) 指令。

例程：设置 variable=3，将一个 VR_INT 映射到两个 MODBUS_REG 地址，换算关系：VR_INT(num)=MODBUS_REG(num)*2^16+MODBUS_REG(num+1)

```
SETCOM(38400,8,1,0,0,3)      '配置掉电存储
```

```
VR_INT(0)=0
```

```
MODBUS_REG(0)=1      '低 16 位值为 1
```

```
MODBUS_REG(1)=2      '高 16 位值为 2
```

```
?VR_INT(0)          '结果: 131073
END
```

3.2. 寄存器

机器人控制器寄存器主要有 TABLE、FLASH、VR、MODBUS 寄存器。

3.2.1. TABLE

[TABLE](#) 是控制器自带的一个超大数组，数据类型为 32 位浮点型（4 系列及以上为 64 位浮点数），掉电不保存。编写程序时，TABLE 数组不需要再定义，可直接使用，索引下标从 0 开始。

XBasic 的某些指令可以直接读取 TABLE 内的值作为参数，比如 CAM, CAMBOX, CONNFRAME, CONNREFRAME, MOVE_TURNABS, B_SPLINE, CAN, CRC16, DTSMOOTH, PITCHSET, HW_PSWITCH 等指令。

1) [TABLE](#) 指令读写数据。

TABLE(0) = 10 'TABLE(0)赋值 10

TABLE(10,100,200,300) '批量赋值, TABLE(10)赋值 100, TABLE(11)赋值 200, TABLE(12)赋值 300

2) [TSIZE](#) 指令可读取 TABLE 空间大小，还可修改 TABLE 空间大小（不能超出 TABLE 最大空间）。

PRINT TSIZE '打印出控制器 TABLE 大小

TSIZE=10000 '设置 TABLE 的大小，不能超过控制器 TABLE 最大 SIZE

3) [TABLESTRING](#) 指令按照字符串格式打印 TABLE 里的数据。

TABLE(100,68,58,92)

PRINT TABLESTRING(100,3) '字符串格式打印数据，转换为 ASCII 码

打印结果: D:\

TABLE 作为参数传递时用法大致相同，以 CAM 凸轮指令为例：

CAM(start point, end point, table multiplier, distance)

start point: 起始点 TABLE 编号，存储第一个点的位置

end point: 结束点 TABLE 编号

table multiplier: 位置乘以这个比例，一般设为脉冲当量值

distance: 参考运动的距离

使用方法示例：

TABLE(10,0,80,75,40,50,20,50,0) 'TABLE 从 10 开始存数据, TABLE(10)赋值 0, TABLE(11)赋值 80

CAM(10,17,100,500) '运动轨迹为 TABLE(10)到 TABLE(17)

查看 TABLE 内数据的方式有 2 种：

第一种：在在线命令行输入?TABLE(10,8)查询 TABLE(10)开始，依次 8 个数据。

3.2.2. FLASH

严格来讲，FLASH 不是寄存器，但它与寄存器密切相关，所以放于此章叙述。

FLASH 具有掉电存储功能，读写次数限制为十万次，长期不上电也不会丢失数据。一般用于存放较大的，不需要频繁读写的数据，比如加工的工艺文件。

读与写时要注意保证要操作的变量，数组等名称和次序高度一致，如果不一致会导致数据错乱。

FLASH 使用时是按块编号，块数 [FLASH_SECTES](#) 指令查看，不同的控制器 FLASH 块数与块数据大小都不同，每块数据大小 [FLASH_SECTSIZE](#) 指令查看。

CAN 通讯设置的参数，IP 地址、APP_PASS、LOCK 密码等系统参数存储到 FLASH。

注意：FLASH 在读取之前先要写入，否则会提示警报 WARN。

FLASH 使用方法：

GLOBAL VAR '变量定义

GLOBAL ARRAY1(200) '数组定义

DIM ARRAY2(100)

'数据存储到 FLASH 块：把 VAR，ARRAY1，ARRAY2 数据依次写入 FLASH 块 1

FLASH_WRITE 1, VAR, ARRAY1, ARRAY2

'FLASH 块数据读取：把 FLASH 块 1 的数据依次读入 VAR，ARRAY1，ARRAY2

FLASH_READ 1, VAR, ARRAY1, ARRAY2 '读取次序与写入次序一致

3.2.3. VR

[VR](#) 寄存器具有掉电存储功能，可无限次读写，但数据空间较小，一般只有 1024 或者更少，最新系列机器人控制器的 VR 空间为 4000，用于保存需要不断修改的数据，例如轴参数、坐标等，数据类型为 32 位浮点型（4 系列及以上为 64 位浮点数）。

可使用 [VR_INT](#) 强制保存为整型，[VRSTRING](#) 强制保存为字符串。VR、VR_INT、VRSTRING 共用一个空间，地址空间是重叠的，VR 和 VR_INT 读写方法相同，VRSTRING 保存 ASCII 码，一个字符占用一个 VR。

VR 的掉电保存原理是控制器内部有断电存储器，但数据容量较小，所以数据量较大的或需要长久保存的数据最好写到 FLASH 块或导出到 U 盘。

例一：VR 使用方法

VR(0) = 10.58 '赋值

aaa = VR(0) '读取

例二：VR 寄存器数据相互转换

VR(100)=10.12

VR_INT(100)=VR(100) '数据转换

?VR_INT(100) '打印结果：10，浮点数转换成整数，丢失小数位

例三：VRSTRING 存储字符串

VRSTRING(0,4) = "abc" '从 VR(0)开始保存字符串

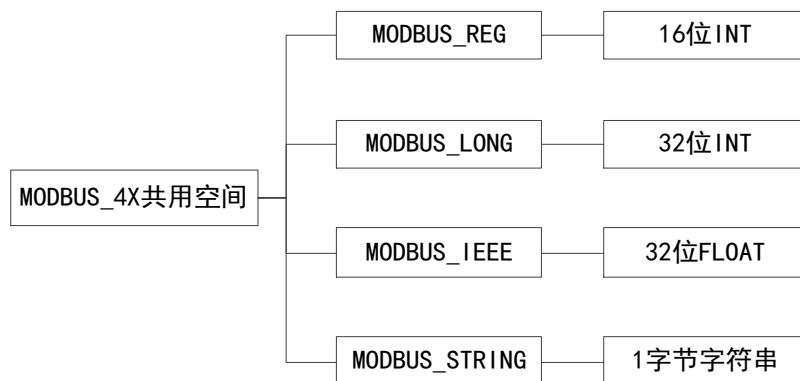
PRINT VRSTRING(0,4) '打印结果：abc

3.2.4. MODBUS

MODBUS 寄存器符合 MODBUS 标准通讯协议，分为位寄存器和字寄存器两类。MODBUS 寄存器的数据掉电不保存。

位寄存器：[MODBUS_BIT](#)，触摸屏一般称为 MODBUS_0X，布尔型。

字寄存器：[MODBUS_REG](#)、[MODBUS_LONG](#)、[MODBUS_IEEE](#)、[MODBUS_STRING](#)，触摸屏一般叫 MODBUS_4X，类型如下图。



控制器中 MODBUS 字寄存器占用同一个变量空间，其中一个 LONG 占用两个 REG 地址，一个 IEEE 也占用两个 REG 地址，使用时要注意错开字寄存器编号地址。

MODBUS_LONG(0)占用 MODBUS_REG(0)与 MODBUS_REG(1)两个 REG 地址。

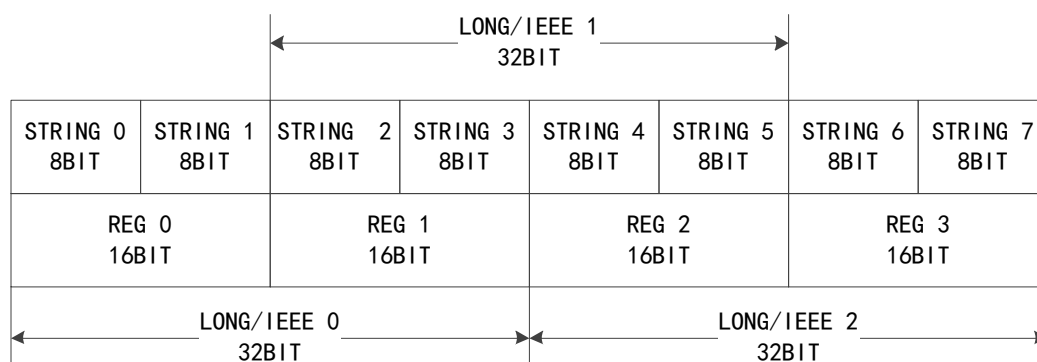
MODBUS_LONG(1)占用 MODBUS_REG(1)与 MODBUS_REG(2)两个 REG 地址。

MODBUS_IEEE(0)占用 MODBUS_REG(0)与 MODBUS_REG(1)两个 REG 地址。

MODBUS_IEEE(1)占用 MODBUS_REG(1)与 MODBUS_REG(2)两个 REG 地址。

所以要注意 MODBUS_REG,MODBUS_LONG,MODBUS_IEEE 地址在用户应用程序中不能重叠。

4X 空间示意图：



例程：

MODBUS_REG(0)=0 '初始化置 0

MODBUS_REG(1)=0 '初始化置 0

MODBUS_LONG(0)=70000 'modbus_long 赋值 70000， modbus_reg 范围-32768~32767

?MODBUS_REG(0),MODBUS_REG(1)

'打印出 reg(0)为 4464， reg(1)为 1， long(0)=reg(1)*2^16+reg(0)

在串口设置(SETCOM 参数)过程中,寄存器选择为 VR 时,此时一个 VR 映射到一个 MODBUS_REG,其中 VR 是 32 位浮点型,MODBUS_REG 是 16 位有符号整数型,从 VR 传递数据给 MODBUS_REG 会丢失小数部分,当 VR 数据超过正负 15 位时,MODBUS_REG 数据会改变;MODBUS_REG 传递数据给 VR 不会有问题,见如下例程,更多信息参见 SETCOM 指令。

例程:

```
VR(0)=0           '初始化 VR(0)和 MODBUS_REG(0)为 0
MODBUS_REG(0)=0
SETCOM(38400, 8,1,0,0,4,0)  '设置 VR 映射到 MODBUS_REG
VR(0)=100.345       '设置 VR(0)=100.345
?MODBUS_REG(0)      '打印结果为 100, VR 已经映射到 REG, 但是 REG 是整型, 所以小数
部分丢失
MODBUS_REG(0)=200    'REG(0)设为 200
?VR(0)              '打印结果为 200, REG 变化也会改变 VR
```

当使用 MODBUS 协议与其他设备通讯时,就需要将数据放在 MODBUS 寄存器内进行传递,比如与触摸屏通讯。不进行 MODBUS 通讯时,亦可将 MODBUS 寄存器作为控制器本地数组使用。

控制器直接从 MODBUS_BIT 地址 10000 开始与输入 IN 口对应,20000 与输出 OUT 口对应(注意读取的 IO 是原始的状态,INVERT_IN 反转输入指令不起作用),30000 与 PLC 编程的 S 寄存器对应。

MODBUS_IEEE 地址 10000 开始对应轴 DPOS 区间,11000 开始对应轴 MPOS 区间,12000 开始对应轴 VP_SPEED 区间;MODBUS_REG 的 13000 开始对应模拟量 DA 输出区间,14000 开始对应模拟量 AD 输入区间。

MODBUS_BIT 地址	意义
0~3999	用户自定义使用
4000~9000	系统内部使用
10000~14095	对应输入 IN 口
20000~24095	对应输出 OUT 口

MODBUS 字寄存器地址	意义
0~3999	用户自定义使用,可混用 MODBUS_REG、MODBUS_IEEE、MODBUS_LONG
4000~12999	系统内部使用
13000~13127	模拟量输出 AOUT,读写用 MODBUS_REG
14000~14255	模拟量输入 AIN,读用 MODBUS_REG

3.3. 多任务编程

3.3.1. 多任务概念

任务是执行 I/O 刷新和用户程序等一系列指令处理的功能,一个任务是指一个正在运行的程序。当前版

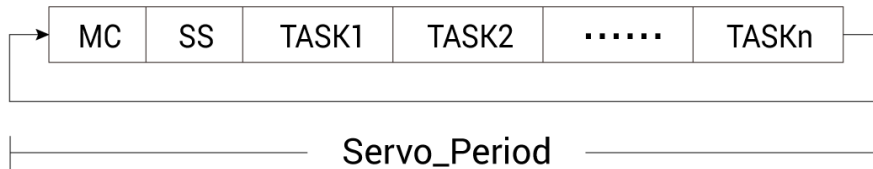
本控制器用户最多可创建四个任务同步并行。

如果多个程序模块能够互不干扰的同时运行，则称为多任务，多任务编程在 XCStudio 软件上实现。

多任务可以将复杂的程序分成几个部分，分别开任务来同时执行，每个部分的任务是独立的，这样就可以使设备的复杂运动过程变得简单明了，编程更灵活，没有多任务的场合程序只能顺序执行，使得程序的执行效率十分低下。

机器人控制器支持多任务编程，每个任务都有自己唯一的编号，此编号没有优先级意义，只是标识当前程序属于哪一个任务，其中主任务默认是 `main.xc` 且不可更改。

每个运动控制周期(Servo Period)包含 MC、SS、以及用户多任务程序的运行，如下图所示：



MC: Motion Control、EtherCAT 通讯、中断的实现。Motion Control 包含：多轴插补运动、机械手正反转算法；EtherCAT 通讯包含 PDO 通讯与 SDO 通讯。

SS: System Service，包含 RS232 串口通讯，RS485 串口通讯，CAN 通讯，EtherNET 通讯（MODBUS RTU 主从通讯以及 XCStudio 相关软件服务）

TASK1、…、TASKn: 对应于各个任务的运行，第 1 个任务到第 n 个任务。

在一个控制周期内，不同的任务根据当前执行的指令的差异，任务占用的时间也会有差异，并不完全相同，任务在默认情形下不存在优先级。

Basic 的所有任务只扫描运行一次（除非程序内有死循环才会一直运行）。一个工程项目下 Basic 文件支持同时存在多个自动运行任务。

控制器同时处理四个任务，任务之间是并行的，互不干扰，控制器下载程序之后这四个文件任务同时启动，同时还能在文件任务执行的时候，使用任务指令开启 SUB 子程序任务或标记任务，SUB 子程序任务或标记任务一旦开启，便与主程序无关，任务运行停止后可重复触发任务执行。

控制器多任务的优势：

程序模块化：用户可以将程序编写成多个较小的、特定的程序，来实现客户设备指定的功能。

并发性：每个任务可以独立运行，任务开启后，不受其他任务的影响。

简化错误处理：划分多任务运行后错误处理变得简单，只需处理出错的任务。

命令交互：程序处于运行状态时，用户也可以随时进行命令交互，如在线修改运动参数，在线命令栏发送指令等，其他程序不受影响。

3.3.2. 多任务状态查看

任务状态有三种，正在运行、停止和暂停，任务状态查看有如下 3 种方式。

1. 任务指令查看

PROC_STATUS: 任务状态查看，只读参数。返回值：0-任务停止，1-任务正在运行，3-任务暂停中。

示例：

PRINT PROC_STATUS(1) '打印任务 1 状态

3.3.3. 多任务启动与停止

1. 多任务操作指令

多任务主要操作指令如下：

END：当前任务正常结束

STOP：停止指定文件运行的任务

STOPTASK：停止指定任务

RUN：启动新任务运行一个文件

RUNTASK：启动新任务运行一个 SUB 或者运行一个带标签的程序

PAUSETASK：暂停指定任务

RESUMETASK：恢复指定任务，恢复后任务从停止处继续往下执行

2. 多任务启动

任务启动有三种方式，分别是自动运行任务号设置、RUN 指令和 RUNTASK 指令，使用指令开启任务时，程序扫描执行到该指令后再开启任务。

开启任务时注意任务编号的填写，任务不能重复开启。

1) RUN 指令将文件作为一个任务启动。

示例：RUN "TuXing_001.bas",2 '将 TuXing_001.bas 文件作为任务 2 启动

2) RUNTASK 指令将 SUB 子程序或带标签程序作为一个任务启动。可跨文件开启全局定义的 SUB 子程序，要开启任务的标签程序只能存在本文件内。

示例：RUNTASK 1,task_home '以任务 1 启动 task_home 子程序

3. 多任务停止

停止任务指令有 STOPTASK，STOP。

任务停止再启动就会从头执行任务。

开启任务时，一般先使用 STOPTASK 停止任务，再 RUNTASK 开启，避免任务出现重复开启报错。

STOPTASK 支持停止文件任务、SUB 子程序任务和带标签的任务。

示例：STOPTASK 2 '停止任务 2

STOP 指令支持停止 Basic 文件任务，推荐使用 STOPTASK 指令，操作更简单。

3.3.4. 任务暂停与恢复

暂停任务使用 PAUSETASK 指令，恢复任务使用 RESUMETASK 指令。恢复后任务从暂停处继续向下

执行。暂停的任务支持停止。

1) PAUSETASK: 暂停指定任务

示例: PAUSETASK 1 '暂停任务 1

2) RESUMETASK: 恢复指定任务

示例: RESUMETASK 1 '继续运行任务 1

示例: 项目内有两个任务, 下载运行后, 任务 0 和任务 1 正在运行中。

3.4. 三种中断类型

XBasic 中断分为三种, 分别为掉电中断、外部中断、定时器中断。

使用中断前必须开启中断总开关, 为了避免程序没有初始化好进入中断, 控制器上电时中断开关缺省是关闭的。

三类中断运行时, 中断程序单独占用一个任务号运行, 不存在压栈的情况。

中断使用注意事项:

各中断之间无优先级, 支持中断嵌套, 多个中断可以同时执行, 同一时间处理的中断函数不宜过多。

控制器内部只有一个任务在处理所有的中断信号响应, 有一个固定的中断任务号, 如果中断处理函数过多, 并且中断处理函数的代码太长, 会造成所有的中断响应变慢, 甚至是中断堵塞, 影响其他中断执行。

解决办法:

尽量减少中断的数量, 很多应用都可以用循环扫描来处理。

如果有一个中断处理函数特别长的话, 调用一个单独的任务来处理中断中的复杂任务, 这样就不会堵塞其他的中断响应。

3.4.1. 掉电中断

必须是全局的 SUB 函数。控制器只有 1 个掉电中断。掉电中断执行的时间特别有限, 只能写少数几条语句, 将数据存储在 VR 里。

相关函数: [INT_ENABLE](#), [ONPOWEROFF](#)。

示例:

```
INT_ENABLE = 1
DPOS(0)=VR(0)           '上电读取保存的数值, 恢复坐标
DPOS(1)=VR(1)
DPOS(2)=VR(2)
END                      '主程序结束

GLOBAL SUB ONPOWEROFF () '掉电中断
    VR(0) = DPOS(0)      '保存坐标到 VR
    VR(1) = DPOS(1)
```

```
VR(2) = DPOS(2)
END SUB
```

3.4.2. 外部中断

可设置上升沿触发或下降沿触发，必须是全局的 SUB 函数，目前只有中断 IN 口 0-31 才可以使用。

相关函数：[INT_ONn](#)，[INT_OFFn](#)。

示例：

```
INT_ENABLE=1          '开启中断
END                   '主程序结束
```

```
GLOBAL SUB INT_ON0 () '外部上升沿中断程序
    PRINT "输入 IN0 上升沿触发"
END SUB
```

```
GLOBAL SUB INT_OFF0 () '外部下降沿中断程序
    PRINT "输入 IN0 下降沿触发"
END SUB
```

3.4.3. 定时器中断

达到设定时间后执行的功能，必须是全局的 SUB 函数定时器中断支持同时开启多个，个数由定时器个数决定，定时器个数根据控制器型号，使用?*max 打印查看。

相关函数：[ONTIMERn](#)。

示例：

```
INT_ENABLE=1          '开启中断
TIMER_START(0,100)    '定时器 0 开启，100ms 后执行一次
END                   '主程序结束
```

```
GLOBAL SUB ONTIMER0() '中断程序
    PRINT "ontimer0 enter"
    'TIMER_START(0,100) '希望周期执行中断，在 SUB 里再次打开定时器
END SUB
```

第四章 通讯方式

4.1. 串口通讯

4.1.1. 串口类型

控制器包含三类串口，RS232、RS485 和 RS422，其中 RS232 串口所有控制器都包含，绝大部分控制器都包含 RS485 串口，只有少数型号控制器有 RS422 串口。

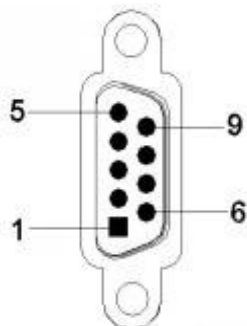
控制器的串口协议均为 MODBUS_RTU，默认做从端，RS232 和 RS485 可通过 SETCOM 指令配置成主端，通讯速率等参数同样也是通过 SETCOM 指令配置，。

控制器串口默认参数：波特率 38400、数据位 8、停止位 1、校验位无，掉电不保存。

1. RS232 串口

控制器的 RS232 接口可以做 MODBUS 主站或从站，支持 1 个主站发送数据，1 个从站接收数据。做主站时，可连接驱动器、变频器、温控仪等，进行数据读出与写入的控制。做从站时，可连接人机界面，用来监控运行状态，常用于连接 PC 或人机界面。

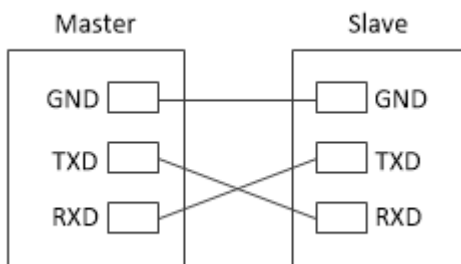
RS232 控制器采用 DB-9 接口，针脚信号说明如下：



针脚号	名称	说明
2	RXD	接收数据引脚
3	TXD	发送数据引脚
5	EGND	电源地
9	E5V	外部电源 5V 输出

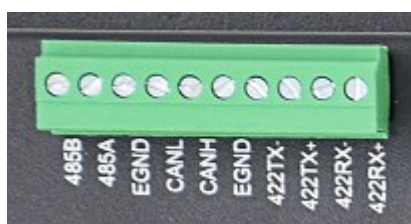
RS232 的标准接线只需要三根线即可，2 根数据信号 TXD 和 RXD，1 根地线 GND，数据信号 TXD 与 RXD 交叉连接，再将 GND 连到一起。

接线参考如下：



2. RS485 串口

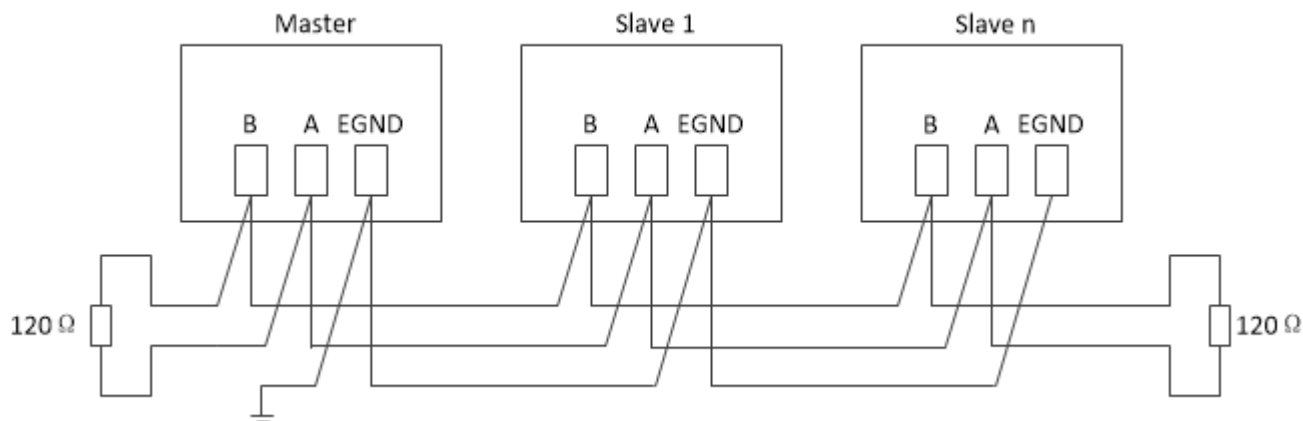
主要提供主/从站的多台通讯设备联机，理论上支持 128 个节点，一主多从。做主站时，可连接驱动器、变频器、温控仪等，进行数据读出与写入的控制；做从站时，能与 PLC 通讯，可连接人机界面，用来监控运行状态。



RS485 接口采用差分传输方式，通过判断 A 与 B 之间的电压差来确定是高电平或低电平。

针脚名称	说明
485B	485-
485A	485+
EGND	电源地

控制器的 RS485 接口采用了简易接线方式，如下图所示，控制器的 485A、485B、GND 地线，分别接第一个从站的 A、B、地线，然后再接第二个从站的 A、B、地线(A 接 A，B 接 B，信号共地)，并且控制器和最后一个从站的 485A 和 485B 要并联 120 欧电阻防止信号反射，线缆需要使用屏蔽双绞线，避免信号干扰，每个节点支线的距离要小于 3m。



3. RS422 串口

RS422 数据传输特性与 485 相同。RS422 采用四线制，分别标示为 RX+/RX-（接收信号），RT+/RT-

（发送信号），一根信号地线，共 5 根线，四线接口由于采用单独的发送和接收通道，因此不必控制数据方向。

引脚名称	说明
422TX-	发送数据-
422TX+	发送数据+
422RX-	接受数据-
422RX+	接受数据+
EGND	电源地

控制器的 RS422 接口采用了简易接线方式，但相比 RS485 和 RS232，布线成本高，接线容易搞错。控制器的 RS422 接口仅支持接入一个设备。

4.1.2. 串口连接方法

串口支持 MODBUS 通讯协议 RTU 模式，常用于连接电脑或触摸屏，通讯时注意串口的参数要匹配，不管哪种串口，除了端口号和接线方法有所不同，默认参数与操作指令都是相同的。

控制器串口默认参数：波特率 38400，数据位 8，停止位 1，校验位无，若串口连接失败检查串口号是否正确，修改电脑的通讯端口 COM 的配置，使其与控制器的默认参数一致。

串口参数设置均使用 SETCOM 指令，串口参数是掉电不保存的，控制器重新上电后，SETCOM 参数会还原成默认值，所以请在程序开头写 SETCOM 设置。

串口默认为 MODBUS 从端，可修改 SETCOM 指令的 MODE=14 设置为主端，或 MODE=0 开启串口自定义通讯，即无协议模式，串口自定义通讯模式下使用 GET #指令从自定义串口通道里读取数据，PRITNT #指令从自定义串口通道里输出字符串，PUTCHAR #发指令从自定义串口通道里输出字符（ASCII 码）。

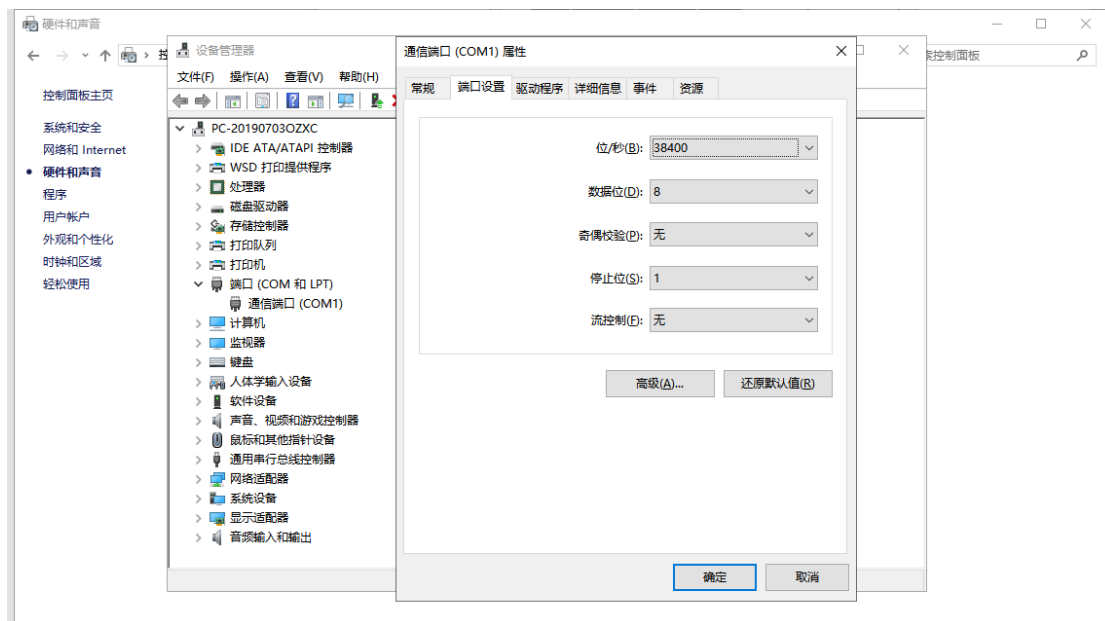
SETCOM 指令 mode 参数配置协议：

Mode 值	描述
0	RAW 数据模式，无协议，此时可以使用 GET #读取；PRITNT #、PUTCHAR #发送
4（缺省）	MODBUS 从端（16 位整数）
14	MODBUS 主端（16 位整数）
15	直接命令执行模式，此时可以直接从串口输入字符串命令(换行符结束)

。

若连接失败，按下面方法依次排查：

1. 查看串口连接线是否为交叉线。
2. “连接到控制器”里的 COM 口编号、参数是否选择正确。
3. 打开电脑“设备管理器”-“端口”-“通信端口（COM）”-“端口设置”，查看 COM 口设置是否正确，控制器串口默认参数：波特率 38400，数据位 8，停止位 1，校验位无。



在“端口设置” - “高级”选项中可更改 com 端口号，通过下拉列表选择。



4. 当通过串口连接到控制器时，对应的控制器串口必须配置为 MODBUS 从协议模式（缺省模式），断电重启即可恢复。
5. COM 口是否已被其他程序占用，如串口调试助手等。
6. PC 端是否有足够的串口硬件。
7. 更换串口线/电脑测试。

4.2. 网口通讯

控制器网口为 EtherNET 接口，支持 MODBUS_TCP 通讯协议，常用于连接电脑或触摸屏，控制器一般有一个 EtherNET 接口，但是底层至少有两个网口通道，当需要使用网口连接多个设备的时候，可借助交换机。



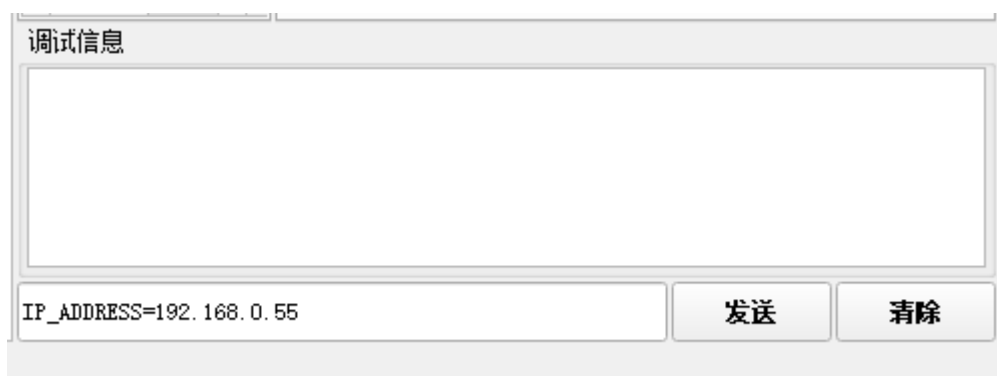
网口连接是需要保证控制器 IP 地址与电脑的 IP 地址处于同一网段，即四段 IP 地址的前三段相同，最后一段不同，否则会连接失败，连接失败时修改控制器 IP 或者电脑 IP 其中之一即可。

控制器出厂的 IP 是 192.168.0.11，IP 一经修改永久保存。

控制器网口也支持自定义通讯，使用 OPEN #指令打开自定义网口通讯，OPEN #指令支持配置网口通讯主从端，GET #指令从自定义网口通道里读取数据，PRITNT #指令从自定义网口通道里输出字符串，PUTCHAR #发指令从自定义网口通道里输出字符（ASCII 码）。

通过 IP_ADDRESS 指令发送在线命令修改。

指令发送修改成功之后自动断开连接，在线命令打印控制器连接错误信息，通过网口连接选择新 IP 地址 192.168.0.23 再次连接控制器，IP 地址修改成功后永久有效。



修改电脑 IP 地址

查看电脑本地 IP 协议版本 4 地址是否为 192.168.0.xxx，前三段与控制器一致，最后一段不能一样，控制器出厂默认 IP 192.168.0.11。如果 IP 地址的第三段不一样，则需要把对应的子网掩码改为 0。

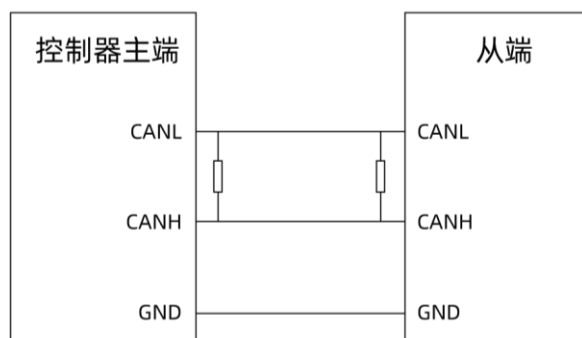
设置好之后，再次打开“连接到控制器”窗口尝试连接到控制器。



4.3. CAN 总线通讯

4.3.1. CAN 接线

控制器的 CAN 总线接口用于接 ZCAN 扩展模块，或连接控制器。CAN 总线连接控制器接线方法与连接 ZCAN 扩展模块相同，不同之处在于板块上没有集成 120 电阻，需要在 CANL 和 CANH 的首尾两端各接一个 120 欧姆电阻，CAN 总线连接扩展模块接线方法参见“[ZCAN 扩展模块](#)”章节。



控制器使用 CAN 总线连接时，此时控制器之间可以使用 CAN 指令通讯，数据通过 TABLE 传递。

控制器缺省做 CAN 通讯的主端，控制器之间的 CAN 通讯需要把一个控制器配置为从端，使用 CANIO_ADDRESS 指令配置主从端，CANIO_ADDRESS=32 配置为主端，CANIO_ADDRESS=其他值配置

为从端。

指令语法: CAN(channel, function, tablenum)

channel: CAN 通道, 0 表示第一个通道, -1 表示缺省通道

function: 功能号 (参见下表, 模式 6/7 适用标准帧, 模式 16/17 适用扩展帧)

tablenum: 数据存储的 TABLE 位置

值	描述
6	接收, 没有数据时, identifier<0
7	发送
16 (需要升级固件)	带扩展支持接收, 没有数据时, identifier<0
17 (需要升级固件)	发送扩展数据, 普通数据使用 7 发送

例子:

'发送端: 第一个控制器

TABLE(0,1,8,1,2,3,4,5,6,7,8) '发送 cobid=1, 8 个字节, 依次为 1-8

CAN(0,7,0) '发送数据

'接收端: 第二个控制器

CANIO_ADDRESS=1 '设置为 CAN 从端, 此参数设置一次即可

CAN(0,6,0) '接收数据

?TABLE(0)

4.4. U 盘接口

机器人控制器都带有一个标准 U 盘接口。

没有控制器的场合, 可以在 XCStudio 根目录新建 udisk 文件夹模拟 U 盘使用, 连接到仿真器, 调试 U 盘指令。

U 盘接口主要有三方面的用途:

1. 加载三次文件

使用 FILE 指令加载 U 盘里保存的三次文件执行。

示例:

```
IF U_STATE=TRUE THEN '判断 U 盘是否插入
  FILE "FIND_FIRST",".Z3P",800 '查找 Z3P 文件
  ?"文件名: "VRSTRING(800,20),"等待下载"
  FILE "COPY_TO",VRSTRING(800,20),VRSTRING(800,20) '下载 Z3P 文件
  ?"下载 Z3P 文件完成"
ENDIF
```

2. U 盘与寄存器数据交互

U 盘支持读写变量和数组。

FLASH 数据拷贝: 多个控制器中 FLASH 存储的数据可以通过 U 盘来相互传递。

VR 寄存器、TABLE 寄存器与 U 盘里的数据互相传递。

读写文件类型为 SD(filename).BIN 或 SD(filename).CSV, 不同的指令可操作的文件类型有所区别。

不同型号的控制器的 U 盘接口的使用方法都是相同的，将 U 盘插在控制器上的 UDISK 端口即可，控制器上电后有 U 盘插入时，U 盘指示灯亮。

在对 U 盘进行操作之前，先使用 U_STATE 指令判断 U 盘的状态，确保 U 盘能成功通讯，再使用 U 盘相关指令操作。

示例：

```
DIM a,array1(2) '变量、数组定义
a=123
array1(0)=10
array1(1)=20

IF U_STATE = TRUE THEN '判断 U 盘是否插入
    U_WRITE 0,a,array1 '将变量、数组写入 U 盘的 SD0 文件
    a=456
    array1(0)=11
    U_READ 0,a,array1 '读取 U 盘文件 SD0 数据
    PRINT a,array1(0) '结果： 123, 10
    IF a <> 123 THEN '判断 U 盘读写是否成功
        PRINT "U 盘读取错误"
    ELSE
        PRINT "U 盘成功读取"
    ENDIF
ELSE
    PRINT "U 盘未插入"
ENDIF
END
```

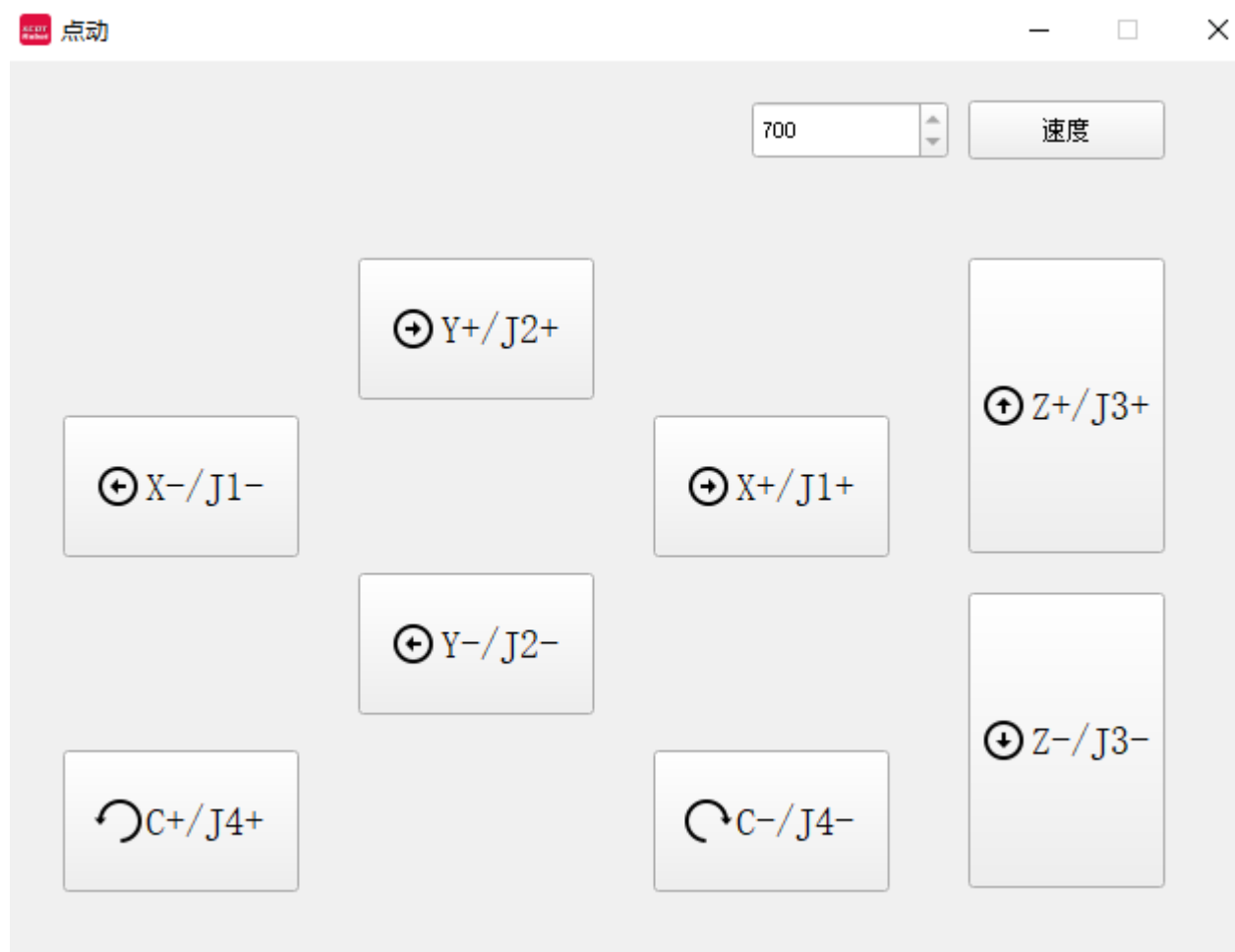
erCAT 相同，参见上节 EtherCAT 的说明。

第五章 运动控制功能

5.1. 常见运动模式

程序运行分为 手动、自动两种，其中手动模式下可以手动点动。
自动模式下可以走直线插补运动、圆弧运动等等。

5.1.1. 单轴点动



5.2. 插补运动

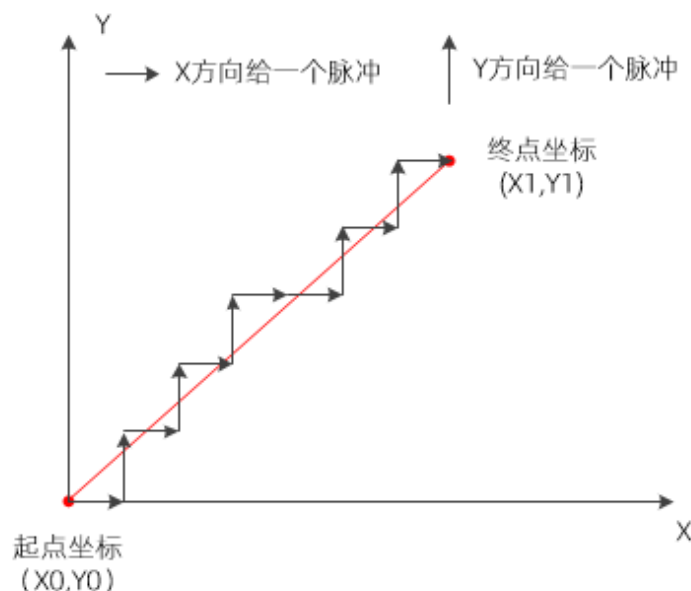
5.2.1. 插补概念

1. 直线插补原理

直线插补方式中，两点间的插补沿着直线的点群来逼近。首先假设在实际轮廓起始点处沿 X 方向走一小段（给一个脉冲当量轴走一段固定距离），发现终点在实际轮廓的下方，则下一条线段沿 Y 方向走一小段，此时如果线段终点还在实际轮廓下方，则继续沿 Y 方向走一小段，直到在实际轮廓上方以后，再向 X 方向走一小段，依次循环类推，直到到达轮廓终点为止。这样实际轮廓是由一段段的折线拼接而成，虽然是折线，但每一段插补线段在精度允许范围内非常小，那么此段折线还是可以近似看做一条直线段，这就是直线插补。

新川控制器采用硬件插补，插补精度在一个脉冲内，所以轨迹放大依然平滑。

假设轴需要在 XY 平面上从点(X0,Y0)运动到点(X1,Y1)，其直线插补的加工过程如下图所示。



2. 圆弧插补原理

圆弧插补与直线插补类似，给出两端点间的插补数字信息，以一定的算法计算出逼近实际圆弧的点群，控制轴沿这些点运动，加工出圆弧曲线。圆弧插补可以是平面圆弧（至少两个轴），还可以是空间圆弧（至少三个轴）。假设轴需要在 XY 平面第一象限走一段逆圆弧，圆心为起点，其圆弧插补的加工过程如下图所示。

控制器的空间圆弧插补功能是根据当前点和圆弧指令参数设置的终点和中间点（或圆心），由三个点确定圆弧，并实现空间圆弧插补运动，坐标为三维坐标，至少需要三个轴分别沿 X 轴、Y 轴和 Z 轴运动。

3. 运动控制器的插补模式

运动控制器的插补运动模式具有以下功能：

- 1) 可以实现直线插补、圆弧插补、空间圆弧插补、椭圆插补、螺旋插补等；
- 2) 可以在多个坐标系多通道进行多轴插补运动。
- 3) 每轴均有运动缓存区，可以实现运动的暂停、恢复等功能，停止插补运动的一个轴，其他轴跟着全部停止；
- 4) 具有缓存区延时和缓存区数字量同步输出的功能；
- 5) 具有预处理功能，控制器自行分析计算目标轨迹，能够实现小线段高速平滑的连续轨迹运动。

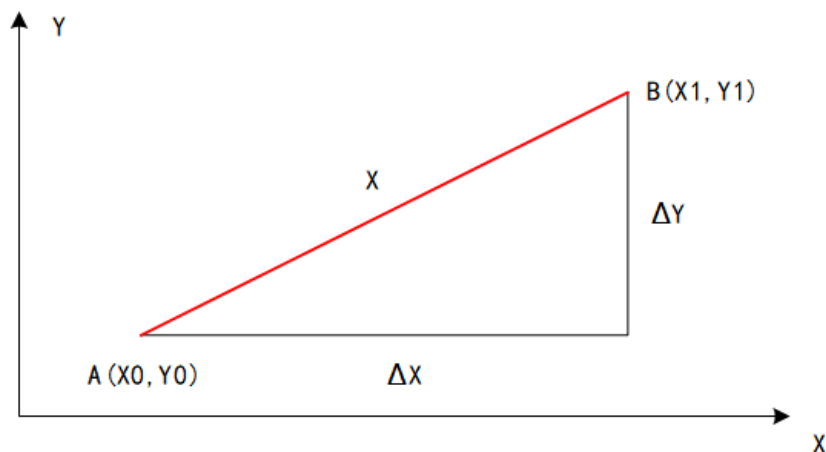
4. 二轴直线插补

轴 0 和轴 1 两轴参与直线插补运动，如下图，2 轴直线插补运动从 A 点运动到 B 点，XY 轴同时启动，并同时到达终点，设置轴 0 的运动距离为 ΔX ，轴 1 的运动距离为 ΔY ，主轴是 BASE 的第一个轴（此时主轴为轴 0），插补主轴运动速度为 S（主轴的设置速度），各个轴的实际速度为主轴的分速度，不等于 S，此时：

主轴运动距离： $X=[(\Delta X)^2+(\Delta Y)^2]^{\frac{1}{2}}$

轴 0 实际速度： $S_0=S \times \Delta X/X$

轴 1 实际速度： $S_1=S \times \Delta Y/X$



5. 三轴直线插补

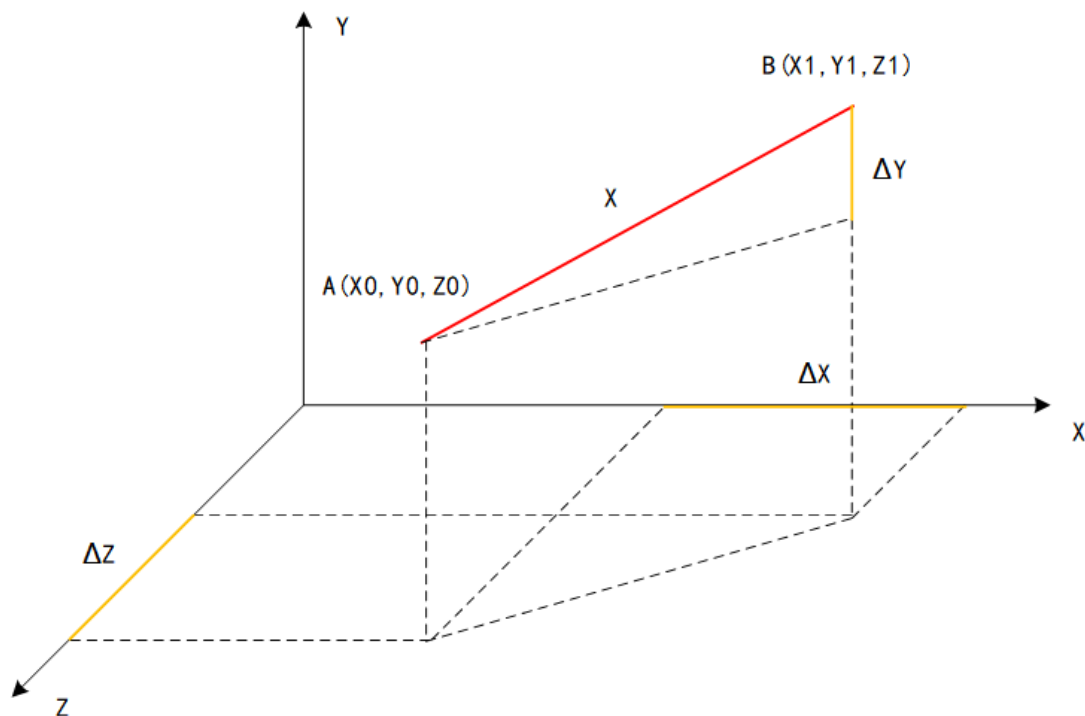
轴 0、轴 1 和轴 2 三轴参与直线插补运动，如下图，3 轴直线插补运动从 A 点运动到 B 点，XYZ 轴同时启动，并同时到达终点，设置轴 0 的运动距离为 ΔX ，轴 1 的运动距离为 ΔY ，轴 2 的运动距离为 ΔZ ，插补主轴轴 0 的运动速度为 S，各个轴的实际速度为主轴的分速度，不等于 S，此时：

主轴运动距离为 $X=[(\Delta X)^2+(\Delta Y)^2+(\Delta Z)^2]^{\frac{1}{2}}$

轴 0 实际速度： $S_0=S \times \Delta X/X$

轴 1 实际速度： $S_1=S \times \Delta Y/X$

轴 2 实际速度： $S_2=S \times \Delta Z/X$



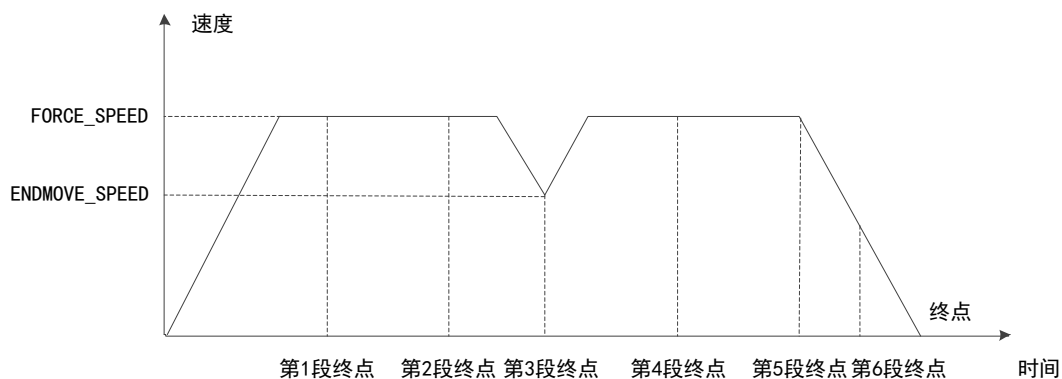
6. 多轴直线插补

多轴直线插补可以理解为轴的多个自由度，是在三维空间里的直线插补。以四轴插补为例，一般是三个轴在 XYZ 平面走直线，另一个轴为旋转轴，按照一定的比例关系做跟随运动。

5.2.2. 连续插补

不开启连续插补，上一条插补运动完成后，执行下一条插补时，会先减速停止，再重新加速执行插补运动，实际应用时这种情况会导致加工效率低下，所以需要使连续的插补运动之间不减速，这就是连续插补功能。

若要使插补动作连续，设置 ARCH(120)以后，相同主轴的插补运动会自动被连续起来，连续两段运动之间不减速，而且 SP 指令可以手动设置运动速度和结束速度。



在实际加工过程中，为追求加工效率会开启连续插补，运动轨迹的拐角处若不减速，当拐角较大时，会对机台造成较大冲击，影响加工精度。若关闭连续插补，使拐角处减速为 0，虽然保护了机台，但是加工效

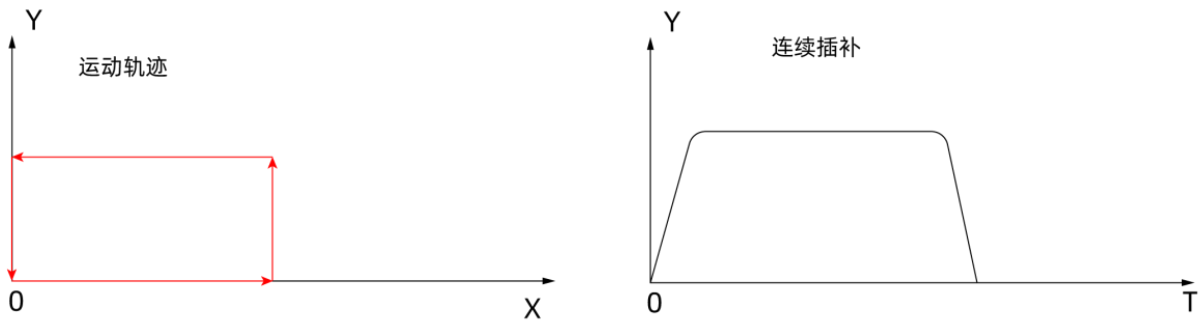
率受到了较大影响，所以提供了前瞻指令，使在拐角处自动判断是否将拐角速度降到一个合理的值，既不会影响加工精度又能提高加工的速度，这就是轨迹前瞻功能的作用。

运动控制器的轨迹前瞻可以根据用户的运动路径自动计算出平滑的速度规划，减少机台的冲击，从而提高加工精度。自动分析在运动缓冲区的指令轨迹将会出现的拐点，并依据用户设置的拐角条件，自动计算拐角处的运动速度，也会依据用户设定的最大加速度值计算速度规划，使任何加减速过程中的加减速都不超过 XACCEL 和 XDECEL 的值，防止对机械部分产生破坏冲击力。

使用轨迹前瞻和不使用轨迹前瞻的速度规划情况：

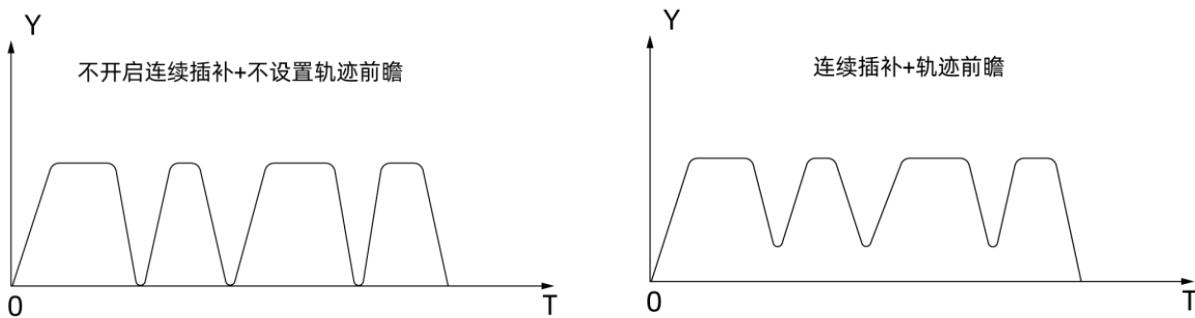
假设运动轨迹如下左图，走一个长方形轨迹，分为四段直线插补运动。

模式一：开启了连续插补后，得出的主轴速度随时间变化的曲线如下右图，主轴的速度是连续的，轨迹拐角处仍不减速，高速运行时拐角处冲击较大。



模式二：模式一条件下，关闭连续插补，得出的主轴速度随时间变化的曲线如下左图，每走完一段直线后便减速到 0 再开始第二段直线运动，加工效率不高。

模式三：模式一条件下，开启连续插补，并设置了轨迹前瞻参数，得出的主轴速度随时间变化的曲线如下右图，拐角处按照一定的比例减速，加工效率比模式二高。



以上模式若希望速度曲线更柔和，只需通过 SRAMP 指令设置速度为 S 曲线。

5.3. 原点回零

原点回归

轴	归零状态
原点	
J1	
J2	
J3	
J4	
J5	
J6	

模式 手动 设置 一键回零

手动模式下点击“一键回零”即可。

5.4. 精准输出

相关指令：

指令	说明	用法
MOVE_OP	缓冲中输出	MOVE_OP (编号,状态)
MOVEOP_DELAY	缓冲输出延时	可提前或延时输出

MOVE_OP 指令默认为普通输出，普通输出操作需要等待一个控制器周期才可执行，而精准输出操作，可在电机发出一个脉冲内响应，能大大提高工艺的精度，同时可以使用 MOVEOP_DELAY 指令调整响应时间（提前或延迟）。

5.5. 机械手

新川机器人控制器支持多种机械手算法，根据机械手的类型机械手连接后使用，能平滑、精准的控制机械手运动。

5.5.1. 机械手相关概念

1. 坐标系

1) 关节坐标系

每个轴相对原点位置的绝对角度，包含机械手所有关节，各关节之间相互独立，坐标单位为角度。操作其中一个关节时不影响其他关节。

2) 直角坐标系

世界坐标系：世界坐标系是被固定在空间上的标准直角坐标系，以机械手的底盘为坐标原点，其位置根据机械手类型确定。虚拟轴操作时就是根据世界坐标系运动，此时各关节会自动解算需要旋转的角度。

用户坐标系：用户对每个作业空间进行定义的直角坐标系，它用于位置寄存器的示教与执行，位置补偿命令的执行等，在没有定义的时候，将由世界坐标系来代替该坐标系。

机械手算法的主要目的是将关节坐标系与直角坐标系建立联系。

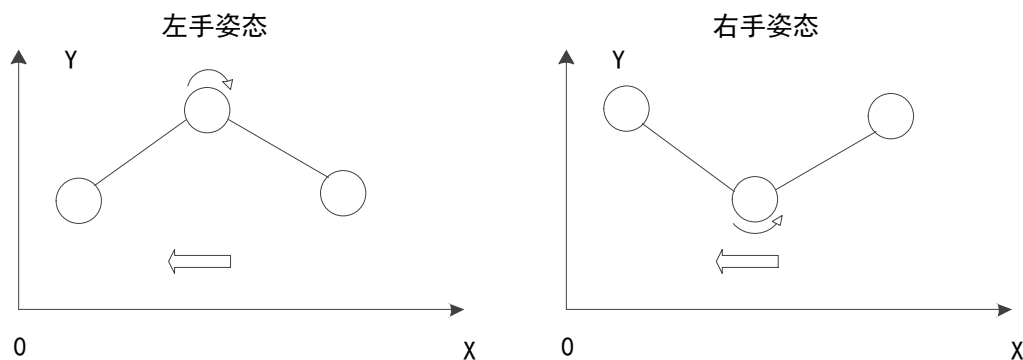
坐标系转换是指在描述同一个空间时，由原来的坐标系转换为另一个坐标系。机械手使用中，常用于确定工件坐标系。

工件坐标系是固定于工件上的笛卡尔坐标系，工件在坐标系相对于世界坐标系存在转换。每个机械手可以拥有若干工件坐标系，用来表示不同的工件，或者表示同一工件在不同的位置。

虚拟轴满足 XYZ 三轴的机械手类型支持此功能。

2. 姿态

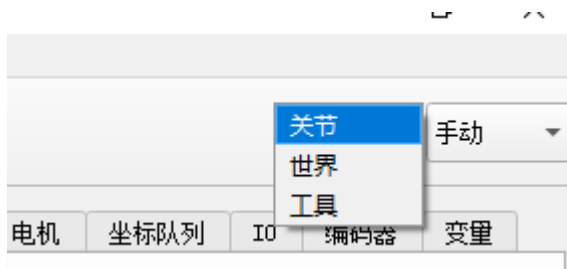
机械手姿态在数学上来说，是同一组虚拟轴数值有多组关节轴的解。即机械手在笛卡尔坐标系中运动到某一坐标点，可以有多种运动轨迹，这些运动轨迹就对应着不同姿态，如下图 SCARA 的两种姿态，在 X 方向运动，关节轴可以有两种运动方式。



3. 奇异点

逆解模式下，机械手运动到某一特定位置时，会失去某个自由度，该位置就叫做奇异点，实际使用过程中应避免运动到奇异点。例如当 SCARA 机械手完全伸直时，此时无法在 X 方向平动，需要操作往 X 负方向运动时，结构无法判断使用哪种姿态运动，此时机械手臂无法运动，出现此状况时，先使用正解模式调整关节轴位置，之后再切换到逆解模式使用。

5.5.2. 正解运动与逆解运动



关节：正解运动

世界：逆解运动

工具：逆解运动

DELTA 机器人一般使用世界（逆解运动）来定位物件。

第六章 运动指令

6.1. 常用运动指令

RAPIDSTOP -- 全部轴停止

类型	多轴运动指令		
描述	所有轴立即停止，如果轴参与插补，也停止插补运动。 RAPIDSTOP 后要调用绝对位置运动，必须先 WAIT IDLE 等待停止完成。		
语法	RAPIDSTOP (mode)		
	mode: 模式选择		
	0（缺省）	取消当前运动	
	1	取消缓冲的运动	
	2	取消当前运动和缓冲运动	
	3	立即中断脉冲发送	
适用控制器	通用		

MOVEABS_P() -- 直线运动到示教点

类型	多轴运动指令
描述	直线插补运动，绝对运动到 P 点。
语法	MOVEABS_P(nub) nub: 示教点标号
适用控制器	通用

XGET_VMOVE — 视觉静态运动

类型	多轴运动指令
描述	接收视觉发的坐标，并执行绝对直线运动。
语法	XGET_VMOVE(n1,n2,n3,n4) n1:X 轴增量偏移 n2:Y 轴增量偏移 n3:Z 轴增量偏移 n4:C 轴增量偏移 如无偏移则填 0，全部无偏移可留空
示例	XGET_VMOVE(0,0,30) ‘收到执行的坐标后将 Z 轴偏移 30 执行。 XGET_VMOVE() ‘所有轴无偏移，执行收到的坐标。

MOVE -- 直线运动

类型	多轴运动指令
描述	<p>直线插补运动，相对运动一段距离。</p> <p>插补运动时只有主轴速度参数有效，主轴是 BASE 的第一个轴，运动参照这个轴的参数。</p> $\text{插补运动距离 } X = \sqrt{X_0^2 + X_1^2 + X_2^2 + \dots + X_n^2}$ <p>运动时间 $T = X / \text{主轴 SPEED}$</p>
语法	MOVE(distance1 [,distance2 [,distance3 [,distance4...]]]) distance1: 第一个轴运动距离 distance2: 下一个轴运动距离
适用控制器	通用

XDMOVE – 同步跟踪运动

类型	多轴运动指令
描述	沿着 X 轴实时跟踪插补运动，跟随外部编码器绝对运动到指定坐标。
语法	<pre> XDMOVE(n1,n2,n3,n4) 'n1:跟踪时间 n2:Y 偏移，默认 0 n3:Z 偏移，默认 0 n4:C 偏移，默认 0 </pre>
示例	<p>【视觉拍照同步跟踪抓取】</p> <pre> BASE (X, Y, Z, C) GLOBAL MACC, MDEC MDEC = 2 MACC = 5 ARCH(160) '倒角 30 度' while 1 XACCEL(MACC) '加速度' XDECEL(MDEC) '减速度' MOVEABS_P(0) '直线运动' MOVE_OP (1, OFF) '运动停止关闭 Y1' MOVEABS_P(1) '直线运动' MOVEABS_P(0) '直线运动' WAIT_ALL() XFIND_POS() '寻找工件位置' XDEL_RANGE() '销毁超限制的坐标' XWAIT_ARR() '等待工件到达' XDMOVE(0, 0, 30, 0) '沿着 X 轴跟踪' 跟踪时间/Y/Z/C XDMOVE() '沿着 X 轴跟踪' Y/Z/C MOVE_OP (1, ON) XDMOVE(0, 0, 20, 0) '沿着 X 轴跟踪' Y/Z/C STOPXDMOVE_P(0) '停止跟踪并且运动到 P0' XDEL_CV() '销毁当前视觉坐标' WEND </pre>

MOVECIRC2ABS -- 三点画弧-绝对

类型	多轴运动指令
描述	<p>圆弧插补，三点画弧，绝对运动。</p> <p>BASE 第一轴和第二轴进行圆弧插补，绝对移动方式。自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。</p> <p>注意：不能用此指令进行整圆插补运动，整圆使用 MOVECIRC 相对圆弧，或连续使用两条此类指令。</p>
语法	<p>MOVECIRC2ABS(mid1, mid2, end1, end2)</p> <p>mid1: 中间点第一个轴坐标，绝对位置</p> <p>mid2: 中间点第二个轴坐标，绝对位置</p> <p>end1: 结束点第一个轴坐标，绝对位置</p> <p>end2: 结束点第二个轴坐标，绝对位置</p> <p>坐标位置请确保正确，否则实际运动轨迹会错误。</p>
适用控制器	通用
相关指令	MOVECIRC2 ， MOVECIRCABS ， *SP

MHELICAL2ABS -- 三点螺旋-绝对

类型	多轴运动指令						
描述	<p>螺旋插补，三点画弧，绝对运动。</p> <p>BASE 第一轴和第二轴进行圆弧插补，第三轴进行螺旋，绝对移动方式。</p> <p>自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。</p> <p>无法螺旋一圈，需要螺旋一圈请使用 MHELICAL 或 MHELICALABS。</p>						
语法	<p>MHELICAL2ABS(mid1, mid2, end1, end2, distance3,[mode])</p> <p>mid1: 中间点第一个轴坐标</p> <p>mid2: 中间点第二个轴坐标</p> <p>end1: 结束点第一个轴坐标</p> <p>end2: 结束点第二个轴坐标</p> <p>distance3: 第三个轴结束点坐标，勘误：20150306 以前的版本此参数有问题，建议使用 MHELICAL2 相对指令</p> <p>mode: 第三轴的速度计算</p> <table border="1"> <tr> <th>值</th><th>描述</th></tr> <tr> <td>0(缺省)</td><td>第三轴参与插补速度计算</td></tr> <tr> <td>1</td><td>第三轴不参与插补速度计算</td></tr> </table> <p>坐标位置请确保正确，否则实际运动轨迹会错误。</p>	值	描述	0(缺省)	第三轴参与插补速度计算	1	第三轴不参与插补速度计算
值	描述						
0(缺省)	第三轴参与插补速度计算						
1	第三轴不参与插补速度计算						
适用控制器	通用						
相关指令	MHELICAL2 ， MHELICALABS ， *SP						

MECLIPSE -- 椭圆

类型	多轴运动指令						
描述	<p>椭圆插补，中心画弧，相对运动，可选螺旋。</p> <p>BASE 第一轴和第二轴进行椭圆插补，相对移动方式，可选第三个轴同步螺旋。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。</p> <p>可画整椭圆。</p> <p>只能画长轴（短轴）与 X 平行或垂直的椭圆。</p>						
语法	<p>MECLIPSE (end1, end2, centre1, centre2, direction, adis, bdis[, end3])</p> <p>end1: 终点第一个轴运动坐标，相对于起始点 end2: 终点第二个轴运动坐标，相对于起始点 centre1: 中心第一个轴运动坐标，相对于起始点 centre2: 中心第二个轴运动坐标，相对于起始点 direction: 0-逆时针，1-顺时针</p> <table border="1"> <tr> <th>值</th><th>描述</th></tr> <tr> <td>0</td><td>逆时针</td></tr> <tr> <td>1</td><td>顺时针</td></tr> </table> <p>adis: 第一轴的椭圆半径，半长轴或者半短轴都可 bdis: 第二轴的椭圆半径，半长轴或者半短轴都可，ab 相等时自动为圆弧或螺旋 end3: 第三个轴的运动距离，需要螺旋时填入</p> <p>坐标位置请确保正确，否则实际运动轨迹会错误。</p>	值	描述	0	逆时针	1	顺时针
值	描述						
0	逆时针						
1	顺时针						
适用控制器	通用						
相关指令	MECLIPSEABS ， *SP						

MECLIPSEABS -- 椭圆-绝对

类型	多轴运动指令
描述	<p>椭圆插补，中心画弧，绝对运动，可选螺旋。</p> <p>BASE 第一轴和第二轴进行椭圆插补，绝对移动方式，可选第三个轴同步螺旋。 自定义速度的连续插补运动可以使用 SP 后缀的指令，见*SP 描述。</p> <p>可画整椭圆。</p>
语法	<p>MECLIPSEABS(end1, end2, centre1, centre2, direction, adis, bdis[, end3])</p> <p>end1: 终点第一个轴运动坐标 end2: 终点第二个轴运动坐标 centre1: 中心第一个轴运动坐标 centre2: 中心第二个轴运动坐标 direction: 0-逆时针，1-顺时针</p>

	值	描述
	0	逆时针
	1	顺时针
	<p>adis: 第一轴的椭圆半径, 半长轴或者半短轴都可</p> <p>bdis: 第二轴的椭圆半径, 半长轴或者半短轴都可, AB 相等时自动为圆弧或螺旋</p> <p>end3: 第三个轴的运动坐标, 需要螺旋时填入</p> <p>坐标位置请确保正确, 否则实际运动轨迹会错误。</p>	
适用控制器	通用	
相关指令	MECLIPSE , *SP	

MSPHERICAL -- 空间圆弧

类型	多轴运动指令										
描述	<p>空间圆弧插补运动, 相对移动方式, 可选螺旋。</p> <p>自定义速度的连续插补运动可以使用 SP 后缀的指令, 见*SP 描述。</p>										
语法	<p>MSPHERICAL(end1,end2,end3,centre1,centre2,centre3,mode[,distance4][,distance5])</p> <p>end1: 第 1 个轴运动距离参数 1</p> <p>end2: 第 2 个轴运动距离参数 1</p> <p>end3: 第 3 个轴运动距离参数 1</p> <p>centre1: 第 1 个轴运动距离参数 2</p> <p>centre2: 第 2 个轴运动距离参数 2</p> <p>centre3 : 第 3 个轴运动距离参数 2</p> <p>mode: 指定前面参数的意义</p> <table border="1"> <tr> <th>值</th><th>描述</th></tr> <tr> <td>0</td><td>当前点, 中间点, 终点三点定圆弧 距离参数 1 为终点距离, 距离参数 2 为中间点距离</td></tr> <tr> <td>1</td><td>当前点, 圆心, 终点定圆弧 走最短的圆弧 距离参数 1 为终点距离, 距离参数 2 为圆心的距离</td></tr> <tr> <td>2</td><td>当前点, 中间点, 终点三点定圆 距离参数 1 为终点距离, 距离参数 2 为中间点距离</td></tr> <tr> <td>3</td><td>当前点, 圆心, 终点定圆 先走最短的圆弧, 再继续走完整圆 距离参数 1 为终点距离, 距离参数 2 为圆心的距离</td></tr> </table> <p>distane4: 第四轴螺旋的功能, 指定第 4 轴的相对距离, 此轴不参与速度计算</p> <p>distane5: 第五轴螺旋的功能, 指定第 5 轴的相对距离, 此轴不参与速度计算</p> <p>坐标位置请确保正确, 否则实际运动轨迹会错误。</p>	值	描述	0	当前点, 中间点, 终点三点定圆弧 距离参数 1 为终点距离, 距离参数 2 为中间点距离	1	当前点, 圆心, 终点定圆弧 走最短的圆弧 距离参数 1 为终点距离, 距离参数 2 为圆心的距离	2	当前点, 中间点, 终点三点定圆 距离参数 1 为终点距离, 距离参数 2 为中间点距离	3	当前点, 圆心, 终点定圆 先走最短的圆弧, 再继续走完整圆 距离参数 1 为终点距离, 距离参数 2 为圆心的距离
值	描述										
0	当前点, 中间点, 终点三点定圆弧 距离参数 1 为终点距离, 距离参数 2 为中间点距离										
1	当前点, 圆心, 终点定圆弧 走最短的圆弧 距离参数 1 为终点距离, 距离参数 2 为圆心的距离										
2	当前点, 中间点, 终点三点定圆 距离参数 1 为终点距离, 距离参数 2 为中间点距离										
3	当前点, 圆心, 终点定圆 先走最短的圆弧, 再继续走完整圆 距离参数 1 为终点距离, 距离参数 2 为圆心的距离										
适用控制器	通用										

第七章 程序结构与流程指令

7.1. 程序符号

' -- 注释

类型	特殊字符
描述	后面全部是注释，直到下一行开始。
适用控制器	通用

_ -- 换行

类型	特殊字符
描述	后面换行继续。 在条件判断语句、内存存储、打印输出时尽量不要使用。
适用控制器	通用

: -- 标号

类型	语法结构
描述	用户过程标号，标号可以作为无参数的 SUB 过程来使用。
语法	Label 标号名称，不能与现有的关键词冲突。
适用控制器	通用
例子	GOTO label1 END '主程序结束 label1: '加：定义标号 END

7.2. 数据定义指令

CONST -- 常量定义

类型	语法指令
----	------

描述	定义符号表示的常数，从而避免直接用数值表示。
语法	<pre>CONST CVARNAME = value</pre> <p>CVARNAME: 常量名 value: 常量值</p>
适用控制器	通用
例子	<p>例一</p> <pre>CONST MAX_VALUE = 100000 '定义常量 TABLE(0)=MAX_VALUE 'table(0)赋值 10000</pre> <p>例二</p> <pre>GLOBAL CONST MAX_AXIS=6 '定义总轴数</pre>
相关指令	DIM

DIM -- 变量定义

类型	语法指令
描述	<p>定义文件模块变量，数组。</p> <p>当变量不定义就赋值时，会自动定义为文件模块变量。 文件模块变量只能在本程序文件内部使用。 数组可以作为字符串使用，一个元素表示一个字节。</p>
语法	<pre>DIM varname, arrayname(space)</pre> <p>varname: 变量名称 arrayname: 数组名称 space: 数组长度</p>
适用控制器	通用
例子	<pre>DIM ARRAY1(100) '定义数组 ARRAY1 DIM VAR1 '定义变量 VAR1 VAR2 = 100 '赋值语句会自动定义文件模块变量 ARRAY1 = "asdf" ARRAY1(0,100,200,300) '对数组进行连续赋值 'ARRAY1(0)=100, ARRAY1(1)=200, ARRAY1(2)=300</pre>
相关指令	CONST , LOCAL , GLOBAL

LOCAL -- 局部定义

类型	语法指令
描述	<p>定义局部变量，局部变量。</p> <p>局部变量主要用于 SUB 中。 同一个 SUB 的局部变量有个数限制，SUB 过程的输入参数自动转化为局部变量。</p>

	不同任务的同一个 SUB 过程调用生成不同的局部变量，同一个任务的 SUB 过程递归调用也生成不同的局部变量。
语法	<pre> SUB subA() LOCAL localname 'localname 局部变量名 ENDSUB </pre>
适用控制器	通用
例子	<pre> SUB aaa() LOCAL v1 '定义局部变量 V1 v1=100 END SUB </pre>
相关指令	DIM , GLOBAL

GLOBAL -- 全局定义

类型	语法指令
描述	定义全局变量，数组；定义全局 SUB 过程。 全局定义的量可以在整个项目的任意程序文件中使用。
语法	语法 1: GLOBAL VAR1 语法 2: GLOBAL SUB SUB1() 语法 3: GLOBAL CONST CVARNAME = value 参数: VAR1: 变量名称 SUB1: 过程名称 CVARNAME: 常量名称 value: 常量值
适用控制器	通用
例子	<pre> GLOBAL SUB g_sub2() '定义全局过程 g_sub2，可以在任意文件中使用 GLOBAL CONST g_convar = 100 '定义全局使用的常量 GLOBAL g_var2 '定义全局变量 g_var2 </pre>
相关指令	DIM , LOCAL

7.3. 数组操作指令

DMINS -- 数组链表插入

类型	语法指令
描述	数组的链表操作，插入后当前元素以及后面的所有元素往后移动位置。 谨慎对超长数组进行操作，特别是数组 TABLE。
语法	DMINS arrayname(pos, size)

	arrayname: 数组名称 pos: 数组索引 size: 要修改的个数, 注意加上 pos 不要超过数组大小
适用控制器	通用
例子	<pre> DIM aa(6) '定义数组 aa FOR i=0 TO 4 '赋值 0,1,2,3,4 aa(i)=i NEXT ?*aa '打印数组所有元素 DMINS aa(0) '插入元素 0, 原有的所有元素都向后移动一个位置 aa(0) = 10 '为插入元素赋值 ?*aa '打印插入后的数组所有元素 </pre>
相关指令	DMDEL , DMCPY

DMADD -- 数组批量增加

类型	语法指令
描述	对数组元素值进行批量增加。 不要一次修改超过 500 元素。
语法	DMADD arrayname (pos, size , data) arrayname: 数组名 pos: 起始的索引 size: 要修改的个数, 注意加上 pos 不要超过数组大小 data: 要增加的值
适用控制器	通用
例子	<pre> DIM aaa(20) '定义一个空间为 20 的数组 ?*aaa '打印, 全为 0 DMADD aaa(10,5,2) '从元素 10 开始, 修改 5 个元素值+2 ?*aaa '打印, 其中 10、11、12、13、14 为 2, 其余为 0 DMADD aaa(10,5,2) '从元素 10 开始, 修改 5 个元素值+2 ?*aaa '打印, 其中 10、11、12、13、14 为 4, 其余为 0 </pre>
相关指令	DMINS , DMCPY

DMDEL -- 数组链表删除

类型	语法指令
描述	数组的链表操作; 删除数组元素, 删除后当前元素后面的所有元素向前移动。

	谨慎对超长数组进行操作，特别是数组 TABLE 。
语法	DMDEL arrayname(pos) arrayname: 数组名称 pos: 数组索引
适用控制器	通用
例子	DIM aa(6) '定义数组 aa FOR i=0 TO 4 '赋值 0,1,2,3,4 aa(i)=i NEXT ?*aa '打印数组所有元素 DMDEL aa(0) '删除数组 a 的第 1 个元素 ?*aa '打印删除后的数组所有元素
相关指令	DMINS , DMCPY

DMCPY -- 数组拷贝

类型	语法指令
描述	数组拷贝，从数组 Src 拷贝到数组 Des 。 谨慎对超长数组进行操作，特别是数组 TABLE 。
语法	DMCPY arraydes(startpos),arraysrc(startpos)[,size] arrayname: 数组名称 startpos: 数组起始索引 size: 拷贝的个数，超过最大值会自动缩减
适用控制器	通用
例子	GLOBAL aa(6),bb(6) '定义数组 aa, bb FOR i=0 TO 4 '赋值 aa 0,1,2,3,4 aa(i)=i NEXT ?*aa '打印数组所有元素 ?*bb DMCPY aa(0), bb(0),6 '把数组 bb 的值赋值给数组 aa ?*aa '打印数组复制后所有元素 ?*bb
相关指令	DMINS , DMDEL

DMSET -- 数组赋值

类型	语法指令
描述	数组区域赋值。 谨慎对超长数组进行操作，特别是数组 TABLE 。

语法	DMSET arrayname(pos, size, data) pos: 起始索引 size: 长度 data: 设置的数值
适用控制器	通用
例子	DMSET TABLE(0,10,2) '数组区域赋值 FOR i=0 TO 9 PRINT "TABLE",i, TABLE(i) '打印数组 NEXT DMSET TABLE(0,10,3) '数组区域赋值 FOR i=0 TO 9 PRINT "TABLE",i, TABLE(i) '打印数组 NEXT
相关指令	DMINS , DMDEL

7.4. 自定义子函数指令

SUB -- 自定义子函数 SUB

类型	语法指令
描述	用户自定义 SUB 过程，可以前面增加 GLOBAL 描述以定义全局使用的 SUB 过程。
语法	SUB label([para1] [,para2]...) ... END SUB 参数： label: 过程名称，不能与现有的关键词冲突 para1: 过程调用时传入的参数，自动作为 LOCAL 局部变量 para2: 过程调用时传入的参数，自动作为 LOCAL 局部变量
适用控制器	通用
例子	SUB sub1() '定义过程 SUB1，只能在当前文件中使用 ?1 ... END SUB GLOBAL SUB g_sub2() '定义全局过程 g_sub2，可以在任意文件中使用 ?2 ... END SUB GLOBAL SUB g_sub3(para1,para2) '定义全局过程 g_sub3，传递两个参数

	<pre>?Para1,para2 ... RETURN para1+para2 '函数返回参数相加 END SUB</pre>
相关指令	SUB PARA , SUB IFPARA

SUB_PARA -- SUB 传递参数

类型	语法指令
描述	选择 SUB 的传入参数。
语法	<pre>SUB_PARA(address) address: 第几个传递参数, 从 0 开始</pre>
适用控制器	通用
例子	<pre>SUB AAA(NUM1,NUM2,NUM3) ?SUB_PARA(0) '调用 AAA 时, 打印传递的第一个 num1 值 ?SUB_PARA(1) '打印传递的第二个 num2 值 ?SUB_PARA(2) '打印传递的第三个 num3 值 END SUB</pre>
相关指令	SUB , SUB IFPARA

SUB_IFPARA -- SUB 传参判断

类型	语法指令
描述	判断 SUB 是否传入参数。
语法	<pre>SUB_IFPARA(address) -1 传入, 0 未传入 address: 第几个传递参数, 从 0 开始</pre>
适用控制器	通用
例子	<pre>AAA(0,100) '传入 num1,num2 AAA(,100) '只传入 num2 END SUB AAA(NUM1,NUM2) IF SUB_IFPARA(0) THEN '调用 AAA 时, 判断 num1 是否传入 ?1 '传入打印 1 ELSE ?0 ENDIF END SUB</pre>
相关指令	SUB , SUB PARA

GOSUB/CALL -- SUB 调用

类型	程序结构
描述	<p>SUB 过程调用，只能调用本文件的 SUB 过程或全局 SUB 过程。</p> <p>直接调用 SUB 过程时，可以省掉 GOSUB 语句。 SUB 过程没有参数传递时，可以省掉括号()。</p> <p>GOSUB 后当前的内容会压栈，不能在调用的 SUB 程序中访问当前的局部变量，RETURN 返回时出栈。</p>
语法	<p>GOSUB/CALL label label: SUB 过程名</p>
适用控制器	通用
例子	<pre>'主程序 main: GOSUB sub1() sub2(1,2) '传入 1 给 para1, 2 给 para2 CALL sub3 END '定义的 SUB SUB sub1() a=100 PRINT "sub1" RETURN SUB sub2(para1,para2) a=200 PRINT "sub2",para1,para2 RETURN GLOBAL SUB sub3() '可以在另外一个程序文件中 a=300 PRINT "sub3" RETURN</pre>

GSUB -- 自定义子函数-G 代码格式

类型	语法指令
描述	<p>用户自定义 GSUB 过程。</p> <p>可以前面增加 GLOBAL 描述以定义全局使用的 GSUB 过程，GSUB 过程调用的时候采用 G 代码语法，不要加括号。</p>

语法	GSUB label([char1] [,char2]...) ... END SUB 参数: Label: 过程名称, 不能与现有的关键词冲突 char1: 过程调用时传入的字母参数, 自动作为 LOCAL 局部变量 char2: 过程调用时传入的字母参数, 自动作为 LOCAL 局部变量 字母参数只能为单字符
适用控制器	通用
例子	G01 X100 Y100 Z100 U100 '调用 G01 END '主程序结束 GLOBAL GSUB G01(X, Y, Z, U) '定义 GSUB 过程 G01 ... END SUB
相关指令	GSUB_PARA , GSUB_IFPARA

GSUB_PARA -- GSUB 传递参数

类型	语法指令
描述	选择 GSUB 的传入参数。
语法	GSUB_PARA(char) char: GSUB 定义时传入的字母参数
适用控制器	通用
例子	GSUB AAA(X,Y,Z) ?GSUB_PARA(X) '调用 AAA 时, 打印传递的 X 值 ?GSUB_PARA(Y) '打印传递的 Y 值 ?GSUB_PARA(Z) '打印传递的 Z 值 END SUB
相关指令	GSUB , GSUB_IFPARA

GSUB_IFPARA -- 判断 GSUB 是否传入参数

类型	语法指令
描述	判断 SUB 是否传入参数。
语法	GSUB_IFPARA(char) char: GSUB 定义时传入的字母参数 值: -1-传入, 0-未传入
适用控制器	通用

例子	<pre> AAA X0 Y100 '传入 X,Y AAA X0 '只传入 X END GSUB AAA(X,Y) IF GSUB_IFPARA(Y) THEN '调用 AAA 时，判断 Y 是否传入 ?1 '传入打印 1 ELSE ?0 ENDIF END SUB </pre>
相关指令	， GSUB PARA

END SUB -- 自定义函数结束

类型	程序结构
描述	用户自定义 SUB 过程结束，参见 SUB。
适用控制器	通用

RETURN -- 函数返回值

类型	程序结构
描述	<p>用户 SUB 过程返回或返回值。</p> <p>返回值缺省 0，在外面可以通过 RETURN 来读取上个 SUB 调用的返回值。</p> <p>不同任务的返回值不同。</p>
语法	RETURN
适用控制器	通用
例子	<pre> CALL sub1 ?RETURN '结果为 111 END '主程序结束 SUB sub1() RETURN 111 '返回 111 END SUB </pre>

7.5. 结构体定义指令

STRUCTURE -- 结构体定义

类型	语法指令
描述	<p>结构体定义。</p> <p>5 系列控制器 20180327 以上固件支持。</p> <p>4 系列控制器 fast 版本 20190107 以上固件支持。</p>
语法	<p>Structure 结构名称</p> <p> Dim 成员 1 名称 [As 数据类型 1]</p> <p> </p> <p> Dim 成员 n 名称[(数组长度)] [As 数据类型 1]</p> <p>End Structure</p> <p>数据类型只支持结构体，每个元素都同数组元素，占用一个数组元素空间。</p> <p>结构体不能递归。</p> <p>结构变量定义：</p> <p>DIM 变量名 AS 结构名</p> <p>DIM 结构数组名[(数组长度)] AS 结构名</p> <p>GLOBAL 变量名 AS 结构名</p> <p>GLOBAL 结构数组名[(数组长度)] AS 结构名</p> <p>预留功能：</p> <p>LOCAL 变量名 AS 结构名</p> <p>支持使用 FLASH_WRITE, FLASH_READ 指令读写结构体定义的变量和数组。</p> <p>FLASH_WRITE id, 结构变量</p> <p>FLASH_WRITE id, 结构数组</p> <p>FLASH_WRITE id, 结构数组(index)</p> <p>FLASH_WRITE id, 结构数组(index).item</p> <p>FLASH_WRITE id, 结构数组(index).item 数组(index)</p> <p>FLASH_READ 同上</p> <p>支持使用数组操作指令操作结构体数组。</p> <p>DMINS 结构数组(index) [,numes]</p> <p>DMINS 结构数组(index).item 数组(index) [,numes]</p> <p>DMDEL 同上</p> <p>DMCPY 结构数组 1(index1), 结构数组 2(index2) [,size]</p> <p>DMSET 只支持对最后一层的数组进行操作，不能对结构数组赋值。</p> <p>DMSET 结构变量.item 数组(index, size, data)</p> <p>DMADD 同上</p>

适用控制器	通用
例子	<pre> '声明结构体 AA GLOBAL Structure ClassAA DIM AA_val1 '成员变量 DIM AA_array(10) '成员数组 END Structure '声明结构体 BB GLOBAL Structure ClassBB DIM BB_val1 AS ClassAA '成员变量为结构体 END Structure '创建结构体变量 GLOBAL Class1 AS ClassAA GLOBAL Class2 AS ClassBB Class1.AA_val1=123 ?Class1.AA_val1 class1.AA_array="abc" ?class1.AA_array Class2.BB_val1.AA_val1=567 ?Class2.BB_val1.AA_val1 Class2.BB_val1.AA_array="zxc" ?Class2.BB_val1.AA_array AA_val1=8 FLASH_WRITE 0,AA_val1 AA_val1=123 FLASH_READ 0,AA_val1 ?AA_val1 END </pre>
相关指令	DIM , GLOBAL , UNION

UNION -- 共用体定义

类型	语法指令
描述	共用体定义。
	5 系列控制器 20180327 以上固件支持。

	4 系列控制器 fast 版本 20190107 以上固件支持。
语法	<p>UNION 结构名称</p> <p> Dim 成员 1 名称 [As 数据类型 1]</p> <p> </p> <p> Dim 成员 n 名称[(数组长度)] [As 数据类型 1]</p> <p>END UNION</p> <p>结构变量定义：</p> <p>DIM 变量名 AS 结构名</p> <p>DIM 结构数组名[(数组长度)] AS 结构名</p> <p>GLOBAL 变量名 AS 结构名</p> <p>GLOBAL 结构数组名[(数组长度)] AS 结构名</p> <p>预留功能：</p> <p>LOCAL 变量名 AS 结构名</p> <p>支持使用 FLASH_WRITE, FLASH_READ 指令读写结构体定义的变量和数组。</p> <p>FLASH_WRITE id, 结构变量</p> <p>FLASH_WRITE id, 结构数组</p> <p>FLASH_WRITE id, 结构数组(index)</p> <p>FLASH_WRITE id, 结构数组(index).item</p> <p>FLASH_WRITE id, 结构数组(index).item 数组(index)</p> <p>FLASH_READ 同上</p> <p>支持使用数组操作指令操作结构体数组。</p> <p>DMINS 结构数组(index) [,numes]</p> <p>DMINS 结构数组(index).item 数组(index) [,numes]</p> <p>DMDEL 同上</p> <p>DMCPY 结构数组 1(index1), 结构数组 2(index2) [,size]</p> <p>DMSET 只支持对最后一层的数组进行操作, 不能对结构数组赋值。</p> <p>DMSET 结构变量.item 数组(index, size, data)</p> <p>DMADD 同上</p>
适用控制器	通用
例子	<p>'声明共用体 AA</p> <p>GLOBAL UNION ClassAA</p> <p> DIM AA_val1 '成员变量</p> <p> DIM BB_val1 '成员变量</p> <p> DIM AA_array(4) '成员数组</p> <p>END UNION</p> <p>'创建共用体变量</p> <p>GLOBAL Class1 AS ClassAA</p> <p>Class1.AA_val1=123</p>

	<pre>?Class1.AA_val1 '结果: 123 Class1.BB_val1=456 ?Class1.BB_val1 '结果: 456 ?Class1.AA_val1 '结果: 456 END</pre>
相关指令	DIM , GLOBAL , STRUCTURE

7.6. 跳转指令

GOTO -- 强制跳转

类型	程序结构
描述	强制跳转，与 GOSUB 的区别是 GOTO 不会压栈。
语法	GOTO label
适用控制器	通用
例子	<pre>a=100 GOTO label1 '强制跳转至 label1 a=1000 END '主程序结束 label1: PRINT a '结果是 a=100 END 'label1 结束</pre>

ON GOSUB -- 条件跳转

类型	程序结构
描述	当 expression 条件为真时，调用过程 label 。
语法	<pre>ON expression GOSUB label expression: 判断条件 label: 跳转 sub、label 名称</pre>
适用控制器	通用
例子	<pre>a=100 ON a>10 GOSUB label1 'a>10 时调用 label 过程 a=1000 PRINT a END '主程序结束</pre>

	label1: PRINT a RETURN	'SUB 过程要返回
--	------------------------------	------------

ON GOTO -- 条件跳转 2

类型	程序结构
描述	条件跳转，当 expression 条件为真时跳转，不会压栈。
语法	ON expression GOTO label expression: 判断条件 label: 跳转 sub、label 名称
适用控制器	通用
例子	<pre> a=100 on a>10 goto label1 a=1000 END '主程序结束 label1: PRINT a END 'goto 跳转无法 return 返回 </pre>

7.7. 条件判断指令

IF -- 条件判断结构

类型	程序结构
描述	条件判断，同标准 BASIC 结构。
语法	<pre> IF <condition1> THEN commands ELSEIF <condition2> THEN commands ELSE commands ENDIF </pre> <p>参数：</p> <pre> condition1: 条件 condition2: 条件 </pre>
适用控制器	通用
例子	例一

	<pre> DIM a '定义变量 a=12 '赋值 IF a>11 THEN '条件判断 TRACE "the val a is bigger then 11" ELSEIF a<11 THEN TRACE "the val a is less then 11" ENDIF 例二 IF IN(0) THEN OUT(0,ON) '当单行时可以用不用 endif </pre>
相关指令	THEN , ENDIF

THEN -- 条件判断结构

类型	程序结构
描述	参见: IF
适用控制器	通用

ENDIF -- 条件判断结构

类型	程序结构
描述	参见: IF
适用控制器	通用

ELSEIF -- 条件判断结构

类型	程序结构
描述	参见: IF
适用控制器	通用

7.8. 循环指令

FOR -- for 循环结构

类型	程序结构
----	------

描述	循环语句，采用标准 BASIC 语法。
语法	<pre>FOR variable=start TO end [STEP increment] commands NEXT variable</pre> <p>参数：</p> <p>variable: 变量名称</p> <p>start: 起始循环值</p> <p>end: 结束循环值</p> <p>increment: 循环步进增量，可选</p> <p>多任务时，请不要使用（非 local 时）相同的 variable 循环变量，否则会相互干扰。</p>
适用控制器	通用
例子	<pre>LOCAL a FOR a=1 TO 100 '1 循环至 100 PRINT a '打印 a NEXT</pre>
相关指令	TO ， STEP ， NEXT

TO -- for 循环结构

类型	程序结构
描述	参见：FOR
适用控制器	通用

STEP -- for 循环结构

类型	程序结构
描述	参见：FOR
适用控制器	通用

NEXT -- for 循环结构

类型	程序结构
描述	参见：FOR
适用控制器	通用

WHILE -- while 循环结构

类型	程序结构
描述	条件满足时执行循环。
语法	<pre>WHILE condition ... WEND</pre>
适用控制器	通用
例子	<pre>a=0 WHILE IN(4)=OFF '直到输入 4 有效，退出循环 a=a+1 PRINT a DELAY(1000) WEND</pre>

WEND -- while 循环结构

类型	程序结构
描述	参见： WHILE
适用控制器	通用

EXIT -- 退出循环

类型	程序结构
描述	循环退出语句。
语法	EXIT FOR, EXIT WHILE
适用控制器	通用
例子	<pre>LOCAL a FOR a=1 TO 100 '1 循环至 100 PRINT a '打印 a IF a> 20 THEN EXIT FOR '必须采用这种方式，否则 IF 和 ENDIF 不匹配 NEXT</pre>

REPEAT -- 条件循环

类型	程序结构
描述	<p>循环语句。</p> <p>循环执行 commands, condition 为真时退出循环。</p>
语法	REPEAT commands UNTIL condition

适用控制器	通用
例子	<pre> a=0 REPEAT '循环执行下面语句 PRINT a a=a+1 DELAY(1000) UNTIL IN(4)=ON '直到输入 4 有效 </pre>

UNTIL -- 条件结构

类型	程序结构
描述	参见: REPEAT 、 WAIT
适用控制器	通用

7.9. 等待执行指令

DELAY -- 延时

类型	语法指令
描述	延时 delay time , 单位毫秒。 别名: wa
语法	DELAY (delay time) delay time: 毫秒数
适用控制器	通用
例子	DELAY (100) '延时 100ms

WAIT UNTIL -- 等待条件满足

类型	程序结构
描述	等待直到条件满足。
语法	WAIT UNTIL condition1 [and condition2 or condition3 ...] 可以通过逻辑运算操作多个条件。
适用控制器	通用
例子	例一 WAIT UNTIL DPOS(0) > 0 '等待轴 0 位置超过 0 例二 与 TICKS 一起使用

	<p>TICKS=2000 'ticks 设为 2000</p> <p>WAIT UNTIL TICKS<0 '等待 2s</p> <p>?"执行下一步"</p> <p>例三 与逻辑条件一起使用</p> <p>WAIT UNTIL IDLE(0)=-1 AND IDLE(1)=-1 AND IDLE(2)=-1</p> <p>'等待轴 0、1、2 都停止</p>
--	---

WAIT IDLE -- 等待轴停止

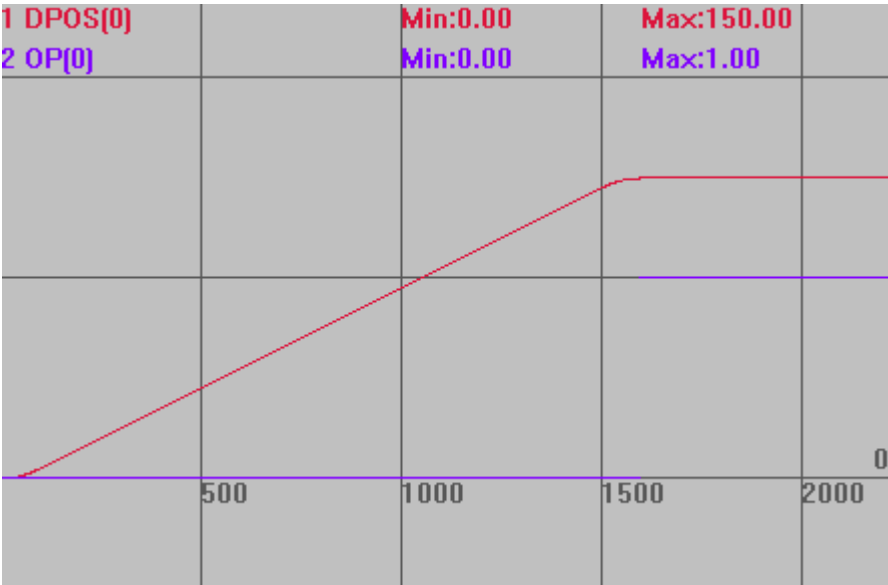
类型	语法指令
描述	等待 BASE 轴运动完成，BASE 轴运动未完成时，不执行后面的程序。 等效于 WAIT UNTIL IDLE。IDLE 作为轴参数，支持轴参数的语法。
语法	WAIT IDLE
适用控制器	通用
例子	<p>例一</p> <p>BASE(0,1)</p> <p>MOVE(100,100)</p> <p>WAIT IDLE '等待当前插补运动结束</p> <p>例二</p> <p>BASE(0,1)</p> <p>MOVE(100,100)</p> <p>BASE(2,3)</p> <p>MOVE(200,200)</p> <p>WAIT UNTIL IDLE(0) AND IDLE(1) AND IDLE (2) AND IDLE(3)</p> <p>'等待轴 0，1，2，3 停止</p> <p>?"运动完成"</p>

WAIT LOADED -- 等待轴缓冲空

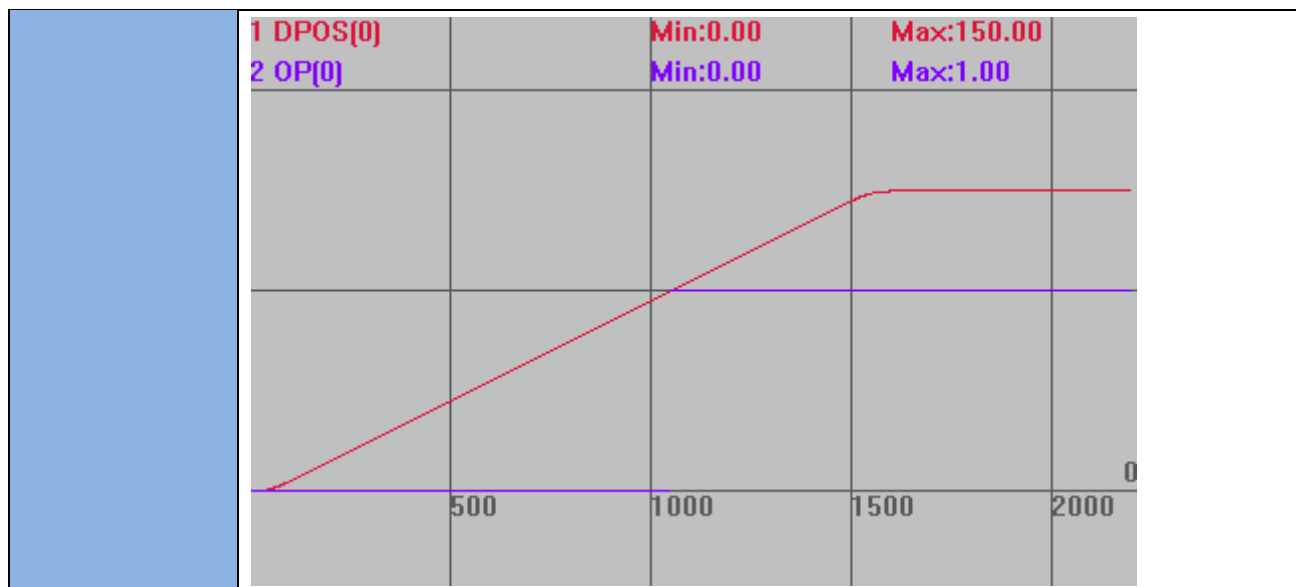
类型	语法指令
描述	等待 BASE 轴运动缓冲空，此命令会阻塞不执行后面的程序。 缓冲区最后一个运动可以正确执行，同时程序向下扫描。 等效于 WAIT UNTIL LOADED，LOADED 作为轴参数，支持轴参数的语法。
语法	WAIT LOADED
适用控制器	通用
例子	<p>与 WAIT IDLE 的区别</p> <p>BASE(0)</p> <p>ATYPE=1</p>

UNITS=100
DPOS=0
SPEED=100
ACCEL=1000
MERGE=1
TRIGGER
MOVE(100) '当前运动
MOVE(50) '缓冲运动，此时缓冲区只有这一条运动
'当本条运动执行时，缓冲区就已经清空
WAIT IDLE '使用 wait idle 时，要等到所有运动完成才能往下执行
OP(0,ON) '打开 op0

运动轨迹
DPOS(0)垂直刻度 100
OP(0)垂直刻度 1



WAIT LOADED '使用 wait loaded 时，运动缓冲空即可往下执行
OP(0,ON)



7.10. ZINDEX 指针指令

ZINDEX_LABEL -- 建立指针索引

类型	语法指令
描述	建立索引指针，便于后续调用指针内容。
语法	Pointer = zindex_label(subname) subname: 数组或 SUB 名称
适用控制器	通用
例子	DIM arr1(100) '定义数组 arr1(0,1) '对数组 0 地址赋值 1 Pointer = ZINDEX_LABEL(arr1) '建立索引指针 PRINT ZINDEX_ARRAY(Pointer) (0) '访问数组，打印数组第一位数据，结果 1
相关指令	<u>ZINDEX_CALL</u> , <u>ZINDEX_ARRAY</u> , <u>ZINDEX_VAR</u>

ZINDEX_CALL -- 访问 SUB 函数

类型	语法指令
描述	通过索引指针来调用 SUB 函数。
语法	ZINDEX_CALL(zidnex) (subpara, ...) zidnex: 通过 ZINDEX_LABEL 生成的索引指针 subpara: sub 的参数调用
适用控制器	通用

例子	<pre> Pointer = ZINDEX_LABEL(sub1) '建立索引指针 ZINDEX_CALL(Pointer) (2) '调用函数 SUB sub1(a) PRINT a END SUB </pre>
相关指令	ZINDEX_LABEL

ZINDEX_ARRAY -- 访问数组

类型	语法指令
描述	通过索引指针来访问数组。
语法	<pre> var = ZINDEX_ARRAY (pointer)(index) pointer: 通过 ZINDEX_LABEL 生成的指针索引 index: 数组索引 </pre>
适用控制器	通用
例子	<pre> DIM arr1(100) '定义数组 arr1(0,1) '对数组 0 地址赋值 1 Pointer = ZINDEX_LABEL(arr1) '建立索引指针 PRINT ZINDEX_ARRAY(Pointer) (0) '访问数组，打印数组第一位数据，结果 1 </pre>
相关指令	ZINDEX_LABEL

ZINDEX_VAR -- 访问变量

类型	语法指令
描述	通过索引指针来访问变量。
语法	<pre> ZINDEX_VAR(zindex) zindex: 通过 ZINDEX_LABEL 生成的索引指针 zindex= ZINDEX_LABEL(varname) ZINDEX_VAR(zindex)=value VAR2 = ZINDEX_VAR(zindex) </pre>
适用控制器	通用
例子	<pre> global gTestVar global VarAdd1 VarAdd1=ZINDEX_LABEL(gTestVar) ZINDEX_VAR(VarAdd1)=10 ?ZINDEX_VAR(VarAdd1) </pre>
相关指令	ZINDEX_LABEL

ZINDEX_STRUCT -- 访问结构体

类型	语法指令
描述	获取结构变量的指针后，通过指针来访问结构变量或数组。
语法	<pre> zindex = ZINDEX_LABEL(structvarname) ZINDEX_STRUCT(structname,index).item = var var = ZINDEX_STRUCT(structname,index).item </pre> <p> zindex: 通过 ZINDEX_LABEL 生成的索引指针 structvarname: 结构体变量名称 structname: 结构体名称 Item: 结构体成员 </p>
适用控制器	<p>结构体指针功能只有特殊固件版本支持：</p> <p>5 系列控制器 20180327 以上固件支持。</p> <p>4 系列控制器 fast 版本 20190107 以上固件支持。</p>
例子	<pre> GLOBAL Structure ClassAA '结构体声明 DIM AA_val1 '成员变量 DIM AA_array(10) '成员数组 END Structure GLOBAL Class1 AS ClassAA '结构体全局变量定义 GLOBAL gStructureAdd Class1.AA_array(0,1,2,3) '结构体数组赋值 ?Class1.AA_array(0) '结果： 1 gStructureAdd = ZINDEX_LABEL(Class1) '建立结构体索引指针 ?ZINDEX_STRUCT(ClassAA,gStructureAdd).AA_array(0) '结果： 1 ZINDEX_STRUCT(ClassAA,gStructureAdd).AA_array(0)= 10 ?ZINDEX_STRUCT(ClassAA,gStructureAdd).AA_array(0) '结果： 10 END </pre>
相关指令	ZINDEX_LABEL

第八章 任务相关指令

XBASIC 支持实时多任务运行，一个文件上也可以同时运行多个任务。通过 RUN 可以从文件第一行开始运行任务，通过 RUNTASK 可以随意指定 SUB 过程开始运行。

8.1. 任务启停指令

RUN -- 启动文件任务

类型	任务指令
描述	<p>新建一个任务来执行控制器上的一个文件。</p> <p>重复启动同一任务会报错。</p> <p>多次使用 RUN 指令不带任务号参数时，会让同一个文件对应多个任务，建议使用 RUNTASK 指令开启任务。</p> <p>多任务操作指令有：</p> <ul style="list-style-type: none">END：当前任务正常结束STOP：停止指定文件STOPTASK：停止指定任务HALT：停止所有任务RUN：启动文件执行RUNTASK：启动任务在一个 SUB 上执行
语法	<p>RUN "filename"[, tasknum]</p> <p>filename：程序文件名，bas 文件可不加扩展名</p> <p>tasknum：任务号，缺省查找第一个有效的</p>
适用控制器	通用
例子	RUN "aaa", 1 '启动任务 1 运行 aaa.bas 文件
相关指令	RUNTASK

RUNTASK -- 启动 SUB 任务

类型	任务指令
描述	<p>把一个 SUB 过程或是标号作为一个新的任务执行。</p> <p>重复启动同一任务会报错。</p>
语法	<p>RUNTASK tasknum, label</p> <p>tasknum：任务号</p> <p>label：自定义 SUB 过程（不能带参数）或标号</p>
适用控制器	通用

例子	<pre> RUNTASK 1, taska '启动任务 1 来跟踪打印位置 MOVE(1000,100) MOVE(1000,100) END taska: '循环打印位置 WHILE 1 PRINT *mpos DELAY(1000) WEND END </pre>
相关指令	RUN

END -- 结束

类型	任务指令
描述	<p>当前任务结束。</p> <p>当一个文件中有主程序和 SUB 过程时，一定要在主程序结束后写 END，如果不写 END，程序执行完主程序后，就会依次再执行 SUB 子程序，直到子程序的 END SUB 才停止。</p>
适用控制器	通用
相关指令	RUN ， RUNTASK

STOP -- 停止文件任务

类型	任务指令
描述	<p>程序强制停止，操作文件。</p> <p>再启动任务前都要停止任务。</p> <p>使用 STOP 指令不带任务号时，一次只会停掉一个任务，而不是此文件对应的所有任务，当一个文件内有多个任务时，建议用 STOPTASK 指令。</p>
语法	<pre> STOP program name, [tasknum] </pre> <p>program name: 程序文件名，bas 文件可不用带扩展名</p> <p>tasknum: 任务号，当程序文件有启动多个任务时，缺省任务号最小的任务</p>
适用控制器	通用
例子	<pre> RUN aaa, 1 '执行 aaa.bas STOP aaa, 1 '停止任务 1 </pre>
相关指令	STOPTASK ， HALT

STOPTASK -- 停止 SUB 任务

类型	任务指令
描述	任务强制停止，操作 SUB 和标记。 再启动任务前都要停止任务。
语法	STOPTASK [tasknum] tasknum: 任务号，缺省当前任务
适用控制器	通用
例子	STOPTASK 2 '停止任务 2
相关指令	STOP ， HALT

PAUSE -- 暂停全部任务

类型	任务指令
描述	暂停全部任务。 使用断点生效后也会进入暂停状态。 此指令仅限 PC 软件调用，BASIC 程序中若使用此指令，会导致整个程序停止，控制器无法运行。 暂停任务再恢复后，任务继续往下执行。
语法	PAUSE
适用控制器	通用
例子	PAUSE '暂停所有任务
相关指令	PAUSETASK

PAUSETASK -- 暂停指定任务

类型	任务指令
描述	暂停某一个任务。 暂停任务再恢复后，任务继续往下执行。
语法	PAUSETASK tasknum tasknum: 任务号，缺省当前任务
适用控制器	通用
例子	PAUSETASK 1 '暂停任务 1
相关指令	RESUMETASK

RESUMETASK -- 恢复指定任务

类型	任务指令
描述	恢复某一个任务。 暂停任务再恢复后，任务继续往下执行。
语法	RESUMETASK tasknum tasknum: 任务号，缺省当前任务
适用控制器	通用
例子	PAUSETASK 1 '暂停任务 1 RESUMETASK 1 '继续运行任务 1
相关指令	PAUSETASK

8.2. 三次文件任务指令

FILE3_RUN -- 执行 FILE3 任务

类型	任务指令
描述	3 次程序文件的启动命令。 3 次程序文件是一种超大文件，可以动态加载，使用 basic 语法。不支持条件判断、跳转等操作。可以通过指令下载，也可以通过工具软件 zfile3view 进行浏览、上传和下载操作。
语法	FILE3_RUN "filename", tasknum filename: 3 次程序的文件名，必须事先下载到控制器里面 tasknum: 任务号，缺省查找第一个有效的
适用控制器	必须带大容量存储的控制器和 2015 以上的固件版本支持。
例子	FILE3_RUN "aaa.z3p", 1 '在任务 1 运行 3 次文件 aaa.z3p
相关指令	FILE3_ONRUN

FILE3_ONRUN --FILE3 回调函数

类型	回调函数
描述	3 次文件启动时会自动调用。
语法	GLOBAL FILE3_ONRUN: 标号 GLOBAL SUB FILE3_ONRUN() 自定义 SUB 过程（不能带参数） 回调函数属于 3 次文件任务。
适用控制器	通用
例子	FILE3_RUN "aaa.z3p", 1 '在任务 1 运行 3 次文件 aaa.z3p

	<pre> END GLOBAL SUB FILE3_ONRUN() '3 次任务启动时自动调用 IF 1= PROCNUMBER THEN BASE(0,1,2) '选择 3 次文件的运行轴列表 SPEED=1000 ACCEL=10000 ELSE BASE(4,5,6) ENDIF END SUB </pre>
相关指令	FILE3_RUN

FILE3_GOTO -- FILE3 强制跳转

类型	任务函数
描述	对三次任务有效，强制跳转到指定的行号开始运行。
语法	<pre>FILE3_GOTO(linenum)</pre> <p>linenum: 要跳转的行号，从 1 开始编号</p>
适用控制器	通用
相关指令	FILE3_LINE , FILE3_RUN

FILE3_LINE -- FILE3 行号

类型	任务函数
描述	返回当前 3 次文件运行的行号，无论是否三次文件因为 SUB 调用进入 BASIC 文件，总是返回 3 次文件的行号。
语法	<pre>VALUE=FILE3_LINE([taskid])</pre> <p>taskid: 三次文件的任务号，不填返回函数调用当前任务的</p>
适用控制器	通用
相关指令	FILE3_RUN , FILE3_GOTO

第九章 运算符及数学函数指令

XBASIC 支持标准 BASIC 的所有运算符，也采用标准 BASIC 的优先级。

优先级顺序：算术运算符 > 比较运算符 > 逻辑运算符。相同优先级的运算符采用从左到右的顺序计算。

算术运算符		比较运算符		逻辑运算符	
描述	符号	描述	符号	描述	符号
求幂	\wedge	等于	=	按位非	Not
负号	-	不等于	\neq	按位与	And
乘	*	小于	<	按位或	Or 或
除	/	大于	>	按位异或	Xor
整除	\	小于等于	<=	按位同或	Eqv
求余	Mod 或 %	大于等于	>=		
加	+				
减	-				
左移位	<<				
右移位	>>				

9.1. 算术运算指令

+ -- 加法运算

类型	运算符
描述	将两个表达式相加。
语法	expression1 + expression2 expression1: 任意有效的表达式 expression2: 任意有效的表达式
适用控制器	通用
例子	在线命令输入 >>PRINT 1+2 输出: 3

- -- 减法运算

类型	运算符
描述	将表达式 1 减去表达式 2。
语法	expression1 - expression2 expression1: 任意有效的表达式 expression2: 任意有效的表达式
适用控制器	通用
例子	在线命令输入 >>PRINT 2-(2-1) 输出: 1

* -- 乘法运算

类型	运算符
描述	将表达式 1 与表达式 2 相乘。
语法	expression1 * expression2 expression1: 任意有效的表达式 expression2: 任意有效的表达式
适用控制器	通用
例子	在线命令输入 >>PRINT 10*(1+2) 输出: 30

/ -- 除法运算

类型	运算符
描述	将表达式 1 除以表达式 2。
语法	expression1 / expression2 expression1: 任意有效的表达式 expression2: 任意有效的表达式
适用控制器	通用
例子	在线命令输入 >>PRINT 10/3 输出: 3.3333

\ -- 整除运算

类型	运算符
描述	整数除法。
语法	expression1 \ expression2 expression1: 任意有效的表达式 expression2: 任意有效的表达式
适用控制器	通用
例子	在线命令输入 >>PRINT 10 \ (1+2) 输出: 3

<< -- 左移位

类型	运算符
描述	左移位。
语法	<p>expression1 << expression2</p> <p>expression1: 任意有效的表达式</p> <p>expression2: 任意有效的表达式</p> <p>运算优先级低于其他四则运算符，共同使用时需要加括号(), 见例三。</p>
适用控制器	通用
例子	<p>例一 直接操作数值</p> <p>在线命令输入</p> <p>>>PRINT 8<<1 '二进制左移一位</p> <p>输出: 16</p> <p>在线命令输入</p> <p>>>PRINT 8<<2 '二进制左移两位</p> <p>输出: 32</p> <p>例二 操作变量、寄存器</p> <p>DIM bb</p> <p>bb=8</p> <p>MODBUS_REG(0)=8</p> <p>PRINT bb<<1,bb<<2</p> <p>PRINT MODBUS_REG(0)<<1,MODBUS_REG(0)<<2</p> <p>例三 优先级比较</p> <p>>>PRINT 8<<1+1 '此时二进制左移 2 位</p> <p>输出: 32</p> <p>>>PRINT (8<<1)+1 '此时二进制左移 1 位</p> <p>输出: 17</p>

>> -- 右移位

类型	运算符
描述	右移位。
语法	<p>expression1 >> expression2</p> <p>expression1: 任意有效的表达式</p> <p>expression2: 任意有效的表达式</p>

	运算优先级低于其他四则运算符，共同使用时需要加括号()，见例三。
适用控制器	通用
例子	<p>例一 直接操作数值</p> <p>在线命令输入</p> <pre>>>PRINT 8>>1 '二进制右移一位</pre> <p>输出：4</p> <p>在线命令输入</p> <pre>>>PRINT 8>>2 '二进制右移两位</pre> <p>输出：2</p> <p>例二 操作变量、寄存器</p> <pre>DIM bb bb=8 MODBUS_REG(0)=8 PRINT bb>>1,bb>>2 PRINT MODBUS_REG(0)>>1,MODBUS_REG(0)>>2</pre> <p>例三 优先级比较</p> <pre>>>PRINT 8>>1+1 '此时二进制右移 2 位</pre> <p>输出：2</p> <pre>>>PRINT (8>>1)+1 '此时二进制右移 1 位</pre> <p>输出：5</p>

MOD -- 求余数

类型	运算符
描述	求余数。
语法	<p>expression1 MOD expression2</p> <p>expression1: 任意有效的表达式，取整数部分。</p> <p>expression2: 任意有效的表达式，取整数部分。</p>
适用控制器	通用
例子	<p>在线命令输入</p> <pre>>>PRINT 10 MOD (1+2)</pre> <p>输出：1</p>

ABS -- 绝对值

类型	数学函数
描述	求绝对值。

语法	ABS(expression) expression: 任意有效的表达式
适用控制器	通用
例子	PRINT ABS(-11) '结果是 11

9.2. 比较运算指令

= -- 比较/赋值运算

类型	运算符
描述	比较运算符: 如果表达式 1 等于表达式 2, 返回 TRUE, 否则返回 FALSE 赋值运算符: 将表达式 2 赋值给前面的变量或参数等。
语法	expression1 = expression2 expression1: 任意有效的表达式 expression2: 任意有效的表达式
适用控制器	通用
例子	<p>例一</p> <p>ON IN(0)=ON GOTO label1 '如果输入通道 0 为 ON, 程序跳转到标识 "label1:" 首行开始执行</p> <p>label1:</p> <p>PRINT 12 '打印 12</p> <p>例二</p> <p>DIM aaa</p> <p>aaa = 100 '变量 aaa 赋值为 100</p> <p>PRINT aaa</p>

<> -- 不等于

类型	运算符
描述	如果表达式 1 不等于表达式 2, 返回 TRUE, 否则返回 FALSE。
语法	expression1 <> expression2 expression1: 任意有效的表达式 expression2: 任意有效的表达式
适用控制器	通用
例子	ON MODBUS_BIT(0)<>0 GOTO label1 '如果 MODBUS 位寄存器 0 非零值, 程序跳转到标识"label1:"开始执行

	label1: PRINT 11 '打印 11
--	---------------------------------

> -- 大于

类型	运算符
描述	如果表达式 1 大于表达式 2，返回 TRUE，否则返回 FALSE。
语法	expression1 > expression2 expression1: 任意有效的表达式 expression2: 任意有效的表达式
适用控制器	通用
例子	<p>WAIT UNTIL MPOS>100 该条语句将使程序循环等待，直到测量反馈位置大于 100 为止。</p> <p>例一</p> <pre>DIM q '定义变量 q= 2>1 '2 大于 1，所以返回 true PRINT q '打印返回值</pre> <p>例二</p> <pre>DIM a '定义变量 a=0 '变量赋值 REPEAT '循环执行 a=a+1 '加一 ?a '打印 DELAY(200) '延时 UNTIL a>10 '条件判断，直到 a 里面的值大于 10 时，停止循环执行</pre>

>= -- 大于等于

类型	运算符
描述	如果表达式 1 大于或等于表达式 2，返回 TRUE，否则返回 FALSE。
语法	expression1 >= expression2 expression1: 任意有效的表达式 expression2: 任意有效的表达式
适用控制器	通用
例子	<pre>DIM a '定义变量 a= 1>=3 '1 小于 3，所以返回 FALSE PRINT a '打印返回值</pre>

< -- 小于

类型	运算符
描述	如果表达式 1 小于表达式 2，返回 TRUE，否则返回 FALSE。
语法	expression1 < expression2 expression1: 任意有效的表达式 expression2: 任意有效的表达式
适用控制器	通用
例子	VAR1=1<0 因为 1 大于 0，所以 VAR1 的值将等于 FALSE(0)。

<= -- 小于等于

类型	运算符
描述	如果表达式 1 小于或等于表达式 2，返回 TRUE，否则返回 FALSE。
语法	expression1 <= expression2 expression1: 任意有效的表达式 expression2: 任意有效的表达式
适用控制器	通用
例子	VAR1=1<=1 VAR1 的值将等于 TRUE (-1)。

9.3. 逻辑运算指令

AND -- 按位与

类型	运算符		
描述	按位与操作符，只操作整数部分。		
	AND		结果
	0	0	0
	0	1	0
	1	0	0
	1	1	1
语法	expression1 AND expression2		
	expression1: 任意有效的表达式 expression2: 任意有效的表达式		
	expression1 和 expression2 的二进制形式的每一个位上的二进制数字进行按位与(AND)		

	运算之后的结果
适用控制器	通用
例子	<p>在线命令输入 >>PRINT 1 AND 2 输出：0</p> <p>具体操作过程 1 对应二进制 01 2 对应二进制 10 与计算后 对应二进制 00 输出十进制 0</p>

OR -- 按位或

类型	运算符																	
描述	按位或操作符，只操作整数部分。 <table><tr><td colspan="2">OR</td><td>结果</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>			OR		结果	0	0	0	0	1	1	1	0	1	1	1	1
				OR		结果												
				0	0	0												
				0	1	1												
				1	0	1												
				1	1	1												
语法	expression1 OR expression2 expression1：任意有效的表达式 expression2：任意有效的表达式 expression1 和 expression2 的二进制形式的每一个位上的二进制数字进行按位或（OR）运算之后的结果																	
适用控制器	通用																	
例子	在线命令输入 >>PRINT 1 OR 2 输出：3 具体操作过程 1 对应二进制 01 2 对应二进制 10 或计算后 对应二进制 11 输出十进制 3																	

NOT -- 按位非

类型	运算符
----	-----

描述	按位非操作符，只操作整数部分，小心对 ON 等整数 NOT。	
	NOT	结果
	0	-1
	1	-2
语法	NOT expression1 expression1: 任意有效的表达式	
适用控制器	通用	
例子	<p>在线命令输入 >>PRINT NOT 1 输出: -2</p> <p>具体操作过程 1 对应二进制 ... 0000 0001 非计算后 对应二进制 ... 1111 1110 输出十进制 -2</p>	

XOR -- 按位异或

类型	运算符	
描述	逻辑异或操作符，按位异或，只操作整数部分。	
	XOR	结果
	0	0
	0	1
	1	0
	1	1
语法	expression1 XOR expression2 expression1: 任意有效的表达式 expression2: 任意有效的表达式	
适用控制器	通用	
例子	<p>在线命令输入 >>PRINT 1 XOR 1 输出: 0</p> <p>具体操作过程 1 对应二进制 01 异或计算后 对应二进制 00 输出十进制 0</p>	

EQV -- 按位同或

类型	运算符
----	-----

描述	按位同或操作符，只操作整数部分。		
	EQV		结果
	0	0	1
	0	1	0
	1	0	0
	1	1	1
语法	expression1 EQV expression2 expression1: 任意有效的表达式 expression2: 任意有效的表达式		
适用控制器	通用		
例子	在线命令输入 >>PRINT 2 EQV 1 输出: -4 具体操作过程 2 对应二进制 ... 0000 0010 1 对应二进制 ... 0000 0001 同或计算后 对应二进制 ... 1111 1100 输出十进制 -4		

9.4. 三角函数指令

SIN -- 三角函数正弦

类型	数学函数
描述	正弦三角函数，输入参数为弧度单位。
语法	SIN(expression) expression: 任意有效的表达式
适用控制器	通用
例子	PRINT SIN(PI/6) '结果是 0.5000

ASIN -- 三角函数反正弦

类型	数学函数
描述	反正弦三角函数，返回值为弧度单位。
语法	ASIN(expression) expression: 任意有效的表达式
适用控制器	通用
例子	PRINT ASIN(0.5) '结果是 0.52360

COS -- 三角函数余弦

类型	数学函数
描述	余弦三角函数，输入参数为弧度单位。
语法	COS(expression) expression: 任意有效的表达式
适用控制器	通用
例子	PRINT COS(PI/3) '结果是 0.5000

ACOS -- 三角函数反余弦

类型	数学函数
描述	反余弦三角函数，返回值为弧度单位。
语法	ACOS(expression) expression: 任意有效的表达式
适用控制器	通用
例子	PRINT ACOS(0.5) '结果是 1.04720=PI/3

TAN -- 三角函数正切

类型	数学函数
描述	求正切三角函数，输入参数为弧度单位。
语法	TAN(expression) expression: 任意有效的表达式
适用控制器	通用
例子	PRINT TAN(PI/3) '结果是 1.732

ATAN -- 三角函数反正切

类型	数学函数
描述	求反正切三角函数，返回值为弧度单位。
语法	ATAN(expression) expression: 任意有效的表达式
适用控制器	通用

例子	PRINT ATAN(1) '结果是 0.7854 = (45/180)*PI
----	--

ATAN2 -- 三角函数反正切 2

类型	数学函数
描述	反正切三角函数，返回值为弧度单位。
语法	ATAN2(y, x) y: y 坐标 x: x 坐标
适用控制器	通用
例子	PRINT ATAN2(1,0) '结果是 1.5708

9.5. 指数运算指令

EXP -- 指数

类型	数学函数
描述	指数函数。
语法	EXP([base,] expvalue) base: 底数，缺省为 e expvalue: 指数
适用控制器	通用
例子	例一 PRINT EXP(2,4) '结果是 16 (2*2*2*2) 例二 PRINT EXP(1) '结果是 2.7183

SQR -- 平方根

类型	数学函数
描述	平方根函数。
语法	SQR(expression) expression: 任意有效的表达式
适用控制器	通用
例子	a= SQR(4) PRINT a '结果是 2

LN -- 自然对数

类型	数学函数
描述	自然对数函数。
语法	LN(expression) expression: 任意有效的表达式
适用控制器	通用
例子	a= LN(1) PRINT a '结果是 0

LOG -- 对数底为 10

类型	数学函数
描述	对数，底数为 10。
语法	LOG(expression) expression: 任意有效的表达式
适用控制器	通用
例子	a= LOG(100) PRINT a '结果是 2

9.6. 数据操作指令

SET_BIT -- 按位设置

类型	数学指令或函数
描述	位操作，只对整数，修改对应位为 1。 分为命令语法和函数语法。
语法	命令语法: SET_BIT(bit#, vr#) 直接操作 VR 寄存器 bit#: 位编号: 0-31 vr#: 要操作的 VR 变量编号，整数部分 命令语法使用时没有返回值，直接操作，修改操作对象的值。 函数语法: ret = SET_BIT(bit#,int) ret: 操作结果 bit#: 位编号: 0-31 int: 要操作的表达式，取整数部分 函数语法使用时返回操作后的结果，操作对象的值不变。

适用控制器	通用
例子	<p>例一 命令语法</p> <pre>VR(23)=0.333 SET_BIT(0,23) 'VR(23)的第 0 位将置为 1，并清除小数部分 ?VR(23) '结果为 1</pre> <p>例二 函数语法</p> <pre>DIM a,b a=0.333 b=0 b=SET_BIT(0,a) '设置 a 的第 0 位的，结果赋值到 b，并清除小数 PRINT a,b '打印结果 0.333,1, a 没有改变, b 为 1</pre>
相关指令	CLEAR_BIT , READ_BIT , READ_BIT2

CLEAR_BIT -- 按位置 0

类型	数学指令或函数
描述	位操作，只对整数，修改对应位为 0。 分为命令语法和函数语法。
语法	<p>命令语法: <code>CLEAR_BIT(bit#,vr#)</code> 直接操作 VR 寄存器</p> <p>bit#: 位编号: 0-31</p> <p>vr#: 要操作的 VR 变量编号，整数部分</p> <p>命令语法使用时没有返回值，直接操作，修改操作对象的值。</p> <p>函数语法: <code>ret = CLEAR_BIT(bit#,int)</code></p> <p>ret: 操作结果</p> <p>bit#: 位编号: 0-31</p> <p>int: 要操作的表达式，取整数部分</p> <p>函数语法使用时返回操作后的结果，操作对象的值不变。</p>
适用控制器	通用
例子	<p>例一 命令语法</p> <pre>VR(23)=3.333 CLEAR_BIT(0,23) 'VR(23)的第 0 位将被清除(设置为 0) ?VR(23) '打印结果 2，小数被去除</pre> <p>例二 函数语法</p> <pre>DIM a,b a=3.333 b=0 b= CLEAR_BIT(0,a) '返回清除 a 的第 0 位和小数后的结果给 b PRINT a,b '结果为 3.333,2, a 不变,b 为 2</pre>
相关指令	SET_BIT , READ_BIT , READ_BIT2

READ_BIT -- 按位读取

类型	数学函数
描述	位操作，只对整数，读取对应位状态。 只能操作 VR 寄存器，非 VR 参考 READ_BIT2
语法	ret = READ_BIT(bit#, vr#) ret: 读取结果: 1 或 0 bit#: 位编号: 0-31 vr#: 要操作的 VR 变量编号
适用控制器	通用
例子	VR(23)=3.333 PRINT READ_BIT(0,23) '读取 VR(23)的第 0 位，结果为 1
相关指令	SET_BIT , CLEAR_BIT , READ_BIT2

READ_BIT2 -- 按位读取 2

类型	数学函数
描述	位操作，只对整数，读取对应位状态。
语法	ret = READ_BIT2(bit#, int) ret: 读取结果: 1 或 0 bit#: 位编号: 0-31 int: 要操作的表达式，取整数部分
适用控制器	通用，20130813 以后的固件版本提供了支持
例子	DIM a,b b=1.64 a=READ_BIT2(0,b) '读取 b 的第 0 位,赋值给 a PRINT a '输出 a 值，结果为 1
相关指令	SET_BIT , , READ_BIT

FRAC -- 返回小数

类型	数学函数
描述	返回小数部分，总是大于 0 的部分。
语法	FRAC(expression) expression: 要操作的数
适用控制器	通用
例子	a=FRAC(1.235)

	PRINT a	'结果是 0.235
--	---------	------------

INT -- 返回整数

类型	数学函数
描述	返回整数部分。
语法	INT(expression) expression: 任意有效的表达式
适用控制器	通用
例子	a=INT(1.235) PRINT a '结果是 1 ?INT(-1.1) '打印结果, -2, 因为小数部分总处理为正数。


SGN -- 返回符号

类型	数学函数
描述	返回符号。 1 大于 0 0 等于 0 -1 小于 0
语法	SGN(expression) expression: 任意有效的表达式
适用控制器	通用
例子	a=SGN(-1.235) PRINT a '结果是-1

IEEE_IN -- 组合浮点数

类型	数学函数
描述	把四个字节组合成一个单精度浮点数。
语法	IEEE_IN(byte0,byte1,byte2,byte3) byte0 - byte3: 四个字节
适用控制器	通用
例子	VAR=IEEE_IN(VR(10),VR(11),VR(12),VR(13)) '将 VR(10)~VR(13)这四个数据合成一个单精度浮点数

IEEE_OUT -- 提取单字节

类型	数学函数												
描述	从一个单精度浮点数里面取一个字节。												
语法	<pre>byte_n = IEEE_OUT(var, n)</pre> <p>var: 单精度浮点数 N: 0-3, 取第几个字节</p>												
适用控制器	通用												
例子	<p>例一 VAR = IEEE_OUT(VR(1),2) '提取 VR(1)的第二个字节</p> <p>例二 GLOBAL VAR0,VAR1,VAR2,VAR3 VAR0=0 VAR1=0 VAR2=0 VAR3=0 VR(1)=123.456 VAR0 = IEEE_OUT(VR(1),0) VAR1 = IEEE_OUT(VR(1),1) VAR2 = IEEE_OUT(VR(1),2) VAR3 = IEEE_OUT(VR(1),3) VR(2)=0 VR(2)=IEEE_IN(VAR0,VAR1,VAR2,VAR3) 运行结果:</p>  <table border="1"> <thead> <tr> <th>监视内容</th><th>值</th></tr> </thead> <tbody> <tr> <td>var0</td><td>66</td></tr> <tr> <td>var1</td><td>246</td></tr> <tr> <td>var2</td><td>233</td></tr> <tr> <td>var3</td><td>121</td></tr> <tr> <td>vr(2)</td><td>123.4560</td></tr> </tbody> </table>	监视内容	值	var0	66	var1	246	var2	233	var3	121	vr(2)	123.4560
监视内容	值												
var0	66												
var1	246												
var2	233												
var3	121												
vr(2)	123.4560												

\$ -- 16 进制

类型	特殊字符
描述	表示紧接着的数据是 16 进制格式。
语法	\$hexnum
适用控制器	通用
例子	<p>在线命令输入 >>PRINT \$F</p>

输出: 15

9.7. 字符串操作指令

CHR -- ASCII 码打印

类型	字符串函数
描述	返回 ASCII 打印，只用于 PRINT。
语法	CHR(expression) expression: 任意有效的表达式
适用控制器	通用
例子	在线命令输入 >>PRINT CHR(66) 输出: B

HEX -- 16 进制打印

类型	字符串函数
描述	返回十六进制格式，只用于 PRINT。
语法	HEX(expression) expression: 任意有效的表达式，只取整数部分
适用控制器	通用
例子	在线命令输入 >>PRINT HEX(15) 输出: f '十六进制

STRLEN -- 返回字符串长度

类型	字符串函数
描述	返回字符串长度。
语法	len=STRLEN(str) str: 字符串
适用控制器	通用
例子	DIM str_a(20) str_a="len123" ?STRLEN(str_a) 打印结果: 6

TOSTR -- 格式化输出

类型	字符串函数
描述	格式化输出函数，变量转换成字符串。
语法	<p>TOSTR(VAR1, [N],[DOT])</p> <p>VAR1: 任意有效的表达式</p> <p>N: 输出数据总位数，包括小数点和符号位。当 N 设置负数时，表示右对齐</p> <p>DOT: 输出的小数个数，当 N 太小没有小数位置时，不输出小数</p> <p>输出的是字符串类型。只有第一个参数默认打印到小数点后四位。</p>
适用控制器	通用
例子	<p>例一</p> <p>在线命令输入</p> <p>>> PRINT TOSTR(2-100,6,2)</p> <p>输出: -98.00</p> <p>例二</p> <p>DIM aa(20)</p> <p>aa="asd13"+TOSTR(354)</p> <p>?aa '打印结果 asd13354.0000</p>

STRCOMP -- 字符串比较

类型	字符串函数
描述	字符串比较函数，根据两个字符串的情况返回>0、=0、<0。 比较长度在 500 字节内，否则返回值出错误。
语法	<p>STRCOMP(str1, str2)</p> <p>str1: 字符串 1</p> <p>str2: 字符串 2</p>
适用控制器	通用
例子	<p>DIM AAA(10)</p> <p>AAA = "abc"</p> <p>在线命令输入</p> <p>>>PRINT STRCOMP(AAA, "abc")</p> <p>输出: 0</p>

STRFIND -- 字符串搜索

类型	字符串函数
描述	字符串搜索函数。
语法	STRFIND(str1, str2 [, firstindex]) str1: 待搜索的字符串 str2: 搜索模板字符串 firstindex: 从哪个位置开始搜索, 缺省 0 返回: ≥ 0 , 返还查找到的 index; < 0 , 没有找到
适用控制器	通用
例子	DIM AAA(10),BBB(3) AAA="AD23GF41" BBB="23G" ?STRFIND(AAA,BBB) '打印搜索到的索引位置, 2

VAL -- 字符转数值

类型	字符串函数
描述	字符串转换为数值。 只能转换数字字符, 遇到字母、符号字符停止。
语法	VAL(str1) str1: 字符串
适用控制器	通用
例子	例一 VAR1 = VAL("123") ?VAR1 '打印结果, 123 例二 VAR2 = VAL("123QWE23") ?VAR2 '打印结果, 123

9.8. 常数指令

PI -- 圆周率

类型	常数
值	3.14159
适用控制器	通用

TRUE -- 真值

类型	常数
值	-1
适用控制器	通用

FALSE -- 假值

类型	常数
值	0
适用控制器	通用

ON -- 开启

类型	常数
值	1
适用控制器	通用

OFF -- 关闭

类型	常数
值	0
适用控制器	通用

9.9. 高级运算指令

CRC16 -- CRC 检验计算

类型	数学函数
描述	CRC16 CCITT 计算。
语法	CRC16(arrayname, index, size[, initial] [, poly]) arrayname: 数据存储所在的数组，一个字节占一个位置

	<p>index: 数据存储所在的数组起始索引</p> <p>size: 计算字节数</p> <p>initial: CRC 计算初始值, 缺省\$FFFF</p> <p>poly: 多项式, 暂时只支持 modbus 的\$A001 和 CCITT 的\$1021, 缺省\$A001</p>
适用控制器	通用
例子	<p>TABLE(0, \$FE, \$48, \$06, \$00, \$6D, \$00, \$00, \$00) '8 个数据存储存储在 TABLE</p> <p>CRCVALUE = CRC16(TABLE, 0, 8) '计算 CRC, 结果\$1A0D</p> <p>TABLE(8)= CRCVALUE\256 '计算的 CRC 加在数据的后面, 大端模式</p> <p>TABLE(9)= CRCVALUE AND \$FF</p>

DTSMOOTH -- table 平滑

类型	数学函数
描述	对 TABLE 存储的点坐标进行平滑调整。
语法	<p>DTSMOOTH (axis, dtfirst, space, points, imode, refradius)</p> <p>axis: 轴数</p> <p>dtfirst: 第一个点的 TABLE 索引</p> <p>space: 两个点之间的索引间隔(就是一个点存储占用的空间)</p> <p>points: 总点个数</p> <p>imode: 0- 绝对方式, 对曲率半径小于参考值的进行调整</p> <p>referradius: 参考曲率半径, 可以根据 半径=(速度平方)/拐弯加速度来计算参考</p>
适用控制器	<p>3X 系列 20161206 以上固件支持</p> <p>4 系列 20170508 以上固件支持</p>
例子	<p>TABLE(0, 0, 0)</p> <p>TABLE(5, 99, 0)</p> <p>TABLE(10, 100, 0)</p> <p>TABLE(15, 100, 1)</p> <p>TABLE(20, 101, 1)</p> <p>TABLE(25, 200, 1)</p> <p>DTSMOOTH(2, 0, 5, 6, 0, 5)</p> <p>?*TABLE(0, 2)</p> <p>?*TABLE(5, 2)</p> <p>?*TABLE(10, 2)</p> <p>?*TABLE(15, 2)</p> <p>?*TABLE(20, 2)</p> <p>?*TABLE(25, 2)</p>

B_SPLINE -- B 样条平滑

类型	数学函数
----	------

描述	将 TABLE 中的数据进行 B 样条平滑。
语法	<p>B_SPLINE(type, data_start, points, data_out, ratio)</p> <p>type: 类型, 目前只支持 1-B 样条</p> <p>data_start: 图形数据在 TABLE 中的起始位置</p> <p>points: 图形数据的个数</p> <p>data_out: 平滑后的图形数据在 TABLE 中起始位置</p> <p>ratio: B_SPLINE 函数的平滑比率, 平滑后的个数为 points * ratio</p> <p>新增加样条控制点的自动计算功能, 此功能配合 MOVESPLINE 样条曲线运动使用, 4 系列以上产品支持, 4 系列控制器固件版本 20170621</p> <p>B_SPLINE(type, axes, dtstartpos, dtendpos, dtlastpos, dtnexpos, dtoutcontrol1, dtoutcontrol2)</p> <p>type:</p> <ol style="list-style-type: none"> 1 兼容原来的功能 11 为连续线段的第一条线段计算样条拟合的控制点. 12 为连续线段的中间线段计算样条拟合的控制点. 13 为连续线段的最后一条线段计算样条拟合的控制点. <p>axes: 参与样条插补的轴数</p> <p>dtstartpos: 线段起点坐标位于的 table 数组索引, 多个轴连续存储不同的 table, 下同</p> <p>dtendpos: 线段终点坐标位于的 table 数组索引</p> <p>dtlastpos: 线段起点的前面点的坐标索引, 用于计算参考, 第一条线段此参数无用</p> <p>dtnexpos: 线段终点的后面点的坐标索引, 用于计算参考, 最后一天线段此参数无用</p> <p>dtoutcontrol1: 输出样条的控制点数据, 贝塞尔的第 1 个控制点(起点也作为控制点除外)</p> <p>dtoutcontrol2: 输出样条的控制点数据, 贝塞尔的第 2 个控制点</p> <p>起点、dtoutcontrol1、dtoutcontrol2、终点一起构成贝塞尔的 4 个控制点。</p>
适用控制器	通用
例子	<p>例一 type1</p> <p>B_SPLINE(1,0,10,100,10) '平滑一个 10 点的图形数据, 源图形点在 table 的位置为从 0 到 9, 平滑为 100 点的数据, 并且平滑后的数据从 table 地址 100 开始存放</p> <p>例二 新增模式</p> <p>TABLE(0,0,0,0,100,100,100,200,100) 'XY 两轴连续 4 点的坐标数据</p> <p>B_SPLINE(11, 2, 0, 2, -1, 4, 100,200) '第 1 条线段</p> <p>?TABLE(100),TABLE(101),TABLE(200),TABLE(201)</p> <p>B_SPLINE(12, 2, 2, 4, 0, 6, 100,200) '第 2 条线段</p> <p>?TABLE(100),TABLE(101),TABLE(200),TABLE(201)</p> <p>B_SPLINE(13, 2, 4, 6, 2, 6, 100,200) '第 3 条线段</p> <p>?TABLE(100),TABLE(101),TABLE(200),TABLE(201)</p>

TURN_POSMAKE -- 旋转坐标计算

类型	数学函数
描述	旋转功能的计算函数，计算旋转台上点(X,Y)旋转 R 度后点的绝对坐标，旋转的正向与 XY 的正向要一致（右手法则）。
语法	TURN_POSMAKE(tablenum, posx, posy, disR, tableout) tablenum: 旋转功能参数存储 table 编号 posx: X 方向的坐标 posy: Y 方向的坐标 disR: 旋转轴的相对偏移 tableout: 存储计算后的坐标
适用控制器	通用
相关指令	MCIRC TURNABS

ZCUSTOM -- 运动参数计算

类型	数学函数												
描述	计算各种运动指令中的参数，详细功能请看下方语法功能描述。												
语法	<p>功能 2：计算空间圆弧或直线上一定距离后的点的位置。 Table 表参数按三点画圆模式填写其他两个点参数。 填写相对坐标，返回也是相对的位置。</p> <p>语法：ZCUSTOM(2,tableend,tablemid,tableout,mode,vectdis)</p> <p>tableend: 存储圆弧终点的 table 索引 tablemid: 存储圆弧中间点的 table 索引，与当前点一起构成圆弧的 3 个点 tableout: 输出计算数据的 table 索引 mode: 模式</p> <table border="1"> <thead> <tr> <th>值</th><th>描述</th></tr> </thead> <tbody> <tr> <td>1</td><td>相对起点的空间圆弧弧长</td></tr> <tr> <td>2</td><td>相对终点的空间圆弧弧长</td></tr> <tr> <td>3</td><td>相对起点的直线距离，此时 tablemid 不起作用</td></tr> <tr> <td>4</td><td>相对终点的直线距离，此时 tablemid 不起作用</td></tr> <tr> <td>5</td><td>相对计算空间圆弧的圆心，此时 vectdis 无效。此时 tableout 依次输出 xyz 圆心，弧度范围，弧长</td></tr> </tbody> </table> <p>vectdis: 要计算的点相对 mode 的距离，负数表示往前。圆弧模式时，正数表示顺时针，负数表示逆时针</p> <p>功能 6：计算空间圆弧的起点和终点的切线角度方向，弧度单位。 填写相对坐标，返回也是相对的位置。</p> <p>语法：ZCUSTOM(6,tableend,tablemid,tableout)</p> <p>tableend: 存储圆弧终点的 table 索引 tablemid: 存储圆弧中间点的 table 索引，与当前点一起构成圆弧的 3 个点</p>	值	描述	1	相对起点的空间圆弧弧长	2	相对终点的空间圆弧弧长	3	相对起点的直线距离，此时 tablemid 不起作用	4	相对终点的直线距离，此时 tablemid 不起作用	5	相对计算空间圆弧的圆心，此时 vectdis 无效。此时 tableout 依次输出 xyz 圆心，弧度范围，弧长
值	描述												
1	相对起点的空间圆弧弧长												
2	相对终点的空间圆弧弧长												
3	相对起点的直线距离，此时 tablemid 不起作用												
4	相对终点的直线距离，此时 tablemid 不起作用												
5	相对计算空间圆弧的圆心，此时 vectdis 无效。此时 tableout 依次输出 xyz 圆心，弧度范围，弧长												

tableout: 输出计算数据的 table 索引, 依次输出:起点 xy 方向, 起点 Z 方向, 终点 xy 方向, 终点 Z 方向, 角度弧度单位

功能 7: 输入速度比例, 计算 MOVESLINK 的从轴位置, 主轴位置。

语法: ZCUSTOM(7, distance, link dist, start sp, end sp, 速度比例, tableout)

distance: 从连接开始到结束, 跟随轴移动的距离, 采用 units 单位

link dist: 参考轴在连接的整个过程中移动的绝对距离, 采用 units 单位

start sp: 启动时跟随轴和参考轴的速度比例, units/units 单位, 负数表示跟随轴负向运动

end sp: 结束时跟随轴和参考轴的速度比例, units/units 单位, 负数表示跟随轴负向运动

速度比例: 需要计算的点的速度比例, 与参数里面的起始结束点比例意义相同

tableout: 正向找的从轴距离, 正向找的主轴距离, 从反向找的从轴距离, 反向找的主轴距离, 占用 4 个 TABLE (一段曲线, 速度比例可能会有多个解)

功能 8: MOVESLINK 输入从轴位置, 反算主轴的位置。

语法: ZCUSTOM(8,distance,link dist,start sp,end sp,distancemoved, tableout)

distance: 从连接开始到结束, 跟随轴移动的距离

link dist: 从连接开始到结束, 主轴移动的绝对距离

start sp: 起始脉冲速度比例

end sp: 结束脉冲速度比例

distancemoved: 从轴已经运动距离

tableout: 输出 table 索引, 输出对应主轴的位置, 如多个解返回第一个

功能 9: FLEXLINK 输入从轴位置, 反算主轴的位置。

语法: ZCUSTOM (9, base_dist, excite_dist, link_dist, base_in, base_out, excite_acc, excite_dec, distancemoved, tableout)

base_dist: 跟随轴匀速运动距离

excite_dist: 跟随轴激励运动距离, 这个与 base_dist 相反的时候无法反算

link_dist: 整个指令过程, 跟随轴运动完成, 主轴移动的距离

base_in: 激励开始前, 跟随轴运动距离占 base_dist 的百分比

base_out: 激励完成后, 跟随轴剩余运动距离占 base_dist 的百分比, 两者相加不要超过 100%

excite_acc: 激励过程中, 跟随轴加速阶段运动距离占 excite_dist 的百分比, excite_dist 为负值时, 为减速阶段

excite_dec: 激励过程中, 跟随轴减速阶段运动距离占 excite_dist 的百分比, excite_dist 为负值时, 为加速阶段

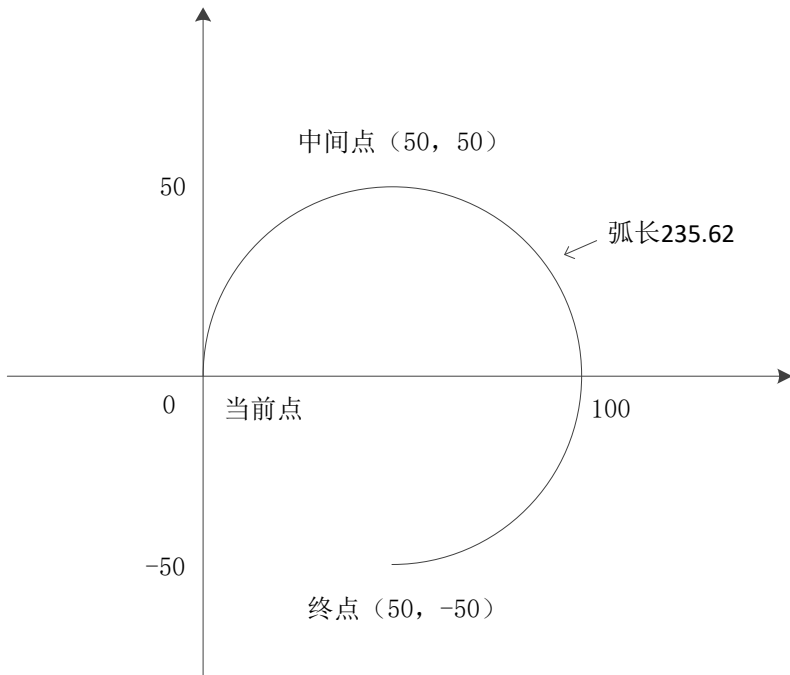
distancemoved: 从轴已经运动脉冲数

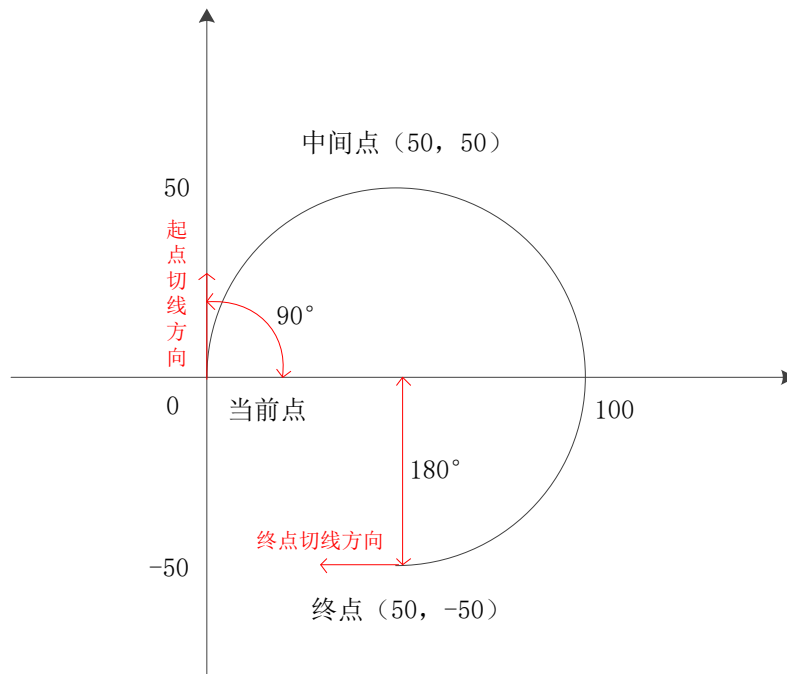
tableout: 输出 Table 索引, 输出对应主轴的位置, 如多个解返回第一个

功能 10: 从工件坐标上的三点在原来坐标系上的坐标来计算 FRAME_ROTATE 旋转转换的参数, 每个点要存储 xyz 的三个方向的坐标。

语法: ZCUSTOM (10, dtzero, dtx, dty, dtout)

dtzero: 工件零点在原来坐标系上的位置

	<p>dtx: 工件坐标系 X 轴上的点在原来坐标系上的位置</p> <p>dty: 工件坐标系 Y 轴上的点在原来坐标系上的位置</p> <p>dtout: 输出 TABLE 索引, 分别存储: X, Y, Z, RX, RY, RZ</p>
适用控制器	通用
例子	<p>例一 功能 2 使用</p>  <p>TABLE(0,50,-50,0) '设置终点坐标, 相对位置</p> <p>TABLE(3,50,50,0) '设置中间点坐标, 相对位置</p> <p>'模式 1, 相对起点</p> <p>ZCUSTOM(2,0,3,10,1,78.54) '相对起点位置顺时针弧长为 78.54 的圆上点</p> <p>?TABLE(10),TABLE(11),TABLE(12) '输出相对坐标为 50,50,0</p> <p>ZCUSTOM(2,0,3,10,1,-78.54) '相对起点位置逆时针弧长为 78.54 的圆上点</p> <p>?TABLE(10),TABLE(11),TABLE(12) '输出相对坐标为 50,-50,0</p> <p>'模式 2, 相对终点</p> <p>ZCUSTOM(2,0,3,10,2,78.54) '相对终点位置顺时针弧长为 78.54 的圆上点</p> <p>?TABLE(10),TABLE(11),TABLE(12) '输出坐标为 50,-50,0</p> <p>ZCUSTOM(2,0,3,10,2,-78.54) '相对终点位置逆时针弧长为 78.54 的圆上点</p> <p>?TABLE(10),TABLE(11),TABLE(12) '输出坐标为 100,0,0</p> <p>'模式 5, 计算此时三点画圆的圆心、弧度、弧长</p> <p>ZCUSTOM(2,0,3,10,5,0) '此时距离参数不起作用</p> <p>?TABLE(10),TABLE(11),TABLE(12) '输出坐标为 50,0,0</p> <p>?TABLE(13),TABLE(14) '输出弧度 4.712, 弧长 235.62</p> <p>例二 功能 6, 计算圆弧起点和终点的切线弧度</p>



```
TABLE(0,50,-50,0)      '设置终点坐标, 相对位置
TABLE(3,50,50,0)       '设置中间点坐标, 相对位置
ZCUSTOM(6,0,3,10)      '计算当前点和终点切线的弧度
?TABLE(10),TABLE(11)   '输出当前点 1.571,0 (1.571=90*PI/180)
?TABLE(12),TABLE(13)   '输出终点-3.142,0 (-3.142=-180*PI/180)
```

例三 功能 8, 反算 MOVESLINK 主轴位置

```
BASE(0,1)
```

```
DPOS=0,0
```

```
UNITS=100,100
```

```
SPEED=100,100
```

```
ACCEL=1000,1000
```

```
TRIGGER
```

```
MOVESLINK(50,100,0,1,1) '建立 MOVESLINK 连接
```

```
MOVEABS(100) AXIS(1)
```

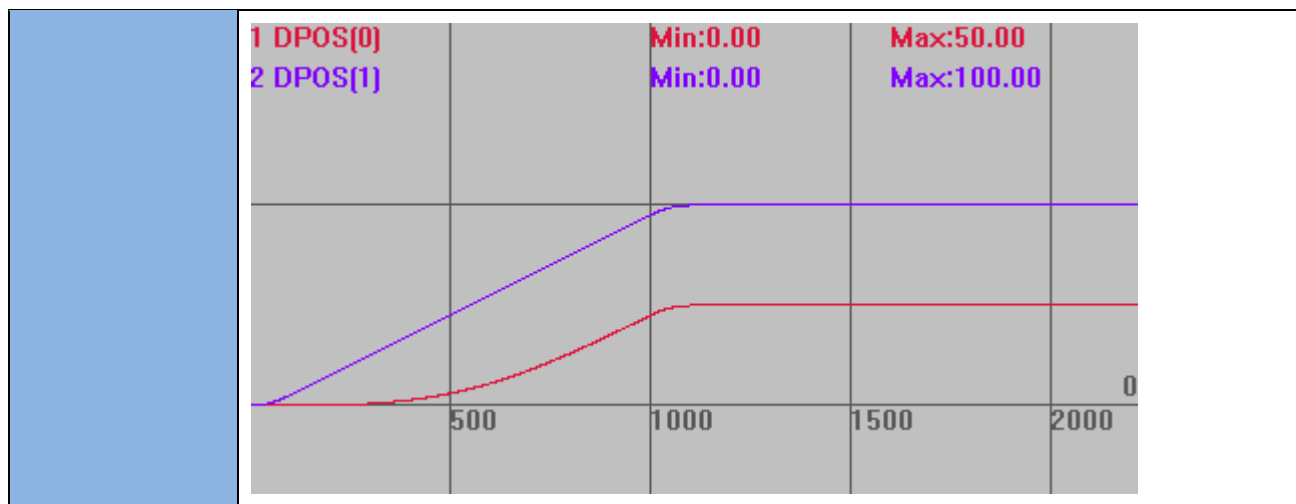
```
ZCUSTOM(8,50,100,0,1,25,10) '根据建立连接的参数, 计算从轴运动 25 时主轴的位置
```

```
?TABLE(10)                '输出主轴位置 73.801
```

运动轨迹

```
DPOS(0)垂直刻度 100
```

```
DPOS(1)垂直刻度 100
```



ZMATH64 -- 64 位计算

类型	64 位计算指令																																																							
描述	<p>对存储 D 寄存器(MODBUS 寄存器)的 64 位数进行计算。</p> <p>一个 64 位有符号整数占用 4 个寄存器(小端模式)。</p> <p>只操作 MODBUS 寄存器，不处理 VR 映射等。</p> <p>4 系列控制器 20170629 固件以上版本支持。</p>																																																							
语法	<p>ZMATH64(opmode, dindex1, dindex2)</p> <p>opmode: 操作编号</p> <p>dindex1、dindex2: MODBUS 寄存器编号</p> <table border="1"> <thead> <tr> <th>操 作 编 号</th><th>执行操作</th><th>说明</th></tr> </thead> <tbody> <tr> <td>1</td><td>64 位整数加法</td><td>D64(dindex1) += D64(dindex2)</td></tr> <tr> <td>2</td><td>64 位整数减法</td><td>D64(dindex1) -= D64(dindex2)</td></tr> <tr> <td>3</td><td>64 位整数乘法</td><td>D64(dindex1) *= D64(dindex2)</td></tr> <tr> <td>4</td><td>64 位整数除法</td><td>D64(dindex1) /= D64(dindex2)</td></tr> <tr> <td>5</td><td>64 位整数余</td><td>D64(dindex1) %= D64(dindex2)</td></tr> <tr> <td></td><td></td><td></td></tr> <tr> <td>11</td><td>64 位整数读取</td><td>D64(dindex1) = D32(dindex2)</td></tr> <tr> <td>12</td><td>64 位整数转换</td><td>D32(dindex1) = D64(dindex2)</td></tr> <tr> <td>13</td><td>64 位整数读取</td><td>D64(dindex1) = D32IEEE(dindex2)</td></tr> <tr> <td>14</td><td>64 位整数转换</td><td>D32IEEE(dindex1) = D64(dindex2)</td></tr> <tr> <td></td><td></td><td></td></tr> <tr> <td>15</td><td>赋值</td><td>D64(dindex1) = double64(dindex2)</td></tr> <tr> <td>16</td><td>赋值</td><td>double64(dindex1) = D64(dindex2)</td></tr> <tr> <td>17</td><td>赋值</td><td>double64 (dindex1) = D32IEEE (dindex2)</td></tr> <tr> <td>18</td><td>赋值</td><td>D32IEEE (dindex1) = double64 (dindex2)</td></tr> <tr> <td></td><td></td><td></td></tr> <tr> <td>21</td><td>double 加法</td><td>double64 (dindex1) += double64 (dindex2)</td></tr> </tbody> </table>		操 作 编 号	执行操作	说明	1	64 位整数加法	D64(dindex1) += D64(dindex2)	2	64 位整数减法	D64(dindex1) -= D64(dindex2)	3	64 位整数乘法	D64(dindex1) *= D64(dindex2)	4	64 位整数除法	D64(dindex1) /= D64(dindex2)	5	64 位整数余	D64(dindex1) %= D64(dindex2)				11	64 位整数读取	D64(dindex1) = D32(dindex2)	12	64 位整数转换	D32(dindex1) = D64(dindex2)	13	64 位整数读取	D64(dindex1) = D32IEEE(dindex2)	14	64 位整数转换	D32IEEE(dindex1) = D64(dindex2)				15	赋值	D64(dindex1) = double64(dindex2)	16	赋值	double64(dindex1) = D64(dindex2)	17	赋值	double64 (dindex1) = D32IEEE (dindex2)	18	赋值	D32IEEE (dindex1) = double64 (dindex2)				21	double 加法	double64 (dindex1) += double64 (dindex2)
操 作 编 号	执行操作	说明																																																						
1	64 位整数加法	D64(dindex1) += D64(dindex2)																																																						
2	64 位整数减法	D64(dindex1) -= D64(dindex2)																																																						
3	64 位整数乘法	D64(dindex1) *= D64(dindex2)																																																						
4	64 位整数除法	D64(dindex1) /= D64(dindex2)																																																						
5	64 位整数余	D64(dindex1) %= D64(dindex2)																																																						
11	64 位整数读取	D64(dindex1) = D32(dindex2)																																																						
12	64 位整数转换	D32(dindex1) = D64(dindex2)																																																						
13	64 位整数读取	D64(dindex1) = D32IEEE(dindex2)																																																						
14	64 位整数转换	D32IEEE(dindex1) = D64(dindex2)																																																						
15	赋值	D64(dindex1) = double64(dindex2)																																																						
16	赋值	double64(dindex1) = D64(dindex2)																																																						
17	赋值	double64 (dindex1) = D32IEEE (dindex2)																																																						
18	赋值	D32IEEE (dindex1) = double64 (dindex2)																																																						
21	double 加法	double64 (dindex1) += double64 (dindex2)																																																						

	22	double 减法	double64 (dindex1) -= double64 (dindex2)
	23	double 乘法	double64 (dindex1) *= double64 (dindex2)
	24	double 除法	double64 (dindex1) /= double64 (dindex2)
	25	double 余, 求小数	double64 (dindex1) %= double64 (dindex2)
	<p>D32IEEE 表示浮点数存储, 同 MODBUS_IEEE。</p> <p>D64 表示 64 位有符号整数存储, 可以通过两个 MODBUS_LONG 来读取高 32 和低 32 位。</p>		
适用控制器	通用		
例子	<p>MODBUS_LONG(0)=100</p> <p>MODBUS_LONG(8)=20</p> <p>ZMATH64(1,8,0) '64 位整数加法计算, 两个数相加后存储在 MODBUS_LONG(8)起始地址</p> <p>?MODBUS_LONG(0) '打印结果, 100</p> <p>?MODBUS_LONG(8) '打印结果, 120</p>		
相关指令	<u>MODBUS_IEEE</u> , <u>MODBUS_LONG</u> , <u>MODBUS_REG</u>		

MODBUS_DOUBLE -- 读取 MODBUS

类型	64 位指令
描述	从 MODBUS 读取 double 数据, 可以赋值给其它变量数组。 3 系列和 3 系列以下等数组为 float 的不支持这个指令。
语法	MODBUS_DOUBLE(index) Index: modbus 寄存器编号
适用控制器	通用
例子	<p>MODBUS_LONG(0)=100</p> <p>MODBUS_LONG(8)=200</p> <p>ZMATH64(16, 0, 8) '64 位赋值</p> <p>?MODBUS_DOUBLE(0) '打印结果, 200</p> <p>?MODBUS_LONG(0) '打印结果, 0</p> <p>?MODBUS_LONG(8) '打印结果, 200</p>
相关指令	<u>ZMATH64</u>

第十章 输入输出相关指令

10.1. 输入相关指令

IN -- 输入口

类型	输入输出函数
描述	<p>读取输入，没有参数时返回 0-31 的状态。</p> <p>读取的是 INVERT_IN 翻转以后的状态。</p> <p>ZIO 扩展板的 IO 通道号与拨码有关，起始值为（16+拨码组合值*16），EIO 总线扩展 IO 使用 NODE_IO 指令，只能设置为 8 的倍数，详细查看硬件手册。</p> <p>注意 IO 映射编号要大于控制器自身最大的 IO 编号，不能与控制器的编号重合。</p>
语法	<p>IN([channel1],[channel2])</p> <p>channel1: 要读取的起始输入通道</p> <p>channel2: 要读取的结束输入通道，没有结束通道时，返回单个通道的输入状态</p>
适用控制器	通用
例子	a=IN(1) '读取通道 1 的输入
相关指令	OP , INVERT_IN

AIN -- 模拟量输入

类型	输入输出函数
描述	<p>读取模拟输入，返回 AD 转换模块的刻度值。</p> <p>12 位刻度值范围 0~4095 对应 0~10V 电压。</p> <p>16 位刻度值范围 0~65536 对应 0~10V 电压。</p> <p>ZAIO 扩展板的 AD 通道号与拨码有关，起始值为（8+拨码组合值*8），ZMIO 总线扩展 AD 使用 NODE_AIO 指令只能设置为 8 的倍数，详细查看硬件手册。</p> <p>注意 AIO 映射编号要大于控制器自身最大的 AIO 编号，不能与控制器的编号重合。</p>
语法	<p>VAR=AIN(channel)</p> <p>channel: 模拟输入通道 0-127</p>
适用控制器	通用
例子	<p>a=AIN(1) '读取通道 1 的 AD</p> <p>a=AIN(1) *10 /4095 '通道 1 的电压值</p>
相关指令	AOUT

ZSIMU_IN -- 仿真 IN 输入

类型	仿真器专用指令
描述	模拟输入 IN 口的输入。
语法	ZSIMU_IN([ionum ,] value) ionum: 输入编号, 从 0 开始, 没有这个参数时输出 0-31 value: 输入状态
适用控制器	通用
例子	ZSIMU_IN(0,1) '仿真输入 0 有效
相关指令	IN

ZSIMU_AIN -- 仿真模拟量输入

类型	仿真器专用指令
描述	模拟模拟量输入口的输入。
语法	ZSIMU_AIN(ionum, value)
适用控制器	通用
例子	ZSIMU_AIN(0,1024) '仿真通道 0
相关指令	AIN

ZSIMU_ENCODER -- 仿真编码器输入

类型	仿真器专用指令
描述	模拟编码器输入口的输入。
语法	ZSIMU_ENCODER(axis num, value) axis num: 轴编号, 从 0 开始 value: ENCODER 的仿真值
适用控制器	通用
例子	ZSIMU_ENCODER(0,1024) 'ENCODER = 1024
相关指令	ENCODER

INVERT_IN -- 反转输入

类型	特殊指令
描述	反转输入状态, 可以读取判断是否有反转。
语法	INVERT_IN(channel, state), VAR1= INVERT_IN(channel)

	channel: 输入通道 state: ON/OFF
适用控制器	通用
例子	INVERT_IN(1,ON) 'ZMC 系列控制器输入 OFF 时认为有信号输入(ECI 系列控制器与之相反) FWD_IN(0)=1 '输入 IN1 作为轴 0 的正向限位信号
相关指令	IN

IN_SCAN -- 扫描输入变化

类型	输入输出函数
描述	<p>扫描输入变动，返回值 1(TRUE)-变动，0(FALSE)-没有变动。</p> <p>此函数必须固定不断的扫描，返回的是两次扫描之间的变动，可以通过 IN_EVENT 读取变动的具体情况，使用的是 INVERT_IN 翻转以后的状态。</p> <p>固件版本 20140214 以后的才提供这个支持，扫描范围有宽度限制。</p> <p>00x 系列控制器只能在单个任务中使用。</p>
语法	VAR1=IN_SCAN([channel1][,channel2]) channel1: 要读取的起始输入通道 channel2: 要读取的结束输入通道，没有结束通道时，扫描单个输入
适用控制器	通用
例子	<pre> WHILE 1 IF IN_SCAN(0,23) THEN '扫描 IN0-23 口电平变化 IF IN_EVENT(0) > 0 THEN 'IN0 上升沿触发 PRINT "IN0 UP", IN_BUFF(0) ELSEIF IN_EVENT(0) < 0 THEN 'IN0 下降沿触发 PRINT "IN0 DOWN", IN_BUFF(0) ENDIF ENDIF WEND </pre>
相关指令	IN_EVENT , SCAN_EVENT , IN_BUFF

IN_EVENT -- 读取输入变化

类型	输入输出函数
描述	<p>读取输入的变化情况。</p> <p>1-上升沿，-1-下降沿，0-没有变化。</p> <p>此函数必须与 IN_SCAN 配合使用。</p>
语法	VAR1 = IN_EVENT(IONUM)
适用控制器	通用

例子	参见 IN_SCAN
相关指令	IN_SCAN , SCAN_EVENT

SCAN_EVENT -- 检测变化

类型	输入输出函数
描述	<p>检测表达式的内容变化。</p> <p>OFF-ON 返回 1, ON-OFF 返回-1, 不变返回 0。</p> <p>不要在循环内或者多任务调用同一个 SUB 内的 SCAN_EVENT。</p> <p>150810 之后固件版本支持, 之前版本用 IN_EVENT 和 IN_SCAN。</p>
语法	<p>ret = SCAN_EVENT (expression)</p> <p>expression: 任意有效的表达式, 结果会转成 BOOL 类型</p>
适用控制器	通用
例子	<p>例一 输入信号扫描</p> <pre> WHILE 1 IF SCAN_EVENT(IN(0)) > 0 THEN 'IN0 上升沿触发 PRINT "IN0 ON" ELSEIF SCAN_EVENT(IN(0))<0 THEN 'IN0 下降沿触发 PRINT "IN0 OFF" ENDIF WEND </pre> <p>例二 寄存器、变量扫描</p> <pre> WHILE 1 IF SCAN_EVENT(TABLE(0)) > 0 THEN 'TABLE0 上升沿触发 PRINT "TABLE0 ON" ELSEIF SCAN_EVENT(TABLE(0))<0 THEN 'TABLE0 下降沿触发 PRINT "TABLE0 OFF" ENDIF WEND </pre> <p>在线命令操作 table(0), 打印相关结果</p>
相关指令	IN_SCAN , IN_EVENT

IN_BUFF -- 读取输入缓冲

类型	输入输出函数
描述	<p>读取 IN_SCAN 缓冲的当前输入, 没有参数时返回 0-31 的状态。</p> <p>读取的是 INVERT_IN 翻转以后的状态。</p>
语法	<p>IN_BUFF([channel1],[channel2])</p> <p>channel1: 要读取的起始输入通道, 必须是 IN_SCAN 的输入范围</p>

	channel2: 要读取的结束输入通道, 没有结束通道时, 返回单个通道的输入状态
适用控制器	通用
例子	参见 IN_SCAN
相关指令	IN_SCAN

INFILTER -- 输入口滤波

类型	系统参数
描述	本地输入口的滤波参数。 值越大, 滤波时间越长, 2-9, 缺省 2。
语法	VAR1 = INFILTER, INFILTER= expression
适用控制器	通用
例子	INFILTER= 5 '在干扰严重场合加大滤波时间'

10.2. 输出相关指令

OP -- 输出口

类型	输入输出指令和函数
描述	<p>输出或读取输出状态。</p> <p>当在表达式中使用时, 自动为函数语法。</p> <p>ZIO 扩展板的 IO 通道号与拨码有关, 起始值为 (16+拨码组合值*16), EIO 总线扩展 IO 使用 NODE_IO 指令, 只能设置为 8 的倍数, 详细查看硬件手册。</p> <p>注意 IO 映射编号要大于控制器自身最大的 IO 编号, 不能与控制器的编号重合。</p> <p>最多可操作 32 个输出口。</p>
语法	<p>OP([ionum],value)</p> <p>或 OP(ionum1, ionum2,value[,mask])</p> <p>或 OP([firstnum],[finalnum])</p> <p>ionum: 输出编号, 从 0 开始</p> <p>value: 输出状态, 多个输出口操作时按位来指明多个口状态</p> <p>ionum1: 要操作的第一个输出通道</p> <p>ionum2: 要操作的最后一个输出通道</p> <p>mask: 用位来指定哪些 IO 需要操作, 不填时第一个通道到最后一个通道都操作</p> <p>firstnum: 输出编号, 从 0 开始</p> <p>finalnum: 输出编号, 从 0 开始, 没有这个参数时, 读取单个输出口状态</p>
适用控制器	通用
例子	<p>例一 单个操作</p> <p>'翻转输出口 0'</p>

	<pre> IF OP (0) = ON THEN OP (0, OFF) ELSE OP (0, ON) ENDIF 例二 区域操作 OP(0,7,\$FF) 'bit0-bit7 全开 DELAY(1000) OP(0,7,0) OP(8,15,\$FF) 'bit8-bit15 全开 DELAY(1000) OP(8,15,0) OP(0,15,\$FFFF) 'bit0-bit15 全开 DELAY(1000) OP(0,15,0) OP(0,31,\$FFFFFFFF) 'bit0-bit31 全开 DELAY(1000) OP(0,31,0) </pre>
相关指令	READ_OP , MOVE_OP

AOUT -- 模拟量输出

类型	输入输出指令或函数
描述	<p>模拟通道输出。</p> <p>12 位刻度值范围 0~4095 对应 0~10V 电压。</p> <p>16 位刻度值范围 0~65536 对应 0~10V 电压。</p>
语法	<pre>AOUT(channel) = value</pre> <p>channel: 模拟输出通道 0-63</p> <p>扩展板 DA 通道号与拨码有关, 起始值为 (4 + 拨码组合值*4), 详细查看硬件手册。</p>
适用控制器	通用
例子	<pre> AOUT(1) = 0 '关闭输出 DA 通道 1 AOUT(1) = 4095 'DA1 口输出 10V 电压 </pre>
相关指令	AIN

READ_OP -- 读取输出口

类型	输入输出函数
----	--------

描述	读取输出状态。 同 OP，多个输出口操作时按位来指明多个口状态。
语法	READ_OP ([firstnum],[finalnum]) firstnum: 起始输出编号，从 0 开始 finalnum: 结束输出编号，从 0 开始，没有这个参数时，读取单个输出口的状态
适用控制器	通用
例子	翻转输出口 0 IF READ_OP (0) = ON THEN OP (0, OFF) ELSE OP (0, ON) ENDIF
相关指令	OP



10.3. PMW 控制指令

PWM_FREQ -- PWM 频率

类型	PWM 控制函数
描述	PWM 频率设置或读取。 PWM 只能通过设置占空比为 0 来关闭，不能通过设置 PWM 频率为 0 实现，不要将频率设为 0，PWM 频率一定要在 PWM 开关之前调整。
语法	PWM_FREQ (index, freq) 或 PWM_FREQ (index)=freq index: 编号，从 0 开始 freq: 频率，硬件 PWM 1M，软件 PWM 2k
适用控制器	支持 PWM 功能的控制器才支持此函数。
例子	PWM_FREQ (0)=1000 '1K 频率 ?PWM_FREQ (0)
相关指令	PWM_DUTY ， MOVE_PWM

PWM_DUTY -- pwm 占空比

类型	PWM 控制函数
描述	PWM 占空比设置或读取。 PWM 只能通过设置占空比为 0 来关闭，不能通过设置 PWM 频率为 0 实现，PWM 频率一定要在 PWM 开关之前调整。 占空比指有效电平占整个周期的比例。 一个周期中先输出有效电平，再输出无效电平。

	<p>占空比为0.5 </p> <p>减小占空比 </p> <p>PWM 的实际输出受输出口的控制，必须输出口打开，PWM 才能输出，否则输出被屏蔽掉。可以先开 PWM 功能，然后再开输出口，这样实现激光电源的首脉冲抑制功能。</p>
语法	<p><code>PWM_DUTY(index, duty)</code> <code>PWM_DUTY(index)=duty</code></p> <p>index: 编号，从 0 开始</p> <p>duty: 占空比，0-1，当设置 0 的时候，PWM 关闭</p>
适用控制器	支持 PWM 功能的控制器才支持此函数。
例子	<p><code>PWM_DUTY(0)=0.5</code></p> <p><code>?PWM_DUTY(0)</code></p> <p>打印结果 0.5</p>
相关指令	PWM_FREQ ， MOVE_PWM

