

电气信息工程丛书

IEC 61131-3

编程语言及应用基础

彭瑜 何衍庆 编著



机械工业出版社
CHINA MACHINE PRESS

更多专业资料请访问电建论坛

ISBN 978-7-111-25645-8

◎ 策划
时静

◎ 封面设计
旭洲企划 刘吉维

电气信息工程丛书

- ◇ PLC 编程及应用 (第3版)
- ◇ S7-300/400 PLC 应用技术 (第2版)
- ◇ CS/CJ 系列 PLC 应用基础及案例
- ◇ 单片机原理与应用技术
- ◇ 西门子人机界面(触摸屏)组态与应用技术
- ◇ Protel 99 SE 原理图与 PCB 设计及电路仿真
- ◇ Protel 2004 电路原理图及 PCB 设计
- ◇ VHDL 数字电路及系统设计
- ◇ Linux PowerPC 详解——核心篇
- ◇ 数字信号处理器原理、结构及应用基础——TMS320F28x
- ◇ 单片机应用及 C51 程序设计
- ◇ OrCAD 电路原理图设计与应用
- ◇ 基于 EDK 的 FPGA 嵌入式系统开发
- ◇ 欧姆龙 CP1H PLC 应用基础与编程实践
- ◇ Freescale 9S12 十六位单片机原理及嵌入式开发技术
- ◇ 嵌入式可配置实时操作系统 eCos 开发与应用 (第2版)
- ◆ IEC 61131-3 编程语言及应用基础

上架建议: 工业技术 / 电气工程

编辑热线: (010)88379753 88379739

地址: 北京市百万庄大街22号 邮政编码: 100037
联系电话: (010)88326294 网址: <http://www.cmpbook.com> (811门广网)
(010)88993821 E-mail: cmp@cmpbook.com
购书热线: (010)88379839 (010)88379641 (010)88379643

定价: 33.00元

ISBN 978-7-111-25645-8



9 787111 256458 >

更多专业资料请访问电建论坛

电气信息工程丛书

IEC 61131-3 编程 语言及应用基础

彭瑜 何衍庆 编著



机械工业出版社

本书共分7章,主要介绍了 PLCopen 组织和编程语言基本概念、标准编程语言的公用元素和程序组织单元、指令表和结构化文本的文本类编程语言、梯形图和功能块图的图形类编程语言、顺序功能表图编程语言和可编程控制器的基本应用,最后并用两个实例说明,可编程控制器的编程方法和注意事项。

本书可作为自动化和仪表专业以及相关专业本、专科学生的教材和编程语言的培训教材,还可以作为工矿企业工程设计人员、科研开发单位工程技术人员的重要参考资料。

图书在版编目(CIP)数据

IEC 61131-3 编程语言及应用基础/彭瑜,何衍庆编著. —北京:机械工业出版社,2009.1
(电气信息工程丛书)

ISBN 978 - 7 - 111 - 25645 - 8

I. 1… II. ①彭… ②何… III. 生产过程 - 自动控制系统 - 程序设计 IV. TP278

中国版本图书馆 CIP 数据核字(2008)第 186492 号

机械工业出版社(北京市百万庄大街 22 号 邮政编码 100037)

责任编辑:李馨馨

责任印制:邓 博

北京诚信伟业印刷有限公司印刷

2009 年 1 月第 1 版·第 1 次印刷

184mm×260mm·19.5 印张·484 千字

0001—4000 册

标准书号:ISBN 978 - 7 - 111 - 25645 - 8

定价:33.00 元

凡购本书,如有缺页、倒页、脱页,由本社发行部调换

销售服务热线电话:(010)68326294

购书热线电话:(010)88379639 88379641 88379643

编辑热线电话:(010)88379739

封面无防伪标均为盗版

前 言

PLC 在当今自动化技术和市场中的重要地位是毋庸置疑的。它的总体设计思想来源于实际生产需要——从硬件、软件以及 I/O 接口等多个方面保证可靠而方便地实现离散流程的逻辑控制和顺序控制的基本功能。显然,按控制的要求编制程序在实际生产中是非常重要的。本书专门就这一方面以 PLC 的编程语言国际标准 IEC 61131-3 为主线展开了详尽的阐述。

1993 年,IEC 颁布可编程控制器的国际标准 IEC 1131(以后改为 IEC 61131),将信息技术领域的先进思想和技术引入工业控制领域,弥补并克服了传统 PLC、DCS 等控制系统开放性差、兼容性差、应用软件可维护性差以及可再用性差等弱点。目前,IEC 61131 标准已在发达国家得到广泛应用,不符合该标准的产品已不被最终用户接受。

IEC 61131-3 编程语言标准已对整个控制领域造成巨大冲击。它不仅适用于 PLC 产品,而且适用于运动控制产品、DCS 和基于工业 PC 的软逻辑、SCADA 等,其市场领域正在不断扩大。采用符合 IEC 61131-3 标准的产品,已经成为工业控制领域的发展趋势。

我国可编程控制器硬件的开发和应用起步较晚,对国际标准编程语言的使用相对较好。1995 年,与国际标准等效的国家标准相继颁布,2006 年,IEC 61131-3 最新国际标准的中文对照版 GB/T 15969.3 出版。但对该标准及有关产品的推广工作还做得不够,许多技术人员至今仍不知道该国际标准和国家标准的存在,一些大专院校还未采用该标准进行教学。

2005 年,我国成立 PLCopen 中国组织(PC5),它标志着我国与国际标准接轨的决心。我国致力于开发具有自主知识产权的 IEC 61131-3 编程系统的工作所取得的长足进展,极大地促进和加快了我国自动化控制设备的发展。但标准的推广介绍、资料翻译、可编程控制器标准产品的各级认证和授权等还有大量工作要做。为推广标准编程语言及其应用,我们编写了本书。

本书完全按照 PLCopen 国际组织为 IEC 61131-3 标准的国际工程师培训所制定的培训大纲的要求编排,而详尽程度则远超过培训大纲的规定。在介绍文本类编程语言、图形类编程语言以及顺序功能图编程语言时,均给出了一定数量的应用实例,力求体现本书对实际应用的参考价值。本书对于顺序功能图语言的内容给予了较多的篇幅,希望能使读者进一步认识这种语言的结构性功能,尤其是它在编程设计的各个阶段可以发挥的作用,以及它与其他 4 种语言的互补关系,也专门有阐述。本书专门开辟编程举例一章,从对控制系统的功能要求出发,说明了按 IEC 61131-3 标准的软件模型的概念完整地完成任务设计的方法。

本书由彭瑜、何衍庆编著。本书的编写工作得到了 PLCopen 中国组织 PC5 的积极支持和帮助,得到了上海工业自动化仪表研究所和华东理工大学等单位的关心和支持,PLCopen、科维、一方梯队、贝加莱、施耐德、Rockwell 等组织和公司的有关技术人员为本书的编写提供了大量的资料和技术支持,谨在此一并表示诚挚谢意。在本书的编写过程中,编者参考了相关专业书籍和产品说明书,在此向有关作者和单位表示衷心感谢。

由于编者水平所限,错漏在所难免,恳请读者不吝指正。

编者

III

目 录

前言

第1章 概述	1
1.1 PLCopen	1
1.1.1 PLCopen 组织	1
1.1.2 认证等级	2
1.1.3 PLCopen 的工作	3
1.2 IEC 61131 标准	5
1.2.1 IEC 61131 的基本情况	5
1.2.2 IEC 61131-3 编程语言	6
1.2.3 标准编程语言的特点	9
第2章 公用元素和程序组织单元	13
2.1 软件模型、编程模型	13
2.1.1 软件模型	13
2.1.2 编程模型	22
2.2 公用元素	24
2.2.1 字符集	24
2.2.2 标识符	25
2.2.3 分界符	25
2.2.4 关键字	27
2.2.5 空格和注释	29
2.3 数据外部表示	30
2.3.1 数值文字	30
2.3.2 字符串文字	31
2.3.3 时间文字	32
2.4 数据类型	34
2.4.1 基本数据类型	34
2.4.2 一般数据类型	35
2.4.3 衍生数据类型	35
2.4.4 数据类型的允许取值范围和 初始化	38
2.4.5 衍生数据类型的应用准则	40
2.5 变量	43
2.5.1 变量的表示	43

2.5.2 变量的属性和初始化	45
2.6 程序组织单元	52
2.6.1 函数	52
2.6.2 功能块	75
2.6.3 程序	93
2.6.4 程序组织单元	97
第3章 文本类编程语言	99
3.1 文本类编程语言及其公用元素	99
3.1.1 文本类编程语言概述	99
3.1.2 文本类编程语言的公用元素	99
3.2 指令表编程语言	100
3.2.1 指令	100
3.2.2 函数和功能块	114
3.2.3 示例	119
3.3 结构化文本编程语言	125
3.3.1 结构化文本的表示	125
3.3.2 语句	129
3.3.3 示例	141
第4章 图形类编程语言	151
4.1 图形类编程语言的公用元素	151
4.1.1 线、模块和流向	151
4.1.2 网络求值和执行控制元素	152
4.2 梯形图编程语言	155
4.2.1 传统梯形图编程语言的缺点	155
4.2.2 梯形图的组成元素	157
4.2.3 梯形图的执行	165
4.2.4 示例	167
4.3 功能块图编程语言	173
4.3.1 功能块图图形符号和功能块的 组合	173
4.3.2 功能块图的编程和执行	175
4.3.3 示例	176
第5章 顺序功能表图编程语言	186

5.1 顺序功能表图的三要素	186	6.1.2 电动机的正、反转控制	235
5.1.1 基本概念	186	6.1.3 电动机的星-三角转换控制	237
5.1.2 步	187	6.1.4 三相步进电动机的相序控制	239
5.1.3 转换	192	6.1.5 电动机带反接制动电阻的可逆运转 反接制动控制	242
5.1.4 有向连线	196	6.2 定时器、计数器的编程	243
5.2 顺序功能表图的程序结构	197	6.2.1 电动机的定时运行	243
5.2.1 单序列结构	198	6.2.2 长定时器的实现	245
5.2.2 选择序列结构	198	6.2.3 信号发生器的实现	245
5.2.3 并行序列结构	198	6.2.4 计数器的应用	247
5.2.4 不安全序列和不可达序列 结构	199	6.3 比较函数的应用	250
5.3 顺序功能表图编程语言	199	6.3.1 旋转定位控制	250
5.3.1 顺序功能表图的进展	199	6.3.2 分度盘工位控制	252
5.3.2 顺序功能表图的兼容	210	6.3.3 自动增益控制	253
5.4 示例	210	6.4 运算函数的应用	254
5.4.1 冲压机控制系统	210	6.4.1 控制算法	254
5.4.2 交通信号控制系统	212	6.4.2 运算函数的应用	258
5.4.3 物料混合控制功能块	217	6.5 移位函数的应用	262
5.5 发挥 SFC 的潜力,改善当前 PLC 程序的开发实践	221	6.5.1 霓虹灯顺序点亮程序	262
5.5.1 程序开发过程中必须遵循的基本 方法	221	6.5.2 操作权限确认控制	266
5.5.2 采用 SFC 作为设计系统控制的 主要工具	222	6.5.3 长循环移位控制	267
5.5.3 重视 SFC 语言与其他 4 种编程语言的 互补关系	222	第 7 章 编程举例	268
5.5.4 SFC 与状态图的对应关系及 异同	224	7.1 火力发电厂蒸汽轮机驱动给水泵 的控制	268
5.5.5 SFC 与 Petri 网的关系及异同	226	7.1.1 过程简介	268
5.6 顺序功能表图程序的转换	227	7.1.2 设计方法	269
5.6.1 顺序功能表图程序中基本序列的 转换	227	7.1.3 控制问题分解	272
5.6.2 顺序功能表图程序中动作控制 功能块的转换	229	7.1.4 程序分解	276
第 6 章 基本应用	231	7.1.5 低层次功能块	279
6.1 电动机控制的编程	231	7.1.6 信号流	280
6.1.1 电动机起保停控制	231	7.2 分选器控制	281
		7.2.1 分选器过程简介	281
		7.2.2 程序设计和优化	281
		第 8 章 实验	287
		8.1 用梯形图和功能块图编程语言 编写程序	287
		8.2 用指令表和结构化文本编程	

语言编写程序.....	288	附录 A MULTIPROG 软件的使用.....	293
8.3 编写衍生功能块和调用功能块		附录 B 习题和思考题	302
的程序.....	290	B.1 填空题	302
8.4 用顺序功能表图编程语言编		B.2 选择题	303
写程序.....	291	B.3 编程和思考题	303
附录	293	参考文献	306

第1章 概 述

1.1 PLCopen

1.1.1 PLCopen 组织

1. PLCopen 组织的宗旨

PLCopen 组织是独立于制造商和产品的国际组织。它成立于 1992 年,总部在荷兰,在北美和日本等国家设有分支机构。它是一个致力于编程语言标准化的非赢利性国际化组织。目前,PLCopen 组织拥有分布在 21 个国家的 100 多个会员。

PLCopen 组织的宗旨是促进 PLC 兼容软件的开发和使用。其主要工作是支持、宣传和推广 IEC 61131-3 国际标准。它以解决与控制编程相关的主题和支持该领域内国际标准的使用为使命。其目标是使用户通过在众多程序开发环境中应用该标准,在不同品牌产品和不同类型控制器之间移植控制程序,实现互换。为此,该组织采用如下方法:

- 1) 采用 IEC 61131-3 国际标准的编程语言。
- 2) 接受 PLCopen 会员的委托,生产或采用符合 IEC 61131-3 标准的可编程控制器产品。
- 3) 市场推介。采用共同的市场策略,如举行展览会或专题研讨会等。
- 4) 支持国际标准化委员会的工作。
- 5) 支持国家标准化委员会工作,推广和介绍有关标准化的产品等。
- 6) 建立有关编程系统的基本级、符合级和可重复使用级的认证体系,由独立机构进行测试以执行必要的检验。

2. PLCopen 组织结构

PLCopen 组织结构如图 1-1 所示。

(1) 技术委员会(Technical Committee)

技术委员会下设 6 个分支机构。

1) TC1:与 IEC 组织合作,共同发展、提高和完善 IEC 61131-3 标准。主要工作包括对 IEC 61131-3 标准的勘误和修改、与 IEC 组织的合作和开发、对标准版本的修订等。

2) TC2:负责定义功能块的程序库,协调功能块的约定。目前主要进行运动控制库的工作,包括逻辑和运动的各种技术的集成等。

3) TC3:制定编程语言一致性的测试标准。检验不同编程系统是否真正具有开放性。由于 IEC 61131-3 仅提供制订编程语言一致性的基本规则,并没有为用户提供实际编程系统的导则。因此,该委员会制订一致性测试的定义并进行测试。编程系统的一致性测试包括对不同编程语言的 3 种不同等级的测试。

4) TC4:负责制订通信接口及应用交换格式等。包括通信界面、与附加软件的接口、应用交换格式、与 Profibus 和 CANopen 等现场总线的映像等。

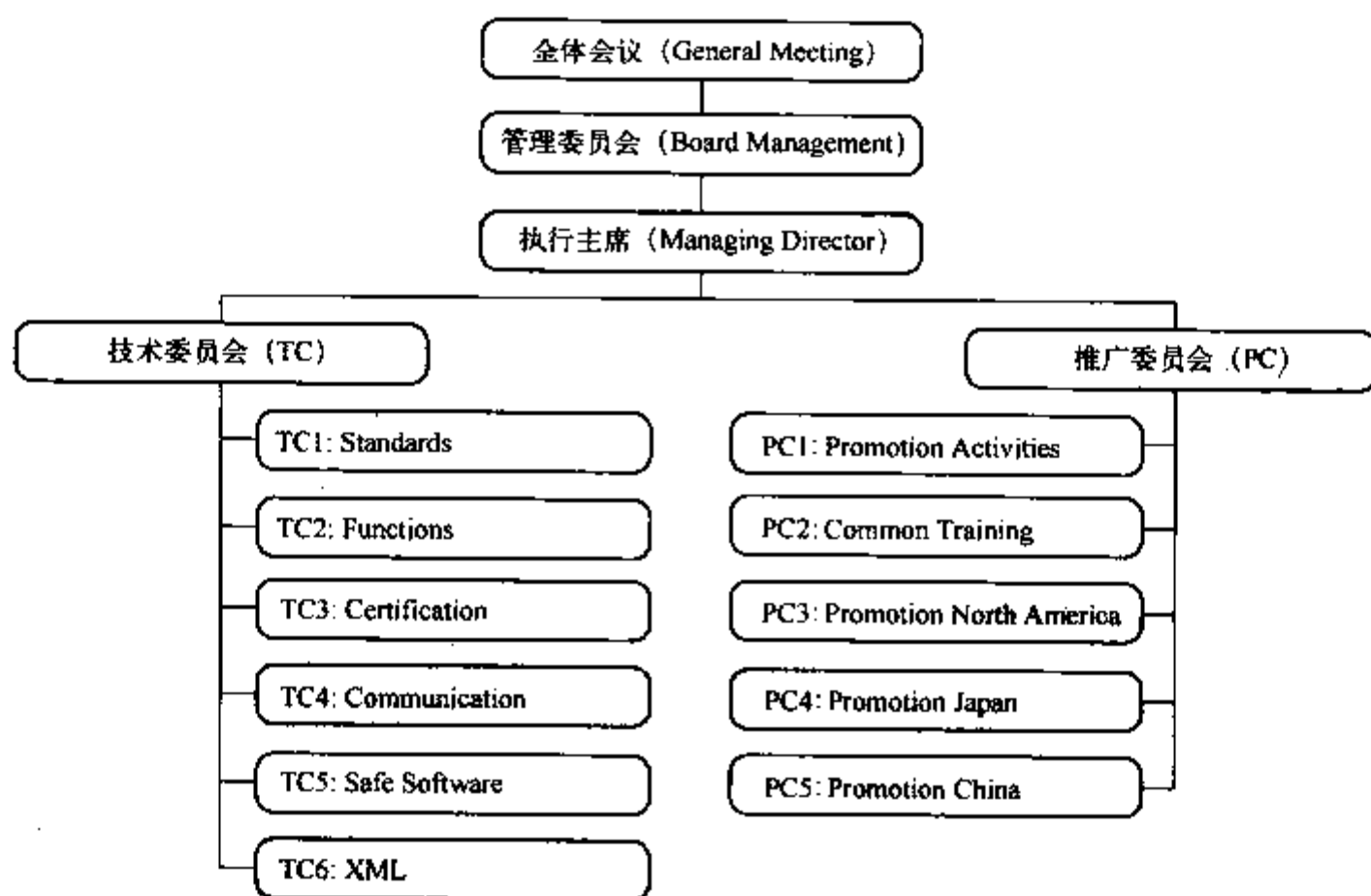


图 1-1 PLCopen 组成结构

5) TC5:负责制订安全软件导则,用 IEC 61508 支持安全编程技术。集中于安全相关系统的功能安全性研究,包括 IEC 标准的使用指南、安全运行的基础、与功能块的结合等。工作重点是 IEC 61508 和 IEC 61511 新安全标准的制定。

6) TC6:制定用于 IEC 61131-3 标准的 XML(Extended Markup Language)格式。它规定 IEC 61131-3 各种编程语言的 XML 格式,包括对图形视觉信息的表示、与其他开发工具的界面、转发功能块库分布等,它使编程环境成为开放系统环境。

(2) 推广委员会(Promotional Committee)

推广委员会下设 5 个分支机构。其任务包括公共培训、各地区的推广和介绍等事务。PC5 委员会主要管理与中国有关的推广和介绍等事务。

1.1.2 认证等级

为使产品等符合标准,设置了基本级、符合级和可重复使用级 3 个等级。图 1-2 所示是 3 个等级的标识。

1) 基本级(BL),表示由该产品开发的程序基本结构与 IEC 61131-3 兼容,能够提供编程语言的基本语言元素。目前,5 种编程语言的基本级定义已经完成,除梯形图编程语言外,其他编程语言的基本级测试软件也已完成。

2) 符合级(CL),表示产品符合 IEC 61131-3 标准的全部 26 种性能,所支持的数据类型与它的服务适配。符合级已经完成,并已经认证了第一批产品。

3) 可重复使用级(RL),表示编程单元函数和功能块可在不同的可编程控制器系统内重复使用,它通过结构化文本语言的简单文本格式转换实现。可重复使用级已经完成,也已经认证了第一批产品。26 种软件都已经被认证。

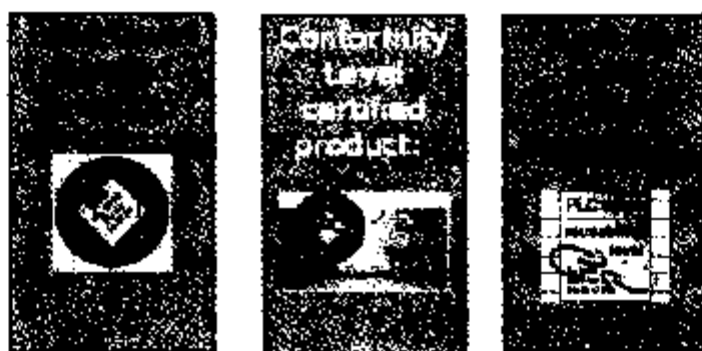


图 1-2 认证等级的标识

此外,对运动控制的产品和安全认证的产品,以及具有培训资质的单位也有相应的认证标识。

只有具有相应认证标识的产品和单位才具有相应的资质,并被用户所接受。

1.1.3 PLCopen 的工作

PLCopen 组织自 1992 年建立以来不懈地支持工业控制编程相关领域内国际标准的推广和发展。其最主要的成果之一就是构筑上述的编程软件包的开发环境;同时,还在这些编程系统的基础上进一步发展,为统一工程平台做了许多基础性的开创工作。这些工作大大推进了可编程控制器软件技术的发展。

为了让厂商能提供符合 IEC 标准的软件产品,又使用户容易辨别出符合 IEC 61131-3 的编程系统,PLCopen 组织开展了编程系统符合 IEC 标准的认证工作。在制定对编程系统进行符合 IEC 61131-3 程度的判据时,PLCopen 组织将它划分为 3 个等级,即基本级(Base Level, BL)、符合级(Conformity Level, CL)、可重复使用级(Reusability Level, RL)。不过,CL 级是和 RL 级组合使用的。如果能使 IL、ST、SFC、LD 和 FBD 这 5 种语言均达到 CL 级和 RL 级,则该编程系统就达到了全兼容、全开放的高度。

PLCopen 组织在 1996 年启动了运动控制功能库的制定工作,其目标是要在 IEC 的研发环境里加入运动控制技术,控制软件的编制则是组合了 PLC 和运动控制的功能。目前已形成了运动控制的 5 个标准:第 1 部分,运动控制库(2001 年 11 月发布),现已由多家供应商实现;第 2 部分,扩展(2004 年 4 月发布);第 3 部分,用户导则(2004 年 4 月发布);第 4 部分,内插多功能协调(待公布);第 5 部分,回零功能(待公布)。运动控制功能库的制定给了 PLCopen 组织一个崭新的定位,使其在推广 IEC 标准的基础上,增加了重要的技术含量。PLCopen 组织的运动控制部分在市场上的成功,也促使 PLCopen 组织的威望进一步提高。与以前相比,PLCopen 组织的发展更显广阔,它不仅着重于推广应用 IEC 标准,还在现有的标准以外附加了新的内容,为进一步丰富 IEC 标准或扩展标准使用的软件环境打下了基础。

PLCopen 组织考虑到编程仅仅是控制软件完整应用开发套件的一个环节,为规范它与其他环节间的数据交换的接口,有必要提供为实现 IEC 61131-3 编程的数据交换的 XML 格式。此外,利用 PLCopen 组织规范的 XML 格式,还可以实现不同硬件定义的 I/O 变量和内部变量之间的变换,从而为控制程序的无障碍移植创造了前提条件。考虑到编程仅仅是控制软件完整应用开发套件的一个组成部分,虽然 XML 并不是 IEC 61131-3 标准的内容,但为规范它与其他组成部分间的数据交换接口,PLCopen 组织还是强调通过为 IEC 61131-3 规定一种 XML

的格式倡导一种开发环境,使得各种不同目的的开发软件工具能克服交换数据的瓶颈,在此基础上构成统一平台。2005 年 4 月正式发布的关于 XML 的文本包括技术文件、XML 模式 (scheme) 和说明文件。XML 的特性是其结构和内容可以与它们的表达方式分隔开来,这样,同一个 XML 源文件可被再写一次,用多种形式来表达。XML 的优点在于它的可扩展性,可以通过它提供的 XML 模式来检查所包含数据的一致性,同时,在不同软件的 XML 模式之间,也可提供一种检查它们之间的不兼容性的方法。

PLCopen XML 规范的主要内容如下:规定了 IEC 61131-3 标准的全部 5 种编程语言的交换格式;类似于建模工具,规定了与图形和逻辑信息的生成程序的接口;类似于文件生成程序和管理程序以及版本管理,规定了与图形和逻辑信息的使用程序接口;还规定了功能块库的分配格式。用图形的方式可以更直观地表达如何实现不同开发工具之间的数据交换。

利用 XML 可在软件集成方面做很多事情。例如,通过在通用建模语言 UML 和 IEC 61131-3 编程语言之间交换数据,使用户有可能通过功能强大的系统级图形化软件开发工具(如 UML)对所开发的项目进行描述,建立整套系统的形式化模型。只要能够建立起正确的系统模型,图形化开发工具就可以根据该模型生成 PLC 或 C 语言的源代码,提供完善的系统流程图、标准化的软件说明文档,并对系统功能进行仿真校验,显著缩短现场调试时间,可以极大提高自控软件的开发效率。再如,在不同的阶段(如设计、组态和管理 3 个阶段)使用不同内容和功能的软件包,可以通过 XML 进行完整耦合,将不同软件集成起来构成统一工程平台的工具,而采用基于变量的寻址方式(Tags-Based Addressing),使用户可以直接运用实名变量,不必使用交叉参考列表来完成变量名与物理地址的转换。在使用多种软件工具时,统一的变量名对应惟一的地址。这种超越传统的解决方式大大节省了工程设计、调试、投运的时间和费用,减少了编程和调试运行中的错误,使技术文档更便于阅读和理解,使维护更加方便。此外,在与其他控制器通信进行数据交换时,在程序中可直接采用参与通信的控制器数据的实名制变量名,而不必采用物理地址。

近年来功能安全倍受关注和重视,特别是涉及安全的控制装置已由硬接线控制发展为可编程控制,再进一步发展到应用安全型现场总线控制。因而涉及安全的环节包括硬件、通信、基础软件(嵌入式操作系统、固件等)和应用软件。在安全方面,PLCopen 组织涉及的工作界定为 PLC 编程语言的功能安全。PLCopen 组织的成员与专业的安全机构 TÜV(国际安全认证的权威机构——德国莱茵技术监督服务公司)一起定义了 IEC 61131-3 标准的开发环境下涉及安全的规范。这必须由集成在 IEC 61131-3 标准的软件开发平台上的安全专用软件工具支持。安全功能性实现的标准化包括:定义与安全相关的函数集和功能块集;在编程环境中的支持,包括编程语言(LD、FBD)和功能性(安全数据类型和说明等);出错处理和诊断等。

为了让使用者对所用 PLC 在具体应用中的性能做出评估,还为了比较不同 PLC 的性能并发现其在具体应用中所表现出来的优点和缺点,PLCopen 组织在 2006 年 6 月底以技术文件的形式公布了《PLC 性能的基准测试方法》。该文件通过所定义的测试概要,以一种客观的方式,为寻求不同 PLC 平台的真实性能提供了标准化的方法。为更贴近实际应用,将基准测试方法划分为 5 种类别,它们分别是:数字式 I/O 处理(典型应用是无伺服驱动的小型机械);使用 SFC/状态机并在每个步序有数字式 I/O 的处理(典型应用是装配自动化);运动控制应用程序(典型应用是包装、印刷);数据处理应用程序(典型应用是测量记录和处理、协议);闭环控制应用程序(典型应用是过程控制)。

1.2 IEC 61131 标准

1.2.1 IEC 61131 的基本情况

1993 年 3 月由国际电工委员会 IEC(International Electro-technical Commission)正式颁布可编程控制器的国际标准 IEC 1131(1131 前面添加 6 后作为国际标准的编号,即 IEC 61131)。IEC 61131 标准将信息技术领域的先进思想和技术(如软件工程、结构化编程、模块化编程、面向对象的思想及网络通信技术等)引入工业控制领域,弥补并克服了传统 PLC、DCS 等控制系统的弱点(如开放性差、兼容性差、应用软件可维护性差以及可再用性差等)。目前,IEC 61131 标准已在发达国家得到广泛应用,不符合该标准的产品已不被最终用户接受,但在我国,对该标准及有关产品的推广工作还做得不够,许多技术人员还不知道有这样的国际标准。

符合 IEC 61131 标准的控制器产品,即使是由不同制造商生产的,其编程语言也是相同的,其使用方法也是类似的。因此,编程、维修技术人员可以一次学习,多次使用,从而大大减少了人员培训、技术咨询、系统调试和系统维护等费用。

IEC 61131 代表不同标准的组合和延续。它参考的其他标准有 IEC 50、IEC 559、IEC 617-12、IEC 617-13、IEC 848、ISO/AFNOR、ISO/IEC 646、ISO 8601、ISO 7185H 和 ISO 7498 等。IEC 61131 是第一个国际(和工业界)必须接受的标准,其最重要的先驱文件见表 1-1。

表 1-1 IEC 61131 标准最重要的先驱文件

年 份	德 国 标 准	国 际 标 准
1977	DIN 40719-6(功能块图)	IEC 848
1979		第一个 IEC 61131 草案工作组启动
1982	VDI 导则 2880,-4,PLC 编程语言	完成第一个 IEC 61131 草案,分为 5 个子工作组
1983	DIN 19239,PLC 编程	Christensen 报告(AB 公司),PLC 编程语言
1985		第一个 IEC 65A WG6 TF3 的成果
1990		完成 IEC 61131 标准第 1 和第 2 部分
1992		IEC 61131-1,IEC 61131-2
1993	DIN EN 661131 第 3 部分	IEC 61131-3
1994	DIN EN 661131 第 1 和第 2 部分	
1995		IEC 61131-4
1996	DIN EN 661131 附加导则(用户导则,IEC 61131-4)	
1994 ~ 2001		IEC 61131-3 的勘误
1995 ~ 1996		类型 2 和 3 的技术报告
1996 ~ 2001		IEC 61131 修订

IEC 61131 标准由 8 部分组成。除了第 6 部分功能安全标准尚未颁布外,其他部分都已颁布。

1) IEC 61131-1 通用信息(1992)。定义可编程控制器及其外围设备,如编程和调试工具(PADT)、人机界面(HMI)等的有关术语。

2) IEC 61131-2 设备特性(1992)。规定适用于可编程控制器及有关外围设备的工作条件、结构特性、安全性及试验的一般要求、试验方法和步骤等。

3) IEC 61131-3 编程语言(2000)。规定可编程控制器编程语言的语法和语义,规定编程语言有文本语言和图形语言,并描述了可编程控制器与第1部分规定的程序登录、测试、监视和操作系统的功能。

4) IEC 61131-4 用户导则(1995)。为从事自动化项目各阶段的用户提供可编程控制器系统应用中除第8部分外的其他方面的参考,如系统分析、装置选择、系统维护等。

5) IEC 61131-5 通信(2000)。规定可编程控制器的通信范围。包括任何设备与作为服务器的PLC通信、PLC与任何设备的通信、PLC为其他设备提供服务和PLC应用程序向其他设备请求服务时PLC的行为特性等。

6) IEC 61131-7 模糊控制编程(2000)。将第3部分编程语言与模糊控制的应用结合,为制造商和用户提基本意义的综合理解,提供不同编程系统间交换可移植模糊控制程序的可能性。

7) IEC 61131-8 编程语言应用和实现导则(2001)。为实现在可编程控制器系统及其程序支持的环境下编程语言的应用提供导则,为可编程控制器系统应用提供编程、组态、安装和维护指南。

在我国,从1992年开始,根据与国际标准等效原则,全国工业过程测量和控制标准化技术委员会着手进行国际标准的翻译和出版,并于1995年颁布了GB/T 15969.1~15969.4等4个可编程控制器的国家标准,分别等效于IEC 61131-1~IEC 61131-4。其后,IEC 61131-5、IEC 61131-7和IEC 61131-8相继颁布,等效的国家标准也随后问世。IEC 61131-6功能安全的有关标准也将于近期颁布。

1.2.2 IEC 61131-3 编程语言

IEC 61131-3 编程语言标准是第一个为工业控制系统提供标准化编程语言的国际标准。该标准针对工业控制系统所阐述的软件设计概念、模型等,适应当今世界软件、工业控制系统的发展方向,是一种非常先进的设计技术。它极大地推动了工业控制系统软件设计的发展,对现场总线设备的软件设计也产生了很大的影响。符合IEC 61131-3标准的软件系统是一个结构完美、可重复使用、可维护的工业控制系统软件,它不仅能应用于可编程控制器,而且能应用于流程过程和制造过程软件中,因此,它是新型的,先进的工业控制编程系统。

1. IEC 61131-3 标准编程语言的主要内容

IEC 61131 标准第3部分 IEC 61131-3 讨论编程语言,1993年颁布第1版,2000年颁布第2版。

IEC 61131-3 是第一个,也是至今为止惟一的为工业自动化控制系统的控制软件设计提供编程语言的国际标准。这个标准将现代软件的概念和现代软件工程的机制与传统的PLC编程语言成功地结合,又对当代种类繁多的工业控制器中的编程概念及语言进行了标准化。IEC 61131-3对可编程控制器软件技术的发展,乃至整个工业控制软件技术的发展,起着举足

轻重的推动作用。可以说,没有编程语言标准化便没有今天 PLC 走向开放式系统的坚实基础。自 IEC 61131-3 正式公布后,经过十来年的推广应用和不断完善,它获得了广泛的接受和支持,在工业控制领域产生了重要的影响,被全球越来越多的制造商和客户所接受,并且成为 DCS、PLC、IPC、PAC、运动控制以及 SCADA 的编程系统事实上的标准。

IEC 61131-3 标准编程语言分为公用元素和编程语言两部分。公用元素部分除了说明各种编程语言中使用的字符集、标识符、关键字等外,还定义了数据的外部表示、数据类型、变量和程序组织单元等,并对顺序功能表图的基本元素等进行了定义。与传统的可编程控制器编程语言不同,在公用元素中,编程语言标准还定义了配置、资源、任务和存取路径等基本概念。

IEC 61131-3 的编程语言部分定义了两类编程语言:文本化编程语言和图形化编程语言。文本化编程语言包括指令表编程语言(Instruction List, IL)和结构化文本编程语言(Structured Text, ST),图形化编程语言包括梯形图编程语言(Ladder Diagram, LD)和功能块图编程语言(Function Block Diagram, FBD)。在标准中定义的顺序功能表图(Sequence Function Chart, SFC)既没有归入文本化编程语言,也没有归入图形化编程语言,它被作为公用元素予以定义。这表示顺序功能表图既可用于文本化编程语言,也可用于图形化编程语言。

IEC 61131-3 标准编程语言对程序中的数据类型进行了严格定义。由于在以前的编程过程中,人们发现许多程序错误是由于在程序的不同部分中,数据类型表达的不同及处理方法的不同所造成的。因此,在 IEC 61131-3 标准编程语言中严格定义了有关变量的数据类型,从而防止发生因对变量定义了不同数据类型而造成的错误。编程语言中对变量数据类型的定义,使程序的可靠性、可维护性和可读性大大提高。

IEC 61131-3 标准编程语言还支持数据结构的定义。由于该标准支持数据结构,因此,相关数据元素如果不是相同的数据类型,也可在程序的不同部分传送,类似于在同一实体内的传送。此外,在不同程序组织单元之间传送的复杂信息,也可像传送单一变量一样。因此,IEC 61131-3 标准编程语言使程序的可读性大大提高,也保证了有关数据存取的准确性。

IEC 61131-3 标准编程语言规定对程序执行具有完全控制能力。传统的可编程控制器对程序的执行是按扫描原理进行的,因此,不能实现对事件驱动的程序执行、程序的并行执行等。IEC 61131-3 标准编程语言允许程序的不同部分在不同的时间条件下,以不同的扫描速率并行执行。它允许对程序的不同部分规定不同的执行次数和执行时间。因此,对程序执行具有完全的控制权。

IEC 61131-3 标准编程语言已对整个控制领域形成巨大冲击。它不仅适用于 PLC 产品,而且适用于运动控制产品、DCS 和基于工业 PC 的软逻辑、SCADA 等。其适用的市场领域正在不断扩大。采用或应用符合 IEC 61131-3 编程语言标准的产品,已经成为工业控制领域的发展趋势。

IEC 61131-3 标准编程语言的制定对可编程控制器的发展,以及整个工业控制软件的发展,起到了十分重要的推动作用。因为,该标准是控制领域第一次制定的有关数字控制软件技术的编程语言标准。它的制定订为可编程控制器走向开放系统奠定了坚实基础,也为其他计算机控制装置数字控制软件的开发提供了统一标准。

工控编程语言是一类专用的计算机语言,建立在对控制功能和要求的描述和表达的基础

上。作为实现控制功能的语言工具,工控编程语言不可能是一成不变的,其进步和发展必然受到计算机软件技术和编程语言的发展,以及它所服务的控制工程在描述和表达控制要求与功能的方法的影响。但是不论其如何发展和变化,这些年来的事实表明,它总是在 IEC 61131-3 标准的基础和框架上展开的。这就是说,IEC 61131-3 标准不仅仅是工控编程语言的规范,也是编程系统的实现架构的参考标准。

IEC 61131-3 标准分为公用元素和编程语言两大部分,可用图 1-3 描述它们的关系。

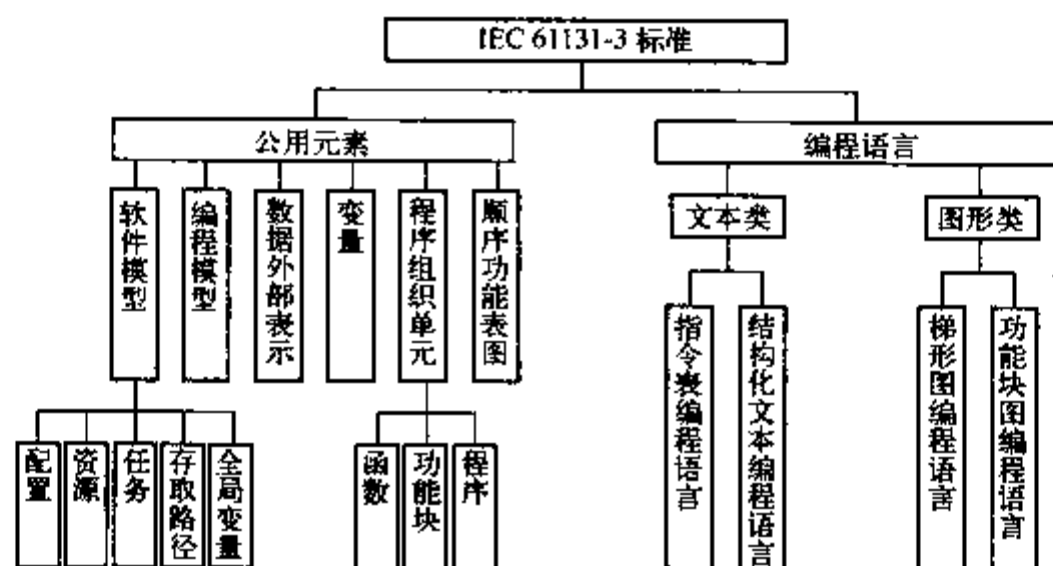


图 1-3 IEC 61131-3 标准的层次结构

IEC 61131-3 标准的公用元素中还包含语言元素,如标识符、分界符、关键字等。标准将顺序功能表图作为公用元素,因为它的动作和转换条件可以用标准规定的其他 4 种编程语言来编程。

可编程控制器的通信模型在 IEC 61131-5 规定。因此,在 IEC 61131-3 中仅作简单说明。

2. IEC 61131-3 的发展

IEC 61131-3 标准被市场广泛接受和支持之后,得到了良好的维护和发展,在应用过程中发现的缺陷逐步被改进;同时为了适应技术的进步和市场要求的环境,IEC 61131-3 标准也进行了适当的修订。IEC 61131-3 标准的第 3 次修改版将与 IEC 61499 标准相协调,在保持 IEC 61131-3 标准原有优点的同时,消除了以下两个最大的缺陷:

1) IEC 61131-3 标准沿用了直接表示与硬件有关的变量的方法,如果不解决与硬件相关变量之间的变换,就难以做到 PLC 系统之间(即使它们的编程系统都符合标准)实现真正意义上的程序可移植。

2) IEC 61131-3 标准只给出一个单一的集中 PLC 系统的配置机制,这显然不能适应分布式结构的软件要求。近些年来,现场总线和以太网在工业中大量应用,给工业自动化的体系结构带来了巨大影响。IEC 61131-3 标准必须适应客观形势的发展,在这方面有所突破。它应该允许功能块不一定集中常驻在单个硬件中,允许分散于不同硬件中的功能块通过通信方式也可以构成一个控制程序。这就是 IEC 61499 标准的主攻方向。制定 IEC 61499《工业过程测量和控制系统用功能块》标准的一个目的,就是对 IEC 61131-3 标准进行适当的扩展。主要是给出使 PLC 具有表达 IEC 61499 标准设备特性的能力。智能传感器和执行器本身就具有执行控制功能的能力,如果 IEC 61131-3 标准能支持分布式系统体系结构的编程,那么只要开发出相

应的软件工具,就能组成灵活性极强的现场总线控制系统。

只要有了符合标准的基本编程系统,即使使用不同的实时操作系统和 CPU 芯片不同的控制器或系统,都可以获得所需的编程软件和运行软件。换句话说,不同目标系统之间的差异并不妨碍使用同一个基本编程系统。近些年来,PLC 编程系统就是沿着这条轨迹在发展。国外商品化工业控制软件的编程系统平台的发展模式是专业化、集中化,即由为数不多的且专门从事工业控制基础软件的小型企业承担,他们向工控界提供一类不具体地依赖于特定 PLC 或其他控制系统硬件产品的开放式编程软件包,如加拿大 ICS Triplex 公司的 ISaGRAF,德国 KW 公司的 MULTIPROG,德国 Infoteam 公司的 OpenPCS,德国 3S 公司的 CoDeSys。许多的工业控制设备厂商(包括像西门子、横河电机、欧姆龙、三菱电机、ABB 公司等)都购买了这些商品化基础软件的使用权,并在此基础上再进行工作量不大的二次开发,或在此基础上将其高附加值的诀窍和控制算法嵌入其中。

近些年来,国内致力于自主知识产权的 IEC 61131-3 编程系统开发的有亚控科技、浙大中自、大连理工大学计控研究所,以及北京凯迪恩自动控制技术公司等。其中亚控科技的 KingAct 已经投入使用,浙大中自的 SunyIEC 实现了 IEC 61131-3 标准中的 5 种控制语言,使目前国内自行开发并拥有自主知识产权的编程系统达到了较高的技术水平,极大地促进和加快了我国自动化控制设备的发展。最有说服力的例子就是在建立了编程开发平台后,浙大中自每开发一个新的控制系统系列,不必再在编程软件方面花费大量重复劳动,因此大大缩短了新产品的开发周期并降低了成本。

IEC 61131-3 第 2 版于 2000 年下半年表决通过并公布施行。第 2 版对第 1 版进行了部分修改,主要包括提高程序组织单元(用 IEC 61131-3 标准编程语言编写的程序、功能和功能块)的可读性和实用性。例如,规定了新字符串 WSTRING 的数据类型,对 ST 编程语言的句法进行改进,以适应多输出连接的需要等。

标准编程语言的发展仍在进行中。为适应数字控制技术的发展,使编程语言能够适用于可编程控制器、DCS、FCS、运动控制及 SCADA 等工业控制领域的应用,还需要不断努力和完善有关编程语言标准。此外,标准编程语言的推广工作也是一项十分重要的工作,必须在一个非赢利国际组织的全面规划和安排下积极推广,才能使标准深入到各种应用中,充分发挥其开放系统的功能。

《控制工程》2004 年报道,目前,在欧洲大约有 70% ~ 80% 的工程师在使用 IEC 61131—3。日本已把 IEC 61131—3 列为日本工业标准(JIS B3503《可编程序控制器——编程语言》)。我国在 1995 年颁布了 PLC 的国家标准 GB/T15969—1995(它等效于国际标准 IEC 61131),并于 2005 年成立 PLCopen 中国组织 PCS,并进而推动了 IEC 61131—3 标准在我国 PLC 和工控市场的应用。

1.2.3 标准编程语言的特点

1. 标准编程语言的多样性

标准编程语言的多样性表现为编程语言有文本编程语言,还有图形编程语言,更有可用于文本编程,也可用于图形编程的顺序功能表图编程语言。语言的多样性是可编程控制器软件发展的产物,它为可编程控制器的应用提供了良好的操作环境。

(1) 易操作性

编程人员可柔性选择编程语言,可根据对编程语言的熟悉程度选用既适应应用项目要求,又能够发挥自身优势的编程语言,从而缩短程序设计时间,缩短调试时间。

(2) 编程的灵活性

不同的工程应用具有不同的最佳编程方式,不同的编程语言具有不同的特点,可根据工程应用的需求选用合适的编程语言。

1) 梯形图编程语言:与电气操作原理图相对应,具有直观性和对应性;电气技术人员易于掌握和学习;与语句表编程语言有一一对应的关系,便于相互转换和对程序的检查;但对复杂控制系统的编程,程序描述仍不够清晰。

2) 功能块图编程语言:以功能块为设计单位,能从控制功能入手,使控制方案的分析和理解变得容易;功能块具有直观性强、容易掌握的特点,有较好的操作性;对复杂控制系统仍可用图形方式清晰描述;但每种功能块要占用程序存储空间,并延长程序执行时间。

3) 语句表编程语言:容易记忆,便于掌握;与梯形图编程语言有一一对应的关系,便于相互转换和对程序的检查;不受显示屏幕大小的限制,输入元素不受限制;对复杂控制系统的编程,程序描述不够清晰。

4) 结构化文本编程语言:可实现复杂控制运算;对编程人员的技能要求高;直观性和易操作性差。

5) 顺序功能表图编程语言:以完成的功能为主线,操作过程条理清楚,便于对程序操作过程的理解和思路的沟通;对大型程序,可分工设计,采用较灵活的程序结构,节省程序设计时间和调试时间;由于只对活动步进行扫描,因此,可缩短程序执行时间。

(3) 多种编程语言的融合,实现程序优化

标准编程语言中,顺序功能表图编程语言作为公用元素,既可用于文本类编程语言,也可用于图形类编程语言,使多种编程语言融合,实现了程序设计的优化。

2. 标准编程语言的兼容性

标准编程语言的兼容性表现为其不仅能够用于不同制造商生产的可编程控制器产品的编程,而且也适用其他数字控制装置的编程。例如,不少 DCS 产品的说明书强调指出,该产品符合 IEC 61131-3 标准,即满足标准编程语言的有关性能,能够用标准编程语言进行控制系统的组态。

标准编程语言能够适用于可编程控制器、分散控制系统、现场总线控制系统、数据采集和监视系统、运动控制系统等,是功能强、应用广、使用方便的国际标准。

标准编程语言的软件模型适应各种不同行业、不同规模、不同结构的工业应用。因此,标准编程语言与所使用的硬件无关。对用户来说,其对硬件的依赖性变得越来越小,从而不必为选用何种产品而烦恼。

3. 标准编程语言的标准化和开放性

IEC 61131-3 标准编程语言是用于可编程控制器编程的国际标准。因此,它定义了编程语言的语法和语义,也定义了语句和句法。标准编程语言包括公用元素和编程语言,对编程语言中所使用的变量、数据类型、程序、功能和功能块等都有统一表达方式和性能定义,这使编程语言的应用变得容易。

编程语言的标准化也使可编程控制器系统成为开放系统。任何一个制造商生产的产品,如果符合标准编程语言,就能够使用该编程语言进行编程,并能够获得同样的执行结果。编程语言的标准化切断了软件与硬件的依赖关系。

从开放系统互连参考模型的构架看,一个开放系统表示该系统能够与其他符合开放系统互连通信模型的其他任何一个系统进行信息交换。从通信协议看,这些系统之间有相同的通信协议和规范。从软件看,开放系统指一个系统中开发的软件可方便地移植到任何一个符合标准编程语言的其他系统中。

编程语言的标准化使开放得以实现。但是,从目前的标准看,要做到软件不修改的完全移植还有困难。例如,不同可编程控制器的外部端子地址不同,移植时还需对输入输出地址重新定义。

4. 标准编程语言的可读性

标准编程语言定义了变量的数据类型,程序中数据运算、传递等需要根据不同的数据类型执行,避免因设计的数据类型选用不当造成程序出错现象的发生,从而提高程序的可读性。

标准编程语言采用数据结构的构架,相关数据元素虽然是不同的数据类型,但可在程序的不同部分传送,类似于在同一实体内的传送。而不同程序组织单元之间传送的复杂信息,也可像传送单一变量一样。这使程序的可读性提高,易操作性增强。

标准编程语言来自工控软件和软逻辑 PLC 软件,因此,编程语言中大量语言的表达方式与常用计算机编程语言的表达方式类似。例如,IF 和 CASE 等选择语句,FOR、WHILE 和 REPEAT 等循环语句都与计算机编程语言类似,这大大方便了用户对标准编程语言用法的理解,提高了程序的可读性。

5. 标准编程语言的易操作性和安全性

易操作性和安全性有时是矛盾的两个方面。但在 IEC 61131-3 标准编程语言中,两者被有机结合。这是因为标准编程语言来自于传统的电气逻辑图,来自于工控软件的成熟软件和软逻辑 PLC。由于标准编程语言是常用计算机编程语言的沿用、改进和扩展,因此,它保留了这些编程语言的优点,克服了它们的缺点,使编程操作变得容易。同时,由于这些编程语言是标准的,因此,出错的情况已被控制到最小,这使标准编程语言变得更安全。

为使标准编程语言能够安全正确地使用,IEC 61131-3 标准还提供了出错原因列表,用户可直接根据列表所提供的出错原因查找,既使操作变得方便,也使安全性大大提高。

在 IEC 61131-3 中的独立实体是程序组织单元,调用时,只需要它的外部接口,而不需要包含在其内部的代码。因此,一旦定义了外部接口,就能够通过编程系统为整个项目服务,保证系统的易操作性和安全性。

6. 标准编程语言对硬件的非依赖性

标准编程语言的基本级测试是离线进行的。它用测试程序检查编程系统语法的行为特性。符合级测试包括附加的在线测试,通过连接一台可编程控制器来测试编程系统的语义行为特性。编程语言对实际编程系统硬件的依赖性程序能够移植的关键。这是因为制造商的可编程控制器硬件性能影响着编程语言的程序结构,因此,编程语言与硬件的关联程度有多低是十分重要的问题。至今为止,由于定位和成本原因,小型可编程控制器产品几乎不需要具备标准所规定的所有性能。因此,IEC 61131-3 规定的所有性能在这些编程系统中实现仍是困

难的。

为便于程序的移植,IEC 61131-3 提供了下列机制:

- 1) 一个程序所需的所有外部信息由符合 IEC 61131-3 标准的全局变量、存取路径或通信功能块提供。
- 2) 使用的硬件输入输出地址必须在程序中或配置说明中进行说明。
- 3) 制造商应随其软件提供一个与硬件有关的性能表格。

第 2 章 公用元素和程序组织单元

本章介绍在文本和图形编程语言中的公用文本和图形元素,包括公用元素、数据外部表示、数据类型、变量、程序组织单元及软件模型、编程模型(功能模型)等。通信模型不在本书介绍。

2.1 软件模型、编程模型

2.1.1 软件模型

IEC 61131-3 标准的软件模型用分层结构表示。每一层隐含其下层的许多特性,从而构成优于传统可编程控制器软件的理论基础。

软件模型描述基本的高级软件元素及其相互关系。这些元素包括:程序组织单元,即程序和功能块;组态元素,即配置、资源、任务、全局变量和存取路径。它是现代 PLC 的软件基础。图 2-1 是 IEC 61131-3 标准的软件模型。

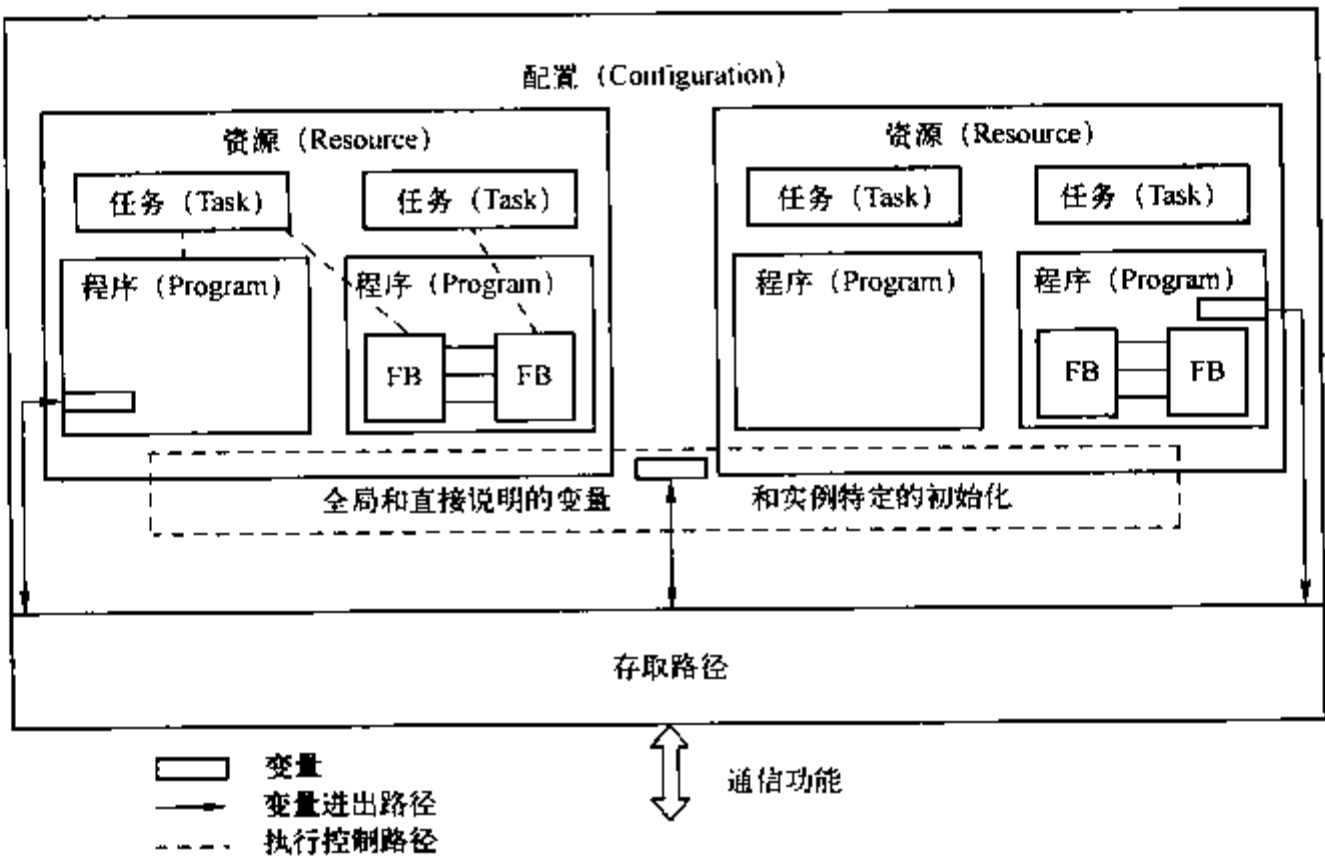


图 2-1 IEC 61131-3 标准的软件模型

IEC 61131-3 软件模型从理论上描述了如何将一个复杂程序分解为若干小的可管理部分,并在各分解部分之间有清晰和规范的接口方法。软件模型描述一台可编程控制器如何实现多个独立程序的同时装载和运行,如何实现对程序执行的完全控制等。

IEC 61131-3 软件模型分为输入输出界面、通信界面和系统界面 3 部分。

1) 输入输出界面。每个可编程控制器系统都需要读取来自实际过程的输入。例如,来自微动开关、压力传感器、热电偶等传感器、变送器的物理通道的信号,它也经物理通道输出信号到各种执行器,如电磁阀、阀门定位器、电加热器和伺服机构等。

2) 通信界面。大多数可编程控制器系统需要与其他设备,包括其他可编程控制器、服务器等交换信息,提供操作显示画面和操作面板等。

3) 系统界面。在可编程控制器软件和硬件之间需要系统界面。系统服务器需要确保程序可初始化和正确运行,提供硬件和有时作为系统固件的嵌入式系统软件之间的组合。

1. 配置

配置(Configuration)是语言元素,或结构元素,相当于 IEC 61131-1 标准所定义的可编程控制系统。它位于软件模型的最上层,是大型的语言元素。

配置是可编程控制器系统的整个软件,它用于定义特定应用的 PLC 系统的特性。通常,配置等效于一个 PLC 所需的软件。对大型复杂的应用,如整个产品线的自动化,可能需要几个 PLC 相互作用,一个配置可以与其他 IEC 配置通过通信接口实现通信。在这种情况下,每个 PLC 的软件可作为一个分离的配置。因此,可将配置认为是一个特定类型的控制系统,它包括硬件装置、处理资源、I/O 通道的存储地址和系统能力,即等同于一个 PLC 的应用程序。

在不同的 PLC 系统内部,配置能够与其他 IEC 配置通过定义界面进行通信,其前提是必须使用标准的编程语言元素。

配置用关键字 CONFIGURATION 开始,随后是配置名称和配置声明,最后用 END_CONFIGURATION 结束。配置声明包括定义该配置的有关类型和全局变量的声明、在配置内资源的声明、存取路径变量的声明和配置变量声明等。下面是一个配置的示例。

【例 2-1】 一个配置的示例。

```
CONFIGURATION CELL_1          (* CELL_1 是配置名称 *)
  VAR_GLOBAL w:UINT;END_VAR    (* w 是在配置 CELL_1 内的全局变量名 *)
  RESOURCE STATION_1 ON PROCESSOR_TYPE_1 (* STATION_1 是资源名 *)
    VAR_GLOBAL z1:BYTE;END_VAR  (* z1 是资源 STATION_1 内的全局变量名 *)
    TASK SLOW_1 (INTERVAL:= t#20 ms, PRIORITY:=2); (* SLOW_1 是任务名 *)
    TASK FAST_1 (INTERVAL:= t#10 ms, PRIORITY:=1); (* FAST_1 是任务名 *)
    PROGRAM P1 WITH SLOW_1;      (* P1 是程序名,它与 SLOW_1 任务结合 *)
      F(xI:= %IX1.1);
    PROGRAM P2 ;G(OUT1 = > w,    (* P2 是程序名,G 程序实例名 *)
      FB1 WITH SLOW_1,          (* FB1 是功能块实例名,它与 SLOW_1 任务结合 *)
      FB2 WITH FAST_1);         (* FB2 是功能块实例名,它与 FAST_1 任务结合 *)
  END_RESOURCE
  RESOURCE STATION_2 ON PROCESSOR_TYPE_2 (* STATION_2 是资源名 *)
    VAR_GLOBAL z2:BOOL;(* z2 是资源 STATION_2 内的全局变量名 *)
    AT %QW5;INT;(* 地址 %QW5 的变量是 STATION_2 内直接表示的全局变量 *)
```

```

END_VAR
TASK PER_2(INTERVAL:= t#50 ms,PRIORITY:=2);(* PER_2 是周期执行的任务名 *)
TASK INT_2(SINGLE:= z2,PRIORITY:=1);(* INT_2 是事件触发的任务名 *)
PROGRAM P1 WITH PER_2:(* P1 是程序名,它与 PER_2 任务结合 *)
    F(x1:= z2, x2:= w);(* 使用全局变量实现数据通信 *)
PROGRAM P4 WITH INT_2:(* P4 是程序名,它与 INT_2 任务结合 *)
    H(HOUT1 = > %QW5,
    FB1 WITH PER_2);(* FB1 是功能块名,它与 PER_2 任务结合 *)
END_RESOURCE
VAR_ACCESS (* 存取路径变量声明 *)
    (* 存取路径变量名 *) (* 存取路径 *) (* 数据类型 *) (* 读写属性 *)
ABLE ;STATION_1. %IX1. 1 ;BOOL READ_ONLY;
BAKER ;STATION_1. P1. x2 ;UINT READ_WRITE;
CHARLIE ;STATION_1. z1 ;BYTE;
DOG ;w ;UINT READ_ONLY;
ALPHA ;STATION_2. P1. y1 ;BYTE READ_ONLY;
BETA ;STATION_2. P4. HOUT1 ;INT READ_ONLY;
GAMMA ;STATION_2. z2 ;BOOL READ_WRITE;
SI_COUNT ;STATION_1. P1. COUNT ;INT;
THETA ;STATION_2. P4. FB2. d1 ;BOOL READ_WRITE;
ZETA ;STATION_2. P4. FB1. c1 ;BOOL READ_ONLY;
OMEGA ;STATION_2. P4. FB1. c3 ;INT READ_WRITE;
END_VAR
VAR_CONFIG (* 配置变量声明 *)
    STATION_1. P1. COUNT ;INT:= 1;
    STATION_2. P1. COUNT ;INT:= 100;
    STATION_1. P1. TIME1 ;TON:= (PT:= T#2. 5s);
    STATION_2. P1. TIME1 ;TON:= (PT:= T#4. 5s);
    STATION_2. P4. FB1. C2 AT %QB25 ;BYTE;
END_VAR
END_CONFIGURATION

```

例 2-1 中,配置有一个配置名称 CELL_1,它有一个全局变量,全局变量名是 w,数据类型是 UINT(无符号整数)。该配置有两个资源,资源名分别是 STATION_1 和 STATION_2。在配置中也声明了配置中有关变量的存取路径变量(在 VAR_ACCESS 和 END_VAR 段声明)和配置变量信息(在 VAR_CONFIG 和 END_VAR 段声明)。图 2-2 是例 2-1 软件模型的图形表示,图中显示了控制系统的配置、资源、任务、程序组织单元、全局变量和存取路径变量等信息。

在 PLC 系统中,配置将系统内所有资源结合成组,它为资源提供数据交换的手段。在一个配置中,可定义在该 PLC 项目中全局可有效使用的全局变量。示例中的变量 w 是在该配置的全局变量。它读取资源 STATION_1 内程序 P2 的变量 OUT1,并传送到资源 STATION_2 内的程序 P1 的变量 x2。

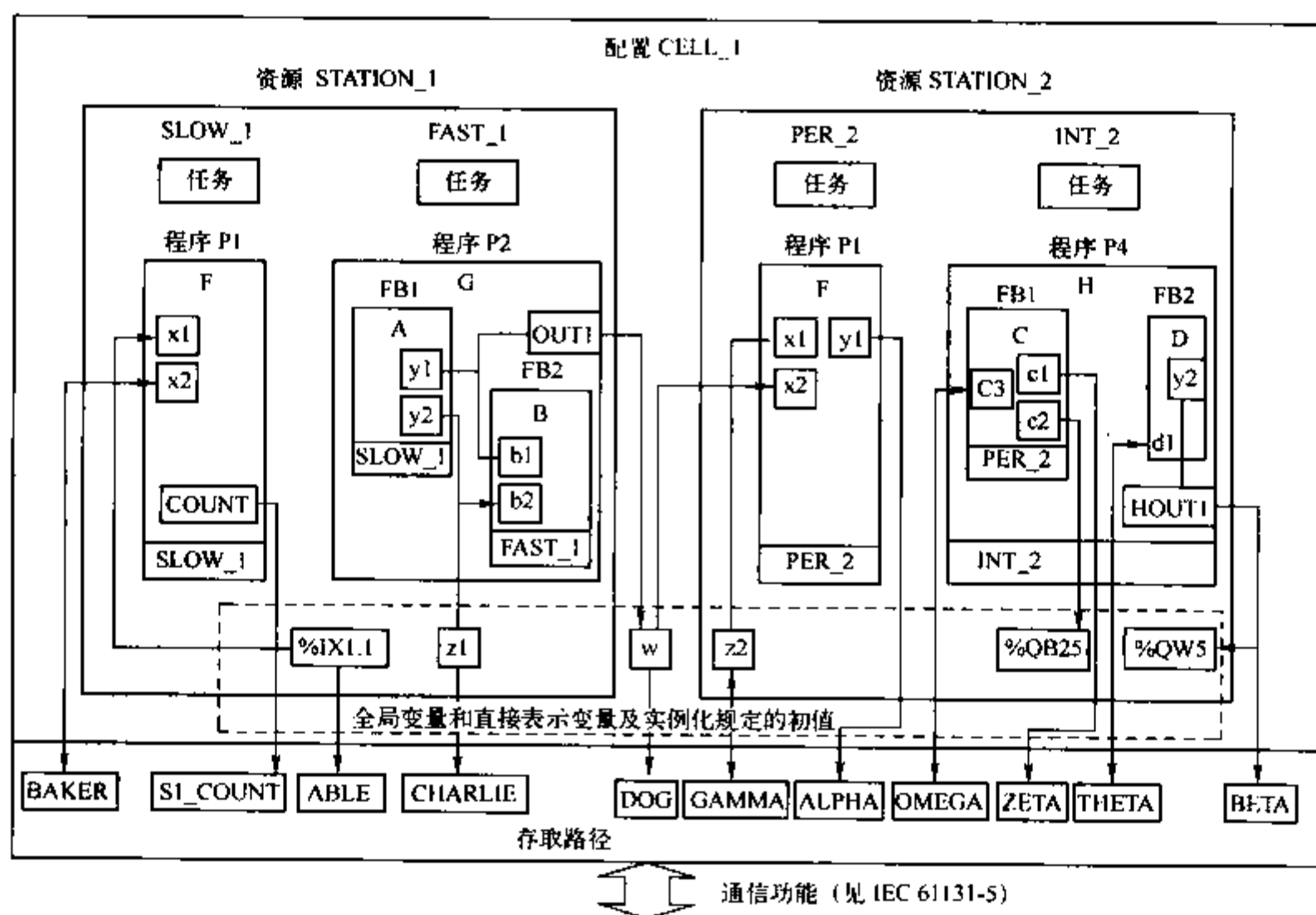


图 2-2 例 2-1 软件模型的图形表示

2. 资源

资源(Resource)位于软件模型的第2层,它为运行程序提供支持系统,是能执行 IEC 程序的处理手段。资源反映了可编程控制器的物理结构,为程序和 PLC 的物理输入输出通道提供了一个接口。

每个配置内可有一个或几个资源。资源提供对所有程序执行所需特性的支持。

在软件工程术语中,资源作为进入能够执行 IEC 程序的虚拟机器(Virtual Machine)的界面,它的主要功能是提供一个支持系统运行的程序。

标准定义资源是一个语言元素,它具有 IEC 61131-1 标准定义的“信号处理功能”、“人机界面”和“传感器执行器界面”的功能。

一个程序只有装入到资源中才能运行。资源的主要功能之一是提供程序和 PLC 物理 I/O 通道之间的界面。

每个资源可支持多于一个的程序。因此,资源使 PLC 能够加载、启动和执行许多总体独立程序。

资源有资源名称,它被分配在一个 PLC 的 CPU 中。因此,可将资源理解为一个 PLC 中的微处理器单元。资源内定义的全局变量在该资源内部是有效的。资源可调用具有输入输出参数的运行期(Run-Time)程序,给一个资源分配任务和程序,并声明直接表示变量。

资源用关键字 RESOURCE 开始,随后是资源名称和 ON 关键字、资源类型名和资源声明,最后用 END_RESOURCE 关键字结束。

资源名是一个符号名,它用于说明可编程控制器系统中的一个微处理器。编程系统提供可编程控制器系统内该资源(即每个 CPU 的命名)的类型和数量,并检验这些资源类型和数量,使系统能够正确地使用这些资源。

例 2-1 中有两个资源。资源名 STATION_1 有一个全局变量,变量名是 z1,其数据类型是字节。该资源的类型名是 PROCESSOR_TYPE_1,它有两个任务,任务名为 SLOW_1 和 FAST_1。还有两个程序,程序名是 P1 和 P2。资源名 STATION_2 有两个全局变量,一个变量名是 z2,其数据类型是布尔量;另一个是直接表示变量,其地址是%QW5,数据类型是整数。需指出,资源 STATION_1 中的全局变量 z1 的数据只能从资源 STATION_1 中存取,不能从资源 STATION_2 存取;反之亦然。如果希望从资源 STATION_2 也能存取,则必须在配置 CELL_1 中作为全局变量进行声明,如配置中的全局变量 w。

资源声明包括在该资源内的全局变量、任务和程序声明等内容。在资源声明段中,ON 关键字用于限定“处理功能”类型、“人机接口”和“传感器执行器接口”功能。例 2-1 中的 PROCESSOR_TYPE_2 限定了资源 STATION_2 内处理器类型的功能。

3. 任务

任务(Task)位于软件模型分层结构的第 3 层。任务用于规定程序组织单元 POU 在运行期的特性。任务是一个执行控制元素,它具有调用能力。

一个资源中可以有多个任务。一旦任务被设置,它就可控制一系列程序组织单元周期地执行,或者根据一个特定的事件触发来执行。

(1) 任务的表示

任务由关键字 TASK 开始,随后是任务名、任务初始化设置(用圆括号将任务的有关参数分列其中)。例如,TASK SLOW_1 (INTERVAL:= t#20 ms,PRIORITY:= 2);表示任务名为 SLOW_1 的任务是周期执行的任务,周期间隔时间 20 ms,优先级为 2。

任务有两种表示方法:文本表示和图形表示。

1) 文本表示。任务的文本表示结构如下:

```
TASK 任务名 (任务属性)
PROGRAM 程序名 WITH 任务名:(程序接口)
```

示例中的任务 SLOW_1 表示为

```
TASK SLOW_1 (INTERVAL:= t#20 ms,PRIORITY:= 2);
```

与该任务结合的程序表示为

```
PROGRAM P1 WITH SLOW_1:F(x1:= %IX1.1);
```

其中,P1 是程序实例名;F 是程序的接口。

程序接口用于为形参提供实参数据。

2) 图形表示。任务的图形表示用矩形框表示,框内用 3 个输入参数表示,框外在对应的输入参数处列写实际的参数值。图 2-3 是示例中任务 SLOW_1 和任务 INT_2 的图形表示。

任务的图形表示仅显示其属性,与程序结合的部分

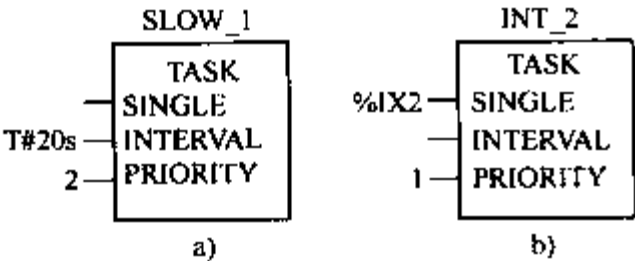


图 2-3 任务的图形表示
a) 周期任务 b) 非周期任务

分没有图形表示形式。

(2) 任务的输入参数

任务除了有任务名称外,还有 3 个输入参数,即 SIGNAL、INTERVAL 和 PRIORITY 属性。

1) SIGNAL。单任务输入端,在该事件触发信号的上升沿,触发与任务相结合的程序组织单元执行一次。例如,任务 INT_2 中 z2 是单任务输入端的触发信号。

2) INTERVAL。周期执行时的时间间隔。当其值不为零,且 SIGNAL 信号保持为零,则表示该任务的有关程序组织单元被周期执行,周期执行的时间间隔由该端输入的数据确定,如任务 SLOW_1,其周期执行时间为 20 ms。当其值为零(不连接),表示该任务是由事件触发执行的,如任务 INT_2。

周期执行时的时间间隔取决于任务执行完成需要多长时间。如果一个任务执行时间有时足够长,有时又比较短时,这类系统称为不确定系统。

3) PRIORITY。当多个任务同时运行时,对任务设置的优先级。0 表示最高优先级,优先级越低,数值越高。

(3) 任务的执行

1) 无优先级(Non-preemptive Scheduling)执行。当一个程序组织单元或操作系统功能的执行完成时,资源上的供电电源有效,则具有最高执行优先级(数值最小)的程序组织单元开始执行。如果多于一个的程序组织单元在最高执行优先级等待,则在最高执行优先级的程序组织单元中等待时间最长的程序组织单元先执行。对不确定系统应采用无优先级执行。

2) 优先级(Preemptive Scheduling)执行。当一个程序组织单元执行时,它能够中断同一资源中较低优先级程序组织单元的执行,即较低优先级程序组织单元的执行被延缓,直到高优先级程序组织单元的执行完成。一个程序组织单元不能中断具有同样优先级或较高优先级的其他单元的执行。

例 2-1 的资源 STATION_1 中,任务名为 SLOW_1 的任务是一个周期执行的任务,它的周期执行间隔是 20 ms,优先级是 2;任务名为 FAST_1 的任务也是一个周期执行的任务,它的周期执行间隔是 10 ms,优先级是 1。资源 STATION_2 中,任务名为 INT_2 的任务是一个事件触发的任务,触发信号来自全局变量 z2,其优先级是 1。

图 2-4 说明无优先调度和优先调度时任务的执行情况。

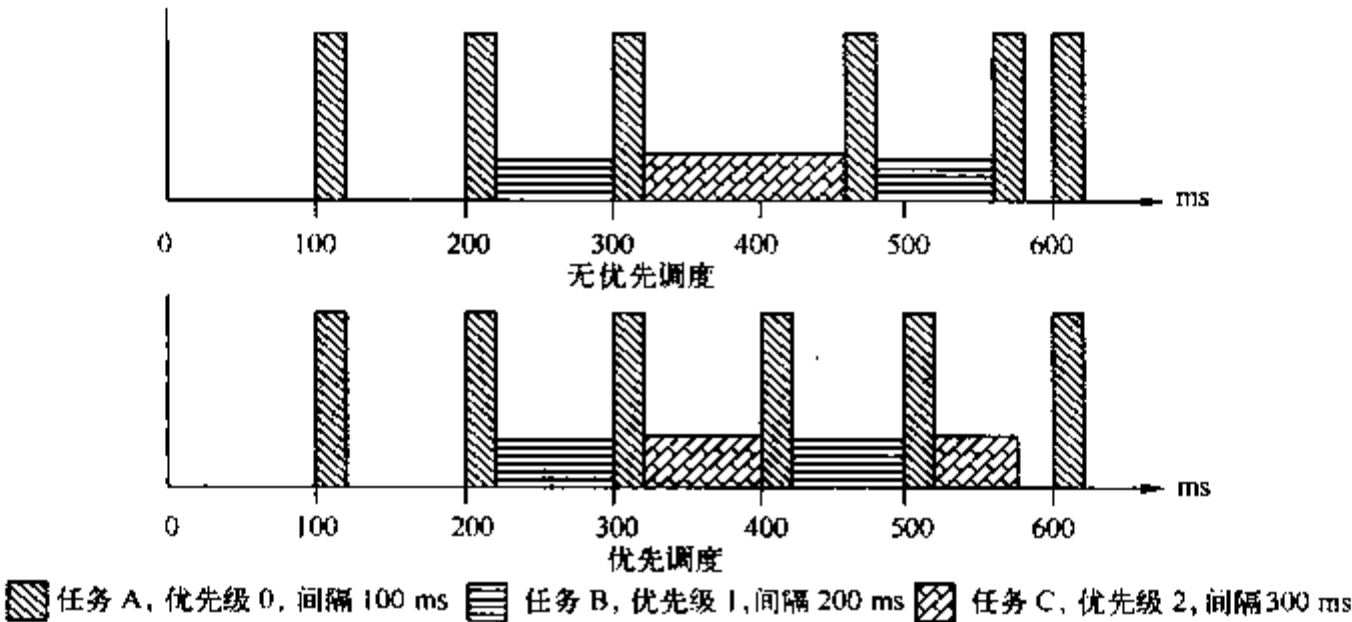


图 2-4 任务的无优先调度和优先调度

图中,任务 A 有最高优先级,每隔 100 ms 执行一次。任务 B 优先级次之,每隔 200 ms 执行一次。任务 C 优先级最低,每隔 300 ms 执行一次。无优先调度时,由于任务 C 的执行时间长,而使任务 A 和 B 被延缓(在 500 ms 后)。

优先调度时,任务 A 每隔 100 ms 执行,任务 B 每隔 200 ms 执行,任务 C 被中断,用于允许任务 A 和 B 的执行。

(4) 任务执行准则

由任务使能的程序组织单元实施的控制遵循下列准则:

1) 单任务输入 SIGNAL 不为零,表示与该任务结合的程序组织单元在 SIGNAL 上升沿时触发执行一次,因此,是事件触发的单任务

2) 当 INTERVAL 不为零,SIGNAL 保持为零(约定值为零),则与该任务结合的程序组织单元周期执行,执行周期由 INTERVAL 数据确定。

3) 当多个任务执行时,有两种解决冲突的方法:

- 立刻中断正在执行的任务,使优先级高的任务先被执行,即 PRIORITY 数值小的任务先被执行,这称为占先调度(执行)。在具有相同优先级的多个任务中,与任务结合的等待时间长的程序组织单元先被执行。
- 不中断正在执行的任务,直到该任务执行完成,然后执行在等待队列中优先级最高的任务,这称为非占先调度(执行)。

【例 2-2】 任务的执行。

```
TASK T_QUICK( INTERNAL:= T#10 ms,PRIORITY:= 1 );  
PROGRAM MOTI0N1 WITH T_QUICK;PROGM1 ( REGPAR:= %MWI,R_VAL => ERR );  
TASK T_INTERRUPT ( SINGLE:= TRIGGER,PRIORITY:=1 );  
PROGRAM LUBE WITH T_INTERRUPT;PROGM2 ;
```

例 2-2 定义两个任务 T_QUICK 和 T_INTERRUPT,前者是具有 10 ms 循环周期的循环任务,后者是具有较高优先级的中断任务。当正常运行时,每隔 10 ms 执行一次任务 T_QUICK,如果与该任务结合的程序 MOTI0N1 的执行时间大于 10 ms,则可编程控制器会发送一个运行期故障的报告。由于程序 LUBE 与 MOTI0N1 有相同的优先级,因此,如果在执行任务 T_QUICK 时,TRIGGER 信号变为 1,因具有相同的优先级,系统仍需在执行完这次 T_QUICK 任务后,才能执行 T_INTERRUPT 任务一次。为使 T_INTERRUPT 任务能够中断正在执行的任务立即执行,可将 T_INTERRUPT 任务的优先级设置为 0。

4) 没有任务结合的程序具有最低的优先级。因此,这样的程序在资源开始执行前执行,并在执行终止后立刻重新执行。

5) 与任务没有直接结合的功能块实例根据正常规则执行,即根据其内部功能块实例被说明的程序组织单元中语言元素的求值次序执行。与任务结合的功能块实例是在任务的排他性控制下执行。它与其内部功能块实例被说明的程序组织单元的求值规则无关。

6) 为确保数据的同步,程序中功能块的执行根据下列规则获得同步。

- 如果功能块接收来自其他功能块的输入信号多于一个,则当该功能块执行时,来自其他功能块的所有输入采用同样的求值结果。

- 如果同一功能块的输出送到两个或多个功能块,并且如果目的功能块是全部与任务有显式或隐式的结合,则到所有这样的目的功能块的输入信号在它们的求值时间内,应与源功能块有同样的求值结果。

7) 对每个任务需要配置一个看门狗定时器 WDT,用于对任务的监视。

4. 全局变量

标准允许变量在不同的软件元素内被声明。变量的范围确定其在哪个程序组织单元中是可用的。范围可能是局部的或全局的。每个变量的范围由它被声明的位置和声明所使用的变量关键字所定义。

在配置中声明的全局变量可在整个配置范围内使用,在资源中声明的全局变量只能在该资源范围内使用。在一个程序内声明的全局变量可以存取在该程序内部的功能块和函数。

全局变量能够用于整个工程项目。全局变量声明的格式如下:

```
VAR_GLOBAL
    全局变量名声明
END_VAR
```

全局变量能与其他网络进行数据交换。一个系统中不能有相同名称的两个全局变量。在每个使用它的程序组织单元中,需用 VAR_EXTERNAL 来声明该全局变量。

全局变量被定义在配置、资源或程序层内部。全局变量提供了在两个不同程序和功能块之间非常灵活的交换数据的方法。

5. 存取路径

存取路径用于将全局变量、直接表示变量和功能块的输入、输出和内部变量联系起来,实现信息的存取。它提供在不同配置之间交换数据和信息的方法。每一配置内的许多指定名称的变量可通过其他远程配置来存取。

(1) 存取路径变量的存取方法

有两种存取方法:读写方式和只读方式。读写(READ_WRITE)方式表示通信服务能够改变变量的值;只读(READ_ONLY)方式表示能够读取变量的值但不能改变变量的值。当不规定存取路径方式时,约定的存取方式是只读方式。

例 2-1 中,CHARLIE 变量没有声明其读写属性,表示该存取路径变量具有只读属性。图形表示时,具有只读属性的变量有可读取的箭头。读写属性的变量还有可写入到其他变量的箭头(即有双向箭头)。

(2) 存取路径变量的表示方法

存取路径变量声明的文本表示用下列结构声明:

```
VAR_ACCESS
    存取路径变量名:外部存取路径:存取路径变量的数据类型和读写属性
END_VAR
```

例 2-1 中,ABLE 是存取路径变量名,外部存取路径是 STATION_1. %IX1. 1,即从外部资源 STATION_1 的直接表示变量%IX1. 1 存取数据。存取路径的数据类型是布尔量(BOOL),存取方式是只读 READ_ONLY(也可不列出),表示从%IX1. 1 读取布尔型数据。

外部存取路径采用串联方式表示。开始是资源名,然后是程序实例名,功能块实例名,最后是变量名,各个名称之间用圆点分隔,中间没有的名称可省略。例 2-1 中的存取变量 ABLE 是从外部读取的,因此,外部存取路径变量从资源 STATION_1 中直接表示变量%IX1.1 获取。而 BAKER 从资源 STATION_1 中程序 P1 的变量 x2 获得,由于该变量是读写变量,因此,既可由 x2 写入 BAKER 的值,也可用 BAKER 类读取 x2 的值。

存取路径变量声明的图形表示见图 2-2。

当变量是结构数据类型变量或数组数据类型变量时,一个存取路径变量只能存取一个结构数据类型变量的一个单项或数组数据类型变量的一个元素。

须注意,在存取路径变量的数据类型声明时,该变量的数据类型应与其他地方对该变量声明的数据类型一致,否则会造成数据类型不匹配的错误。

(3) 存取路径变量特性

存取路径变量的特性见表 2-1。存取路径变量的示例见例 2-1。

表 2-1 存取路径变量的特性

序 号	特 性 描 述	示 例
1	在 CONFIGURATION 内的 VAR_ACCESS...END_VAR 结构	例 2-1 中的 ABLE 等变量
2	在 CONFIGURATION 内对 GLOBAL 变量的存取	例 2-1 中的 w 变量
3	在 PROGRAM 内对 GLOBAL 变量的存取	例 2-1 中资源 STATION_1 中程序 P2 对 w 的写入
4	在 PROGRAM 内对变量的存取	例 2-1 中资源 STATION_2 中程序 P1 对 w 的读取
5 ^①	在 FUNCTION BLOCK 内对变量的存取	例 2-1 中资源 STATION_1 中功能块 FB2 对 z1 的读写

① 该特性被支持时,存取层是与执行有关的参数。

应用时须注意,由 VAR_CONFIG...END_VAR 结构提供的实例特定初始值通常超过该数据类型规定的初始约定值。此外,定义实例特定初始值的方法不能用于在 VAR_TEMP,VAR_EXTERNAL,VAR_CONSTANT 或 VAR_IN_OUT 结构中声明的变量。

6. 软件模型的特点

IEC 61131-3 软件模型具有下列特点:

- 1) 能够灵活地用于宽范围的不同的 PLC 体系结构。由于该软件模型是一个国际标准的软件模型,它并不针对某一具体的 PLC 系统,因此,具有很强的适用性,能够应用于不同制造商的 PLC 产品。
- 2) 适合小规模系统和大型分散系统。
- 3) 在一台 PLC 中可同时装载、启动和执行多个独立程序。IEC 61131-3 标准允许一个配置内有多个资源,每个资源可支持多个程序,因此,在一台 PLC 中可以同时装载、启动和执行多个独立程序。而传统的 PLC 程序只能同时运行一个程序。
- 4) 增强了分级设计的分解。一个复杂程序软件可以通过分层分解,最终分解为可管理的程序组织单元。程序可被定义为一个功能模块和函数的网络。
- 5) 软件能够被设计成可重复使用的程序组织单元,即程序、功能模块和函数。软件模型的可重复使用性是 IEC 61131-3 软件模型的重要优点。

- 6) 实现对程序执行的完全控制能力。IEC 61131-3 标准采用“任务”机制,保证了 PLC 系统对程序执行的完全控制能力。传统 PLC 程序只能顺序扫描和执行程序,对某一段程序不能按用户的实际要求定时执行,而 IEC 61131-3 程序允许程序的不同部分在不同的时间、以不同的比率并行执行,扩大了 PLC 的应用范围。
- 7) 可经通信网络提供交换信息的工具。

2.1.2 编程模型

IEC 61131-3 编程模型用于描述库元素如何产生衍生元素。可编程控制器的编程模型也称为功能模型,因为它描述可编程控制器系统所具有的功能。包括信号处理功能、传感器和执行器接口功能、通信功能、人机接口功能、编程、调试和测试功能及电源功能等。图 2-5 所示为可编程控制器的编程模型。

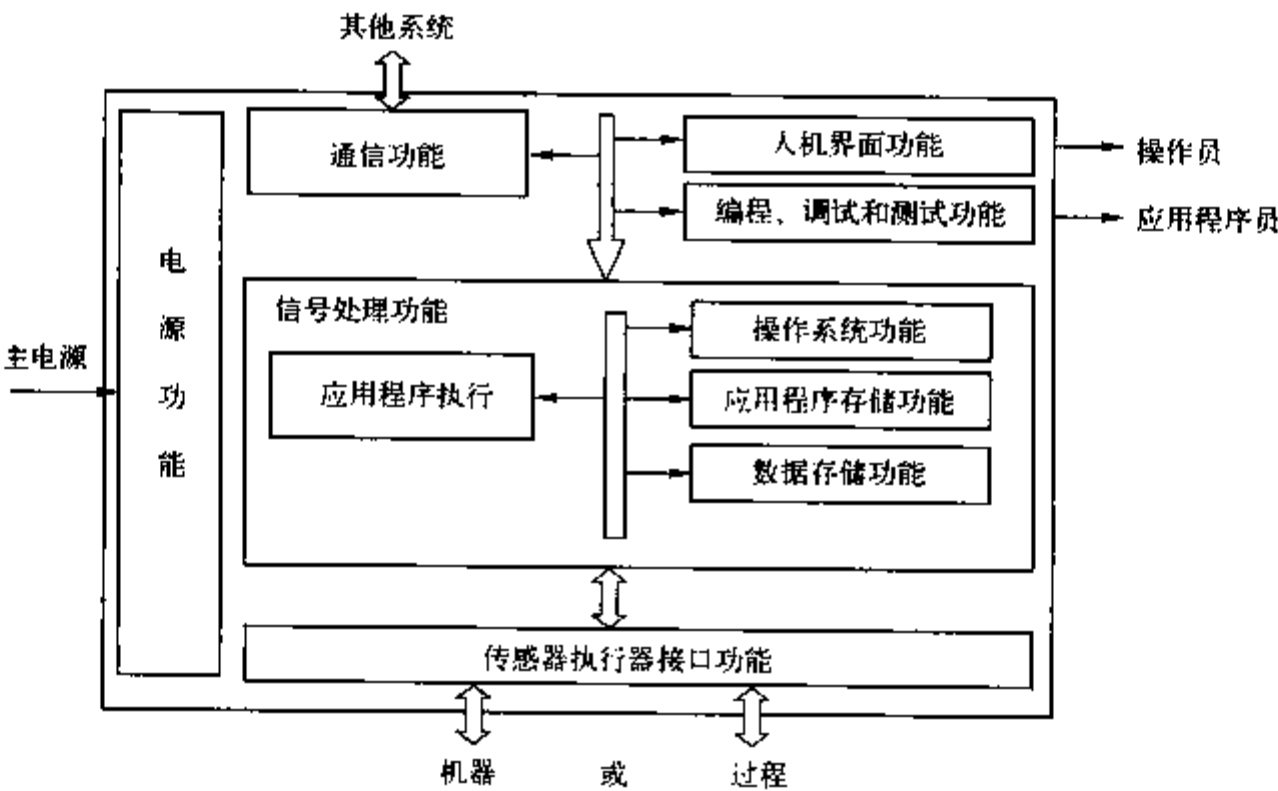


图 2-5 编程模型

(1) 信号处理功能

信号处理功能由应用程序寄存器功能、操作系统功能、数据寄存器功能、应用程序执行功能等组成。它根据应用程序,处理从传感器及内部数据寄存器获得的信号,处理后输出信号送执行器及内部数据寄存器。表 2-2 所示为可编程序功能

- 1) 应用程序寄存器,为寄存一系列指令提供存储器单元,指令的执行是周期性的或事件驱动的。
- 2) 应用数据寄存器,为应用程序执行所需的输入输出映像表和数据提供存储器单元。
- 3) 存储器类型、容量和利用,可用的存储器有 RAM、ROM、PROM、EEPROM 等。存储器容量与存储器单元数有关,它包括可用的最小配置容量和最大配置容量。应用程序所需存储器容量与可编程控制器所需功能和类型有关。应用数据寄存器的存储容量与被寄存的数据类型和数量有关。

表 2-2 可编程序功能

功能组别		示 例	功能组别		示 例
逻辑控制	逻辑	与、或、非、异或、触发	数据处理	人机接口	显示、命令
	定时器	接通延迟、断开延迟、定时脉冲		打印机	信息、报表
	计数器	脉冲信号加和减		大容量存储器	记录
	顺序控制	顺序功能表图		执行控制	周期执行、事件驱动执行
数据处理	数据处理	选择、传送、格式、传送、组织		系统配置	状态校验
	模拟数据	PID、积分、微分、滤波	运算	基本运算	加、减、乘、除、模除
	接口	模拟和数字信号的输入输出		扩展运算	平方、开方、三角函数
	其他系统	通信协议		比较	大于、小于、等于
	输入输出	BCD 转换			

4) 操作系统,负责对可编程控制器系统内部相互关联的功能进行管理。

5) 应用程序执行,应用程序的总响应时间是信号从现场输入端,经可编程控制器处理,最后到现场输出端所需各局部时间之和。

(2) 传感器执行器接口功能

将来自机器或过程的输入信号或数据转换为合适的信号电平,和将信号处理功能的输出信号和/或数据转换为合适的电平信号,传送到执行器或显示器。通常,它包括输入输出信号类型及输入输出系统特性的确定等。

1) 输入输出信号类型。来自机械或过程的状态信息和数据以二进制信号、数字信号、增量信号或模拟信号形式被传送到输入输出系统。合适的二进制信号、数字信号、增量信号或模拟信号把可编程控制器系统处理后的结果传送给机械或过程。为此,应有宽范围的输入输出信号,来适应这种类型的传感器和执行机构。

2) 输入输出系统特性。输入输出系统应使用各种信号处理、转换和隔离方法。为适应系统扩展需要,应能够扩展到最大配置。输入输出系统应可安装到紧靠信号处理功能的位置,也可远离信号处理功能而靠近执行机构的位置安装。

(3) 通信功能

提供与其他系统,如其他可编程控制器系统、机器人控制器、计算机等装置的通信,用于实现程序传输、数据文件传输、监视、诊断等。通常采用符合国际标准的硬件接口(如 RS232、RS485)和通信协议(如 X.25)等实现。

(4) 人机界面功能

它为操作员提供与信号处理、机器或过程之间信息相互作用的平台,也称为人机接口功能。主要包括为操作员提供机器或过程运行所需的信息,允许操作员干预可编程控制器系统及应用程序,如对参数调整和超限判别等。

(5) 编程、调试和测试功能

它可作为可编程控制器的整体,也可作为可编程控制器的独立部分来实现。它为应用程序员提供应用程序生成、装载、监视、检测、调试、修改及应用程序文件编制和存档的操作平台。

1) 应用程序写入,包括应用程序生成、应用程序显示等。应用程序的写入可采用字母、数字或符号键,也可应用菜单、下拉式菜单和鼠标、球标等光标定位装置。应用程序输入时应保

证程序和数据的有效性和一致性。应用程序的显示是在应用程序写入时,将所有指令能逐句或逐段立即显示。通常,可打印完整的程序。不同编程语言的显示形式可能不同,用户可选择合适的显示形式。

2) 系统自动启动,包括应用程序的装载、存储器访问、可编程控制器系统的适应性、系统自动状态显示、应用程序的调试和应用程序的修改等。可编程控制器系统的适应性是系统适应机械或过程的功能,包括对连接到系统的传感器和执行机构进行检查的测试功能、对程序序列运行进行检查的测试功能和常数置位和复位功能等。

3) 文件,包括硬件配置及与设计有关的注释的描述、应用程序文件、维修手册等。应用程序文件应包括程序清单、信号和数据处理的助记符、所有数据处理用的交叉参考表(输入/输出、内部储存数据、定时器、计数器等内部功能)、注释、用户说明等。

4) 应用程序存档,为提高维修速度和减少停机时间,应将应用程序存储在非易失性的存储介质中,并且应保证所存储的程序与原程序的一致性。

(6) 电源功能

提供可编程控制器系统所需电源,为设备同步起停提供控制信号,提供系统电源与主电源的隔离和转换等。可根据供电电压、功率消耗及不间断工作的要求等使用不同的电源供电。

可编程控制器的通信模型见有关资料。

2.2 公用元素

对所有编程语言,下面介绍的公用元素是公用的,即所有编程语言由这些公用元素规定。

2.2.1 字符集

根据国家标准 GB/T 15969.3—2005,可编程控制器使用的文本和图形类编程语言中的文本元素应依据国家标准 GB 1988 字符集的“基本代码表”的 3~7 列的字符组成,并根据 GB 2312—1980《信息交换用汉字编码字符集 基本集》来表示汉字。

支持小写字母时,字母的大小写具有相同的意义。例如,Control 与 CONTROL 是相同的变量名或标识符。

制造商根据表 2-3 字符集特性的规则选择。

表 2-3 字符集特性

序 号	选择 1	选择 2
1	数符号 #	英镑符号 £
2	美元符号 \$	货币符号 ¥
3	垂直线	惊叹号 !

1) 英镑符号应使用在数符号的位置,前者占据国家执行 GB1988 字符集的 2/3 字符位置。

2) 货币符号应使用在美元符号的位置,前者占据国家执行 GB1988 字符集的 2/4 字符位置。

3) 当 GB1988 字符集中 7/12 字符位置被国际字符集中的另外字符使用时,在 2/1 位置处的惊叹号! 被用于表示垂直线。

4) 汉字字符集依据 GB 2312—1980。国家标准字符集中字符的使用是典型的扩展应用。

2.2.2 标识符

根据 IEC 61131-3 标准,它必须由字母、数字和下划线字符组成,并被命名为语言元素(Language Element)。标识符用于表示在 IEC 语言中的不同元素,包括变量、标号和函数、功能块、程序组织单元等名称。例如,TIA_201 表示某个温度报警变量,START_WEIGHTING 表示某个开始称重的标号等。使用标识符的规则如下:

1) 标识符的第 1 个字符必须是字母或下划线,最后一个字符必须是字母或数字,中间字符只允许是字母、数字或下划线。因此,其他字符,如空格、钱币符号、小数点、各种括号等不允许作为标识符。例如,T_123、_123T 等是有效标识符,而 SW 1、T-123、V_123_、\$123、T12.3s 等都是无效的标识符。

2) 标识符中字母的大小写具有相同的意义。因此,标识符 SW_123 和 sw_123 是相同的标识符。

3) 下划线是标识符的一部分,但标识符中不允许有两个或两个以上连续的下划线。因此,_LIM_SW5 和 W123__PV 是不正确的标识符。由于,标识符的结尾不能用下划线。因此,LIM_SW5_也是不正确的标识符。

4) 标准规定,为便于识别,在支持使用标识符的所有编程系统中,至少应支持 6 个标识符,即对具有 6 个有效位的编程系统中,Control_123 与 Control_223 被系统认为是相同的标识符。因此,为防止对标识符的误解,编程员应确保所编写标识符的前 6 个字符是惟一的。一个标识符中允许的最多字符数是与执行有关的参数。表 2-4 是标识符的性能和示例。

表 2-4 标识符的性能和示例

序 号	特 性 描 述	示 例
1	大写字母和数字	SW123,U235,T123,TNT,BBC
2	大小写字母,数字,中间的下划线字符	上述所有加:LIM_SW_5,cal_123,CNN_BAD
3	大小写字母,数字,开头或中间的下划线字符	上述所有加:_MAN,_T_V_701,_CHINA_TIBET

2.2.3 分界符

分界符(Delimiter)用于分隔程序语言元素的字符或字符组合。它是专用字符,不同的分界符具有不同的含义。表 2-5 列出了各种分界符及应用示例。

表 2-5 分界符及应用示例

分 界 符	应 用 场 合	备注和示例
空格	允许在 PLC 程序中插入空格	不允许在关键字、文字和枚举值中插入空格
(*	注释开始符号	用户的注释。可设置在程序允许空格的任何位置 不允许注释嵌套,例如,不允许(* (* A=2 *) *)
*)	注释结束符号	
+	十进制数字的前缀符号	+456
	加操作	19+96

(续)

分界符	应用场合	备注和示例
-	十进制数字的前缀符号	-789
	年-月-日的分隔符	D#2005-05-19
	减操作	5-6
	水平线	图形编程语言中表示水平连接线
#	基底数的分隔符	2#1111_1111 或 16#FF(表示十进制 255)
	数据类型分隔符	SINT#234,INT#16000,BOOL#0
	时间文字的分隔符	T#14 ms,T#25h_15m,TOD#05:30:35.28
.	整数和小数的分隔符	3.14159265,2.718281828
	分级寻址分隔符	%IW2.5.7.1
	结构元素分割符	MOD_5_CONFIG.CHANNEL[5].RANGE
	功能块结构分割符	TMR_1.Q,SR_103.SI
e 或 E	实指数分界符	1.0e+6,1.2345E6
'	字符串开始和结束符号	'SWITCH'
\$	串中特殊字符的开始	'\$L'表示换行,'\$R'表示回车,'\$P'表示换页等
:	时刻文字分隔符	TOD#15:36:35.25
	类型名称/指定分隔符	REAL:1.0;
	变量/类型分隔符	ANALOG_DATA:INT(-4095..4095);
	步名称终结符	STEP STEP5:END_STEP
	程序名/类型分隔符	PROGRAM PI WITH PER_2;
	存取名/路径/类型分隔符	ABLE;STATION_1.%IX1.1;BOOL.READ_ONLY;
	指令标号终结符	L1;LD %IX1
	网络标号终结符	NEXT1;后接梯形图程序
:=	初始化操作符	MIN_VAL:INT:= -4095;
	输入连接操作符	TASK INT_2(SINGLE:= z2,PRIORITY:=1);
	赋值操作符	J:=J+2;
()	枚举表分界符	V:(BI_10V,UP_10V,UP_1_5V):=UP_1_5V;
	子范围分界符	ANALOG_DATA:INT(-4095..4095);
	多重初始化,重复因子	ARRAY(1..2,1..3) OF INT:=1,2,3(4),6;
	指令表修正符/操作符	(A>B)
	函数自变量	A+B-C*ABS(D)
	子表达式分级	(A*(B-C)+D)
	功能块输入表分界符	CMD_TMR(IN:=%IX5.1,PT:=T#100 ms);
[]	数组下标分界符	MOD_5_CFG.CH[5].RANGE:=BI_10V;
	串长度分界符	A_ARRAY[%MB6,SYM]=I_ARRAY[2]+I_ARRAY[5];

(续)

分界符	应用场合	备注和示例
,	枚举表分隔符	V:(BL_10V,UP_10V,UP_1_5V):= UP_1_5V;
	初始值分隔符	ARRAY(1..2,1..3) OF INT:=1,2,3(4),6;
	数组下标分隔符	ARRAY(1..2,1..3) OF INT:=1,2,3(4),6;
	被声明变量的分隔符	VAR_INPUT A,B,C;REAL;END_VAR
	功能块初始值分隔符	TERM_2 (RUN:=1,A1:=AUTO,XIN:=START);
	功能块输入表分隔符	SR_1 (S1:=%IX1,RESET:=%IX2);
	操作数表分隔符	ARRAY(1..2,1..3) OF INT:=1,2,3(4),6;
	函数自变量表分隔符	LIMIT (MN:=4.0,IN:=%IW0,MX:=20.0);
	CASE 值表分隔符	CASE TW OF 1,5:DISPLAY:=OVEN_TEMP;
;	类型分隔符	TYPE R;REAL;END_TYPE
	语句分隔符	QU:=5*(A+B);QD:=4*(A-B);
..	子范围分隔符	ARRAY(1..2,1..3)
	CASE 范围分隔符	CASE TW OF (1..5):DISPLAY:=OVEN_TEMP;
%	直接表示变量的前缀	%IX1.2,%QB5
=>	输出连接操作符	C10(CU:=%IX10,Q=>OUT);
或 !	垂直线	图形编程语言中表示垂直线

- 注:1. 用于逻辑运算和算术运算等的操作符号为中间操作符,如 NOT、MOD、+、-、*、/、**、<、>、<=、>=、=、<>、&、AND、OR、XOR。
2. 用于表示时间、时刻等时间文字的操作符号为时间文字分界符,如 T#、D、H、M、S、MS、DATE#、D#、TIME_OF_DAY#、TOD#15:36:35.25、DATE_AND_TIME#、DT#。

2.2.4 关键字

关键字(Keyword)是语言元素特征化的词法单元。关键字是特定的标准标识符。在 IEC 61131-3 标准中,关键字作为编程语言的字,被用于定义不同结构或启动和终止特定的软件元素。

部分关键字配对使用,如 FUNCTION 与 END_FUNCTION 等。部分关键字单独使用,如 TASK、ABS 等。关键字不能用于任何其他目的,如不能作为变量名或扩展名,即不能用 TON 作为变量名,不能用 VAR 作为扩展名等。

关键字是标准标识符,因此,不能包含空格。标准第 2 版将时间文字的有关关键字作为分界符处理;增加了 TEMP、CONFIG 变量的关键字类型;增加 EN、ENO 关键字;结构数据类型 STRUCTURE ...END_STRUCTURE 改为 STRUCT...END_STRUCT;将边沿检测 EDGE 分为上升沿检测 R_EDGE 和下降沿检测 F_EDGE;提供了存取方式的关键字等。表 2-6 列出了 IEC 61131-3 标准规定的关键字和示例。

表 2-6 关键字和示例

关 键 字	说 明	示 例
CONFIGURATION END_CONFIGURATION	配置段开始 配置段结束	见例 2-1
RESOURCE ON END_RESOURCE	资源段开始 资源段结束	见例 2-1
TASK	任务	见例 2-1, 例 2-2, 例 2-66
PRAGRAM END_PROGRAM	程序段开始 程序段结束	见例 2-1, 例 2-66, 图 2-26
PROGRAM WITH	与任务结合的程序	见例 2-1
FUNCTION END_FUNCTION	函数段开始 函数段结束	见例 2-28, 例 2-29
FNCTION_BLOCK END_FUNCTION_BLOCK	功能块段开始 功能块段结束	见图 2-17, 例 2-59, 例 2-60
ABS, ADD, GT, BCD_TO_INT 等	函数	见例 2-30 ~ 例 2-58
SR, TON, TOF, R_TRIG 等	功能块	见表 2-42 ~ 表 2-44, 例 2-62, 例 2-63
VAR END_VAR	内部变量段开始 变量段结束	见表 2-13, 表 2-18, 例 2-19
VAR_INPUT END_VAR	输入变量段开始 变量段结束	见例 2-20, 图 2-10
VAR_OUTPUT END_VAR	输出变量段开始 变量段结束	见例 2-21
VAR_IN_OUT END_VAR	输入输出变量段开始 变量段结束	见例 2-22, 表 2-41
VAR_GLOBAL END_VAR	全局变量段开始 变量段结束	见例 2-23, 表 2-21
VAR_EXTERNAL END_VAR	外部变量段开始 变量段结束	见表 2-26, 表 2-41
VAR_ACCESS END_VAR	存取路径变量段开始 变量段结束	见例 2-25, 例 2-1
VAR_TEMP END_VAR	暂存变量段开始 变量段结束	见例 2-24
VAR_CONFIG END_VAR	组态变量段开始 变量段结束	见例 2-1, 例 2-27
RETAIN NON_RETAIN	具有掉电保持功能的变量 不具有掉电保持功能的变量	见表 2-20
CONSTANT	常数变量	见表 2-19, 例 2-11
ARRAY OF	数组	见表 2-15, 例 2-15
INT, REAL, BOOL, WORD 等	数据类型名称	见表 2-13
AT	直接表示变量的地址	见表 2-20, 表 2-21
EN, ENO	使能端输入和输出	见图 2-8, 表 2-38
TRUE FALSE	逻辑真 逻辑假	见表 2-8, 表 3.3
TYPE END_TYPE	数据类型段开始 数据类型段结束	见表 2-14, 例 2-13 ~ 例 2-16
STRUCT END_STRUCT	结构段开始 结构段结束	见表 2-6, 例 2-16,

(续)

关 键 字	说 明	示 例
IF THEN ELSIF ELSE END_IF	选择语句 IF	见例 3 - 43 ~ 例 3 - 46
CASE OF ELSE END_CASE	选择语句 CASE	见例 3 - 47 ~ 例 3 - 49
FOR TO BY DO END_FOR	循环语句 FOR	见例 3 - 50, 例 3 - 51
REPEAT UNTIL END_REPEAT	循环语句 REPEAT	见例 3 - 53
WHILE DO END_WHILE	循环语句 WHILE	见例 3 - 52
WITH	与任务结合的程序组织单元	见例 2 - 1, 例 2 - 2, 例 2 - 66
RETURN	跳转返回符	见例 3 - 42
NOT, AND, OR, XOR	逻辑操作符	见表 2 - 29, 例 2 - 43
STEP END_STEP	步段开始 步段结束	见例 5 - 7, 例 5 - 8, 表 5 - 4
INITIAL_STEP END_STEP	初始步段开始 初始步段结束	见表 5 - 1, 表 5 - 3
TRANSITION FROM TO END_TRANSITION	转换段开始 转换段结束	见例 5 - 7, 例 5 - 8
ACTION END_ACTION	动作段开始 动作段结束	见表 5 - 3
R_EDGE F_EDGE	上升沿 下降沿	见表 2 - 43, 例 2 - 60
READ_WRITE READ_ONLY	读写 只读	见例 2 - 1, 例 2 - 25

下列功能模块和函数的标识符被保留作为关键字:

- 1) 标准数据类型名称: BOOL, REAL 等。
- 2) 标准函数名和标准功能块名: SIN, COS, RS, SR, TON 等。
- 3) 指令表语言中的文本操作符: LD, ST, ADD, DIV, GT, LE 等。
- 4) 结构化文本语言中的文本操作符: NOT, MOD, XOR, AND 等。

2.2.5 空格和注释

在程序文本的任何地方允许插入一个或多个空格(GB1988 字符集中 2/0 码位)。但在关键字、标识符、分界符等内不允许包含空格。表 2 - 7 是空格的应用和示例。

表 2 - 7 空格的应用及示例

序 号	特 性 描 述	示 例
1	允许的空格	LD %IX0.2;SR 1(SETI ; = START,RESET; = STOP);
2	不允许的空格	LD %IX0.2;SR1(SETI;=START,RESET;=STOP);

为说明程序中语句的功能,可在程序中添加注释。注释在以“(* ”开始,以“ *)”结束的分界符内。程序中允许空格的地方都可以添加注释。注释不应有语法或语义。

不允许嵌套注释,如表 2 - 5 所示。

2.3 数据外部表示

可编程控制器的编程语言中,数据外部表示(External Representation of Data)由数值文字、字符串文字和时间文字组成。

2.3.1 数值文字

数值文字(Numeric Literal)用于定义一个数值,它可以是十进制数或其他进制的数。数值文字分为两类:整数文字和实数文字。

一个数值文字可定义为二进制、八进制、十进制和十六进制数。十进制符号表示的数中,用小数点的是否存在表示它是实数或整数。十进制数值,为表示数值的正负,可在数值文字前添加前缀分界符,如-15,-273.15。布尔数据用值0或1的整数文字表示。

为说明数值的基,可用元素数据类型名称和“#”符号表示。例如,2#1111_1111与16#FF都表示十进制的255。十进制数的基10#不需要表示,因此,可直接表示为255。

数值的基(即2,8,10和16)不允许前置分界符(+或-)。因此,-8#340是错误的数据外部表示,应表示为8#-340。对基底为16时,应使用字母A~F的扩展数字,用于分别表示十进制的10~15。

在数值文字中单一的下划线字符是没有意义的,因此,3.14_159是数值文字,表示3.14159,2#1111_1111表示二进制值11111111,下划线的其他用法是不允许的。

在已知的执行过程中,每种数值文字的最大数应在整个范围内可以表示,并对所有数值类型有精确的值,这种数值类型在执行过程中用文字表示。表2-8是数值文字的性能和示例。

表 2-8 数值文字的性能和示例

数值文字类型	表示方法	示 例
整数文字	[整数类型名#] 符号整数或二(八、十、十六)进制整数	INT#-34_5,UINT16#3FAE
	符号整数	-12,0,123_45
	二进制整数	2#1101_1011(219)
	八进制整数	8#377(255)
	十六进制整数	16#A3(163)
实数文字	[实数类型名#] 符号整数. 整数[指数]	REAL#4.2,6.3e-7
	符号整数. 整数	3.1416,2.7_1828
布尔数	[布尔文字类型名#] 0或1,或FALSE或TRUE	0,1,BOOL#0
	[布尔文字类型名#] FALSE或TRUE	TRUE,BOOL#FALSE
整数类型名	有符号整数类型名:SINT,INT,DINT,LINT	SINT 表示单整数
	无符号整数类型名:USINT,UINT,UDINT,ULINT	UDINT 表示无符号双整数
实数类型名	REAL,LREAL	LREAL 表示长实数
布尔文字类型名	BOOL	BOOL 表示布尔数

注:方括号内的内容是可选项。因此,INT#34_5与345是等效的。下划线可增加可读性。UINT#16#9AF表示十六进制无符号整数9AF。

实数数据类型主要用于下列场合：

- 1) 处理模拟输入信号。例如,来自压力传感器、热电偶和转速表的信号。
- 2) 用于闭环控制。例如,进行 PID 运算。
- 3) 模拟输出。例如,输出模拟信号到阀门定位器等。

2.3.2 字符串文字

字符串文字(Character String Literal)由单字节字符串或双字节字符串组成。

单字节字符串文字由一系列通用的字节表示或\$'、英文双引号"、\$与十六进制数字组成。它们用单引号在其前后标识,如'ABC'、' '、' '\$'、'\$C4'等。当美元符号与两个十六进制数组成 3 字节字符串文字时,应将该字符串的两个十六进制数组成的数对应解释为 8 位字符码的十六进制数。例如,'\$0A'表示 LF 字符的长度为 1 的串,它与字符'\$L'等效。

双字节字符串文字由一系列通用的字节表示或\$"、英文单引号'、\$与十六进制数字组成。它们用双引号在其前后标识,如"A"、" ' "、""、"\$""、"\$R"、"\$00C4"等。当美元符号与 4 个十六进制数组成 5 字节字符串文字时,应将该字符串的 4 个十六进制数组成的数对应解释为 16 位字符码的十六进制数。例如,"\$R \$L"表示包含 CR 和 LF 字符的长度为 2 的双字节字符串。

字符串文字应用于下列场合：

- 1) 处理批量的标识,如'Batch number AX08_0321'。
- 2) 操作显示的报文,如'Start to feed'。
- 3) 经通信设备发送报文到另一个设备,如 PARTNER: = 'PLC1'。

表 2-9 是字符串文字的特性和示例。表 2-10 是字符串中双字符组合的解释。

表 2-9 字符串文字的特性和示例

单字节字符串示例	双字节字符串示例	说 明
''	""	长度为零的空串
'A'	"A"	单字符 A 的长度为 1 的字符串
' '	" "	空格字符的长度为 1 的字符串
'\$'	" '\$"	含单引号的长度为 1 的字符串
' "\$'	" '\$"	含双引号的长度为 1 的字符串
'\$R \$L'	"\$R \$L"	含 CR(回车)和 LF(换行)的长度为 2 的字符串
'\$\$5.00'	"\$\$5.00"	打印为\$5.00 的长度为 5 的字符串
STRING# 'OK'	WSTRING# 'OK'	字符为 OK 的长度为 2 的字符串
'AE'	"AE"	长度为 2 的字符串,C4 和 CB 是 A 和 E 的 ASCII 码
'\$C4 \$CB'	"\$00C4 \$00CB"	

注意,字符串可以是空串,如''或""。单字节字符串用双引号开始,不能用单引号开始。双字节字符串用单引号开始,不能用双引号开始。为了表示该字符串是单字节字符串,可用单字节字符串类型名 STRING 和#,如 STRING# 'YES'。同样,对双字节字符串,可用双字节字符串类型名 WSTRING 和#,如 WSTRING# "YES"。

用美元符号和单或双引号可以组成一些特殊字符。例如,要表示“`This 'book' cost $5`”,用单字节字符串,可表示为`'This $" book $" cost $$5'`,用双字节字符串,可表示为`"This $' book $' cost $$5"`。

字符串中双字符为组合形式见表 2-10。

表 2-10 字符串中双字符组合

组 合	打印时的解释	组 合	打印时的解释
<code>\$\$</code>	美元符号	<code>\$P 或 \$p</code>	换页
<code>\$'</code>	单引号	<code>\$R 或 \$r</code>	回车
<code>\$L 或 \$l</code>	换行	<code>\$T 或 \$t</code>	制表
<code>\$N 或 \$n</code>	新行	<code>\$"</code>	双引号

- 注1. 新行字符为物理的输入输出和文件的输入输出提供一个与执行有关的方法,用于定义数据行的结束,并另起一行。用于打印时,其效果是数据行的结束,并在下一行开始重新恢复打印。
2. `$'`的组合仅对双引号内的文字串有效。
3. `$"`的组合仅对单引号内的文字串有效。

2.3.3 时间文字

时间文字(Time Literal)用于表示时间。有 4 种与时间有关的数据类型提供外部表示。

1. 持续时间

持续时间(Duration)用于表达一个控制事件通过的时间。例如,定时器设定时间是一个持续时间。它表示定时器计时开始后,要持续到设定时间后才能有输出。持续时间应用于下列场合。

- 1) 定义过程阶段的持续时间。例如,热处理时工件在给定温度下保持的时间。
- 2) 定义特定事件发生的暂停时间。例如,规定发出阀关闭信号到阀真正闭合的持续时间,如果阀门没有在该时间内关闭则发送报警信号。

持续时间用左面的标识符 T 或 TIME 及时间文字分界符#与用天、小时、分、秒和毫秒或其任意组合表示的持续时间表示,如 `T#3D4H5M6S`。

持续时间标识符是 T 或 TIME,因字母的大小写具有相同意义,因此,持续时间标识符也可表示为 t 或 time。其中,T 或 t 称为短前缀,TIME 或 time 称为长前缀。

表示时间单位的天、小时、分、秒和毫秒等字母和持续时间标识符的大小写都具有相同意义。因此,`T#12.3s` 与 `t#12.3S` 等效。

时间文字分界符#用于分隔持续时间标识符和持续的时间。由于下划线没有意义,因此,持续时间文字单位可用下划线分隔。例如,`T#1D_2H_3M` 表示持续时间为 1 天 2 小时 3 分。它与 `T#1D2H3M` 具有相同的含义。用下划线时应注意,不能用连续的两个或两个以上的下划线来分隔。因此,`T#3H__2M` 是错误的持续时间表示。

持续时间的时间单位有天(D)、小时(H)、分(M)、秒(S)和毫秒(MS)。可以用小数表示持续时间,如 `T#12.3s`、`T#4.5d`。

持续时间的数值允许有负值,如 `T#-23.4h14.5m`,但不允许表示为 `-T#23.4h14.5m`。持续时间具有超载属性,即允许数据超过有效的时间单位。例如,`T#25h80m` 中 25 小时已经超过

1 天 24 小时的有效时间单位。同样,80 分钟也超出了分钟的有效时间单位,但这样的表示是允许的,编程系统会自动将它转换为不超过有效时间单位的数据,即它与 T#1d2h20m 是等效的。表 2-11 是持续时间文字的特性和示例。

表 2-11 持续时间文字的特性和示例

序 号	特 性 描 述	示 例
1a	不带下划线的持续时间	短前缀 T#14 ms,T#-12 ms,T#12.3s,T#1.2d,t#27h18m,t#1d2h3m4s5.6 ms
1b		长前缀 TIME#14 ms,time#12.3s,TIME#3d2h
2a	带下划线的持续时间	短前缀 T#27h_18m,t#1d_2h_3m_4s_5.6 ms
2b		长前缀 TIME#27h_18m,TIME#1d_2h_3m_4s_5.6 ms

2. 一天中的时间

一天中的时间(Time of Day)用于表示在一天中的时间,也称为时刻,例如,当前时间等。

一天中的时间用时刻标识符、时间文字分界符#和时刻表示。时刻标识符是 TOD 或 tod,它们称为时刻标识符的短前缀。TIME_OF_DAY 或 time_of_day 是时刻标识符的长前缀。例如,TOD#12:34:56 和 time_of_day#12:34:56 是等效的。

时刻文字分界符#用于分隔时刻标识符和时刻。时刻文字中,时刻的分界符用冒号,不用下划线。例如,TOD#12:34:56.25 表示在 12 时 34 分 56.25 秒。因此,TOD#12_34 和 TOD#12-34 都是错误的时刻表示。

时刻的单位是时、分、秒,没有毫秒单位,但可用小数表示时刻。例如,TOD#12:34:56.25 中 56.25 表示 56.25 秒。由于时刻是一天中的时间,因此,不需要天的时间单位。

时、分和秒的时间数据不能缺省,当最小时间单位没有时,可省略。例如,TOD#12 表示 12 时。

3. 日期

日期(Date)用于表示当天是某年某月某日。例如,当天是 2005 年 5 月 10 日,可表示为 D#2005-05-10。

日期和一天中的时间这两种时间文字用于下列场合:

- 1) 为了故障诊断和过程警告目的,记录事件和报警条件的日期和时间。
- 2) 某天、某周或某年内特定事件发生的控制。例如,每个周一早上反应器自动地被预先加热用于该周的批量运行,每天晚上 10 点打印日报表等。
- 3) 当电源故障和电源重新开始时,记录系统的停止时间。

日期用日期标识符、时间文字分界符#和日期表示。日期标识符的短前缀用 D 或 d 表示,长前缀用 DATE 或 date 表示。因此,D#2008-08-08 与 DATE#2008-08-08 等效。

时间文字分界符#用于分隔日期标识符和日期。日期文字中,日期的分界用连字号“-”,不用冒号“:”或下划线。此外,由于表示日期,因此,年、月和日的数据不能缺省。例如,不能表示为 D#08-08。

4. 日期和时刻

日期和时刻(Date and Time)用于表示某年某月某日某时某分某秒某毫秒的时间。

由于它既有日期,又有时刻,因此,须注意时间之间分界符的不同。

- 1) 年、月和日的分界符,与日期分界符相同,用连字号“-”分界。

2) 时、分、秒和毫秒的分界符,与时刻分界符相同,用冒号“:”分界。

3) 日期与时刻时间之间用连字号分界。例如,DT#2008-05-12-14:28:04 表示 2008 年 5 月 12 日 14 时 28 分 04 秒。

须注意,持续时间文字中的数据可以缺省,如 T#3h2s、t#5D38S 等。但后 3 种时间文字中的数据不能缺省。只有当最小时间单位没有时,才可以省略该部分。例如,TOD#12:20 表示 12 时 20 分。DT#2008-05-12-14:28 表示 2008 年 5 月 12 日 14 时 28 分。

标准规定字母的大小写没有意义。因此,只要时间文字分界符的字母与上述 4 种类型一致,都表示对应的时间。例如,Dt#2005-05-19-21:05:8.25 表示日期和时刻,tod#15:36:22.25 表示一天中的时间等。表 2-12 所示是其他 3 种时间文字的特性和示例。

表 2-12 其他 3 种时间文字的特性和示例

序 号	特 性 描 述		示 例
1a	一天中的时间	短前缀	TOD#15:30:22.25,tod#12:12
1b		长前缀	TIME_OF_DAY#15:30:22.25,time_of_day#12:12
2a	日 期	短前缀	D#2008-08-08,d#2008-05-12
2b		长前缀	DATE#20080-08-08,date#2008-05-12
3a	日期和时刻	短前缀	DT#2008-08-08-18:30:22.25,dt#2005-05-19-21:05:08.25
3b		长前缀	DATE_AND_TIME#2008-08-08-18:30:22.25 date_and_time#2008-05-12-14:28

2.4 数据类型

IEC 61131-3 的数据类型分为基本数据类型、一般数据类型和衍生数据类型三类。数据类型与它在数据存储器中所占用的数据宽度有关。

定义数据类型可防止因对数据类型的不同设置而发生出错。数据类型的标准化是编程语言开放性的重要标志。

2.4.1 基本数据类型

基本数据类型(Elementary Data Type, EDT)是在标准中预先定义的标准化数据类型。它有表 2-13 所示的约定关键字、数据元素位数、数据允许范围及约定的初始值。

基本数据类型名可以是数据类型名、时间类型名、位串类型名、STRING、WSTRING 和 TIME。

表 2-13 基本数据类型的性能

数 据 类 型	关 键 字	位 数(N)	允许取值范围	约定初始值
布尔	BOOL	1	0 或 1	0
短整数	SINT	8	-128 ~ +127,即 $-2^7 \sim 2^7 - 1$	0
整数	INT	16	-32768 ~ 32767,即 $-2^{15} \sim 2^{15} - 1$	0
双整数	DINT	32	$-2^{31} \sim 2^{31} - 1$	0

(续)

数据类型	关键字	位数(N)	允许取值范围	约定初始值
长整数	LINT	64	$-2^{63} \sim 2^{63} - 1$	0
无符号短整数	USINT	8	$0 \sim +255$, 即 $0 \sim 2^8 - 1$	0
无符号整数	UINT	16	$0 \sim +65535$, 即 $0 \sim 2^{16} - 1$	0
无符号双整数	UDINT	32	$0 \sim +2^{32} - 1$	0
无符号长整数	ULINT	64	$0 \sim +2^{64} - 1$	0
实数	REAL	32	按 SJ/Z9071 对基本单精度浮点格式的规定	0.0
长实数	LREAL	64	按 SJ/Z9071 对基本双精度浮点格式的规定	0.0
持续时间	TIME			T#0s
日期	DATE			D#0001-01-01
时刻	TOD			TOD#00:00:00
日期和时刻	DT			DT#0001-01-01-00:00:00
变量长度单字节字符串	STRING	8	与执行有关的参数	"单字节空串"
8 位长度的位串	BYTE	8	$0 \sim 16\#\text{FF}$	
16 位长度的位串	WORD	16	$0 \sim 16\#\text{FFFF}$	
32 位长度的位串	DWORD	32	$0 \sim 16\#\text{FFFF_FFFF}$	
64 位长度的位串	LWORD	64	$0 \sim 16\#\text{FFFF_FFFF_FFFF_FFFF}$	
变量长度双字节字符串	WSTRING	16	与执行有关的参数	"双字节空串"

注:1. 对时间数据类型和字符串数据类型,它们的数据宽度和数据允许范围与具体的实现有关。

2. SJ/Z9071 标准与 IEC 60559 标准等价。

3. 第 1 版的 EDGE 边沿检测布尔量数据类型已取消,第 2 版增加了 WSTRING 双字符串数据类型。

基本数据类型的允许范围是这类数据允许的取值范围。约定初始值是在对该类数据进行声明时,如果没有赋初始值时取用的是由系统提供的约定初始值。

2.4.2 一般数据类型

一般数据类型(Generic Data Type,GDT)用前缀“ANY”标识。它采用分级结构,见图 2-6。使用一般数据类型时应遵循图 2-6 所示规则。

- 1) 用户说明的程序组织单元不能用一般数据类型。
- 2) 子范围衍生类型的一般数据类型(见表 2-14 序号 3)应为“ANY_INT”。
- 3) 直接衍生类型的一般数据类型(见表 2-14 序号 1)与由此基本元素衍生的一般数据类型相同。
- 4) 所有其他衍生类型的一般数据类型在表 2-14 中定义为“ANY_DERIVED”。

在标准函数和标准功能块的输入和输出连接时使用一般数据类型的数据,详见下述。

2.4.3 衍生数据类型

衍生数据类型(Derived Data Type,DDT)是用户在基本数据类型的基础上,建立的由用户定义的数据类型,因此,也称为导出数据类型。这类数据类型所定义的变量是全局变量。它可用与基本数据类型所使用的相同方法对变量进行声明。

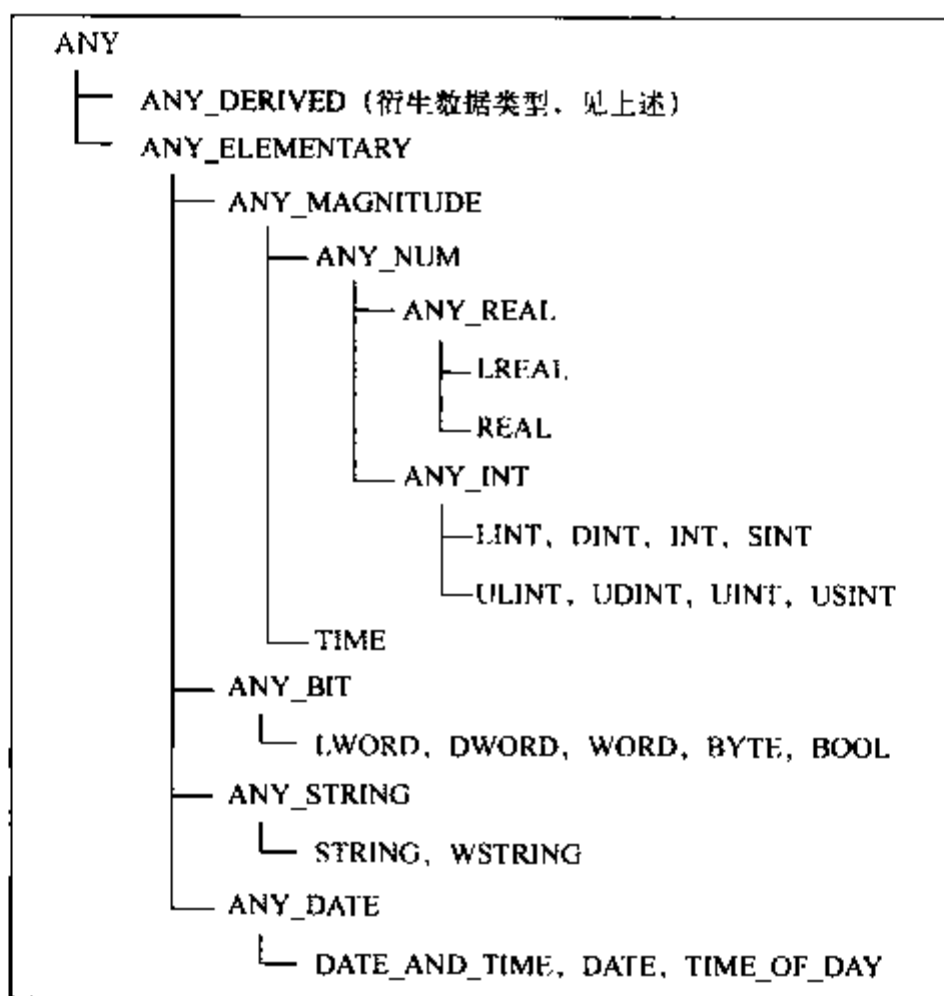


图 2-6 一般数据类型的分级

表 2-14 是衍生数据类型的特性。一般数据类型中衍生数据类型的确定见上述。5 种衍生数据类型都用 TYPE...END_TYPE 的文本结构声明。

表 2-14 衍生数据类型的特性

序 号	衍生数据类型特性	示 例	说 明
1	直接衍生的数据类型	TYPE PI; REAL; = 3.1415927; END_TYPE	PI 衍生数据类型用于表示 REAL 实数数据其初始值是 3.1415927
2	枚举数据类型	TYPE AI_Signal; (Single_Ended, Differential); END_TYPE	AI_Signal 是枚举数据类型, 它有两种数据类型: Single_Ended (单端) 和 Differential (差分)
3	子范围数据类型	TYPE Analog; INT(0..16000); END_TYPE	Analog 数据类型是整数数据类型, 其允许范围为 0 ~ 16000
4	数组数据类型	TYPE AI; ARRAY [1..5, 1..8] OF Analog; = (20(0), 20(16000)); END_TYPE	AI 数组数据类型是 5 × 8 维数组, 其数据元素的数据类型由 Analog 确定。其中, 前 20 个初始值为 0, 后 20 个的初始值为 16000
5	结构化数据类型	TYPE AI_Board; STRUCT Range; SIGNAL_RANGE; Min; Analog; Max; Analog; END_STRUCT END_TYPE	AI_Board 数据类型是结构化数据, 由 Range、Min 和 Max 组成。其中, Range 的数据类型是 SIGNAL_RANGE, Min 和 Max 的数据类型是 Analog

衍生数据类型有 5 种。分别是基本数据类型直接衍生的数据类型、枚举数据类型、子范围数据类型、数组数据类型和结构化数据类型等。

(1) 直接衍生的数据类型

直接衍生的数据类型见表 2-14 序号 1。示例中,用户用缩写的 PI 来表示实数数据类型 REAL,其初始值是 3.1415927。因此,经用这种方式衍生的数据类型,在以后的应用中就可直接用 PI 表示实数数据类型,即圆周率。

(2) 枚举数据类型

枚举数据类型见表 2-14 序号 2,它是衍生数据类型。示例声明衍生数据类型 AI_Signal 由两种信号的数据类型组成,它们是 Signal_Ended 和 Differential。因此,变量可以用枚举表中的某一个数据类型名作为其数值。

【例 2-3】 枚举数据类型示例。

所有设备在特定系统内可以有不同的操作模式。可用枚举数据类型表示。

```
TYPE DEVICE_MODE;  
    (INITIALISING,RUNING,STANDBY,FAULTY);  
END_TYPE
```

示例中规定 DEVICE 有初始化、运行、准备和故障 4 种操作模式。

该设备的运行模式可表示为 DEVICE_MODE#RUNING。

(3) 子范围数据类型

子范围数据类型见表 2-14 序号 3。当衍生数据类型的数据范围在该数据类型允许的范围内时,需要定义子范围数据类型。例如,基本数据类型 INT 的允许取值范围是 -32768 ~ 32767,如果某类数据只允许取值为 0 ~ 16000,则需要定义子范围数据类型,如示例所示。例如,模拟量被量化为整数 0 ~ 16000,在编程时就可使用该子范围数据类型的整数表示。

【例 2-4】 子范围数据类型示例。

为限定直流电动机的输入电压为 -6.0 ~ 12.0 V,可设置子范围数据类型如下:

```
TYPE  
    MOTOR_VOLTS:REAL(-6.0..12.0);  
END_TYPE
```

因此,采用子范围数据类型可约束数据的实际允许范围。

(4) 数组数据类型

数组数据类型见表 2-14 序号 4。一个数组由多个相同数据类型的数据元素组成。因此,数组数据定义为衍生数据类型。数组数据类型用 ARRAY 表示,用方括号内的数据定义其范围。当维数大于一维时,用逗号分隔,如表 2-14 所示,数组 AI 由 5×8 维数据组成。

【例 2-5】 数组数据类型示例。

数组数据类型被广泛应用于存储数组变量或多元素变量。

某反应器有 9 个温度检测点,可用数组数据表示如下:

```
TYPE REACTOR_TEMP_DATA;  
    ARRAY[1..3,1..3] OF TEMPERATURE;  
END_TYPE
```

示例表示反应器温度分 3 层,每层 3 个检测点,用 REACTOR_TEMP_DATA 数据变量表示。

(5) 结构化数据类型

结构化数据类型见表 2-14 序号 5。结构化数据用 STRUCT 关键字开始,中间是结构化数据的说明,最后用 END_STRUCT 结束。结构化数据类型用于将多个不同的数据类型组合在一起。第 1 版采用关键字 STRUCTURE...END_STRUCTURE 对结构化数据类型界定。

【例 2-6】 结构化数据类型示例

对上述示例的 TEMPERATURE 可采用结构化数据表示如下:

```
TYPE TEMPERATURE;  
  STRUCT  
    INPUT:      TEMP;  
    STATUS:     BOOL;  
    RANGE:      REAL;  
    CALIBRATION: DATE;  
    HIGH_LIMIT: REAL;  
    LOW_LIMIT:  REAL;  
    ALARM_COUNT: INT;  
  END_STRUCT  
END_TYPE
```

2.4.4 数据类型的允许取值范围和初始化

1. 不同数据类型的允许取值范围

除一般数据类型外,基本数据类型和衍生数据类型都有该类型数据的允许取值范围。基本数据类型的允许取值范围见表 2-13。

衍生数据类型应根据其衍生数据的类型,有不同的允许取值范围。

1) 直接衍生的数据类型与原数据类型的允许取值范围一致,见表 2-13。

2) 枚举数据类型的允许取值范围应根据枚举表列举的数据范围取值。枚举表是枚举数据值的有序集,它有一个最小数据值,位于枚举表的开始,最大数据值位于枚举表的结束。对枚举值,不同枚举数据类型可使用相同的标识符。最大允许的枚举值是一个与执行有关的参数。为在特定上下文使用时能够惟一识别,枚举文字可以用一个前缀限定。前缀由它们有关的数据类型名称和“#”符号组成,如 SINT#等。

3) 子范围数据类型的取值范围由子范围确定。因此,子范围数据类型取值只能取在特定的上限和下限之间,包括上下限。如果子范围数据值落在特定的取值范围之外则出错。例 2-4 中,MOTOR_VOLTS 的允许取值范围只能在 -6.0 ~ 12.0 V 内。

4) 数组数据类型的取值根据该数据类型中单元素的数据类型取值范围确定。例如,该元素的数据类型是 INT,则取值范围是 -32768 ~ 32767。

5) 结构化数据类型规定这类数据元素应包含能由特定名称存取的特定子元素。例如, AI_Broad 数据类型的元素是 Range、Min 和 Max,它们的数据类型分别是 SIGNAL_RANGE, Analog和 Analog。这些数据类型是衍生数据类型,因此,应分别根据它们的数据类型所允许的

取值范围确定。例如,如果是 Analog 数据类型,则根据表 2 - 14 的序号 3 确定,其取值范围是 0 ~ 16000 的整数。

2. 初始化

数据类型的初始化是可编程控制器启动时对有关变量赋予初始值的过程。当对变量不提供初始值时,初始值直接采用系统中该数据类型约定的初始值。当在变量声明中说明变量所赋予的初始值时,该变量被赋予由变量声明所指定的初始值。

1) 基本数据类型的约定初始值见表 2 - 13。衍生数据类型应根据其衍生数据的类型,有不同的约定初始值。

2) 直接衍生数据类型的约定初始值与原数据类型的约定初始值相同。例如,表 2 - 14 序号 1 的 PI,由于约定其初始值,因此,初始值是 3. 1415927,不是 0. 0。

3) 枚举数据类型的约定初始值是枚举表中的第 1 个枚举值。例如,表 2 - 14 序号 2 中 AI_Signal 的约定初始值由 Signal_Ended 确定。

4) 子范围数据类型的约定初始值是该子范围中的第 1 个限值。例如,表 2 - 14 序号 3 中子范围数据类型 Analog 的取值范围是 0 ~ 16000 的整数,而约定初始值是 0。

5) 数组数据类型的约定初始值是其基本数据类型的约定初始值。初始化表给出的初始值个数超过数组项的个数,则超过的(最右面)的初始值被忽略。如果初始值个数小于数组项的个数,则余下的数组项用相应数据类型的默认初始值填充。

6) 结构化数据类型由多种不同数据类型组合,根据各自约定初始值确定其初始值。

当某数据类型需要由用户赋予特定的初始值时,可直接用赋值符。其格式是:

数据类型: = 特定初始值;

数据类型赋初始值的示例见表 2 - 15。

表 2 - 15 数据类型赋初始值

数据类型	示 例	说 明
基本数据类型	VAR II:INT; END_VAR	变量 II 是整数,初始值未赋值,因此,用系统约定的初始值 0
	VAR RR:REAL; = 6. 85; END_VAR	变量 RR 是实数,初始值为 6. 85
	VAR STR:STRING; END_VAR	变量 STR 是单字节字符串,初始值是空串
枚举数据类型	TYPE ANALOG_SIGNAL_RANGE; (BIPOLAR_10V,(* -10V TO +10V DC *) UNIPOLAR_10V,(* 0V TO +10V DC *) UNIPOLAR_1_5V,(* 1V TO +5V DC *) UNIPOLAR_0_5V,(* 0V TO +5V DC *) UNIPOLAR_4_20mA,(* +4mA TO +20mA DC *) UNIPOLAR_0_20mA (* 0mA TO +20mA DC *)); = UNIPOLAR_1_5V ; END_TYPE	枚举数据 ANALOG_SIGNAL_RANGE,如果没有设置初始值,则初始值是枚举表中第 1 个枚举数据 BIPOLAR_10V 的约定初始值 - 10 V。本例中,初始值是 UNIPOLAR_1_5V 的约定初始值 1 V
子范围数据类型	TYPE DD:INT (-2047.. 2048); END_TYPE	子范围数据 DD,初始值为 -2047

(续)

数据类型	示 例	说 明
数组数据类型	TYPE AI_DATA_1; ARRAY [1..8] OF A_D; = {4(1024),4(-1024)}; END_TYPE	数组数据类型的变量 AI_DATA_1 有 8 个元素,它的前 4 个元素的初始值均为 1024,后 4 个元素的初始值均为 -1024
结构化数据类型	TYPE AI_Board; STRUCT Range:SIGNAL_RANGE; Min;Analog; = 4; Max;Analog; = 20; END_STRUCT END_TYPE	结构化数据类型的变量 AI_Board 中, Range 数据类型是 SIGNAL_RANGE,其初始值由该数据类型初始值确定; Min 的数据类型是 Analog,初始值为 4; Max 的数据类型是 Analog,初始值是 20

7) 类型 STRING 和 WSTRING 元素的默认最大长度是与执行有关的参数,除非在相关的声明中用括号括起来的最大长度另有规定。例如,类型 STR10 声明为

```
TYPE STR10;STRING[10]; = 'ABCDEF';END_TYPE
```

上述类似 STR10 的声明表示类型 STR10 的最大长度、默认初始值和默认初始长度分别是 10 个字符,'ABCDEF'和 6 个字符。

2.4.5 衍生数据类型的应用准则

应用衍生数据类型时应遵循下列准则。

- 1. 衍生数据类型是用户为应用需要而定义的数据类型
因此,衍生数据类型应便于建立数据模型,即衍生数据类型应是面向应用的数据类型。
- 2. 衍生数据类型可作为其他衍生数据类型的基础

衍生类型的单元素变量能够使用在它的父本类型变量能够使用的任何地方。例如,衍生数据类型的数组或结构可作为另一个衍生数据的数据类型,即衍生数据类型可以嵌套。但须注意,如果这种嵌套造成递归,则这种嵌套是非法的。

【例 2-7】 衍生数据类型的递归使用。

```
TYPE R1;REAL; = 2.0 ;END_TYPE
TYPE R2;R1 ;END_TYPE
VAR
  R3;R2 ;
END_VAR
```

例 2-7 中,R2 的数据类型是 R1,即实数,初始值为 2.0,它作为变量 R3 的数据类型。

【例 2-8】 嵌套结果造成递归的示例。

```
TYPE
  S1
  STRUCT
```

```

    A1:INT;
    A2:S2;
    A3:STRING:= ' WRONG ';
END_STRUCT;
S2
STRUCT
    A1:S1;
    A2:REAL;
    A3:BOOL:= ' FALSE ';
END_STRUCT;
END_TYPE

```

例 2-8 中,结构化数据类型 S1 中的 A2 使用结构化数据类型 S2,但因结构化数据类型 S2 中的 A1 又使用 S1,造成了递归结构。因此,这种嵌套递归结构的结构化数据类型是非法的,在编程时必须防止。

3. 数组数据类型中,为赋予初始值,可采用重复因子

当数组中有连续的若干数据类型都需要赋予相同的初始值,则圆括号前的数值说明相同的数据类型个数,圆括号内的值是其初始值。

【例 2-9】 重复因子的使用。

```

TYPE
    A1:ARRAY[1..8] OF UINT := 8(4);
END_TYPE

```

例 2-9 中,初始值为 4,相同的初始值共 8 个。因此,示例声明 A1 数据是数组数据类型,共有 8 个元素,采用无符号整数的基本数据类型,初始值都为 4。

4. 数组数据类型中,为赋予初始值,可用枚举方式

枚举表在方括号内列出,各初始值用逗号分隔。如果其中有部分数据类型有相同的初始值,也可在子序列中使用重复因子。须注意,与基本数据类型的初始化类似,如果使用标准约定的初始值,也可不列出。

【例 2-10】 枚举数据和子序列的使用。

```

TYPE
    A11:ARRAY[1..8] OF REAL:= [4,20,4,5(20)];
    D11:ARRAY[1..16] OF BOOL:= [3(0),5(1,0),2(1)];
END_TYPE

```

例 2-10 中,A11 数组数据类型由 8 个实数类型的数据元素,其中,A11[1]:=4;A11[2]:=20;A11[3]:=4;A11[4..8]:=20。D11 数组有 16 个数据元素,其中,D11[4]:=1;D11[5]:=0;D11[6]:=1;D11[7]:=0;D11[8]:=1;D11[9]:=0;D11[10]:=1;D11[11]:=0;D11[12]:=1;D11[13]:=0;D11[14..15]:=1。由于 D11 数组第 16 个元素没有特定初始值,因此,赋予约定初始值 0,即 D11[16]:=0。

5. 常数的初始值可直接用赋值符号将常数赋予该数据类型

须注意,常数是基本数据类型的特殊类型。因此,用 CONSTANT 标识符标识。常数具有

掉电保持属性。

【例 2-11】 常数赋予初始值。

```
VAR CONSTANT
    PI:REAL:=3.1416;
END_VAR
```

例 2-11 中,PI 变量是常数,其数据类型是基本数据类型中的实数,其初始值为 3.1416。

6. 功能块实例的初始化,可分别对其输入和输出赋予初始值

【例 2-12】 功能块实例的初始化。

```
VAR
    TIC_101:PID ( Kc:=1.5,Ti:=T#3.5M,Td:=T#1M);
END_VAR
```

例 2-12 中,功能块 PID 的实例名是 TIC_101,PID 功能块有 3 个参数,分别是增益 Kc,积分时间 Ti 和微分时间 Td。其中,Kc 初始值是 1.5。积分时间和微分时间的初始值用时间文字中的持续时间 T#表示。因此,积分时间初始值为 3.5 分,而微分时间初始值为 1 分。

【例 2-13】 枚举数据类型的应用。

```
TYPE
    VALVE_MODE:( OPEN,SHUT,FAULT );
    PUMP_MODE:( RUNNING,OFF,FAULT );
END_TYPE;
...
IF AX100 = PUMP_MODE#FAULT THEN XV23 = VALVE_MODE#OPEN;
```

例 2-13 中,定义了阀和泵的几种操作模式。当泵模式为故障时,应打开阀,即特定枚举文字的数据类型能够用前缀〈数据类型#〉来特定。

7. 子范围数据类型通常约束其数据范围

设置子范围数据类型的数据范围可确保子范围数据类型的变量仅能够设置数值在该范围内。

【例 2-14】 子范围数据类型的数据范围。

```
TYPE
    MOTOR_VOLTS:INT(-6..12);
END_TYPE
```

例 2-14 中,定义直流电动机的电压范围是 -6 ~ 12 V,从而保证当电压超出范围时,数据仍能在该范围内。

8. 数组数据类型常用于存储数组的值,有时可作为多元素变量,存放有关数据

【例 2-15】 用于不同检测点的一系列操作压力的数据表示。

某罐区各罐压力可采用数组数据表示为:

```
TYPE VESSEL_MATRIX;
    ARRAY[1..3,1..4] OF VESSEL_PRESSURE_DATA;
END_TYPE
```

例 2-15 定义了 3 个区域,每个区域有 4 个压力检测点的数组数据。

9. 结构化数据类型的约定初始值可在数据类型定义时重新定义

【例 2-16】 结构化数据类型定义新的约定初始值。

```
TYPE
PRESSURE;REAL:= 0.1;          (* 约定压力值为 0.1 MPa *)
END_TYPE;
TYPE PRESSURE_SENSOR;
STRUCT
    INPUT;PRESSURE := 0.2;      (* 过压约定值为 0.2 MPa *)
    STATUS;BOOL:= 0;           (* 约定为 0 *)
    CALIBRATION;DATE:= DT#2007-06-01; (* 约定安装时间为校验时间 *)
    HIGH_LIMIT;REAL:= 3.0;      (* 约定的限值为 3.0 MPa *)
    ALARM_COUNT;INT:=0;         (* 约定没有报警 *)
END_STRUCT
END_TYPE;
TYPE GAS_PRESS_SENSOR;
PRESSURE_SENSOR( PRESSURE := 0.4;      (* 重新定义约定压力值为 0.4 MPa *)
HIGH_LIMIT:= 4.0);                (* 重新定义约定限值为 4.0 MPa *)
END_TYPE
```

例 2-16 中,GAS_PRESS_SENSOR 是从 PRESSURE_SENSOR 衍生的数据类型,对 PRESSURE 和 HIGH_LIMIT 重新约定了初始值。

2.5 变量

与数据的外部表示相反,变量提供能够改变其内容的数据对象的识别方法。例如,可改变与可编程控制器输入和输出或存储器有关的数据。变量被声明为基本数据类型、一般数据类型或衍生数据类型。

2.5.1 变量的表示

在 IEC 61131-3 标准的第 1 版和第 2 版中,对变量的分类有所不同。变量分为单元素变量和多元素变量两类。

1. 单元素变量

单元素变量(Single-Element Variable)表示基本数据类型的单一数据元素、衍生的枚举数据类型或衍生子范围数据类型的数据元素、或上述数据类型的衍生数据元素。单元素变量可以是直接(表示)变量或符号变量。

(1) 直接变量

直接变量(Direct Variable)用百分数符号“%”(GB1988 码表中位置 2/5)开始,随后是位置前缀符号和大小前缀符号,如果有分级,则用整数表示分级,并用小数点符号“.”分隔的无符号整数表示直接表示变量,如%IX0.0,%QW0 等。

直接变量类似传统可编程控制器的操作数,它对应于某一可寻址的存储单元。标准对存储单元进行分类,如输入单元、输出单元等。

直接变量也称为直接表示变量。它用于程序、功能块、配置和资源的声明中。一个可编程控制器系统程序到另一个可编程控制器中存取数据的分级寻址的使用是语言的扩展。

表 2 - 16 是直接表示变量中前缀符号的定义和特性。表 2 - 17 是直接表示变量的示例。

分级用小数点符号表示。最左侧的域是分级的最高层,较低的域相继出现在它的右面,如表 2 - 17 中的%IW2.3.4.5。

表 2 - 16 直接表示变量中前缀符号的定义和特性

序 号	前 缀 符 号		定 义	约定数据类型
1	位置前缀	I	输入单元位置	
2		Q	输出单元位置	
3		M	存储器单元位置	
4	大小前缀	X	单个位	BOOL
5		无	单个位	BOOL
6		B	字节位(8 位)	BYTE
7		W	字位(16 位)	WORD
8		D	双字位(32 位)	DWORD
9		L	长(4)字位(64 位)	LWORD
10	*		在 VAR_CONFIG...END_VAR 结构声明中,“*”表示还未特定位置的内部变量	

表 2 - 17 直接表示变量的示例

示 例	说 明
%IX1.3 或%I1.3	表示输入单元 1 的第 3 位
%IW4	表示输入字单元 4(即输入单元 8 和 9)
%QX75 和%Q75	表示输出位 75
%MD48	表示双字,位于存储器 48
%Q*	表示输出在一个未特定的位置
%IW2.3.4.5	表示 PLC 系统第 2 块 I/O 总线的第 3 机架(Rack)上第 4 模块的第 5 通道(字)输入

(2) 符号变量

符号变量(Symbolic Variable)用符号表示的变量。其地址对不同的可编程控制器可以不同,从而为程序的移植创造条件。例如,VAR_INPUT SW_1 AT %IX1.1;BOOL;END_VAR 的变量声明中,用符号变量 SW_1 表示从%IX1.1 地址读取布尔量,当实际地址更改时,在程序的其他部分仍是该符号变量。因此,只需对该地址进行修改,就可完成整个程序的移植,实现程序的重复使用。AT 是定位前缀关键字。

2. 多元素变量

多元素变量(Multi-Element Variable)包括衍生数据类型中数组数据类型的变量和结构数据类型的变量。

(1) 数组数据类型的变量

一个数组(Array)是一个同样数据类型数据元素的组合,它们是用方括号括起来的一个或多个下标参数并以逗号分隔。数组数据类型的变量也称为数组变量,它用符号变量名,随后是下标表表示,下标表被包含在一对方括号内。它是一系列用表达式表示的下标,用逗号分隔。例如,数组变量 AI:ARRAY [1..3,1..8] OF REAL 表示数组变量 AI,它是由 3×8 个实数数据类型的变量组成。各组成变量是 AI[1,1],...,AI[1,8],AI[2,1],...,AI[2,8],AI[3,1],...,AI[3,8]。

在结构化文本编程语言中,数组变量的使用示例如下:

```
OUTARY [%MB6,SYM]; = INARY[0] + INARY[7] -INARY[%MB6] * %IW62;
```

(2) 结构数据类型的变量

结构数据类型的变量也称为结构变量,由结构变量名组成,用于声明前面已经规定为数据结构类型的变量。

结构变量的一个元素可表示为用小数点“.”分隔的两个或多个标识符或数组变量。第一个标识符表示结构元素的名称,其后的标识符表示在数据结构内存取特定数据的组件名顺序。例如,表 2-15 中,如果结构变量 MODULE_5 已经被声明为 AI_Board 的数据类型,则在用结构化文本编程语言编写程序时,MODULE_5.Range 是 SIGNAL_RANGE,而根据枚举数据类型的定义,其值为 BIPOLAR_10V。因此有

```
MODULE_5.Range:= BIPOLAR_10V;
```

2.5.2 变量的属性和初始化

1. 变量的类型和属性

IEC 61131-3 标准对变量定义了属性。通过设置变量属性将它们的有关性能赋予变量。第 2 版对变量的属性分类有增加。变量也可根据其应用范围分类。表 2-18 是变量的类型和属性表。除了上述变量类型外,IEC 61131-3 标准还对变量提供变量的附加属性,见表 2-19。

表 2-18 变量的类型和属性

变量类型关键字	变量属性	外部读写	内部读写
VAR	内部变量,程序组织单元内部的变量	不允许	读/写
VAR_INPUT	输入变量,由外部提供,在程序组织单元内部不能修改	读/写	读
VAR_OUTPUT	输出变量,由程序组织单元提供给外部实体使用	写	读/写
VAR_IN_OUT	输入-输出变量,由外部实体提供,能在程序组织单元内部修改	读/写	读/写
VAR_EXTERNAL	外部变量,能在程序组织单元内部修改,由全局变量组态提供	读/写	读/写
VAR_GLOBAL	全局变量,能在对应的配置、资源内使用	读/写	读/写
VAR_ACCESS	存取变量,用于与外部设备的不同程序间变量的传递	读/写	读/写
VAR_TEMP	暂存变量,在程序或功能块中暂时存储的变量	读/写	读/写
VAR_CONFIG	配置变量,实例规定的初始化和地址分配	不允许	读

表 2-19 变量的附加属性

变量附加属性关键字	变量附加属性
RETAIN	表示变量附加保持属性,即电源掉电时能够保持该变量的值
NON_RETAIN	表示变量附加不保持属性,即电源掉电时不具有掉电保持功能。
CONSTANT	表示该变量是一个常数。因此,程序执行时,该变量的值保持不变(不能修改)
AT	变量存储的地址
R_EDGE	对输入变量设置上升沿检测
F_EDGE	对输入变量设置下降沿检测
READ_WRITE	对存取变量设置读写属性
READ_ONLY	对存取变量设置只读属性

变量必须在不同程序组织单元的每个定义开始时声明。变量的不同属性取决于变量的用途,如被作为输入变量、输出变量或作为内部变量等。

并非所有变量类型具有附加属性。附加属性遵循下列应用准则:

1) 在 VAR、VAR_INPUT、VAR_OUTPUT 和 VAR_GLOBAL 段内声明的变量允许使用附加属性 RETAIN 和 NON_RETAIN。

2) 当功能块或程序实例中使用附加属性 RETAIN 和 NON_RETAIN 时,所有实例的成员都被处理为具有 RETAIN 和 NON_RETAIN 的属性。除非成员本身是功能块,或在功能块或程序类型的声明中明确声明被作为 RETAIN 和 NON_RETAIN 使用。

3) 在 VAR_CONFIG 实例中允许使用附加属性 RETAIN 和 NON_RETAIN。这时,所有该结构变量的成员,包括嵌套结构的成员都具有相应的附加属性。

4) 没有声明附加属性的变量初始化时,应根据热启动特性确定初始值。

5) CONSTANT 附加属性声明该变量是不允许改变其值的特殊变量(常数)。因此,同时对某个变量附加 CONSTANT 和 RETAIN 属性是没有必要的。这时,只需要设置 CONSTANT 的附加属性。因掉电后的热启动时,该变量仍可保持该常数值。

6) 对 VAR 和 VAR_GLOBAL 变量,可附加变量的 CONSTANT 常数属性。

7) 上升沿和下降沿检测属性只对输入变量有效。读/写和只读属性只对存取变量有效。

8) 其他附加属性的关键字是紧跟在变量关键字后的,如 VAR CONSTANT; VAR_OUTPUT RETAIN 等。但上升沿、下降沿检测属性及读写、只读属性的关键字是在变量数据类型后,如 VAR RI:REAL R_EDGE; VAR RW:REAL READ_WRITE。

9) 使用关键字 AT 可完成符号变量的物理或逻辑地址的分配,如上述。

10) 可以用星号“*”表示没有特定地址的变量。

11) 输入输出变量既作为 POU 的输入,也作为 POU 的输出参数的变量列表,它能够在 POU 内被修改。这类变量的典型应用是例 2-17 的控制功能模块模式设置。

【例 2-17】 控制功能模式设置。

```

TYPE
    MODE_LIST:( INIT,READY,RUNNING,STOPPED);
END_TYPE

```

```
VAR_IN_OUT
  AUTO:MODE_LIST;
END_VAR
```

例 2-17 中,定义了模式列表数据类型,并用输入输出变量定义 AUTO 是模式列表数据类型,考虑功能模块带输入输出变量 AUTO,假设功能模块参数 AUTO 连接到一个外部变量 MODE_LIST。因此,当功能模块执行时,即进行功能模块初始化 INT,它自动将 MODE_LIST 的值存放到 AUTO 变量。功能模块能够通过写来实现模式的改变,即用 READY 直接写到 AUTO 参数。

变量属性可用文字表示,也可用图形表示。图 2-7 是变量属性的图形描述。

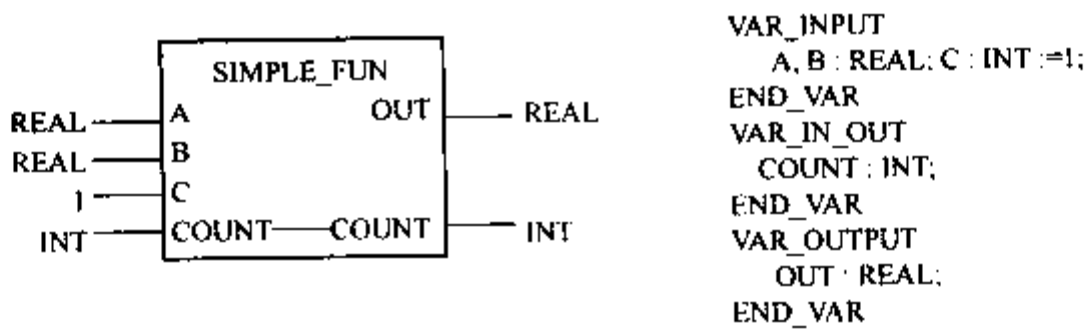


图 2-7 变量属性图形描述示例

图中,变量 A、B 和 C 是输入变量,OUT 是输出变量,COUNT 是输入输出变量。变量的数据类型也同时显示。图中还设置变量 C 的初始值为 1。

2. 变量的初始化

变量在系统启动时进行初始化(Initialization)。初始化后变量的值根据下列准则确定。

- 1) 当系统停止初始化时变量具有的被保持的值。例如,再启动时的掉电前的保持值。
- 2) 用户规定的初始值。
- 3) 根据变量的有关数据类型提供的约定初始值。

电源掉电后的再启动,称为系统的热启动(Warm Restart)。这时,变量的值应根据是否由附加属性 RETAIN 来确定。如果具有该属性,则变量恢复到掉电前的值。如果没有该属性,则称为系统的冷启动(Cold Restart)。这时,变量初始值由用户规定的初始值或该变量对应的数据类型的约定初始值(当没有用户规定初始值时)确定。

变量初始值取值有优先级。准则表明,RETAIN 提供最高优先级(即准则中的 1)),系统约定初始值提供最低优先级(即准则中的 3))。

须注意,用户不能规定从外部输入的变量的初始值。例如,VAR_EXTERNAL、VAR_INPUT 变量段声明的变量不能赋予初始值。表 2-20 是变量初始化的示例。

表 2-20 变量的初始化

特 性	示 例	说 明
直接表示变量的初始化	<pre>VAR AT %QX5.1;BOOL:= 1; AT %MW8;INT:= 66; END_VAR</pre>	布尔类型,输出单元 5 的第 1 位,赋予初值为 1 初始化存储器字,赋予整数值 66

(续)

特 性	示 例	说 明
直接表示保持变量的初始化	<pre>VAR RETAIN AT %QW6;WORD:= 16#FF00; END_VAR</pre>	冷启动时,在输出单元6的16位串中的高8位初始置1,低8位初始置0
符号变量地址的初始值配置	<pre>VAR VLV_POS AT %QW28;INT:= 100; END_VAR</pre>	配置输出单元字28到整数变量VLV_POS(符号变量),并置初始值为100
数组变量地址和初始值配置	<pre>VAR OUT_ARY AT %QW6;ARRAY[0..9] OF INT:= [10(1)]; END_VAR</pre>	把10个整数的数组配置给以%QW6开始的相邻输出单元,各初始值置1 数组变量名OUT_ARY
符号变量初始化	<pre>VAR MYBIT;BOOL:= 1; OKAY;STRING[10]:= 'OK'; END_VAR</pre>	变量MYBIT是布尔值,设置初值为1 字符串OKEY的最大长度为10,初始化后长度为2,置'OK'(79和75)
数组初始化	<pre>VAR BITS;ARRAY[0..7] OF BOOL:= [1,1,0,0,0,1,0,0]; TBT;ARRAY[1..2,1..3] OF INT:= [1,2,3(4),6]; END_VAR</pre>	BITS[0..7]的布尔初值分别为[1,1,0,0,0,1,0,0] TBT[1,1]:=1,TBT[1,2]:=2,TBT[1,3]:=4, TBT[2,1]:=4,TBT[2,2]:=4,TBT[2,3]:=6
数组保持的声明和初始化	<pre>VAR RETAIN RTBT;ARRAY[1..2,1..3] OF INT:= [1,2,3(3)]; END_VAR</pre>	RTBT[1,1]:=1,RTBT[1,2]:=2,RTBT[1,3]:=3, RTBT[2,1]:=3,RTBT[2,2]:=3,RTBT[2,3]:=0
结构化变量的初始化	<pre>VAR MODULE_8_CONFIG; AI_I6_CONFIG:= (SIGNAL_TYPE:= DIFFERENTIAL, CHANNEL:= [4((RANGE:= UNIPOLAR_I_ 5V)), (RANGE:= BIPOLAR_10_V, MIN_SCALE:= 0,MAX_SCALE:= 500)]); END_VAR</pre>	衍生数据类型的初始化(见表2-13) 本例声明变量MODULE_8_CONFIG的初始值分别是:SIGNAL_TYPE的初始值是DIFFERENTIAL; CHANNEL的初始值包括4个RANGE为UNIPOLAR_1_5V等
常数的初始化	<pre>VAR CONSTANT PI;REAL:= 3.141593; END_VAR</pre>	声明PI(π),并赋值
功能块实例的初始化	<pre>VAR TIC_112; PID:= (PropBand:= 2.5, Integral:= T#5s,Derivative:= T#1.5s); END_VAR</pre>	对功能块实例TIC_112的PID功能块的比例带、积分时间和微分时间设置初始值

3. 变量声明

变量声明用于建立变量与它的数据类型之间的关系,在变量声明中可对一些变量设置用户的初始值,变量声明和初始化在变量声明段同时完成。

(1) 变量声明的格式

变量声明段以表2-18的变量类型关键字开始,它表示该变量段内声明的变量类型。例如,VAR_INPUT表示该变量声明段用于声明的变量是输入变量。中间部分是变量声明段本体,变量声明段以END_VAR结束。每个变量声明段对应一种变量类型,但可包含多个变量。

变量声明段本体的格式如下:

变量名;变量数据类型(及初始值)和分号“;”

【例 2-18】 变量声明段本体示例。

```
START;BOOL;      (* 变量名 START,其数据类型是布尔量 *)  
START;BOOL:=1;    (* 变量名 START,数据类型是布尔量,初始值为1 *)
```

需设置用户变量初始值时,在上述格式中圆括号内的初始值用“:=”和初始值表示。如果使用系统默认初始值,则上述格式中圆括号内的内容可省略。

(2) 变量声明的规则

1) 具有相同数据类型的变量可以集中声明。例如,A、B、C 都是整数类型的变量,则可表示为 VAR A,B,C:INT;END_VAR,相同数据类型的变量间用逗号分隔。如果 A 和 B 是整数类型变量,C 和 D 是实数类型变量,则表示为 VAR A,B:INT;C,D:REAL;END_VAR。表 2-21 是变量声明的示例。

表 2-21 变量声明的示例

特 性	示 例	声 明
直接表示变量的声明 ^①	VAR AT %IW6.6 :WORD; AT %MW8 :INT; END_VAR	16 位串; 16 位整数,初始值为 0
直接表示保持变量的声明 ^①	VAR RETAIN AT %QW4:WORD; END_VAR	冷启动时,初始值是 16 位串,其值为 16#0000,是系统默认的初始值
符号变量地址声明 ^①	VAR_GLOBAL LIM_SW_S5 AT %IX27;BOOL; CONV_START AT %QX25;BOOL; TEMPERATURE AT %IW28;INT; C2 AT %Q*;BYTE; END_VAR	分配输入位 27 到布尔变量 LIM_SW_S5 分配输出位 25 到布尔变量 CONV_START 分配输入字 28 到整数变量 TEMPERATURE 分配尚无特定地址的输出字节到长度为 8 位的位串变量 C2
数组地址分配 ^①	VAR INARY AT %IW6:ARRAY [0..9] OF INT; END_VAR	声明由 10 个整数组组成数组被分配到以%IW6 开始的相邻单元组成的变量 INARY
数组声明	VAR THREE; ARRAY[1..5,1..10,1..8] OF INT; END_VAR	分配 5×10×8(若 400)个存储器单元给三维整数数组 THREE
符号变量自动存储器地址分配	VAR CONDITION_RED:BOOL; IBOUNCE:WORD; MYDUB:DWORD; AWORD,BWORD,CWORD:INT; MYSTR:STRING[10]; END_VAR	存储器位分配给 CONDITION_RED 存储器字分配给 16 位串变量 IBOUNCE 双存储字分配给 32 位串变量 MYDUB 3 个分开的存储器字分给整数变量 AWORD、BWORD 和 CWORD 存储器包含最大长度为 10 字符的串,初始化后,串长为 0,包含空串
保持数组的声明	VAR RETAIN RTBT; ARRAY[1..2,1..3] OF INT; END_VAR	2×3(共 6)个保持存储器单元分给二维整数数组变量 RTBT,冷启动时所有元素初始值为 0
结构变量的声明	VAR MODULE_8_CONFIG; ANALOG_16_INPUT_CONFIGURATION; END_VAR	衍生数据类型的变量声明(见表 2-15)

(续)

特 性	示 例	声 明
枚举变量的声明	VAR Color: (Red, Yellow, Green); END_VAR	声明枚举变量 Color 由 Red、Yellow、Green 三原色组成
子范围变量的声明	VAR Z: SINT(5..95); END_VAR	子范围变量的声明, Z 由短整数 5~95 组成

① 只能用于 PROGRAM 和 VAR_GLOBAL 声明中。

2) 内部变量的声明。POU 内部使用的变量(本地变量)用 VAR 关键字声明。

【例 2-19】 内部变量的声明。

```
VAR
    AVE_SPEED: REAL;    (* 用 AVE_SPEED 变量表示实数数据类型, 平均转速 *)
    Inhibit: BOOL; = 1;  (* 用 Inhibit 变量表示布尔数据类型, 禁止标志, 初始值为 1 *)
    AT %I10: WORD; = 2#0000_1010; (* 初始输入存储位 10, 其值为二进制的 1010 *)
END_VAR
```

3) 输入变量的声明。输入到一个 POU 作为其输入的变量, 用 VAR_INPUT 关键字声明。程序、功能模块和函数需要输入变量。

【例 2-20】 输入变量的声明。

```
VAR_INPUT
    SPEED: REAL; = 0.0; (* SPEED 变量表示实数数据类型, 实际转速, 初始值为 0.0 *)
    START: BOOL;        (* 用 START 变量表示布尔数据类型, 启动信号 *)
END_VAR
```

4) 输出变量的声明。作为输出参数, 提供数值写入到外部变量的变量。用 VAR_OUTPUT 关键字声明。输出变量在程序、功能模块中需要。但函数中不需要, 因为函数只有返回值。

【例 2-21】 输出变量的声明。

```
VAR_OUTPUT
    FV_I02: REAL;        (* FV_I02 变量表示实数数据类型, 送执行阀 FV_I02 的信号 *)
    M_I03_STATUS: BOOL; (* 用 M_I03_STATU 变量表示布尔数据类型, 电动机 M_I03 运行状态 *)
END_VAR
```

5) 输入输出变量的声明。既可作为 POU 输入, 也可作为 POU 输出参数的变量, 它能够在 POU 内被修改。用 VAR_IN_OUT 关键字声明。输入输出变量接受来自外部的变量值, 能够修改在 POU 内的值。从外部看, 这些变量的值用同样的方法可存取作为其输出参数。它是外部到 POU 的变量, 通常用于控制功能模块的模式。

例如, 带输入输出变量 AUTO 的功能模块, 其 AUTO 被连接到一个外部变量 MAIN_MODE。通过在功能模块外部的软件将 INIT 写入 MAIN_MODE 来初始功能模块模式的值。当功能模块执行时, 它存取到 AUTO 变量, 使功能模块初始化。功能模块的模式可通过 READY 直接写到 AUTO 参数, 并存储在 AUTO 内, 也可从源变量 MAIN_MODE 存取。

【例 2-22】 输入输出变量的声明。

```
TYPE
    MODE_LIST:( INT,READY,RUNNING,STOPPED);
END_TYPE
VAR_IN_OUT
    AUTO:MODE_LIST;(* 用 AUTO 输入输出变量,它的数据类型是枚举数据类型 *)
END_VAR
```

6) 全局变量的声明。全局变量可在配置、资源或程序层进行声明。它们能够被在配置、资源或程序内的任何 POU 存取。在 POU 内的变量被声明为外部,它就可以存取在 POU 外部声明的全局变量的值。在功能模块声明的外部变量能够对一个包含该功能模块的配置、资源或程序内定义的全局变量进行访问。

通常,全局变量和外部变量被用于提供存取在程序和功能模块中的关键数值。

【例 2-23】 全局变量的声明。

```
VAR_GLOBAL
    LINESPEED:LREAL;
    JOB_NUMBER:INT;
END_VAR
```

例 2-23 定义了一个双精度浮点数(LREAL)用于表示线速及一个 16 位整数用于计数。

7) 暂存变量的声明。暂存变量在 POU 内声明,它可放置在暂存的存储区,如堆栈。当 POU 宣布终止时,这些变量的值被清除。

【例 2-24】 暂存变量的声明。

```
VAR_TEMP
    RESULT:REAL;                                (* 定义 RESULT 作为暂存变量 *)
END_VAR
RESULT:= AF18 + XV23 * XV67 + 54.23;  (* 用于中间结果的计算 *)
OUT1:= SQRT(RESULT)
```

8) 存取变量的声明。存取变量声明用于提供远程设备和其他基于 IEC 61131-3 的程序能够存取的特定变量。存取变量提供 IEC 标准调用什么存取路径到被命名的变量。存取变量仅涉及下列类型变量:一个程序的输入或输出变量、全局变量和直接表示变量。

存取变量可能涉及多元素数据类型的变量,例如,数组或多元素数据类型变量的被选择元素。存取变量的只读属性,用于说明远程设备仅能读有关变量。读写属性说明该变量既可读也可写。存取变量声明的格式如下:

存取路径:存取变量名:数据类型 变量的读写属性;

【例 2-25】 存取变量的声明。

```
VAR_ACCESS
    LINE_START_UP:LINE1.START_UP:BOOL READ_WRITE;
    LINE_SPEED:SPEED:REAL READ_WRITE;
    GOOD_CABLE:LENGTH:INT READ_ONLY;
```

```

    ABLE;STATION_2. P1. y1;BYTE  READ_ONLY;
END_VAR

```

例 2-25 对 4 个存取变量进行声明。一个程序名为 LINE1 的输入参数 LINE1, START_UP;两个全局变量 SPEED 和 LENGTH;一个资源名为 STATION_2,程序名为 P1 的输入参数 y1。可用存取路径 LINE_START_UP、LINE_SPEED、GOOD_CABLE 和 ABLE 来读写 LINE1, START_UP 和 SPEED 及只读 LENGTH 和 STATION_2. P1. y1。

9) 外部变量的声明。外部变量在 POU 内部被声明,用于提供在配置、资源和程序层定义的全局变量的存取。

【例 2-26】 外部变量的声明。

```

VAR_EXTERNAL
    LINESPEED:LREAL;
    JOB_NUMBER:INT;
END_VAR

```

10) 配置变量的声明。配置变量在配置中用关键字 VAR_CONFIG 声明。用于声明配置内实例的特性,如初始化和有关地址分配。

【例 2-27】 配置变量的声明。

```

VAR_CONFIG
    STATION_1. P1. COUNT    :INT:=1;
    STATION_1. P1. TIME1    :TON:=(PT:=T#2.5s);
    STATION_2. P1. TIME1    :TON:=(PT:=T#4.5s);
    STATION_2. P4. FB1. C2   AT %QB25 :BYTE;
END_VAR

```

例 2-27 中,用 STATION_1. P1. COUNT 配置变量来对资源 STATION_1,程序 P1 的输入参数 COUNT 设置为整数数据类型,其初始值被设置为 1 等。

2.6 程序组织单元

程序组织单元(Program Organization Unit,POU)由程序组织单元的声明部分和程序组织单元的本体两部分组成,它是用户程序的最小软件单位。对应于传统可编程控制器的程序块、组织块、顺序块和功能块等。程序组织单元按功能分为函数、功能模块和程序。程序组织单元的标准部分(如标准函数、标准功能块等)由可编程控制器制造商提供。用户根据程序组织单元的定义设计用户的程序组织单元,对程序组织单元调用和执行。程序组织单元不是递归的,即程序组织单元的调用不会造成同类型另一个程序组织单元的调用。

2.6.1 函数

为了可编程控制器编程语言的应用,一个函数(Function)被定义为一个程序组织单元。函数是一种可以赋予参数,但没有静态变量(没有记忆)的程序组织单元。当用相同的输入参数调用某一函数时,该函数总能够生成相同的结果作为其函数值(输出)。函数曾被译为功

能,但表达不够确切。函数是可重复使用的软件元素。函数可用于标准规定的各种编程语言中,函数分为标准函数和用户自定义函数两类。

所有函数的重要特征是它们不能在内部变量存储数值。这与功能模块形成鲜明对照,功能模块可在内部变量或输出变量中存储数值。

标准和用户自定义的函数可在其他函数、功能模块和程序中使用。例如,在文本类编程语言中,函数的调用结果可作为表达式中的一个操作数。

1. 函数的表示

函数可用文本类编程语言或图形类编程语言表示。

结构化文本编程语言表示的函数如下：

```
FUNCTION  Z:REAL ;
VAR
    (* 函数的变量声明 *)
    X,Y,Z,RES1,RES2:REAL;
    EN1,V:BOOL;
END_VAR
(* 函数本体 *)
RES1:= DIV( IN1:=COS(X), IN2:= SIN(Y), ENO => EN1) ;
RES2:= MUL( SIN(X),COS(Y));
Z:= ADD ( EN:= EN1,IN1:= RES1,IN2:= RES2,ENO => V) ;
END_FUNCTION
```

功能块图编程语言表示的函数如图 2-8 所示。

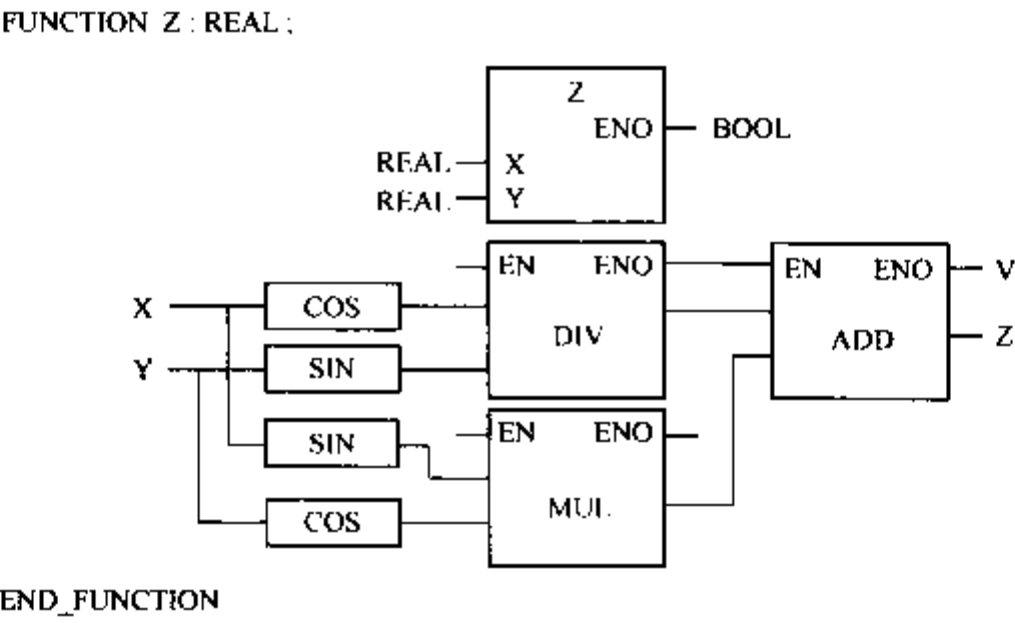


图 2-8 函数用法示例

图 2-8 所示为对相同功能的两种不同的描述方式。在两个不同描述方式之间不需要支持任何自动的转换。图中,变量 EN1 未画出。

函数由关键字 FUNCTION、函数名、冒号、返回值数据类型、变量声明和函数本体组成,函数用 END_FUNCTION 结束。可以用文字或图形类编程语言表示函数本体程序。

函数有多个输入变量,有一个函数值作为该函数的返回值。作为任意数据类型,函数返回值可以是多值的。例如,可以是一个任意数据或结构。图 2-8 给出了 SIN、COS、MUL、DIV 和 ADD 函数的使用,也给出了 EN 和 ENO 属性的使用。



函数应不含内部状态信息,即具有相同自变量(输入变量 VAR_INPUT 和输入输出变量 VAR_IN_OUT)的一个函数调用,应得到相同结果(输出变量 VAR_OUTPUT,输入输出变量 VAR_IN_OUT 和函数返回值)。

(1) 文本类编程语言中函数的表示

在文本类编程语言中,函数的表示遵循下列准则。

1) 输入自变量的配置应遵循表 2-22 中给出的规则。

表 2-22 布尔信号的图形符号

序 号	特 性	表 示
1	反相输入	
2	反相输出	

注:如果对函数来说,若支持上述两个特性的每一个,则它也支持功能块,反之亦然;对输入输出变量,这些限制的用法是严禁的。

2) 函数的输出变量配置既可以是空的,也可以是变量。

3) 配置到 VAR_IN_OUT 的自变量应是变量。

4) 配置到 VAR_INPUT 的自变量可以是空的、常数、变量或函数调用。在函数调用时,函数是作为实际的自变量被调用的。

(2) 图形类编程语言中函数的表示

在图形类编程语言中,函数的描述遵循下列规则:

1) 函数的图形形式应是矩形或正方形。其大小和比例可根据输入和其他需要显示信息的数量而变化。函数的名称或符号应按如下规定,位于函数图形符号的内部:

- 函数的处理方向应从左到右(输入变量在左,输出变量在右)。
- 用函数图形符号表示时,输入输出变量名称应显示在图形符号的左面和右面。
- 在标准函数中,如果输入变量没有给出名称,则默认的名称是 IN1、IN2 等。例如,ADD 函数的输入是 IN1,IN2 等。对单个无名称的输入变量,使用约定的变量名称 IN。约定的变量名称可以,但不必一定要出现在函数图形符号内部的左面。
- 当使用附加 EN 和/或 ENO 属性时,附加输入的 EN 和/或输出 ENO 应显示在函数图形符号的左和右的最上面。
- 函数的返回值显示在函数图形符号的左或右的最上面,通常位于图形符号的最右上面。但当存在 ENO 时,可在其下面的位置显示。须注意,返回值没有变量名称。
- 布尔信号的反相可采用位于连接图形符号输入或输出线外部相切的一个空圆显示,如表 2-22 所示。

2) 所有输入和输出(包括函数返回值)应采用单线表示在函数图形符号相应侧外部。单线连接不表示被连接元素是单元素变量,它也可表示被连接数据元素是多元素变量。

3) 函数输出和返回值可连接到一个变量,用于作为其他函数或功能块的输入,也可以左面不连接。

4) IEC 标准的第 2 版允许用 VAR_IN_OUT 变量。VAR_IN_OUT 变量能够图形地连接到右面,或用文字赋值语句“:=”操作符,它可用于修改函数本体。


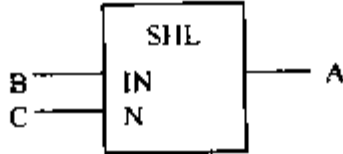
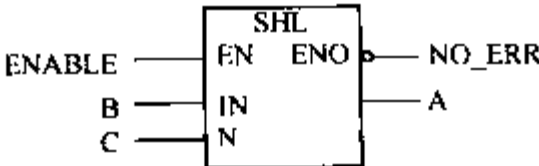
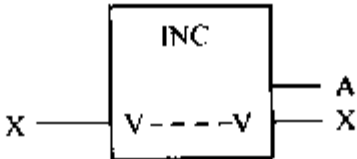
【例 2-28】 VAR_IN_OUT 变量用于修改函数 INCREMENT。

```
FUNCTION INCREMENT;BOOL
  VAR_IN_OUT
    COUNT :INT;
  END_VAR
  VAR_INPUT
    MAX:INT;
  END_VAR
  COUNT:= COUNT +1;
  INCREMENT:= (COUNT >= MAX);
END_FUNCTION
```

(3) 示例

表 2-23 是函数的文字和图形表示的示例。

表 2-23 函数的文字和图形表示

示例的图形表示	示例的文字表示	说 明
<p>ADD 函数的图形用法</p> 	<p>ADD 函数的 ST 语言文字用法</p> <p>A:= ADD(B,C,D);</p>	无形参变量名
<p>SHL 函数的图形用法</p> 	<p>SHL 函数的 ST 语言文字用法</p> <p>A:= SHL(IN:=B,N:=C);</p>	有形参变量名
<p>SHL 函数的图形用法</p> 	<p>SHL 函数的 ST 语言文字用法</p> <p>A:= SHL(EN:=ENABLE,IN:=B,N:=C,NOT ENO =>NO_ERR);</p>	有形参变量名 使用 EN 输入和 ENO 的反相输出
<p>用户自定义的 INC 函数的图形用法</p> 	<p>INC 函数的 ST 语言文字用法</p> <p>A:= INC(V:=X);</p>	有形参变量名 对 VAR_IN_OUT 用形参名

示例显示了标准函数(ADD 和 SHL)、带 EN 和 ENO 属性的标准函数和用户自定义的带形参名的 INC 函数。

2. 函数声明

函数声明(Declaration)指可用图形和文字两种形式表示。

(1) 函数声明的图形形式

图形形式表示的函数是一个矩形框,框内表示函数名和有关的输入变量、输入输出变量等。由于函数只有函数返回值,因此框内未列出输出变量名。与各输入变量对应,在框外绘制

连接线,并标注各变量对应的数据类型名称。

图 2 - 9 是函数 COMPENSATION_PT 图形形式的声明表示。

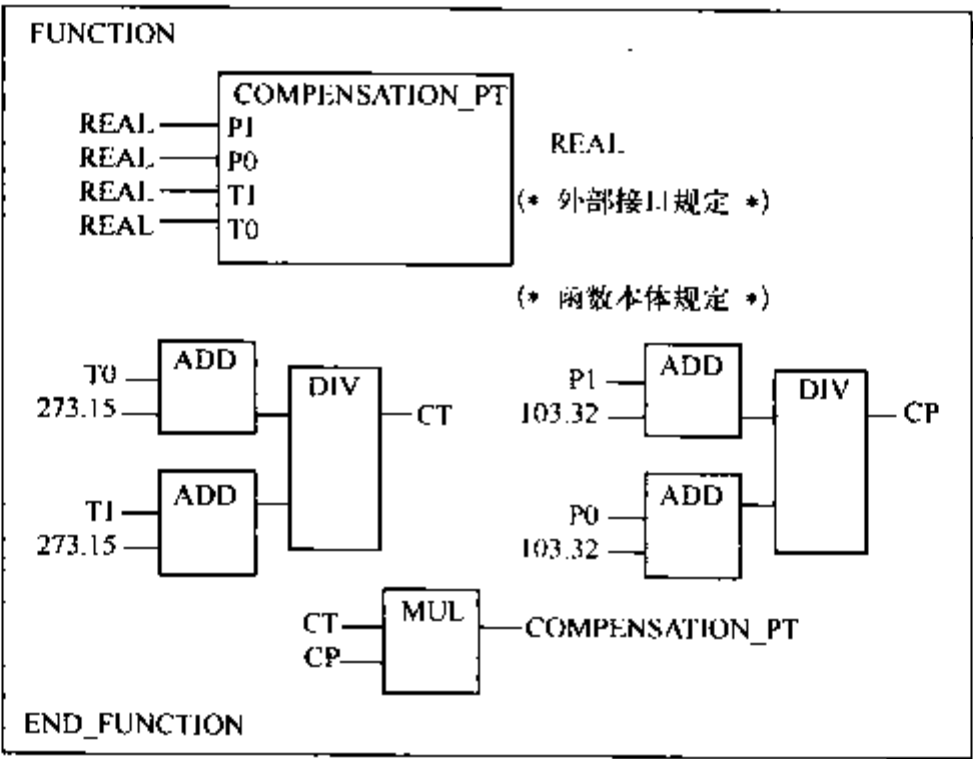


图 2 - 9 COMPENSATION_PT 函数的图形表示

函数声明的文字形式和图形形式都必须用 FUNCTION…END_FUNCTION 结构。文本形式中函数变量声明和本体用文本类编程语言编写,图形形式中函数变量声明和本体用图形类编程语言编写。须注意,在图形形式中无法表示用户设置的变量初始值。

(2) 函数声明的文字形式

文字形式表示的函数是由 FUNCTION 关键字开始,随后是函数名、冒号和返回值的数据类型、变量声明,中间是函数体,最后是 END_FUNCTION 关键字结束的函数段组成。

例 2 - 29 是函数 COMPENSATION_PT 的文字形式的声明。

例 2 - 29 中,函数 COMPENSATION_PT 是衍生函数,其函数名是 COMPENSATION_PT。它用于气体流量的温度压力补偿。函数变量声明段表示该函数有 4 个输入变量,变量名为 P1、P0、T1、T0,为实数数据类型。增加两个内部变量 CT 和 CP,它们是温度补偿系数和压力补偿系数。返回值 COMPENSATION_PT 为实数数据类型。

COMPENSATION_PT 函数本体包括加法运算、除法运算和乘法运算。CT 用于计算实际温度与设计温度不同时的温度补偿系数,CP 用于计算实际压力和设计压力不同时的压力补偿系数。运算结果是两个补偿系数之积,它作为该函数的返回值。

【例 2 - 29】 COMPENSATION_PT 函数。

```
FUNCTION COMPENSATION_PT :REAL
  VAR_INPUT          (* 输入变量声明段 *)
    P1,P0 :REAL ;    (* 实际压力和设计压力,单位 kPa *)
    T1,T0 :REAL ;    (* 实际温度和设计温度,单位℃ *)
  END_VAR
  VAR                (* 内部变量声明段 *)
```

```

CP,CT:REAL ;
END_VAR
CT:= DIV( ADD( T0 +273.15),ADD(T1 +273.15)) ;
CP:= DIV( ADD( P1 +103.32),ADD(P0 +103.32)) ;
COMPENATION_PT:= MUL(CT,CP) ;
END_FUNCTION

```

【例 2-30】 标准函数 SEL。

图 2-10 所示为标准函数 SEL 的图形形式和用 ST 语言编写的文字形式。

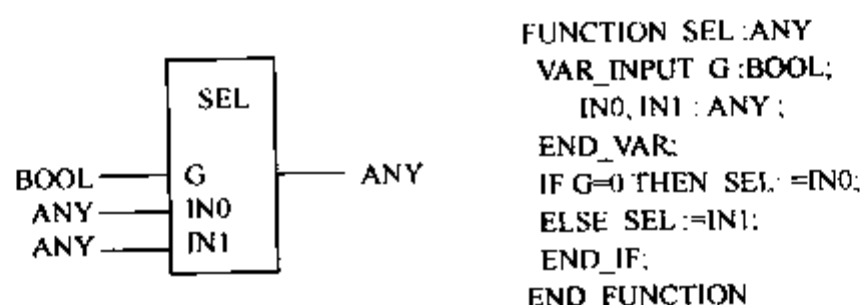


图 2-10 标准函数 SEL 的图形形式和用 ST 语言编写的文字形式

标准函数也由函数变量声明和函数本体两部分组成。

(3) 函数中的变量声明

函数段内分为函数变量声明和函数本体两部分。函数变量声明段按函数内所用的变量分类,以 VAR 或 VAR_INPUT 等开始的变量声明段,用于声明内部变量、输入变量等的变量名及对应的数据类型,如果有初始值,则声明初始值,并用 END_VAR 结束。

函数中的变量声明用于对函数中各变量进行声明。包括变量的类型、变量的数据类型和初始值等。函数本体用有关表达式或图形表达方式表示实现该函数所需的运算。

在函数变量声明段,可声明的变量类型有 VAR、VAR_INPUT 等,用于特定函数的内部、输入变量名称和数据类型。变量还包括输入输出变量、输出变量等类型。

3. 标准函数

IEC 61131-3 标准定义了 8 类标准函数,它的作用类似于数学函数。例如,SIN 函数用于计算输入变量的正弦值,SQRT 函数用于计算输入变量的开方等。

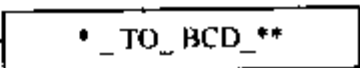
(1) 类型转换类函数

类型转换类函数(Type Conversion Function)用于数据类型的转换。由于 IEC 61131-3 语言要求严格的数据类型检查。因此,必须确保被使用数据类型的一致性。例如,为进行实数运算,整数数据类型需要转换为实数,可用函数 INT_TO_REAL 函数。表 2-24 所示为类型转换类函数的 4 种不同形式。

表 2-24 类型转换类函数的图形形式和文字形式

序 号	图 形 形 式	ST 语言表示的文字形式
1	<p>*: 输入数据类型, 如 INT **: 输出数据类型, 如 REA *_TO_*: 函数名, 如 INT_TO_REAL</p>	A:=INT_TO_REAL(B);
2		A:=TRUNC(B);

(续)

序 号	图 形 形 式	ST 语言表示的文字形式
3	* —  — **	A := WORD_BCD_TO_INT(B);
4	* —  — **	A := INT_TO_BCD_WORD(B);

使用数据类型转换函数时应注意下列事项:

1) 数据类型转换时可能引起误差。例如,从实数转换为整数时,根据四舍五入的原则转换。

【例 2-31】 实数数据类型转换函数引入误差。

$\text{REAL_TO_INT}(1.6) = 2; \text{REAL_TO_INT}(-1.5) = -2; \text{REAL_TO_INT}(1.4) = 1; \text{REAL_TO_INT}(-2.5) = -2;$

2) TRUNC 函数用于将一个 REAL 或 LREAL 表示的浮点数的整数部分转换为一个整数类型的整数,而不考虑其小数部分(截去小数部分)。

【例 2-32】 TRUNC 数据类型转换函数的取整。

$\text{TRUNC}(567.123) = 567; \text{TRUNC}(567.89) = 567;$

3) 当相应的位串变量包含 BCD 码数据时, *_BCD_TO_** 函数和 **_TO_BCD_* 函数执行 BYTE、WORD、DWORD、LWORD 类型(用 * 表示)的变量和 USINT、UINT、UDINT、ULINT 类型(用 ** 表示)变量之间的转换。

【例 2-33】 数据类型转换函数示例。

$\text{USINT_TO_BCD_BYTE}(25) = 2\#0010_0101; \text{WORD_BCD_TO_INT}(2\#0001_0110_1000) = 168;$

4) 类型转换函数的输入或输出是 STRING 或 WSTRING 时,字符串数据应符合相应数据的外部表示。

【例 2-34】 字符串数据的类型转换函数示例。

$\text{STR} := \text{DINT_TO_STRING}(\text{DINT}\#16\#\text{AAAA}, \text{FORMAT});$

例 2-34 中,STR 和 FORMAT 是字符串数据类型,运行结果 $\text{STR} = '43690'$ 。

5) 当运行程序时可能发生数据类型不能转换为合适数据值的情况。例如,当 REAL 转换到 SINT 时,由于数值太大不能存储在 SINT(只有 8 位),这时,PLC 可提供运行期出错检测和报告的工具。

(2) 数值类函数

数值类函数(Numerical Function)用于对数值变量进行数学运算。该函数图形表示是将数值函数名称填写在函数图形符号内,并连接有关输入和输出变量。

一个单一的数值变量函数的标准图形形式、函数名、输入输出变量类型和函数描述如表 2-25 所示。这些数值函数的输入和输出的数据类型是相同的。

单一数值变量的标准数值函数包括绝对值(ABS)、平方根(SQRT)、自然对数(LN)、常用对数(LOG)、指数(EXP)、正弦(SIN)、余弦(COS)、正切(TAN)、反正弦(ASIN)、反余弦(ACOS)和反正切(ATAN)等 11 种标准数值函数。

表 2-25 数值函数的图形形式和文字形式

图形格式			用法示例
<div><div><div>*</div><div></div><div>**</div></div><div></div><div><div>*</div><div></div><div>**</div></div></div> <div>* 输入输出数据类型 ** 函数名称</div>			A:=SIN(B);
序号	函数名	输入输出类型	描述
一般函数			
1	ABS	ANY_NUM	绝对值
2	SQRT	ANY_REAL	平方根
对数函数			
3	LN	ANY_REAL	自然对数
4	LOG	ANY_REAL	以 10 为底的对数
5	EXP	ANY_REAL	自然指数
三角函数			
6	SIN	ANY_REAL	以弧度输入的正弦函数
7	COS	ANY_REAL	以弧度输入的余弦函数
8	TAN	ANY_REAL	以弧度输入的正切函数
9	ASIN	ANY_REAL	反正弦主值
10	ACOS	ANY_REAL	反余弦主值
11	ATAN	ANY_REAL	反正切主值

使用数值类函数时应注意下列事项。

1) 除绝对值函数的输入输出数据类型是 ANY_NUM 外,其他数值函数的输入输出数据类型是一般数据类型的 ANY_REAL。

【例 2-35】 绝对值函数示例。

```
B:=ABS(SINT#-102)
```

例 2-35 的运算结果为 102。须注意,变量 B 数据类型应为 SINT,才能保证数据类型一致。

2) 正弦、余弦和正切函数的输入数据以弧度为单位,反正弦、反余弦和反正切函数计算获得主值。

【例 2-36】 三角函数示例。

```
A:=ASIN(0.50)*180.0/3.14159265;
```

例 2-36 是计算反正弦函数的值并转换为度。运算结果 A 为 30.0,表示是 30°。

3) LN 是自然对数,EXP 是自然指数,都以自然对数的基 e 为基础。

【例 2-37】 对数函数示例。

```
A:=LN(2.0)/LN(10.0);
```

为计算 lg2,可用例 2-37 的计算方法,运算结果 A 为 0.3010。

4) 数值函数只有一个输入变量和一个输出变量(返回值)。

5) 数值函数支持数据类型是过载的,见下述。

6) 数值函数的运算结果与输入必须有相同的数据类型。例如,例 2-35 中 B 的数据类型必须是 SINT,例 2-36 和例 2-37 中的 A 必须是实数数据类型。

(3) 算术类函数

算术类函数 (Arithmetic Function) 用于计算多个输入变量的函数,包括加 (ADD)、减 (SUB)、乘 (MUL)、除 (DIV)、模除 (MOD)、幂 (EXPT) 和赋值 (MOVE) 等。

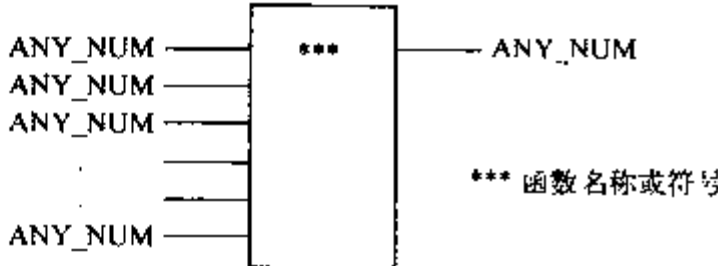
使用算术类函数时注意下列事项。

1) 输入变量和输出变量的数据类型都是 ANY_NUM,都只有一个输出变量(返回值)。

2) 赋值函数将输入变量的数据移动赋值到函数的返回值,即 OUT:=IN。

3) 两个或多个变量算术函数的标准图形形式、函数名、符号及描述如表 2-26 所示。当这些函数的计算结果超出函数输出的数据类型规定的数值范围,或企图除以零时,则出错。

表 2-26 标准算术函数

图 形 格 式				用 法 示 例
				A: = ADD(B,C,D); 或 A: = B + C + D;
序号 ^①	函 数 名		符 号	描 述
可扩展的算术函数				
1 ^②	加	ADD	+	OUT: = IN1 + IN2 + ... + INn
2	乘	MUL	*	OUT: = IN1 * IN2 * ... * INn
不可扩展的算术函数				
3 ^③	减	SUB	-	OUT: = IN1 - IN2
4 ^④	除	DIV	/	OUT: = IN1 / IN2
5 ^④	模除	MOD		OUT: = IN1 MOD IN2
6 ^⑤	幂	EXPT	**	指数: OUT: = IN1 ^{IN2}
7 ^⑥	赋值	MOVE	:=	OUT: = IN

① 当支持用函数名表示时,一致性语句中用后缀 n 表示。例如,1n 表示符号 ADD,2n 表示符号 MUL。当支持用符号名表示时,一致性语句中用后缀 s 表示。例如,1s 表示符号 +,2s 表示符号 *。

② 这些函数输入和输出的一般类型是 ANY_MAGNITUDE。

③ 整数相除的结果是相同数据类型的整数,向零截断取位。例如,7/3=2,(-7)/3=-2。

④ 对该函数,IN1 和 IN2 应是一般类型的 ANY_INT。该函数的计算结果等效于执行下列语句:IF (IN2=0) THEN OUT:=0;ELSE OUT:= IN1-(IN1/IN2)*IN2;END_IF。

⑤ 对该函数,IN1 数据类型是 ANY_REAL,IN2 数据类型是 ANY_NUM,输出与 IN1 的数据类型相同。

⑥ MOVE 函数有一个数据类型 ANY 的输入 IN 和一个数据类型 ANY 的输出 OUT。

4) 除了加和乘运算函数可有多个输入变量,赋值函数只有一个输入变量外,其他函数只有两个输入变量,并将第1个输入变量与第2个输入变量进行有关的算术运算。例如, $IN1 - IN2$, $IN1/IN2$, $IN1 \text{ MOD } IN2$ 等。因此,加和乘函数称为可扩展的算术函数。

【例2-38】 可扩展算术函数的示例。

$FAULT_COUNT := ADD(DEV1, DEV2, AB_34, AB_32, AX_32);$

它表示有5个输入变量进行加的运算。须注意,输入变量必须与输出返回值有相同的数据类型。

5) 输入输出变量的数据类型符合一般算术运算的要求,例如,整数除法的结果是向零截断的整数,实数除法的结果是实数等。

【例2-39】 算术函数的示例。

$A := (273.15 + 50.0) / (273.15 + S1);$

例2-39 用于温度补偿计算,%ID0中存放实际过程温度采样值S1(用摄氏度表示),它的数据类型是实数。其中,50.0是设计温度。

6) 整数相除时,结果与输入的被除数数据类型相同,商数截尾。

【例2-40】 整数除法函数的示例。

$C := INT\#128 / INT\#3;$

运算结果 $C = 42$ 。其余数2不采用四舍五取,而被截尾。变量C必须是整数。

7) 模除操作只对整数有效。

【例2-41】 模除函数的示例。

$D := INT\#128 \text{ MOD } INT\#3;$

例2-41 运算结果 $D = 2$,表示用模除可获得两个整数相除的余数,而例2-40获得商。

8) 幂函数的指数输入I2只能取任意整数ANY_INT,运算结果与输入I1数据类型相同。

【例2-42】 幂函数的示例。

$E := REAL\#2.0 ** INT\#3;$

运算结果 $E = 8.0$ 。这里,幂函数的指数应为整数。

(4) 位串类函数

位串类函数(Bit String Function)包括位串移位运算和位串的按位布尔函数。

1) 位串移位函数(Bit Shift Function)有两个输入变量,第1个变量是需要移位的位串,数据类型是ANY_BIT。这表示位串移位函数能用于过载数据类型,即位串可以是BOOL、BYTE、WORD、DWORD和LWORD。第2个变量是移动的位数,数据类型是ANY_INT。函数返回值仍是位串,数据类型是ANY_BIT。所有位串移位函数有相同格式:

$RESULT := \text{位串函数名}(IN := \text{输入的位串}, N := \text{移位的数量});$

表2-27所示为位串函数的图形形式和文字形式。

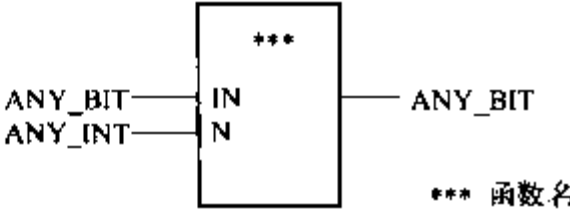
使用位串移位函数时注意下列事项:

- 移位函数分不循环左移(SHL)、不循环右移(SHR)、循环左移(ROL)和循环右移

(ROR)4 种。

- SHL 函数将输入位串 IN 左移 N 位,右面填 0,N 是大于 0 的整数。
- SHR 函数将输入位串 IN 右移 N 位,左面填 0,N 是大于 0 的整数。
- ROL 函数对输入位串 IN 进行左移循环移位 N 位,N 是大于 0 的整数。
- ROR 函数对输入位串 IN 进行右移循环移位 N 位,N 是大于 0 的整数。
- 输入位串和输出返回值的数据类型是 ANY_BIT,可以是 LWORD、DWORD、WORD、BYTE 或 BOOL 中的任何一种数据类型。
- 移动的位数 N 是整数,不能小于 0,否则出错。

表 2-27 位串函数的图形形式和文字形式

FBD 编程语言图形形式		ST 编程语言的文字形式示例
		A := SHL(IN := B, N := 5); A := SHR(IN := B, N := 4); A := ROL(IN := B, N := 3); A := ROR(IN := B, N := 2);
序 号	函 数 名	示例描述
1	SHL	A := SHL(IN := B, N := 5);输入位串 B 左移 5 位,右面填 0
2	SHR	A := SHR(IN := B, N := 4);输入位串 B 右移 4 位,左面填 0
3	ROL	A := ROL(IN := B, N := 3);输入位串 B 循环左移 3 位
4	ROR	A := ROR(IN := B, N := 2);输入位串 B 循环右移 2 位

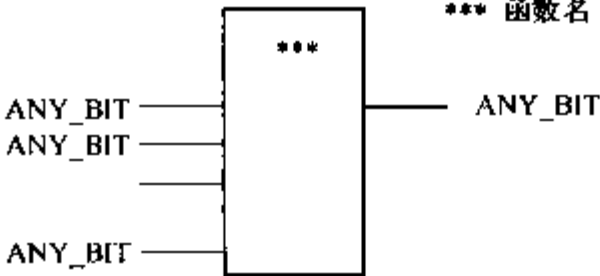
【例 2-43】 位串移位函数示例。

```
VAR
    T_8,G_8,H_8,K_8,L_8:BYTE;
END_VAR
T_8:= BYTE#2#0011_0101;( * T_8 = 16#35 * )
G_8:= SHL(T_8,4);( * G_8 = 2#0101_0000 =16#50 * )
H_8:= SHR(T_8,3);( * H_8 = 2#0000_0110 =16#06 * )
K_8:= ROL(T_8,3);( * K_8 = 2#1010_1001 =16#A9 * )
L_8:= ROR(T_8,3);( * L_8 = 2#1010_0110 =16#A6 * )
```

例 2-43 中将 T_8 的位串进行 4 种移位,并赋值给 G_8、H_8、K_8 和 L_8。

2) 位串的按位布尔函数(Bitwise Boolean Function)用于对位串按位进行逻辑运算,如与、或、非和或非运算。表 2-28 所示为位串的按位布尔函数的图形形式和文字形式。

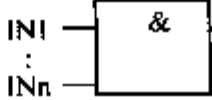
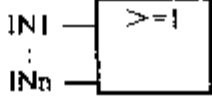
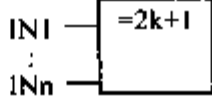
表 2-28 按位布尔函数的图形形式和文字形式

FBD 编程语言的图形形式	ST 编程语言的文字形式用法和示例
	OUT := AND(IN1, IN2, ..., INn);或 OUT := IN1 & IN2 .. & INn; OUT := OR(IN1, IN2, ..., INn);或 OUT := IN1 OR IN2 .. OR INn; OUT := NOT(IN);或 OUT := NOT IN; OUT := XOR(IN1, IN2, ..., INn);或 OUT := IN1 XOR IN2 .. XOR INn;

使用位串的按位布尔函数时应注意下列事项。

- 按位布尔函数可采用函数名或符号两种文字形式。表 2-29 所示为按位布尔函数的函数名和符号的两种文字形式。

表 2-29 按位布尔函数的两种文字形式

符号	函数名	图形表示示例	符号表示的示例	函数名表示的示例
&	AND		$OUT := IN1 \& IN2 \& \dots \& INn;$	$OUT := AND(IN1, IN2, \dots, INn);$ $OUT := IN1 \text{ AND } IN2 \dots \text{ AND } INn;$
≥ 1	OR		不能用文字符号表示形式	$OUT := OR(IN1, IN2, \dots, INn);$ $OUT := IN1 \text{ OR } IN2 \dots \text{ OR } INn;$
$= 2k + 1$	XOR		不能用文字符号表示形式	$OUT := XOR(IN1, IN2, \dots, INn);$ $OUT := IN1 \text{ XOR } IN2 \dots \text{ XOR } INn;$
	NOT	见表 2-22		$OUT := NOT IN;$

- OR 和 XOR 函数不能用文字符号形式表示,其他函数都可用两种不同的文字形式表示。
- ≥ 1 和 $= 2k + 1$ 的操作符号仅用于梯形图和功能块图编程语言中表示布尔逻辑运算。在文本类编程语言中不能使用。
- 非运算函数可采用位于连接块的输入或输出线外部相切的一个空圆表示,如表 2-22 所示。
- 按从上到下的顺序, $IN1, IN2, \dots, INn$ 符号表示输入, OUT 表示函数输出的返回值。
- 除了 NOT 非运算函数外,其他按位布尔函数都是可扩展的函数,即可有两个或两个以上的输入和支持过载数据类型 ANY_BIT。

【例 2-44】 按位布尔函数示例。

$TRIP_1 := OR(OVERTEMP1, OVERTEMP2, OVERTEMP3);$

例 2-44 用于反应器温度超限报警,只要 3 个温度中有一个超限,则报警 $TRIP_1$ 为 1。

【例 2-45】 多台电动机的起停控制。

位串的按位布尔函数进行按位逻辑运算,可对多台电动机进行起停控制。

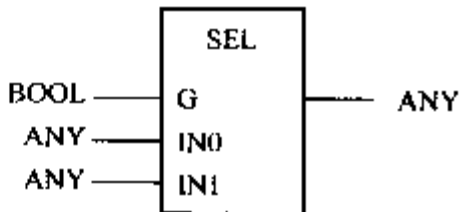
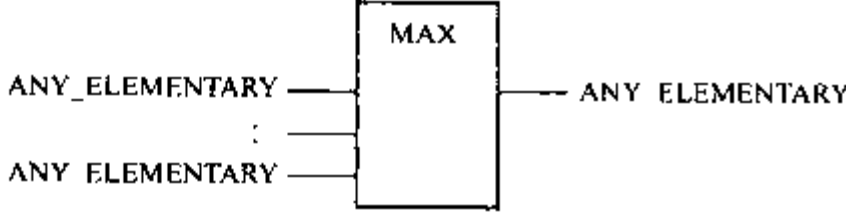
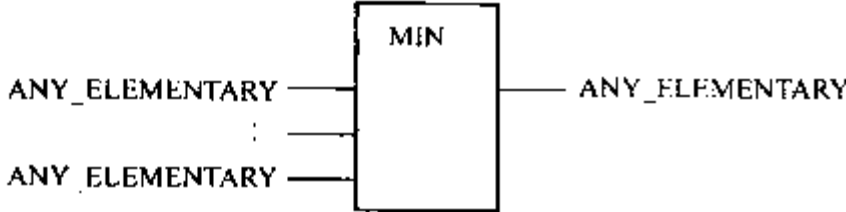
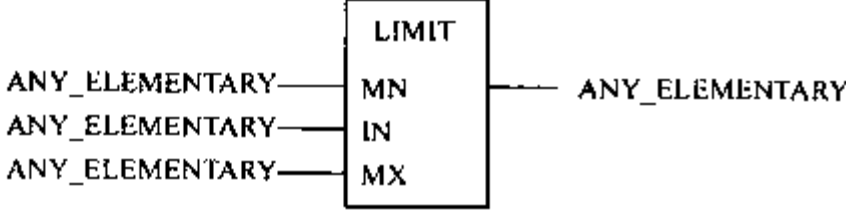
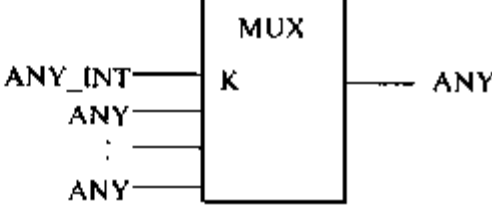
$RUN := (START \text{ OR } RUN) \text{ AND } NOT \text{ STOP};$

程序中, $START$ 、 $STOP$ 和 RUN 分别由 8 个电动机对应的起动、停止和接触器组成。它们采用字节数据类型。IEC 61131-3 标准编程语言允许用字节、字存储器等表示触点,因此,8 个电动机的起停控制只有一句语句,大大简化了程序。

(5) 选择和比较类函数

1) 选择类函数。选择类函数(Selection Function)用于根据条件来选择输入信号作为输出返回值。选择的条件包括单路选择,或输入信号本身的最大、最小、限值和多路选择等。表 2-30 所示为选择函数的图形形式和文字形式。

表 2-30 选择函数的图形形式和文字形式

序号	FBD 编程语言的图形形式	说明及 ST 编程语言文字形式的示例
1		两位选择:根据 G 取输入之一作为输出 $OUT := IN0 \quad \text{IF } G = 0$ $OUT := IN1 \quad \text{IF } G = 1$ 示例: $A := SEL(G := 0, IN0 := X, IN1 := 5);$
2		可扩展的最大函数:输入的最大值作为输出 $OUT := MAX(IN1, IN2, \dots, INn);$ 示例: $A := MAX(B, C, D);$
3		可扩展最小函数:输入的最小值作为输出 $OUT := MIN(IN1, IN2, \dots, INn);$ 示例: $A := MIN(B, C, D);$
4		限值器:输出被限幅在最大值和最小值之间 $OUT := MIN(MAX(IN, MN), MX);$ 示例: $A := LIMIT(MN := 0, IN := B, MX := 5);$
5		可扩展多路选择器: 根据输入 K,选择 N 个输入中的一个。 示例: $A := MUX(0, B, C, D);$ 与 $A := B;$ 有相同结果

使用选择类函数时应注意下列事项:

- 不同选择函数的各输入变量允许的数据类型见表 2-30。
- 从上到下顺序, $IN1, IN2, \dots, INn$ 符号表示输入, OUT 表示输出的返回值多路选择器 MUX 函数中无名称的输入,从上到下顺序,有缺省的名称 $IN0, IN1, \dots, INn$,其中, n 是输入变量的总数。这些名称可以显示,也可不在图形形式中显示。
- 多路选择器 MUX 函数也可用 MUX_*_* 的文字形式表示, $*$ 是输入变量 K 的数据类型, $**$ 是其他输入和输出变量的数据类型。例如, $MUX_INT_REAL(A, B, C)$ 。
- 应用时, MUX 的输入变量 K 的数据应在被选择变量个数 n 的范围内,即输入变量 K 的值应为 $0 \sim n-1$ 。例如, $A := MUX(0, B, C, D);$ 表示 $K = 0$,因此,选择第二个输入变量 B 作为其函数的返回值。
- 除 SEL 选择函数的 G 输入和 MUX 多路选择器的 K 输入外,其他选择类函数中的输入和输出需采用相同的数据类型,以保证数据类型的正确。
- LIMIT 函数是限幅函数。当输入 IN 在下限 MN 和上限 MX 之间时,输出的返回值与输入相等。当低于下限时,输出返回值被限幅到下限输出。反之,当高于上限时,输出返回值被限幅到上限输出。图 2-11 是 LIMIT 函数的输入输出关系曲线。

- 选择函数运算结果的数据类型与输入有相同数据类型。对特定的函数调用,所有选择函数的输入应有相同的数据类型。

【例 2-46】 反应器温度控制示例。

TEMPMAX:= MAX(TEMP1,TEMP2,TEMP3,TEMP4);(* 选择 4 个温度的高值 *)

示例取 4 个温度测量值的最高值,并用最高温度控制反应器温度,防止反应器过热。

【例 2-47】 通道切换示例。

VALUE:= MUX (K:= CHANN0, IN0:= CHAN0, IN1:= CHAN1, IN2:= CHAN2);

根据 CHANN0 的值,将来自 3 个通道中的某一通道测量值赋值给 VALUE。

2) 比较类函数。比较类函数(Comparison Function)用于比较多个输入变量数值的大小。表 2-31 给出了比较类函数的图形形式和文字形式。

表 2-31 比较类函数的图形形式和文字形式

图 形 形 式	文 字 形 式 的 示 例
<div> <div> <div>***</div> <div>图 5</div> </div> <div> <div>*** 名称或符号</div> <div>BOOL</div> </div> </div> <div> <div>ANY_ELEMENTARY</div> <div>:</div> <div>ANY_ELEMENTARY</div> </div>	<div> <div>A:= GT(B,C,D);</div> <div>或</div> <div>A:= (B > C) & (C > D);</div> </div>

使用比较类函数时注意下列事项:

- 比较类函数采用文字形式时,有函数名和符号等两种表示形式。表 2-32 是比较类函数的两种文字形式。

表 2-32 比较类函数的两种文字形式

序 号	符 号	函 数 名	示 例
1	>	GT	递减序列:OUT:= (IN1 > IN2) & (IN2 > IN3) & ... & (INn-1 > INn) OUT:= GT(IN1,IN2,...,INn)
2	>=	GE	单调序列:OUT:= (IN1 >= IN2) & (IN2 >= IN3) & ... & (INn-1 >= INn) OUT:= GE(IN1,IN2,...,INn)
3	=	EQ	相等:OUT:= (IN1 = IN2) & (IN2 = IN3) & ... & (INn-1 = INn) OUT:= EQ(IN1,IN2,...,INn)
4	<=	LE	单调序列:OUT:= (IN1 <= IN2) & (IN2 <= IN3) & ... & (INn-1 <= INn) OUT:= LE(IN1,IN2,...,INn)
5	<	LT	递增序列:OUT:= (IN1 < IN2) & (IN2 < IN3) & ... & (INn-1 < INn) OUT:= LT(IN1,IN2,...,INn)
6	<>	NE	不相等(不可扩展):OUT:= (IN1 <> IN2) OUT:= NE(IN1,IN2)

- 从上到下顺序,IN1,IN2,...,INn 符号表示输入,OUT 表示输出的返回值。
- 所有比较函数返回值的数据类型是布尔值数据类型。因此,比较函数可改变结果存储

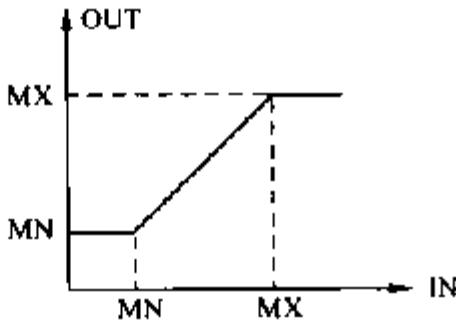


图 2-11 LIMIT 的输入输出关系

器内存储的数据类型。

- 表中,& 表示与逻辑运算,也可用 AND 符号,它表示各个比较需同时满足。
- 比较的输入变量应具有相同的数据类型,可以比较所有数据类型,例如,可比较数值,也可以比较字符串。
- NE 函数用于比较两个输入变量的不相等,它是不可扩展的,即只有两个变量进行比较。其他比较类函数都有多个输入变量可选,因此,是可扩展的。
- 大于比较是递减序列的比较,小于比较是递增序列的比较,大于等于比较是单调序列的比较,小于等于比较也是单调序列的比较。
- 除了不等于的比较函数 NE 外,其他比较函数可以有多个输入,比较结果是各比较结果的与逻辑。见表 2-32 序号 1~6。
- 支持函数名表示时,一致性语句用后缀 n 表示,支持符号名表示时,一致性语句用后缀 s 表示。例如,1n 表示比较函数 GT,2s 表示符号 >=。
- 两个位串的长度不等时,可在短串的右面填充零,然后进行比较运算。因此,位串的比较应与无符号整数变量的比较具有相同的结果。

【例 2-48】 字符串比较的示例。

A = 'abc' > 'abcdef';

示例比较字符串'abc'与'abcdef',对短串'abc'的右面填充 0,成为'abc000',因此,比较结果显示,A = FALSE。

- 位串的比较从最高有效位(最左面)开始,逐位向最低有效位(最右面)进行比较。因此,'ABC'比'CBA'小,'AX'比'AGGY'大。规定位串比较的集合是 IEC 646。
- 比较类函数的符号在文本类编程语言中也作为操作符号使用。
- 可以比较的输入变量个数是与执行有关的参数。

【例 2-49】 锅炉压力检查的示例。

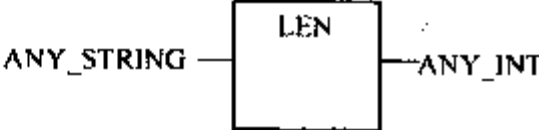
PRESSURE_OK := GT(P1,P2,P3,P4);

示例检查锅炉各检测点压力,如果 P1 到 P4 各点的压力递减(即 $P1 > P2 > P3 > P4$),则表示锅炉压力系统运行正常。

(6) 字符串类函数

字符串类函数(Character String Function)用于对输入的字符串进行处理,例如,确定字符串的长度、对输入的字符串进行截取、处理后的新字符串作为该函数的返回值。标准字符串函数能简化字符串的处理,例如,用于建立传送到远程设备的操作员报文或信息。表 2-33 显示字符串函数的图形形式和文字形式的示例。

表 2-33 字符串函数的图形形式和文字形式

函 数	图 形 形 式	文字形式的示例
字符串长度		A := LEN('WSTRING'); 等效于 A := 7;

(续)

函 数	图 形 形 式	文 字 形 式 的 示 例
截取 IN 的最左面 L 个字符		A: = LEFT(IN: = 'ASTR',L: =3); 等效于 A: = 'AST';
截取 IN 的最右面 L 个字符		A: = RIGHT(IN: = 'STAR',L: =3); 等效于 A: = 'TAR';
截取 IN 的 L 个字符, 开始于第 P 个字符		A: = MID(IN: = 'STAR',L: =2,P: =2); 等效于 A: = 'TA';
字符串串联扩展		A: = CONCAT('AB','CD','E'); 等效于 A: = 'ABCDE';
将 IN2 插入 IN1, 从 IN1 第 P 字符后面插入		A: = INSERT(IN1: = 'ABC',IN2: = 'XY',P: =2); 等效于 A: = 'ABXYC';
从 IN1 第 P 字符开始, 删除 IN1 中的 L 个字符		A: = DELETE(IN1: = 'ABXYC',L: =2,P: =3); 等效于 A: = 'ABC';
从 IN1 的第 P 个字符开始, IN2 替换 IN1 的 L 个字符		A: = REPLACE(IN1: = 'ABCDE', IN2: = 'X',L: =2,P: =3); 等效于 A: = 'ABXE';
寻找 IN1 中 IN2 第一次出现的位置, 如果未找到, 则 OUT: =0		A: = FIND(IN1: = 'ABBCBCBC',IN2: = 'BC'); 等效于 A: =3;

使用字符串类函数时注意下列事项:

- 1) 字符串的位置从最左面开始编号, 依次为 1,2,3, ..., L。L 是字符串长度。
- 2) 字符串类函数中, 用于定位的输入变量应是大于零的整数。定位的整数应使运算能够正确进行。如果函数运算结果要存取一个在字符串中不存在的字符位置时, 系统就出错。
- 3) 字符串串联扩展 CONCAT 函数是可扩展的, 即输入字符串的个数可扩展。
- 4) 最大串的长度是与执行有关的参数。

【例 2 - 50】 建立批量控制的有关字符串。

```
VAR
    Recipe_Spec;STRING(12):= 'Recipe_x_141';( * Recipe_Spec 字符串变量的初始值 * )
    Recipe;STRING(14);
    Job_Code;STRING(3);
    Batch  ;STRING(20);
    Batch_ID;INT:= 7;
END_VAR
Job_Code:= RIGHT( IN:= Recipe_Spec,L:= 3);    ( * 取 141 作为 Job_Code 的代码 * )
Recipe:= REPLACE ( IN1:= Recipe_Spec,IN2:= 'A7X',L:= 1,P:= 8);
    ( * 建立新的配方名称为 Recipe_A7X_141 * )
Batch:= CONCAT ( Recipe,'_',INT_TO_STRING( Batch_ID));
    ( * 建立该批量的描述是,Recipe_A7X_141_7 * )
```

(7) 时间数据类函数

时间数据类函数(Function of Time Data Types)是上述有关函数对时间数据类型数据的扩展。包括时间数据的类型转换函数、时间数据类型的算术类函数、时间数据类型的字符串类函数等。

1) 类型转换函数。用于将时间数据类型提取部分数据作为另一种时间数据类型。分 DT 到 TOD 的转换和 DT 到 DATE 的转换两种。DT 到 TOD 转换函数 DT_TO_TOD 将日期和时刻数据中提取一天中的时间。DT 到 DATE 的转换 DT_TO_DATE 将日期和时刻数据中提取日期。

【例 2 - 51】 时间数据类型转换的示例。

```
VAR
    EVENT_M;DT:= DT#2005-05-19-21:05:08;
END_VAR
EVENT_TIME:= DT_TO_TOD(EVENT_M);
EVENT_DATE:= DT_TO_DATE( EVENT_M);
```

运算结果是:EVENT_TIME = TOD#21:05:08;EVENT_DATE = DATE#2005-05-19。

2) 算术函数。持续时间数据类型的算术类函数特别适合有关时间的转换,见表 2 - 34。

表 2 - 34 时间数据类型的算术函数

算 术 函 数					
序 号	函 数 名	符 号	IN1	IN2	OUT
1a	ADD	+	TIME	TIME	TIME
1b	ADD_TIME	+	TIME	TIME	TIME
2a	ADD ^注	+ ^注	TIME_OF_DAY	TIME	TIME_OF_DAY
3a	ADD ^注	+ ^注	DATE_AND_TIME	TIME	DATE_AND_TIME
4a	SUB	-	TIME	TIME	TIME
4b	SUB_TIME	-	TIME	TIME	TIME

算术函数					
序 号	函 数 名	符 号	IN1	IN2	OUT
5a	SUB ^注	_并	DATE	DATE	TIME
6a	SUB ^注	_并	TIME_OF_DAY	TIME	TIME_OF_DAY
7a	SUB ^注	_注	TIME_OF_DAY	TIME_OF_DAY	TIME
8a	SUB ^注	_注	DATE_AND_TIME	TIME	DATE_AND_TIME
9a	SUB ^注	_注	DATE_AND_TIME	DATE_AND_TIME	TIME
10a	MUL ^注	* 注	TIME	ANY_NUM	TIME
11a	DIV ^注	/ 注	TIME	ANY_NUM	TIME
12	CONCAT_DATE_TOD		DATE	TIME_OF_DAY	DATE_AND_TIME
类型转换函数					
13	DT_TO_TOD	DT 类数据转换为 TOD 类数据			
14	DT_TO_DATE	DT 类数据转换为 DATE 类数据			

注：部分用法不再在本标准的将来版本使用。

虽然，一些算术函数可以实现，例如，TOD 数据与 TIME 数据进行 ADD 函数运算，结果是 TOD 数据。TIME 数据与 ANY_NUM 数据进行 MULTIME 函数，结果是 TIME 数据等。但标准没有赞成其他算术函数用法（主要是符号表示的函数用法）。因此，这里未全部列出，它们在将来的版本中也不再使用。

【例 2-52】 时间数据的加和减函数示例。

A:=T#35s+T#2h5s;B:=T#1h3s-T#1h18s;

运算结果：A=T#2h40s;B=T#-15s。由于持续时间的数值允许有负值，因此，运算结果 B 的负值是允许的。

【例 2-53】 时间数据的乘函数示例。

```
VAR
    TIME_BASE:TIME:= T#100 ms;
    ON_TIME:TIME;
    SCALE:INT:=100;
END_VAR
ON_TIME:= MUL(TIME_BASE,SCALE);
```

示例用于计算以时间基 TIME_BASE 的整数倍数的接通时间 ON_TIME。

当支持用函数名表示时，在一致性语句中，用后缀 n 表示函数名，因此，4n 表示 SUB。当支持用符号名表示时，在一致性语句中，用后缀 s 表示符号名，因此，1 s 表示 +。

3) 字符串函数。表 2-34 的 CONCAT_DATE_TOD 函数将日期和一天中的时间串联扩展组成日期和时刻类型的数据。

【例 2-54】 建立 DT 时间文字。

```
VAR
```

```
ATTACKDATE;DATE:D#2001-09-11;
ATTACKTIME;TIME:= TOD#08:46:15;
TIMESTAMP;DATE_AND_TIME ;
END_VAR
TIMESTAMP:= CONCAT( STARTDATE,ALARMTIME);
( * 时间标记为 2001-09-11-08:46:15 * )
```

示例将攻击日期和攻击时间组合成为时间标记,即 911 事件发生的时间标记。

(8) 枚举数据类型函数

枚举数据类型函数(Functions of Enumerated Data Types)适用于选择和比较类型函数。可以看到,选择函数 SEL 和 MUX 的输入变量是 ANY 类型,因此,它适用于衍生数据类型。当用于枚举数据时,输入和输出的枚举数据个数应相同。例如,在正常工况,某个变量可用三原色的某种组合显示,在某限值(例如,下限)时,该变量以另一种颜色组合显示,用于作为警告信号。当达到另一限值(例如,下下限)时,需用第三种颜色的组合显示并报警。这类应用项目就可用枚举变量的三种不同颜色组合作为多路选择器的 3 个输入变量,并用 K 来选择输出采用哪种颜色组合,K 对应 3 种不同的工况(用 0、1 和 2 表示)。比较函数 EQ 和 NE 的输入变量是 ANY_ELEMENTARY,它也适用于衍生数据类型。表 2-35 是能用于枚举类数据类型的选择和比较函数。

表 2-35 用于枚举类数据类型的选择和比较函数

序 号	函 数 名	符 号	对 应 特 性	描 述
1	SEL		表 2-30 序号 1	根据布尔条件选择两个枚举数据值的一个
2	MUX		表 2-30 序号 5	根据整数选择器的值从许多枚举数据类型值中选择一个
3	EQ	=	表 2-32 序号 3	检测两个有相同数据类型的枚举值是否相等
4	NE	<>	表 2-32 序号 6	检测两个有相同数据类型的枚举值是否不相等

枚举数据类型适用于比较类函数的 EQ 和 NE 函数。如果输入变量的枚举数据对应相等,则 EQ 函数的返回值为 1。如果两个输入变量的枚举数据不相等,则 NE 函数的返回值为 1。

4. 衍生函数

衍生函数是用户自定义的函数,它是用户编写的函数,可以是标准函数的组合或调用,也可以是根据应用项目的要求编写的函数。

【例 2-55】 球面积函数 SPHERE。

需要计算球面积可编写球面积函数。图 2-12 是球面积函数图形形式和文字形式。

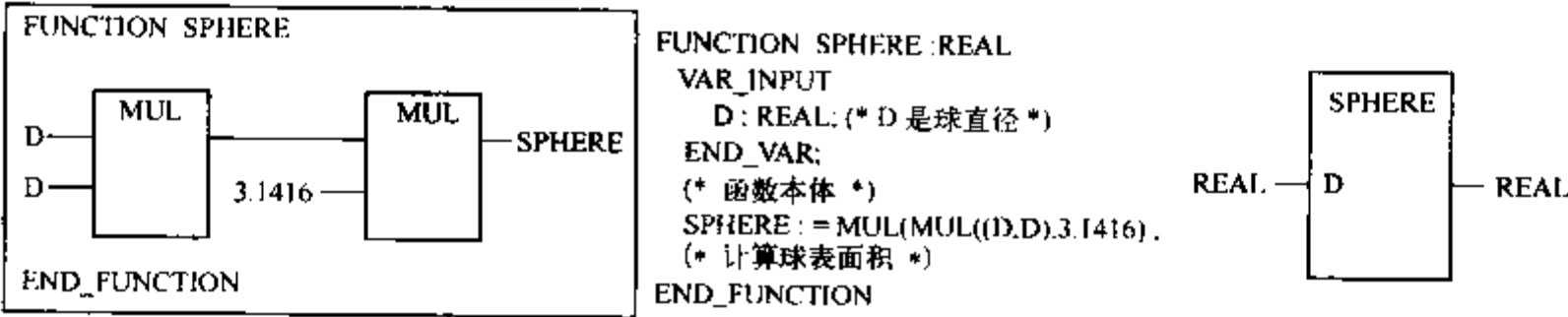


图 2-12 SPHERE 函数的图形形式和文字形式

在该函数中,输入变量 D、返回值 SPHERE,都为实数数据类型,如图 2-12 所示。

球面积 SPHERE 函数中,有输入变量 D(球直径),它是 REAL 数据类型,不设置初始值,圆周率直接输入实数数据类型,输出返回值为 SPHERE。在 SPHERE 函数中,用两个标准 MUL 函数,第一个 MUL 函数实现球直径的平方运算,第二个 MUL 函数实现球径平方与圆周率的相乘运算,SPHERE 作为该函数的返回值。也可用可扩展的 MUL 函数(三个输入相乘)直接计算返回值 SPHERE。

须注意,函数只有返回值,因此,如果希望根据球直径 D,同时计算球表面积、球体积等数值,可采用用户功能块。

5. 函数的附加属性

函数具有附加属性。例如,过载属性、可扩展属性等。

(1) 过载属性

如果函数的输入变量不限于某类单一的数据类型,而可用于不同数据类型,例如,一般数据类型,则称该函数具有过载(Overloading)属性或超载属性。标准函数都具有过载属性,它能够适用于不同的数据类型。如果函数只适用于某类数据类型,则需在函数名中给予声明,这称为函数的类型化(Typed)。例如,加运算的输入变量只能是整数数据类型,则该加 ADD 函数名表示为 ADD_INT。函数的类型化使函数的使用面缩小,并容易造成数据类型的不匹配而使系统出错。

过载属性使函数的应用变得方便,用户可不必担心数据类型是否匹配等问题。但应注意,不同函数对输入变量数据类型的要求是不同的,因此,应用时,应选用上述各函数图形形式或文字形式中允许的数据类型。

过载属性实质是将不同的数据类型经数据类型转换函数转换为所需的数据类型,并进行有关的函数运算,运算结果再转换为所需的数据类型。

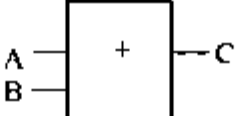

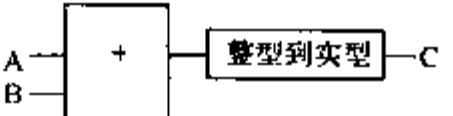
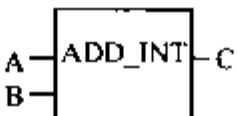

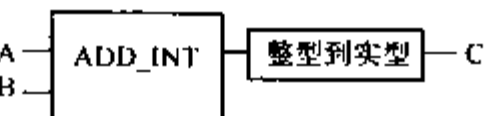
在使用时应注意,标准仅对标准函数定义了函数的过载属性。对非标准的衍生函数并未定义它的过载属性。当某系统支持类型化属性时,应声明其提供的标准函数中,哪些函数具有过载属性,哪些函数是类型化属性。表 2-36 的用法 a 和 b 是过载属性和类型化属性的显式形式数据类型转换示例。

【例 2-56】 过载属性示例。

```
VAR
    A,SQRT_A:REAL;
    B,SQRT_B:LREAL;
END_VAR
SQRT_A:=SQRT(A);(* 采用类型化属性时,可表示为:SQRT_A:=SQRT_REAL(A);*)
SQRT_B:=SQRT(B);(* 采用类型化属性时,可表示为:SQRT_B:=SQRT_LREAL(B);*)
```

示例中,SQRT 函数既可用于实数 A 的开方,也可用于长实数 B 的开方。但须注意,当 SQRT 函数用于实数开方时,计算结果应为实数数据类型,用于长实数开方时,计算结果应为长实数数据类型。当类型化时,应将函数与可应用的数据类型之间用下划线分隔。

表 2 -36 过载属性的显式形式数据类型转换示例

序 号	1	2	3
类型声明	VAR A: = INT ; B: = INT ; C: = INT ; END_VAR	VAR A: = INT ; B: = REAL ; C: = REAL ; END_VAR	VAR A: = INT ; B: = INT ; C: = REAL ; END_VAR
用法 a	 C: = A + B ;	 C: = INT_TO_REAL(A) + B ;	 C: = INT_TO_REAL(A + B) ;
用法 b	 C: = ADD_INT(A, B) ;	 C: = ADD_REAL(INT_TO_REAL(A), B) ;	 C: = INT_TO_REAL(ADD_INT(A, B)) ;

(2) 可扩展属性

函数的输入变量个数可以扩展的属性称为函数的可扩展属性。例如,ADD 函数的输入变量可以不局限于两个,它可以实现多个输入变量的加法,因此,ADD 函数具有可扩展属性。此外,乘函数、与逻辑、或逻辑等函数也具有可扩展属性。但减法 SUB 函数只能对两个输入变量进行运算,因此,SUB 不具有可扩展属性。具有可扩展属性的函数可简化程序,降低所需的存储空间。例如,例 2 -54 可用可扩展乘函数直接将 3 个输入进行相乘。图 2 -13 是具有可扩展属性的一些函数示例。

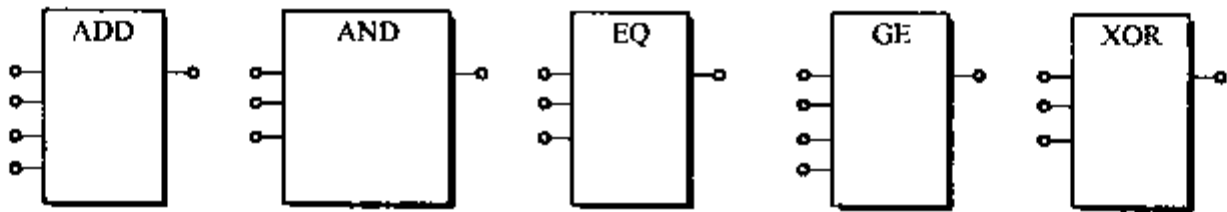


图 2 -13 具有可扩展属性的函数示例

【例 2 -57】 过载属性示例。

LD	RPM	(* 读取 RPM 的转速数值 *)
GE	RPM1	(* 与 RPM1 比较,是否 RPM 大于 RPM1 *)
GE	RPM2	(* 与 RPM2 比较,是否 RPM1 大于 RPM2 *)
ST	% QX0.0	(* 比较结果满足,则输出% QX0.0 置 1 *)

(3) 调用属性

函数有函数名,函数具有调用属性,函数的调用是通过函数名调用实现的。函数的文本调用属性由函数名及其后面的自变量列表组成。

【例 2 -58】 函数调用示例。

A: = LIMIT (EN: = COND,IN: = B,MX: =5,ENO =>TEMPL) ;
B: = SEL(C: = SW_1,IN0: = D1,IN1: = D3);

自变量列表中的各变量用逗号分隔,用圆括号将自变量列表括起来。函数的调用中,自变

量列表分形参和非形参两类。表 2 - 37 是函数调用时的示例。

表 2 - 37 函数调用的形参和非形参列表

序号	特 性				用 ST 编程语言时函数调用的示例
	调用类型 (Invocation Type)	变量配置 (Variable Assignment)	变量阶次 (Variable Order)	变量个数 (Number of Variables)	
1	形参	是	任意	任意	A := LIMIT(EN := COND, IN := B, MX := 5, ENO => TEMPL) ;
2	非形参	否	固定	固定	A := LIMIT(1, B, 5) ;

注:序号 1 中,MN 参数的初始值设置为 0,示例中未显示。序号 2 的示例与下列形参变量的调用等效:A := LIMIT (EN := TRUE, MN := 1, IN := B, MX := 5) ;。

函数调用用操作符“:=”设置输入变量和输入输出变量的值。用操作符“=>”设置输出变量的值到变量,如表 2 - 37 所示。函数调用的方法详见下述。

(4) EN 和 ENO 属性

在梯形图和功能块图编程语言中,函数具有其他编程语言所不具有的属性。即 EN 和 ENO 属性。EN 和 ENO 分别是该函数的使能输入和使能输出。EN 和 ENO 可在函数中使用任一个、或两者或都不使用。附加的 EN 和 ENO 属性用于控制执行过程的进行与否。

使能输入和使能输出的应用原则如下:

- 1) 当该函数被调用时,EN 的值是 0 (FALSE),则该函数体定义的操作不被执行,同时,ENO 的值也被复位到 0 (FALSE)。
- 2) 当该函数被调用时,EN 的值是 1 (TRUE),则该函数体定义的操作被执行,同时,ENO 的值也被置位到 1 (TRUE)。
- 3) EN 和 ENO 属性是附加属性,其变量需要在变量声明段声明。

【例 2 - 59】 EN 和 ENO 的变量声明。

```
VAR_INPUT
    EN:BOOL:=1;
END_VAR
VAR_OUTPUT
    ENO:BOOL;
END_VAR
```

示例中将 EN 变量的初始值设置为 1,可保证在上电后,具有 EN 和 ENO 附加属性的函数能够被执行。

- 4) 当函数执行过程中出错时,例如,类型转换出错、数值结果超出该数据类型的范围或除以零等,则 ENO 的值被自动复位到 0 (FALSE)。
- 5) 如果函数 ENO 输出的求值结果为 FALSE,则所有该函数的输出 (VAR_OUTPUT, VAR_IN_OUT 和函数返回) 值是与执行有关的 (Implementation-dependent)。

【例 2 - 60】 EN 和 ENO 在梯形图程序中的应用。

图 2 - 14 给出了 EN 和 ENO 在梯形图程序中的应用示例。

图中,SUB 函数的 EN 直接连接到梯形图的左电源轨线,因此,上电后,直接执行减法运算。其 ENO 输出直接连接到比较函数 LT 的 EN,因此,只要 SUB 运算没有出错,就执行 LT 的

比较运算。如果运算出错,例如,PTK 数据类型不是整数,或运算结果超出整数允许的数值范围,则 LT 运算将不执行,即 SUB 的输出 ENO 为 0(FALSE)。

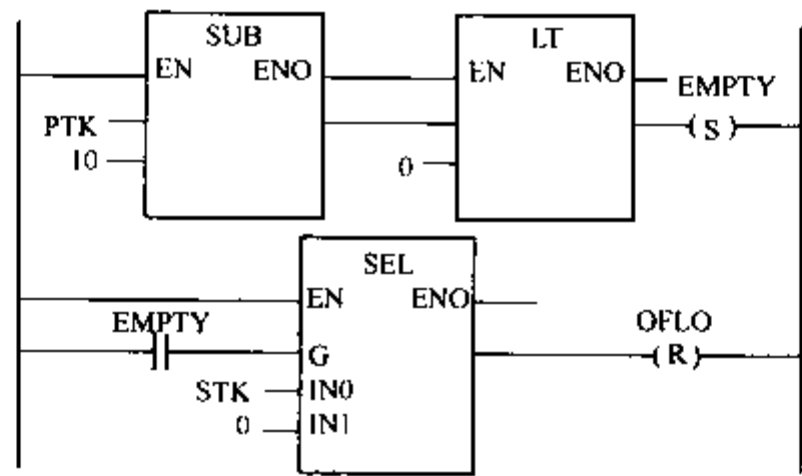


图 2 - 14 EN 和 ENO 在梯形图程序中的应用示例

类似地,SEL 选择函数的 EN 也直接连接到梯形图的左电源轨线,因此,上电后其值为 1 (TRUE),当其输入 G(来自上一梯级的 EMPTY)为 1 时,才能执行 STK 和 0 的选择,并将结果送输出 OFLO(溢出标志)。

当 EMPTY 为 1 时,选择函数返回值为 IN1 的值 0,反之,如果 EMPTY 为 0,SEL 选择函数的返回值取 STK 的值。表 2 - 38 给出了 EN 和 ENO 的使用示例。

表 2 - 38 EN 和 ENO 的使用示例

序号	特 性	示 例	说 明
1	在梯形图程序中 EN 和 ENO 的使用		当 ADD_EN 为 1 时,才能执行 ADD 函数的运算,运算不出错时,ENO 被置 1
	在结构化文本编程语言程序中 EN 和 ENO 的使用	<pre> EN := ADD_EN; C := ADD(A,B); ADD_OK := ENO; </pre>	
2	在功能块图程序中没有使用 EN 和 ENO 的 ADD 函数		上电后执行 ADD 的运算
3	在功能块图程序中没有使用 ENO 的 ADD 函数		当 ADD_EN 为 1 时,才能执行 ADD 函数的运算,运算出错时,没有 ENO 的出错标志
4	在功能块图程序中没有使用 EN 的 ADD 函数		上电后执行 ADD 的运算,如果运算正确,则标志 ENO 为 1,使 ADD_OK 置 1

2.6.2 功能块

功能块(Function Block)是在执行时能够产生一个或多个值的程序组织单元。在控制系统中,功能块实际是某种控制算法,例如 PID 功能模块被用于闭环控制,其他功能模块可用于计数器、斜坡和滤波等。

功能块类似于传统可编程控制器的功能块。传统可编程控制器中,通常采用功能块编号,例如,定时器 T16 等。IEC 61131-3 标准用功能块实例名来定义被调用的功能块,因此,编程人员可以使用具有相同类型的不同定时器或计数器,而不需要对其名称进行检查。

1. 功能块表示和功能块声明

变量的实例化是编程人员在变量声明部分用指定变量名和相应数据类型来建立变量的过程。同样,功能块实例化是编程人员在功能块声明部分用指定功能块名和相应的功能块类型来建立功能块的过程。每个功能块实例有它的功能块名和其内部变量、输出变量及可能的输入变量数据结构。该数据结构的输出变量和必要的内部变量的值能够从这次执行保持到下一次执行。功能块实例的外部只有输入和输出变量是可存取的。功能块的内部变量对用户来说是隐含的。功能块的图形表示见图 2-15。

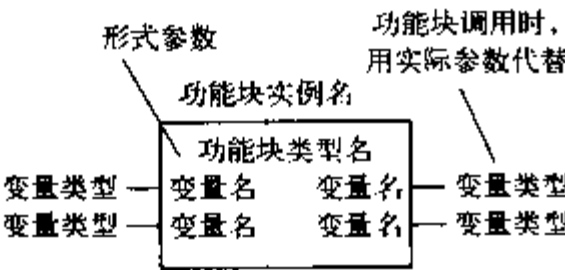


图 2-15 功能块的图形表示

(1) 功能块表示

用表示函数的相同方法来表示功能块。功能块可以用文本形式和图形形式表示。功能块采用 FUNCTION_BLOCK...END_FUNCTION_BLOCK 结构。在功能块结构中,包括功能块名、功能块变量声明和功能块本体。

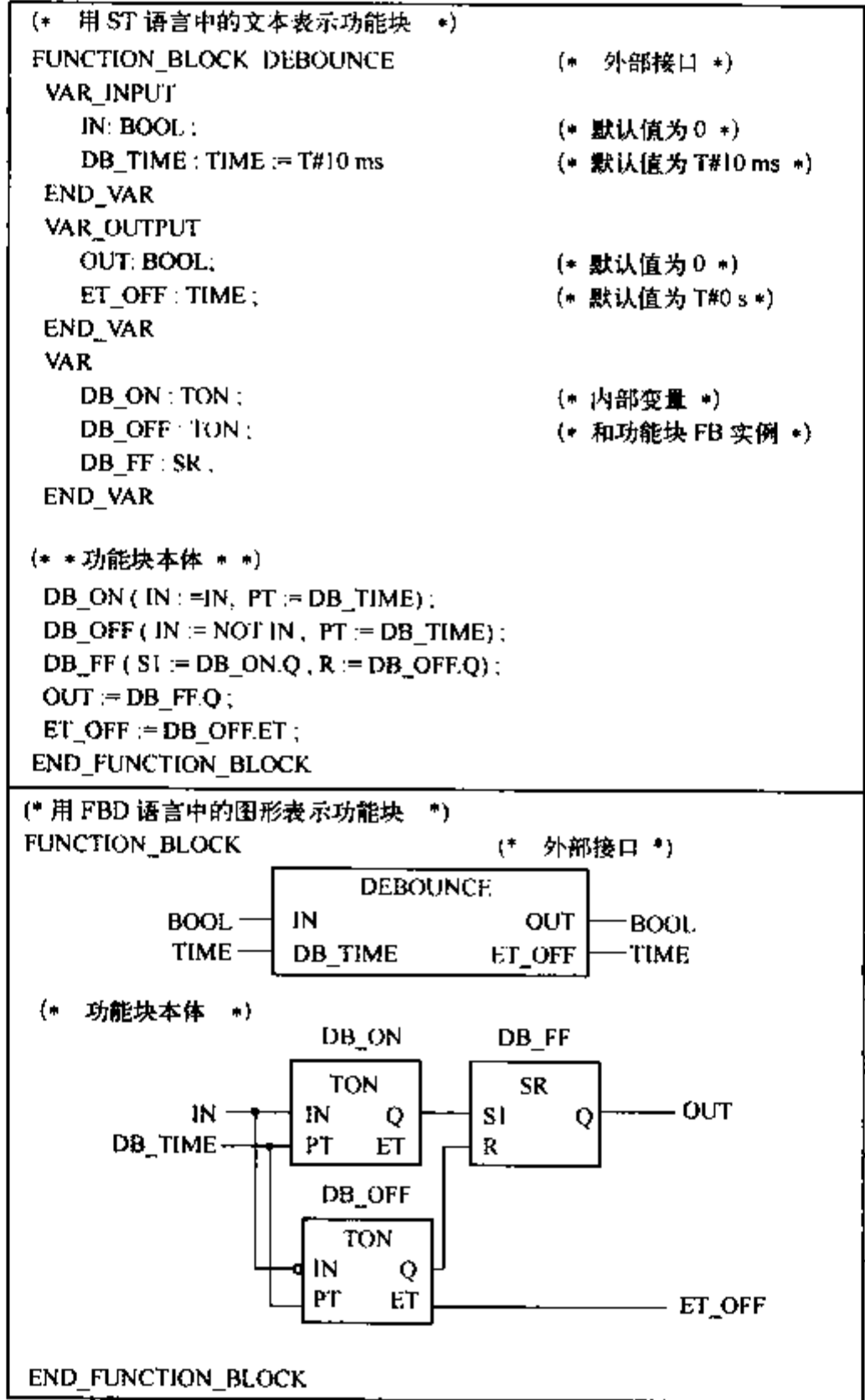
图 2-16 给出了一个衍生功能块文本形式和图形形式的表示示例。衍生功能块名是 DEBOUNCE。它的输入变量 IN 是布尔数据类型,输入变量 DB_TIME 是时间数据类型,初始值为 T#10 ms。它的输出变量 OUT 是布尔数据类型,输出变量 ET_OFF 是时间数据类型。它有 3 个内部变量,DB_ON 和 DB_OFF 是 TON 功能块实例名,DB_FF 是 SR 功能块实例名。功能块实例的本体是 3 个标准功能块的调用和运算,实现延时闭合和延时断开的功能。

功能块与函数的区别是:功能块没有返回值的数据类型声明,它有输出变量;对功能块内部变量也有不同的使用类型;功能块具有静态变量(即具有记忆功能)。因此,当用相同的输入变量调用时,功能块的输出与内部变量和外部变量的状态有关。函数则总是生成相同的结果。功能块有输出,因此有输出变量。而函数只有返回值,没有输出变量。在功能块内,允许使用 VAR、VAR_INPUT、VAR_IN_OUT、VAR_OUTPUT、VAR_EXTERNAL 变量,不允许使用 VAR_GLOBAL、VAR_ACCESS 等变量。由于没有返回值,因此,功能块表示中,没有冒号和返回值数据类型。

功能块变量声明包括:输入变量声明,输出变量声明,及输入输出变量声明,外部变量声明,变量声明和保持变量声明等。

功能块本体是标准定义的图形或文字(含 SFC)类编程语言编写的程序。

功能块实例是保持在由功能块类型定义结构中特定数据类型的集合。它是一个数据结构,有一个在功能块类型内被定义的有关算法。



示例中,功能块实例可作为输入变量、输入输出变量或外部变量被声明。

(2) 功能块中变量的声明

功能块中变量声明与函数中变量声明类似。编写功能块表示时应注意下列事项:

1) 功能块的内部和输出变量可用限定属性 RETAIN,用于表示该变量具有保持功能。而输入和输入输出变量只能在调用时声明具有保持属性。

2) 属性 R_EDGE 和 F_EDGE 用于表示布尔输入的边沿检测。

【例 2-62】 功能块的边沿检测示例。

```
FUNCTION_BLOCK OR_EDGE
VAR_INPUT
    X:BOOL R_EDGE;    (* X 变量被声明为上升沿边沿检测的布尔变量 *)
    Y:BOOL F_EDGE;    (* Y 变量被声明为下降沿边沿检测的布尔变量 *)
END_VAR
VAR_OUTPUT
    Z:BOOL;            (* 布尔变量 Z 是输出变量 *)
END_VAR
Z := X OR Y;          (* ST 语言的示例 *)
END_FUNCTION_BLOCK
```

示例中的衍生功能块 OR_EDGE 是边沿检测功能块名。X 和 Y 中只要有一个是边沿检测的布尔量输入时,功能块 OR_EDGE 就有输出 Z。

3) 通过采用 VAR_EXTERNAL 结构传递的功能块变量值能够从该功能块内部修改。

4) 功能块实例能用文本方式,通过在 VAR...END_VAR 变量声明的结构中,使用已声明的功能块来声明数据元素和标识。

5) 如果功能块的实例名在 VAR_INPUT 变量声明中被声明作为一个功能块的输入变量,或在 VAR_IN_OUT 变量声明中被声明作为一个功能块的输入输出变量,则功能块的实例名就能用于作为一个功能块的输入。

6) 功能块实例用图形形式表示。表示方法与函数的图形表示方法相同。

7) 一般不允许对功能块输出变量赋值。只有当输入作为功能块的调用部分时,才允许对功能块输入变量赋值。

8) 由于功能块允许调用函数和功能块,因此,如上所示,也可将调用功能块实例作为其他功能块实例的变量等。如 DB_FF(SI := DB_ON, Q, R := DB_OFF, Q)。

9) 使用时应根据软件产品中有关功能块的参数名表示。例如,SR 功能块的 S 端常用 SET 表示等,因此,可用 DB_FF(SET1 := DB_ON, Q, RESET := DB_OFF, Q)表示。

10) 功能块输入的不赋值和不连接表示保持它们的初始值或最近调用的值。

功能块允许的输入输出用法见表 2-39。功能块实例的示例见表 2-40。

表 2-39 功能块输入输出变量用法示例

用 法	功能块内部	功能块外部
读输入	IF IN1 THEN ...	不允许 ^{①②}
对输入赋值	不允许 ^①	FB_INST (IN1 := A, IN2 := B);

(续)

用 法	功能块内部	功能块外部
读输出	OUT:= OUT AND NOT IN2 ;	C:= FB_INST. OUT;
对输出赋值	OUT:= 1;	不允许(注1)
读输入输出	IF INOUT THEN ...	IF FB1.INOUT THEN ...
对输入输出赋值	INOUT:= OUT OR IN1;③	FB_INST(INOUT:= D);

- ① 本表所列不允许的用法会导致与执行有关的不可预知的影响。
- ② 一个功能块的输入、输出和内部变量的读和写可以用 IEC 61131-3 的“通信功能”,“操作员接口功能”或“编程、测试和监测功能”完成。
- ③ 功能块内部允许修改在 VAR_IN_OUT 变量段中声明的变量。

11) 重复使用功能块的限制。可重复使用的功能块存在下列限制:

- 为确保功能块不依赖于硬件,功能块的变量声明中不允许将具有固定赋值的 PLC 硬件地址变量(即直接表示变量,例如,%IX1.1,%IW4,%QM25)作为局部变量,但在 VAR_EXTERNAL 变量中可使用 PLC 硬件地址变量作为全局变量。
- 功能块中不允许对 VAR_ACCESS 变量类型或 VAR_GLOBAL 变量的存取路径进行声明。但可用 VAR_EXTERNAL 变量间接存取路径来存取全局变量。
- 只能用功能块、程序的程序组织单元接口,将参数和外部数据传递到功能块。

表 2-40 示例 1 的 FF31 是标准功能块 SR 的实例名,它在 VAR 变量段声明。调用 FF31 是用 FF31 的实参代入形参,即%IX1.0 作为 SR 功能块的 S1 输入,%IX1.1 作为 SR 功能块的 R 输入。调用 SR 功能块,赋值输出是调用 SR 功能块的输出 FF31.Q1,并赋值给%QX1.0。

表 2-40 示例 2 的 MyTon 是标准功能块 TON 的实例名,与其他内部变量 a、b、r 和 out 一起在 VAR 变量段声明。调用 MyTon 是用实参代入形参,示例中还将调用的结果直接用赋值操作符“=>”赋值给输出变量 out。

表 2-40 功能块实例的示例

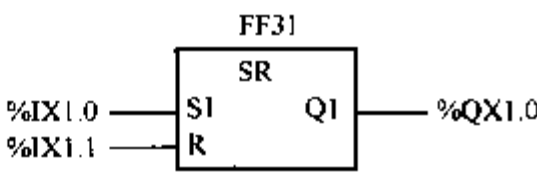
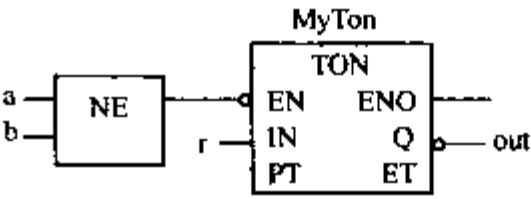
表示形式	示例 1	示例 2
图形 (FBD 语言)		
文本 (ST 语言)	<pre>VAR FF31:SR ;(* FF31 是 SR 功能块实例名 *) END_VAR FF31 (S1:= %IX1.0, R:= %IX1.1);(* 调用 FF31 *) %QX1.0:= FF31.Q1;(* 赋值输出 *)</pre>	<pre>VAR a,b,r,out:BOOL;(* 内部变量声明 *) MyTon:TON ;(* MyTon 是 TON 功能块实例名 *) END_VAR MyTon (EN:= NOT (a <> b),IN:= r,NOT Q => out); (* 调用 MyTon *)</pre>

表 2-41 是功能块声明和用法特性。功能块实例名作为输入的图形用法见图 2-17。

表 2-41 功能块声明和用法特性

序 号	描 述	示 例
1a	内部变量的限定符 RETAIN	VAR RETAIN X;REAL;END_VAR
1b	内部变量的限定符 NON_RETAIN	VAR NON_RETAIN X;REAL;END_VAR
2a	输出变量的限定符 RETAIN	VAR_OUTPUT RETAIN X;REAL;END_VAR
2b	输入变量的限定符 RETAIN	VAR_INPUT RETAIN X;REAL;END_VAR
2c	输出变量的限定符 NON_RETAIN	VAR_OUTPUT NON_RETAIN X;REAL;END_VAR
2d	输入变量的限定符 NON_RETAIN	VAR_INPUT NON_RETAIN X;REAL;END_VAR
3a	内部功能块的限定符 RETAIN	VAR RETAIN TMR1;TON;END_VAR
3b	内部功能块的限定符 NON_RETAIN	VAR NON_RETAIN TMR1;TON;END_VAR
4a	VAR_IN_OUT 声明(文本)	VAR_IN_OUT A;INT;END_VAR
4b	VAR_IN_OUT 声明和用法(图形)	见图 2-18a~d
4c	VAR_IN_OUT 声明带赋值到不同变量(图形)	见图 2-18d
5a	功能块实例名作为输入(文本)	VAR_INPUT I_TMR;TON;END_VAR EXPIRED:= I_TMR.Q;(见注1)
5b	功能块实例名作为输入(图形)	见图 2-17a
6a	功能块实例名作为 VAR_IN_OUT (文本)	VAR_IN_OUT IO_TMR;TOF;END_VAR IO_TMR(IN:= A_VAR, PT:= T#10S); EXPIRED:= IO_TMR.Q;(见注1)
6b	功能块实例名作为 VAR_IN_OUT(图形)	见图 2-17b
7a	功能块实例名作为外部变量(文本)	VAR_EXTERNAL EX_TMR;TOF;END_VAR EX_TMR(IN:= A_VAR, PT:= T#10S); EXPIRED:= EX_TMR.Q;(见注1)
7b	功能块实例名作为外部变量(图形)	见图 2-17c
8a 8b	上升沿和下降沿输入的声明(文本) 也可用上升沿和下降沿边沿检测功能块 见注 2	FUNCTION_BLOCK AND_EDGE (* 见注 2 *) VAR_INPUT X;BOOL R_EDGE; Y;BOOL F_EDGE; END_VAR VAR_OUTPUT Z;BOOL; END_VAR Z:= X AND Y;(* ST 语言的示例 *) END_FUNCTION_BLOCK
9a 9b	上升沿和下降沿输入的声明(图形) (文字形式见注 2)	FUNCTION_BLOCK (* 见注 2 *) (* 外部接口 *)  (* 功能块本体 *)  (* FBD 语言示例 *) END_FUNCTION_BLOCK
10a	在功能块类型声明内的 VAR_EXTERNAL 声明	

(续)

序 号	描 述	示 例
10b	在功能块类型声明内的 VAR_EXTERNAL CONSTANT 声明	
11	在功能块类型声明内的 VAR_TEMP 声明	

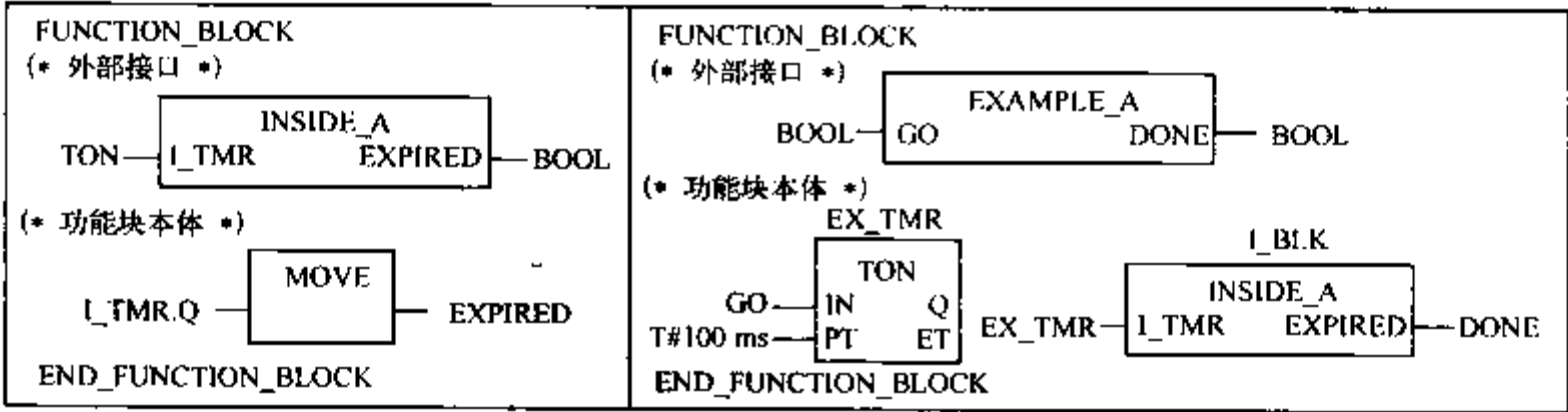
注:① 示例中假设变量 EXPIRED 和 A_VAR 已经声明是 BOOL 类型。
② 示例中功能块 AND_EDGE 声明等效于下列语句。见本节关于功能块 R_TRIG 和 F_TRIG 边沿检测的声明。

```
FUNCTION_BLOCK AND_EDGE
VAR_INPUT
    X,Y:BOOL;
END_VAR
VAR_OUTPUT
    Z:BOOL;
END_VAR
VAR
    X_TRIG:R_TRIG ;(* X_TRIG 是上升沿边沿检测功能块的实例名 *)
    Y_TRIG:F_TRIG ;(* Y_TRIG 是下降沿边沿检测功能块的实例名 *)
END_VAR
X_TRIG ( CLK:= X ) ;
Y_TRIG ( CLK:= Y ) ;
Z:= X_TRIG.Q AND Y_TRIG.Q ;
END_FUNCTION_BLOCK
```

图 2-17a 中,I_TMR 未采用图形表示,因为它是在 INSIDE_A 功能块内部的 I_TMR 隐含调用,根据编写功能块表示时的注意事项,它是被禁止的。

图 2-17b 的 INSIDE_B 功能块中,I_TMR 作为 VAR_IN_OUT。因此,当使用 EXAMPLE_B 功能块时,I_TMR 被作为 INSIDE_B 功能块实例名 I_BLK 的输入输出变量。

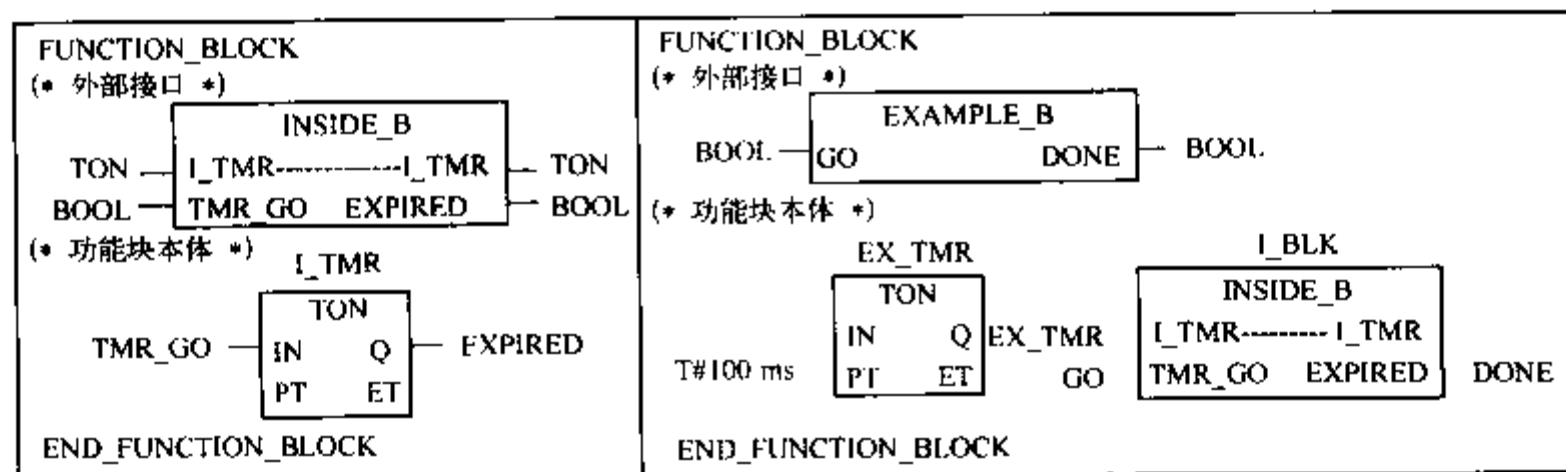
图 2-17c 中,EX_TMR 被声明为外部变量(TON 标准功能块),并在 I_BLK 的调用时,作为外部变量调用。



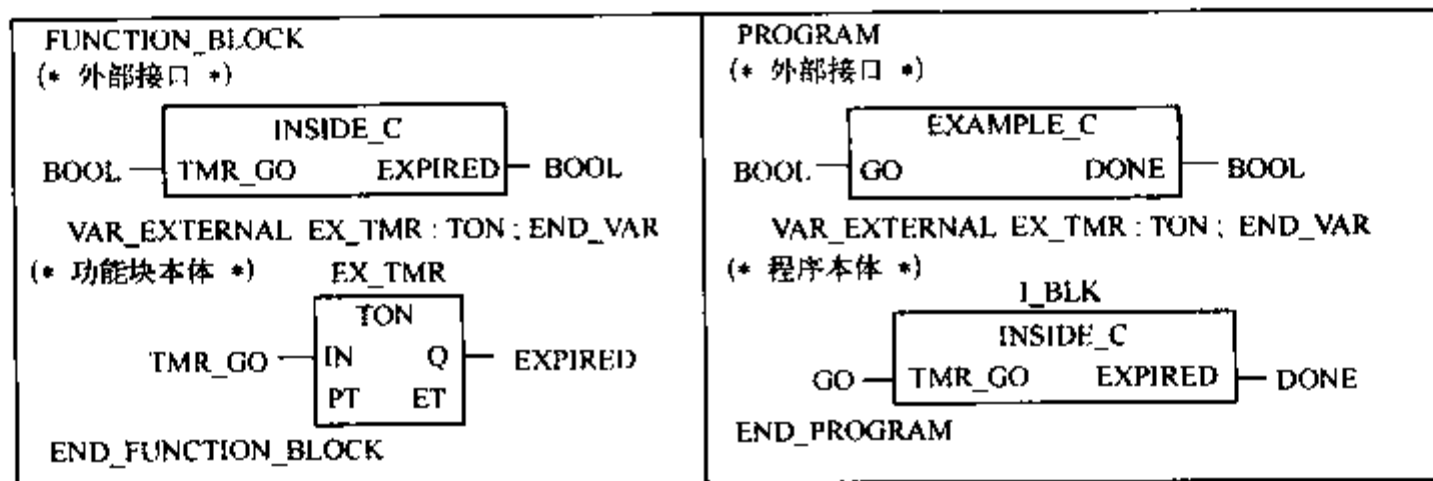
a)

图 2-17 变量的图形用法

a) 功能块名作为输入变量的图形用法(表 2-41 特性 5b) b) 功能块名作为输入输出变量的图形用法(表 2-41 特性 6b) c) 功能块名作为外部变量的图形用法(表 2-41 特性 7b)



b)



c)

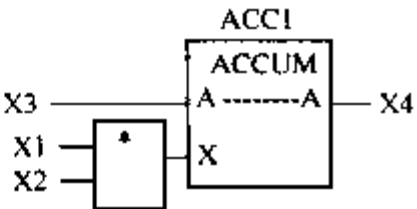
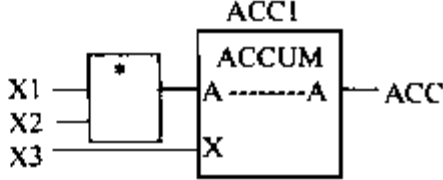
图 2-17(续)

12) 在功能块内部变量的声明中可使用星号,见表 2-16。可使用 EN 和 ENO 附加属性,使用方法与函数的使用方法相同。

13) 只有变量或功能块实例名能够经 VAR_IN_OUT 结构传递进入功能块,见表 2-18。

表 2-42 功能块中输入输出变量的声明和用法

序号	图形形式	文字形式
12a		<pre> FUNCTION_BLOCK ACCUM VAR_IN_OUT A:INT;END_VAR VAR_INPUT X:INT;END_VAR A := A + X; END_FUNCTION_BLOCK </pre>
12b		<p>变量声明如下:</p> <pre> VAR X1,X2,ACC:INT;END_VAR </pre> <p>执行结果为:</p> <pre> ACC := ACC + X1 * X2; </pre>
12c		<p>变量声明在 12b 中,即</p> <pre> VAR X1,X2,X3,X4,ACC:INT;END_VAR </pre> <p>执行结果为:</p> <pre> ACC := ACC + X1 * X2 + X3 * X4; </pre>

序号	图形形式	文字形式
12d		变量声明如下: VAR X1,X2,X3,X4:INT;END_VAR 执行结果为: X3:=X3+X1*X2; X4:=X3;
12e		不符合规定的用法 连接到输入输出 A 的不是变量或功能块名

14) 在文本声明中,限定字 R_EDGE 和 F_EDGE 能用于表示布尔输入的边沿检测。第 2 版增加了边沿检测功能块,因此,也可用边沿检测功能块对输入变量进行边沿检测。当使用 GB1988 字符集(与 IEC 60617-2 等效)时,大于(>)和小于(<)字符应与功能块的边沿对齐。当采用图形或半图形表示时,应使用用于动态输入的 GB4728. 12(与 IEC 60617-2 等效)约定的符号。

15) 使用 VAR_INPUT 和 VAR_OUTPUT 会造成过多的存储区开销,为此,在功能块编程时,可使用 VAR_IN_OUT 替代,减少对存储区的需要。

16) 功能块实例的输入参数能够保持其值,直到下一次调用。功能块实例的输出变量也能够保持其值,直到下一次调用。

17) 直接表示变量不能作为局部变量声明,只能用 VAR_EXTERNAL 变量声明为全局变量,从外部引入。

18) 功能块实例名经 VAR_INPUT、VAR_IN_OUT 或 VAR_EXTERNAL 结构传递的,其功能块实例输出值可以存取,但不能从该功能块内部修改。

19) 经 VAR_IN_OUT 或 VAR_EXTERNAL 结构传递功能块实例名,能够在功能块内调用,即函数或功能块输出不能用这种结构传递。因此,它能防止这些输出被无意中修改。VAR_IN_OUT 结构的串级是允许的。

20) 功能块具有 EN 和 ENO 的附加属性。EN 和 ENO 可在功能块中使用任一个、两个都使用或都不使用。附加的 EN 和 ENO 属性用于控制执行过程的进行与否。使能输入和使能输出的应用原则与函数的使能输入和使能输出的应用原则相同,不多述。

2. 标准功能块

标准功能块是由 IEC 61131-3 标准规定的功能块。标准功能块具有与函数一样的调用属性和 EN、ENO 属性等。

(1) 双稳元素功能块

双稳元素(Bitstable Element)功能块有两个稳态,根据两个输入变量都为 1 时输出稳态值的不同,可分为置位优先(SR)和复位优先(RS)两类。表 2-43 给出了双稳元素功能块的图形形式、文字形式和功能块本体结构。图 2-18 给出了 SR 和 RS 功能块的信号波形。

表 2-43 双稳元素功能块的图形形式、文字形式和功能块本体结构

功能块的图形形式	功能块本体	功能块的文字形式
置位优先的双稳功能块		
		<pre>FUNCTION_BLOCK SR VAR_INPUT S1,R;BOOL;END_VAR; VAR_OUTPUT Q1;BOOL;END_VAR; Q1:=S1 OR (NOT R AND Q1); END_FUNCTION_BLOCK</pre>
复位优先的双稳功能块		
		<pre>FUNCTION_BLOCK RS VAR_INPUT S,R1;BOOL;END_VAR; VAR_OUTPUT Q1;BOOL;END_VAR; Q1:= NOT R1 AND (S OR Q1); END_FUNCTION_BLOCK</pre>

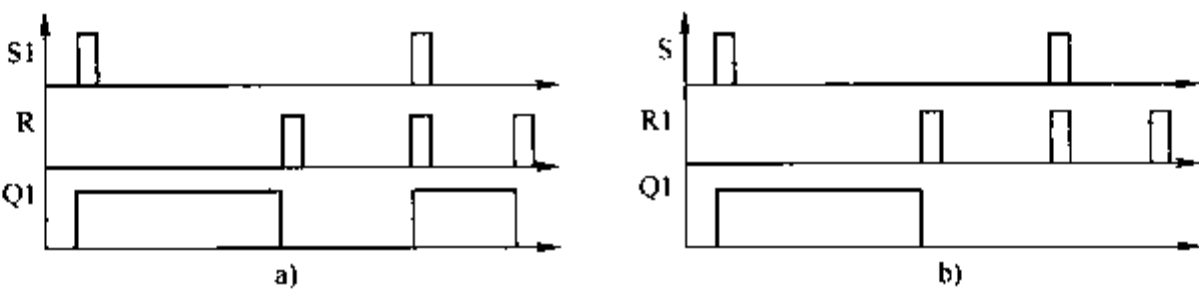


图 2-18 功能块的信号波形
a) SR 功能块 b) RS 功能块

使用双稳元素功能块时应注意下列事项：

- 1) 双稳元素功能块有两个输入变量,S 是置位端,R 是复位端。双稳元素功能块类型名中,S 在前表示置位优先,即 SR 表示置位优先双稳元素功能块。R 在前表示复位优先,即 RS 表示复位优先双稳元素功能块。同样,输入变量名中,添加 1 表示该变量是优先的。因此,S1 表示置位优先的 S 端,R1 表示复位优先的 R 端。
- 2) 双稳元素功能块的表示符号应仅可能选成与 IEC 60617-12 的符号 12-09-01 和 12-09-02 一致。
- 3) 当两个输入变量都为 1 时,双稳元素功能块的输出 Q1 为 1,称为置位优先。当两个输入变量都为 1 时,双稳元素功能块的输出 Q1 为 0,称为复位优先。双稳元素功能块具有记忆功能。例如,对 RS 功能块,当输入变量 S 从 1 变到 0 后,输出能够对功能块的状态记忆,其输出 Q1 变到 1,并能够保持到输入变量 R1 为 1,一旦 R1 从 1 变回到 0,输出 Q1 仍能够保持其输出为 0。这是双稳元素功能块具有的记忆功能。
- 4) 双稳元素功能块有输出变量。从功能块框图可见,功能块输出变量为 Q1,而函数的输出线虽然与函数的框有连接,但没有连接到输出变量。
- 5) 不同编程软件中,S 也写为 SET,R 也写为 RESET,应根据产品说明书规定。

(2) 边沿检测功能块

边沿检测(Edge Detection)功能块用于对输入信号的上升沿和下降沿进行检测。分为上升沿边沿检测(R_TRIG)功能块和下降沿边沿检测(F_TRIG)功能块两类。传统可编程控制器

中,称为上微分和下微分指令。表 2-44 给出了边沿检测功能块的图形形式、文字形式和功能块体。

表 2-44 边沿检测功能块的图形形式、文字形式和功能块体

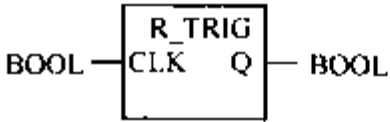
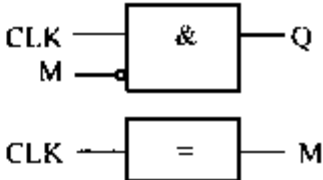
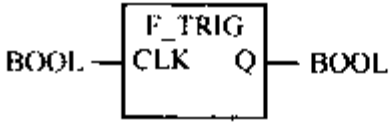
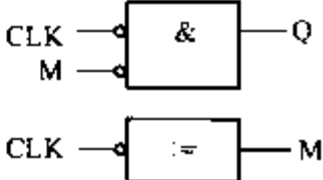
功能块的图形形式	功能块本体	功能块的文字形式
上升沿边沿检测功能块 R_TRIG		
		<pre>FUNCTION_BLOCK R_TRIG VAR_INPUT CLK:BOOL;END_VAR; VAR_OUTPUT Q:BOOL;END_VAR; VAR M:BOOL;END_VAR; Q:=CLK AND NOT M ; M:=CLK; END_FUNCTION_BLOCK</pre>
下降沿边沿检测功能块 F_TRIG		
		<pre>FUNCTION_BLOCK F_TRIG VAR_INPUT CLK:BOOL;END_VAR; VAR_OUTPUT Q:BOOL;END_VAR; VAR M:BOOL;END_VAR; Q:=NOT CLK AND NOT M; M:=NOT CLK ; END_FUNCTION_BLOCK</pre>

图 2-19 显示边沿检测功能块的输出是宽度为一个扫描周期的脉冲信号。

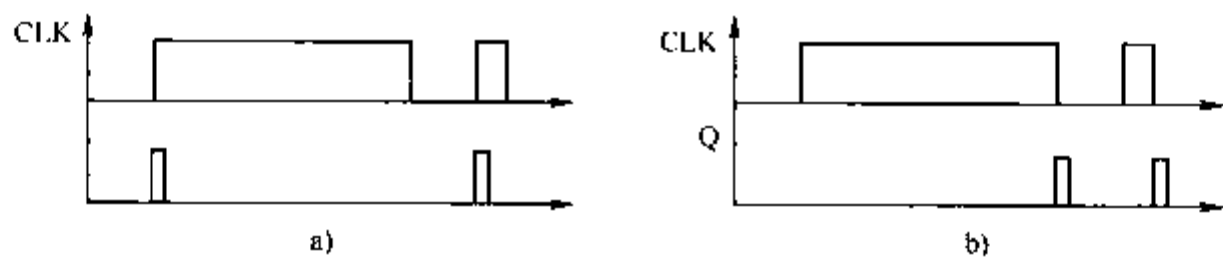


图 2-19 功能块信号波形
a) R_TRIG 功能块 b) F_TRIG 功能块

使用边沿检测功能块时应注意下列事项：

- 1) 边沿检测功能块对输入信号的变化灵敏,因此,用于检测输入信号的跳变。
- 2) 一个上升沿边沿检测 R_TRIG 功能块的输出 Q(见图 2-20a)应随 CLK 输入从 0 到 1 的转换,在功能块的这次执行到下一次执行期间保持布尔值为 1,在下次执行时返回到 0。
- 3) 一个下降沿边沿检测 F_TRIG 功能块的 Q 输出(见图 2-20b)应随 CLK 输入从 1 到 0 的转换,在功能块的这次执行到下一次执行期间保持布尔值为 1,在下次执行时返回到 0。
- 4) 在冷启动的第一次执行时,如果 R_TRIG 实例的 CLK 输入为 1 或 F_TRIG 实例的 CLK 输入为 0 时,则对输入 CLK 的边沿进行检测。
- 5) 边沿检测功能块也可用设置变量的边沿检测属性实现。见表 2-41 的特性。

(3) 计数器功能块

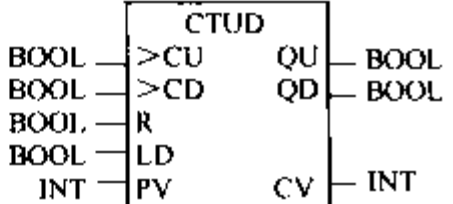
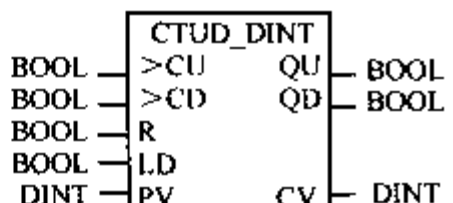
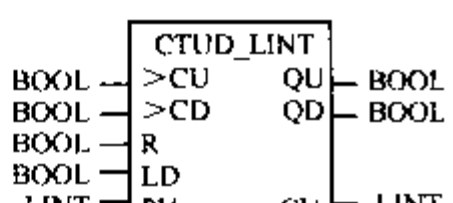
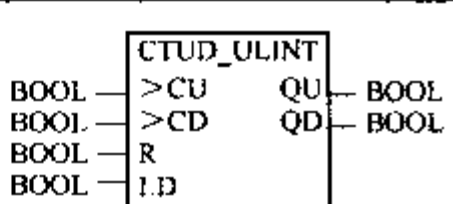
计数器(Counter)功能块有 3 种基本功能块,它们是加计数器、减计数器和加减计数器。用于计数的变量 PV 是整数数据类型。根据不同的整数类型数据,可将计数器功能块类型化。例如,用于双整数的计数器功能块是 CTU_DINT 等。表 2-45 给出了标准规定的各种过载属

性的计数器功能块,它的最大计数设定值根据所选整数数据类型的允许范围确定。

表 2 -45 基本计数器功能块的图形形式和文字形式

序号	功能块类型	功能块的图形形式	功能块的文字形式
加计数器 CTU			
1a	标准型		FUNCTION_BLOCK CTU VAR_INPUT CU:BOOL R_TRIG ; R:BOOL ;PV:INT ;(* 根据过载属性改变 *) END_VAR VAR_OUTPUT Q:BOOL; CV:INT ;(* 根据过载属性改变 *) END_VAR
1b	双整数型		END_VAR VAR_OUTPUT Q:BOOL; CV:INT ;(* 根据过载属性改变 *) END_VAR
1c	长整数型		END_VAR VAR Pvmax:INT ;(* 根据过载属性改变 *) END_VAR IF R THEN CV: = 0 ; ELSEIF CU AND (CV < Pvmax) THEN CV: = CV + 1 ; END_IF ; Q: = (CV >= PV) ; END_FUNCTION_BLOCK
1d	无符号双整数型		END_FUNCTION_BLOCK
1e	无符号长整数型		根据不同过载属性,改变 PV、CV 和 Pvmax 的数据类型
减计数器 CTD			
2a	标准型		FUNCTION_BLOCK CTD VAR_INPUT CD:BOOL R_TRIG ; LD:BOOL ;PV:INT ;(* 根据过载属性改变 *) END_VAR VAR_OUTPUT Q:BOOL; CV:INT ;(* 根据过载属性改变 *) END_VAR
2b	双整数型		END_VAR VAR_OUTPUT Q:BOOL; CV:INT ;(* 根据过载属性改变 *) END_VAR
2c	长整数型		END_VAR VAR Pvmin:INT ;(* 根据过载属性改变 *) END_VAR IF LD THEN CV: = PV ; ELSEIF CD AND (CV > Pvmin) THEN CV: = CV - 1 ; END_IF ; Q: = (CV <= 0) ; END_FUNCTION_BLOCK
2d	无符号双整数型		END_FUNCTION_BLOCK
2e	无符号长整数型		根据不同过载属性,改变 PV、CV 和 Pvmin 的数据类型

(续)

序号	功能块类型	功能块的图形形式	功能块的文字形式
加减计数器 CTUD			
3a	标准型		<pre>FUNCTION_BLOCK CTUD VAR_INPUT CU:BOOL R_TRIG ; CD:BOOL R_TRIG ; R,LD:BOOL ;PV:INT ;(* 根据过载属性改变 *) END_VAR VAR_OUTPUT QU,QD:BOOL; CV:INT ;(* 根据过载属性改变 *) END_VAR IF R THEN CV:=0;ELSIF LD THEN CV:=PV; ELSE IF NOT (CU AND CD) THEN IF CU AND (CV < PV max) THEN CV:=CV+1; ELSIF CD AND (CV > Pmin) THEN CV:=CV-1; END_IF; END_IF; END_IF; QU:=(CV >= PV); QD:=(CV <= 0); END_FUNCTION_BLOCK 根据不同过载属性,改变 PV、CV 和 Pmin、Pmax 的数据类型</pre>
3b	双整数型		<pre>FUNCTION_BLOCK CTUD_DINT VAR_INPUT CU:BOOL R_TRIG ; CD:BOOL R_TRIG ; R,LD:BOOL ;PV:DINT ;(* 根据过载属性改变 *) END_VAR VAR_OUTPUT QU,QD:BOOL; CV:DINT ;(* 根据过载属性改变 *) END_VAR IF R THEN CV:=0;ELSIF LD THEN CV:=PV; ELSE IF NOT (CU AND CD) THEN IF CU AND (CV < PV max) THEN CV:=CV+1; ELSIF CD AND (CV > Pmin) THEN CV:=CV-1; END_IF; END_IF; END_IF; QU:=(CV >= PV); QD:=(CV <= 0); END_FUNCTION_BLOCK 根据不同过载属性,改变 PV、CV 和 Pmin、Pmax 的数据类型</pre>
3c	长整数型		<pre>FUNCTION_BLOCK CTUD_LINT VAR_INPUT CU:BOOL R_TRIG ; CD:BOOL R_TRIG ; R,LD:BOOL ;PV:LINT ;(* 根据过载属性改变 *) END_VAR VAR_OUTPUT QU,QD:BOOL; CV:LINT ;(* 根据过载属性改变 *) END_VAR IF R THEN CV:=0;ELSIF LD THEN CV:=PV; ELSE IF NOT (CU AND CD) THEN IF CU AND (CV < PV max) THEN CV:=CV+1; ELSIF CD AND (CV > Pmin) THEN CV:=CV-1; END_IF; END_IF; END_IF; QU:=(CV >= PV); QD:=(CV <= 0); END_FUNCTION_BLOCK 根据不同过载属性,改变 PV、CV 和 Pmin、Pmax 的数据类型</pre>
3d	无符号长整数型		<pre>FUNCTION_BLOCK CTUD_ULINT VAR_INPUT CU:BOOL R_TRIG ; CD:BOOL R_TRIG ; R,LD:BOOL ;PV:ULINT ;(* 根据过载属性改变 *) END_VAR VAR_OUTPUT QU,QD:BOOL; CV:ULINT ;(* 根据过载属性改变 *) END_VAR IF R THEN CV:=0;ELSIF LD THEN CV:=PV; ELSE IF NOT (CU AND CD) THEN IF CU AND (CV < PV max) THEN CV:=CV+1; ELSIF CD AND (CV > Pmin) THEN CV:=CV-1; END_IF; END_IF; END_IF; QU:=(CV >= PV); QD:=(CV <= 0); END_FUNCTION_BLOCK 根据不同过载属性,改变 PV、CV 和 Pmin、Pmax 的数据类型</pre>

注:限定变量 PVmax 和 PVmin 的数字值是与执行有关的参数。

1) 加计数器有 3 个输入变量,两个输出变量。输入变量 CU 是上升沿触发的计数脉冲,它是边沿触发的脉冲信号,在图形形式表示中,可在矩形框该形式参数 CU 旁用“>”表示是上升沿触发脉冲信号。计数器的复位输入 R 用于将加计数器的计数值恢复到零。计数器的计数设定值由输入变量 PV 送入。每次计数脉冲上升沿,加计数器将计数值加 1,当计数值 CV 大于等于设定值 PV 时,计数器输出 Q 被置 1。计数器当前计数值由 CV 输出。信号波形见图 2-20a。

2) 减计数器的工作原理与加计数器类似,3 个输入变量中,输入变量 CD 是上升沿触发的计数脉冲信号,计数器的设定值由 PV 输入变量送入。复位输入 LD 用于将减计数器的当前计数值恢复到计数设定值 PV。每次计数脉冲上升沿,减计数器将当前计数值 CV 减 1,当计数值小于等于零时,减计数器的输出 Q 被置 1。减计数器的当前计数值由 CV 输出。信号波形见图 2-20b。

3) 加减计数器有 5 个输入变量和 3 个输出变量。输入变量 CU 是加计数的脉冲信号,输入变量 CD 是减计数的脉冲信号。复位变量 R 用于将加减计数器当前计数值 CV 置为 0,复位变量 LD 用于将加减计数器当前计数值 CV 置为 PV。每次 CU 计数脉冲的上升沿使 CV 加 1,每次 CD 计数脉冲的上升沿使 CV 减 1。当计数值 CV 大于等于 PV,则输出变量 QU 被置 1,如果计数值 CV 小于等于零,则输出变量 QD 被置 1。信号波形见图 2-20c。

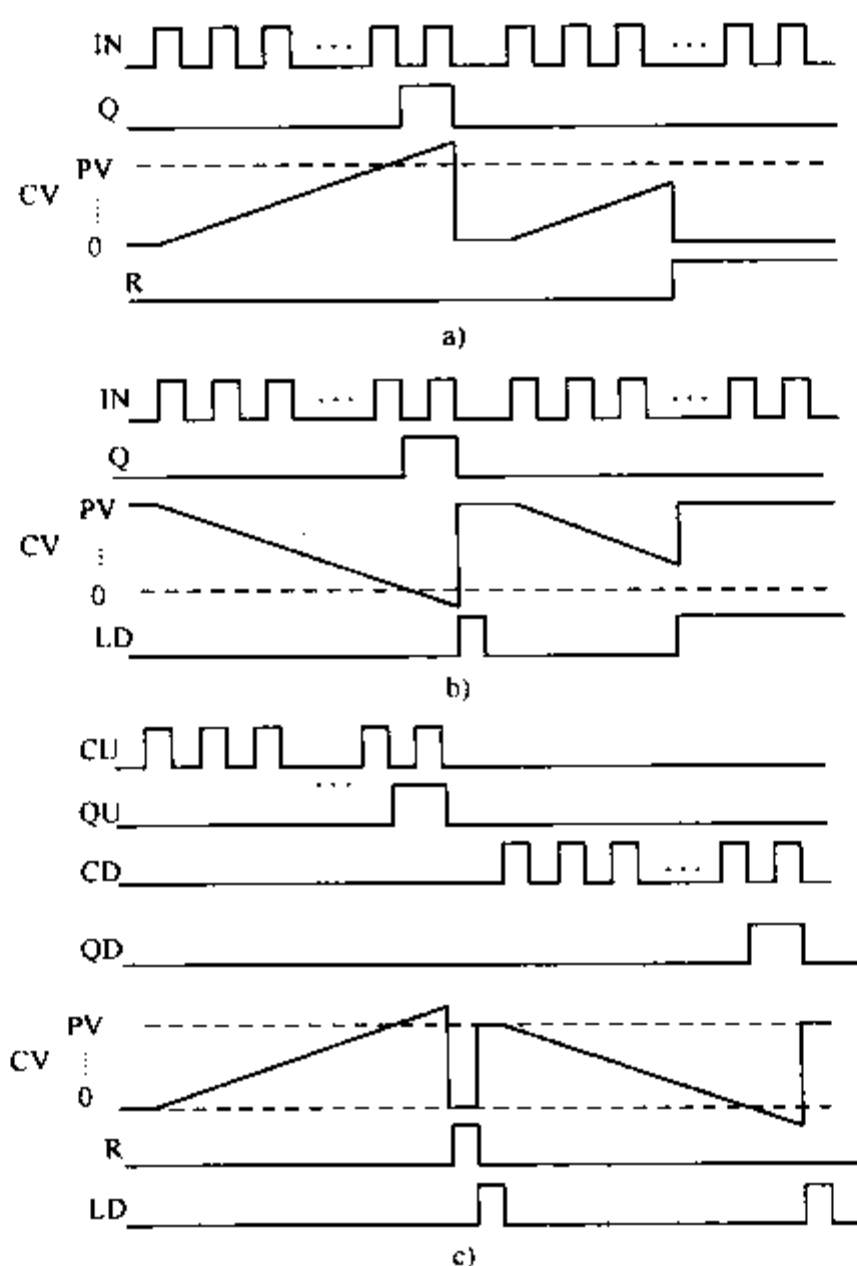


图 2-20 计数器输入输出信号波形

a) 加计数器的输入输出信号波形 b) 减计数器输入输出信号波形 c) 加减计数器输入输出信号波形

使用计数器功能块时应注意下列事项：

1) 计数器功能块的过载属性是对标准型计数器计数设定值 PV 数据类型的扩展。当 PV 变量数据类型是 LINT, 同时, 当前计数值的数据类型是 LINT 时, 该计数器扩展成为长整数 LINT, 相应的计数器功能块应在原计数器功能块名后添加“_LDINT”, 余类推。计数器功能块的过载属性使计数器计数范围大大扩展, 也简化了程序。标准型计数器没有类型化的后缀, 它也表示具有过载属性。

2) 计数器触发的脉冲信号都是上升沿触发脉冲。上升沿触发脉冲信号变量的附加属性是 R_EDGE, 在变量声明段应予声明。图形符号表示时, 上升沿触发脉冲信号用“>”号表示。

【例 2-63】 标准型加计数器输入变量的声明。

```
VAR_INPUT
    CU: BOOL R_EDGE;
    R: BOOL;
    PV: INT;
END_VAR;
```


3) 加计数器、减计数器和加减计数器的当前计数值都是输出变量 CV,它是阶梯变化的整数数值。设定的计数值都从 PV 输入。

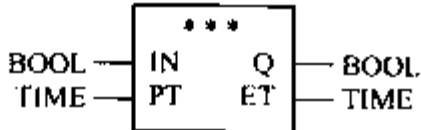
4) 计数器功能块的功能块体中, PV_{max} 和 PV_{min} 的值是与执行有关的限定值。实际应用时,通常, PV_{max} 大于设定值 PV, PV_{min} 小于 0。

(4) 定时器功能块

定时器(Timer)功能块用定时器实现接通延迟、断开延迟和定时脉冲。表 2-46 显示了 3 类定时器的图形形式和描述。

1) 接通延迟定时器。用 TON 或 T--0 代替表 2-46 中 *** 的定时器组成接通延迟定时器。接通延迟定时器输入输出信号的时序关系见图 2-21。ET 是当前的计时时间。

表 2-46 定时器的文本形式和图形形式

序 号	名 称	表 示	图 形 形 式
1	脉冲定时器	*** 用 TP 表示	
2a 2b	接通延迟定时器	*** 用 TON 表示 *** 用 T--0 表示	
3a 3b	断开延迟定时器	*** 用 TOF 表示 *** 用 0---T 表示	

注:1. 2b 和 3b 的描述在文本类编程语言中不能使用。

2. 计时过程中改变 PT 值的结果,例如,TP 实例中改变 PT 值到零进行复位的操作是与执行有关的参数。

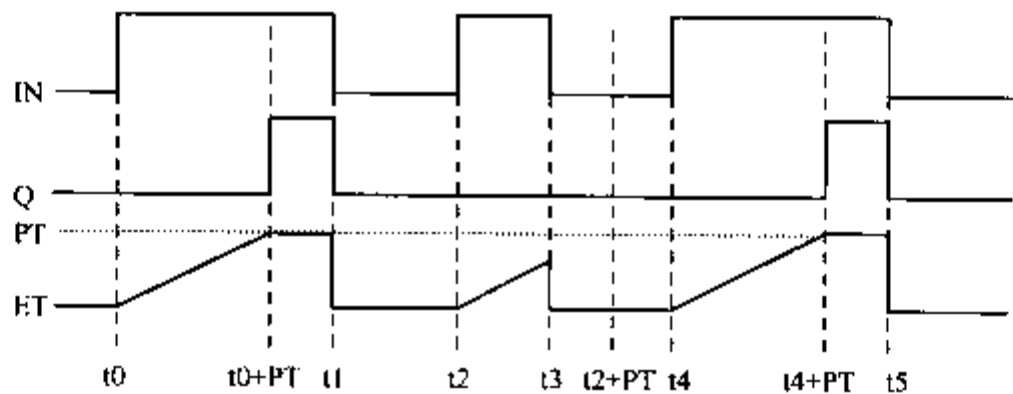


图 2-21 接通延迟定时器的时序图

接通延迟定时器的工作过程:当输入变量 IN 为 1,定时器开始计时,当前计时值 ET 作为定时器功能块的输出。当当前计时时间 ET 等于由输入变量 PT 输入的计时设定值时,定时器功能块才有输出 Q 为 1。当 IN 为 0 时,输出 Q 也回到 0。如果在计时过程中(未达到计时设定值 PT)输入 IN 回到 0,则当前计时值 ET 也回到 0。

2) 断开延迟定时器。用 TOF 或 0---T 代替表 2-46 中 *** 的定时器组成断开延迟定时器。断开延迟定时器输入输出信号的时序关系见图 2-22。

断开延迟定时器的工作过程:当输入变量 IN 为 1,定时器输出 Q 立刻为 1,当输入变量 IN 回到 0,定时器开始计时,当前计时值 ET 作为定时器功能块的输出。当计时时间等于由输入变量 PT 输入的计时设定值时,定时器功能块才使输出 Q 为 0。如果在计时过程中(未到计时设定值 PT)输入 IN 变到 1,则当前计时值 ET 回到 0。

须注意,冷启动时,断开延迟定时器在 IN 输入第一个上升沿出现前,ET 等于 PT。

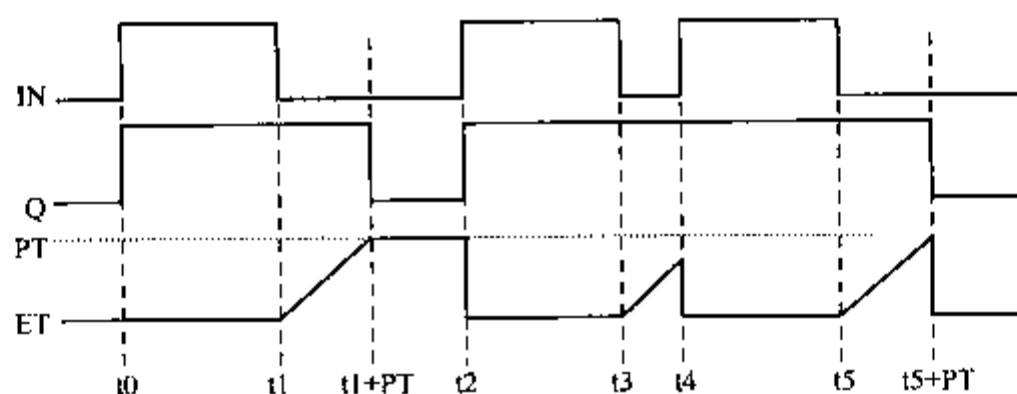


图 2-22 断开延迟定时器的时序图

3) 定时脉冲定时器。用 TP 代替表 2-46 中 *** 的定时器组成定时脉冲定时器。定时脉冲定时(PULSE TIMING)器输入输出信号的时序图见图 2-23。

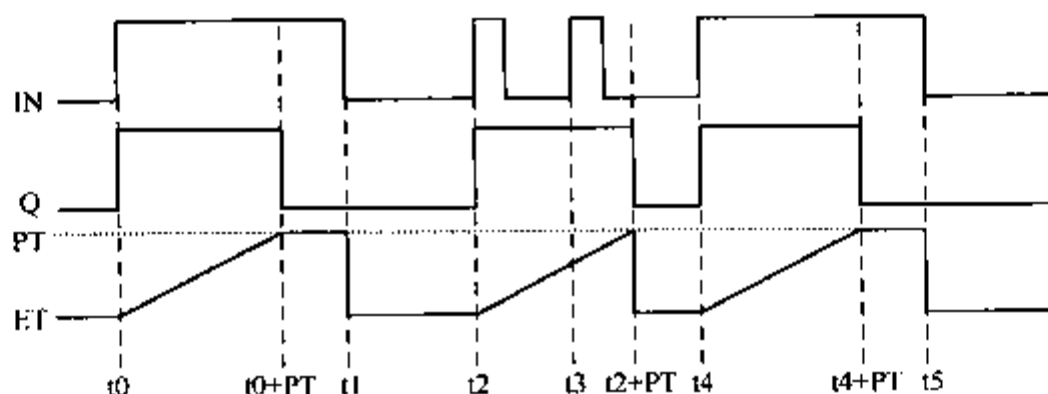


图 2-23 定时脉冲定时器的时序图

定时脉冲定时器的工作过程:当输入 IN 变为 1 时,定时器输出立刻为 1,同时,定时器开始计时,当前计时时间 ET 等于设定时间 PT 时,定时器输出回到 0,即定时器输出 Q 的脉冲宽度等于设定时间 PT。如果,计时过程中输入 IN 回 0,或输入 IN 又为 1,不影响计时过程的进行,即计时过程仍继续(如图中的 t3),直到计时时间到后,输出 Q 才回 0。

使用定时器功能块时注意下列事项。

- 1) 定时器的当前计时值由 ET 输出。
- 2) 定时器计时过程中,改变计时设定值 PT,定时器行为与具体的实现有关。
- 3) 定时器的计时开始时间是输入 IN 的上升沿,但不是脉冲,因此,从功能块图看,IN 前没有“>”符号(上升沿脉冲的符号)。

3. 衍生功能块

衍生功能块是由标准功能块和函数导出的功能块。

【例 2-64】 PID 功能块。

在控制系统中,常采用 PID 控制算法实现控制规律,即 PID 控制器输出为

$$u(k) = K_p \left(e(k) + \frac{1}{T_i} \sum_{i=0}^k e(i) T_s + T_d \frac{e(k) - e(k-1)}{T_s} \right) + u(0) \quad (2-1)$$

式中, e 是偏差; T_s 是采样周期; K_p 是增益; T_i 是积分时间; T_d 是微分时间; u 是控制输出。

图 2-24 给出了 PID 功能块的图形形式和功能块体。

功能块输入变量包括手动和自动开关 AUTO、过程测量信号 PV、设定信号 SP、手动输出 X0、控制器功能块增益 K_P 、积分时间 T_I 、微分时间 T_D 和采样时间 T_S 。其中,积分和微分时间是用实数表示的以毫秒为单位的时间值。采样时间 T_S 采用时间数据类型,应根据被控过程的

时间常数设置。内部变量 ERR 用于计算获得的控制器功能块偏差信号,等于设定 SP 减去测量 PV。内部变量 OUTI 和 OUTD 分别计算偏差积分控制输出和偏差微分控制输出。内部变量 X1 是上一采样时刻的偏差 ERR 信号。用简单差分近似其微分。

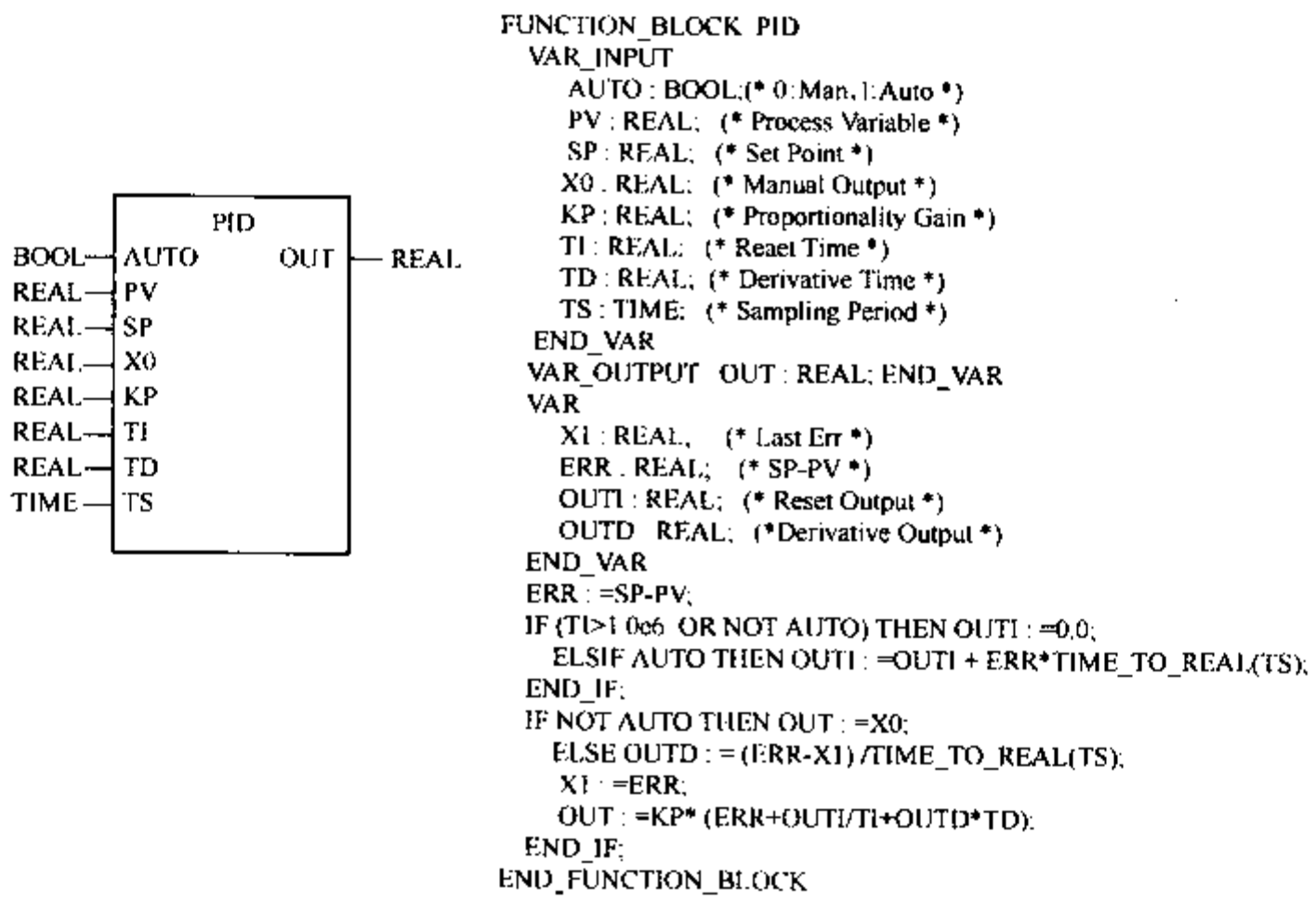


图 2-24 PID 功能块的图形形式和功能块文字形式

功能块体中,当积分时间 TI 的值大于 1.0e6 时,认为没有积分作用,积分输出 OUTI 为 0.0。

功能块输出 OUT 可以是手动输出或 PID 运算的自动输出。

功能块仅表示 PID 控制运算,没有对积分饱和和手自动无扰动切换提供相应手段。可以看到,功能块调用有关函数计算,例如,逻辑比较,选择和类型转换等函数。功能块也可调用其他功能块实例,见下述。

【例 2-65】 COUNTER 功能块。

COUNTER 功能块有一个输入 MODE,它是一个枚举数据类型,有 3 个值:RESET,COUNT 和 HOLD。一个整数输出 OUT。当功能模块在 RESET 模式,输出 OUT 为 0;当在 COUNT 模式,每次执行时 OUT 自动加一;当在 HOLD 模式,OUT 保持最近的值。

先定义枚举数据 Modetype 如下:

```

TYPE
    (* 对 Modetype 枚举数据进行类型化 *)
    Modetype: ( RESET,COUNT,HOLD ); = RESET; (* 初始模式为 RESET *)
END_TYPE
  
```

COUNTER 功能块定义如下:

```

FUNCTION_BLOCK COUNTER
VAR_INPUT
  
```

```

        MODE; ModeType; = RESET;
    END_VAR
    VAR_OUTPUT
        OUT; INT; = 0;
    END_VAR
    IF MODE = RESET THEN OUT; = 0;
        ELSIF MODE = COUNT THEN OUT; = OUT + 1;
    END_IF;
END_FUNCTION_BLOCK

```

下面说明在程序中如何调用 COUNTER：

```

PROGRAM COUNT1
    VAR_INPUT
        InputMode; ModeType ;
    END_VAR
    VAR_OUTPUT
        Max_Count; INT ;
    END_VAR
    VAR
        C1; COUNTER ;
    END_VAR
    C1( MODE; = InputMode );
    Max_Count; = C1. OUT;
END_PROGRAM

```

程序中,C1 是 COUNTER 的实例名。假设功能块实例 C1 和父程序在同一任务中执行。程序执行时,功能块 COUNTER 被调用的情况如表 2 - 47 所示。输出变量值与最近的功能块调用有关。

表 2 - 47 输出变量与功能块调用

变 量	在调用时间之间的数值					
	初始状态	1	2	3	4	5
InputMode	RESET	RESET	COUNT	COUNT	HOLD	HOLD
C1. Mode	RESET	RESET	RESET	COUNT	COUNT	HOLD
C1. OUT	0	0	0	1	2	2
Max_Count	0	0	0	1	2	2

4. 功能块性能

(1) 记忆功能

功能块具有记忆功能,因此,相同输入参数调用时,其输出变量的值取决于其内部变量和外部变量的状态。与函数的不同,它是数据的持续。在两个采样时间间隔内,这些变量能够保持其值。此外,与函数不同,功能块有输出变量,输入输出变量和外部变量等。功能块实例的记忆功能表现在下列两方面：

- 1) 功能块实例的输入参数在下次调用前应保持不变。
- 2) 功能块实例的输出参数在两次调用之间应保持不变。

功能块能够保持在每次执行之间的数值,因此,它被用于提供宽范围的系统用于建立需要保持状态的模块。

功能块的记忆功能表示功能块实例在多次调用过程中能够存储各功能块实例的输入、输出和其内部变量。因此,对功能块而言,记忆功能对一些功能块十分重要,例如,对 RS、SR 双稳触发功能块、计数器功能块等。因为,这些功能块的动作取决于触发器的当前状态和计数器的当前值。

(2) 结构化功能

功能块的实例产生结构化变量。其表现如下:

- 1) 可包含定时器或计数器的实际状态。
- 2) 可像数据结构一样描述功能块的调用接口,即其输入变量、输出变量、输入输出变量等。
- 3) 代表调用功能块的一种方法。

【例 2-66】 功能块结构化变量的示例。

```

TYPE CTU;                                (* 计数器 CTU 的结构化变量声明 *)
STRUCT
    (* 输入部分 *)
    CU :BOOL;                            (* 输入的加计数脉冲 *)
    R  :BOOL;                            (* 复位信号 *)
    PV :INT;                             (* 计数器设定值 *)
    (* 输出部分 *)
    Q  :BOOL;                            (* 计数器输出 *)
    CV :INT;                             (* 计数器当前计数值 *)
END_STRUCT
END_TYPE

```

示例的数据结构显示加计数器 CTU 的形式参数和输出参数,即表示了功能块调用的有关连接关系。

功能块数据结构由编程系统或运行系统自动管理,便于对功能块赋值。

(3) 调用功能

功能块的重要性能是它可以调用。功能块类似于一个子程序,当调用时,将有关的形式参数用实际参数代入,就能获得这些实际参数下功能块的输出。由于功能块的可调用性,用户可对一些程序重复使用,缩短了程序开发时间。例如,上述 PID 功能块就被用于各种被控对象的控制回路,RS 或 SR 功能块被大量用于两位式控制的应用场合等。

功能块调用是用实际参数代替功能块的形式参数。例如,RS 功能块实例名为 FF75,则该功能块调用可表示为:FF75(S:=%IX1.1,R1:=%IX1.2);该语句表示将输入单元 1 的第 1 位信号作为 RS 功能块的置位信号,将输入单元 1 的第 2 位信号作为 RS 功能块的复位信号,并调用 FF75(即 RS 功能块)。

图 2-16 显示的 DEBOUNCE 衍生功能块中,有 DB_ON 和 DB_OFF 两个 TON 功能块实

例, DB_FF 是 SR 功能块实例。在功能块体内用 DB_ON(IN:= IN, PT:= DB_TIME); 的形式进行 TON 功能块实例的调用, 在 DB_FF 调用时, 直接将上述调用结果作为实际参数代入, 即 DB_FF(SI:= DB_ON. Q, R:= DB_OFF. Q); 最后, 用 OUT:= DB_FF. Q1 的形式将功能块 DB_FF 输出 Q1 赋值给 OUT 变量。

功能块调用后, 功能块的输出用功能块实例名、“.” 和功能块输出变量名表示。例如, RS 功能块实例名为 DB_FF, 则其输出用 DB_FF. Q1 表示。TON 功能块实例名为 ION_TMR, 则其输出用 ION_TMR. Q 表示, 其当前计时值用 ION_TMR. ET 表示。

(4) EN 和 ENO 属性

功能块具有 EN 和 ENO 的附加属性。它与函数中 EN 和 ENO 的使用方法类似。

5. 通信功能块

可编程控制器的通信功能块由 IEC 61131-5 标准规定。它提供可编程的通信功能, 诸如设备确认、查询数据的获得、编程数据获得、参数控制、联锁控制、编程报警报告及连接管理和保护等。详细内容见有关资料。

2.6.3 程序

在 IEC 61131-3 标准中, 程序是最大的程序组织单元, 它能在资源层被声明。在概念上, 一个程序非常类似于功能模块, 它提供大的可重复使用的软件组件。

1. 程序的表示

程序(Program) 被定义为“所有可编程语言元素和结构的逻辑组合, 它对于加强采用可编程控制器系统进行过程或机械控制所需的信号处理是必要的”。程序通常作为大型软件建立的模块, 被用于控制一个工厂的主要项目, 例如, 蒸汽透平, 反应器或产品单元等。

函数和功能块用于构成用户子程序, 程序用于构成用户主程序, 因此, 程序被认为是全局的。程序是程序组织单元的最大形式。

程序可以在资源内声明。从概念看, 程序很像功能块, 它提供大量可重复使用的软件组件。程序由程序类型定义, 与功能块类似, 它由输入变量、输出变量和内部变量等的声明段和程序的本体组成。

程序图形形式与函数或功能块图形形式类似。图 2-25 是一个程序的图形形式示例。

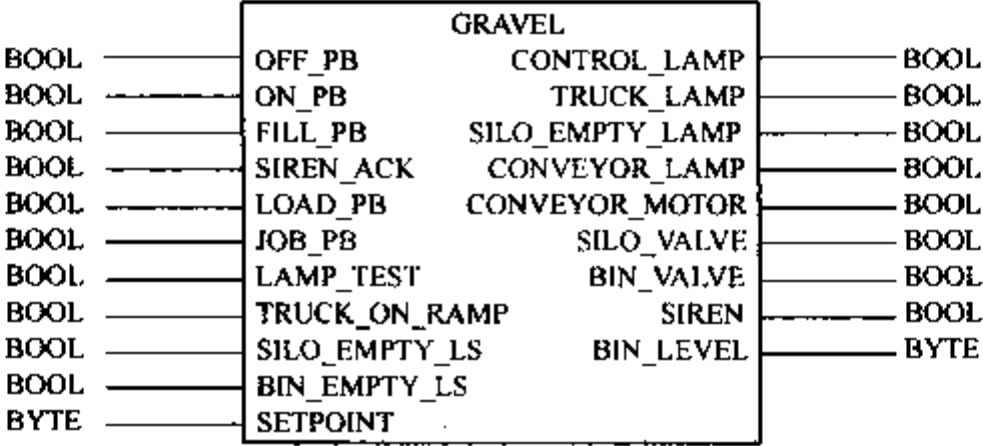
图 2-25 的上部用图形形式显示 GRAVEL 程序的有关外部连接关系, 例如, 输入变量和输出变量等。图的下部用 SFC 编程语言描述了该程序的顺序执行关系。

程序文字表示的格式如下:

```
PROGRAM      程序名
  程序变量声明
  程序本体
END_PROGRAM
```

程序变量声明包括在整个程序中所使用变量的声明。程序声明中的变量类型比函数或功能块更丰富。程序中允许使用第 2.5 节所介绍的各种变量类型。

PROGRAM GRAVEL (* 程序名 GRAVEL *)



(* 程序本体 *)

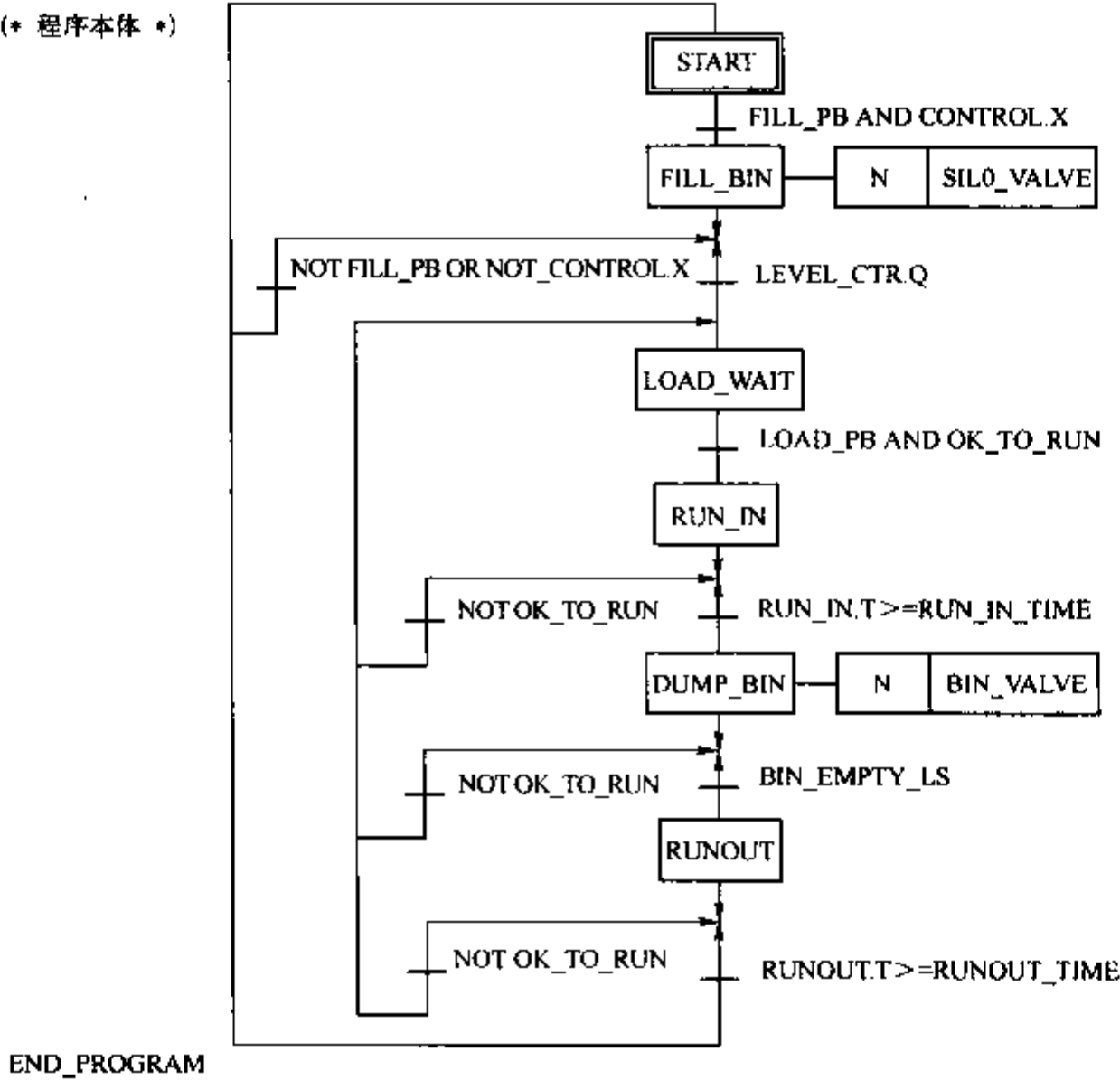


图 2-25 程序的图形形式示例

【例 2-67】 程序实例的声明。

```
PROGRAM LINE1 ;FERMENTER ( Reagent_Code := A1, Sterilise := A2,
    Ferment_Period := FTIME, Yield => AJ_43, Status => KX56 )
```

程序 FERMENTER 的实例名 LINE1,使用资源变量或全局变量 A1、A2 和 FTIME,它们是由程序输入 Reagent_Code、Sterilise、Ferment_Period 对应提供的。程序的输出有 Yield 和 Status,它们被写入 AJ_43 和 KX56。

【例 2-68】 程序变量声明的示例。

```

PROGRAM FERMENTER          (* 发酵 FERMENTER 程序 *)
  VAR_INPUT                 (* 输入变量声明段 *)
    Reagent_Code :INT ;
    Sterilise :BOOL ;
    Ferment_Period :TIME;
  END_VAR
  VAR_OUTPUT                (* 输出变量声明段 *)
    Yield :REAL ;
    Status :WORD ;
  END_VAR
  VAR                       (* 内部变量声明段 *)
    pH_Loop, Temp_Loop :PID; (* 两个 PID 功能块的实例名 *)
    Phase :INT := 1;
  END_VAR
  (* 程序本体 *) (* 用 ST、FBD、LD、SFC 或 IL 编程语言编写 *)
END_PROGRAM

```

2. 程序的性能

除了具有功能块的性能外,程序还具有下列性能:

1) 可对 VAR_ACCESS 和 VAR_GLOBAL 变量进行声明及存取。程序可包含一个 VAR_ACCESS...END_VAR 结构。它提供特定已命名变量的方法,能够经 IEC 61131-5 规定的通信方法由远程设备来存取。存取路径与程序内的每个输入、输出或内部变量有关。声明的形式和用法在 IEC 61131-5 中描述。

2) 可对 VAR_GLOBAL 和 VAR_EXTERNAL 变量添加 CONSTANT 属性,对这些变量进行常数的限定。

3) 可对 VAR_TEMP 变量进行声明和存取。

4) 一个程序可包含地址的配置。允许声明存取 PLC 物理地址的直接表示变量,直接表示的地址配置仅用于程序中内部变量的声明。直接表示变量允许分级寻址方法描述。

5) 程序不能由其他程序组织单元显式调用。但程序与配置中的一个任务可以结合,使程序实例化,形成运行期程序,并可由资源调用。

6) 程序不能包含其他程序的实例,即程序不能嵌套。

7) 程序仅在资源中实例化。在资源内被声明。程序的实例只需将程序与一个任务结合。而功能块仅能在程序或其他功能块中实例化。

8) 没有与任务结合的程序具有最低的优先级。功能块没有与任务结合时,在其对应的程序中同样的任务中被执行。

9) 程序包含功能模块实例,这些实例可在不同任务下被可选择地执行。

10) 程序可以调用函数和功能块,功能块可以调用函数和功能块,函数只能调用函数。

11) 须注意,IEC 61131-3 标准明确规定程序组织单元不能直接或间接调用其自身,即程序组织单元不能调用由相同类型和/或相同名称的程序组织单元实例。这样做可以保护程序,防止造成程序的出错。但在一般计算机编程语言中,这种递归调用是允许的。

12) 有必要设置程序,以便运行在特定任务的控制下。关键字 WITH 用于设置一个任务

到程序实例或到一个在程序内设计的功能模块实例。

【例 2-69】 程序实例与任务的结合。

```
PROGRAM LINE2 WITH Slow_task :Packaging_Line(  
    Speed:= %IW21, Product_Rate => %QW33  
    Sampler1 WITH Fast_Task,  
    Sealer1 WITH Fast_Task );
```

实例中, Packaging_Line 是程序 LINE2 类型的实例名,它运行在 Slow_task 的任务控制下。由来自直接表示变量的值%IW21 提供程序的输入变量 Speed,程序输出 Product_Rate 被写到一定的 PLC 存储位置%QW33,在程序中被声明的功能块实例 Sampler1 和 Sealer1 被定义运行在 Fast_Task 任务控制下(与该任务结合)。

I3) 操作符 => 在 ST 语句和功能模块调用时,表示赋值,即表示在该操作符左侧的输出变量值被赋值到该操作符右面的变量。

3. 程序与函数、功能块的性能比较

程序、函数、功能块都是程序组织单元,但有不同性能。表 2-48 是它们的性能比较。

表 2-48 程序、函数和功能块的性能比较

性 能	函 数	功 能 块	程 序
允许使用 VAR	是	是	是
允许使用 VAR_INPUT	是	是	是
允许使用 VAR_OUTPUT	否	是	是
允许使用 VAR_IN_OUT	否	是	是
允许使用 VAR_EXTERNAL	否	是	是
允许使用 VAR_GLOBAL	否	否	是
允许使用 VAR_ACCESS	否	否	是
允许使用 VAR_TEMP	否	是	是
允许使用 VAR_CONFIG	否	否	是
函数值	是	否	否
可调用函数	是	是	是
可调用功能块	否	是	是
可调用程序	否	否	否
可递归调用	否	否	否
可间接功能块调用	否	是	是
具有过载和可扩展性 ^①	是	否	否
具有边沿检测的可能性	否	是	是
具有局部和输出变量的保持功能	否	是	是
可对直接表示变量声明 ^②	否	否	是
可对局部变量声明	是	是	是
可对功能块实例声明	否	是	是

注:① 用于标准函数。

② 仅用于具有 VAR_EXTERNAL 的功能块。

4. 程序指令

程序指令 `pragma` 用于编辑和预编辑时影响变量的属性,它作为程序的附属文本。

`Pragma` 指令用花括号 `||` 表示。程序编辑器不解释执行程序指令,即花括号内的指令。在花括号内的指令被作为注释处理。

2.6.4 程序组织单元

在传统的可编程控制器中,制造商设计了各种块,例如,组织块、程序块、数据块、功能块等。IEC 61131-3 标准将其统一为 3 种基本类型,即函数、功能块和程序。图 2-26 显示了各种传统的块向标准规定的程序组织单元演变。

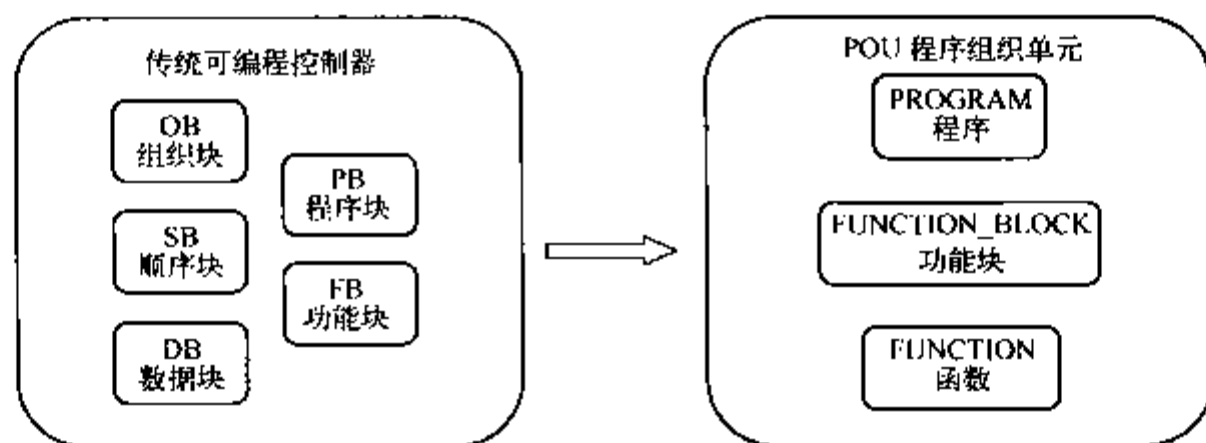


图 2-26 传统可编程控制器的模块演变为标准的程序组织单元

IEC 61131-3 标准除了简化块的类型外,还隐含了程序组织单元的类型含义,使其统一并进行了简化。程序组织单元是一个封装的单元,它可独立地由其他程序进行编译。被编译的程序组织单元能够相互连接组成一个完整程序。

程序组织单元由函数、功能块和程序组成,它是用户程序中的最小独立软件单元。根据用户应用要求,用户可使用程序组织单元组成所需的应用程序,完成所需的应用功能。

程序组织单元都由其各自的关键字(`FUNCTION`、`FUNCTION_BLOCK` 和 `PROGRAM`)开始,随后是名称(函数名、功能块名和程序名),中间是声明部分和程序组织单元的本体部分,最后以各自的结束关键字(`END_FUNCTION`、`END_FUNCTION_BLOCK` 和 `END_PROGRAM`)结束。声明部分是用于该程序组织单元的变量声明,包括变量类型、变量名、数据类型和初始值等。程序组织单元的本体是该程序组织单元要完成的任务,它可用 IEC 61131-3 标准规定的任何一种编程语言编写,也可以组合编写。

整个 PLC 工程中,程序组织单元名称是惟一的,一个程序组织单元声明后,其名称及调用接口就被该项目中其他所有程序组织单元所认识,即程序组织单元的名称具有全局性。

POU 的变量声明是按变量的类型分类进行的。对被声明的变量,应定义其数据类型,可选的初始值和其他属性,例如,过载或保持属性等。每类变量声明段,可同时对该本来类型的变量进行声明。相同数据类型的变量可集中声明。变量的次序由用户决定。

POU 具有调用属性,因此,POU 的接口特性应在变量声明段定义。POU 的接口包括调用属性、返回值和全局接口。

调用属性包括下列的 POU 形式参数的连接:

(1) 输入形式参数

在 VAR_INPUT 变量段声明。它将实际参数作为参数送 POU,即进行实际数据的复制。这些数据是根据其值调用的,称为“按值调用”(Call by Value)。因此,输入变量是不能修改的。

(2) 输入输出形式参数

实际参数以存储位置的指针形式传送到被调用的 POU,它可由被调用的 POU 读和写,因此它可被修改,称为“引用调用”(Call by reference)。这种修改能改变被调用 POU 外部声明的变量。对数组变量和结构变量,采用输入输出形式参数,因只使用指针,而不需复制数据,因此,可提高效率,但存在变量不受保护的缺点。

(3) 输出形式参数和返回值

函数的返回值不能被传送到被调用的 POU,它是由 POU 提供的返回值,因此,返回值不属于调用的接口参数。称为“以值返回”(Return by Value)。它允许调用的 POU 来读取该返回值。

调用程序类的 POU 时,由资源提供结合有实际参数的输出参数,并将输出参数赋值给有关变量,用于后续处理。

形式参数和返回值具有 POU 外部的可视性。因此,调用 POU 能显式使用其名称来输入参数数值,这使提供 POU 调用接口文件更容易,且可省略参数或更改它们的顺序,输入输出变量也可避免受到非授权的读写,而获得保护。

全局变量、存取变量和外部变量用于在所声明的配置、资源和程序内相互进行数据的交换,因此,具有全局使用的属性。

程序组织单元不是递归的,即程序组织单元的调用不造成同类型另一个程序组织单元的调用。不同类型的程序组织单元允许存取的变量不同,调用的范围也不同。采用程序组织单元可简化用户程序,使程序标准化。

POU 的本体部分可采用标准规定的 5 种编程语言编写。如果使用其他编程语言,则应注意变量的使用,其实现的方法必须与标准规定的编程语言相同,即采用相同的变量声明方法,而对功能块和函数的调用也必须符合 IEC 61131-3 标准等。

程序组织单元具有下列特点:

1) 可对每个应用领域设置用户的功能块库,便于工程的应用。例如,建立运动控制功能块库等。

2) 可对功能块进行测试和记录。

3) 能够提供全球范围内的库存取功能。

4) 可重复使用,使用的次数无限制。

5) 可改变编程,用于建立功能块网络。

6) 可节省工厂投资成本。

第3章 文本类编程语言

3.1 文本类编程语言及其公用元素

3.1.1 文本类编程语言概述

IEC 61131-3 标准规定了两种文本类编程语言：一种是指令表(Instruction List,IL)编程语言,用一系列指令组成程序组织单元本体部分。另一种是结构化文本(Structured Text,ST)编程语言,用一系列语句组成程序组织单元本体部分。指令表编程语言是低层编程语言,结构化文本编程语言是高层编程语言。

指令表编程语言中,通常,一条指令由一个操作符或一个函数与一定数量的操作数组合,用于实现一定的操作功能。操作符或函数用于说明进行什么操作,操作数用于说明操作的对象。操作符通常只有一个操作数,函数可能有一个或多个(或没有)操作数。对操作符也可带修正符,例如,取反(N)、结果为布尔量1时执行(C)等修正符。

指令表编程语言是类似汇编语言的编程语言,它是低层语言,具有容易记忆、便于操作的特点。因此,适合用于解决小型的容易控制的系统编程。

结构化文本编程语言是高级编程语言,类似于高级计算机编程语言 PASCAL。它由一系列语句,例如,选择语句、循环语句、赋值语句等组成,用于实现一定的功能。它不采用面向机器的操作符,而采用能够描述复杂控制要求的功能性抽象语句,因此,具有清晰的程序结构,利于对程序的分析。它具有强有力的控制命令语句结构,使复杂控制问题变得容易解决。但它的编译时间长,执行速度慢。

3.1.2 文本类编程语言的公用元素

IEC 61131-3 标准规定在文本类编程语言(指令表编程语言和结构化文本编程语言)中可使用的公用程序结构元素,如表 3-1 所示。

表 3-1 用于文本类编程语言的程序结构元素

程序结构元素	说 明	备 注
TYPE...END_TYPE	数据类型段声明	
VAR...END_ VAR	内部变量段声明	
VAR_INPUT...END_ VAR	输入变量段声明	
VAR_OUTPUT...END_ VAR	输出变量段声明	
VAR_IN_OUT...END_ VAR	输入输出变量段声明	
VAR_EXTERNAL...END_ VAR	外部变量段声明	
VAR_TEMP...END_ VAR	暂存变量段声明	

(续)

程序结构元素	说明	备注
VAR_ACCESS...END_ VAR	存取路径变量段声明	
VAR_GLOBAL...END_ VAR	全局变量段声明	
VAR_CONFIG...END_ VAR	组态变量段声明	
FUNCTION...END_ FUNCTION	函数段声明	
FUNCTION_BLOCK...END_ FUNCTION_BLOCK	功能块段声明	
PROGRAM...END_ PROGRAM	程序段声明	
STEP...END_STEP	步段声明	用于 SFC 语言
TRANSITION...END_TRANSITION	转换段声明	用于 SFC 语言
ACTION...END_ ACTION	动作段声明	用于 SFC 语言

文本类编程语言的程序结构元素用于程序组织单元中的声明部分。当 SFC 语言用结构化文本编程语言表示时,上述 SFC 语言的公用元素用于程序组织单元中的本体部分。

与其他编程语言相同,编程语言用于程序组织单元中本体的编程,它必须与程序组织单元中声明部分相适应。

3.2 指令表编程语言

3.2.1 指令

指令表编程语言以一系列指令作为编程语言。与传统 PLC 的指令表编程语言比较,IEC 61131-3 标准的指令表编程语言更为简单,其原因是采用了修正符、函数和功能块,一些原来用指令执行实现的操作可通过修正符、函数和功能块的调用方便地实现。

指令表编程语言的主要特点是非常简单和容易学习。对于小型的简单过程控制问题是理想的编程语言。大多数传统的可编程控制器都提供指令表编程语言。其区别是传统可编程控制器的操作指令很多,对不同的数据类型的同一类型运算要有不同的指令等。

由于指令表编程语言是低层编程语言,它较容易转换为可编程控制器的机器代码,因此,容易将指令表编程语言的程序下载,而不需要对程序进行编译和建立等过程,而用结构化文本编程语言编写的程序必须经编译后才能下载和运行。指令表编程语言常常被作为基础编程语言,其他编程语言能够方便地转换为指令表编程语言。但是,指令表编程语言对大型的复杂控制问题缺少有效的工具,因此,在大型复杂的控制问题中,通常不采用指令表编程语言。

1. 操作符、修正符和操作数

指令表编程语言的指令表由一系列指令组成。每个指令表示一个新行的开始。它由操作符(可带修正符)和操作数组成。多于一个的操作数,可用逗号分隔。IEC 61131-3 标准将指令表编程语言的操作符简化为 24 种,并规定如表 3-2 所示的操作符和修正符。

表 3-2 指令表编程语言规定的操作符和修正符

操作符	修正符	操 作 数	说 明	传统 PLC 的操作符和修正符示例
LD	N	ANY	设置当前值等于操作数	LD, LDI, STR, LD NOT
ST	N	ANY	存储当前值到操作数位置	OUT, OUT NOT,
S		BOOL	如果当前值是布尔 1, 则操作数置位到 1	S, SET
R		BOOL	如果当前值是布尔 1, 则操作数复位到 0	R, RESET, RST
AND	N, (ANY	布尔逻辑与	AND, OR, NOT, ANDI, ORI, ANI, OI, AD, ORD, INV
&	N, (ANY	布尔逻辑与	
OR	N, (ANY	布尔逻辑或	
XOR	N, (ANY	布尔逻辑异或	
NOT		ANY	布尔逻辑反(取反)	
ADD	(ANY	加	ADD, SUB, MUL, DIV
SUB	(ANY	减	
MUL	(ANY	乘	
DIV	(ANY	除	
MOD	(ANY	模除	
GT	(ANY	比较: 大于	CMP, GT, GE, EQ, NE, LE, LT
GE	(ANY	比较: 大于等于	
EQ	(ANY	比较: 等于	
NE	(ANY	比较: 不等于	
LE	(ANY	比较: 小于等于	
LT	(ANY	比较: 小于	
JMP	C, N	LABEL NAME	跳转到标号	JMP, JME
CAL	C, N		调用功能块	
RET	C, N		从被调用的函数、功能块或程序返回	
)			计算延缓的操作	

- 注: 1. CAL 的操作数应是一个被调用的功能块实例名。
2. NOT 操作的结果是当前结果的位取反, 修正符 N 表示取反操作。
3. RET 操作符不需要操作数。
4. JMP 指令操作需配有一个可以跳转执行的有该标号 LABEL 的指令。
5. 操作数是 ANY 数据类型的操作具有过载功能。即操作数可以是基本数据类型的任何一种。
6. 修正符 C 表示有关指令只有在当前运算的结果是布尔 1(或当操作符布尔值为 0, 并与“N”修正符结合时)时才执行。
7. 操作符可多于一个修正符, 既可同时, 也可只用一个或不用。例如, OR 有 OR, OR(, ORN 和 ORN(等 4 种格式, JMP 有 JMP, JMP C, JMPN, JMP C 等 4 种格式。

左圆括号“(”表示操作符的运算被延缓直到遇到右圆括号“)”。因此, 对传统 PLC 中的程序块操作, 主控操作等都可采用该操作符实现。

比较操作是将当前结果与比较操作的操作数比较, 满足比较条件时, 比较的结果为布尔 1。例如, 当前结果为 2#0000_0101, 则比较指令 GT %IB5 执行时, 如果 %IB5 字节的内容大于 2#0000_0101, 比较结果为 1。反之, 比较结果为 0。

操作数可以是直接表示变量或符号变量等。例如, LD A 表示设置当前值等于符号变量 A 所对应的数值。AND %IX1.3 表示将当前结果与输入单元 1 的第 3 位进行与逻辑运算, 结果作为当前值。JMP ABC 表示当前计算值为布尔值 1 时, 从标号 ABC 的位置开始执行。RET 是无操作数的操作符, 当执行到该指令时, 程序将返回到原断点后的指令处执行。断点是由于函数调用、功能块调用或中断子程序等造成的。

数据存储单元中,单元地址用“字节·位”表示。例如,%IX1.1表示第1输入单元的第1位。字节、字、双字和长字的单元地址用相应的符号B,W,D和L表示。例如,%QB5表示第5输出单元的一个字节。

传统 PLC 中对数据存储单元地址的分类与标准的分类有所不同。首先,标准用%表示这些存储单元是直接表示变量的地址。用位置前缀表示输入(I)、输出(Q)和存储器(M)单元。用位(X)、字节(B)、字(W)、双字(D)和长字(L)表示数据存储单元的大小前缀。使用时应注意,传统 PLC 中,不同制造商产品的地址是不同的,例如,可以直接用0000表示存储器的位0000,也可用I1.0表示第1输入单元的第0位等。

2. 指令

IEC 61131-3 标准的指令表编程语言对传统指令表编程语言进行了总结,取长补短,采用函数和功能块,使用数据类型的超载属性等,使编程语言更简单灵活,指令更精简。其主要优点如下:

1) 采用函数和功能块调用,简化指令集。使原有指令集简化为9类24种指令。例如,移位指令、算术运算指令、位串类指令等都可以用标准提供的函数直接实现。定时器和计数器指令也可用标准提供的定时器和计数器功能块的调用实现。

2) 数据类型的超载属性使运算变得方便。传统 PLC 指令对不同数据类型的运算要用不同的指令,IEC 61131-3 标准采用相同的指令实现。例如,整数加和实数加指令在传统 PLC 中是不同的指令,而标准中,采用相同的指令 ADD 即可实现。此外,数据类型的超载属性简化了数据类型转换指令等。

3) 采用圆括号可以方便地将程序块组合,并实现主控等指令。传统 PLC 指令集对程序块的操作采用专用指令,例如,AND LD、OR LD、ANB、ORB 等,标准提供圆括号,使程序块可以在圆括号内部先执行,从而简化指令。此外,主控指令和主控返回指令,条件转移和返回指令等也可用圆括号的方式实现。

4) 采用边沿检测属性的方法,对信号设置微分功能,简化了指令集。传统 PLC 中,各种指令都可以采用微分指令,即在边沿条件满足时只执行一次。标准中将这些指令用数据类型的边沿检测属性来区分,从而大大简化指令。例如,传统 PLC 指令集的 PLS 和各种指令的微分指令等在标准中都被边沿检测属性代替。

5) 数据传送类指令可直接用赋值函数 MOVE 实现。由于该函数具有数据类型超载属性,因此,数据传送类指令得到简化。

6) 设置时间类型文字和数据类型,使定时器定时设定信号的输入变得简单。传统 PLC 的定时器设定时间不仅与设置数值有关,还与所采用的时基时钟信号有关。标准中,用 Time 时间类型数据,可直接设置有关时间。此外,设定时间也因数据类型超载属性而使设定时间范围大大扩展,能够满足工业应用的要求。传统 PLC 由于存储单元范围的限制,对长计时或长计数的应用项目,通常要用多个定时器和计数器串联来实现。

7) 数据存储变得简单。IEC 61131-3 标准对数据存储采用按数据类型分类存储的方法,因此,数据存储更简单,并且不会发生错误。传统 PLC 沿用电气控制的习惯,采用输入、输出、专用继电器等分类,因此,容易出现数据存储的出错。

8) 不需要 END 标志。由于程序组织单元等有 END_PROGRAM 等结束标志,因此采用标准规定的指令编程时,程序的结束不需要 END 标志。

9) 与汇编语言不同,指令表编程语言没有特定的处理器状态位,因此,用比较操作的结果是否满足作为条件跳转或函数调用的依据。

3. 指令的格式

在指令表编程语言中,指令具有如下格式:

标号:操作符/函数 操作数 注释

【例 3-1】 指令格式的示例。

下列程序中,有 4 条指令:

标号	操作符和修正符	操作数	注释
START:	LD	%IX1.1	(* 按下起动按钮 *)
	OR	%QX10.1	(* 自保触点 *)
	ANDN	%IX1.2	(* 按下停车按钮 *)
	ST	%QX10.1	(* 起动设备 *)

示例程序用于对某设备进行起保停控制。程序中,标号为 START 的指令读取可编程控制器第 1 号输入单元第 1 位信号(起动按钮),存放到运算结果累加器。第 2 行是第 2 个指令,用于将第 1 行指令结果和第 10 号输出单元第 1 位信号(设备的自保信号)进行或逻辑运算,运算结果仍存放到运算结果存储器。第 3 行指令用于将第 2 行运算结果和第 1 号输入单元第 2 位信号(停止按钮)取反后的结果进行与逻辑运算,结果仍存放到运算结果存储器。第 4 行指令用于将运算结果存储器存放的信号传送到第 10 号输出单元的第 1 位存放。

指令中,操作符用于规定操作的方法,例如,和运算结果存储器进行或逻辑运算,和运算结果存储器进行与逻辑运算等。操作数是操作的对象。例如,%IX1.1 表示对第 1 号输入单元的第 1 位进行操作。

指令表编程语言提供了一个存储当前结果的存储(累加)器,与传统的可编程控制器使用的累加器不同,这种标准累加器的存储位数是可变的,即标准指令表编程语言提供了一种存储位数可变的虚拟累加器,其存储位数取决于正在处理的操作数的数据类型。同样,虚拟累加器的数据类型也可发生变化,以适应最新运算结果的操作数的数据类型。

指令执行过程中,数据存取采用的方法是:

运算结果 := 当前运算结果 操作 操作数

因此,在操作符规定的操作下,当前运算结果与操作数进行由操作符规定的操作运算。运算结果作为新的运算结果存放回当前运算结果的累加器。

在指令中,可根据应用要求添加注释。须注意,除注释不能嵌套外,注释也不允许将操作符关键字中断,和不允许将操作数中断。注释应在程序行的最后面,不允许在行首和中间。

操作符与操作数之间至少需要有一个空格来分隔。标号与操作符之间用冒号分隔。标号用于一些操作,例如,JMP 操作的操作数表示跳转的目的地址。

IEC 61131-3 标准规定的操作符和修正符如表 3-2 所示。修正符是对操作符的修正,例如,修正符 N 表示对操作符运算结果取反。修正符 C 表示所连接的操作符指令只有在当前运算的结果是布尔 1(或与“N”修正符结合时,当操作结果布尔值为 0)时才执行。

4. 指令表编程语言的特点

指令表编程语言的特点如下:

- 1) 指令表编程语言的指令具有简单易学的特点。适用于小型较简单控制系统的编程。
- 2) 操作符被用于操纵所有基本数据类型的变量、调用函数和功能块。
- 3) 指令表编程语言是能够直接在 PLC 内部解释的语言,适用于大多数 PLC 制造商。
- 4) 大多数指令表编程语言编写的程序要在运行期才能检出程序的出错。
- 5) 指令表编程语言编写的程序较难转换到其他编程语言,其他编程语言编写的程序容易转换到指令表编程语言。

5. 各类操作指令

指令表编程语言简化的 9 类指令说明如下。

(1) 数据存取类指令

数据存取类指令是读取数据存储单元内容的操作。标准指令集采用 LD 和 LDN 指令表示存取和存取取反指令。编程语言的格式如下：

LD 操作数 (* 将操作数规定的数据存储单元中的内容作为当前结果存储 *)
 LDN 操作数 (* 将操作数规定的数据存储单元中的内容取反后作为当前结果存储 *)

LD 是 Load 的缩写,LDN 是 Load Not 的缩写。

数据存取类指令的操作对象,即 LD 或 LDN 的操作对象,是操作数。它是对操作数对应的数据存储单元内容进行的读取操作。读取的数据被存放在运算结果累加器,该数据也被称为当前值。

继电器控制系统中,读取操作是获得物理触点的状态,由于继电器的触点数有限制,因此,继电器控制系统中不允许对同一编号的接点多次读取(通常不大于 8 次)。

对常开触点的的数据读取用 LD 指令,对常闭触点的的数据读取用 LDN 指令。传统 PLC 中,通常用 LD(Load)、LD NOT、STR(Start)、XIC、XIO 等指令表示。

与传统 PLC 不同,标准对数据存取类指令的操作数的数据类型未作限制性的规定(即被存取数的数据类型可以是表 2 - 13 基本数据类型性能表中的任意类型)。因此,可以直接存取一个字节或更大数据存储单元的内容,例如,存取一个长字用 LD %H2,存取一个字节用 LD %IB5 等。

与继电器逻辑电路相似,对常开触点,即动合接点,采用存取 LD 指令,例如,LD %IX0.0 指令执行存取操作数地址为%IX0.0 接点(第 0 输入单元的第 0 位)状态的操作。从寄存器看,该操作过程是把地址为%IX0.0 的输入状态寄存器状态传送到运算结果累加器 R。图 3 - 1 显示了数据存取类指令的执行过程。

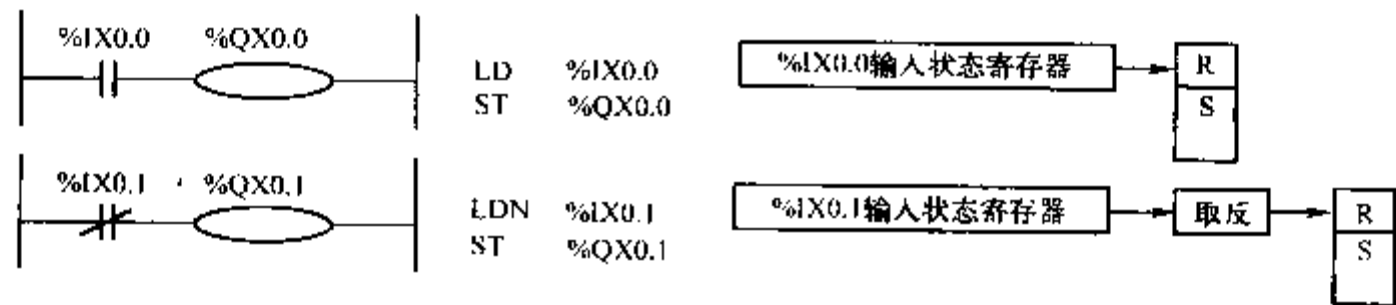


图 3 - 1 LD 和 LDN 指令的示例和操作过程

对常闭接点,即动断接点,采用逻辑取反 LDN 指令,例如,LDN %IX0.1 指令执行存取操作数地址为%IX0.1(第 0 输入单元的第 1 位)接点状态的操作。从寄存器看,该操作过程是把地址为%IX0.1 的输入状态寄存器状态取反,并把取反的结果传送到运算结果累加器 R。

由于运算结果累加器位于堆栈的第 1 层,因此,存放数据时,原存放在运算结果累加器的数据被压到堆栈的下 1 层。

图中也显示了梯形图编程语言中 LD 和 LDN 指令相应的图形形式。表 3-3 是 LD 和 LDN 指令的示例。

表 3-3 LD 和 LDN 指令的示例

指 令	说 明	当前结果累加器的数据类型
LD FALSE	当前值等于 FALSE	布尔量
LD TRUE	当前值等于 TRUE	布尔量
LD 3.1415	当前值等于 3.1415	实数
LD 100	当前值等于 100	整数
LD T#0.5S	当前值等于时间常数 0.5 s	时间数据
LD START	当前值等于变量 START 的状态值	根据变量 START 的数据类型确定
LD ANY_VAR1	当前值等于任何变量 ANY_VAR1 的值	可以是布尔量、整数、实数等
LD TIME_VAR1	当前值等于时间变量 TIME_VAR1 的值	时间数据
LDN BOOL_VAR1	当前值等于布尔变量 BOOL_VAR1 的值取反	布尔量

(2) 输出类指令

输出类指令用于将运算结果累加器 R 的内容传送到输出状态寄存器。标准指令集采用 ST 和 STN 指令表示存取和存取取反指令。编程语言的格式如下：

- ST 操作数 (* 将当前结果存储到操作数规定的数据存储单元 *)
- STN 操作数 (* 将当前结果取反后,存储到操作数规定的数据存储单元 *)

须指出,在执行 ST 或 STN 指令后,当前运算结果仍被保留在运算结果累加器的存储单元,ST 是 Store 的缩写,STN 是 Store Not 的缩写。传统 PLC 中,输出类指令用 OUT、Q、OUT NOT、OTE 等。

与继电器逻辑电路相似,对激励线圈,用 ST 指令,例如,ST %QX0.0 指令执行输出到%QX0.0 激励线圈的操作;从寄存器看,该操作过程是把运算结果累加器 R 的状态传送到地址为%QX0.0 的输出状态寄存器。图 3-2 显示了输出类指令的执行过程。

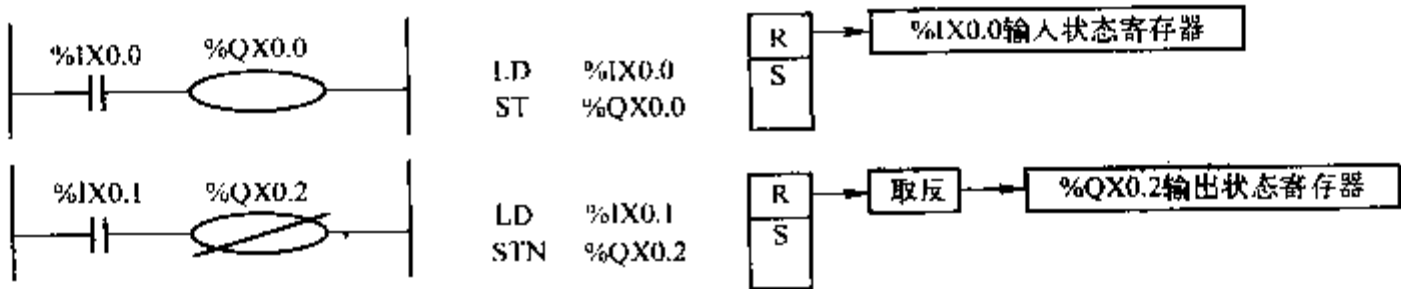


图 3-2 ST 和 STN 指令的示例和操作过程

对失励线圈,用 STN 指令,例如,STN %QX0.2 指令执行输出到%QX0.2 失励线圈的操作;从寄存器看,该操作过程是把运算结果累加器 R 的状态取反,并把取反的结果传送到地址为%QX0.2 的输出状态寄存器。

传统 PLC 的存取机制。传统 PLC 有 3 类存储器。它们是输入、输出存储器,当前结果存

储器和堆栈。当执行 LD 指令时,CPU 从操作数对应的输入存储器读取数据并存放到目前结果存储器。执行 OUT 输出指令时,将当前结果存储器的内容传送到操作数对应的输出存储器。

IEC 61131-3 标准规定的存取机制与传统 PLC 存取机制类似,但有下列区别:传统 PLC 存取机制中,存储器是位存储器,因此,只能存储位的内容。标准规定的存储器可以存储位、字节、字、双字、长字及字符等各种类型的数据或变量。因此,标准规定的存取范围大大扩展。

【例 3-2】 标准规定的存取机制。

```
VAR
    A1,A2,A3:INT:=5;
    STR1:STRING[20]:= 'WRONG';
    STR2:STRING[30];
END_VAR
LD    A1      (* 将 A1 内容送当前结果存储器,即当前结果存储器内容为整数 5 *)
ADD   A2      (* 将 A2 内容与当前结果存储器内容相加后,即结果为 10,送当前结果存储器 *)
ST    A3      (* 将当前结果存储器内容,即 10,送 A3,使 A3 内容从 5 变为 10 *)
LD    STR1    (* 将 STR1 内容送当前结果存储器,即当前结果存储器内容为字符串'WRONG' *)
ST    STR2    (* 将当前结果存储器内容,即'WRONG'送 STR2 *)
```

例 3-2 中,A1,A2 和 A3 是内部变量,初始值均为 5,STR1 和 STR2 是字符串数组变量,其中,STR1 的初始值为'WRONG'。须注意,执行过程中,当前结果存储器的内容从整数数据类型变为字符串数据类型,因此,存储器的空间发生变化。

【例 3-3】 8 台电动机的起保停控制。

```
VAR_INPUT
    STT : AT %IB0 ;
    STP : AT %IB1 ;
END_VAR
VAR_OUTPUT
    MOTOR : AT %QB0 ;
END_VAR
LD    STT      (* 读取 STT 在%IB0 的数据 *)
OR    MOTOR    (* 与 MOTOR 的数据进行 OR 运算 *)
ANDN  STP      (* 将当前结果存储器内容与 STP 在%IB1 的数据进行 ANDN 的操作 *)
ST    MOTOR    (* 将当前结果存储器内容送在存储器%QB0 的变量 MOTOR *)
```

例 3-2 用字节%IB0 存放 8 个电动机的起动按钮信号,字节%IB1 存放 8 个电动机的停止按钮信号。用字节%QB0 连接 8 个电动机的接触器 KM,用一个程序实现了对 8 台电动机的起保停控制。而传统 PLC 要用 8 个同样格式的程序实现。图 3-3 是该系统的接线图。

• 由于存储器存储数据类型可以不同,因此,连续的两个运算之间应注意数据类型的匹配。

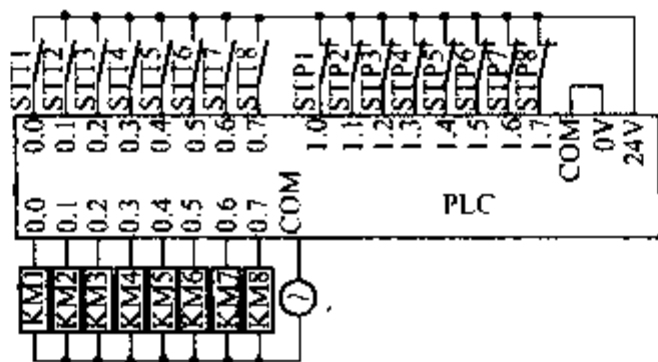


图 3-3 8 台电动机起保停控制接线图

【例 3-4】 数据类型的错误匹配。

```

VAR
    A1,A2:INT :=5;
    STR1:STRING[20] := 'WRONG';
    BB :DWORD;
END_VAR
LD      A1      (* 将 A1 内容送当前结果存储器,即当前结果存储器内容为 5 *)
ADD     33      (* 将整数 33 与当前结果存储器内容相加后,即结果为 38,送当前结果存储器 *)
ST      A2      (* 将当前结果存储器内容,即 38,送 A2,使 A2 内容从 5 变为 38 *)
LD      STR1    (* 将 STR1 内容送当前结果存储器,即当前结果存储器内容为'WRONG' *)
ST      BB      (* 出错,因 BB 数据类型 DWORD 与 STR1 数据类型不匹配 *)
    
```

示例中,执行 LD A1 后,当前结果存储器内数据是整数 A1,它是整数数据类型,因此,可与整数 33 相加,结果 38 送回当前结果存储器。但在执行 LD STR1 后,当前结果存储器内数据是字符串数据类型,因此,不能将该数据输出到 DWORD 数据类型的 BB 中。

- 当前结果存储器内的数据类型可以改变。可用表 3-4 所示操作符组使当前结果存储器的数据类型改变或数值改变。标准并没有规定操作符组,不同编程系统可用不同方法实现操作符组。

表 3-4 能够改变当前结果存储器数据类型或数值的操作符组

改变当前结果存储器数据类型的操作符组	缩 写	示 例
Create(建立)	C	LD
Process(处理)	P	GT,GE,LT,ADD,SUB,AND,OR
Leave unchanged(保持不变)	U	ST,JMPC,CALC,RETC(见注 2)
Set to undefined(设置为未定义)	-	CAL 功能块无条件调用(见注 1)

- 注:1. 功能块调用时,随后的指令必须重新装载当前结果存储器,因为,当从被调用的功能块返回时,当前结果存储器内是未定义值的。为此,在功能块中的第一个指令应是 LD,JMP,CAL 或 RET 指令,因这些指令是不要求有效的当前结果存储器内容的。
2. 该类指令将上一指令的当前结果存储器内容传送到下一指令,并且不改变其数据类型和数据。例如,ST 指令执行后,在当前结果存储器中仍保持原有的输出结果。

【例 3-5】 改变当前结果存储器数据类型和数值。

```

VAR
    
```

```

A1,A2,A3;INT:=8;
END_VAR
LAB1: LD      A1(* 将 A1 内容送当前结果存储器,即当前结果存储器内容为 8 *)
      ADD     2(* 整数 2 与当前结果存储器内容相加后的结果 10 送当前结果存储器 *)
      ST      A2(* 将当前结果存储器内容,即 10,送 A2,使 A2 内容从 8 变为 10 *)
      ST      A3(* 当前结果存储器内容没有改变,为 10,因此,10 被送到 A3 *)
      GT      5(* 当前结果存储器内容大于 5,使当前结果存储器内容变为布尔量 1 *)
      JMPCL   LAB2(* 当前结果存储器内容为 1,跳转执行 LAB2 开始的程序 *)
      JMP     LAB3(* 当前结果存储器内容为 0,跳转执行 LAB3 开始的程序 *)
LAB2: ....

```

例 3-5 中,GT 5 指令使当前结果存储器内容从整数数据类型改变为布尔数据类型。JMPC LAB2 指令应根据 LAB2 标号开始的程序重新设置当前结果存储器内容。

表 3-5 是 ST 和 STN 指令的示例。

表 3-5 ST 和 STN 指令的示例

指 令		说 明
LD	TRUE	当前值等于 TRUE
ST	START	当前值等于 TRUE,START 变量值等于 TRUE
STN	STOP	当前值等于 TRUE,STOP 变量值等于 FALSE
LD	3.1415	当前值等于 3.1415
ST	PI	当前值等于 3.1415,PI 变量值等于 3.1415
LD	T#0.5S	当前值等于 0.5 s
ST	TON_1.PT	当前值等于 0.5 s,TON_1 的输入 PI 值等于 0.5 s

(3) 置位和复位类指令

标准指令集采用 S 和 R 指令表示置位和复位类指令。编程语言的格式如下:

S 操作数 (* 当前结果为布尔 1 时,将操作数对应的数据存储单元内容设置为 1,并保持 *)

R 操作数 (* 当前结果为布尔 1 时,将操作数对应的数据存储单元内容设置为 0,并保持 *)

这类指令具有记忆属性。执行 S 操作数后,操作数对应的数据存储单元内容被设置为 1,并且该数据存储单元内容被记忆和保持到执行 R 操作数指令,执行 R 操作数指令使操作数对应的数据存储单元内容被设置为 0。同样,该存储单元的内容要保持到执行 S 操作数的指令,并使其内容设置到 1 为止。

S 和 R 指令可用 SR 和 RS 功能块的调用实现。与功能块比较,其不同点是:S 指令和 R 指令的执行是根据程序中的先后位置确定执行先后次序,因此,优先级的确定与 RS 和 SR 有所不同。此外,功能块要先设置 S 和 R 端,才能执行调用指令。

与电气控制中起动和停止控制比较,S 指令实现起动控制,R 指令实现停止控制。不同点是:电气控制的起动和停止控制中,起动按钮信号是常开触点,停止按钮信号是常闭触点。而 S 和 R 指令中,这两个信号都用常开触点。此外,对起动优先和停止优先的控制,继电器电气控制的接线也与其不同。

S 是 Set 的缩写,R 是 Reset 的缩写。这类指令在传统 PLC 指令中,常采用 S、SET、R、RST

指令表示置位和复位指令。表 3-6 是 S 和 R 指令的示例。

表 3-6 S 和 R 指令的示例

指 令			说 明
SETEX:	LD	TRUE	当前值等于 TRUE
	S	START	当前值等于 TRUE, START 变量值置 TRUE, 并保持
	LD	FALSE	当前值等于 FALSE
	S	STOP	当前值等于 FALSE, STOP 变量值置 FALSE, 并保持
RESETEX:	LD	TRUE	当前值等于 TRUE
	R	STOP	当前值等于 TRUE, STOP 变量值置 FALSE, 并保持

S 表示有条件的输出 STC 指令, R 表示有条件输出 STCN 指令。因此, 当前结果存储器为 1 时, S 操作数指令执行设置输出操作数为 1 (置位) 的操作, 同样, R 操作数指令执行设置输出操作数为 0 (复位) 的操作, 即置位取反的操作。

(4) 逻辑运算类指令

标准指令集规定的逻辑运算类指令有: AND(N)、OR(N)、XOR(N) 和 NOT 等。编程语言格式如下:

逻辑运算操作符 操作数 或 逻辑运算操作符 N 操作数

可表示为下列各种形式:

AND 操作数 或 ANDN 操作数 或 & 操作数 或 &N 操作数
OR 操作数 或 ORN 操作数
XOR 操作数 或 XORN 操作数
NOT 操作数

1) 逻辑运算操作符。操作数用于将当前结果存储器内容与操作数对应的数据存储单元内容进行规定的逻辑运算, 运算结果作为新的当前结果, 存放在当前结果存储器内。逻辑运算操作符包括与 (AND 或 &)、或 (OR)、非 (NOT) 和异或 (XOR) 逻辑运算。

2) 逻辑运算操作符 N。操作数用于将当前结果存储器内容与操作数对应的数据存储单元内容的取反结果进行规定的逻辑运算, 运算结果作为当前结果, 存放在当前结果存储器内。

标准指令表编程语言中, 逻辑运算类指令的操作数只能是一个。但操作范围可以是对位、字节、字、双字和长字的逻辑运算, 它们是对其中的每个位进行逻辑运算。

传统 PLC 指令表编程语言中也有逻辑类运算指令, 但操作范围一般局限于位。指令除 AND、OR 等外, 也有用 A、O 等指令。小型 PLC 通常不设 XOR 指令。

【例 3-6】电动机控制程序的示例。

LD A (* 存取符号变量 - 起动按钮 A 的信号 *)
OR C (* 与输出变量 C 进行或运算, 实现触点自保 *)
ANDN B (* 与停止按钮 B 的取反信号进行与逻辑运算 *)
ST C (* 输出到输出变量 - 接触器 C 的信号 *)

本例与例 3-3 类似, 是典型的电动机控制程序。须注意, 在标准编程语言中, 输入变量

A、B 和输出变量 C 都是符号变量,需在变量声明部分说明其实际地址,如例 3-3 所示。由于,程序中没有实际地址,因此,程序可以重复使用。只需在实际应用时,将有关变量在声明部分赋予实际地址。因此,与传统 PLC 程序比较,本例的程序具有可重复使用的性能。

(5) 算术运算类指令

这类指令包括 ADD(加)、SUB(减)、MUL(乘)、DIV(除)和 MOD(模除)等。编程语言的格式如下:

ADD 操作数 (* 当前结果加操作数对应的数据存储单元内容,运算结果存当前结果存储器 *)
SUB 操作数 (* 当前结果减操作数对应的数据存储单元内容,运算结果存当前结果存储器 *)
MUL 操作数 (* 当前结果乘操作数对应的数据存储单元内容,运算结果存当前结果存储器 *)
DIV 操作数 (* 当前结果除以操作数对应的数据存储单元内容,运算结果(即商)存当前结果存储器 *)
MOD 操作数 (* 当前结果与以操作数对应的数据存储单元内容为模进行模除的运算结果(即余数)存当前结果存储器 *)

与传统 PLC 指令表编程语言比较,IEC 61131-3 标准算术运算类指令有下列特点:

1) IEC 61131-3 标准的算术运算类指令可适用于各种不同的数据类型,因为具有过载属性,因此,功能大大增强。指令也更简单。此外,当前结果存储器数据类型成为一般数据类型,因此,能适应各种算术运算需要。

2) IEC 61131-3 标准的 DIV 运算将商作为当前结果,而 MOD 运算将余数作为当前结果。传统 PLC 中,进行除法运算时,商和余数存放在不同数据存储单元地址。

3) 传统 PLC 对数据类型有严格规定,例如,单字或双字的加运算等。为此,在双字运算时,有时,必须先进行有关高位字节清零等操作,使程序复杂化。

【例 3-7】 温度补偿系数的计算。

LD	273.15	(* 存取开尔文零度的温度值,即实数 273.15 *)
ADD	T1	(* 与实数变量 T1 进行加法运算 *)
DIV	373.15	(* 除以设计温度 100℃ 对应的开尔文温度值 *)
ST	COMP	(* 输出到输出变量 COMP,作为温度补偿系数 *)

例 3-7 用于对气体流量进行温度补偿。其中,T1 是实际温度,单位是℃。程序第 1 行读取 273.15;第 2 行将温度实际值 T1 与当前值 273.15 相加,并作为当前值;第 3 行将该当前值除以设计温度值(已转换为开尔文温度),结果存放在当前值存储器;第 4 行将运算结果作为温度补偿数值存放在 COMP 变量。可以看到,程序中,ADD 和 DIV 的运算都是实数数据类型的运算。

(6) 比较运算类指令

这类指令包括:GT(>)、GE(>=)、EQ(=)、NE(<>)、LE(<=)和 LT(<)。编程语言的格式如下。

GT 操作数 (* 当前结果 > 操作数对应的数据存储单元内容,运算结果 1 送当前结果存储器 *)
GE 操作数 (* 当前结果 ≥ 操作数对应的数据存储单元内容,运算结果 1 送当前结果存储器 *)
EQ 操作数 (* 当前结果 = 操作数对应的数据存储单元内容,运算结果 1 送当前结果存储器 *)
NE 操作数 (* 当前结果 ≠ 操作数对应的数据存储单元内容,运算结果 1 送当前结果存储器 *)

LE 操作数 (* 当前结果 \leq 操作数对应的数据存储单元内容,运算结果 1 送当前结果存储器 *)

LT 操作数 (* 当前结果 $<$ 操作数对应的数据存储单元内容,运算结果 1 送当前结果存储器 *)

这类指令用于将当前结果与操作数对应的数据存储单元内容比较,满足操作符规定的比较条件时,当前结果被置 1,反之,当前结果被清零。比较类指令将当前结果存储器的数据类型改变为布尔数据类型。应用比较类指令时应注意下列问题:

1) 传统 PLC 中,用 CMP 等比较类指令,它将比较结果存放在专用存储单元,用户根据该专用存储单元的状态(0 或 1)确定后续程序的执行。此外,比较的功能单一,例如,只有大于、小于和等于等指令,没有大于等于和小于等于等指令,为此,要将有关指令的结果用或逻辑运算后才能实现大于等于和小于等于等指令的功能。

2) IEC 61131-3 标准的比较运算类指令适用于对不同数据类型的变量比较,而不局限于单一位的比较,因此,应用范围大大扩展。

【例 3-8】 比较运算类指令的示例。

LD	A1	(* 存取实数变量 A1 *)
GT	20.0	(* 变量 A1 与 20.0 进行大于比较 *)
ST	RED	(* 如果大于 20.0,表示 A1 超限,因此,红色报警灯 RED 置 1 *)
STN	GREEN	(* 如果不大于等于 20.0,则绿色 GREEN 灯置 1 *)

例 3-8 中,变量 A1 是过程测量值,当其值大于 20.0 时,表示测量值超限,红色报警灯 RED 被点亮,反之,测量值在允许范围,GREEN 灯点亮。

(7) 跳转和返回指令

IEC 61131-3 标准的跳转指令是 JMP 指令。IEC 61131-3 标准的返回指令是 RET 指令。编程语言的格式如下:

JMP 标号 (* 跳转到标号的位置继续执行 *)

RET (* 返回到跳转时的断点后继续执行 *)

跳转指令的操作数是标号,不是操作数对应的数据存储单元地址。

返回指令是没有操作数的指令,用于调用函数、功能块和程序的返回。

JMP 是 Jump 的缩写。执行该指令时,如果当前结果为布尔 1,则跳转条件满足,程序在该点中断,并跳转到该标号所在的程序行继续执行。它与 RET 指令配合,用于实现子程序的执行。可以带修正符 C 和 N,表示根据当前结果存储器内容执行或取反。

跳转指令与传统 PLC 指令表编程语言中的主控指令和跳转指令类似。

RET 是 Return 的缩写。执行该指令后,程序返回,并从原断点后第一条指令开始执行。可以带修正符 C 和 N,表示根据当前结果存储器内容执行或取反。

使用跳转和返回指令时注意下列事项:

1) 跳转指令是从主程序跳转到子程序的指令。子程序不能用跳转指令跳转到主程序,只能用返回指令返回。

2) 子程序开始标志是标号,子程序结束标志是 RET 指令。

3) 程序中标号具有惟一性。在子程序中,标号必须位于其第一行的首位,标号与其分隔号“:”之间应有空格。标号应是字母开始的标识符。标号的字符长度与系统有关。

4) SFC 编程语言中,在 ACTION...END_ACTION 结构内使用 JMP 指令时,操作数应是在

同一结构中的一个标号。

【例 3-9】 跳转指令的示例。

```
LD      AUTO1      ( * 存取布尔变量 AUTO1 * )
JMP      AUTOPROG   ( * 如果 AUTO1 为 1,则跳转到 AUTOPROG 子程序 * )
JMP      MANPROG    ( * 如果为 0,则跳转到 MANPROG 子程序 * )
```

例 3-9 用于自动和手动控制的程序切换。当 AUTO1 开关切到自动位置,则 AUTO1 为 1, 跳转指令 JMP 指令在当前值为 1(即 AUTO1 为 1)时,条件执行跳转操作,因此,程序跳转到 AUTOPROG 子程序,即执行在自动条件下的有关程序。当跳转条件不满足时,执行 JMP 指令, 因此,程序跳转到 MANPROG 子程序,即执行手动操作时的有关程序。须注意,AUTOPROG 和 MANPROG 是子程序的标号,不是程序名。

(8) 调用指令

IEC 61131-3 标准的调用指令是 CAL 指令。编程语言的格式如下:

```
CAL 操作数 ( * 调用操作数表示的函数、功能块或程序 * )
```

调用指令是十分重要的指令。通过执行该指令,可以调用函数、功能块和程序,使程序结构简化,程序描述清晰。其他调用格式见下述。

CAL 是 Call 的缩写,表示调用。也可不列出 CAL,而用函数或功能块的参数调用,见下述。CAL 指令的操作数是函数名或功能块实例名。实例名中的参数用逗号分隔。

(9) 圆括号指令

IEC 61131-3 标准采用圆括号对指令进行修正,即进行优先执行的操作。

左圆括号“(”用于将当前结果存储器内容压入堆栈,并将操作符的操作命令存储,这时,堆栈的其他内容下移一层。右圆括号“)”用于将堆栈最上层的内容弹出,并与当前结果存储器内容进行相应的操作(根据存储的操作命令),操作结果存放在当前结果存储器内。这时,堆栈的其他内容上移一层。因此,圆括号被称为指令的延迟操作,它产生的瞬时结果不影响当前结果存储器。须注意,指令和注释中的圆括号应采用英文模式的圆括号“(”和“)”,不能采用中文模式下的圆括号“(”和“)”。表 3-7 是圆括号的表达特性。

表 3-7 圆括号的表达特性

序 号	描述/示例	
1	圆括号表达开始于显式操作符	AND(LD %IX1 (* 见注 *) OR %IX2)
2	圆括号表示(短格式)	AND(%IX1 (* 见注 *) OR %IX2)

注:在格式 1,LD 操作符可被修改或 LD 操作可由其他操作或功能调用来替代。

【例 3-10】 圆括号对算术运算操作的修正。

```
LD  X      ( * 将 X 送当前结果存储器 * )
ADD ( B    ( * X 压入堆栈,延迟进行加,当前结果存储器内容为 B * )
      MUL ( C      ( * B 压入堆栈,延迟进行乘,当前结果存储器内容为 C * )
```

```

        ADD D      ( * 将 C 加上 D,当前结果存储器内容为 C + D * )
    )              ( * 将 C + D 乘以从堆栈弹出的 B,当前结果存储器内容为 B * (C + D) * )
)                ( * 将 B * (C + D)加上从堆栈弹出的 X,当前结果存储器内容为 X + B * (C + D) * )
ST  A1           ( * 将当前结果存储器内容 X + B * (C + D)送 A1 * )

```

从例 3-10 可见,整个操作过程中,数据类型需保持一致。此外,数据类型被传递。操作从最里层的圆括号开始,逐层向外操作,直到最外层圆括号。例 3-10 中,原数据类型是整数,如果运算过程中有实数,则需将数据类型全部设置为实数,或将数据类型转换为所需数据类型。通常,使用数据类型的过载属性来简化程序。

【例 3-11】 圆括号对逻辑运算操作的修正。

```

LD  %IX1.1       ( * 将第 1 输入单元第 1 位内容送当前结果存储器 * )
AND ( %IX1.2      ( * 第 1 输入单元第 1 位内容压入堆栈,当前结果存储器内容为第 2 位内容 * )
    OR ( %IX1.3    ( * 第 2 位内容压入堆栈,当前结果存储器内容为第 3 位内容 * )
AND %IX1.4        ( * 第 3 位内容与第 4 位内容与运算的结果存入当前结果存储器 * )
    )              ( * 堆栈原第 2 位内容与当前结果存储器内容或运算结果存当前结果存储器 * )
    )              ( * 堆栈原第 1 位内容与当前结果存储器内容与运算结果存当前结果存储器 * )
ST  SS            ( * 将当前结果存储器内容送 SS * )

```

采用括号指令可方便地实现传统 PLC 中的程序块串联和并联操作。

【例 3-12】 圆括号在程序块并联中的应用。

```

LD  FALSE        ( * 读取 FALSE,作为当前值 * )
OR( %IX0.0        ( * 读取 %IX0.0 的值 * )
    AND %IX0.1     ( * 与 %IX0.1 的内容进行与运算 * )
    )              ( * 运算结果压入堆栈 * )
OR( %IX0.2        ( * 读取 %IX0.2 的值 * )
    AND %IX0.3     ( * 与 %IX0.3 的内容进行与运算 * )
    )              ( * 运算结果与堆栈的内容进行或运算 * )
ST  STOP          ( * 最终结果存放在 STOP 变量 * )

```

程序要用 LD 开始。为此示例中,可用 LD FALSE(或 TRUE,根据后续指令确定)实现。两个 OR 开始的指令是两个程序块,是两个触点串联的程序,最后经或运算后,将运算结果存放到 STOP 变量中。

在数学运算中,圆括号具有与括号相类似的功能,即括号外的操作被延迟执行。

【例 3-13】 圆括号的延迟功能。

```

LD  A            ( * 读取变量 A 的值 * )
ADD B            ( * 加上 B,将结果 A + B 存放到当前结果存储器 * )
MUL( C           ( * 对 C 的乘被延迟,即 C 被压入堆栈,乘运算操作被存储 * )
SUB D            ( * 将 C-D 的结果放在堆栈 * )
    )            ( * 存储的乘运算被执行,运算结果存当前结果存储器 * )

```

例 3-14 中,由于左圆括号“(”,其右侧的乘运算操作被延迟到右圆括号“)”,因此,运算结果是 $(A + B) * (C - D)$ 。

下面的示例说明当前结果存储器和堆栈之间的关系。

【例 3-14】 当前结果存储器和堆栈的数据关系。

```
LD    A      (* 读取变量 A 的值,并存在当前结果存储器 *)
ADD(  B      (* 延迟加,读取的 B 被压入堆栈 *)
MUL(  C      (* 延迟乘,读取的 C 被压入堆栈 *)
SUB D      (* C-D 运算,并存回堆栈 *)
)          (* 堆栈内数据 B 弹出,乘运算被执行 *)
)          (* 当前结果存储器与堆栈内数据 A 弹出,加运算被执行 *)
```

例 3-14 中的堆栈内数据和当前结果存储器内容如表 3-8 所示。

表 3-8 堆栈内数据和当前结果存储器内容的变化

指 令	1	2	3	4	5	6
当前结果存储器	A	A	A	A	A	$A+B*(C-D)$
堆栈 S-1	-	B	B	B	$B*(C-D)$	-
堆栈 S-2	-	-	C	$C-D$	-	-

对较复杂的运算关系,用梯形图编程语言实现较方便。从圆括号指令内进行跳转会产生不可预测的结果,因此应注意避免。

3.2.2 函数和功能块

1. 函数和函数调用

指令表编程语言中,函数的调用比较简单。

(1) 函数调用方法

指令表编程语言提供 3 种不同的 CAL 操作符调用格式,函数调用的编程格式和示例如表 3-9 所示。它可表示为下列两种调用格式。

表 3-9 函数调用的编程格式和示例

方 式	编 程 格 式		示 例	
单参数	LD	参数	LD	0.5 (* 读取 0.5 弧度 *)
	函数名		COS	(* 调用 COS 函数 *)
	ST	返回值	ST	A (* 运算结果 0.87758 存放在 A 变量 *)
双参数	LD	参数 1	LD	A (* 读取变量 A 的值 *)
	函数名	参数 2	ADD	B (* 与变量 B 的值相加 *)
	ST	返回值	ST	C (* 运算结果,即返回值存放在 C *)
多参数	LD	参数 1	LD	G (* 读取布尔变量 G 的值 *)
	函数名	参数 2,...,参数 n	SEL	IN0,IN1 (* 根据 G 选择 IN0 或 IN1 作为返回值 *)
	ST	返回值	ST	A (* 返回值存放在 A *)

带非形参表的函数调用。编程语言的格式如下：

函数名 非形参,非形参,...,非形参

带形参表的函数调用。编程语言的格式如下：

函数名 (第一形参 := 实参, ..., 最后形参 := 实参)

(2) 函数调用时的注意事项

1) 非形参表即实参表,是实际应用的参数组成的参数表。形参表是函数中定义的参数,在实际应用时必须用实际参数代入。

2) 非形参的函数调用指令中,第一个参数是当前结果存储器存储的结果。因此,在函数调用前,用有关指令将被调用函数的第一参数送当前结果存储器。

【例 3-15】 非形参的函数调用示例。

```
LD 1      (* 将 1 送当前结果存储器 *)
ADD 2,3,4  (* 将 2,3,4 与当前结果存储器内容 1 相加,结果 10 送当前结果存储器 *)
ST A      (* 当前结果存储器内容 10 送变量 A *)
```

例 3-15 中,用 ADD 函数直接实现多个数值的相加运算。因此,与传统 PLC 的相加运算比较,程序被简化。须注意,一些标准软件产品只允许有一个操作数。例如,ADD 2。这时,可用几行相加指令实现,即 ADD 2;ADD3;ADD4。也可建立用户的函数,例如,建立 ADDN 函数,实现多个变量的相加。

【例 3-16】 多个变量的相加。

建立用户函数 ADDN 如下:

```
FUNCTION ADDN ;INT
  VAR_INPUT
    A1,B1,B2,B3 :INT ;
  END_VAR
  LD  A1
  ADD B1
  ADD B2
  ADD B3
  ST  ADDN
END_FUNCTION
```

其中,A1、B1、B2、B3 和 ADDN 都是整数数据类型。

计算时可直接调用 ADDN 函数(在建立 ADDN 时应声明其返回值的数据类型是整数数据类型)。程序如下:

```
LD 1
ADDN 2,3,4
ST A
```

【例 3-17】 非形参的函数调用示例。

```
LD 1.0      (* 将 1.0 读入当前结果存储器,作为函数 LIMIT 的最小值参数 *)
LIMIT B,5.0 (* 将 B 作为函数 LIMIT 的输入信号,5.0 作为 LIMIT 的最大值 *)
ST A        (* 调用函数 LIMIT 的返回值输出到 A *)
```

例 3-17 中,调用 LIMIT 函数,第一参数用 LD 读入,其他参数用调用函数指令实现,各参

数用逗号分隔。ST 指令将函数调用后的返回值输出到 A。

3) 函数调用与指令的区别是函数可有多个参数。示例中,ADD 作为函数调用时,可附多个用逗号分隔的参数,程序简单,描述清晰。ADD 作为指令使用时,对上述示例,要用圆括号实现,程序复杂,并且程序所占存储空间大。须注意,函数没有修正符。

4) 函数调用时,数据类型应匹配。例如,调用 LIMIT 函数运算时,如果,变量 A 是整数,B 是实数,C 是双整长,则需先转换为实数数据类型,再进行函数 LIMIT 的调用。

【例 3-18】 非形参的函数调用示例。

```
LD      A      (* 将整数 A 读入当前结果存储器 *)
INT_TO_REAL      (* 调用整数到实数的类型转换函数 *)
ST      AR      (* 将整数 A 转换为实数,并存放在 AR *)
LIMIT  B,5.0    (* 将 B 作为函数 LIMIT 的输入信号,5.0 作为 LIMIT 的最大值 *)
ST      CR      (* 调用限幅函数后的实数结果存放在 CR *)
REAL_TO_DINT     (* 调用实数到双整长数的类型转换函数 *)
ST      C      (* 转换后结果是双整长数,存放在 C *)
```

例 3-18 中,第 3 条和第 5 条指令可以不编入,示例仅用于说明数据类型的变化。

5) 函数调用时,只有函数返回值,它可用于后续操作。但须注意,函数返回值通常与该函数输入具有相同的数据类型。必要时,为进行后续操作需要进行数据类型的转换。

6) 调用带形参的函数时,应将实参代替形参,可用赋值语言对形参赋值。

【例 3-19】 带形参的函数调用示例。

```
INSERT (
  IN1 := 'ABC',
  IN2 := 'XYZ',
  P := 2
)
```

例 3-19 中,调用 INSERT 函数,第一参数是 IN1,用实参'ABC'赋值给 IN1,同样,用实参'XYZ'赋值给 IN2,用实参 2 赋值给 P,调用该函数后,当前结果存储器存放的字符串是'ABXYZC'。需注意,一些软件系统不一定具有带形参的函数调用性能。

7) 为简化,在调用带形参的函数时,可先将第一参数存在当前结果存储器。然后,直接调用有关函数。

【例 3-20】 带非形参的函数调用示例。

```
LD 'ABC'          (* 读取 INSERT 函数的输入 IN1 *)
INSERT 'XYZ',2    (* 调用 INSERT *)
ST  A             (* 调用 INSERT 的返回值存放到 A 字符串变量 *)
```

8) 可以添加 EN 和 ENO 参数。可将函数状态传递到下一个指令。须注意,对 ENO 参数的输出需用赋值语言。一些软件系统不一定具有 EN 和 ENO 的函数调用性能。

【例 3-21】 带 EN 和 ENO 参数的函数调用示例。

```
INSERT(
  EN:=CON,
```

```

IN1 := 'ABC',
IN2 := 'XYZ',
P := 2,
ENO => TTT
)

```

例 3-21 中,增加的 EN 参数表示当布尔量 CON 为 1 时,才能调用插入 INSERT 函数,运行后,ENO 的布尔值 1 被赋值给 TTT 参数,用于后续指令。

不同制造商的软件系统并非都支持所有的函数调用格式。

2. 功能块和功能块调用

(1) 功能块调用方法

指令表编程语言中,功能块调用有下列 4 种格式。

1) 带非形参表的功能块调用。编程语言的格式如下:

```
CAL 功能块实例名(非形参表)
```

2) 带形参表的功能块调用。编程语言的格式如下:

```
CAL 功能块实例名(形参表)
```

3) 带参数读/存储的功能块调用。编程语言的格式如下:

```
CAL 功能块实例名
```

4) 使用功能块输入操作符的功能块调用。编程语言的格式如下:

```
参数名 功能块实例名
```

并非所有软件系统能够实现上述 4 种编程格式。应根据实际可编程控制器系统所提供的编程方法进行编程。

(2) 功能块调用示例

下面以调用 TON 功能块为例说明 4 种功能块调用的方法。程序中变量声明见例 3-22。

【例 3-22】 4 种功能块调用方法的比较。

4 种功能块调用方法的变量声明相同,编程语言的格式如下:

```

VAR
    AT  %IX1.1;BOOL;      (* 设置 TON 功能块的输入 *)
    OUT1 ;BOOL;           (* 设置 TON 功能块的输出 *)
    TMR1 ;TON;            (* 定义标准功能块 TON 的功能块名为 TMR1 *)
    TV  ;TIME;            (* 设置定时器当前计时输出变量 TV 的数据类型为 TIME *)
END_VAR

```

1) 带非形参表的功能块调用。程序如下:

```

CAL  TMR1(%IX1.1, T#500ms, OUT1, TV);
(* 用非形式参数表调用 TON 的实例 TMR1,形参表内变量的顺序应与功能块 TON 一致 *)
(* 下面的程序用于调用 TMR1 实例后的输出送有关变量 *)
LD   TMR1.Q      (* 读取定时器 TON 实例 TMR1 的输出 Q *)

```

```

ST    OUT1      ( * 存放到 OUT1 * )
LD    TMR1. ET  ( * 读取定时器 TON 实例 TMR1 的 ET * )
ST    TV        ( * 存放到 TV * )

```

2) 带形参表的功能块调用。程序如下:

```

CAL TMR1 ( PT := T#500ms, IN := %IX1.1, Q = > OUT1, ET = > TV );
( * 带形参表的功能块调用时,各形参的名称被显示,因此,其先后的顺序可用户确定,见例 3-22 * )
( * 由于输出参数可直接赋值给有关变量,因此,不像上述调用方法要有输出的有关程序 * )
( * 输出用 = > 操作符表示,它将该操作符左面输出的值赋值给其右面的变量 * )

```

3) 带参数读/存储的功能块调用。程序如下:

```

LD    %IX1.1    ( * 读取%IX1.1的数据 * )
ST    TMR1. IN  ( * 赋值给功能块 TON 实例 TMR1 的 IN 变量 * )
LD    T#500ms   ( * 读取时间数据 500ms * )
ST    TMR1. PT  ( * 赋值给功能块 TON 实例 TMR1 的 IN 变量 * )
CAL    TMR1     ( * 调用功能块实例 TMR1 * )
( * 下面的程序用于调用 TMR1 实例后的输出送有关变量,与第一种方法相同 * )
LD    TMR1. Q   ( * 读取定时器 TON 实例 TMR1 的输出 Q * )
ST    OUT1      ( * 存放到 OUT1 * )
LD    TMR1. ET  ( * 读取定时器 TON 实例 TMR1 的 ET * )
ST    TV        ( * 存放到 TV * )

```

4) 使用功能块输入操作符的功能块调用。这种调用功能块实例的程序如下:

```

LD    %IX1.1    ( * 读取%IX1.1的数据 * )
IN    TMR1      ( * 直接赋值给功能块 TON 实例 TMR1 的 IN 变量 * )
LD    T#500ms   ( * 读取时间数据 500ms * )
PT    TMR1      ( * 直接赋值给功能块 TON 实例 TMR1 的 IN 变量,并调用 TMR1 * )
( * 下面的程序用于调用 TMR1 实例后的输出送有关变量,与第一种方法相同 * )
LD    TMR1. Q   ( * 读取定时器 TON 实例 TMR1 的输出 Q * )
ST    OUT1      ( * 存放到 OUT1 * )
LD    TMR1. ET  ( * 读取定时器 TON 实例 TMR1 的 ET * )
ST    TV        ( * 存放到 TV * )

```

调用功能块时,功能块实例名与功能块参数之间需用小数点分割。

(3) 功能块调用的注意事项

1) 调用功能块时,如果参数不存在,系统自动取初始值或在前面程序中已经设置的最新值。

2) 调用操作符与修正符 C 结合时表示当前结果存储器内容为真时执行调用。一个条件功能块调用时,所有参数表内的参数在条件为真时,仅在调用时一起执行。调用操作符与修正符 NC 结合时表示当前结果存储器内容为假时执行调用。须注意,N、C 等修正符与操作符之间不应有空格。

3) 采用功能块输入操作符进行功能块调用时,只允许对标准功能块进行。其他 3 种调用方法也适用于衍生功能块的调用。输入操作符与调用的功能块如表 3-10 所示。

表 3 - 10 功能块输入操作符和调用的功能块

功能块类型		输入操作符	功能块类型		输入操作符
SR	置位优先双稳	S1,R	CTD	减计数器注	CD,PV
RS	复位优先双稳	S,R1	CTUD	加减计数器注	CU,CD,R,PV
R_TRIG	上升沿变沿检测	CLK	TON	延时闭合定时器	IN,PT
F_TRIG	下降沿变沿检测	CLK	TOF	延时断开定时器	IN,PT
CTU	加计数器	CU,R,PV	TP	脉冲输出定时器	IN,PT

注:CTD 和 CTUD 计数器的 LD 输入已功能地包含在 PV 中,因此,LD 不作为输入操作符。

4) 形参和非形参变量在组成变量表时的特性如表 3 - 11 所示。

表 3 - 11 形参和非形参变量的特性

调用类型 (Invocation type)	变量配置 (Variable assignment)	变量阶次 (Variable order)	变量个数 (Number of variables)
形参	是	任意	任意
非形参	否	固定	固定

5) 需要使用 EN 和 ENO 时,应采用带形参表的功能块调用方法。

【例 3 - 23】 使用 EN 和 ENO 时功能块的调用。

```
VAR_INPUT
    EN:BOOL;
END_VAR
VAR_OUTPUT
    ENO,TEMPL:BOOL;
END_VAR
VAR
    TMR :TON ;
END_VAR
TMR (EN:=TRUE,IN:=%IX1.1,PT:=T#5S, ENO=>TEMPL);
```

6) 在形参表中,输出用赋值操作符“=>”将输出变量的值赋值到变量。例如,带形参表的功能块调用程序中的 Q=>OUT1,ET=>TV;例 3. 23 中的 ENO=>TEMPL 等。

3. 2. 3 示例

1. 称重显示函数的示例

称重显示函数的示例用于说明如何建立用户的称重函数 WEIGH,及用函数 WEIGH 来实现称重显示。

(1) 称重控制系统的要求

称重装置将物料称重后的毛重数据(BCD 码数据)存储在 PLC 存储器,称重函数将毛重减去皮重,并将净重转换为 BCD 码,用 BCD 码形式显示。

假设:毛重变量:GROSS_WEIGHT;皮重变量:TARE_WEIGHT。

为控制称重信号的执行,需设置手动信号作为称重命令,用布尔变量 START1 表示。

数据类型设置为:毛重变量是 BCD 数,用 WORD 数据类型。皮重变量是实数,用 REAL 数据类型。

(2) 称重函数 WEIGH 的编程

WEIGH 函数的编程分为声明部分和函数本体两部分。

1) 声明部分。WEIGH 函数的声明部分包括对如下 3 个输入变量的声明:

```
FUNCTION  WEIGH ;WORD          ( * 用 BCD 编码的字作为该函数的返回值 * )
  VAR_INPUT
    CROSS_WEIGHT:  WORD;  ( * 毛重,用 BCD 编码 * )
    START1:        BOOL;   ( * 称重开始手动信号 * )
    TARE_WEIGHT:    REAL;   ( * 皮重信号,实数 * )
    EN:             BOOL;   ( * 函数的使能输入信号 * )
  END_VAR
  VAR_OUTPUT
    ENO:            BOOL;   ( * 函数的使能输出信号 * )
  END_VAR
  函数本体    ( * 见下述 * )
END_FUNCTION
```

用户函数名是 WEIGH,有 3 个输入变量。其中,毛重信号来自称重装置,皮重信号由操作员输入到特定地址,称重开始信号 START1 由操作员输入。

为便于控制函数 WEIGH 的执行,设置 EN 和 ENO 信号,它们分别在输入和输出变量中声明,数据类型是 BOOL。函数 WEIGH 返回值的数据类型是 WORD。

2) 函数 WEIGH 的图形表示。根据上述,WEIGH 函数的图形符号如图 3-4 所示。

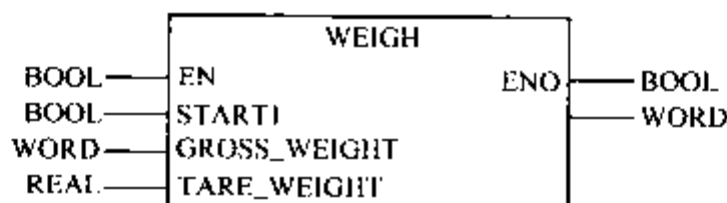


图 3-4 WEIGH 函数的图形符号

3) 函数本体部分。函数本体是该函数编程的主体。函数 WEIGH 本体程序如下:

```
LD      START1      ( * 读取称重开始信号 * )
JMPC    WEIGHTING    ( * 需称重时跳转到 WEIGHTING 执行 * )
ST      ENO          ( * 跳转条件不满足时,将 ENO 设置为 0 * )
RET      ( * 返回 * )
WEIGHTING: LD      GROSS_WEIGHT ( * 读毛重信号,BCD 数 * )
          BCD_TO_REAL ( * 毛重信号转换为实数信号 * )
          SUB      TARE_WEIGHT ( * 减去皮重 * )
          REAL_TO_BCD ( * 净重信号转换为 BCD 数据 * )
          ST      WEIGH ( * 作为函数 WEIGH 的返回值 * )
```

函数体中,用标号 WEIGHTING 表示称重过程。当称重开始信号 START1 为 1 时,跳转到该标号执行。如果称重开始信号为 0,则 ENO 被置 0,程序返回。

WEIGHTING 称重过程。先读取毛重变量 GROSS_WEIGHT 信号(BCD 码表示),并转换为实数信号,减去皮重变量 TARE_WEIGHT 信号后的净重信号再被转换为 BCD 编码,并将该 BCD 数据作为函数 WEIGH 的返回值,用于显示等。

2. 计算整数相除的商和余数的功能块示例

功能块 DIVREMBK 是用户编写的衍生功能块。它用于计算两个整数相除的商和余数。当除数为零时,功能块输出 DIVERR 为真。

(1) 运算要求

该功能块有两个输入,即 DIVIDEND 变量表示被除数, DIVISOR 变量表示除数,它们都是整数数据类型。有 3 个输出,即 QUOTIENT 变量表示商, DIVREM 变量表示余数,它们是整数数据类型。 DIVERR 变量表示除数为零的出错标志,数据类型是布尔量。

(2) 功能块 DIVREMBK 的变量声明

根据运算要求,变量声明段如下:

```
FUNCTION_BLOCK  DIVREMBK
VAR_INPUT
    DIVIDEND:  INT;  ( * 被除数,整数数据类型 * )
    DIVISOR:   INT;  ( * 除数,整数数据类型 * )
END_VAR
VAR_OUTPUT
    QUOTIENT:  INT;  ( * 商,整数数据类型 * )
    DIVREM:    INT;  ( * 余数,整数数据类型 * )
    DIVERR:    BOOL; ( * 除数为零标志,布尔数据类型 * )
END_VAR
( * 功能块本体程序 * )
END_FUNCTION_BLOCK
```

(3) 功能块的图形形式

功能块 DIVREMBK 的图形形式如图 3-5 所示。

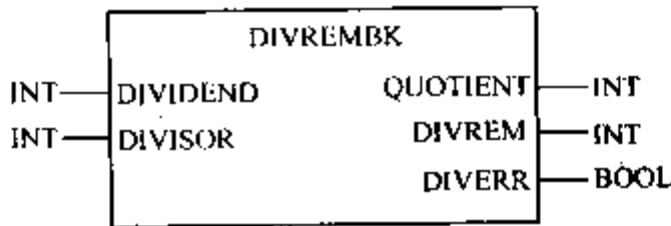


图 3-5 DIVREMBK 功能块的图形形式

(4) 功能块本体

功能块本体要先判别除数是否为 0,如果为 0 则 DIVERR 置 1,反之,进行相除运算过程。用指令表编程语言编写的功能块本体程序如下:

```

LD      DIVISOR      ( * 读取除数 * )
EQ      0             ( * 与零比较,是否等于零? * )
JMPC    ERROR        ( * 如果满足,则跳转到 ERROR 标号处执行 * )
LD      DIVIDEND     ( * 除数不为零时,读被除数 * )
DIV     DIVISOR      ( * 除以除数 * )
ST      QUOTIENT     ( * 商存放在 QUOTIENT * )
LD      DIVIDEND     ( * 读被除数 * )
ERROR:  MOD          DIVISOR      ( * 对除数进行模除 * )
ST      DIVREM       ( * 余数存放在 DIVREM * )
JMP     END          ( * 无条件跳转到结束标号 END * )
LD      0             ( * ERROR 标号开始的子程序,读取整数值零 * )
ST      QUOTIENT     ( * 将商置 0 * )
ST      DIVREM       ( * 将余数置 0 * )
LD      0             ( * 须注意这里的 0 是布尔量,不是整数 0 * )
STN     DIVERR       ( * 取反后,使 DIVERR 置 1 * )
END:    RET          ( * 返回 * )

```

计算余数可采用模除方法,也可采用被除数减去商与除数之积的方法计算。示例中采用模除方法计算,程序较简单。读者可自行编写有关程序进行比较。此外,当除数为零时,将商和余数置零,并将出错标志 DIVERR 置 1。由于,在置零后,当前值在输出指令执行时不发生变化,因此,可用多个(示例中是两个)输出指令实现置 0 操作,简化了程序。须注意, DIVERR 是布尔量,因此,需重新设置当前值,然后,才能将 DIVERR 置 1。

(5) 功能块调用

可调用衍生功能块 DIVREMBK,实现两个整数的相除。下面是一个应用示例。

DIVREMBK 的实例名是 DDD,程序计算 30 除以 4 的商 A 和余数 B,出错标志存放在 C。程序如下:

```

PROGRAM P1                      ( * 程序名 P1 * )
VAR
  A,B :INT;                     ( * 变量 A 和 B 是整数数据类型 * )
  C :BOOL;                      ( * 变量 C 是布尔数据类型 * )
  DDD :DIVREMBK;                ( * DDD 是功能块 DIVREMBK 的实例名 * )
END_VAR
LD      30                      ( * 读取 30 * )
ST      DDD.DIVIDEND           ( * 存放在 DDD 中作为被除数 * )
LD      4                       ( * 读取 4 * )
ST      DDD.DIVISOR            ( * 存放在 DDD 中作为除数 * )
CAL     DDD                    ( * 调用 DDD 功能块实例 * )
LD      DDD.QUOTIENT           ( * 读取 DDD 功能块的商 * )
ST      A                      ( * 将商存放到 A 变量 * )
LD      DDD.DIVREM             ( * 读取 DDD 功能块的余数 * )
ST      B                      ( * 将余数存放到 B 变量 * )
LD      DDD.DIVERR             ( * 读取 DDD 功能块的出错标志 * )

```

```

      ST      C
END_PROGRAM

```

(* 将出错标志存放到 C 变量 *)

程序运行结果,A 变量的值是 7(商),B 变量的值是 2(余数),C 变量的值是 0。

3. 循环计算的示例

(1) 控制要求

计算 1 到 10 的累加和及阶乘的程序,可采用 JMP C 指令实现。

(2) 变量声明

变量声明如下:

```

VAR
  A,SUM,FACTORIAL :DINT; ( * 变量 A、FACTORIAL 和 SUM 是双整长整数数据类型 * )
END_VAR

```

(3) 程序本体

循环累加计算程序如下:

```

      LD      DINT#1      ( * 读取双整长 1 ,作为 A 和 FACTORIAL 的初始值 * )
      ST      A           ( * 置 A 为 1 * )
      ST      FACTORIAL   ( * 置 FACTORIAL 为 1 * )
      LD      DINT#0      ( * 读取双整长 0 ,作为 SUM 的初始值 * )
      ST      SUM         ( * 置 SUM 为 0 * )
      LD      SUM         ( * 循环开始,读取 SUM * )
      ADD     A           ( * 加 A,进行累加运算 * )
      ST      SUM         ( * 累加和存放在 SUM * )
STRT:  LD      FACTORIAL   ( * 循环开始,读取 FACTORIAL * )
      MUL     A           ( * 乘 A,进行阶乘运算 * )
      ST      FACTORIAL   ( * 阶乘结果存放在 FACTORIAL * )
      LD      A           ( * 读取 A * )
      ADD     DINT#1      ( * 加 1 运算 * )
      ST      A           ( * 存放到 A * )
      LE     DINT#10      ( * 如果 A 的累加值小于等于 10,则当前值置 1 * )
      JMP C     START     ( * 当前值为 1,则进行循环,跳转到 START * )
      RET           ( * 返回 * )

```

上述程序可简单地计算累加和及阶乘,运算结果在 SUM 中存放 55,在 FACTORIAL 中存放 3628800。须注意,当运算结果大于变量设置的数据类型允许范围时,结果被置 0。例如,如果计算 1 到 50 的累加和及阶乘时,阶乘的结果超过双整长整数的允许范围,这时,计算结果被置 0。为此,可将变量的数据类型设置为实数。

本程序说明用跳转类指令和比较指令可实现高级编程语言中的条件语句功能。

4. 计算两点之间距离的示例

(1) 控制要求

数控机床中,常常需要计算两个操作点之间的距离。假设两点的位置由 X 和 Y 坐标给出。根据三角学知识,两点间的距离为

$DISTANCE := \sqrt{(X1-X2)^2 + (Y1-Y2)^2}$;

如果两点距离小于 TMax, 函数引入 ENO, 表示已经成功计算移动距离, 如果两点距离超过 TMax, 则计算产生一个超出被控机器范围的距离, 这时, ENO 不被置位。

(2) 函数 DISTANCE

编写函数 DISTANCE 如下:

```

FUNCTION DISTANCE :REAL
  VAR_INPUT
    X1,X2,Y1,Y2 :REAL; (* X,Y 坐标 *)
    Tmax :REAL; (* 设置的最大距离 *)
  END_VAR
  VAR
    Temp :REAL; (* 既可设置为内部变量,也可设置为暂存变量 *)
  END_VAR
  LD Y1 (* 取坐标 Y1 的值 *)
  SUB Y2 (* 减坐标 Y2 的值 *)
  ST Temp (* Y1 - Y2 的值暂存在 Temp 变量中 *)
  MUL Temp (* Temp 变量自乘 *)
  ADD(X1 (* 结果延迟进行加,取坐标 X1 的值 *)
  SUB X2 (* 减坐标 X2 的值 *)
  ST Temp (* X1 - X2 的值暂存在 Temp 变量中 *)
  MUL Temp (* Temp 变量自乘 *)
  ) (* 结果(X1 - X2)^2 与堆栈中的(Y1 - Y2)^2 相加,存入当前
    结果存储器 *)
  CAL SQRT (* 当前结果存储器的内容进行开方 *)
  ST DISTANCE (* 开方的结果送 DISTANCE *)
  GT TMax (* 与 TMax 比较 *)
  JMP ERR (* 如果超过则跳转到 ERR *)
  S ENO (* 如果不超过,则 ENO 置位 *)
  RET (* 无条件返回 *)
ERR: RET (* ERR 标号,执行无条件返回 *)
END_FUNCTION

```

5. 最大值计算

(1) 控制要求

反应器有 3 个温度检测点,控制要求是按最大温度点进行控制,为此需计算最大值。

(2) 最大值计算程序

```

VAR_INPUT
  TT1,TT2,TT3 :REAL;
END_VAR
VAR_OUTPUT
  TMAX :REAL;
END_VAR

```

```
LD  TT1      ( * 读第一温度检测点温度,存当前结果存储器 * )
MAX  TT2      ( * 读第二温度检测点温度,与当前结果存储器比较,取大者存当前结果存储器 * )
MAX  TT3      ( * 读第三温度检测点温度,与当前结果存储器比较,取大者存当前结果存储器 * )
ST  TMAX      ( * 最大值从当前结果存储器传送到 TMAX * )
( * 后续程序用于控制 * )
```

3.3 结构化文本编程语言

3.3.1 结构化文本的表示

结构化文本编程语言是高层编程语言,类似于 Pascal 编程语言。它不采用低层的面向机器的操作符,而是用高度压缩的方式提供大量抽象语句来描述复杂控制系统的功能。它特别为工业控制应用而开发。

结构化文本编程语言是通用目的的高级语言,用于表示不同的行为类型,调用各种不同数据类型和行为。它特别适合复杂的算法计算。

结构化文本编程语言具有下列特点:

- 1) 编程语言采用高度压缩化的表达形式,因此,程序紧凑,结构清楚。
- 2) 强有力的控制命令流的结构。例如,选择语句,迭代循环语句等控制命令的执行。
- 3) 程序结构清晰,便于编程人员和操作人员的思想沟通。
- 4) 采用高级程序设计语言,可完成较复杂的控制运算,例如,递推运算等。
- 5) 执行效率较低。源程序要编译为机器语言才能执行,因此,编译时间长,执行速度慢。
- 6) 对编程人员的技能要求较高。需要有一定的高级编程语言知识和编程技巧。

1. 结构化文本的程序结构

结构化文本编程语言的程序由语句组成。语句由表达式和关键字等组成。图 3-6 是结构化文本编程语言的程序结构。

(1) 结构化文本编程语言的特点

结构化文本编程语言编写的程序是结构化的,它具有下列特点:

- 1) 在结构化编程语言中,没有跳转语句。它通过条件语句实现程序的分支。
- 2) 结构化编程语言中的语句用分号“;”分隔,一个语句的结束用一个分号。因此,一个结构化编程语言的语句可以分成几行编写,也可将几个语句编写在同一行,只需在语句结束用分号分隔即可。分号表示一个语句的结束,换行表示在语句中的一个空格。
- 3) 结构化编程语言的语句可以注释,注释的内容包含在符号“(* ”和“ *)”之间。
- 4) 一个语句中可有多个注释,但注释符号不能嵌套,即不能用(* (* 注释内容 *) *)。
- 5) 注释可以设置在语句的任何空格位置,其内容部分也可包含空格。

语句 A: = K(* 增益 *) * (E(* 偏差 *) + INTE(E)(* 偏差的积分项 *) * TS/TI); 中

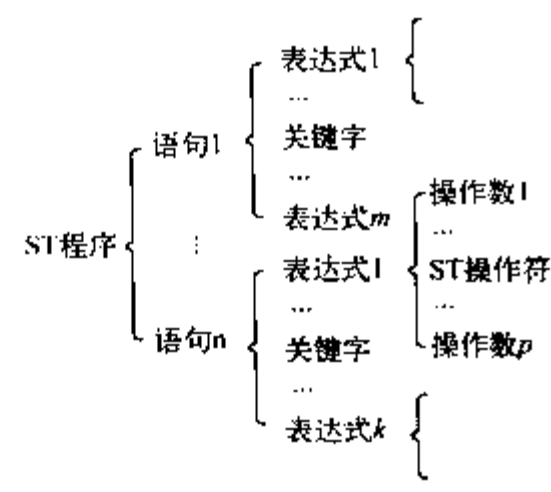


图 3-6 结构化文本编程语言的程序结构

有 3 个注释项,即(* 增益 *)、(* 偏差 *)和(* 偏差的积分项 *)。而 INTE(E)是用户定义的积分累积运算函数。E、K、TS 和 TI 分别是偏差变量、增益、采样时间和积分时间变量。

6) 结构化编程语言中的基本元素是表达式。

(2) 表达式

表达式是操作符和操作数的结合。

(3) 关键字

结构化文本编程语言的常用关键字见表 3 - 12。

表 3 - 12 结构化文本编程语言常用的关键字

说 明	关键字或表示方法	功 能 描 述	示 例
赋值操作符	: =	将操作符号右面表达式的值赋给左面的变量	A: = 1. 0;
功能块调用	功能块实例名 (参数表)	调用功能块实例名表示的功能块(带参数表)	TMR1(IN: = A, PT: = T#5s);
选择	IF	根据布尔条件的计算结果选择执行有关语句组	IF A > B THEN C: = 1. 2; ELSE C: = 2. 2; END_IF;
	CASE	根据表达式的值选择这些有关语句组	CASE A OF 1: C: = 1. 2; 2: C: = 1. 5; ELSE C: = 1. 8; END_CASE;
循环	FOR	根据初值、增量和终值,确定循环执行语句组	FOR I: = 2 TO 20 BY 2 DO A[I]: = 1 * 2. 0; END_FOR;
	WHILE	根据开始定义的循环条件确定执行语句组	WHILE 1 > 100 C: = C + 2; 1: = 1 + 1; END_WHILE;
	REPEAT	根据结束定义的循环条件确定执行语句组	REPEAT M: = M * 1. 2; J: = M + 2. 2; UNTIL M > 300. 0 END_REPEAT;
循环结束	EXIT	中断循环语句,跳出该循环执行过程	EXIT;
返回	RETURN	提供从函数、功能块或程序的出口	RETURN;
语句结束	;	表示该语句的结束,或表示空语句	; A: = 2; B: = A + 2;

关键字中没有跳转(JUMP)指令,由于它是非结构化的,因此不被采用,但它可用 IF 语句结构实现。

2. 结构化文本编程语言的表达式

结构化文本编程语言的表达式由操作符和操作数组成。表达式用于生成处理语句所需的数值。它是一个结构,是由若干变量和/或函数调用的组合来生成数值的各类语句的一部分。表达式用于计算或估计从其他变量或常数导出的数值。表达式通常产生一个特定数据类型的数值,它既可以是基本数据类型,也可以是衍生数据类型。一个表达式能够调用一个或多个操作符、变量和函数。

(1) 操作符

表 3-13 显示了结构化文本编程语言的操作符和优先级。

表 3-13 结构化文本编程语言的操作符和优先级

操作符	操作符功能	优先级	操作符	操作符功能	优先级
(表达式)	圆括号	最高	+, -	加,减	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></</div></div>

如表所示, 操作符的操作具有优先级。箭头表明操作优先级, 圆括号内的操作具有最高的操作优先级, 布尔或运算操作具有最低的操作优先级。

(2) 操作数

操作数可以是数据外部表达的数据文字、字符串文字、时间文字、单元素变量和多元素变量、函数调用和其他表达式。例如, 操作数可以是 23、'asd'、T#500ms、START1、ANALOG_IN [1, ..., 8]、SIN(A)、(A > B) & (C <= D) 等。

(3) 使用表达式的注意事项

1) 结构化文本编程语言中, 表达式的执行操作有先后之分。优先级高的操作符对应的操作先被执行, 优先级低的操作符对应的操作后被执行, 如表 3-13 所示。

【例 3-24】 优先级高的操作符对应的操作先被执行。

表达式 $5 * (3 + 2) - \text{LN}(3)$ 中, 先执行 $3 + 2$, 然后执行 $\text{LN}(3)$, 及 $5 * (3 + 2)$ 和 $-\text{LN}(3)$ 的操作。示例中, 圆括号内的 $+$ 操作符对应的 $3 + 2$ 最先进行, 其次, 是函数求值, 即计算 $\text{LN}(3)$, 随后, 才进行 $*$ 操作符对应的乘操作和 $-$ 操作符对应的减操作。

2) 对具有相同优先级的多个操作符, 根据表达式从左到右的次序执行程序。

【例 3-25】 表达式按从左到右的次序执行程序。

表达式 $60 \text{ MOD } 24 \text{ MOD } 10$ 的求值过程中, 按两个模除具有相同的优先级, 因此, 先进行 $60 \text{ MOD } 24$ 的运算, 结果是 12, 再进行 $12 \text{ MOD } 10$ 的求值, 结果为 2。如果要先进行后一个模除运算, 则表达式应改写为: $60 \text{ MOD } (24 \text{ MOD } 10)$, 即后一个模除结果为 4, 而最终模除结果为 0。

3) 当一个操作符具有多个操作数时, 根据从左到右的次序执行操作。

【例 3-26】 多个操作数时, 按从左到右的次序进行操作。

表达式 $\text{SIN}(A) * \text{COS}(B)$ 中, $*$ 操作符两边的操作是先计算 $\text{SIN}(A)$, 然后计算 $\text{COS}(B)$, 最后计算两者之积。

4) 多重圆括号的表达式中, 求值从最内层的圆括号开始, 并根据上述原则对圆括号内的表达式求值, 然后, 逐层向外, 直到获得最后结果。利用圆括号可使运算次序改变, 同时, 也使程序的可读性增强。

【例 3-27】 多重圆括号的表达式操作。

多重圆括号的表达式 $5 * (1 + 3 * (7 + 6) - 8)$ 计算时, 先计算最内层的 $7 + 6$, 结果 13 与 3

相乘后得 39,再用 1 + 39 减去 8 得 32,最后乘以 5 得 160。

5) 对布尔表达式的求值只需要求值到可决定最终结果的程度,这种求值方法可简化程序的执行。

【例 3-28】 布尔表达式的求值。

如果 $A > B$,则表达式 $(A > B) \text{ OR } (C < D)$ 的求值只需计算得 $(A > B)$ 即可确定结果为 1。

表达式 $(A < B) \& (C < D)$ 的求值也只需计算得 $(A < B)$ 即可确定结果为 0。

6) 函数调用给出函数的返回值。因此,函数调用是一个表达式。

【例 3-29】 函数调用。

$\text{INSERT}(\text{IN1} := 'ABC', \text{IN2} := 'XYZ', \text{P} := 2)$ 是调用函数 INSERT 的表达式。

$\text{SIN}(A) * \text{COS}(B)$ 是调用 SIN 和 COS 函数的表达式,并计算其乘积。

7) 功能块调用提供输出值,因此,功能块调用不是表达式,而是语句。

【例 3-30】 功能块调用。

$\text{RS}_1(\text{S} := \text{START}, \text{R1} := \text{STOP})$; 是调用功能块 RS 的实例 RS_1 的语句。

$\text{TON}_1(\text{IN} := \% \text{IX0.0}, \text{PT} := \text{T\#2S})$; 是调用功能块 TON 的实例 TON_1 的语句。

8) 函数调用与某些操作符操作具有相同求值结果。功能名和符号都可用于函数调用。例如, $3 + 5$ 与 $\text{ADD}(3, 5)$,前者用 + 操作符号将操作数 3 与 5 相加,后者调用 ADD 函数名。须注意,符号名和函数名标识符的操作优先级不同。

【例 3-31】 不同的操作次序。

表达式 $3 + 5 + \text{ADD}(2, 4)$ 先调用 ADD 函数进行 $2 + 4$ 的操作,然后,进行 $3 + 5$ 及 $+6$ 的操作。而表达式 $3 + 5 + 2 + 4$ 先进行 $3 + 5$,然后进行运算结果 $+2$,及 $+4$ 的操作。

9) 函数调用方法有非形参表函数调用和形参表函数调用两种。编程格式如下:

函数名(实参表)

函数名(形参表)

【例 3-32】 非形参表函数调用。

$\text{LIMIT}(4, \% \text{IX1.1}, 20)$; 用实参表表示函数 LIMIT 的参数。这时,各参数的先后次序应与该函数的参数位置一致。即对应地,4 被赋值给 MN 变量, $\% \text{IX1.1}$ 信号送 IN 变量,而 20 被赋值给 MX 变量。

【例 3-33】 形参表函数调用。

$\text{LIMIT}(\text{MN} := 4, \text{MX} := 20, \text{IN} := \% \text{IX1.1})$; 用形参表调用函数,因形参和实参之间用赋值语句建立了对应关系,因此,输入次序可任意。例如,示例中 MX 的次序被提前。

10) 函数调用时,不允许采用全局变量,因为,全局变量在程序执行过程中,其数值可能变化。而函数的性能是相同输入参数下应具有相同的返回值,由于采用全局变量后会影响到返回值,因此,禁止采用全局变量。

11) 函数调用时,如果所有形参具有过载属性,即是一般数据类型时,所有实参也应具有相同的数据类型。必要时,应调用数据类型转换函数进行数据类型的转换。例如,表达式 $\text{INT_TO_REAL}(A + B)$ 表示 A 与 B 是整数,运算结果转换为实数数据类型,它被用于作为其他表达式的操作数。

12) 如果除数为零、操作数是不正确的数据类型、或数值求值的结果超过它的数据类型值

的允许范围,则被作为出错处理。

3.3.2 语句

结构化文本编程语言中的语句有 4 种类型。即赋值语句、函数和功能块控制语句、选择语句和循环语句,它们以分号作为语句的结束标志。

1. 赋值语句

(1) 赋值语句格式和功能

赋值语句(Assignment Statement)用于将赋值操作符号右侧表达式计算的值赋予在左侧的单元或多元素变量。赋值语句格式为

变量 := 表达式;

其中,“:=”表示赋值操作符。

【例 3-34】 赋值语句示例。

```
TYPE MULTIVAR                (* 用类型名 MULTIVAR 定义类型变量 *)
STRUCT
  VAR1:INT;  VAR2:REAL; (* 设置结构变量 VAR1 和 VAR2 *)
END_STRUCT
END_TYPE
VAR
  A:INT;
  B:ARRAY [1..8] OF INT;
  C:REAL:=20.2;
  D:MULTIVAR(VAR1:=5,VAR2:=2.0);
  E:MULTIVAR;
END_VAR
A:=15;                        (* 对变量 A 赋值 15 *)
B[1]:=A*3;  B[2]:=A;          (* 分别根据表达式对数组变量 B 的两个下标变量赋值 *)
A:=REAL_TO_INT(C);            (* 函数调用后赋值给 A *)
E:=D;                          (* 将 D 中各变量的值赋值给 E 变量 *)
```

示例说明将 15 赋值给整数变量 A,A*3 和 A 分别赋值给整数数据类型的数组变量 B,B 的其他数组元素被赋值为初始值 0。然后,用数据类型转换函数将实数 C 转换为整数,并赋值给 A,程序也将类型变量 MULTIVAR 的值赋值给类型变量 E。

(2) 赋值语句使用注意事项

1) 数据类型的匹配。如果赋值操作符两侧的数据类型不同,应调用数据类型转换函数。例如,A 是 INT 类型,C 是实数类型,则 C 的初始值被赋值给 A 时,应调用 REAL_TO_INT 的类型转换函数。

【例 3-35】 数据类型的匹配。

```
VAR
  A,B:INT;                    (* A 和 B 是整数数据类型的变量 *)
  C:REAL;                     (* C 是实数数据类型的变量 *)
```

END_VAR

C := INT_TO_REAL(A + B); (* A + B 运算结果是整数,要用 INT_TO_REAL 类型转换函数转换为实数 *)

2) 一行中的语句可以多于一个,例如,示例中,B[1] := A * 3; B[2] := A;可列写在同一行。一个语句也可以延续几行,但最大的允许长度是与执行有关的参数。

【例 3-36】 一行中可有多个语句。

VAR

A, B : INT := 5 ; (* A 和 B 是整数数据类型的变量,初始值为 5 *)

C, D : INT ; (* C 和 D 是整数数据类型的变量 *)

END_VAR

C := A * 3; D := A * B + C;

3) 结构化变量赋值时,(例 2-34 中 E := D;)赋值操作符右侧变量所有元素的当前值应对应地赋值给左侧变量的值(对应于变量 E 的 VAR1 用 D 的 5 替代,VAR2 用实数 2.0 替代)。

4) 变量声明中,可用赋值语句来设置变量的初始值。例 3-36 中,A, B : INT := 5;用于设置变量 A 和 B 的初始值为 5。

5) 函数调用时,函数返回值被赋值作为表达式的值,它应是最新的求值结果。

【例 3-37】 函数调用的返回值作为表达式的值。

STR := INSERT(IN1 := 'ABC', IN2 := 'XYZ', P := 2);

6) 函数体内,赋值语句的执行是强制性的,即它将表达式的求值结果强制赋予给函数的有关参数。同样,函数的数据类型应与表达式结果的数据类型匹配。

2. 函数和功能块控制语句

函数和功能块控制语句(Function and Function Block Control Statement)用于调用函数和功能块。它包括对函数和功能块的调用及调用函数和功能块体后的返回控制两部分。

(1) 函数控制语句

函数调用后将返回值作为表达式的值赋值给变量。例如,A := SIN(X);语句中,调用函数 SIN(X),并将返回值赋值给变量 A。语句格式如下:

变量 := 函数名(参数表)

参数表可以是形参表或实参表,必要时用赋值语句将实参赋值到形参。

【例 3-38】 函数控制语句示例。

A := LIMIT(IN := %IX0.0, MN := MIN1, MX := MAX1);

(* 调用 LIMIT 函数,计算结果赋值给 A *)

BB := ADD(2,3);

(* 调用 ADD 函数,计算结果赋值给 BB *)

(2) 功能块调用和控制语句

分为功能块调用和功能块返回控制两个语句。

1) 功能块调用语句。功能块调用语句采用功能块实例名的调用实现。例如,TMR1 是 TON 功能块实例名,调用 TMR1 用 TMR1(IN := %IX2.1, PT := T#500ms);语句实现。功能

块调用语句格式如下:

功能块实例名 (参数表);

参数表可以是形参表或实参表,必要时用赋值语句将实参赋值到形参。

【例 3-39】 功能块调用语句示例。

```
RS_1(S:= %IX0.0,R1:= %IX0.1); (* 调用 RS 功能块实例 RS_1 *)
RTRIG_1(CLK:= START);          (* 调用 R_TRIG 功能块实例 RTRIG_1 *)
```

2) 功能块返回控制语句。功能块返回控制语句是功能块的输出语句,例如, A:= TMR1.Q;。这里,TMR1 是功能块 TON 的实例名,而 Q 是功能块 TON 的输出变量。

返回控制语句格式如下:

变量:= 功能块实例名.参数名;

【例 3-40】 功能块返回控制语句示例。

```
A:= TIM_1.Q; B:= RS_1.Q1; C:= RTRIG_1.Q; (* 功能块 TON 实例 TIM_1 输出赋值给
                                           A 等 *)
```

标准的第 2 版允许直接用 => 操作符将输出直接赋值给某一变量。它使功能块的调用和返回控制语句合并。例如,可直接编写为

```
TMR1(IN:= %IX2.1,PT:= T#500ms,Q=>A);
RS_1(S:= %IX0.0,R1:= %IX0.1,Q1=>B);
RTRIG_1(CLK:= START,Q=>C);
```

3) 功能块调用规则。可根据上述函数调用规则类似执行。例如,用形参表或实参表;用形参表时输入变量的次序可任意。例如,语句 RS_1(R1:= %IX0.0,S:= %IX0.1);中,变量的次序是任意的。

【例 3-41】 功能块返回控制语句示例。

```
FUNCTION_BLOCK FCNBLK
VAR_INPUT
    VIN:INT:=4;
END_VAR
VAR_OUTPUT
    VOUT:INT;
END_VAR
VOUT:= VIN*2;
END_FUNCTION_BLOCK
...
VAR
    FBK:FCNBLK;
END_VAR
FBK(); (* 调用 FBK,变量值用初始值,因此,结果为 FBK.VOUT=8 *)
FBK(VIN:=5) (* 调用 FBK,VIN 变量值用 5,结果为 FBK.VOUT=10 *)
```

示例建立用户功能块 FCNBLK,其输入变量 VIN,输出变量 OUT,功能块实例名为 FBK。调用 FBK 时,可使用功能块的初始约定值,如 FBK();语句所示,也可使用形参表进行赋值。如 FBK(VIN:=5);语句所示。

4) RETURN 语句。RETURN 语句用于提供从函数、功能块或程序(例如,作为一个 IF 语句的求值结果)的出口。因此,在执行该语句前,需根据函数名对变量赋值。如果在功能块内没有对功能块的输出变量赋值,则输出变量采用输出变量数据类型的初始值或最新的存储数值进行赋值。RETURN 语句没有操作数。

【例 3-42】 RETURN 语句示例。

加计数器 CTU 功能块也可用 RETURN 语句编写如下,其变量声明如下:

```
VAR_INPUT
    CU:BOOL  R_TRIG;
    R:BOOL;
    PV:INT;
END_VAR
VAR_OUTPUT
    Q:BOOL;
    CV:INT;
END_VAR
IF NOT (CU) THEN
    Q:=FALSE;
    CV:=0;
    RETURN;
END_IF;
IF R THEN
    CV:=0;
ELSIF (CV < PV) THEN
    CV:=CV+1;
END_IF;
Q:=(CV >= PV);
```

示例的程序与表 2-44 的程序比较,可以发现,由于采用 RETURN 语句,当没有脉冲信号输入时,将输出 Q 复位,CV 设置回复到零,并终止程序,返回到程序起点,从而缩短程序执行时间。只有当有输入脉冲 CU 信号后,程序才执行后续语句,完成 CV 加 1 操作或复位操作。因此,后续程序中不需要(CV < PV)和 CU 的与逻辑运算。

5) 参数表的格式。分形参表和实参表。

- 形参表格式。形参表格式如下:

变量名 := 表达式 或 变量名 => 变量

例如,MX := 10 或 ENO => TEMPL。

- 实参表格式。实参表格式如下:

表达式

例如,IN:= %IX2.1,PT:= T#500ms。

6) 结构化文本编程语言中 EN 和 ENO 的使用。结构化文本编程语言可调用函数和功能块,它对函数和功能块中 EN 和 ENO 变量的使用是可选项。不选用 EN 和 ENO 变量时,函数或功能块的调用将被执行,而且能够将状态传递到下一个函数或功能块。选用 EN 和 ENO 变量时,需要根据变量的值确定函数或功能块是否被执行,函数或功能块是否有出错等状态需要信息的传递。

EN 是输入的布尔变量。当函数或功能块的 EN 输入为真时,函数或功能块才被执行,否则,被调用的函数或功能块不被执行。同时,被调用的函数将没有返回值,被调用的功能块输出将不改变。在函数或功能块中,不能改变 EN 的值。

ENO 是函数和功能块的输出布尔变量。当函数或功能块被执行时,ENO 被赋值为真。因此,当函数或功能块被调用时,如 ENO 的值为真表示该函数或功能块的操作正确,反之,则表示函数或功能块有错误发生。

3. 选择语句

选择语句(Selection Statement)根据规定的条件选择表达式来确定执行它所组成的语句之一。有 IF 和 CASE 两类选择语句。

(1) IF 选择语句

IF 选择语句中,只有条件表达式的值为布尔 1(TRUE)的一组语句被执行。如果条件为 0(FALSE),则没有语句被执行或者在 ELSE 关键字(或 ELSIF 关键字)所示条件中的一组语句被执行。

IF 选择语句的格式如下:

```
IF 表达式1 THEN 语句组1;  
ELSIF 表达式2 THEN 语句组2;  
ELSE 语句组;  
END_IF;
```

图 3-7 显示了 IF 选择语句的语法结构。

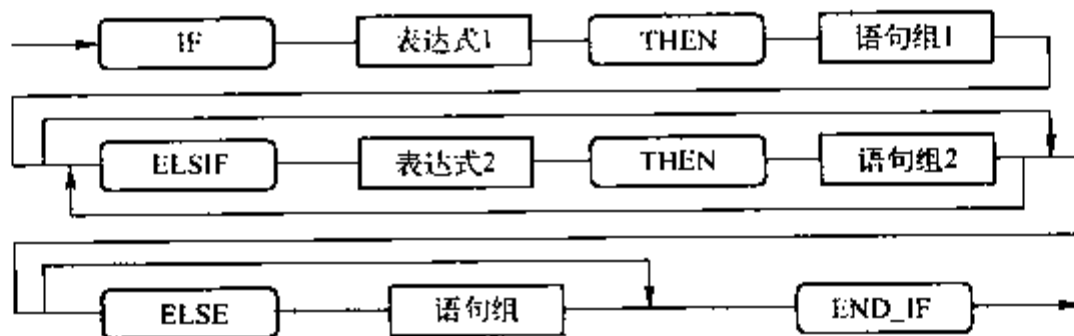


图 3-7 IF 选择语句的语法结构

IF 选择语句执行过程:当表达式 1 为 1(TRUE)时,执行语句组 1;如果为 0(FALSE),则当表达式 2 为 1 时,执行语句组 2,否则执行语句组。可用图 3-8 表示 IF 选择语句的执行过程。

使用 IF 选择语句时注意下列事项。

1) “ELSIF 表达式 2 THEN 语句组 2”的语句可重复多次。应注意关键字是 ELSIF,不是 ELSEIF。

2) 一些 IF 选择语句中,可以没有 ELSIF 和 ELSE 的语句段,见例 3-43。

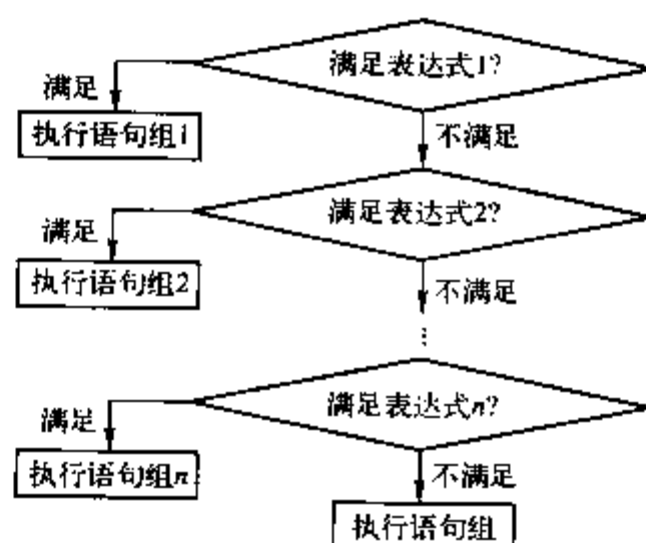


图 3-8 IF 选择语句的执行过程

【例 3-43】 IF 选择语句的示例。

```

IF  MAN AND NOT ALM THEN  (* 如果在手动(MAN)和未报警(NOT ALM) *)
    LI02 := MAN_LEVEL;      (* 则液位 LI02 是手动测量值 *)
    BX111 := BI12 OR BI23 ; (* BX111 是 BI12 或 BI23 *)
ELSIF OVER_MODE THEN      (* 如果在结束模式 *)
    LI02 := MAN_LEVEL;      (* 则液位 LI02 是手动测量值 *)
ELSE  LI02 := (LV2 * 100) / SCALE; (* 否则液位 LI02 是自动测量值 LV2 * 100,除以量程 SCALE *)
END_IF;
IF  OVERFLOW THEN          (* 如果过流 *)
    ALM_LEVEL := TRUE;      (* 则设置 ALM_LEVEL 为真 *)
END_IF;
  
```

3) 表达式用后续的关键字 THEN 判别其是否结束。

4) 语句组是一个或多个用分号分隔的语句。语句组结束用关键字 END_IF 判别。

【例 3-44】 计算一元二次方程根的程序。

```

VAR  A,B,C,D,X1,X2:REAL; NROOT:INT;  END_VAR
D := B * B - 4.0 * A * C;              (* 计算判别式 *)
IF D < 0.0 THEN NROOT := 0;             (* 判别式小于0,方程无根 *)
ELSIF D = 0.0 THEN NROOT := 1; X1 := -B / (2.0 * A); (* 判别式等于0,两个重根 *)
ELSE NROOT := 2;                       (* 否则,有两个根 *)
X1 := (-B + SQRT(D)) / (2.0 * A);      (* 计算一个根 *)
X2 := (-B - SQRT(D)) / (2.0 * A);      (* 计算另一个根 *)
END_IF;
  
```

程序中,计算判别式 D 的值,如 $D < 0.0$ 的条件为真,则表示没有实数根, NROOT 被赋值 0。反之,如 $D = 0.0$ 的条件为真,则表示有一个重根 X1,其值为 $-B / (2.0 * A)$ 。如果 $D > 0.0$ 的条件为真,则有两个不等根,分别为 X1 和 X2,其值如程序所示。

5) 可通过整数表达式、数值表和相应的语句组实现多重选择,见例 3-45。

【例 3-45】 多重选择的示例。

```
VAR A:INT; END_VAR
...
IF A = 1 THEN DISP('ONE'); END_IF; (* 如果 A 为 1,显示“ONE” *)
IF A = 2 THEN DISP('TWO'); END_IF; (* 如果 A 为 2,显示“TWO” *)
IF A = 3 THEN DISP('THREE'); END_IF; (* 如果 A 为 3,显示“THREE” *)
IF A = 4..8 THEN DISP('MORE'); (* 如果 A 为 4 到 8 的整数,显示“MORE” *)
ELSE DISP('ERROR'); (* 如果 A 不是上述的数值,显示“ERROR” *)
END_IF;
...
```

示例根据 A 的值执行有关的语句。例如, A = 1, 执行显示 ONE 的操作等。可看到, 数值格式可以是一个或多个用逗号分隔的整数常数, 也可以用“..”分隔的整数常数范围, 例如, 4..8 表示整数 4, 5, 6, 7, 8。程序调用 DISP 功能块用于显示输入的字符串。

6) IF 选择语句至少由一个 IF、一个 THEN 和一个 END_IF 组成。

【例 3-46】 3-8 译码。

```
VAR S0,S1,S2,SUM:INT; KEY1,KEY2,KEY3:BOOL; END_VAR
...
IF KEY1 THEN S0:=1; END_IF; (* 按键 1,则对应变量的值为 1,否则为 0 *)
IF KEY2 THEN S1:=2; END_IF; (* 按键 2,则对应变量的值为 2,否则为 0 *)
IF KEY3 THEN S2:=4; END_IF; (* 按键 3,则对应变量的值为 4,否则为 0 *)
SUM:=S0+S1+S2;
```

示例中,用最简单的 IF 语句结构判别按键是否按下,如果按下,则对应的变量被赋值,如果没有按下,则对应的变量被设置为整数的约定初始值 0。3 个选择语句执行后,将有关的变量值相加,获得 0~7 的整数值。

(2) CASE 选择语句

CASE 选择语句由数据类型为 ANY_INT 或枚举数据类型变量求值的表达式和一组语句列表组成。

CASE 选择语句的语法结构见图 3-9。

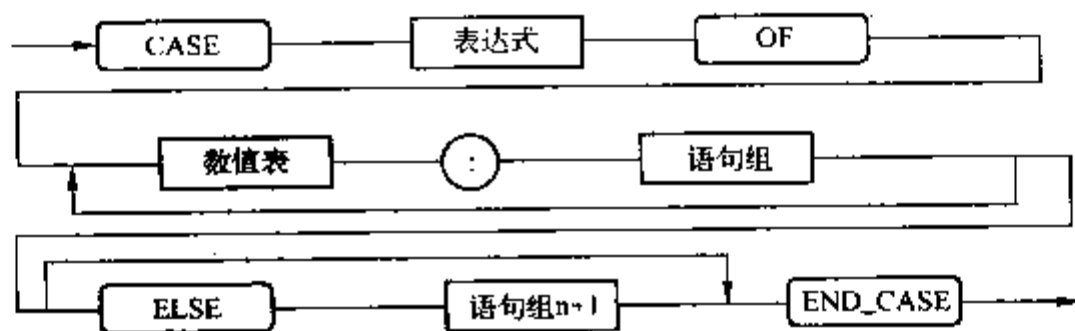


图 3-9 CASE 选择语句的语法结构

CASE 选择语句格式如下：

```
CASE 表达式 OF
  数值表元素 1:语句组 1;
  ...
```



```

数值表元素  $n$ : 语句组  $n$ ;
ELSE      语句组  $n+1$ ;
END_CASE;

```

CASE 选择语句格式中,表达式的值与数值表元素比较,当它与数值表元素 1 相同时,执行语句组 1;当它与数值表元素 m ($1 < m \leq n$) 相同时,执行语句组 m ;如果都不相同,则执行语句组 $n+1$ 。如果语句段没有 ELSE 的语句,则都不相同时,不执行任何语句。

数值表元素可以是一个或几个整数或整数值,也可以是枚举值或整数值的范围。图 3-10 显示了 CASE 语句的执行过程。

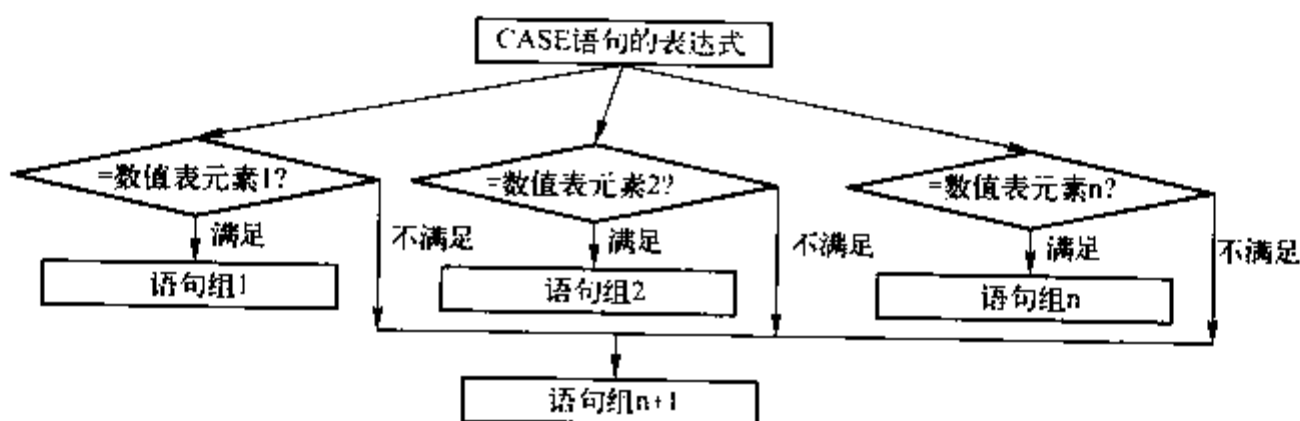


图 3-10 CASE 语句的执行过程

【例 3-47】 多路开关选择的 CASE 语句示例。

```

TW := BCD_TO_INT(THUMBWHEEL);
TW_ERROR := 0;
CASE TW OF
    1,5 : DISPLAY := OVEN_TEMP;           (* 开关在 1 或 5 位置,显示烤箱温度 *)
    2 : DISPLAY := MOTOR_SPEED;           (* 开关在 2 位置,显示电机转速 *)
    3 : DISPLAY := GROSS_TARE;             (* 开关在 2 位置,显示净重 *)
    4,6..10 : DISPLAY := STATUS(TW_4);    (* 开关在 4,6,7,,8,9,10 位置 *)
                                           (* 显示 TW_4 的状态 *)
    ELSE DISPLAY := 0; TW_ERROR := 1;     (* 开关在其他位置,表示出错位置 *)
END_CASE;
QW100 := INT_TO_BCD(DISPLAY);            (* 转换为 BCD 代码显示 *)

```

示例根据由拨盘输入并转换为整数的 TW 值来确定 DISPLAY 值。当 TW 是 1 或 5,即其位置在 1 或 5 时,DISPLAY 值是 OVEN_TEMP,即显示烤箱的温度值;当 TW 是 2 时,DISPLAY 值是 MOTOR_SPEED,即显示电动机的转速;当 TW 是 3 时,显示毛重减皮重,即净重;当 TW 是 4,6,7,8,9 或 10 时,都显示 TW_4 状态;否则,如果 TW 不在上述位置,则 DISPLAY 值为 0。同时,将 TW_ERROR 设置为 1,表示 TW 位置设置不正确。最后一个语句是将 DISPLAY 值转换为 BCD 数据,用于数码管显示。

使用 CASE 选择语句时应注意下列事项:

1) CASE 选择语句中,表达式的值必须是整数。为此,需将非整数数据类型的数据转换为整数数据类型。例如,上例中的 `TW := BCD_TO_INT(THUMBWHEEL);` 语句将 BCD 数据类型转换为整数数据类型,以便 CASE 选择语句使用。

2) 可采用连续符号“..”表示连续多个整数。例如,例 3-47 中,6..10 表示整数 6 或 7,8,9,10。图 3-11 给出了数值表的结构。

3) 不同整数数值用逗号“,”分隔表示,用于选择执行相同的语句组。例如,上例中,1,5 表示 TW 是 1 或 5 时都将 DISPLAY 赋值为 OVEN_TEMP。

4) ELSE 选项是可选的。一些程序可只有 CASE...OF...END_CASE 结构。

5) CASE 选择语句常用于程序出错代码的显示。

【例 3-48】 出错代码的 CASE 语句示例。

```

CASE ERROR_CODE OF
    1: ERR_MSG := '无法处理变量';           (* 代码是 1 表示无法处理变量 *)
    2: ERR_MSG := '函数返回值数据类型失配'; (* 代码是 2 表示函数返回值数据类型失配 *)
    3: ERR_MSG := '将 EN 和 ENO 作为变量';   (* 代码是 3 表示将 EN 和 ENO 作为变量 *)
    4: ERR_MSG := '非法的直接表示变量';     (* 代码是 4 表示非法直接表示变量 *)
    ...
    255: ERR_MSG := '变量的初始值无效';      (* 代码是 255 表示变量的初始值无效 *)
    ELSE ERR_MSG := '未知错误';              (* 代码是其他数值,表示未知的出错 *)
END_CASE;

```

【例 3-49】 七段码显示。

七段显示码用图 3-12 所示码管。为显示 0~9,对应的码管被点亮,见表 3-14。

表 3-14 数字 0-9 与七段显示码的关系

数字	0	1	2	3	4	5	6	7	8	9
A	1	0	1	1	0	1	1	1	1	1
B	1	1	1	1	1	0	0	1	1	1
C	1	1	0	1	1	1	1	1	1	1
D	1	0	1	1	0	1	1	0	1	1
E	1	0	1	0	0	9	1	9	1	0
F	1	0	0	0	1	1	1	1	1	1
G	0	0	1	1	1	1	1	0	1	1

```

CASE NUMB OF
    1: A := 1; B := 1; C := 1; D := 1; E := 1; F := 1; G := 0;
    2: A := 0; B := 1; C := 1; D := 0; E := 0; F := 0; G := 0;
    3: A := 1; B := 1; C := 1; D := 1; E := 0; F := 0; G := 1;
    4: A := 0; B := 1; C := 1; D := 0; E := 0; F := 1; G := 1;
    5: A := 1; B := 0; C := 1; D := 1; E := 0; F := 1; G := 1;
    6: A := 1; B := 0; C := 1; D := 1; E := 1; F := 1; G := 1;
    7: A := 1; B := 1; C := 1; D := 0; E := 0; F := 1; G := 0;
    8: A := 1; B := 1; C := 1; D := 1; E := 1; F := 1; G := 1;
    9: A := 1; B := 1; C := 1; D := 1; E := 0; F := 1; G := 1;
END_CASE;

```

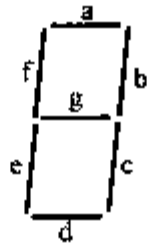


图 3-12 七段显示码

示例中,用 NUMB 的值与数值表的值比较,并将有关段的显示码管导通,从而实现数字显示。实际应用时,也可用于显示十六进制的数字 A ~ F。

4. 循环语句

循环语句 (Iteration Statement) 用于对部分语句组进行重复执行。分为 FOR、WHILE 和 REPEAT 3 种语句结构。为了在循环执行过程中终止循环,可采用 EXIT 语句,它用于在终止条件满足前终止循环。

(1) FOR 循环语句

当循环次数可以事先确定时,采用 FOR 循环语句实现语句组的循环执行。FOR 循环语句的格式如下:

```
FOR 控制变量: = 初值表达式 TO 终值表达式 BY 增量表达式 DO
语句组;
END_FOR;
```

FOR 循环语句格式中,控制变量是在循环执行过程中不断变化的变量,在每次循环执行后,该变量的值增加增量表达式所计算的值,即控制变量 := 控制变量 + 增量表达式。因此,如果增量表达式的值是负数,表示每次循环执行后,控制变量的值减小。

初值表达式是控制变量起始值,终值表达式是控制变量终止值,增量表达式是控制变量的每次增量。每次循环执行后控制变量的值变化,即当前控制变量值加增量,如果其值没有超过终止值,则继续执行循环,反之,如果超过终止值,或执行过程中执行到 EXIT 语句,则循环执行过程终止。语句组是需要循环执行的语句集合。

图 3-13 显示 FOR 循环语句的结构。

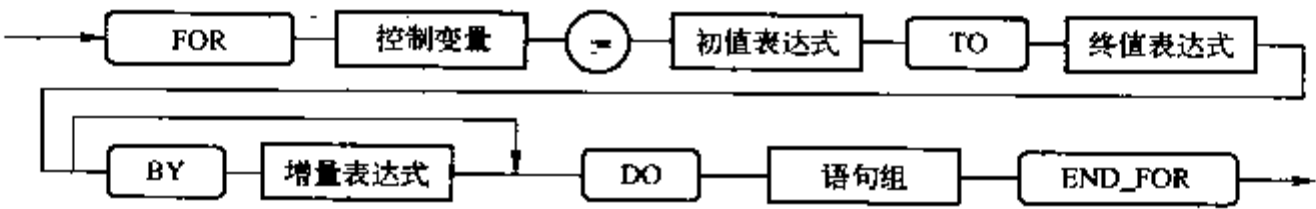


图 3-13 FOR 循环语句的语法结构

【例 3-50】 FOR 循环语句计算累加和阶乘。

```
SUM: =0; FACTORIAL: =1;
FOR I: =1 TO 100 BY 1 DO
    SUM: =SUM + I;
    FACTORIAL : = FACTORIAL * I;
END_FOR;
```

示例中,计算 1 ~ 100 的累加和结果 SUM 和阶乘结果 FACTORIAL。

【例 3-51】 FOR 循环语句用于赋值。

```
FOR I: =1 TO 3 DO
    FOR J: = 1 TO 10 DO
        X[1,J] : = I + J*2;
    END_FOR;
```

END_FOR;

示例用于给数组元素赋值。

使用 FOR 循环语句时应注意下列事项:

1) 控制变量初值、终止值和增量的数据类型应一致,必须是相同的整数类型(例如,SINT, INT 或 DINT)。

2) 控制变量起始值和增量的约定值都为 1,因此,如果程序没有设置相应表达式,系统自动设置它们的值为 1。BY 关键字是可选的,当增量表达式的值为 1 时,可省略 BY 和增量表达式。例如,例 3.50 中,可以列写为

```
FOR 1 TO 100 DO
```

3) 终止条件的测试在每次循环开始前进行,如果控制变量当前值超过终止值时,循环语句中的语句组不被执行。每次循环执行后进行赋值运算,即控制变量 := 控制变量 + 增量表达式,并更新控制变量值。控制变量当前值超过终止值表示控制变量已经不在由起始值和终止值组成的循环范围内。当控制变量当前值等于终止值时,执行最后一次循环操作。

4) 控制变量起始值、终止值和增量的表达式在执行过程中只计算一次。与其他循环语句不同,它们不允许被任何循环语句内的语句所改变。

5) FOR 循环语句执行后,控制变量的值与具体的实现有关,应根据不同制造商的产品说明书确定。

(2) WHILE 循环语句

WHILE 循环语句根据表达式条件是否为真(满足)确定是否执行有关循环语句。因此,循环次数在循环语句执行前是不确定的。WHILE 循环语句的格式如下:

```
WHILE 条件表达式 DO
语句组;
END_WHILE;
```

WHILE 循环语句格式中的表达式是一个布尔表达式,其值为真时,循环语句组被执行,反之,则不被执行。图 3-14 显示了 WHILE 循环语句的语法结构。



图 3-14 WHILE 循环语句的语法结构

【例 3-52】 WHILE 循环语句计算累加和阶乘。

```
J:=0; SUM:=0; FACTORIAL:=1;
WHILE J < 100 DO
J:=J+1; SUM:=SUM+J; FACTORIAL:=FACTORIAL*J;
END_WHILE;
```

与例 3-50 比较,本示例同样用于计算 1~100 的累加和 SUM 和阶乘 FACTORIAL。但执行 WHILE 循环语句中的语句组时,用于判别循环语句执行与否的表达式值不断变化。因此,每次循环过程开始,该循环语句先判别表达式是否为真,一旦条件满足(示例中,表达式中变量 J 的值大于等于 100)则结束循环语句的执行。

使用 WHILE 循环语句时注意下列事项：

- 1) WHILE 循环语句的循环次数事先不确定。循环语句执行过程中,表达式值变化。
- 2) WHILE 循环语句中的表达式是布尔表达式。
- 3) WHILE 循环语句中,对表达式的判别运算在循环执行前进行。
- 4) 实际应用时,须注意循环语句中语句先后次序对运算结果的影响。例如,例 3-52 中,如果, $J := J + 1$; 语句放在语句组的最后,则循环判别表达式应改为: $J \leq 100$ 。同时, $J := 0$ 的初始赋值语句改为 $J := 1$ 。
- 5) WHILE 循环语句不能用于获得内部过程间的同步,例如,用外部确定的终止条件来作为一个“等待循环”。但可用 SFC 元素实现过程间的同步。

(3) REPEAT 循环语句

REPEAT 循环语句根据表达式的条件是否为真(满足)确定是否执行有关的循环语句。因此,循环次数在 REPEAT 循环语句执行前是不确定的。REPEAT 循环语句的格式如下:

```
REPEAT
语句组 ;
UNTIL 条件表达式
END_REPEAT ;
```

REPEAT 循环语句执行语句组,然后对表达式进行判别,当表达式值为假(FALSE)时,循环不被执行。图 3-15 显示了 REPEAT 循环语句的语法结构。

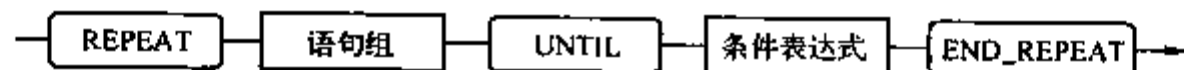


图 3-15 REPEAT 循环语句的语法结构

【例 3-53】 REPEAT 循环语句示例。

```
J := 0; SUM := 0; FACTORIAL := 1;
REPEAT
    J := J + 1; SUM := SUM + J;
    FACTORIAL := FACTORIAL * J;
UNTIL J = 100
END_REPEAT;
```

示例同样用于计算 1 ~ 100 的累加和 SUM 和阶乘 FACTORIAL。

REPEAT 循环语句使用注意事项与 WHILE 循环语句类似,两种循环语句的主要区别是对表达式判别的先后。WHILE 循环语句先判别表达式的值,再执行循环语句组;而 REPEAT 循环语句先执行循环语句组,然后判别表达式的值。因此,当初始表达式为假时,REPEAT 循环语句执行一次语句组,而 WHILE 循环语句则不执行该语句组。图 3-16 显示了两循环语句的区别。

(4) EXIT 语句

EXIT 语句用于循环语句执行过程中中断循环过程的执行,而不管循环的终止条件是否满足。

当 EXIT 语句位于嵌套循环语句内时,EXIT 语句退出循环发生在 EXIT 语句所在循环的

最内环。即在 EXIT 语句所在循环回路终止关键字 (END_FOR, END_WHILE, END_REPEAT) 后面的语句。

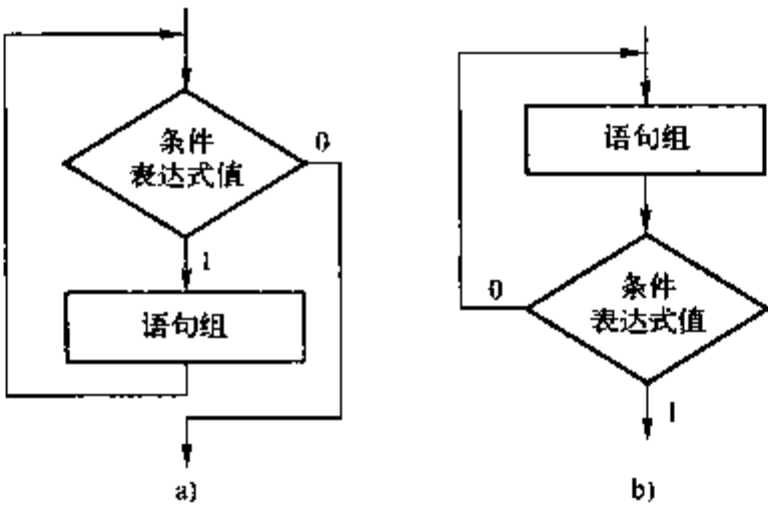


图 3-16 两种循环语句的区别
a) WHILE 语句 b) REPEAT 语句

【例 3-54】 EXIT 语句示例。

```
SUM := 0;
FOR I := 1 TO 3 DO
  FOR J := 1 TO 2 DO
    IF FLAG THEN EXIT;          (* 标志 FLAG 为 1, 则结束循环 *)
    END_IF;
    SUM := SUM + J;              (* 表 3.15 中用 SUM1 表示 *)
  END_FOR;
  SUM := SUM + I;               (* 表 3.15 中用 SUM2 表示 *)
END_FOR;
```

示例中, 当标志 FLAG 的值为假(0)时, 选择语句不被执行, 因此, SUM 的结果为 15。如果 FLAG 的值为真(1), 则执行 IF 选择语句时执行 EXIT 语句, 循环将退出到控制变量 I 层, 使 SUM 的结果为 6。表 3-15 显示了各循环执行过程中 SUM 值的变化。

表 3-15 各循环执行过程中 SUM 值的变化

FLAG	0						1					
I	1		2		3		1		2		3	
J	1	2	1	2	1	2	1	2	1	2	1	2
SUM1	1	3	5	7	10	12	0	0	1	1	3	3
SUM2	5		7		SUM = 15		1		3		SUM = 6	

WHILE 和 REPEAT 语句不能用于获得内部过程间的同步, 例如, 用外部确定的终止条件来作为“等待循环”。为此, 可采用顺序功能表图编程语言实现。

3.3.3 示例

1. 函数 WEIGH 的 ST 语言编程

以第 3.2.3 节的 WEIGH 函数为例, 说明用结构化文本编程语言的编程方法。该函数的

函数声明如第 3.2.3 节所述。用 ST 语言编写的函数体如下：

```
IF START1 THEN
    WEIGH := INT_TO_BCD(BCD_TO_INT(GROSS_WEIGHT) - TARE_WEIGHT);
END_IF;
```

示例显示用 ST 编程语言编写的函数体十分简单,它由选择语句和赋值语句组成。其中,调用标准类型转换函数和进行减法运算。

2. 功能块 HYSTERESIS 的 ST 语言编程

功能块 HYSTERESIS 用于滞环过程。图 3-17 所示滞环过程中,输出 Q 为 1 时,只有当输入信号 IN1 小于 IN2 - EPS 时,输出才切换到 0。输出 Q 为 0 时,只有当输入信号 IN1 大于 IN2 + EPS 时,输出才切换到 1。

(1) 功能块 HYSTERESIS 的变量声明

功能块 HYSTERESIS 有 3 个输入信号,输入信号送 IN1,比较信号送 IN2,滞后值送 EPS。有一个输出变量 Q,用于表示输出信号是否切换。功能块 HYSTERESIS 的变量声明如下：

```
VAR_INPUT
    IN1 :REAL;    (* 输入信号,实数数据类型 *)
    IN2 :REAL;    (* 比较信号,实数数据类型 *)
    EPS :REAL;    (* 滞后值,实数数据类型 *)
END_VAR
VAR_OUTPUT
    Q : BOOL := 0;(* 功能块输出,初始值为 0 *)
END_VAR
```

图 3-18 是功能块 HYSTERESIS 变量的图形形式。

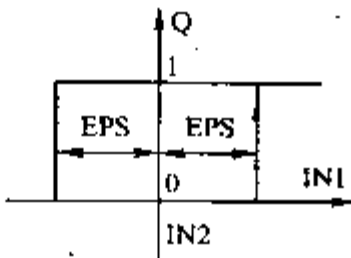


图 3-17 滞环过程

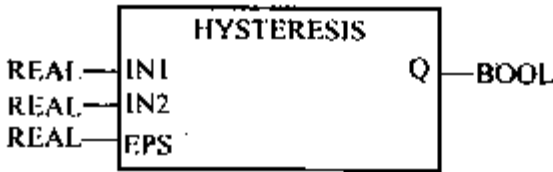


图 3-18 HYSTERESIS 功能块图形形式

(2) 功能块 HYSTERESIS 的功能块本体

功能块本体用于判别输入信号与 HL 的差。

```
IF Q THEN
    IF IN1 < (IN2 - EPS) THEN
        Q := 0; (* IN1 减小 *)
    END_IF;
ELSIF IN1 > (IN2 + EPS) THEN
    Q := 1; (* IN1 增加 *)
END_IF;
```

(3) 功能块 HYSTERESIS 的应用

HYSTERESIS 功能块可用于过程的位式控制。其中,IN1 连接过程被控变量 PV,IN2 连接过程设定变量 SP,EPS 连接所需的控制偏差 EPS。程序的变量声明如下:

```
VAR
    PV      :REAL;          (* 过程测量,实数数据类型 *)
    SP      :REAL;          (* 过程设定,实数数据类型 *)
    EPS     :REAL;          (* 偏差,实数数据类型 *)
    QQ      :BOOL;          (* 位式控制器输出 *)
    HYS_1   :HYSTERESIS     (* HYS_1 是 HYSTERESIS 功能块的实例名 *)
END_VAR
```

程序本体如下。

```
HYS_1(IN1 := PV,IN2 := SP,EPS := EPS);  (* 调用 HYS_1 实例 *)
QQ := HYS_1.Q;                          (* 实例 HYS_1 输出送 Q *)
```

示例程序也可直接用:

```
HYS_1(IN1 := PV,IN2 := SP,EPS := EPS,Q => QQ);
```

3. 功能块 DELAY 的 ST 语言编程

功能块 DELAY 是时滞功能块,它与 HYSTERESIS 滞后功能块不同。输出信号在时间上滞后输入信号的时间称为时滞。生产过程的被控对象常用一阶滤波环节加时滞描述。这里介绍时滞功能块。一阶滤波环节功能块的介绍见第 4 章。

时滞环节的传递函数是:

$$Y(s) = e^{-\tau s} X(s) \quad (3-1)$$

假设采样周期为 T_s ,则离散化后,得

$$Y(k) = X(k - N) \quad (3-2)$$

式中, X 是时滞环节的输入信号; Y 是时滞环节的输出信号。设离散化所采用的采样周期是 T_s ,则时滞 τ 与采样周期 T_s 之比称为滞后拍数 N 。

(1) 功能块 DELAY 的变量声明

采用存储器存储输入信号,存储器各单元的内容存储不同时刻的采样数据,即第 1 单元存储时刻 $1 \times T_s$ 的采样值,第 i 单元存储时刻 $i \times T_s$ 的采样值。时滞时间 τ 与采样周期 T_s 之比的整数值是 N (N 的小数部分去除后,用 N 表示)。因此,如果某时刻,输入信号存储在第 N 单元,则经时滞的输出信号应从第 1 存储单元输出。

```
VAR_INPUT
    XIN :REAL;              (* XIN 是输入信号 *)
    AUTO :BOOL;             (* 自动手动标志 *)
    TS :TIME;               (* 采样周期 *)
    DT1 :TIME;              (* 时滞时间 *)
END_VAR
VAR_OUTPUT
    XOUT :REAL;             (* 经滞后环节处理后的输出 *)
END_VAR
```



```

VAR
  N :XIN INT;                                ( * 滞后拍数 * )
  X1 :ARRAY [0.. 2047] OF REAL :=0;         ( * 先进先出的堆栈 * )
  IXIN :INT;                                  ( * X 数组的下标,用于输入 * )
  IXOUT :INT;                                 ( * X 数组的下标,用于输出 * )
  R_TRIG_1 :R_TRIG;                          ( * 将自动信号转换为脉冲 * )
  TON_1:TON;
END_VAR

```

(2) 功能块 DELAY 的本体

用结构化文本编程语言编写的功能块 DELAY 的本体程序如下。

```

N := TIME_TO_INT(DTI)/ TIME_TO_INT(TS); ( * 计算滞后拍数 N * )
R_TRIG_1(CLK:= AUTO);                   ( * 调用上升沿检测功能块,将自动开关
                                         信号转换为脉冲信号 * )

IF R_TRIG_1.Q THEN
  IXIN := N; IXOUT := 0;                 ( * 自动时设置输入和输出的初始值 * )
END_IF;

TON_1(IN := NOT TON_1.Q, PT := TS);     ( * 调用采样时刻 * )
IF TON_1.Q AND AUTO THEN                ( * 在自动和采样时刻 * )
  IXIN := (IXIN + 1) MOD 2000;           ( * 输入信号的下标计算 * )
  X1[IXIN] := XIN;                      ( * 输入信号采样 * )
  IXOUT := (IXOUT + 1) MOD 2000;         ( * 输出信号的下标计算 * )
  XOUT := X1[IXOUT];                   ( * 输出信号输出 * )
END_IF;

```

功能块本体采用两个下标窗口来管理输入和输出信号的存取和输出。输入信号数据存放在数组 X 的 IXIN 下标地址,初始值等于滞后拍数。输出信号在数组 X 的 IXOUT 下标地址,输出初始值等于 0。采用模除的方法确定每次的存放和输出的地址,并在每次执行操作后,将原地址加 1。保证下次执行操作时,存放该次的输入数据和将前 N 次输入信号作为该次输出。

存储器单元的数量与时滞大小和采样周期有关。当时滞越大,采样周期越小时,所需的存储器单元越多。一般可根据应用的大小使滞后拍数 N 大于存储器单元总数即可。

示例中,要求滞后拍数 N 小于 2000(存储单元有 2048 个)。此外,数组的存储器单元从 0 地址开始,实际应用从地址 0 开始。图 3-19 显示了输入窗口和输出窗口的关系。

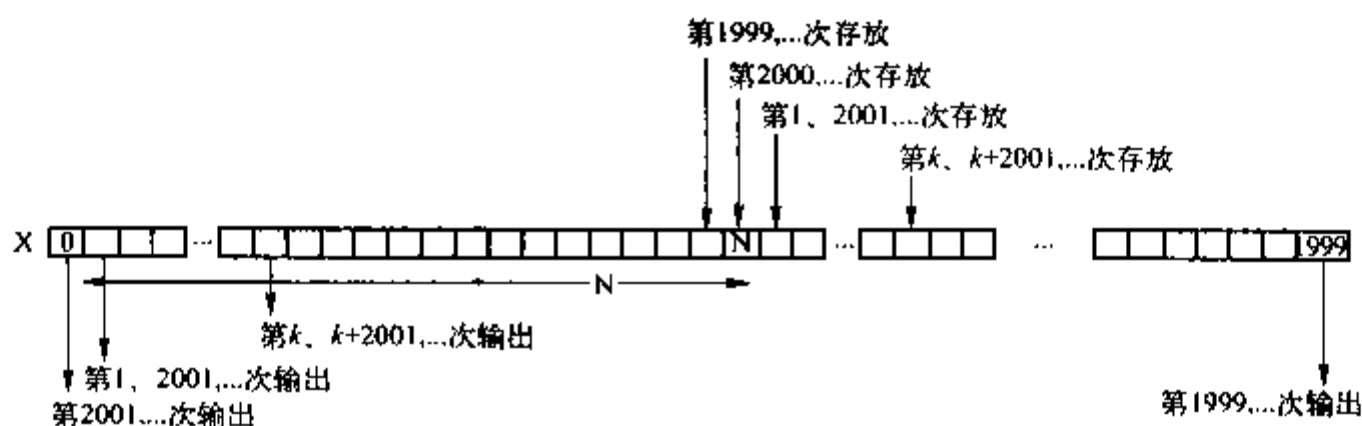


图 3-19 输入窗口和输出窗口的关系

(3) 使用功能块 DELAY 注意事项

1) 该功能块图程序使用数组数据类型,应在具有数组数据类型的软件系统中实现。

2) 滞后拍数 N 与时滞、采样周期有关,程序中用运行状态切换到自动状态的信号作为初始值设定的脉冲信号。

由于采用高级编程语句,例如,选择和循环语句等,通常结构化文本编程语言编写的程序较难转换为低级的编程语言,例如梯形图和指令表编程语言。

3) 该功能块可与一阶滤波(惯性)环节组合,用于模拟实际生产过程,进行控制系统仿真研究。此外,应设置周期性的任务实现。

4. 功能块 DIVREMBK 的 ST 语言编程

第2章介绍的功能块 DIVREMBK 是用户编写的功能块,它用于计算两个整数相除时的商和余数。当除数为零时,功能块输出 DIVERR 为真。它采用指令标编程语言编写。这里,用结构化文本编程语言编写。

(1) 功能块的变量声明

变量声明与第2章所编写的变量声明相同。

(2) 功能块本体

功能块本体判别 DIVISOR 变量是否为零,如果为零,则将商和余数赋值为零,并将 DIVERR 赋值为 1。反之,用除法操作计算商,用模除操作计算余数,并将 DIVERR 赋值为 0。用结构化文本编程语言编写的功能块本体的程序如下:

```
IF DIVISOR <> 0 THEN
    QUOTIENT := DIVIDEND / DIVISOR;      (* 计算商 *)
    DIVREM := DIVIDEND MOD DIVISOR;      (* 计算余数 *)
    DIVERR := 0;                          (* 表示除数不为零 *)
ELSE
    QUOTIENT := 0;                        (* 除数为零时,商置 0 *)
    DIVREM := 0;                          (* 除数为零时,余数置 0 *)
    DIVERR := 1;                          (* 表示除数为零,出错标志置 1 *)
END_IF;
```

(3) 功能块的应用

程序 EXP1 用于调用功能块 DIVREMBK,程序中,被除数是 44,除数是 5。程序如下:

```
PROGRAM EXP1
VAR
    A1 :INT := 44;                        (* A1 表示被除数,其初始值为 44 *)
    B1 :INT := 5;                          (* B1 表示除数,其初始值为 5 *)
    D1,R1,E1 :INT;                        (* 分别表示商、余数和出错标志 *)
    DD :DIVREMBK;                          (* 用实例名 DD 表示功能块 DIVREMBK *)
END_VAR
DD(DIVIDEND := A1, DIVISOR := B1); (* 调用 DD *)
D1 := DD. QUOTIENT;                      (* 功能块输出变量 QUOTIENT 的值赋值给 D1 *)
R1 := DD. DIVREM;                         (* 功能块输出变量 DIVREM 的值赋值给 R1 *)
```

```

        E1 := DD.DIVERR;                ( * 功能块输出变量 DIVERR 的值赋值给 E1 * )
    END_PROGRAM

```

5. 计算最大、最小和平均

一些应用中,常常要计算若干测量值的最大、最小或平均值。下面是用结构化文本语言编写的示例:

```

TYPE T_CHANNEL;
    STRUCT
        Value : REAL; State : BOOL;
    END_STRUCT;
END_TYPE;
TYPE
    T_Inputs : ARRAY[1..32] OF T_CHANNEL;
END_TYPE;
VAR
    Max : REAL := 0.0; Min : REAL := 2000.0;
    Input : T_Inputs AT %IW130;
    Sum : LREAL := 0.0;
    Average : LREAL;
END_VAR;
Sum := 0;
FOR I := 1 TO 32 DO
    Sum := REAL_TO_LREAL( Input[I].Value ) + Sum;
    IF Input[I].Value > Max THEN Max := Input[I].Value; END_IF;
    IF Input[I].Value < Min THEN Min := Input[I].Value; END_IF;
END_FOR;
Average := Sum / 32.0;

```

示例计算某窑炉内 32 个通道温度测量值的最大 Max、最小 Min 和平均 Average。采用 FOR...DO 语句扫描所有输入通道,计算平均、最大和最小,Sum 是总和。计算数据类型采用双精度浮点数,可提高计算精确度。

6. 函数和功能块调用的嵌套

下面示例说明如何进行函数和功能块调用的嵌套。

(1) 距离函数 DISTANCE

DISTANCE 函数用于计算平面上两点间距离。在第 2 章三角形面积计算时,用指令表编程语言已经编写过。其中,(X1、Y1)与(X2、Y2)两点之间距离 D 用下列计算公式:

$$D = \sqrt{(X2 - X1)^2 + (Y2 - Y1)^2} \quad (3-3)$$

DISTANCE 函数程序如下:

```

FUNCTION DISTANCE:REAL    ( * 函数名 DISTANCE,返回值数据类型实数 * )
    VAR_INPUT
        X1,Y1:REAL;        ( * X1,Y1 表示点 1 的平面坐标 * )
        X2,Y2:REAL;        ( * X2,Y2 表示点 2 的平面坐标 * )

```

```

END_VAR
DISTANCE := SQRT((X2 - X1) ** 2.0 + (Y2 - Y1) ** 2.0);
(* 计算距离,作为函数的返回值 *)
END_FUNCTION

```

(2) 面积功能块 AREA

AREA 功能块用于根据三角形的三点坐标数值计算三角形面积。已知平面上的三点坐标 (X1, Y1)、(X2, Y2) 和 (X3, Y3), 则三角形三边的长度 S1、S2 和 S3 可用距离函数计算。三角形面积 SS 的计算公式如下:

$$SS = \sqrt{(SA - S1) * (SA - S2) * (SA - S3) * SA} \quad (3-4)$$

式中, SA 是三边和的一半。因此, 面积功能块 AREA 的程序如下:

```

FUNCTION_BLOCK AREA
VAR_INPUT
    X1, Y1: REAL;          (* X1, Y1 表示点 1 的平面坐标 *)
    X2, Y2: REAL;          (* X2, Y2 表示点 2 的平面坐标 *)
    X3, Y3: REAL;          (* X3, Y3 表示点 3 的平面坐标 *)
END_VAR
VAR_OUTPUT
    SS: REAL;               (* SS 表示由三点组成的三角形面积 *)
*)
END_VAR
VAR
    S1, S2, S3: REAL;       (* 三角形的三条边长 *)
    SA: REAL;               (* 三边长之和的一半 *)
END_VAR
S1 := DISTANCE(X1, Y1, X2, Y2); (* 调用距离函数, 计算边长 S1 *)
S2 := DISTANCE(X1, Y1, X3, Y3); (* 调用距离函数, 计算边长 S2 *)
S3 := DISTANCE(X2, Y2, X3, Y3); (* 调用距离函数, 计算边长 S3 *)
SA := ADD(S1, S2, S3) / 2.0;    (* 计算三边长之和的一半 *)
SS := SQRT(SA * (SA - S1) * (SA - S2) * (SA - S3)); (* 计算三角形面积 *)
END_FUNCTION_BLOCK

```

(3) 调用 AREA 功能块计算三角形面积

图 3-20 显示三角形面积, 其计算程序如下:

```

VAR
    MJ1: AREA;              (* MJ1 是面积功能块的实例名 *)
    AA: REAL;
END_VAR
MJ1( X1 := 2, X2 := 4, X3 := 4, Y1 := 2, Y2 := 2, Y3 := 4 );
(* 调用 AREA 功能块实例计算面积 *)
AA := MJ1.SS;              (* 计算结果送 AA *)

```

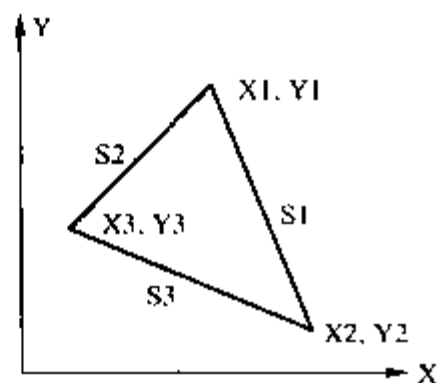


图 3-20 三角形的面积

上述程序在调用功能块 AREA 过程中,先调用函数 DISTANCE,因此,这是函数和功能块调用的嵌套。从程序可见,功能块实例名必须在内部变量声明段声明,但函数不需要再对变量声明。

7. 容器液位控制功能块

在粮食生产和药品生产过程中,采用加料和排料阀来控制容器的液位,容器的重量用称重单元监视。

功能模块 TankControl 监视容器重量,通过比较实际重量与两个满和空的输入重量确定容器是否充满或排空。图 3-21 是工艺过程简图。

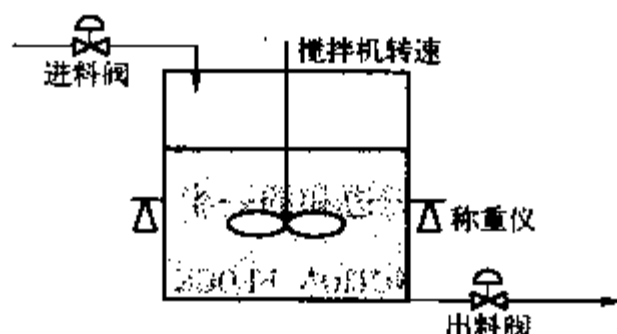


图 3-21 容器液位用称重仪进行控制

(1) 系统分析

功能模块 TankControl 提供命令输入有 4 种模式:加料,保持,搅拌,出料。如果容器在正确的状态,相应的阀门打开或关闭来控制容器的液位。

输入变量有 4 个:命令 Command、称重 Weight、满罐重量 FullWeight 和空罐重量 EmptyWeight。其中,命令指上述 4 种操作模式,因此,采用 SINT 数据类型,其他输入变量采用 REAL 数据类型。

输出变量有 3 个:进料阀 FillValve、出料阀 EmptyValve 和搅拌电动机转速 StirSpeed。设置枚举数据类型 2 个:容器状态 T_STATE 和阀状态 T_VALVE。

(2) 变量声明

```

TYPE
    T_STATE : ( FULL, NOT_FULL, EMPTIED );
END_TYPE;
    ( * 容器的状态有满、不满和空 3 种 * )
TYPE
    T_VALVE : ( OPEN, SHUT );
END_TYPE;
    ( * 阀门状态有开和关两种 * )
    
```

(3) 功能块编写

编写功能块 TankControl 如下:

```

FUNCTION_BLOCK TankControl
    VAR_INPUT
        Command : SINT;                ( * 操作模式设置 * )
        Weight : REAL;                  ( * 实际称重 * )
        FullWeight, EmptyWeight : REAL; ( * 满罐和空罐的重量 * )
    END_VAR;
    VAR_OUTPUT
        FillValve : T_VALVE := SHUT;    ( * 进料阀,初始状态为关 * )
        EmptyValve : T_VALVE := SHUT;    ( * 出料阀,初始状态为关 * )
        StirSpeed : REAL := 0.0;         ( * 搅拌机转速,初始转速为 0.0 * )
    
```

```

END_VAR;
VAR
    State : T_STATE := EMPTED;          (* 容器状态,初始状态为空 *)
END_VAR;
IF Weight >= FullWeight THEN State := FULL; (* 如果称重已达满罐重量,容器状态置满 *)
ELSIF Weight <= EmptyWeight THEN State := EMPTED;
    (* 称重小于空罐重量,则容器状态置空 *)
ELSE State := NOT_FULL;                  (* 否则,容器状态被设置为不满 *)
END_IF;
CASE Command OF
    1: EmptyValve := SHUT;                (* 容器加料操作模式,排料阀关闭 *)
        FillValve := SEL( G:= State = FULL, IN0 := OPEN, IN1 := SHUT );
        (* 如果容器未满,则开进料阀 *)
    2: EmptyValve := SHUT;                (* 容器保持模式 *)
        FillValve := SHUT;                (* 进料阀关闭 *)
    4: FillValve := SHUT;                 (* 出料模式,容器满,关进料阀 *)
        EmptyValve := OPEN;               (* 打开排料阀 *)
END_CASE;
StirSpeed := SEL( G:= ((Command = 3) AND (State = FULL)), IN0 := 0.0, IN1 := 100.0 );
(* 搅拌模式时,如果容器状态为满,则设置搅拌机转速为 100.0,否则转速置为 0.0 *)
END_FUNCTION_BLOCK

```

程序中,第一个 IF 语句,监视容器重量,并设置容器状态 State,即 FULL、NOT_FULL、EMPTY。

CASE 语句检查命令方式,并选择阀门的设置。FillValve 的设置是根据比较表达式 State = FULL 的结果,当 State 不是 FULL,则表达式的值为 0,FillValve 加料阀就选择 IN0,即 OPEN。一旦 State 为 FULL,则加料阀 FillValve 选择 IN1,即 SHUT。进一步的选择确保只有在 Command 为 3,及容器状态为 FULL 时,StirSpeed 才被设置为 100.0。

实际应用时,还需要考虑对阀状态的检查,例如,是否有堵塞等,及需要对称重信号进行滤波,例如,可采用 DELAY 功能块实现一阶滤波环节等。有时,可根据称重的变化率了解容器状态和阀状态。

8. 斜坡信号发生器

斜坡信号发生器可用于零部件的老化处理。由于需要按某一速率升温或降温,因此,需要斜坡信号发生器。

(1) 变量声明

斜坡信号发生器要求产生一个输出 XOUT,它从初始值 X0 开始,在规定的时间内,增加到 XE。

建立斜坡信号发生器功能块 RAMP。变量声明如下:

```

VAR_INPUT
    RUN : BOOL;          (* 斜坡为 1,复位为 0 *)

```

```

        X0,XE:REAL;      (* 斜坡初始值和斜坡目标值 *)
        TR:TIME;         (* 斜坡的持续时间 *)
        CYCLE:TIME;      (* 采样周期 *)
    END_VAR
    VAR_OUTPUT
        RAMP:BOOL;       (* 当斜坡输出时为 1 *)
        XOUT:REAL;        (* 斜坡输出 *)
    END_VAR
    VAR
        XI:REAL;         (* 中间的斜坡值 *)
        T:TIME:=T#0s;    (* 进入斜坡的时间 *)
    END_VAR

```

(2) 功能块本体程序

斜坡信号需要在 TR 时间内从 X0 匀速增加到 XE,系统采样周期为 CYCLE,因此,每次采样时刻,输出应增加的值为

$$\Delta XOUT = (XE - X0) * CYCLE / TR$$

因此,斜坡信号发生器功能块本体程序如下:

```

    RAMP := RUN;
    IF RUN THEN
        IF T >= TR THEN
            RAMP := 0; XOUT := XE;
        ELSE
            XOUT := XI + (XE - XI) * TIME_TO_REAL(T) / TIME_TO_REAL(TR);
            T := T + CYCLE;
        END_IF;
    ELSE
        XOUT := X0;
        XI := X0;
        T := T#0s;
    END_IF;

```

程序中,只有 RUN 为 1 时,才能够发生斜坡信号。到达 XE 终值后,输出 XOUT 保持在该值。当设置的 XE 小于 X0 时,可得到减速的斜坡信号。

第 4 章 图形类编程语言

4. 1 图形类编程语言的公用元素

4. 1. 1 线、模块和流向

IEC 61131-3 标准规定了两种图形类编程语言。梯形图 (Ladder Diagram, LD) 编程语言用一系列梯级组成梯形图, 表示工业控制逻辑系统中各变量之间的关系。功能块图 (Function Block Diagram, FBD) 编程语言用一系列功能块的连接表示程序组织单元的本体部分。




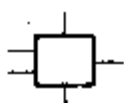
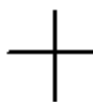
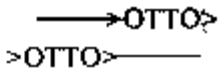
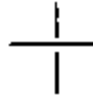
1. 线和模块的表示

图形类编程语言 (梯形图编程语言和功能块图编程语言) 中, IEC 61131-3 标准规定可使用表 4-1 所示的线和模块等图形或半图形元素来表示公用元素。

线可用表 4-1 所示的连接的图形元素进行扩展, 数据存储或与数据元素的结合与连接元素的使用无关, 因此, 连接的图形元素不应与程序中与数据元素有关的标识符相同。

本节介绍的图形类编程语言公用元素适用于图形类编程语言, 也适用于顺序功能表图编程语言。图形类编程语言中的文字元素应根据国家标准 GB 1988 字符集的“基本代码表”和 GB 2312 的“信息交换用汉字编码字符集 - 基本集”表示。

表 4-1 线和模块等图形或半图形元素

特 性	示 例	特 性	示 例
水平线 ISO/IEC 10646-1 的“负”字符 图形或半图形		连接和不连接的角 ISO/IEC 10646-1 的字符 图形或半图形	
垂直线 ISO/IEC 10646-1 的“垂直线”字符 图形或半图形		有连接线的块 ISO/IEC 10646-1 的字符 图形或半图形	
水平垂直连接 ISO/IEC 10646-1 的“加”字符 图形或半图形		用 ISO/IEC 10646-1 的字符连接 连接 连接线的继续 图形或半图形连接	
线的交叉, 不连接 ISO/IEC 10646-1 的字符 图形或半图形			

2. 流向

在图形编程语言中, 为表示概念量的流动, 采用“流向”的概念。

(1) 能流 (Power Flow)

能流类似电磁继电器系统中的电能流动。能流用于继电器逻辑控制系统的梯形图。能流

也称为功率流或电源流。能流的流向是从左到右。

(2) 信号流(Signal Flow)

信号流类似信号处理系统中元素之间的信号流动。信号流用于图形编程语言的功能块图。信号流的流向是从一个函数或功能块的输出(右侧)流向被连接的另一个函数或功能块的输入(左侧)。

(3) 活动流(Activity Flow)

活动流类似一个生产过程中的各工序之间或一个电动机顺序器中各步之间控制的流动。活动流用于顺序功能表图编程语言。活动流从步的底部,经合适的转换流到相应后级步的顶部。

4.1.2 网络求值和执行控制元素

图形类编程语言采用“网络”概念。网络是内部连接的图形元素的最大集合。每个网络有一个位于网络左面的网络标号。网络标号用冒号“:”定界。网络标号可以是一个标识符或无符号十进制整数。例如,Network_Label1:N001;等都可表示为网络标号。使用网络标号,可使系统方便地从一个网络通过跳转进入另一个网络。

通常,编程系统对网络进行自动连续标号。在编程过程中,插入或删除一个网络时,所有网络的自动标号被自动更新。

IEC 61131-3 标准没有规定网络标号和网络图之间可插入注释。但为说明网络范围,可插入注释,注释表示方法与文本类编程语言相同,例如,(* 网络1 *)。注释不允许嵌套。

1. 网络的求值

网络图由图形对象组成。图形对象分为图形元素、连接和连接线,见表4-1。

(1) 梯形图编程语言中网络的求值

梯形图编程语言中,一个网络由一个左电源轨线和一个右电源轨线定界。电源轨线(Power Line)表示能流的起点和终点。通常,左电源轨线表示能流的起点,右电源轨线表示能流的终点。

网络对象位于左右电源轨线之间,例如,触点、线圈等逻辑元素、连接和连接线。图形元素可以相互连接或交叉,连接线可以汇合或分离,通过图形对象的各种组合表示它们的连接关系。网络的求值过程如下:

1) 从左电源轨线开始,其求值结果为1。

2) 根据连接的图形元素,确定能流是否能够流过该图形元素,若能够流过该图形元素,其求值结果为1,若不能流过该图形元素,其求值结果为0。

3) 依次从左向右进行求值,直到最后的线圈。如果线圈左侧的求值结果为1,则该线圈被执行,反之,如果该线圈左侧的求值结果为0,则该线圈不被执行。

4) 网络的求值过程是组成网络的图形元素状态的传递过程。

对梯形图网络的每个网络进行上述的求值过程,直到整个网络图求值结束,并开始新一轮的求值过程。

(2) 功能块图编程语言中网络的求值

功能块图编程语言中,一个网络由网络图形元素组成。网络图形元素包括函数、功能块、水平连线、垂直连线、连接和控制执行元素等。一个函数或功能块用于实现特定函数或运算。它有若干输入端和若干输出端,用于与其他变量、函数或功能块连接。功能块图网络的求值过

程如下：

- 1) 对每个网络元素的所有输入进行求值。
- 2) 根据网络元素的功能,确定其输出。
- 3) 如果网络元素的输入信号来自前级的网络元素,应先确定该前级网络元素的输出。
- 4) 执行控制元素前所有网络元素的求值过程结束后,确定该执行控制元素是否执行。

2. 网络元素求值规则

(1) 网络求值基本规则

网络和网络元素求值规则如下：

- 1) 网络和网络元素的求值次序可以与它们的标号或显示的次序不同。
- 2) 在已给网络的求值被重复以前,不需要对所有网络进行求值。
- 3) 直到对网络元素的所有输入求值后,才进行网络元素的求值。
- 4) 一个网络元素的求值过程只在该网络元素的所有输出状态被求值后才算完成。
- 5) 一个网络的求值过程只在该网络元素的所有输出被求值后才算完成,而不管该网络元素是否包含执行控制元素。
- 6) 梯形图编程语言中,除了被执行控制元素修改外,网络求值次序从上向下进行。
- 7) 功能块图编程语言中,当程序组织单元包含多于一个网络时,制造商应提供与执行有关的方法,通过该方法用户能够确定网络求值的次序。

(2) 反馈路径的求值规则

在功能块编程语言中,一个网络输出参数的值返回到同一网络中作为输入参数时,称该连接为反馈路径,相应的变量称为反馈变量。当反馈路径不明显画出时,称为隐式回路,反之则是显式反馈路径。

图 4-1 中,反馈变量是 RUN。图 4-1a 中,从与(&)函数输出到或(≥ 1)函数输入的连接线是显式反馈路径。图 4-1b 中,用反馈变量(重复变量名)RUN 表示该变量来自网络一个输出并作为该网络的一个输入,这是隐式回路。

在图形编程语言中使用反馈路径时,应服从下列规则：

- 1) 功能块图编程语言中,使用如图 4-1a 所示显式回路表示反馈路径。用户可根据显式反馈路径,使用与执行有关的方法来确定该显式回路中网络求值的次序。如图 4-1b 所示反馈路径是用反馈变量 RUN 来组成一个隐式回路。

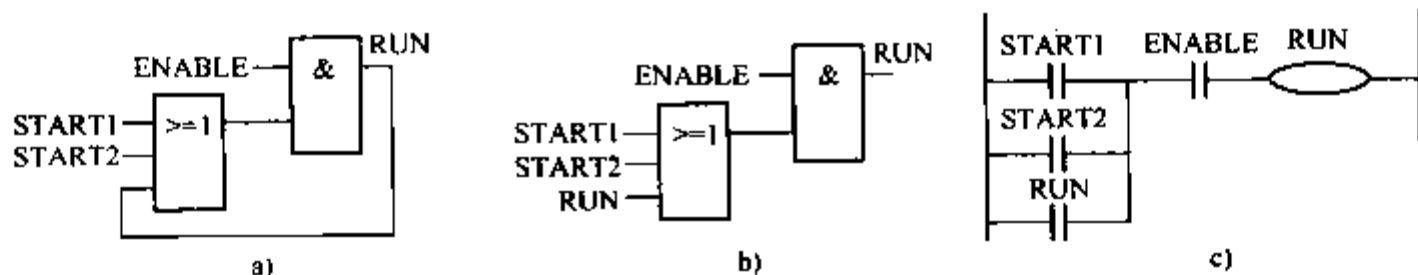


图 4-1 反馈路径示例

a) 显示回路 b) 隐式回路 c) LD 语言等价的梯形图

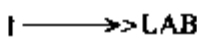
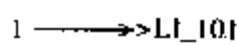
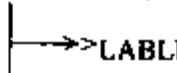
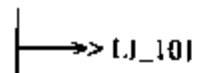
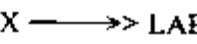
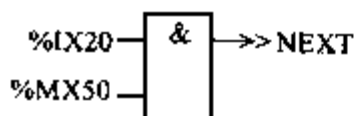
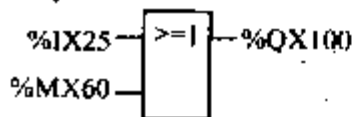
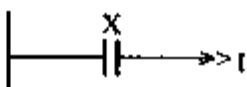
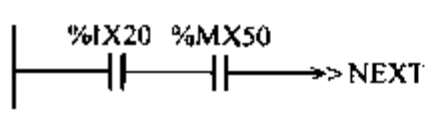
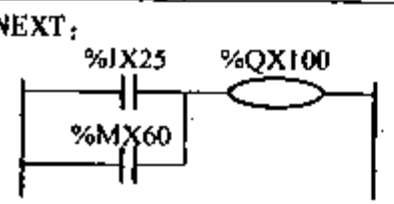
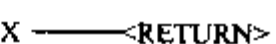
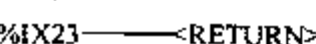
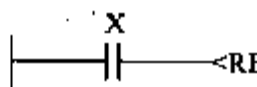
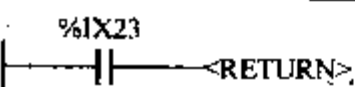
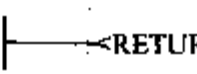

- 2) 在第一次网络求值时使用反馈变量的初始值,例如,它可采用该数据类型的约定初始值,其后,反馈变量的值根据反馈信号确定。在网络元素的下一次求值前,反馈变量的新值能够被作为该网络的输入使用。图 4-1c 是用梯形图编程语言表示的反馈路径。

3) 一旦带有反馈变量的元素作为输出已经被求值,则反馈变量的新值被保持到在该反馈变量下一次求值以前。

3. 执行控制元素

在图形编程语言中使用执行控制元素实现程序执行的控制,例如,跳转、跳转返回等。表 4-2 显示了图形执行控制元素。

表 4-2 图形执行控制元素

执行控制类型		执行控制元素的图形符号	示 例	说 明
无条件跳转	FBD 语言			它是 X = 1 的条件跳转的特例
	LD 语言			直接连接到左电源轨线
条件跳转	FBD 语言	 当 X 条件满足时跳转到标号为 LABELB 处执行		当 %IX20 和 %MX50 都为 1 时跳转到 NEXT
				跳转目标为标号 NEXT 开始的程序
	LD 语言	 当 X 条件满足时跳转到标号为 LABELB 处执行		当 %IX20 和 %MX50 都为 1 时跳转到 NEXT
				跳转目标为标号 NEXT 开始的程序
跳转返回	FBD 语言			条件跳转返回
	LD 语言			当 %IX23 为 1 时条件跳转返回
		 END_FUNCTION 或 END_FUNCTION_BLOCK	 FUNCTION_BLOCK 或 END_FUNCTION_BLOCK	无条件跳转返回从函数、功能块无条件跳转返回

(1) 跳转执行控制元素

跳转 (Jump) 执行控制元素用终止于双箭头的布尔信号线表示。跳转信号线开始于一个布尔变量、一个函数或功能块的布尔输出或梯形图的能流线。

跳转分无条件跳转和条件跳转两类。

梯形图编程语言中,当跳转信号线开始于梯形图的左电源轨线时,该跳转是无条件的。功能块图编程语言中,如果跳转信号线开始于布尔常数 1 时,该跳转是无条件的。

当跳转信号线开始于一个布尔变量、函数或功能块输出时,该跳转是条件跳转。只有程序

控制执行到特定网络标号的跳转信号线,而其布尔值为1(True)时才发生跳转。

无条件跳转是条件跳转的特例,即跳转条件恒为1的条件跳转。

(2) 跳转目标

在程序组织单元中,跳转目标(Target)是发生跳转的该程序组织单元内的一个网络标号。它表示跳转发生后,程序将从该目标开始执行。例如,表4-2中的NEXT就是一个跳转目标。

(3) 跳转返回

跳转返回(Return)分条件跳转返回和无条件跳转返回两类。

条件跳转返回适用于从函数、功能块的条件返回,当条件跳转返回的布尔输入为真(1)时,程序执行将跳转返回到调用的实体。当布尔输入为假(0)时,程序执行将继续在正常方式进行。

无条件跳转返回由函数或功能块的物理结束来提供。如表4-2所示。在梯形图编程语言中,将RETURN语句直接连接到左轨线表示无条件返回。

【例4-1】 跳转语句的示例。

电动机控制中,输出电动机起动信号为A,如果发送输出起动信号A后2s内,电动机没有起动,即电动机起动的反馈信号B为0,应跳转报警程序(跳转目标为C)。变量声明如下:

```
VAR_INPUT
  A, B: BOOL;
END_VAR
VAR
  TMR1: TON;
END_VAR
```

图4-2为该报警跳转示例的FBD程序。示例中,采用定时器功能块、与运算函数及非运算函数。如果2s计时到,电动机起动的反馈信号B为0,则经非运算和与运算,使跳转条件满足,因此,程序跳转到标号C的报警程序。这里,C是标号,不是输出变量。因此,在变量声明段中不需要声明。

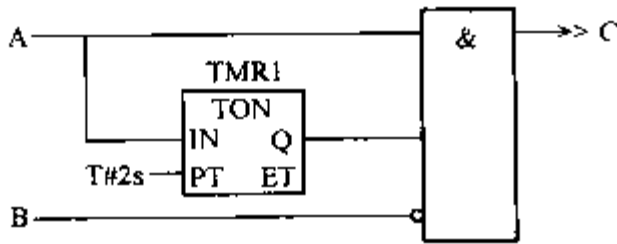


图4-2 报警跳转示例的FBD程序

4.2 梯形图编程语言

4.2.1 传统梯形图编程语言的缺点

- (1) 不同的PLC产品,梯形图的图形符号和工具不统一
- (2) 结构化和递解程序的分解缺乏有效工具

很难用优先等级来中断复杂的梯形图程序。即很难中断大程序使其进入较小的程序部分,因此,只能在梯形图程序的一部分设置读和设置触点,用程序的另一部分来输出。而封装,即在程序块内隐藏内部信息,使它不被程序的其他部分干扰的能力对建立良好质量和维护软件是非常重要的。

数据封装的缺失在由不同编程人员编写的大程序中是十分严重的问题。每个编程人员必须确信当大程序的不同部分之间通信时,只有一定量的全局寄存器和存储区被修改。通常,程

序块的一个内部数据寄存器被另一个模块不完整的代码修改是极危险的,显然,用 PLC 控制的大型工厂和机械时,这样的故障会造成不可预见和灾难性的后果。

(3) 对软件的再重复使用仅提供有限的工具

大型程序常常反复使用同样的逻辑策略或算法,例如,大型石化工业的油气安全系统中,这些系统使用同样的逻辑策略用于检测成百上千的工厂区域的火焰压力,并使用火焰检测表决策法。由于火焰检测器不可靠,因此,控制策略是在一个区域至少应有 2 个或以上的检测器检测到火灾时才启动灭火器。这样的系统需要成百上千遍地编写重复的梯形图梯级,而只需要极少部分的修改来读取不同输入或设置不同输出。这造成了程序规模的扩大和程序管理的困难。

在功能块中,重复的逻辑功能通过被多次调用,来缩小程序的大小,使生成的程序更容易维护。但是,很多基于梯形图程序的传统 PLC 中,采用调用方法的可重复使用的软件工具通常是有限的,且不便于使用。

(4) 对寻址和数据结构的使用缺乏有效工具

梯形图中不能使用结构化数据。典型的数据存储和寻址只能在单一的位存储器或寄存器。使用单一的位存储对保持单一的标志位和数据输入/输出。类似地,寄存器通常可保持 16 位,可用于计数器和模拟数值。然而,对许多复杂应用,通常有必要将数据组合起来组成结构。传统 PLC 的梯形图程序通常对结构化数据没有合适处理工具,而结构化数据至今仍是大多数高级语言,例如,PASCAL 和 C 的标准特性。

例如,考虑检测压力的传感器被连接到 PLC 的数字输入,当开始压力达到某一定值时,PLC 的硬件 I/O 管理系统将在 PLC 的内存建立单一的位,用于反映被连接的传感器状态。然而,单一的位不能全面反映传感器的所有状态,实际应用可能需要传感器被禁止或被设置到测试模式,即在传感器成为活动状态时才被记录,传感器工作在预先描述的周期时传感器的报警被解除,然后再继续。

所有与传感器有关的额外信息存储在单一的结构中,它能够用通用的惟一名称来寻址。对复杂的梯形图程序,这样的数据需要经 PLC 输出用于作为电子表格,以致不同传感器的偶发的寻址数据位变化的概率非常高。而没有格式化数据结构的支持,传统 PLC 编程系统通常不能在这些数据出错时给出警告。

(5) 复杂顺序的建立工具有限

许多工业应用需要 PLC 程序有一个或几个操作顺序,它取决于被控过程的各种操作状态。例如,考虑一条生产线传送带皮带的起动:程序先需检查某些辅助装置是否在已知的初始状态;然后,它可能需要去设置速度来驱动传送带马达作为一系列的步,在每个速度改变时,PLC 需要检查整条传输线的速度是否稳定等。当达到工作速度后,PLC 需要激活在传输线上的其他机器和装置等。

对于大型程序,可能有几百步这样的顺序,由于顺序控制与应用逻辑混合在一起,以至整个性能不容易理解。

(6) 程序执行方面缺乏有效工具

梯形图程序执行时,从第一梯级开始,程序扫描其输入、处理和输出,最后所有输出送 PLC 的物理地址。这种简单的执行过程很容易理解。对大多数 PLC 应用这种程序执行方法是足够的。一般来说,梯形图程序执行的频率取决于梯形图程序的长度和复杂程度。增加梯级使扫描时间变长。当设计系统需要在规定周期内响应时,这会造成问题。

例如,考虑 PLC 系统控制一台注塑机,PLC 需要控制电动机来保持塑料给料螺旋和挤压退出的同时,控制机械联锁、操作员显示和管理塑料给料的传送。为保护价格昂贵的挤压桶,当某报警信号检测到熔化压力过压时,PLC 应在 100ms 内快速驱动电动机。熔化塑料的压力或熔融压力是挤压机的临界参数。由于挤压桶阻塞,熔融压力的突然增加造成高内压,会使挤压桶扭曲或使其爆破。

这就要求梯形图的临界段检测报警条件需要在 50ms 内执行来适应 100ms 的死时,为此,编程员需要花费时间和努力设置程序的不同部分,进行时间的分配来适应这些控制要求。通常,可采用运行更快的梯级逻辑,例如,每 50 ms 执行,并切换来实现。

IEC 61131-3 标准的软件模型规定任务可具有周期执行的功能,因此,不同程序的执行控制可采用不同的周期。

(7) 算法操作的工具使用麻烦

梯形图执行一些算法非常麻烦。虽然有不同的技术可采用,但最常用的一种方法是用算法模块,这些模块能够由于梯级的激活而触发。但对复杂的算法,梯形图就十分复杂而且维护也变得困难。

4.2.2 梯形图的组成元素

梯形图编程语言是历史最久远的一种编程语言。梯形图源于电气系统的逻辑控制图,逻辑图采用继电器、触点、线圈和逻辑关系图等表示它们的逻辑关系。图 4-3 表示电气系统逻辑图,用于电气设备的开停控制。

IEC 61131-3 标准规定梯形图可采用的图形元素有电源轨线、连接元素、触点、线圈、函数和功能块等。

传统的梯形图中,触点假设有 ON 和 OFF 状态,等效于 1 和 0,触点表示有关布尔变量的状态。它提供只读的存取到布尔变量的状态,它不能用于改变变量的值。线圈提供只写的存取,仅能够用于更新有关布尔变量的状态。

IEC 61131-3 标准中没有规定触点和线圈的数据类型,它可以是字节、字或双字等数据类型,因此,它可用一个类似的梯级表示多个相同的控制逻辑。例如,8 个电动机控制用一个梯级实现。

1. 电源轨线

梯形图电源轨线(Power Rail)的图形元素亦称为母线。其图形表示是位于梯形图左侧和右侧的两条垂直线。左侧的垂直线称为左电源轨线,或左母线。右侧的垂直线称为右电源轨线,或右母线。在梯形图中必须绘制左电源轨线,但有时可不绘制右电源轨线。图 4-4 表示左电源轨线、右电源轨线和附加的水平连接线。

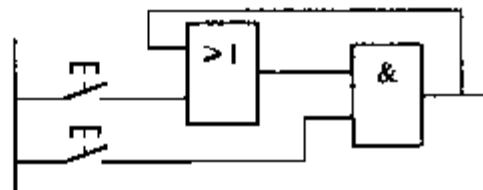


图 4-3 电气系统的逻辑图

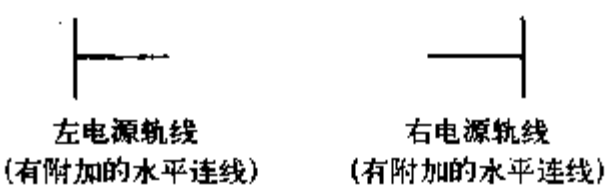


图 4-4 电源轨线的图形表示

梯形图中,能流从左电源轨线开始,向右流动,经连接元素和其他连接在该梯级的图形元素后到达右电源轨线。采用图形元素的状态表示能流的流动状态。

2. 连接元素和状态

(1) 连接元素

梯形图中,各图形符号用连接元素(Link Element)连接。连接元素的图形符号用水平线和垂直线表示。图4-5是水平和垂直连接元素的图形表示。

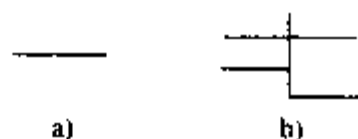


图4-5 连接元素的图形表示

a) 水平连接元素

b) 垂直连接元素

(带连接的水平连接)

(2) 连接元素的状态

连接元素的状态是一个布尔量。连接元素将最靠近该元素左侧图形符号的状态传递到该元素的右侧图形元素。因此,某一连接元素的状态为1,表示最靠近该元素左侧图形符号的图形元素状态为1,或表示能流已经流过左侧图形符号表示的图形元素。反之,如果某一连接元素的状态为0,表示最靠近该连接元素左侧图形符号的图形元素状态为0,或能流不能流过左侧图形符号表示的图形元素。

(3) 连接元素状态的传递规则

连接元素的状态从左向右传递,实现能流的流动。状态的传递遵守下列规则:

1) 连接到左电源轨线的连接元素,其状态在任何时刻为1,它表示左电源轨线是能流的起点。右电源轨线类似于电气图中的零电位,因此,对右电源轨线不定义其状态。

2) 水平连接元素用水平线表示,水平连接元素从它的紧靠左侧的图形元素开始将该图形元素的状态传递到紧靠它右侧的图形元素。

3) 垂直连接元素总是与一个或多个水平连接元素连接,即它由一个或多个水平连接元素在每一侧与垂直线相交组成。垂直连接元素的状态根据与其连接的各左侧水平连接元素的状态或运算表示。因此,垂直连接元素的状态根据下列规则确定:

- 如果左侧所有连接的水平连接元素的状态为0,则该垂直连接元素的状态为0。
- 如果左侧的一个或多个水平连接元素的状态为1,则该垂直连接元素的状态为1。
- 垂直连接元素的状态被传递到与其右侧连接的所有水平连接元素,但不能传递到与其左侧连接的所有水平连接元素。

【例4-2】 连接元素及状态传递的示例。

图4-6是连接元素及其状态的示例。连接元素1与左电源轨线连接,其状态为1;连接元素2与连接元素1连接,其状态从连接元素1传递,因此,其状态为1;连接元素3是垂直连接元素,它与水平连接元素1连接,由于连接元素1的状态为1,因此,连接元素3的状态为1;同样,连接元素3的状态传递给连接元素4,使连接元素4的状态为1。连接元素2和4将其状态1传递给图形元素5

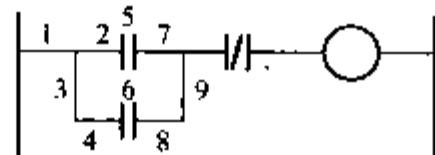


图4-6 连接元素及状态的示例

和6,由于图形元素5和6为常开触点,因此,连接元素7和8的状态经图形元素的传递而成为0;由于连接元素9的左侧所有水平连接元素的状态为0,因此,连接元素9的状态也为0。

如果图形符号5的状态为1,即闭合状态,则连接元素7的状态为1,从而使连接元素9的状态为1。同样,如果图形符号6的状态为1,即闭合状态,则连接元素8的状态为1,它也使连接元素9的状态为1。

4) 连接元素的输入和输出数据类型必须相同。标准中,触点和线圈等图形元素的数据类型并不局限于位,因此,连接元素的输入输出数据类型相同才能保证状态正确传递。

3. 触点

(1) 触点的类型


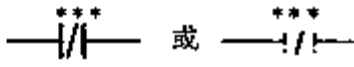
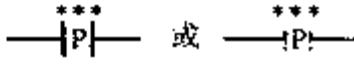
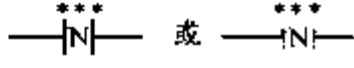
触点是梯形图图形元素。梯形图的触点(Contact)沿用了电气逻辑图的触点术语,用于表示布尔变量状态的变化。触点是向其右侧水平连接元素传递一个状态的梯形图元素。该状态是触点左侧水平连接元素状态与相关变量和直接地址状态进行布尔与运算的结果。触点不改变相关变量和直接地址的值。

按静态特性分类,触点分为常开触点(Normally Open Contact, NO)和常闭触点(Normally Closed Contact, NC)。常开触点指在正常工况下,触点断开,其状态为0。常闭触点指在正常工况下,触点闭合,其状态为1。

按动态特性分类,触点分为上升沿触发触点,或正跳变触发触点(Positive Transition Contact)和下降沿触发触点,或负跳变触发触点(Negative Transition Contact)。表4-3是触点图形元素的图形符号表示。

触点的变量名称直接标注在触点的上方。

表 4-3 触点图形元素的图形符号表示

类 型		图 形 符 号	说 明
静触点	常开触点		当该触点左侧连接元素的状态为1时,如果触点布尔变量的值为1,则状态1被传递,使右侧连接元素的状态为1。反之,如果该触点的布尔变量值为0,则右侧连接元素的状态为0
	常闭触点		当该触点左侧连接元素的状态为1时,如果触点布尔变量的值为0,则状态1被传递,使右侧连接元素的状态为1。反之,如果该触点的布尔变量值为1,则右侧连接元素的状态为0
转换触点	正跳变触发触点		当该触点左侧连接元素的状态为1的同时,检测到有关变量从0转变为1,则该触点右侧连接元素的状态从0跳变到1,并保持一个求值周期,然后返回到0。其他时间该触点右侧连接元素的状态为0
	负跳变触发触点		当该触点左侧连接元素的状态为1的同时,检测到有关变量从1转变为0,则该触点右侧连接元素的状态从0跳变到1,并保持一个求值周期,然后返回到0。其他时间该触点右侧连接元素的状态为0

注:触点图形符号的垂直线也可用符号“!”表示,如图所示。***表示触点的布尔变量名称。触点的状态是该布尔变量的值。

(2) 状态传递规则

根据触点的状态和与该触点连接的左侧连接元素的状态,按下列规则确定其右侧图形符号的状态。

1) 当静态触点的左侧图形元素状态为1时,才能将其状态传递到触点右侧图形元素,根据下列原则进行传递:

- 如果触点状态为1,则该触点右侧图形元素的状态为1。
- 如果触点状态为0,则该触点右侧图形元素的状态为0。

2) 当静态触点的左侧图形元素状态为0时,不管触点的状态如何,都不能将其状态传递到触点的右侧图形元素,即其右侧图形元素的状态为0。

3) 正跳变触发触点在触点左侧图形元素状态为1的同时,其有关变量从0 转变为1,则该触点的右侧图形元素状态从0 跳变到1,并保持一个求值周期,然后自动跳变到0。其他时间该触点右侧图形元素状态为0。这称为上升沿触发。

4) 负跳变触发触点在触点左侧图形元素状态为1的同时,其有关变量从1 转变为0,则该触点的右侧图形元素状态从0 跳变到1,并保持一个求值周期,然后自动跳变到0。其他时间该触点右侧图形元素状态为0。这称为下降沿触发。

4. 线圈

(1) 线圈的类型

线圈是梯形图的图形元素。梯形图中的线圈(Coil)沿用了电气逻辑图的线圈术语,用于表示布尔变量状态的变化。

根据线圈的不同特性,可分为瞬时线圈、锁存线圈和跳变触发(检测)线圈等。正和负跳变线圈用于检测能流的状态,用这些类型的线圈,当有关的变量仅设置为 ON 时,对梯级执行一次求值。表 4-4 是不同线圈的图形符号表示。

表 4-4 线圈的图形符号

图 形 符 号		说 明
瞬时 线圈	线圈 *** ——()——	左侧连接元素的状态被传递到有关的布尔变量和右侧连接元素
	取反线圈 *** ——(/)——	左侧连接元素的状态被取反,并被传递到右侧连接元素。如果左侧连接元素状态为0,则该线圈的布尔变量状态为1,反之亦然
锁存 线圈	置位线圈 *** ——(S)——	当左侧连接元素的状态为1,该线圈的布尔变量被置位并保持,直到由 RESET(复位)线圈复位
	复位线圈 *** ——(R)——	当左侧连接元素的状态为1,该线圈的布尔变量被复位并保持,直到由 SET(置位)线圈置位
跳变触 发线圈	正跳变触发线圈 *** ——(P)——	当左侧连接元素从0 跳变到1 时,该线圈的布尔变量状态变为1,并保持一个求值周期,然后,返回到0。在其他时刻,左侧连接元素的状态被传递到右侧连接元素
	负跳变触发线圈 *** ——(N)——	当左侧连接元素从1 跳变到0 时,该线圈的布尔变量状态变为1,并保持一个求值周期,然后,返回到0。在其他时刻,左侧连接元素的状态被传递到右侧连接元素

注:线圈图形符号也可用圆或椭圆表示,如图 4-7 所示。*** 表示线圈的布尔变量名称。线圈的状态是该布尔变量的值。在修正版中,取消保持线圈的图形符号,因为它们可用 RETAIN 属性表示。

【例 4-3】 线圈的示例。

图 4-7a 中,触点 1 是常开触点,触点 2 是常闭触点,触点 3 是瞬时线圈 3 所带的自保常开触点。

手动按下触点 1 对应的启动按钮,使触点 1 对应的布尔变量为1。根据上述状态传递规则,因触点 2(对应于停止按钮)的布尔变量为0,因此,触点 1 的状态 1 经触点 2 的传递,传送到瞬时线圈 3,使线圈 3 对应的布尔变量为1。在下一个求值周期,触点 3 的状态为1,因此,不管触点 1 的状态是否为1,根据传递规则,线圈 3 的布尔变量值保持为1。

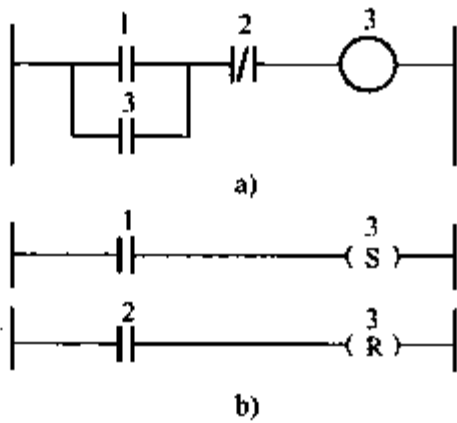


图 4-7 线圈的示例

a) 瞬时线圈 b) 锁存线圈

手动按下停止按钮,触点 2 对应的布尔变量值变为 1,触点 2 右侧连接元素的状态变为 0,触点 3 的状态值 1 不能传递到线圈 3,使线圈 3 对应的布尔变量变为 0。在下一个求值周期使触点 3 的布尔变量值变为 0,并使线圈 3 的布尔变量值保持为 0,直到下一次手动按下启动按钮。

图 4-7b 中,触点 1 和触点 2 都是常开触点,当触点 1 对应的布尔变量状态为 1 时,其右侧连接元素状态为 1,它使置位线圈 3 的布尔变量值变为 1,并保持该值。

当按下停止按钮时,触点 2 的状态变为 1,其右侧连接元素的状态为 1,它使复位线圈的布尔变量值变为 1,并保持到下一次触点 1 的状态为 1。

上述两种图形表示的控制系统都是用于复位优先的控制系统,即当置位和复位按钮同时按下时,线圈应处于复位(停止)的状态。

须注意的是,采用不同线圈时,为获得相同的控制功能,触点的类型是不同的。例如,图 4-7a 中采用一个常开触点,一个常闭触点。而图 4-7b 中都采用常开触点。此外,从网络看,图 4-7a 是一个梯级(网络),图 4-7b 是两个梯级(网络)。

(2) 线圈的传递规则

线圈是将其左侧水平连接元素状态毫无改变地传递到其右侧水平连接元素的梯形图元素。在传递过程中,将左侧连接的有关变量和直接地址的状态存储到合适的布尔变量中。取反线圈将其左侧水平连接元素状态取反后传递到其右侧水平连接元素的梯形图元素。

置位和复位线圈将左侧水平连接元素状态用于对线圈对应的变量进行置位和复位,并保持变量的值直到下一次复位和置位。

正跳变和负跳变线圈在其左侧水平连接元素状态从 0 到 1 和从 1 到 0 的瞬间,将有关线圈的变量保持一个求值周期,其他时间将其左侧水平连接元素状态传递到其右侧水平连接元素。

标准没有规定线圈的右侧不能连接其他图形元素,因此,其应用范围扩展,程序可简化。例如,可允许出现双线圈(指两个线圈串联连接),允许在线圈右侧连接触点和线圈等。例如,图 4-22 所示的程序中,由两个线圈组成了信号灯的闪烁程序。

【例 4-4】 线圈状态的传递。

图 4-8 显示了线圈状态的传递过程。图中,线圈 a 连接到左轨线,因此,其状态为 1,并传递到其右侧的连接元素,它将状态 1 传递给触点 b。当触点 b 闭合时,触点 b 右侧的连接元素状态为 1,并经水平和垂直连接元素分别连接到线圈 c 和 e。由于触点 b 闭合,因此,其状态被传递到线圈 e,使其状态为 1。同时,使线圈 c 的状态为 1,并经状态的传递使线圈 d 的状态也为 1。注意,传统 PLC 中不允许图 4-8 所示的梯形图程序。

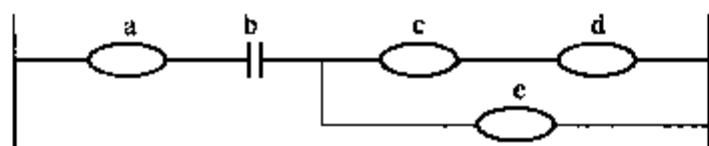


图 4-8 线圈状态的传递

5. 函数和功能块

梯形图编程语言支持函数和功能块的调用。在函数和功能块调用时应注意下列事项:

- 1) 在梯形图中,函数和功能块用一个矩形框表示。函数可以有多个输入参数和一个返回

参数。功能块可以有多个输入参数和多个输出参数。输入列于矩形框的左侧,输出列于矩形框的右侧。函数和功能块名称显示在框内的上中部,函数和功能块的实例名列于框外的上中部。用函数和功能块的实例名作为其在项目中的惟一识别。

2) 为了保证能流可以通过函数或功能块,每个被调用的函数或功能块至少应有一个输入和一个输出(或返回)参数。为了使被连接的功能块执行,至少应有一个布尔输入经水平梯级连接到垂直的左电源轨线。

3) 功能块调用时,可以直接将实际参数值填写在该内部形参变量名的功能块外部的连接线附近。

【例 4-5】 功能块调用时实参的设置。

图 4-9 调用延时输出 TON 功能块。功能块的形参 PT 设置为 T#5s,其值可直接填写在与 PT 连接的功能块外部连接线附近。同样,形参 ET 被连接到变量 ETIM,该变量也被直接填写在与 ET 连接的功能块外部连接线附近。实际应用时,只需选中有关形参变量,就可从弹出的对话框输入对应的实际参数值。

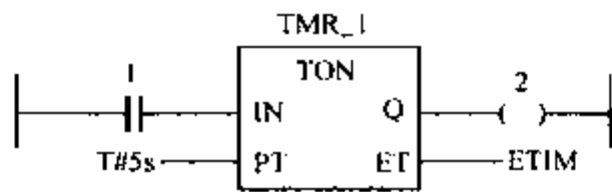


图 4-9 调用功能块时实参的设置

可以看到,功能块 TON 的输出 Q 被连接到瞬时线圈 2。这表示当触点 1 的布尔变量值 1 保持 5 s 后,输出线圈 2 就被激励,直到触点 1 的布尔变量值返回到 0 时,输出线圈 2 才失励。如果触点 1 的布尔变量值 1 保持的时间小于 5 s,则线圈 2 不会被激励。

示例中,TMR_1 是功能块 TON 的实例名。当不需要使用 ET 变量值时,可以不连接该变量。例如,图中可不连接 ETIM。

【例 4-6】 功能块连接。

图 4-10 是在梯形图中,电动机控制功能模块的连接。

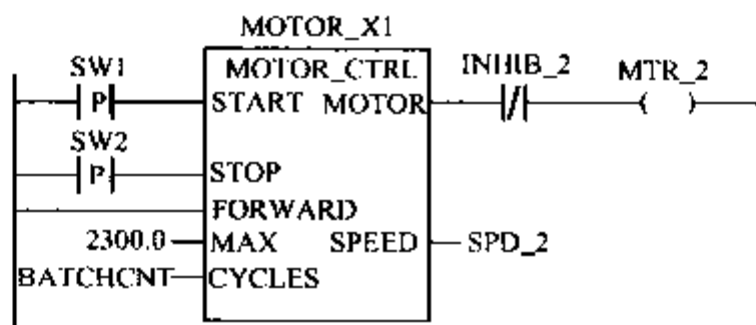


图 4-10 在梯形图中功能模块的连接

图中的功能模块 MOTOR_CTRL 是用于控制电动机的专用功能模块。它的布尔量输入 START 和 STOP 直接从电源轨线引出,来控制触点,因此,SW1 和 SW2 是布尔变量,表示外部的按钮状态。触点是正跳变触发,因此,功能模块仅当开关第一次被按压时,接收能流。FORWARD 输入直接连接到左电源轨线,表示该输入永久在 ON(True)的状态。

与传统的梯形图不同,标准规定梯形图中的触点输入可以是其他数据类型的变量。例如,在上面的示例中 CYCLES 输入的值来自变量 BATCHCNT,它是时间数据类型。此外,功能模

块的输入也可以连接到常数,例如,MAX 连接到实数文字值 2300.0。功能模块输出也可通过其他运算后再输出,例如,图中仅当功能模块输出 MOTOR 为 ON 和变量 INHIB_2 为 OFF 时,变量 MTR_2 才为 ON。

4) 功能块调用时,输入的形参变量也可连接到其他函数或功能块的返回值或输出参数。输出的形参变量也可连接到其他函数或功能块的输入参数。

【例 4-7】 功能块参数的连接。

图 4-11 中,功能块 TON 实例 TON_1 的输入 IN 连接到函数 AND 的返回值。同样,RS 功能块实例 RS_1 的输入 SET 连接到函数 AND 的返回值,RESET1 连接到功能块 TON 实例 TON_1 的输出 Q,该输出也作为 TON 实例 TON_2 的输入 IN。

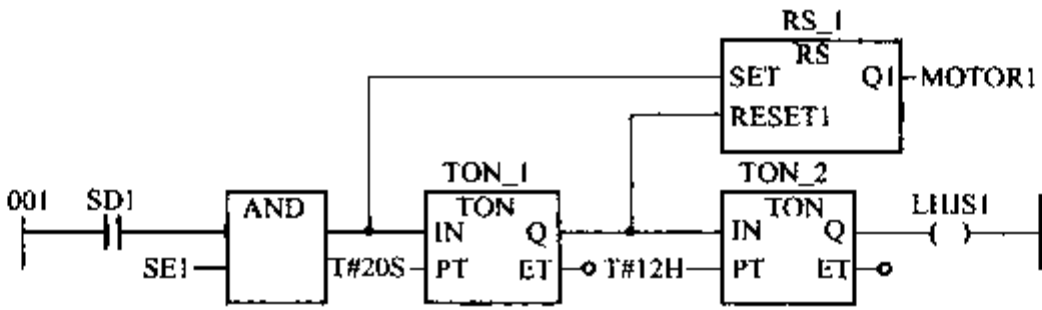


图 4-11 功能块参数连接的示例

5) 输入或输出变量的取反可用 NOT 函数或直接在连接线处绘制圆。实际应用时,可选中相应的参数,并从弹出对话框中选择取反函数,其图形显示通常是与矩形边相切的圆。

【例 4-8】 显示变量取反的示例。

图 4-12 中,TON 功能取块实例 TON_1 的输入 IN 是变量 RUN_2 的取反信号,同样,RS 功能块实例 RS_1 的输出取反后送变量 LH2。

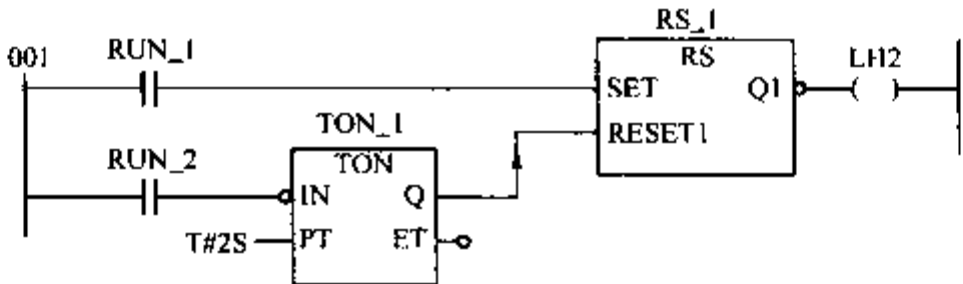


图 4-12 显示变量取反的示例

6) 如果没有 EN 和 ENO 专用输入输出参数,则函数和功能块自动执行。并将允许的状态传递到下游。功能块调用时,至少应有一个输入和输出参数是布尔数据类型。这些变量必须有一个与左电源轨线和右电源轨线进行直接或间接的连接。例如,例 4-9 中,IN 间接地通过触点 1 连接到左电源轨线,Q 间接地通过线圈 2 连接到右电源轨线。

【例 4-9】 功能块调用中 ENO 的设置。

图 4-13 显示了使用和不使用 EN 和 ENO 时的梯形图。当不使用 EN 和 ENO 时,布尔输入 START_1 用于定时器 PUMP_1 的启动,定时器输出作为线圈 P_1RUN 的激励信号。使用 EN 和 ENO 时,EN 的信号 A 可来自其他触点、函数或功能块输出信号,而 ENO 输出信号可用于出错报警,从而为控制系统提供更多信息。

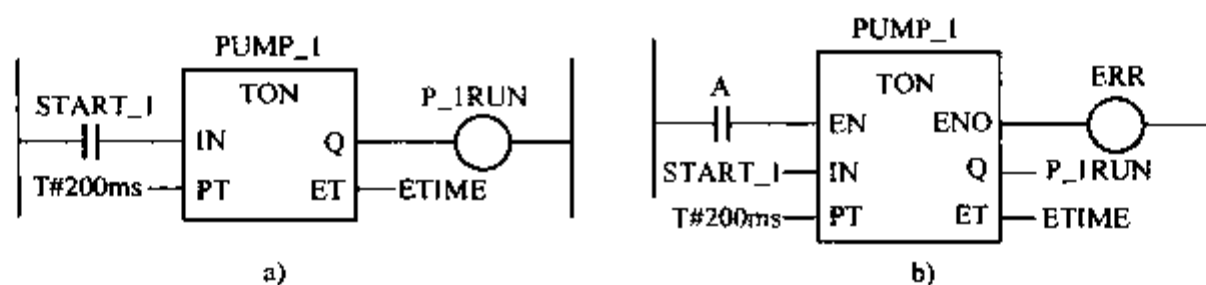


图 4-13 EN 和 ENO 的使用
a) 不使用 EN 和 ENO b) 使用 EN 和 ENO

7) 使用 EN 和 ENO 时,应在变量声明中声明 EN 和 ENO 变量的数据类型,它们分别是输入和输出变量。当使用这些变量时,应根据下列规则确定功能块的操作:

- 当功能块实例调用时,EN 的值是 FALSE(0),则功能块本体定义的操作不被执行,ENO 的值由可编程控制器系统设置到 FALSE(0)。
- 如果 ENO 的值被可编程控制器系统设置到 TRUE(1),则功能块输入的实际赋值被实现,功能块本体定义的操作被执行,这些操作包括赋布尔值到 ENO。
- 如果求 ENO 输出的值为 FALSE(0),则功能块输出(VAR_OUTPUT)的值保持在它们上一次调用时的求值。

【例 4-10】 EN 和 ENO 的实际应用。

某加热罐用 3 个温度检测点检测温度,当最高温度大于 500.0℃时,就使 COOL 冷风机运转。温度的检测定时进行,由 CHECK 给出脉冲信号触发。

图 4-14 是使用 EN 和 ENO 来控制加热罐的示例。当定时检测信号 CHECK 从 OFF 变到 ON 时才执行该梯级的求值。首先,正跳变脉冲使 MAX 函数求值,它的返回值是 3 个温度检测点的最高值。同时,MAX 的 ENO 输出为 1,其次,该使能输出使 GT 函数求值,它将来自 MAX 函数的最高温度与 500.0 比较,如果最高温度超过 500.0,则 GT 函数的返回值被置位,即变量 COOL 变为 ON,否则,COOL 被置 OFF。任何一种情况下,与 GT 求值的同时,其 ENO 输出置位,并设置 COMPLETE 为 1,表示温度检测已经完成。

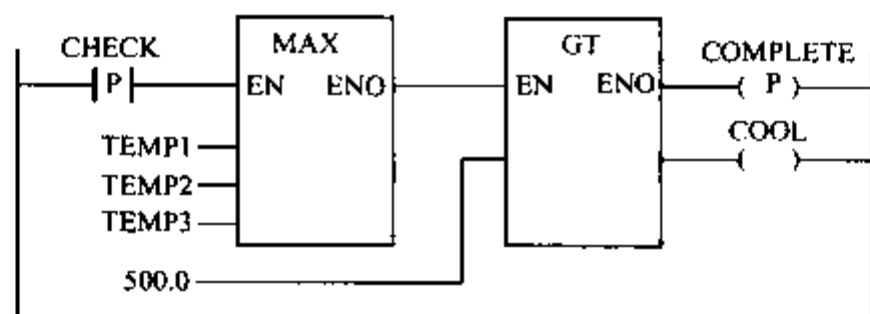


图 4-14 EN 和 ENO 的实际应用

8) 标准功能块总有一个输出变量 Q(或 QU,QD,Q1),它是布尔数据类型。

函数可以提供附加的 EN 输入和附加的 ENO 输出。当用图形编程语言时,EN 提供一个能流信号到函数,当 EN 为 ON,即 1 时,函数使能,因此,能够根据它的输入求值来获得输出。当函数的求值完成,ENO 输出能被用于提供能流到其他函数或线圈。

可建立包含反馈回路的梯形图梯级,反馈回路中,有一个或多个值用于触点,当梯级求值时,函数和功能模块的输入来自这些变量值的更新。

4.2.3 梯形图的执行

梯形图的执行过程依据从上到下、从左到右的顺序进行。

1. 梯形图的执行过程

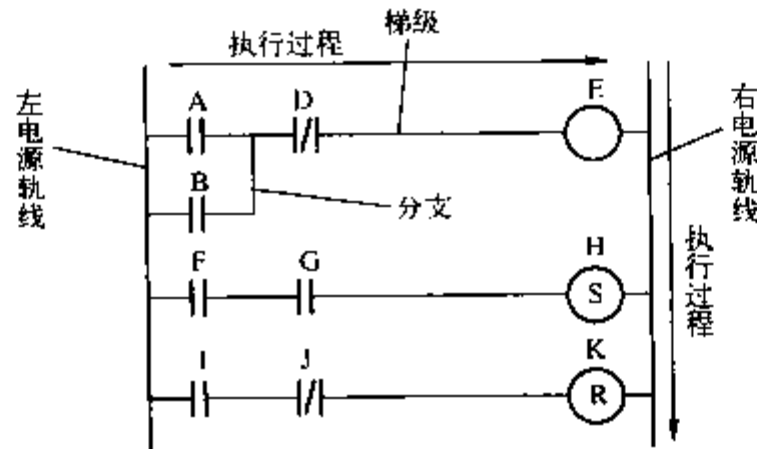
梯形图采用网络结构,一个梯形图的网络以左电源轨线和右电源轨线为界。

梯级是梯形图网络结构的最小单位。从输入条件开始,到一个线圈的有关逻辑的网络称为一个梯级(Ladder Rung),一个梯级包含输入指令和输出指令。

输入指令在梯级中执行比较、测试的操作,并根据操作结果设置梯级的状态。例如,测试梯级内连接的图形元素状态的结果为1,输入状态就被置1。输入指令通常执行一些逻辑运算操作、数据比较操作等。

输出指令检测输入指令的结果,并执行有关操作和功能,例如,使某线圈激励等。通常,输入指令与左电源轨线连接,输出指令与右电源轨线连接。

梯形图执行时,从最上层梯级开始执行,从左到右确定各图形元素的状态,并确定其右侧连接元素的状态,逐个向右执行,操作执行的结果由执行控制元素输出,直到右电源轨线。然后,进行下一个梯级的执行过程。图4-15显示了梯形图的执行过程。



当梯级中有分支出现时,同样依据从上到下、从左到右的执行顺序分析各图形元素的状态,对垂直连接元素根据上述有关规则确定其右侧连接元素的状态,从而逐个从左向右、从上向下执行求值过程。在梯形图中,没有反馈路径的求值不是很明确。其所有外部输入值与这些有关的触点必须在每个梯级求值以前被求值。

2. 梯形图的执行控制

为了使控制梯形图的执行按非常规的执行过程进行,可采用4.3.1节介绍的执行控制的有关图形元素。

(1) 跳转和跳转返回

梯形图网络结构中,用跳转和跳转返回等图形符号表示跳转的目标、跳转的返回及跳转的条件等。图4-16显示了跳转指令的执行过程。

图中,当跳转条件满足时,程序跳转到 LABEL 标号的梯级开始执行,直到该部分程序执行到 RETURN 时,程序返回到原断点后的一个梯级,并继续执行。

(2) 反馈

使用反馈变量时应注意,反馈变量只有在其值能够惟一确定时才是正确的表示。例如,图

4-1c 的表示方法是正确的。调用函数和功能块时,如果上游函数或功能块输入是其下游函数或功能块的输出时,由于在求值过程中,并不知道下游函数或功能块的输出值,因此,图 4-17a 所示连接是不正确的。图 4-17b 的连接是正确的。

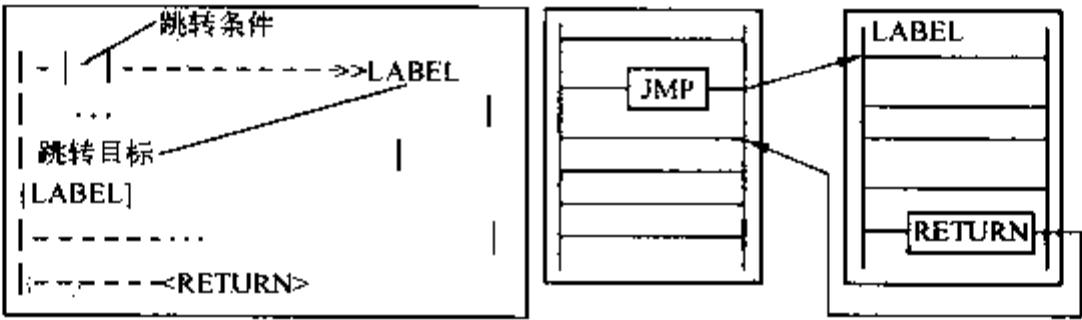


图 4-16 跳转和跳转返回指令的执行过程

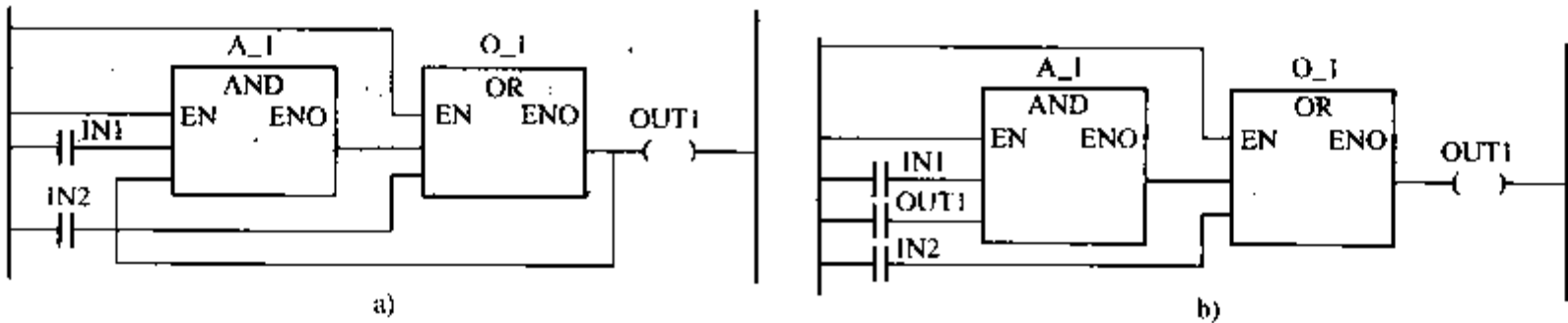


图 4-17 反馈变量的编程方法
a) 不允许的显式编程 b) 允许的隐式编程

在图 4-17b 中,与函数 A_1 的输入信号是 OUT1,它在第一次求值的执行过程时,采用其初始值,它可以是该输出变量的约定初始值,也可以是该输出变量数据类型的约定初始值,因此,求值过程可以进行。在下次求值的执行过程中,因 A_1 函数已有上次的 O_1 函数输出,因此,可用该输出值作为本次求值的数据。编程时应注意反馈变量在初次求值执行过程中是否有值,如果有值,则该程序是可执行的,反之,图 4-17a 所示程序在第一次求值时,A_1 函数没有 OUT1 初始值,不能够执行求值过程,因此,这种连接是错误的。

(3) 扫描方式执行程序

梯形图程序按该程序先后次序用扫描方式执行。

【例 4-11】 采用扫描方式执行梯形图程序的示例。

图 4-18 是一个示例程序,用于说明梯形图程序的扫描执行过程。

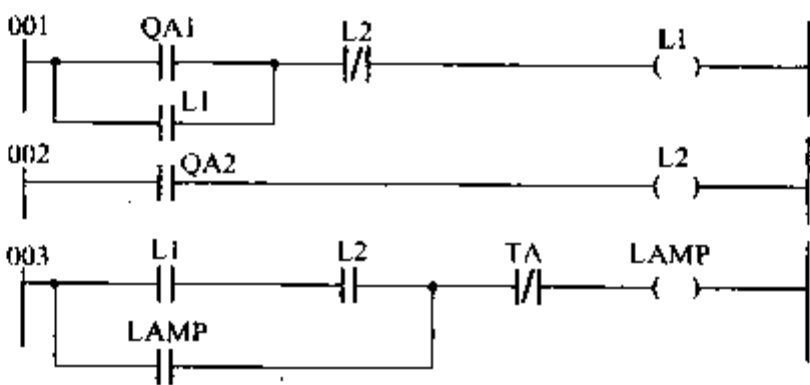


图 4-18 扫描方式执行梯形图程序的示例

该程序有两个按钮信号 QA1 和 QA2,一个停止按钮 TA,一个输出信号灯 LAMP,中间变量是 L1 和 L2。程序执行时,按下 QA1,则 L1 状态为 1,并经自保使其继续保持 L1 状态为 1。它使第三梯级(网络 003)的 L1 的状态为 1。

按下 QA2,则 L2 状态变为 1,根据程序顺序扫描原则,它先扫描第三梯级,因 L1 已经为 1,当 L2 为 1 时,使 LAMP 的状态变为 1,并经自保,使 LAMP 继续保持在 1 的状态,因此,信号灯 LAMP 点亮,并保持。

如果将上述梯形图用电气逻辑图表示为图 4-19,当按下按钮 QA1 后,J1 继电器激励,其触点 J1 闭合,除自保外,还使第三行触点 J1 闭合。当按下按钮 QA2 时,J2 继电器激励,其常闭触点先断开,使 L1 继电器失励,并使常开触点 J2 闭合,但因 J1 失励,第三行 J1 触点已经断开,J3 继电器不能被激励,信号灯 LAMP 也不能点亮。

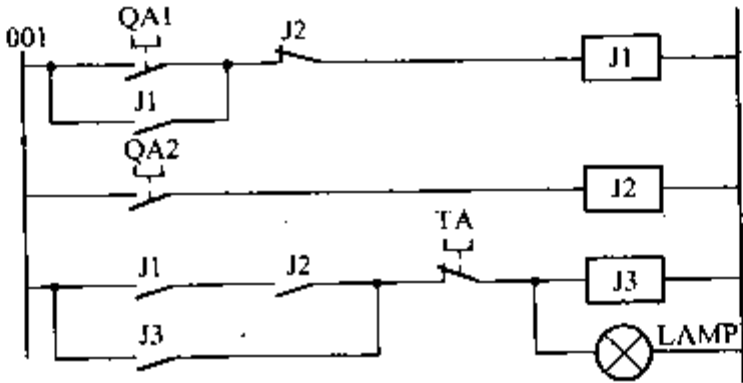


图 4-19 电气逻辑图

图 4-18 所示梯形图中,如果将第 001 梯级与第 002 梯级的位置上下互换,则按下按钮 QA2 时,根据程序扫描顺序,将使 L2 常闭触点状态变为 0,最终不能点亮信号灯。示例说明下列两点:

1) 电气逻辑图与梯形图的执行顺序不同,电气逻辑图采用并行执行方式,梯形图程序采用顺序扫描方式。一些电气逻辑图不能实现的逻辑关系,用类似的梯形图程序可以实现。反之,也存在一些电气逻辑图可实现而梯形图程序不能实现的情况。

2) 梯形图程序采用顺序扫描方式执行程序时,程序执行的先后顺序关系到程序的执行结果。示例说明因执行程序顺序不同,一些程序可从能够实现某种逻辑运算变为不能实现。

4.2.4 示例

1. 液位控制系统

(1) 控制要求

图 4-20 所示加热储罐中,进料液体经进料泵加压进入储罐,在蒸汽加热后,料液变成蒸汽从储罐顶部排出。

控制要求是使液位在 LSH-1 和 LSL-1 之间变化,防止进料被全部蒸发或进料过多,液相进入蒸汽排出管。

(2) 系统分析

该控制系统有两个输入信号,采用液位开关

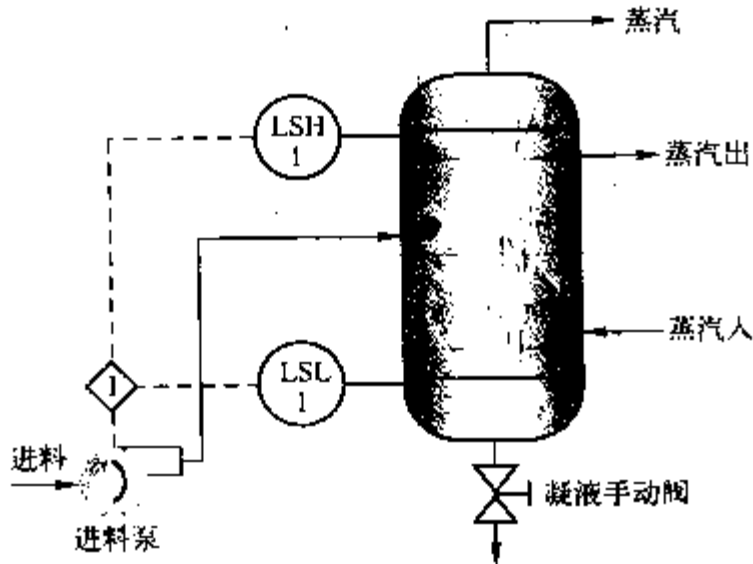


图 4-20 液位控制系统流程简图

检测液位,假设所采用的液位开关接点的信号状态如下:当液位高于液位开关的检测位置时,接点断开;当液位低于液位开关的检测位置时,接点闭合。

液位开关接点用于控制进料泵的起停。根据上述液位开关接点的状态,控制要求如下:

- 1) 实际液位低于低液位开关 LSL-1 的检测位置时,如果按下自动运行起动按钮 START,则进料泵电动机起动。
- 2) 实际液位高于低液位开关 LSL-1 的检测位置,但仍低于高液位开关 LSH-1 的检测位置时,进料泵应保持运转。
- 3) 实际液位高于 LSH-1 高液位开关的检测位置时,进料泵自动停止运转。
- 4) 按下自动运行停止按钮 STOP 时,不管液位实际位置在何处,进料泵停运。

根据控制要求分析,需增加两个输入信号,即自动运行起动信号 START 和自动运行停止信号 STOP。此外,进料泵的起动信号 PUMP 作为 PLC 的输出信号,当 PUMP 为 1 时,经电动机控制中心,进料泵运转,当 PUMP 为 0 时,进料泵停运。为说明泵运行状态是自动和手动状态,增加一个自动运行状态输出信号 AUTO。输入输出地址分配如表 4-5 所示。

表 4-5 输入输出地址分配表

变 量 名	LSL_1	LSH_1	START	STOP	AUTO	PUMP
地址	%IX0.0	%IX0.1	%IX0.2	%IX0.3	%QX0.0	%QX0.1
用途	低液位信号	高液位信号	运行起动	运行停止	自动状态	进料泵起动

(3) 编程

1) 变量声明。程序中变量声明如下:

```
VAR_INPUT
    START , STOP , LSL_1 , LSH_1 :BOOL;
END_VAR
VAR_OUTPUT
    AUTO , PUMP :BOOL;
END_VAR
```

2) 程序本体编程。根据上述分析和变量地址分配表,用梯形图编程语言编写的梯形图程序如图 4-21 所示。

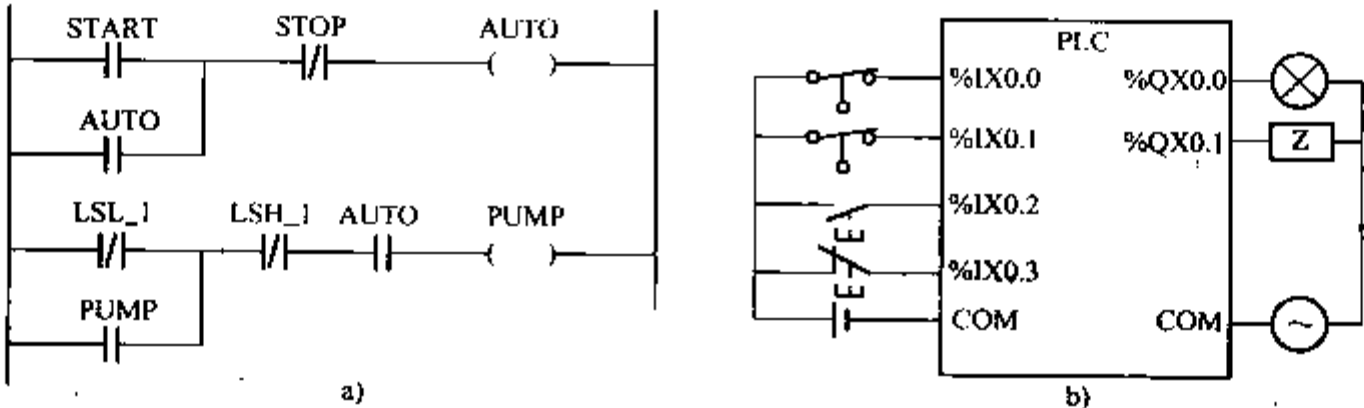


图 4-21 液位控制系统接线图和梯形图程序

a) 梯形图程序本体 b) PLC 接线图

图中,Z 是进料泵的接触器线圈;AUTO 信号用信号灯显示;输入信号的电源由 PLC 的内部电源供应,也可采用外部电源。

3) 程序执行。程序执行过程如下:

- 按下 START 按钮后,梯形图程序的第一梯级激励,输出变量 AUTO 为 1。因此,对应的信号灯点亮,同时,经自保接点使运行状态保持为自动状态。
- 开始时,液位低于低液位开关的检测位置,因此,LSL_1 为 1,常闭接点保持在闭合位置,因此,第二梯级激励,输出变量 PUMP 为 1,经接触器使泵电动机运转。
- 当液位高于低液位开关的检测位置,但低于高液位开关的检测位置时,进料泵仍为激励状态,直到实际液位高于高液位开关的检测位置才停止泵运转。
- 进料泵停运后,由于进料被蒸汽加热蒸发,液位开始下降,直到液位低于低液位开关的检测位置,起动进料泵为止。重复上述过程,能够保持液位在所需范围内变化。
- 自动运行过程中按下停止按钮,STOP 变为 0,AUTO 为 0,使进料泵停运。

实际应用时,应考虑液位开关一旦失效,会造成生产事故。为此,还需设置高高液位开关和低液位开关,用于联锁停车。

2. 闪烁信号灯

(1) 控制要求

用定时器 TON 实例 TON_1 和函数 XOR 可组成闪烁信号灯线路。该线路输出可使信号灯按一定的周期点亮和熄灭。用方波信号发生器实现。

(2) 编程

程序用图 4-22 所示的梯形图实现。

可以看到,IEC 61131-3 标准规定的梯形图程序中,允许多个函数和功能块连接在同一梯级,如图 4-22 所示。因为梯形图中各连接元素的状态可传递,因此,可以将信息传递到其连接的后续元素。

图 4-22 所示程序中有两个线圈串联连接,用于说明状态传递过程。程序实现 LAMP1 和 LAMP 的交替点亮和熄灭,实现闪烁信号灯的控制要求。

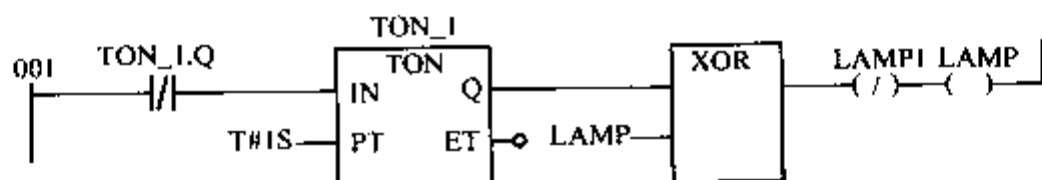


图 4-22 闪烁信号灯的梯形图程序

示例中,如果将 LAMP1 和 LAMP 位置改变,就无法实现所需控制要求。此外,传统可编程控制器的梯形图编程语言不允许在一个梯级上有双线圈串联出现,但标准是根据连接元素的右面状态来确定该连接元素的状态的。这表明,标准梯形图允许双线圈,也允许在双线圈之间有其他触点和调用等,因此,标准规定的梯形图程序应用更方便,范围更广泛。

3. DIVRMBK 功能块的梯形图描述

(1) 运算要求

用 DIVIDEND 和 DIVISOR 计算其商和余数。运算要求与第 3.2.3 节的功能块相同。

(2) 编程

功能块的变量声明与第 3.2.3 节的 DIVRMBK 功能块变量声明相同。用梯形图编程语言编写的功能块本体如图 4-23 所示。

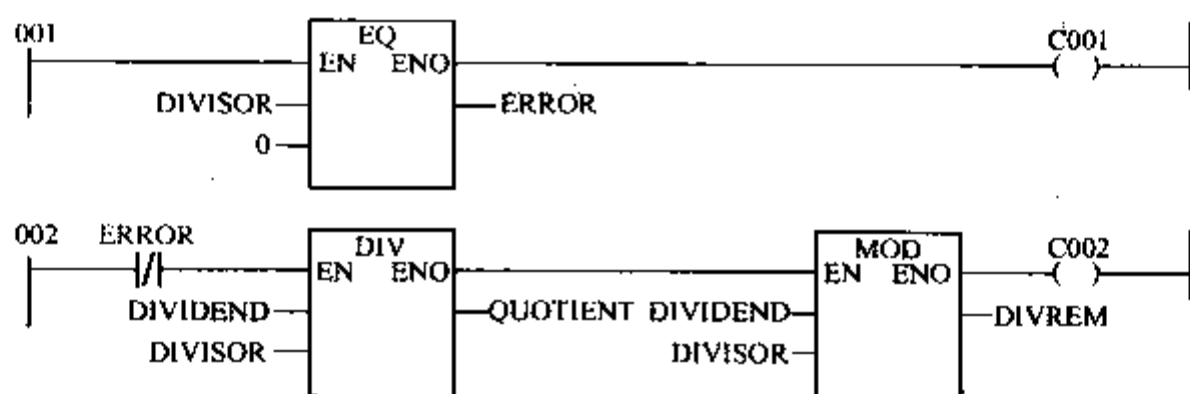


图 4-23 DIVRMBK 功能块的梯形图程序

该程序采用 EN 和 ENO 属性,第一梯级判别除数是否为零,第二梯级计算商和余数。

4. 火警报警系统

(1) 控制要求

图 4-24 是某火警区域示意图。它由 3 个火警监视器 FD1、FD2 和 FD3 监视,还有手动按钮 MAN1 用于清除火警报警。为避免火警监视器的不可靠而错误发出报警信号,该系统采用两个或两个以上的监视器翻转时才报警的控制手段,该系统被称为 3 取 2 出(2oo3:2 out of 3 voting)联锁系统。

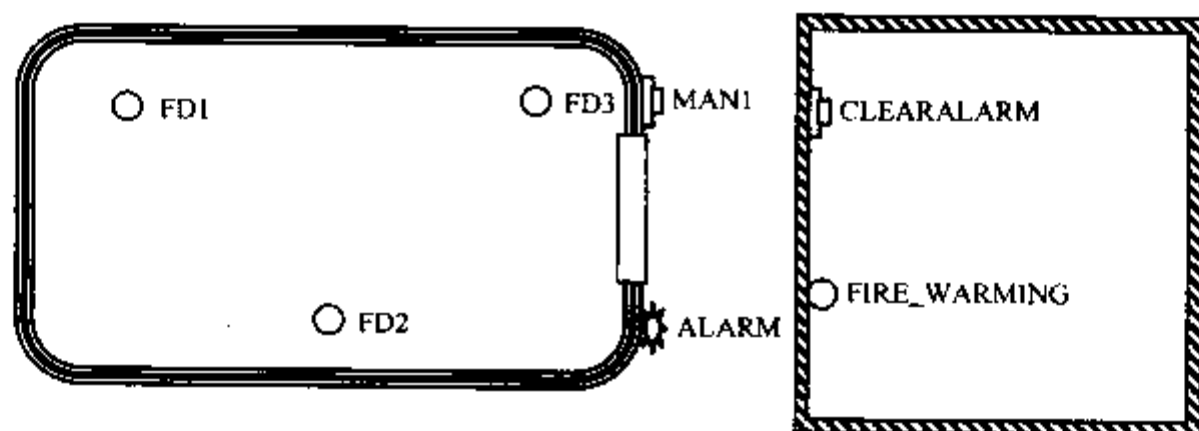


图 4-24 火警区域示意图

为清除报警,应设置清除报警的按钮 CLEARALARM。

(2) 编程

整个火警控制系统的变量声明如下:

```
VAR_INPUT
    FD1 AT %IX0.0 :BOOL;
    FD2 AT %IX0.1 :BOOL;
    FD3 AT %IX0.2 :BOOL;
    MAN1 AT %IX0.4 :BOOL;
    CLEARALARM AT %IX0.5 :BOOL;
END_VAR
VAR_OUTPUT
    ALARM :AT %QX0.0 :BOOL;
    FIREWARMING :AT %QX0.1 :BOOL;
END_VAR
```

火警控制系统程序本体用图 4 - 25 的梯形图表示。

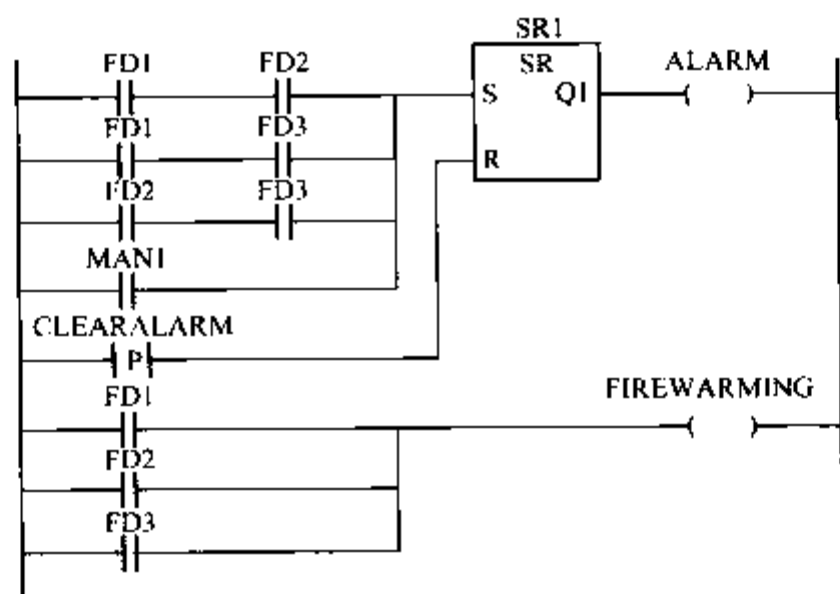


图 4 - 25 火警控制系统的梯形图程序

图中,如果任两个检测器为 ON,则标准双稳功能模块 SR 的实例 SR1 报警功能模块被设置为 ON,它将驱动报警线圈 ALARM 工作。

通过按 CLEARALARM 按钮才能清除报警。采用正跳变触点可以确保如果按钮保持在按下时,一个新的报警仍可发生。标准双稳功能模块 SR 是置位优先于复位的双稳功能模块,它能确保在 CLEARALARM 信号发生而两个或多个检测器是 ON 时报警仍可继续在 ON。

如果任何一个火警检测器为 ON,一个火警警告的指示器 FIREWARMING 被点亮。如果在报警被清除后它仍保持点亮,则既可能是检测器故障,也可能是仍有一个火警。

由 CLEARALARM 按钮产生的正跳变脉冲信号也可用边沿检测功能块 R_TRIG 的输出获得。示例仅说明梯形图的应用,实际应用时,控制系统要复杂得多。

5. 抢答器

(1) 控制要求

知识竞赛时,需要有抢答器。控制要求是主持人先按清除(兼开始)按钮,熄灭各抢答者处的信号灯,并宣读有关试题,当主持人讲开始抢答时按下清除(兼开始)按钮,则最早抢答者的信号灯被点亮。

(2) 系统分析

假设抢答器系统有 4 位抢答者,则系统有 4 个抢答输入,记为 NUM1 ~ NUM4,有 4 个抢答输出,记为 LAMP1 ~ LAMP4,有一个清除兼开始按钮,记为 START。

抢答器系统当主持人按钮释放时开始抢答,因此,可采用下降沿触发的单按钮启停控制抢答的开始。为此,需要 CI、C2 和 RUN 3 个内部变量。

(3) 编程

变量声明如下:

```
VAR_INPUT
    START AT %IX1.0 :BOOL;
    NUM1 AT &IX0.0 :BOOL;
    NUM2 AT &IX0.1 :BOOL;
    NUM3 AT &IX0.2 :BOOL;
```

```

        NUM4 AT &IX0.3 :BOOL;
END_VAR
VAR_OUTPUT
    LAMP1 AT &QX0.0 :BOOL;
    LAMP2 AT &QX0.1 :BOOL;
    LAMP3 AT &QX0.2 :BOOL;
    LAMP4 AT &QX0.3 :BOOL;
END_VAR
VAR
    C1, C2, RUN :BOOL;
END_VAR

```

图 4-26 是抢答器系统的程序。图中,001 梯级的程序是下降沿单按钮触发程序。002 梯级的程序是抢答器部分的程序。当抢答者人数增加或减少时,只需要增加或减少有关的分支程序和对应的触点。在主持人按开始按钮前抢答的参赛者,其按下的动作无效。

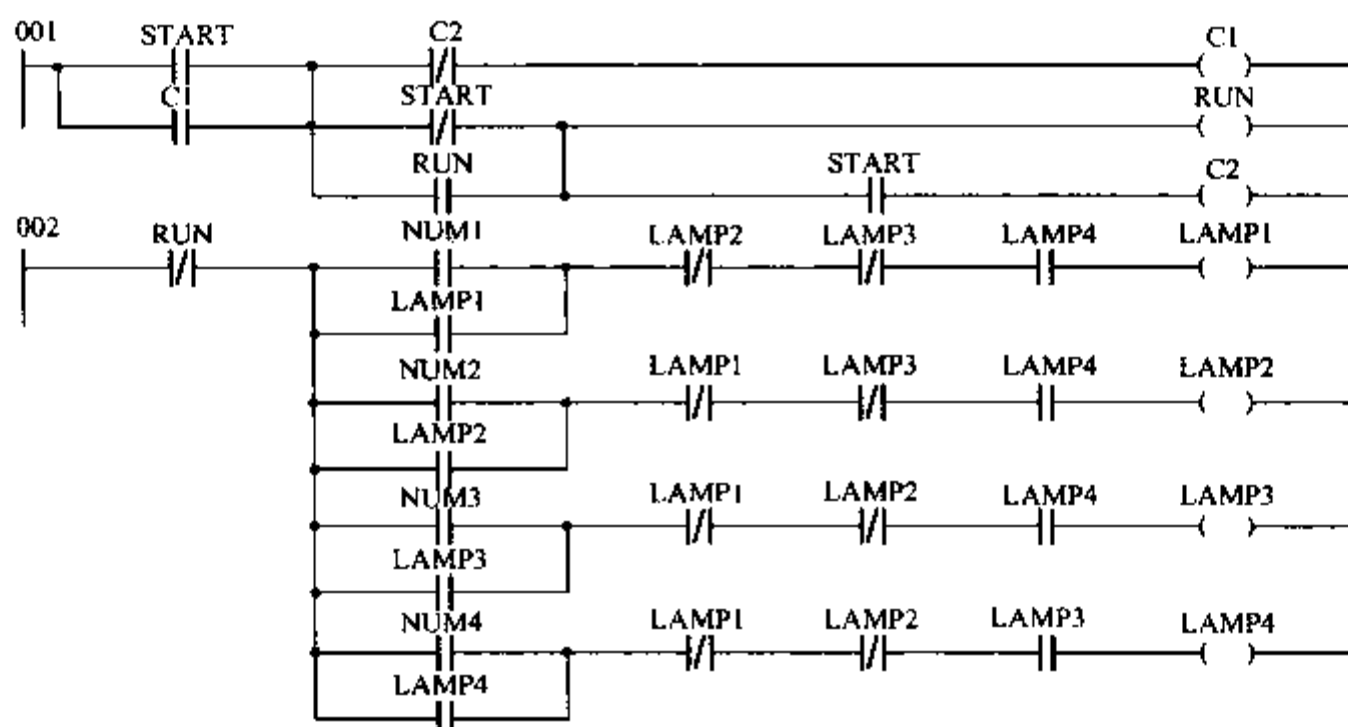


图 4-26 抢答器系统程序

6. pH 控制系统

(1) 控制要求

在废水处理过程或发酵过程中,常常采用 pH 控制系统。pH 控制系统的被控对象具有非线性和时滞特性,常用非线性和时滞补偿控制方案。但在简单控制方案中也可采用如下的控制策略进行控制:当 pH 测量值超过某一设定酸度时,等待一定时间,然后加入一定时间的碱液。控制要求如图 4-27 所示。设 pH 超过设定限值时,接点 PHH 闭合,加碱阀为 A。控制方案的设计原则是“看一看,调一调”。

1) 当 pH 控制在线性区段时,可假设控制过程中 pH 的变化呈现线性特性,即加碱液或酸液进行中和时,pH 的变化呈现线性特性。通常,当设定值上限 SP_H 和设定值下限 SP_L 之差较小时,线性关系成立。

2) 设发酵过程中 pH 值从 SP_L 变化到 SP_H 所需时间为 T,加碱后 pH 值从 SP_H 变化到 SP_L

的时间为 T_2 , 则延时时间可设置为 $T_1 = T/2$ (即程序中的 100S), 加碱控制阀打开的时间为 T_2 (即程序中的 50S)。

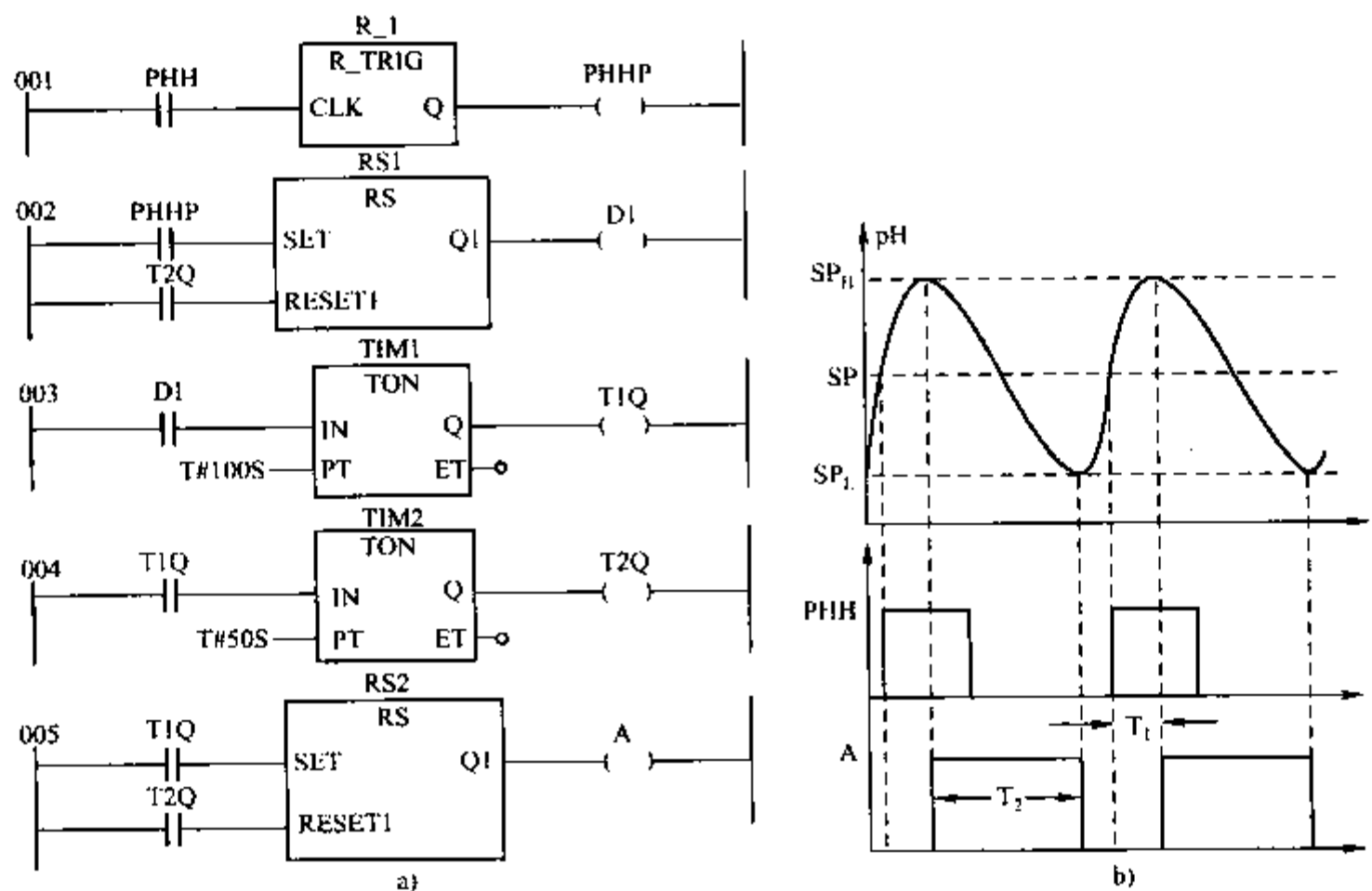


图 4-27 pH 控制梯形图程序和信号波形

- 3) 实际 pH 控制的设定值 $SP = (SP_H + SP_L)/2$ 。减小 SP_H 和 SP_L 之间的差值有利于提高控制精度。
- 4) 加碱阀 A 的启动条件是定时器 T1 的计时时间到, 因此, 程序中用 T1Q 作为启动条件, 加碱阀 A 的停止条件是定时器 T2 的计时时间到, 因此, 程序中用 T2Q 作为停止条件。
- 5) 定时器 T1 的启动条件是 pH 到设定限值 SP , 因此, 用触点 PHH 的上升沿触发 R_1 功能块, 并用 RS 功能块将其信号 D1 记忆。定时器 T2 的启动条件是定时器 T1 的计时到。
- 6) D1 的停止条件是定时器 T2 计时到, 以便下一周期的控制。

(2) 编程

根据上述控制要求, 用梯形图编程语言编写 pH 控制系统, 如图 4-27 所示。程序中, 将 PHH 的上升沿触发信号作为到达 SP_H 的标志。

4.3 功能块图编程语言

4.3.1 功能块图图形符号和功能块的组合

功能块图编程语言将各种功能块连接起来实现所需控制功能。功能块图编程语言源于信号处理领域, 它是 IEC 61499 标准的基础。由于该编程语言具有图形符号, 可图形连接, 且操作方便, 因此, 它将被广泛采用。功能块图编程语言的图形符号由函数、功能块和连接元素

组成。

功能块图编程语言能用于表示程序、函数和功能模块的行为。它也用于描述在 SFC 中的步、动作和转换。功能块图用于调用控制块之间的信号流。一个功能块图网络能够看作一个电气电路图,电气的连接用于描述在组件之间信号流的路径。功能块图的典型应用包括描述控制回路和逻辑。熟悉电气电路或使用图形编程技术的一些程序开发人员希望用功能块图的图形编程语言而不用结构化文本的文本语言。

1. 函数和功能块

(1) 在功能块图编程语言中函数和功能块的表示

函数和功能块是功能块图编程语言的基本图形元素。函数和功能块包括标准函数和功能块,以及衍生函数和功能块。功能块图程序由函数、功能块和执行控制元素组成。

函数图形符号是一个矩形框,矩形框内有函数名和函数参数。函数与外部连接是将函数参数用外部实参代入实现的。函数没有输出参数,但有返回值。函数的输入参数相同时,其返回值是相同的,因此,函数不具有记忆功能。通常认为返回值是函数的输出。

功能块图形符号也是一个矩形框。与函数的不同点是功能块有输出参数,功能块具有记忆功能。因此,当输入参数相同时,输出参数可能不同。例如,RS 功能块当输入参数都为 0 时,输出参数 Q 可以是 1(如果以前的输入参数 S 曾为 1)或 0(如果以前的输入参数 R1 曾为 1,或以前的输入参数 S 和 R1 都为 0)。

函数可设置 EN 和 ENO 参数连接。当 EN 为 1 时,该函数才被执行求值,它使返回值更新,并使 ENO 置为 1。当 EN 为 0 时,该函数不被执行求值,因此,返回值保持原值,同时,ENO 自动被设置为 0,它常被用作出错报警的信号。

功能块与函数类似,可以设置 EN 和 ENO 参数。

(2) 函数和功能块的连接原则

函数和功能块连接时,应遵循下列原则:

- 1) 上游函数或功能块的输出连接到下游函数或功能块的输入。
- 2) 函数返回值或功能块的输出可连接多个下游函数或功能块的输入。
- 3) 函数或功能块的输入只能连接一个变量、函数或功能块的输出,这体现了输入信号的惟一性。当多个函数或功能块的输出要同时传递数据到一个函数或功能块时,表示这些函数或功能块的输出是经过一个或运算函数输出的。

【例 4-12】 函数和功能块的连接示例。

图 4-28 是函数和功能块连接的示例。

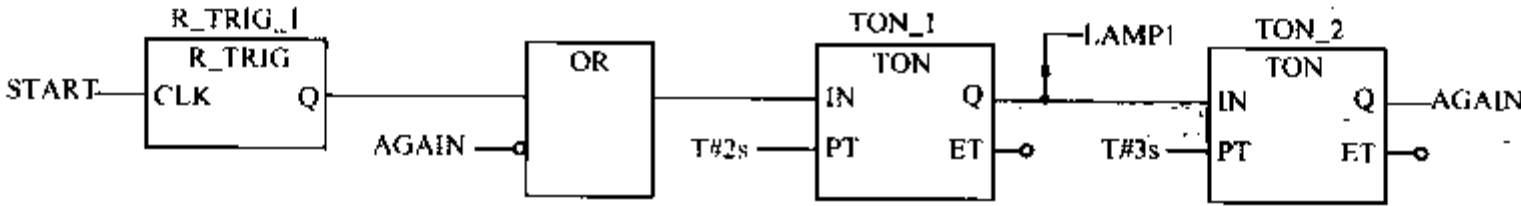


图 4-28 函数和功能块连接的示例

4) 函数或功能块的反馈回路宜采用反馈变量的形式,即用隐式显示描述。虽然,IEC 标准规定,功能块图编程语言也可采用显式形式描述。例 4-12 也显示反馈变量 AGAIN 采用隐式连接。

5) 函数或功能块的输入或输出可以不连接有关变量或函数、功能块。这时,输入变量取其数据类型的约定初始值,输出变量可不被存储。例 4-12 中,TON 功能块实例 TON_1 和 TON_2 的 ET 输出可以不连接变量,它表示在程序中,已经计时的时间不需要显示和观测。

6) 函数或功能块的输入或输出信号取反,可采用 NOT 函数连接,也可对相应的变量进行取反操作,即绘制一个圆,表示取反操作,例 4-12 中,AGAIN 变量取反后作为 OR 的输入信号。

7) 功能块中输入信号的上升沿触发信号和下降沿触发信号,可在功能块的矩形框内部用 > 图形符号表示,也可用边沿检测功能块实现。例 4-12 中的 R_TRIG_1 功能块实例用于实现上升沿边沿检测功能。

函数和功能块调用时,功能块图编程语言与梯形图编程语言采用类似的方法。

2. 网络结构

功能块图网络由函数、功能块、执行控制元素、连接元素和连接组成。函数和功能块用矩形框图形符号表示。连接元素的图形符号是水平或垂直的连接线。连接线用于将函数或功能块的输入和输出连接起来,也用于将变量与函数、功能块的输入、输出连接起来。当连接线汇合或分离时,用连接的图形符号表示。连接的图形符号是一个圆点,位于连接线的汇合或分离处。当连接线交叉,没有连接的圆点符号时,表示这些连接线没有相互的影响,即它们是相互独立的连接。

功能块图中不允许有多个输出连接线汇合,当发生这种情况时,可添加一个或(OR 或 >=1)函数(称为线或 Wire OR)来进行连接,如图 4-29 所示。

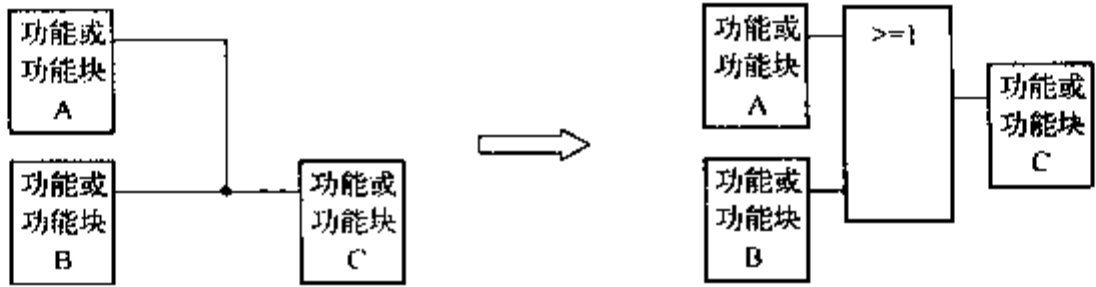


图 4-29 多个输出线连接的转换

执行控制元素用于控制程序的执行次序。例如,表 4-2 所示的跳转执行控制元素。因此,执行控制元素与标号应相呼应,以防止出现没有标号的现象。

函数和功能块输入和输出的显示位置并不影响其连接。不同 PLC 系统中,其位置可能不同,应根据制造商提供的函数和功能块显示参数的位置进行正确连接。

函数和功能块上连接的变量或参数显示位置也因不同产品而异,应根据实际产品的显示位置确定参数的连接位置。

一般而言,函数和功能块的形参显示在矩形框内,实参显示位置应在矩形框外,靠近该形参的位置。

4.3.2 功能块图的编程和执行

功能块编程语言中,采用函数和功能块编程,其编程方法类似于单元组合仪表的集成方法。它将控制要求分解为各自独立的函数或功能块,并用连接元素和连接将它们连接起来,实现所需控制功能。

标准没有对一个功能块图网络的大小和复杂性定义特定的限制。但它允许一个执行的限

制来限制支持一个配置的功能模块类型的数量和功能模块实例的数量。因此,这些限制影响由一定的 PLC 产品产生的功能块图网络的大小和复杂性。

1. 网络求值顺序

功能块编程语言编制的程序组成网络。对网络求值的原则如下:

- 1) 网络中某一函数或功能块的求值只有在它的所有输入已经求值后才能进行。
- 2) 网络中函数或功能块的求值次序有两种方法——系统设置或手动设置。它们都必须根据网络求值原则设置网络中各函数或功能块的求值次序。
- 3) 连接反馈变量的函数或功能块,宜采用隐式显示方式编程。在初始求值时,可对该反馈变量设置初始值,或使用该变量数据类型的约定初始值。在以后的求值时,可用上次该反馈变量的值。例如,一个单回路控制系统中,控制器的偏差信号是设定信号减测量信号,测量信号是反馈变量,因此,初始求值时,测量变量的值可置为 0 或其他值。便于网络求值过程进行。对有交叉的多个反馈变量,应确定各反馈变量初始值及求值次序。
- 4) 网络中函数或功能块的求值还与函数或功能块是否有 EN 和 ENO 信号有关。如果没有这些信号,则网络求值根据设置的次序进行。如果有这些信号,则有些函数或功能块可能因 EN 信号为 0 而不被执行。
- 5) 注意执行控制元素的介入。当网络中有执行控制元素时,如果跳转条件满足,网络的求值次序就会根据跳转的目标而变化。此外,返回也会改变求值执行的次序。
- 6) 用功能块图编程语言编写的程序组织单元包含多个网络时,制造商应提供与其执行有关的方法,通过这种方法,应用技术人员可方便地确定网络执行的次序。

2. 执行控制

功能块图编程语言中的执行控制元素有跳转、返回和反馈等类型。跳转和返回分为条件跳转或返回及无条件跳转或返回,详见第 4.3.1 节。反馈并不改变执行控制的流向,但它影响下次求值中的输入变量。

标号在网络中应是惟一的,标号不能作为网络中的变量使用。

一些编程系统中,因受显示屏幕的限制,当网络较大时,显示屏的一个行内不能显示多个有连接的函数或功能块,这时,可采用表 4-1 所示的连接符连接,连接符与标号不同,它仅用于说明网络的接续关系。一些编程系统采用滚屏方式显示,不使用连接符。

4.3.3 示例

1. 信号灯顺序点亮控制

(1) 控制要求

信号灯顺序点亮系统是一个时间顺序控制系统,其控制要求如下:

- 1) 信号灯顺序点亮控制系统的控制要求。闭合 START 开关后,每隔 1 s 点亮一个信号灯,共有 5 个信号灯,最后一个信号灯点亮后隔 1 s 全部信号灯熄灭。并重复上述过程,直到 START 开关断开。信号灯运行的波形图如图 4-30 所示。
- 2) 时间循环功能块。为编写信号灯顺序点亮控制系统,先编写时间循环功能块。该功能块的控制要求是当输入信号 START 为 1 后,先延时时间 T1,然后点亮,点亮时间为 T2,再熄灭 T3 时间,并开始循环(即点亮 T2,熄灭 T3)。

用 3 个定时器实现上述功能。功能块 CYCTIME 有一个输入 START,3 个定时器设定信号

T1、T2 和 T3,有一个输出 Q。

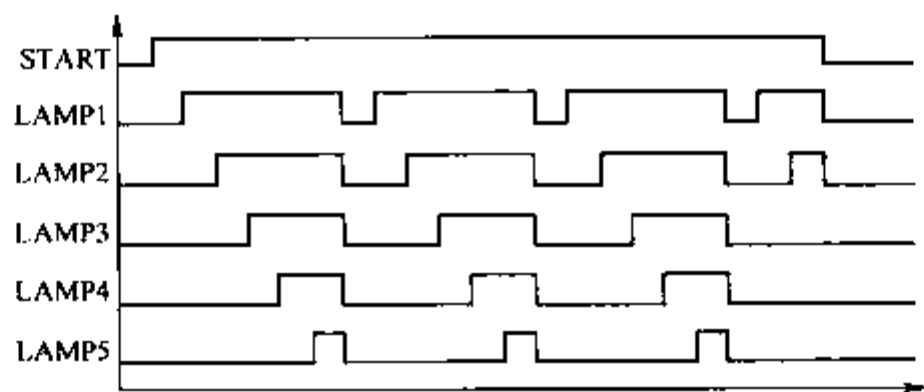


图 4-30 信号灯顺序点亮控制系统的信号波形图

① 变量声明。该功能块的变量声明如下：

```
VAR_INPUT
    START :BOOL;
    T1, T2, T3 :TIME;
END_VAR;
VAR_OUTPUT
    Q:BOOL;
END_VAR;
VAR
    TON_1,TON_2,TON_3 :TON;
END_VAR;
```

② 功能块程序。功能块本体程序用功能块图编程语言编写如图 4-31 所示。该功能块的图形表示如图 4-32 所示。

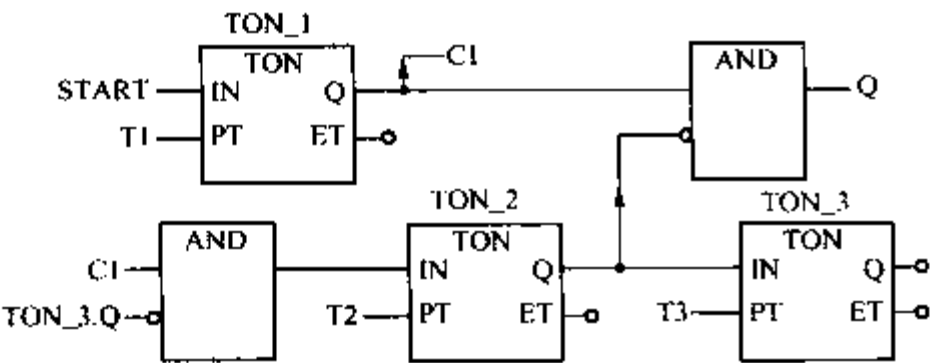


图 4-31 功能块 CYCIME 程序



图 4-32 CYCIME 功能块的图形描述

3) 变量声明。控制系统有一个输入变量 START,假设其地址为%IX0.0。有 5 个信号灯,作为输出变量,假设其地址分别为%QX0.0 ~ %QX0.4。采用 5 个 CYCTIME 功能块实例 CYC-TIME_1 ~ CYCTIME_5,变量声明如下：

```
VAR_INPUT
    START AT %IX0.0 :BOOL;
END_VAR
VAR_OUTPUT
```

```
LAMP1 AT %QX0.0:BOOL;
LAMP2 AT %QX0.1:BOOL;
LAMP3 AT %QX0.2:BOOL;
LAMP4 AT %QX0.3:BOOL;
LAMP5 AT %QX0.4:BOOL;
END_VAR
VAR
    CYCTIME_1, CYCTIME_2, CYCTIME_3, CYCTIME_4, CYCTIME_5:CYCTIME;
END_VAR
```

由于采用 CYCTIME 功能块,因此,各信号灯的程序十分简单,见图 4-33。

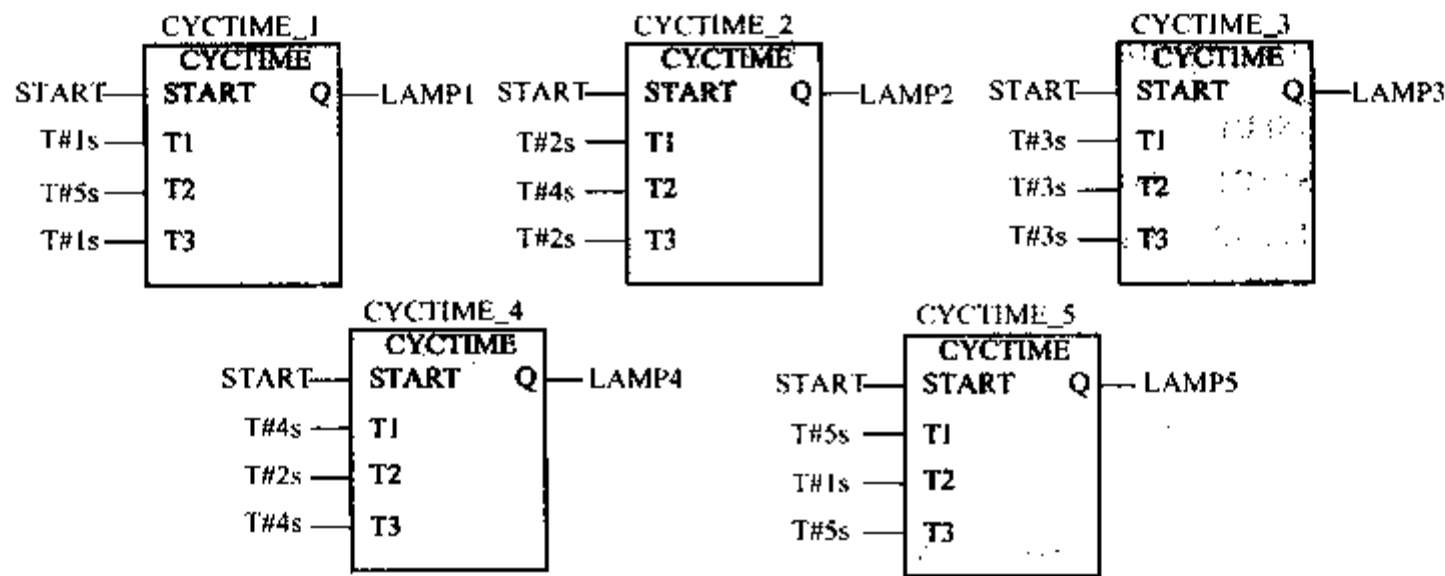


图 4-33 信号灯点亮控制系统的功能块图程序

信号灯 LAMP1 在 START 为 1 后,延时 1 s 后点亮,点亮时间为 5 s,熄灭时间为 1 s。因此,功能块 CYCTIME 的实例 CYCTIME_1 的时间 T1 = T#1 s, T2 = T#5 s, T3 = T#1 s。类似地,其他信号灯的各时间见表 4-6。时间循环系统中,各功能块实例的 T2 + T3 应恒定。

采用 CYCTIME 功能块时,由于 T#0s 同样需要一定的扫描时间,因此,可以保证不同 CYCTIME 实例的同步。

表 4-6 各信号灯的时间设置

信 号 灯	LAMP1	LAMP2	LAMP3	LAMP4	LAMP5
T1	T#1 s	T#2 s	T#3 s	T#4 s	T#5 s
T2	T#5 s	T#4 s	T#3 s	T#2 s	T#1 s
T3	T#1 s	T#2 s	T#3 s	T#4 s	T#5 s

CYCTIME 功能块可用于各种时间循环的程序中,只需要设置有关时间和启动信号。

2. 时钟显示

为显示当前时间,要将实时时钟的输出分解为时、分、秒显示。为此,建立新的功能块 RTCC。

(1) 变量声明

RTCC 功能块仅有 3 个输出变量,Hour、Minute 和 Second。变量声明如下:

```
VAR_OUTPUT
```

```
Hour , Minute , Second ;INT;  
END_VAR
```

须注意,输出变量的数据类型采用整数,不能采用时间数据类型。

(2) 功能块本体程序

程序采用科维公司的系统实时时钟功能块 RTC_S,用 RIGHT 和 LEFT 函数提取时、分、秒的字符串形式数值,并将它们转换为整数,如图 4-34 所示。

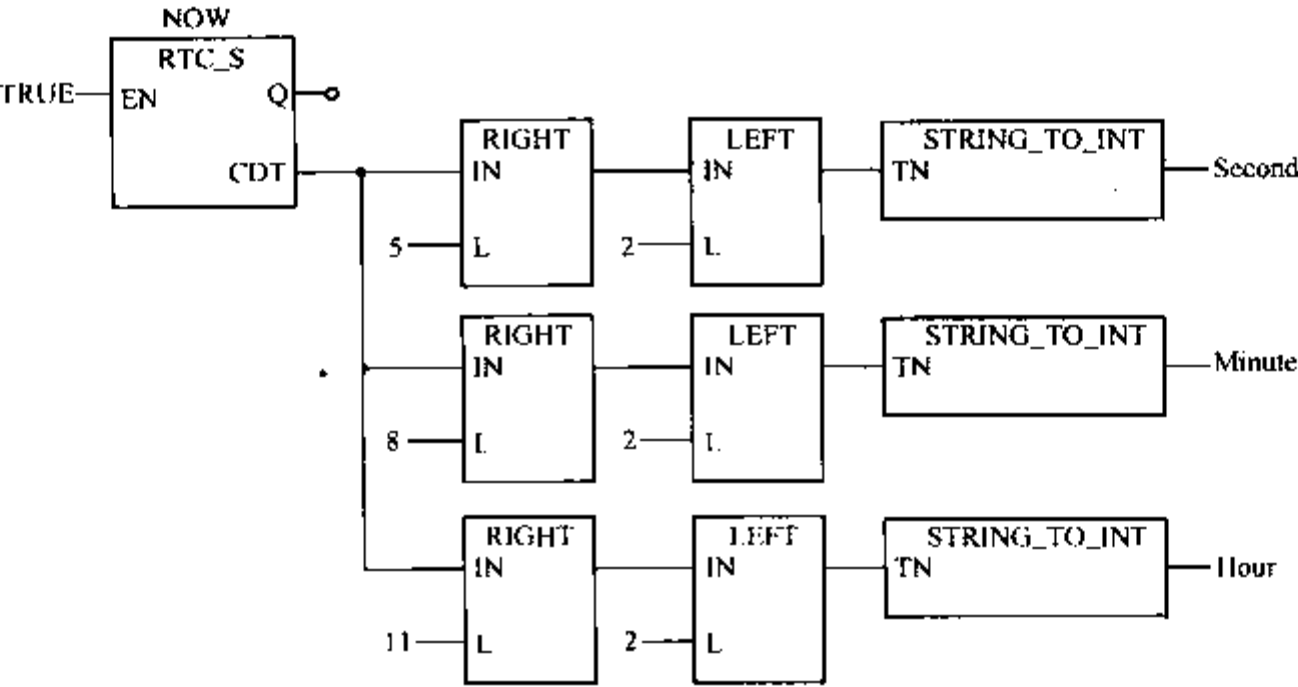


图 4-34 时钟显示功能块本体程序

根据系统实时时钟的字符串数据形式为 DT#YYYY-MM-DD-HH;MM;SS,MS,前面的 MM 表示月份,后面的 MM 表示分钟,功能块本体程序见图 4-33。先用 RIGHT 函数提取整个字符串右侧的 11 个字符,再用 LEFT 函数提取该 11 字符中左侧的 2 个字符,获得时的字符串,再将字符串转换到整数,得到时的数据。同样,对分数据,先提取右侧的 8 个字符串,再提取该字符串左侧的 2 个字符串,最后,转换为分数据。对秒数据,先提取右侧的 5 个字符串,再在该字符串左侧提取 2 个字符串,最后,转换为秒数据。

类似地,也可用 RIGHT、LEFT 和 STRING_TO_INT 的函数将年、月、日分别显示。例如,RIGHT 的 L 为 22,LEFT 的 L 为 4,则 STRING_TO_INT 函数的输出是年份;RIGHT 的 L 为 17,LEFT 的 L 为 2,则 STRING_TO_INT 函数的输出是月份;RIGHT 的 L 为 14,LEFT 的 L 为 2,则 STRING_TO_INT 函数的输出是日期。

编写程序时应注意,一些软件系统不允许字符串进行连接,这时,可将字符串连接到对应的字符串变量,例如,CDT 输出连接到 S001,则各 RIGHT 函数的 IN 输入连接到 S001,余类推。

(3) 时钟显示功能块的调用

为调用上述的衍生功能块,用功能块图编程语言编写的程序如图 4-35 所示。变量声明如下:

```
VAR  
    HH ;INT;  
    MM ;INT;
```

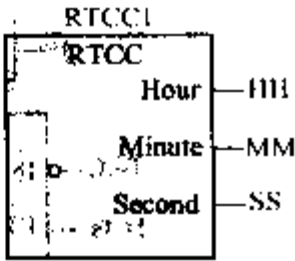


图 4-35 调用功能块

```

SS ;INT;
RTCC1 ;RTCC ;
END_VAR

```

3. 定时交替的电动机正、反转切换控制

一些应用场合,需要电动机正转一定时间后自动切换到反转,反转运转一定时间后自动切换到正转。采用定时交替正、反转切换方法来控制电动机的运转。

另一种应用情况是电动机正转一定时间后自动停止,停止一定时间后电动机自动反转,反转一定时间后自动停止,然后停止一定时间后电动机自动正转,如此循环运转和停止。

(1) 没有停止过程的电动机交替正、反转控制

图 4-36 是定时交替的电动机正、反转控制程序。

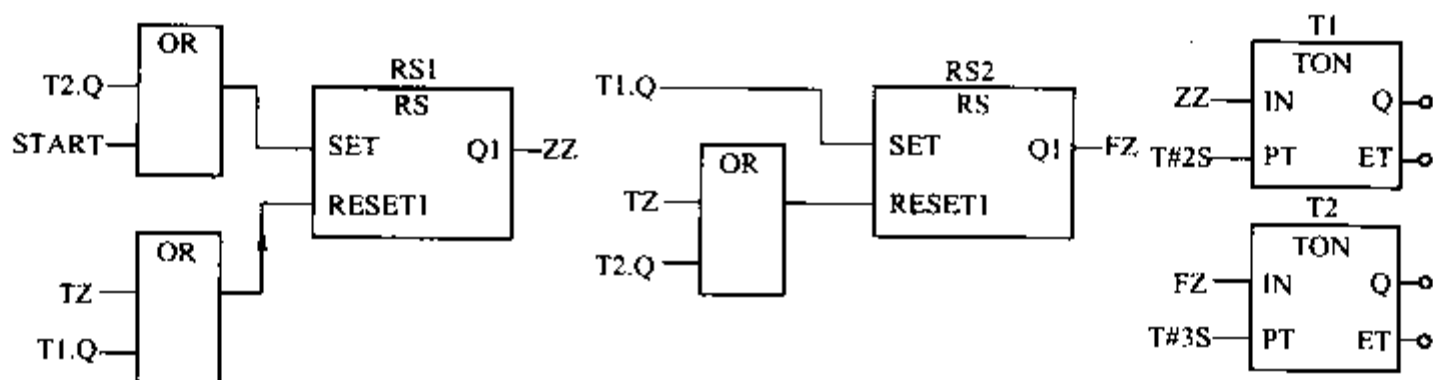


图 4-36 定时交替的电动机正、反转控制程序

该程序用一个按钮 START 作为起动信号,用按钮 TZ 手动停止电动机的运转。用两个定时器对正、反转的运行时间定时,定时时间可在 T1 和 T2 定时器功能块中设置。例如,图中,正转时间设定为 2 s,反转时间设定为 3 s。

程序中,采用两个 RS 功能块,用两个定时器组成循环过程。由于定时器的循环过程中,只有在 T2 定时器计时到时才重新更新,因此,对正转 ZZ 的双稳元素功能块可采用置位优先的 SR 功能块。此外,采用启动信号可防止系统一旦上电,就开始运转。同样,停止信号也采用开关获得,这类程序可用于类似洗衣机的控制程序中。

(2) 有停止过程电动机交替正反转控制

图 4-37 是有停止过程电动机正反转控制程序。

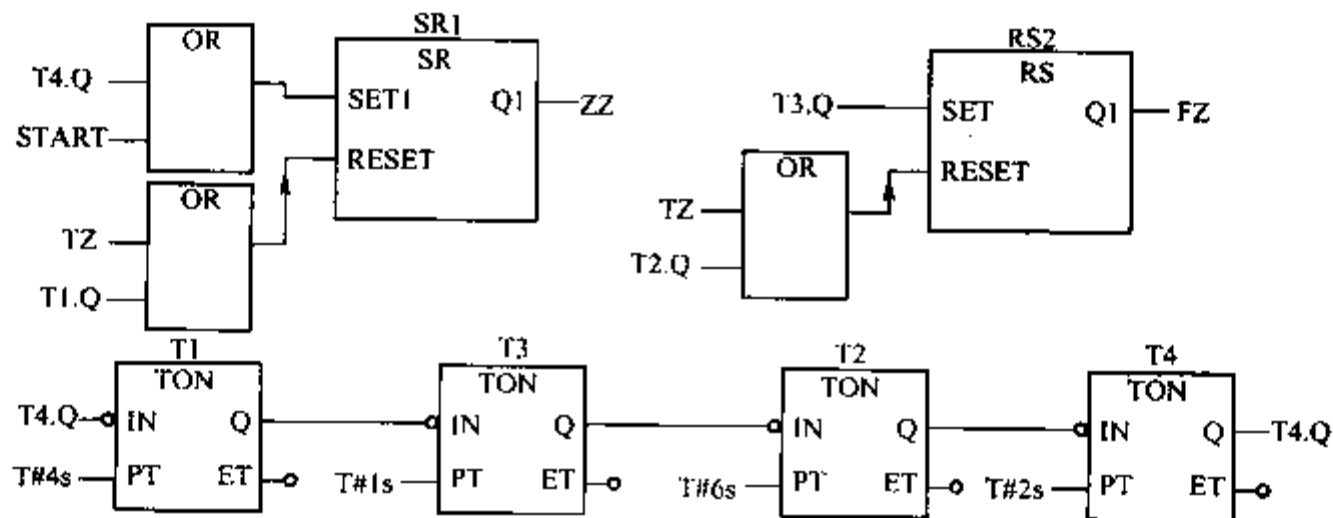


图 4-37 有停止过程定时交替的电动机正、反转控制程序

程序需要有停止按钮 TZ 和起动按钮 START,用 4 个定时器分别表示电动机切换时的延时时间和运行时间。

这类程序要求有电动机的停止阶段,因此,需要设置停止按钮。并且起动后电动机开始是正转,然后是停止和反转。

4. 电动机的顺序控制

一些应用,例如皮带输送系统中,要求电动机顺序控制,即在开车时,一系列电动机应逆序起动,停车时,一系列电动机要顺序停止。例如,在皮带输送机系统中,一系列皮带输送机的电动机的开车应先开后级电动机,再开前级电动机。当系统停车时,应先停前级电动机,后停后级电动机。这称为逆序起动,顺序停止。这类控制系统的控制程序如图 4-38 所示,示例中有 3 个电动机。

当控制的电动机数大于 3 台时,第一台电动机的程序不变,最终电动机的程序与图中第 3 台电动机的程序类似,仅将 STOP3 改接最终一台的停止按钮。中间的电动机与图中第 2 台电动机程序类似。仅起动条件的与函数连接该台电动机的起动按钮和其下一台电动机的运转信号,停止条件的与函数连接该电动机的停止按钮和其上一台电动机的运转信号(取反)。

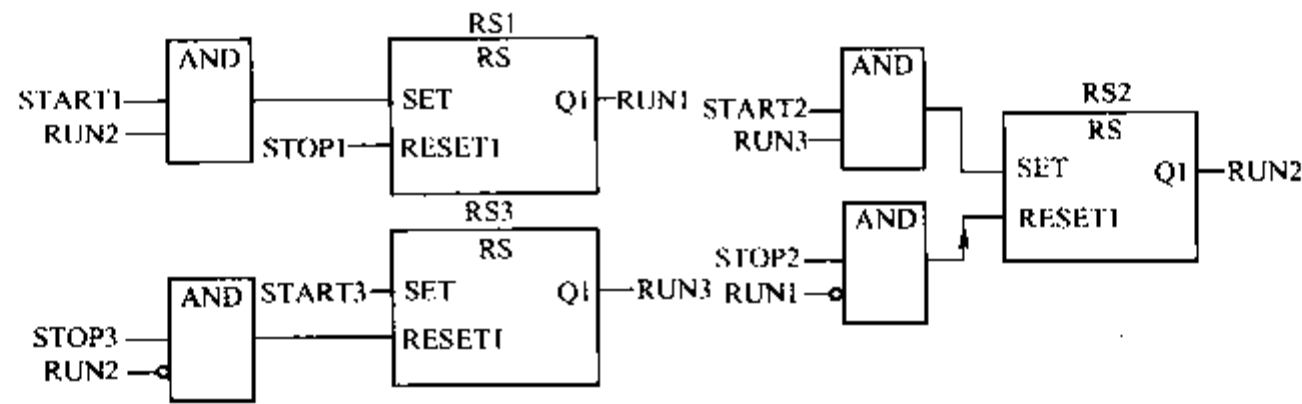


图 4-38 电动机的顺序控制

实际应用时,还需要考虑各电动机的延时时间,以便于皮带输送机上的物料能够不堆积或溢出,造成事故。

5. 周期信号发生器

(1) 脉冲信号发生器

一些应用场合需要周期的脉冲信号,用定时器可实现上述控制要求。例如,三相步进电动机转速控制时所需脉冲信号就可用脉冲信号发生器产生的脉冲信号。

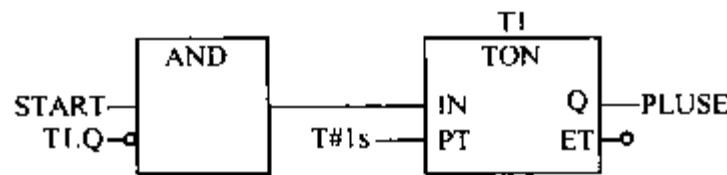


图 4-39 周期脉冲信号发生器程序

图 4-39 所示程序能产生一个周期为 1 s 的脉冲信号。改变定时器的设定时间可改变脉冲周期。

须注意,脉冲定时器功能块 TP 提供的脉冲宽度是由该功能块的 PT 参数给出的。本程序中的脉冲宽度由整个程序的扫描周期确定。

(2) 方波信号发生器

方波信号发生器要求输出信号周期接通和断开。通常,将接通时间和总周期之比称为占空比。在时间比例控制系统中将控制器输出以对应的占空比来接通和断开输出信号,实现连续量信号输入和开关量输出的控制规律。

1) 一个延时接通定时器组成方波信号发生器。图 4-40 显示了用一个延时接通定时器实现平衡方波信号发生器的程序。信号的周期时间是定时器设定时间的 2 倍。平衡方波是指信号对称,即占空比为 1: 2。通常这种信号发生器称为平衡振荡信号发生器。

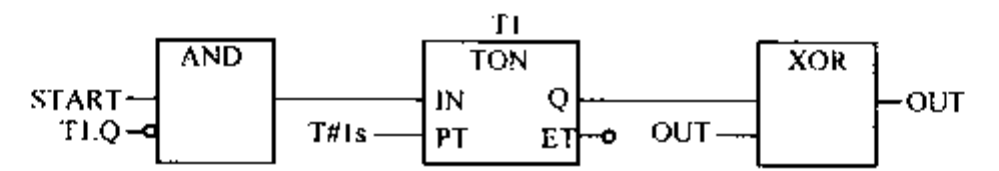


图 4-40 一个延时接通定时器组成方波信号发生器程序

2) 两个延时接通定时器组成方波信号发生器。图 4-41 显示了用两个延时接通定时器实现平衡方波信号发生器的程序。信号的周期是 T1 和 T2 的设定时间之和。当输出 OUT 连接到定时器输出取反信号(如图示),占空比为 T1 设定时间与总周期之比。START 闭合时,输出 OUT 先闭合后断开。当输出 OUT 连接到定时器输出信号,占空比为 T2 设定时间与总周期之比。START 闭合时,输出先断开后闭合。通常,这种信号发生器称为不平衡振荡信号发生器。

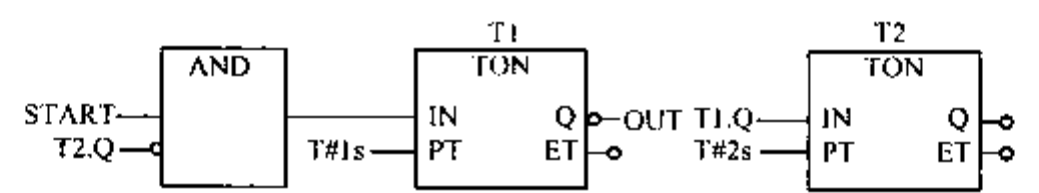


图 4-41 两个延时接通定时器组成方波信号发生器程序

3) 一个延时接通定时器和一个脉冲信号定时器组成方波信号发生器。图 4-42 显示了一个延时接通定时器和一个脉冲信号定时器组成方波信号发生器的程序。信号周期由 T1 定时器设定时间给出,占空比为 T2 设定时间与信号周期之比,因此,编制程序时应注意,T2 设定时间应小于 T1 设定时间,否则,系统输出为常闭。

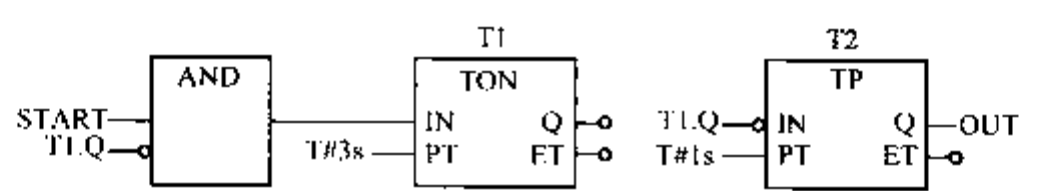


图 4-42 延时接通定时器和脉冲信号定时器组成方波信号发生器程序

4) 一个延时接通定时器和一个计数器组成方波信号发生器。图 4-43 显示了一个延时接通定时器和一个计数器组成的方波信号发生器程序。

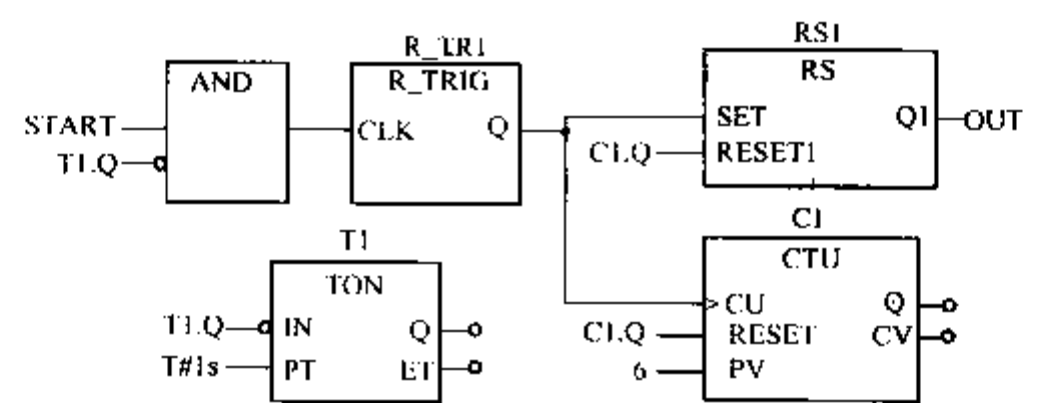


图 4-43 延时接通定时器和计数器组成方波信号发生器程序

其中,计数器的设定值与定时器设定时间之积是输出 OUT 的闭合时间,占空比为计数器设定值与设定值加 1 之比。

6. 一阶滤波环节功能块 LAG1

(1) 一阶滤波环节的数学模型

工业生产过程中被控对象测量值需要滤波处理,工业对象也常用多个一阶滤波环节串联描述。常用的一阶滤波环节数学模型描述为

$$XOUT(s) = \frac{1}{TIs + 1} XIN(s) \quad (4-1)$$

一阶滤波环节的离散数学模型。一阶滤波环节传递函数中,用差分近似微分,得离散化算式

$$XOUT(k+1) = M * XIN(k) + (1 - M) XOUT(k) \quad (4-2)$$

式中,平滑系数由 $M = \frac{TS}{TS + TI}$ 确定; k 表示第 k 次采样时刻的值。

根据式(4-2),采用反馈变量 XOUT 获得下一次求值的输入。

平滑系数的计算需要先将时间类型的数据 TS 和 TI 转换为实数数据类型,并进行相除运算。式(4-2)的计算需要用乘、加和减函数实现。

(2) 一阶滤波环节功能块 LAG1

LAG1 功能块用于输入信号的一阶滤波处理。在控制系统仿真时,该功能块也可作为被控对象的数学模型,它也可作为扰动通道对象的数学模型。该功能块实例可串联连接,组成多个一阶滤波环节的串联系统。

1) 变量声明。变量声明如下。

```
VAR_INPUT
    XIN : REAL;      (* XIN 是输入信号 *)
    TI : TIME;       (* TI 是一阶滤波环节的时间常数 *)
    TS : TIME;       (* 采样时间间隔 *)
END_VAR
VAR_OUTPUT
    XOUT : REAL;     (* 经一阶滤波后的输出 *)
END_VAR
VAR
    M : REAL;        (* 一阶滤波环节的平滑系数 *)
END_VAR
```

2) 功能块本体程序。根据式(4-2),图 4-44 显示用功能块编程语言编写的功能块本体程序。

一些软件系统如果没有直接将时间数据类型的数据转换为实数的函数时,可建立用户的衍生函数,例如,将时间先转换为双整数,再将其转换为实数等,也可先将时间数据类型相加及相除,再进行数据转换。

(3) 一阶滤波环节功能块的应用

一阶滤波环节也称为一阶惯性环节。它常被串联组成多阶环节,用于模拟被控对象、检测变送环节和执行装置等,也被用于作为信号的滤波环节。

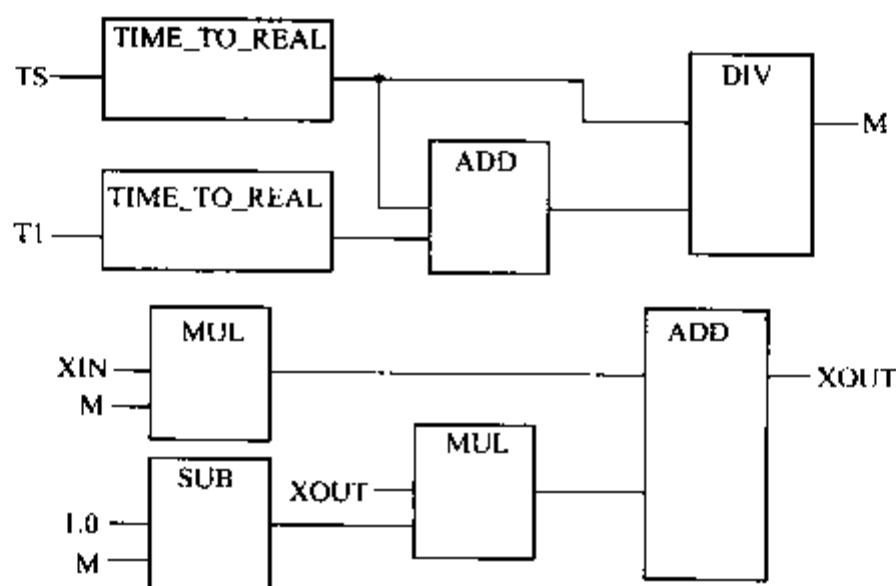


图 4-44 用功能块编程语言编写的程序

图 4-45 是获得三阶环节单位阶跃响应曲线的程序。先用 MULTIPROG 系统软件建立三阶环节,用 SEL 函数实现单位阶跃信号的输入,当 START 为 1 时,SEL 函数输出一个单位阶跃信号,它被送到 LAG1_3、LAG1_2 和 LAG1_1 实例,其输出为 OUT1。

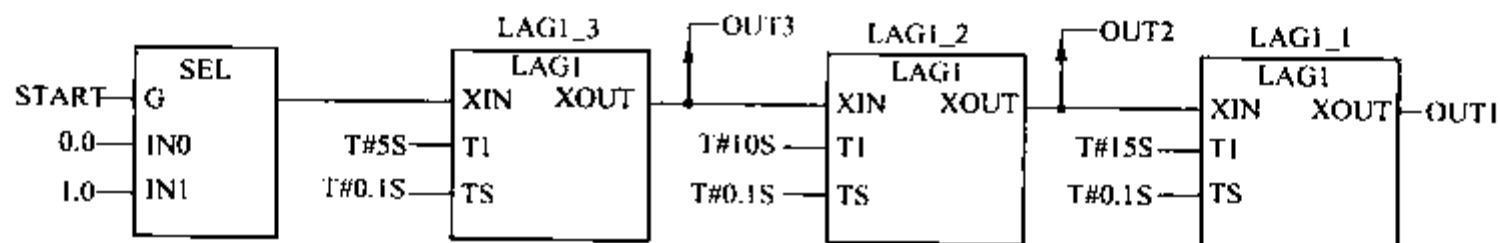


图 4-45 确定三阶环节单位阶跃响应曲线的程序

用 MULTIPROG 软件提供的逻辑分析器,可方便地获得各环节的输出响应曲线。图 4-46 是上述系统在单位阶跃输入下的响应曲线。

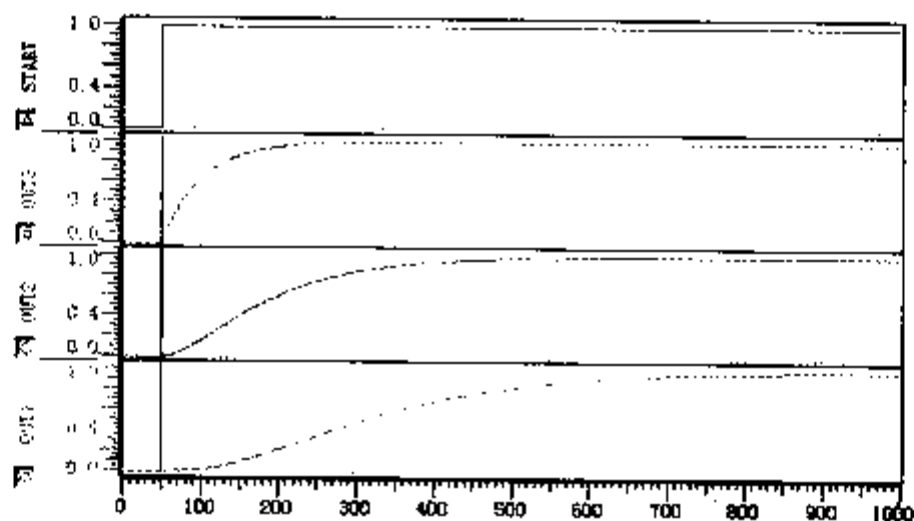


图 4-46 三阶系统在单位阶跃输入下的响应曲线

图中,横坐标是采样次数,各环节的采样周期都是 0.1 s,将采样次数乘以采样周期可转换为时间。例如,阶跃响应输入时的采样次数是 50,表示记录开始时间是第 5 s。

为提高仿真精度,应减小采样周期 TS,通常,采样周期约为最小时间常数的十分之一左右。示例中,为提高仿真精度,采用的采样周期是 0.1 s,因此,从 OUT3 响应曲线的 0.632 处

测得时间为 5 s,它等于该环节的时间常数。

由于该软件系统采用采样次数作为横坐标,因此,设置各环节的采样周期应相同,例如,示例都采用 0.1 s。当用于控制系统仿真时,应根据上述原则选用合适的采样周期。

7. SP 脉冲定时器

(1) 控制要求

某公司的脉冲定时器功能块具有图 4-47a 所示的信号波形。为实现这种脉冲定时器的功能,需要编写用户的功能块。原功能块输入信号有启动信号 IN、复位信号 RST 和脉冲时间 PT,脉冲输出信号是 Q。为便于了解脉冲实际时间,用户功能块 SP 可增加一个输出 ET。因与标准脉冲功能块 TP 的波形不同,需编写用户功能块 SP,如图 4-47b 所示。

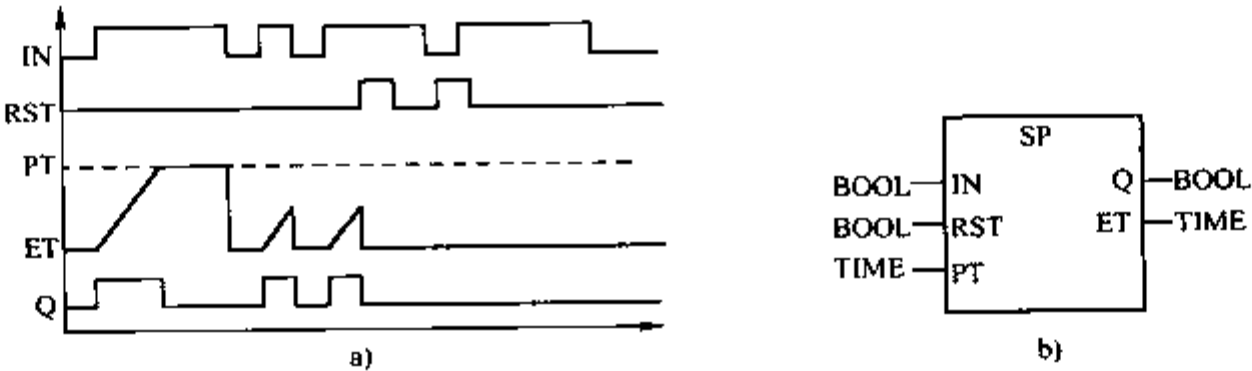


图 4-47 SP 脉冲定时器功能块
a) SP 脉冲定时器波形图 b) SP 脉冲定时器功能块图形表示

(2) 系统分析

从图 4-47a 可见,原脉冲定时器输出脉冲信号宽度由 3 个因素确定。如果输入信号 IN 的持续时间足够长,则脉冲信号宽度等于输入设定时间 PT。如果输入信号 IN 的持续时间不足 PT,则脉冲信号宽度等于输入信号的持续时间。如果输入信号 IN 的持续时间中有复位 RST,则脉冲信号在复位信号上升沿结束。

(3) 编程

功能块图编程语言编写的功能块 SP 程序见图 4-48。

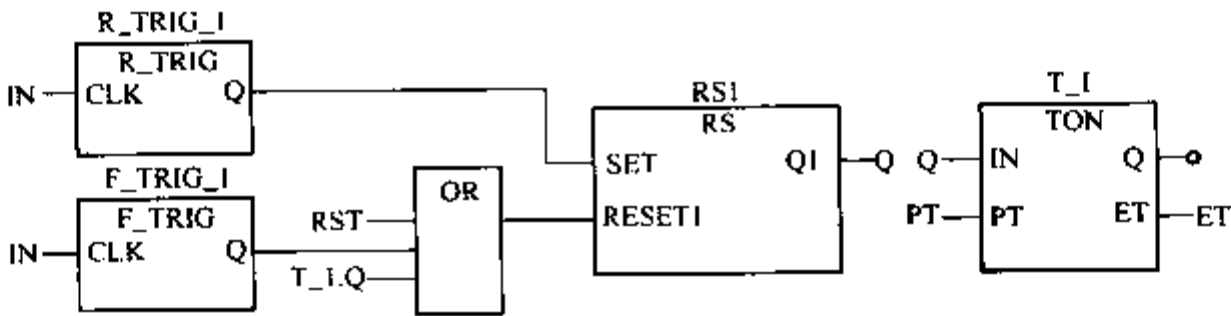


图 4-48 SP 功能块的功能块图程序

程序由 RS 的实例 RS1、R_TRIG 的实例 R_TRIG_1、F_TRIG 的实例 F_TRIG_1、TON 的实例 T_1 及 OR 函数组成。程序中,脉冲信号的触发由 IN 和 R_TRIG_1 实现,当 IN 持续时间足够长,则由 T_1 定时器复位。如果在 IN 持续时间中有复位信号输入,则由 RST 对脉冲输出 Q 信号复位。如果 IN 持续时间不足,则由其下降沿边沿检测功能块触发,对脉冲输出 Q 信号复位。

第5章 顺序功能表图编程语言

5.1 顺序功能表图的三要素

IEC 61131-3 标准中,顺序功能表图(Sequence Function Chart,SFC)是作为编程语言的公用元素定义的。它是采用文字叙述和图形符号相结合的方法描述顺序控制系统的过程、功能和特性的一种编程方法。它既可作为文本类编程语言,也可作为图形类编程语言,但通常将它归为图形类编程语言。因此,通常讲 IEC 61131-3 标准有 3 种图形类编程语言。

顺序功能表图最早由法国国家自动化促进会(ADEPA,法文为 Graphe de Commande Etape-Transistion,GRADCET)提出的,它是针对顺序控制系统的控制条件和过程提出的一套表示逻辑控制功能的方法。由于该方法精确严密,简单易学,有利于设计人员和其他专业人员的沟通 and 交流,因此,该方法公布不久,就被许多国家和国际电工委员会所接受,并制订了相应的国家标准和国际标准,如 IEC 60848 标准,GB/T26988.6—1993 标准等。

顺序功能表图编程语言的基本图形符号是步、转换和有向连线。

5.1.1 基本概念

绘制一个控制系统功能表图的基础是要确定该控制系统的边界和顺序功能表图的范围。通常,把一个控制系统分为施控系统和被控系统两个相互依赖的部分。图 5-1 表示了施控系统和被控系统之间的关系。

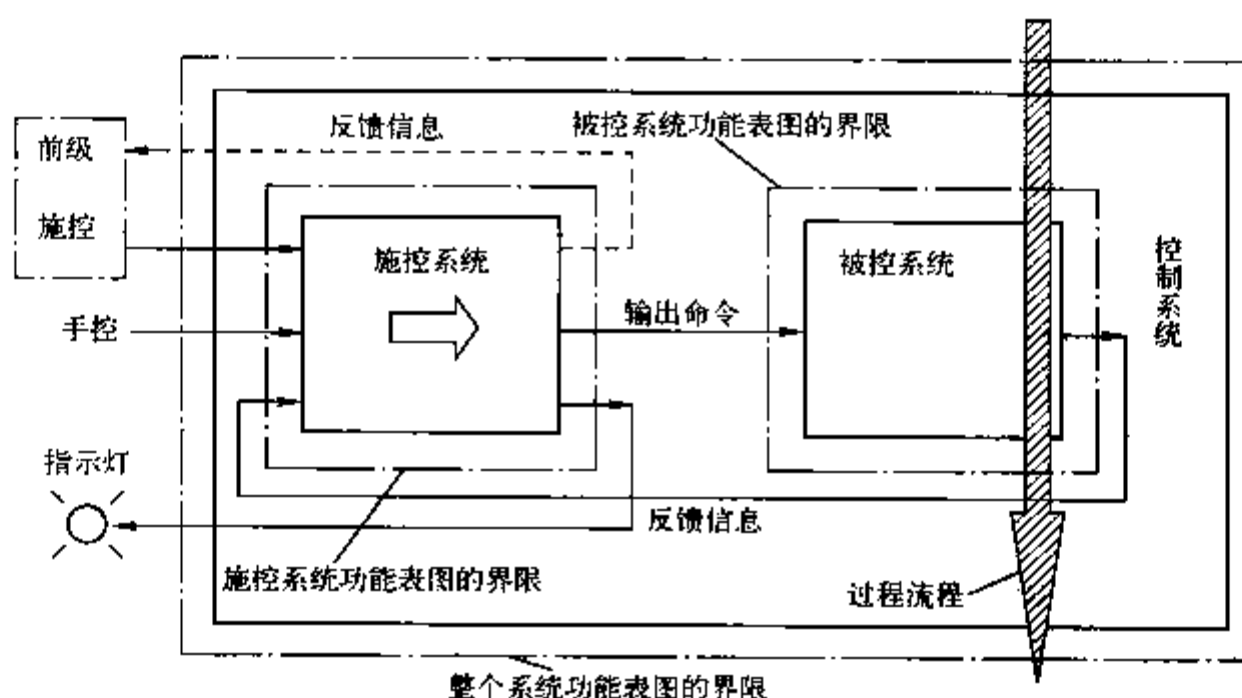


图 5-1 控制系统的划分和功能

施控系统接受来自操作员、过程等的信息,并向被控系统发出操作命令。

被控系统包括执行实际过程的操作设备,它接受来自施控系统的命令,并为施控系统提供反馈信息。图 5-1 中表示了这些系统顺序功能表图的界限。

施控系统的输入是操作员和可能的前级施控系统的命令及被控系统的反馈信息,它的输出包括送到操作员的反馈信息,前级施控系统的输出命令和送到被控系统的命令。

被控系统的输入是施控系统的输出命令和输入过程流程的参数,它的输出包括反馈到施控系统的信息、过程流程中执行的动作,它使该流程具有所需的特性。

施控系统的顺序功能表图描述控制设备的功能,由设计人员根据其对过程的了解来绘制,并作为详细设计控制设备的基础。

被控系统的顺序功能表图描述操作设备的功能,由工程设计人员绘制,作为操作设备工程设计的基础,它也用于绘制施控系统顺序功能表图。

顺序功能表图只提供描述系统功能的原则和方法,不涉及系统所采用的具体技术,因此,用顺序功能表图可以描述控制系统的控制工程、功能和特性,可描述控制系统组成部分的技术特性,而不必考虑具体的执行过程。它适用于绘制电气控制系统的顺序功能表图,也适用于绘制非电气控制系统(如气动、液动和机械的)的顺序功能表图。

【例 5-1】 被控系统和施控系统的示例。

图 5-2 显示了数控机床的被控系统和施控系统。图中,被控系统是机床,施控系统是数控装置。控制系统用于将材料加工成为零件。

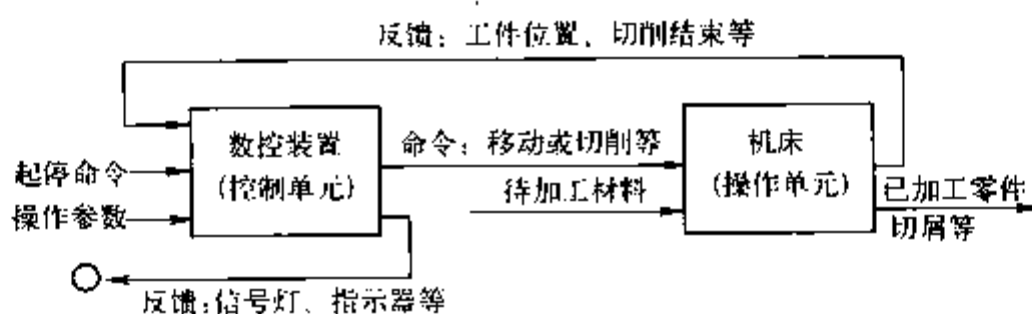


图 5-2 数控机床的被控和施控系统

5.1.2 步

顺序功能表图编程语言把一个过程循环分解成若干个清晰的连续的阶段,称为“步”(Step)。步与步之间由“转换”分隔。当两步之间的转换条件得到满足时,转换得以实现,即上一步的活动结束而下一步的活动开始,因此,不会出现步的重叠。一个步可以是动作的开始、持续或结束。一个过程循环分解的步越多,过程的描述也越精确。

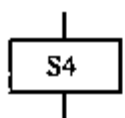
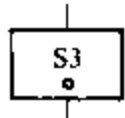
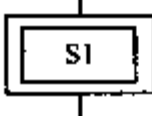
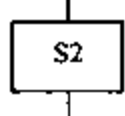
1. 步的表示

过程控制仅接收前一级的过程信息,这些信息产生过程控制的稳定状态。为了描述各种稳定状态,在顺序功能表图中采用“步”的概念。

(1) 步的图形表示

用一个带步名的矩形框表示步,如表 5-1 所示。在逻辑图上,步的图形符号用存储元件图形符号描述。采用矩形框图形符号表示步时,框内应有标识符表示的步名。

表 5-1 步的图形表示

图形符号	说 明	图形符号	说 明
	步的一般符号,矩形框的长宽比任意。步必须有步名,如图中的步 4 用步名 S4 表示		为便于分析,在步的图形符号中添加一个小圆或星号表示该步是活动步,如图中的步 S3 是活动步
	用带步名的双线矩形框表示初始步		在步的图形符号中没有小圆或星号,表示该步是非活动步(仅用于分析时),如图中的步 S2 是非活动步

(2) 步的文本表示

用 STEP...END_STEP 结构表示步。步的结构文本表示如下:

```
STEP XXX;      ( * XXX 是步名  * )
    ( * 步本体 * )
END_STEP
```

XXX 是步名,例如,RUN_1、COOLING 等。步本体是与该步连接的动作控制功能块。

(3) 初始步

控制过程开始阶段的活动步与初始状态相对应,称为“初始步”,它表征施控系统的初始动作。在顺序功能表图中,初始步用带步编号的双线矩形框表示,如表 5-1 的步 S1 所示。每个功能表图至少应有一个初始步。为了说明某步是初始步,在文本结构说明中,用 INITIAL_STEP...END_STEP 结构表示初始步。初始步的结构文本表示如下:

```
INITIAL_STEP XXX:  ( * XXX 是初始步名  * )
    ( * 步本体 * )
END_STEP
```

2. 步状态和步消逝时间

(1) 步状态或步标志

步有两种状态:活动状态和非活动状态。

在控制过程进展的某一给定时刻,一个步可以处于活动状态也可以处于非活动状态。当步处于活动状态时,该步称为活动步。当步处于非活动状态时,该步称为非活动步。

用布尔结构元素 ***.X 的逻辑值表示步名为 *** 步的状态,也称为步标志(Step Flag)。因此,当步名为 *** 的步是活动步时,其布尔值为 1,即 ***.X = BOOL#1,即不能标志为 ***.X := 1。步标志可用布尔变量 ***.X 直接连接到 *** 步的右面,如图 5-3 所示。

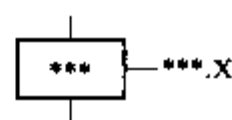


图 5-3 步标志

从网络角度看,活动步是取得令牌(Token)的步,它可以执行相应的命令或动作。非活动步是未取得令牌的步,它不能执行相应的命令或动作。因此,某一步处于活动状态意味着与该步相连接的命令或动作被执行。

为便于分析,当某步是活动步时,可在表示步的图形符号中添加一个小圆或一个星号表示该步处于活动状态。如表 5-1 中的 S3 表示它是活动步。

(2) 步消逝时间

步从开始成为活动步到成为非活动步的时间称为步消逝时间(Elapsed Time),用 ***.T

表示。当步成为非活动步时,步消逝时间的值保持在步失活时所具有的值。当步激活(成为活动步)时,步消逝时间的值被复位到 T#0s,并开始计时。

(3) 应用步时的注意事项

应用步时应注意下列事项:

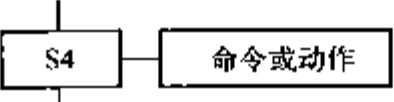
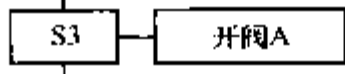
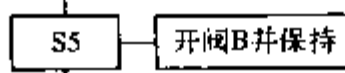
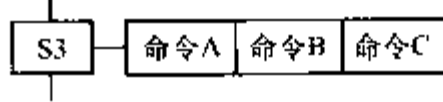
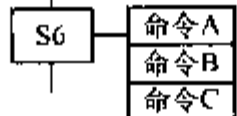
- 1) 程序组织单元的状态由活动步的置位和它的内部和输出变量的值定义。
- 2) 步所出现的程序组织单元中,步名、步标志和步消逝时间是局部变量。它们是写保护的,因此,用户不能直接改变步名、步标志和步消逝时间。
- 3) 程序组织单元的初始状态用它的内部变量和输出变量的初始值,及初始步的设置(即初始为活动的步)表示。
- 4) 每个 SFC 网络都有一个初始步。整个网络从初始步开始进行演变。
- 5) 步初始消逝时间的约定值为 T#0s。普通步初始状态(步标志)的约定值为 BOOL#0,初始步初始状态(步标志)的约定值为 BOOL#1。
- 6) 当一个功能块实例程序被声明具有保持(RETAIN)属性时,在程序或功能块中的所有步状态和消逝时间应保持系统的初始化值。
- 7) 实际应用时,一些软件系统并未提供步消逝时间,这时,可在该步对应的动作功能块中设置定时器,其输入 IN 是该步的激活条件或步标志,其输出 ET 是该步的消逝时间。

3. 步与动作的连接

在活动步阶段,与活动步相连接的命令或动作被执行。在顺序功能表图中,命令或动作矩形框内的文字或符号语句表示,该矩形框与相连接的步图形符号连接。

施控系统输出一个或数个命令,被控系统执行一个或数个动作。表 5-2 表示命令或动作与步之间的关系。详细的动作或命令如表 5-2 所示。

表 5-2 命令或动作与步的关系

图形符号	说 明
	与步相对应的命令或动作矩形框内的文字或符号语句表示 矩形框与步的图形符号用短线连接
	非存储型命令 当步 S3 是活动步时,开阀 A 当步 S3 是非活动步时,阀 A 关闭
	存储型命令 当步 S5 是活动步时,开阀 B 当步 S5 是非活动步时,阀 B 保持打开状态。
	多个命令或动作与同一步相连,采用水平布置表示 图中表示 3 个命令 A、B、C 与步 S3 相连
	多个命令或动作与同一步相连,采用垂直布置表示 图中表示 3 个动作 A、B、C 与步 S6 相连

每个步都会与一个或多个动作或命令有联系。一个步如果没有连接动作或命令,则称为空步,它表示该步处于等待状态,即等待后续转换条件成为真。

(1) 命令或动作的声明

命令或动作可用表 5-3 所示的多种形式进行声明。一个命令或动作的声明范围对包含声明的程序组织单元是局部的。通常,将命令和动作通称为动作(Action)。一个动作(Action)可以是一个布尔变量、LD 语言中的一组梯级、SFC 语言中的一个顺序功能表图、FBD 语言中的一组网络、ST 语言中的一组语句或 IL 语言中的一组指令。在表 5-3 中,序号 2L、2F、3S 和 3I 有不同的编程语言,但都表示相同的动作。

表 5-3 动作声明的形式

序 号	特 性	
1	任何用 VAR 或 VAR_OUTPUT 块声明的布尔变量,或它们的等效图形声明是一个动作的声明	
序 号	示 例	特 性
2L		ACTION_4 动作控制功能块,用 LD 语言的图形描述进行声明
2S		在 OPN_VALVE 动作控制功能块中包含 SFC 元素
2F		ACTION_4 动作控制功能块,用 FBD 语言的图形描述进行声明
3S	<pre>ACTION ACTION_4: %QX1.7 := %IX1 & %MX3 & S8.X ; FF28(SI := (C < D)); %MX10 := FF28.Q ; END_ACTION</pre>	ACTION_4 动作控制功能块,用 ST 语言的文本描述进行声明
3I	<pre>ACTION ACTION_4: LD S8.X AND %IX1 AND %MX3 ST %QX1.7 LD C LT D SI FF28 LD FF28.Q ST %MX10 END_ACTION</pre>	ACTION_4 动作控制功能块,用 IL 语言的文本描述进行声明

注:用于这些示例中的步标志 S8.X 用于获得所希望的结果,即当 S8 是非活动时,%QX17 := 0。

(2) 动作与步的连接

步与动作的连接 (Action Association with Step) 可采用文字形式,也可用图形形式,如表 5-2 和表 5-4 所示。

表 5-4 步与动作的连接形式

序 号	示 例	特 性
1		动作控制功能块物理或逻辑地靠近步的图形符号 动作控制功能块的矩形框与步的图形符号用短线连接
2		动作控制功能块物理或逻辑地靠近步的图形符号 动作控制功能块的矩形框与步的图形符号用短线连接
3	STEP S8; ACTION_1 (L, T#10s, DN1); ACTION_2 (P); ACTION_3 (N); END_STEP	步本体的文本表示 示例表示有 3 个动作控制功能块与步 S8 连接
4		动作控制功能块的动作本体域 使用本形式时,对应的动作名不能用于任何其他动作控制功能块 每步可连接的最大动作控制功能块的数量与实际使用的产品有关。不同产品最大可连接动作控制功能块的数量不同

每步可连接的最大动作控制功能块的数量与实际使用的产品有关。不同产品最大可连接动作控制功能块的数量不同。

(3) 动作控制功能块

1) 动作控制功能块。动作控制功能块由限定符、动作名、布尔指示器变量和动作本体组成。在 IL、ST、LD 和 FBD 编程语言中,动作控制功能块的标准图形符号如图 5-4 所示。在梯形图编程语言中的动作控制功能块如图 5-5 所示。功能块图编程语言中的动作控制功能块如图 5-6 所示。图 5-5 和图 5-6 表示相同的动作,即 ACT1 是一个非存储型动作控制功能块,它的反馈变量是 DN1。

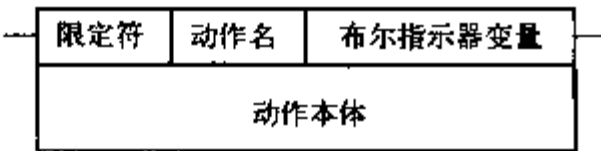


图 5-4 动作控制功能块图形符号

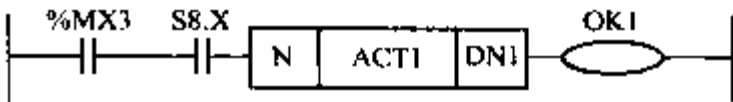


图 5-5 梯形图编程语言中的动作控制功能块

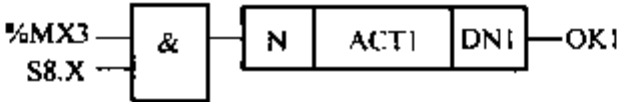


图 5-6 功能块图编程语言中的动作控制功能块

2) 动作控制功能块的限定符。限定符用于限定动作控制功能块的处理方法。表 5-5 显示了可用的动作控制功能块限定符。当限定符是 L、D、SD、DS 和 SL 时,需要有一个 TIME 类

型的持续时间。

表 5-5 动作控制功能块的限定符

序号	限定符	限定功能说明	序号	限定符	限定功能说明
1	无	非存储(空限定符)	7	P	脉冲(Pulse)
2	N	非存储(Non-stored)	8	SD	存储和延迟(Stored and time Delayed)
3	R	复位优先(overriding Reset)	9	DS	延迟和存储(Delayed and Stored)
4	S	置位(存储)(Set Stored)	10	SL	存储和时限(Stored and time Limited)
5	L	时限(time Limited)	11	P1	脉冲(上升沿)(Pulse rising edge)
6	D	延迟(time Delayed)	12	P0	脉冲(下降沿)(Pulse falling edge)

例如,表 5-4 中序号 3 表示步 S8 连接 3 个动作控制功能块,其中,ACTION_1 是时限型动作,延时时间为 10 s,反馈变量是 DN1;ACTION_2 是脉冲型动作;ACTION_3 是非存储型动作。

3) 动作名。动作名是该动作控制功能块的惟一名称。

4) 布尔指示器变量。它用于显示该动作是否完成、时间是否已经用完等反馈信息。因此,也称为反馈变量。在动作控制功能块中,反馈变量是布尔变量,通常用后级转换条件表示,如表 5-4 所示,该变量可不列出。

5) 动作本体。动作本体是动作控制功能块的执行内容,用于控制动作的执行。

5.1.3 转换

顺序功能表图中,步活动状态的进展按有向连线规定的路线进行。进展由一个或多个转换的实现来完成,并与控制过程的发展相对应。

1. 转换

一个转换(Transition)表示从一个或多个前级步沿有向连线变换到后级步所依据的控制条件。转换的图形符号是垂直于有向连线的水平线。通过有向连线,转换与有关步的图形符号相连。转换也称为变迁或过渡。每个转换有一个相对应的转换条件,它是一个单布尔表达式的求值结果。步经过有向连线连接到转换,转换经过有向连线连接到步。

2. 使能转换和实现转换

(1) 使能转换

转换分为使能转换和非使能转换两类。如果通过有向连线连接到转换符号的所有前级步都是活动步,该转换称为“使能转换”,否则该转换称为“非使能转换”。

(2) 实现转换

如果转换是使能转换,同时该转换相对应的转换条件满足,则该转换称为“实现转换”或“触发”(Firing)。因此,实现转换需要两个条件:① 该转换是使能转换;② 相对应的转换条件满足,即转换条件为真。

(3) 实现转换产生的结果

实现转换产生两个结果:

- 1) 与该转换相连的所有前级步成为非活动步,即转换的清除。
 - 2) 与该转换相连的所有后级步成为活动步。转换的实现使过程得以进展。
- 只有当某步的所有前级步都是活动步时,该步才有可能通过转换的实现成为活动步。

3. 转换条件

(1) 转换条件

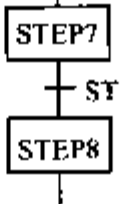
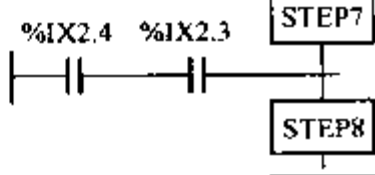
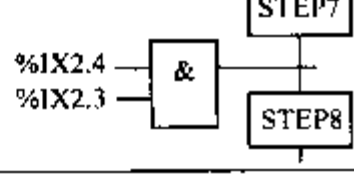
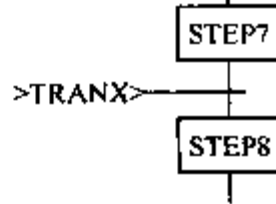
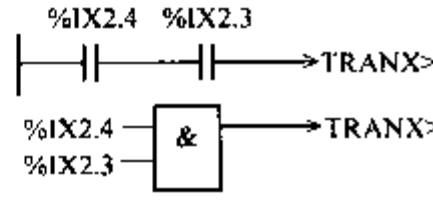
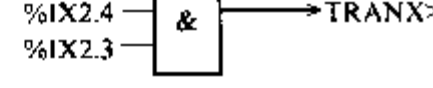
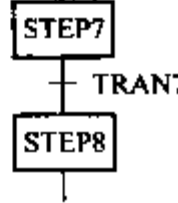
转换条件是指与每个转换相关的逻辑命题,它们可能是真也可能是假。如果用一个相应

的逻辑变量表示转换条件,则当转换条件为真时,该逻辑变量的值为1。反之,如果转换条件为假,则该逻辑变量的值为0。因此,转换条件是一个单布尔表达式求值的结果。

(2) 转换条件的表示

转换条件有直接将转换条件写在转换符号附近、采用连接符、采用文本声明或采用转换名等4种表示方法。表5-6列出了转换和转换条件的基本表示方法。

表 5-6 转换和转换条件的表示

序号	方法	示 例	描 述
1 ^①	转换条件直接写在转换附近		前级步 STEP7 用 ST、FBD 或 LD 语言时,转换条件物理或逻辑地靠近转换 后级步 STEP8
2 ^①			前级步 STEP7 用 LD 语言时,转换条件物理或逻辑地靠近转换 后级步 STEP8
3 ^①			前级步 STEP7 用 FBD 语言时,转换条件物理或逻辑地靠近转换 后级步 STEP8
4	使用连接符		前级步 STEP7 用 FBD 或 LD 编程语言编写的转换条件连接到转换连接符 后级步 STEP8
4 ^①			用 LD 语言表示转换条件(用连接符)
4 ^②			用 FBD 语言表示转换条件(用连接符)
5 ^② 6 ^②	使用文本声明	STEP STEP7 : END_STEP TRANSITION FROM STEP7 TO STEP8; (* ST 或 IL 编程语言编写的转换条件 *) END_TRANSITION SETP STEP8 : END_STEP	用 ST 语言编写转换条件: : = %IX2.4 & %IX2.3; 或 IL 语言编写转换条件: LD %IX2.4 AND %IX2.3
7 ^①	使用转换名		前级步 STEP7 用 ST、IL、FBD 或 LD 编程语言编写的转换条件赋值给转换名(示例为 TRAN78,见例 5-9 ~ 例 5-12) 后级步 STEP8

① 如果支持表 5-4 的序号 1 特性,则也支持本表的序号 1,2,3,4 或 7 的一个或多个特性。

② 如果支持表 5-4 的序号 2 特性,则也支持本表的序号 5 或 6 的一个或两个特性。

1) 转换条件直接写在转换附近。下列 3 种表示方法将转换条件用不同编程语言表示在

转换的图形符号附近。不同编程语言表示的转换条件的图形形式或文字形式不同。

- 用 ST 编程语言表示转换条件。转换条件设置在垂直有向连线的附近,它用合适的布尔表达式描述。

【例 5-2】 ST 编程语言直接将转换条件写在转换符号附近。

图 5-7 采用 ST 编程语言表示转换条件。转换条件直接写在转换的图形符号附近。

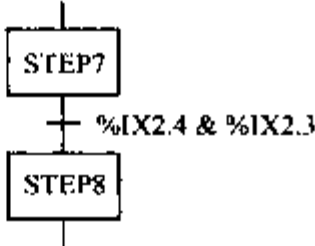


图 5-7 ST 编程语言表示转换条件

- 用 LD 编程语言表示转换条件;转换条件设置在垂直有向连线的附近,它是一个梯形图网络。

【例 5-3】 LD 编程语言直接将转换条件写在转换符号附近。

图 5-8 采用 LD 编程语言表示转换条件。转换条件直接写在转换的图形符号附近。

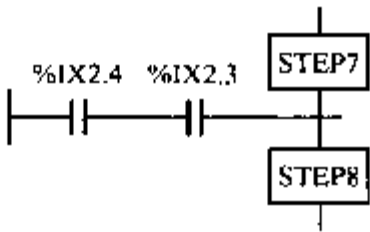


图 5-8 LD 编程语言表示转换条件

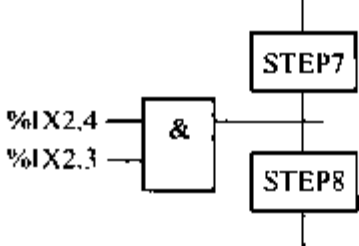


图 5-9 FBD 编程语言表示转换条件

2) 使用连接符。图形类编程语言中,可通过 LD 或 FBD 的网络,将转换条件的输出用一个连接符与垂直有向连线相交叉。表 4-1 规定了连接的符号,连接符号的名称可由用户规定。例如,表 4-1 中的 OTTO 及下面示例的 TRANx 等。

- 用 LD 编程语言表示连接符:用 LD 编程语言编写的转换条件程序,用连接符连接到转换条件的结果。

【例 5-5】 LD 编程语言编写的转换条件用连接符号表示其结果。

图 5-10 采用 LD 编程语言编写转换条件,并将其运算结果用连接符表示。

例 5-5 中,用连接符 TRANx 表示转换 x。

- 用 FBD 编程语言表示连接符:用 FBD 编程语言编写转换条件程序,用连接符连接到转换条件的结果。

【例 5-6】 FBD 编程语言编写的转换条件用连接符号表示其结果。

图 5-11 采用 FBD 编程语言编写转换条件,并将其运算结果用连接符表示。

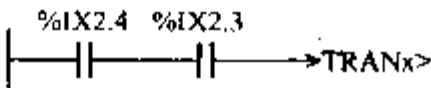


图 5-10 LD 编程语言表示转换符

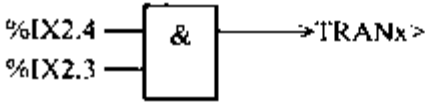


图 5-11 FBD 编程语言表示转换符

3) 文本声明。在 ST 和 IL 编程语言中,可用文本声明表示转换条件。

- 使用 ST 编程语言的文本声明:用 TRANSITION...END_TRANSITION 结构描述转换条件。转换条件应包括下列内容:

关键字 TRANSITION FROM 和其后的前级 (Predecessor) 步 (如果有多于一个的前级步, 用前级步的一个括号表, 例如, (S_1, S_2) 表示两个前级步) 的步名。

关键字 TO 和其后的后级 (successor) 步 (如果有多于一个的后级步, 用后级步的一个括号表) 的步名。

赋值操作符 (:=), 和其后用 ST 编程语言表示的一个布尔表达式描述的转换条件。

终止关键字 END_TRANSITION。

【例 5-7】 ST 编程语言的文本声明。

```
STEP STEP7 : END_STEP           (* 第 7 步的文本声明 *)
  TRANSITION FROM STEP7 TO STEP8: (* 从步 7 到步 8 的转换 *)
    := %IX2.4 & %IX2.3;          (* 用 ST 编程语言编写转换条件 *)
  END_TRANSITION
SETP STEP8 : END_STEP           (* 第 8 步的文本声明 *)
```

- 使用 IL 编程语言的文本声明: 用 TRANSITION...END_TRANSITION 结构描述转换条件。转换条件应包括下列内容:

关键字 TRANSITION FROM 和其后的前级步 (如果有多于一个的前级步, 用前级步的一个括号表) 的步名。

关键字 TO 和其后的后级步 (如果有多于一个的后级步, 用后级步的一个括号表) 的步名, 及冒号 (:)。

从一个单独的行开始, 用 IL 编程语言编制的一个指令表, 它的计算结果, 即在累加器的最终结果确定转换条件。注意, 不需要用 ST 语句来存储转换条件的运算结果。

用一个独立的行表示的终止关键字 END_TRANSITION。

【例 5-8】 IL 编程语言的文本声明。

```
STEP STEP7: END_STEP           (* 第 7 步的文本声明 *)
  TRANSITION FROM STEP7 TO STEP8: (* 从步 7 到步 8 的转换 *)
    LD    %IX2.4                 (* 用 IL 编程语言编写转换条件 *)
    AND   %IX2.3
  END_TRANSITION
SETP STEP8: END_STEP           (* 第 8 步的文本声明 *)
```

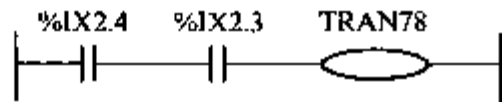
4) 采用转换名。通过用在有向连线右面的一个标识符形式的转换名。转换名标识符被列写在 TRANSITION...转换名 END_TRANSITION 结构中, 它的求值将导致布尔值赋值到由转换名标记的变量。对 4 种编程语言转换名的表示方法相同, 但因编程语言不同, 因此, 转换条件编写也不同。

- 用 LD 编程语言编写转换名。

【例 5-9】 LD 编程语言编写转换名。

图 5-12 采用 LD 编程语言编写转换条件, 并将其运算结果赋值给转换名。

须注意, LD 编程语言编写的转换条件需赋值给转换名 TRN78, 在转换的声明中, 应说明该转换名, 如下所示:



TRANSITION TRN78 FROM STEP7 TO STEP8:

图 5-12 LD 编程语言表示转换名

(* 用 LD 编程语言编写的转换条件,运算结果赋值给转换名 'TRAN78 *)
END_TRANSITION

与例 5-7 和例 5-8 不同,采用转换名的转换声明需要插入转换名,以便将转换条件的运算结果赋值给转换名。

- 用 FBD 编程语言编写转换名。

【例 5-10】 FBD 编程语言编写转换名。

图 5-13 采用 FBD 编程语言编写转换条件,并将其运算结果赋值给转换名。与 LD 编程语言编写转换名的方法相同,在转换的声明中,应说明该转换名。

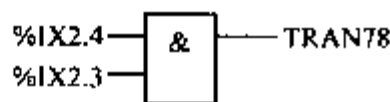


图 5-13 FBD 编程语言
表示转换名

- 用 IL 编程语言编写转换名。

【例 5-11】 IL 编程语言编写转换名。

```
LD    %IX2.4
AND   %IX2.3
ST    TRAN78
```

采用转换名表示转换条件时,需要将 IL 编程语言编写的转换条件运算结果赋值给转换名。因此,与文本声明的方法不同,它应包含 ST 语句。

- 用 ST 编程语言编写转换名。

【例 5-12】 ST 编程语言编写转换名。

```
TRAN78 := %IX2.4 & %IX2.3;
```

采用转换名表示转换条件时,需要将 ST 编程语言编写的转换条件运算结果赋值给转换名,即采用赋值语句实现。

转换名的范围对转换所在的程序组织单元是局部变量。

列写转换条件时应避免所有冲突的发生。例如,两个使能转换以同一步为条件时,应确保与这两个转换相连的转换条件是不相容的,即不会同时为真。

【例 5-13】 转换条件示例。

图 5-14 显示了一个分支结构,转换条件 A 与 B 应不相容,因此,通常,选用 B = NOT A,才能保证只有一个转换条件能够满足。

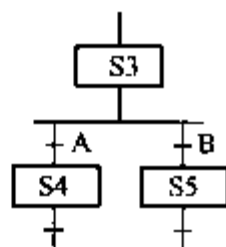


图 5-14 分支中
转换条件的设置

5.1.4 有向连线

在顺序功能表图中,步之间的进展按有向连线规定的路线进行。有向连线也称为弧(Arc)或连接。

1. 有向连线的表示

有向连线图形符号是水平或垂直的直线。为使画面更清晰,有时也可用斜线。通常,连接到步的有向连线用连接到步顶部的垂直线表示。从步引出的有向连线用连接到步底部的垂直线表示。

有向连线用 TRANSITION...END_TRANSITION 结构的文本表示,如例 5-7、例 5-8 和例 5-9 所示。

SFC 元素提供可编程控制器程序组织单元的划分方法,它是一组步、通过有向连线内部连接的转换的划分方法。与步有关的是一组动作,而与转换有关的是一个转换条件。步、转换和

有向连线之间的关系描述为：步经有向连线连接到转换，转换经有向连线连接到步。

2. 有向连线的连接

步的进展方向总是从上到下或从左到右，因此，有向连线的方向也从上到下或从左到右。如果不遵守上述规定，必须对有向连线添加箭头。

步激活状态的进展由一个或几个转换的实现引起，并沿着有向连线的方向进行。转换实现的同时，使有向连线连接到相应转换符号的所有前级步立刻成为非活动（或复位）步，这是转换的清除（Clear），同时导致所有后继步的激活，这是转换的实现。

如果垂直有向连线与水平有向连线之间没有内在联系，允许它们交叉，但当有向连线与同一进展相关时，不允许交叉。

在复杂的图中或在几张图中表示而使有向连线必须中断时，应在中断点处指出下一步的步名称和该步所在的页号或来自上一步的步名称和所在步的页号。

5.2 顺序功能表图的程序结构

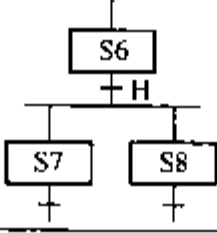
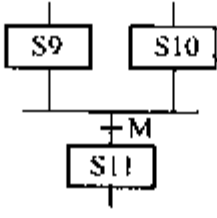
步的进展由下列基本程序结构组成。表 5-7 是各种程序结构的图形表示。

当转换的实现导致同时有几个步激活时，这些步的序列被称为同步序列（Simultaneous Sequence）或并行序列。当它们被同步激活时，这些序列的每个进展都是独立的。为了强调这类序列结构的特殊性，同步序列的分支和合并常用一个水平双线的图形符号表示。

要防止不安全序列或不可达序列结构的发生。在这些序列结果中，可能出现不安全的或不可控的步，或出现不可激活的步。为此，PLC 制造商提供了对有关语法的检查，防止发生或出现这样的序列结构。

表 5-7 程序结构的图形符号

序列名称		图形符号	说 明
单序列			只有当 S3 是活动步（S3.X=1），转换条件为真（B=1）时，才发生 S3 到 S4 的进展。转换实现后 S3 成为非活动步，S4 成为活动步。当 S4 是活动步（S4.X=1），转换条件为真（C=1）时，才发生 S4 到 S5 的进展。转换实现后 S4 成为非活动步，S5 成为活动步。这种程序结构是单序列程序结构
选择序列	开始：分支		如果 S6 是活动步（S6.X=1），则当转换条件 D 为真（D=1）时，发生 S6 到 S7 的进展，转换实现后 S6 成为非活动步，S7 成为活动步。如果 S6 是活动步（S6.X=1），转换条件 E 为真（E=1）时，发生 S6 到 S8 的进展，转换实现后 S6 成为非活动步，S8 成为活动步。应注意，选择序列中，转换条件 D 和 E 不能同时为真 选择序列的分支结构中，根据满足转换条件的先后，确定步的进展方向
	结束：合并		如果 S9 为活动步（S9.X=1），并且转换条件 F 为真（F=1），则发生 S9 到 S11 的进展，转换实现后 S9 成为非活动步，S11 成为活动步。如果 S10 为活动步（S10.X=1），并且转换条件 G 为真（G=1），则发生 S10 到 S11 的进展，转换实现后 S10 成为非活动步，S11 成为活动步 选择序列的合并结构中，S11 的前级转换条件同时满足，才能使步发生进展。为防止不安全或不可达，应采用并行序列的合并结构

序列名称		图形符号	说明
并行序列	开始：分支		如果 S6 是活动步 (S6. X = 1), 则当转换条件 H 为真 (H = 1) 时, 同步发生 S6 到 S7 的进展和 S6 到 S8 的进展, 转换实现后 S6 成为非活动步, S7 和 S8 都成为活动步 并行序列也称为同步序列, 并行序列分支结构中, 如果转换条件满足, 可使多个后续步成为活动步
	结束：合并		水平双线上面的 S9 和 S10 都为活动步, 并且转换条件 M 为真 (M = 1), 则发生 S9 和 S10 到 S11 的进展, 转换后 S9 和 S10 都成为非活动步, S11 成为活动步 并行序列合并结构中, 所有前级步必须都是活动步, 并满足转换条件, 才能发生步的进展

5.2.1 单序列结构

单序列结构由一系列相继激活的步组成。在此结构中, 每个步后面仅连接一个转换, 而每个转换由一个步使能。

5.2.2 选择序列结构

选择序列结构分为选择序列的开始和选择序列的结束两类。

1. 选择序列的开始：分支

在几个序列中进行选择时, 转换条件的数量与序列的数量相同 (即每个序列都有一个转换条件), 且在执行转换时, 只能有一个转换条件成立。转换的图形符号只允许绘制在选择序列开始的水平 (双) 线下面。须注意, 如果只选择一个序列, 则在同一时刻与若干个序列相关的转换条件中只能有一个为真, 应用时需要防止所有的冲突发生。对序列进行选择的首选次序可在注明转换条件时规定。

2. 选择序列的结束：合并

几个序列合并到一个公用序列时, 用与需要重新组合的序列相同数量的转换表示。转换的图形符号只允许绘制在选择序列结束的水平 (双) 线上面。

5.2.3 并行序列结构

并行序列结构分为并行序列的开始和并行序列的结束两类。

1. 并行序列的开始：分支

当转换的实现导致几个序列同时激活时, 这些序列称为并行序列。它们同时被激活后, 每个序列活动步的进展是独立的。只允许在并行序列开始的水平 (双) 线上面绘制一个转换的图形符号。

2. 并行序列的结束：合并

为了使几个序列同时同步停止, 采用并行序列合并的结构。只允许在并行序列结束的水平 (双) 线下面绘制一个转换的图形符号。并行序列的活动或非活动可以分成一段或几段实现。并行序列中转换的完整符号包括转换符号和水平 (双) 线。

选择序列与并行序列结构的区别是转换条件是用选择的还是公用的, 在功能表图中, 应看

转换符号是在水平双线的上面还是下面。

除了上述序列结构外,还有序列跨接(Sequence Skip)、序列回路(Sequence Loop)等,它们是选择序列的特例。

5.2.4 不安全序列和不可达序列结构

在编程时应注意防止出现不安全序列(Unsafe SFC)和不可达序列(Unreachable SFC)结构。在不安全序列结构中,会在同步序列外出现不可控制和不能协调的步的激活。在不可达序列结构中,可能包含始终不能激活的步。

5.3 顺序功能表图编程语言

顺序功能表图编程语言由步、转换和有向连线组成。它们总是保持步/转换和转换/步的交替更叠,即有下列规则:

- 1) 两个步不能直接相连,它们通常由一个转换来分隔。
- 2) 两个转换不能直接连接,它们通常由一个步来分隔。
- 3) 步经有向连线连接到转换,转换经有向连线连接到步。

5.3.1 顺序功能表图的进展

1. 动作和动作控制块

顺序功能表图编程语言中,当步成为活动步时,与其连接的动作或命令被执行。这些动作或命令用动作控制功能块描述。

一个动作控制功能块用于描述一组动作或命令。如上述,动作控制功能块的限定符用于限定动作的执行功能。例如,S 限定符说明执行的动作将被存储。因此,当连接的步成为非活动步时,该动作仍将继续被执行,直到被限定符 R 限定为复位的动作被执行。D 限定符说明被执行动作在连接的步成为活动步后延时一定持续时间 T 后才开始执行。

动作是 SFC 网络中一个步的命令和动作,或 LD、FBD 等网络中在一定条件下执行的指令序列。如图 5-4 所示,动作控制功能块不仅包括一个动作名还包括动作执行条件和布尔指示器变量。

动作名用于定义所执行的动作。它在程序组织单元中是局部变量。动作名与一个布尔变量或一组语句、指令有关。动作名用于声明动作或该动作执行的布尔变量。

动作执行条件由限定符规定。限定符的限定条件用于声明动作执行的开始时间、动作是否被存储、动作执行时间等。

一个动作的执行时间可以由一个动作控制功能块定义,例如,N、P 等限定符定义的动作控制功能块,也可由多个动作控制功能块定义,例如,S、SD、DS、SL 等限定符定义的动作控制功能块所连接的动作,需要用 R 限定符定义的动作控制功能块来复位。

布尔指示器变量用于反映动作的状态,因此,称为反馈变量。当使用布尔指示器变量时,应在变量声明区段进行声明。布尔指示器变量的值为布尔值 TRUE,表示该动作正被执行,即处于活动状态。它是在相关联的动作描述外为用户提供的步状态信息。一般编程系统尚未提供该反馈变量。

动作或动作控制功能块的输入变量是步标志变量。根据功能块是否带最终扫描,动作控制功

能块的输出有输出 Q 和输出 A。动作的执行根据步标志变量和动作控制功能块输出变量 Q 的值确定。当变量的值为 1 时,动作或指令等被执行,当变量的值为 0 时,动作或指令等不执行。

【例 5-14】 动作控制功能块的示例。

图 5-15 所示的动作控制功能块与步 S8 连接。限定符为 N,动作名为 ACTION_8。动作本体由三组指令组成,分别是对%IX1 与%MX3 进行与运算,并将结果赋值给%QX17;调用功能块 FF28 实例;将调用 FF28 求值的结果赋值给%MX10。示例说明,当 TRAN89 的转换条件满足时,上述三组指令将不再执行(采用 N 限定符,因此,动作不被存储)。

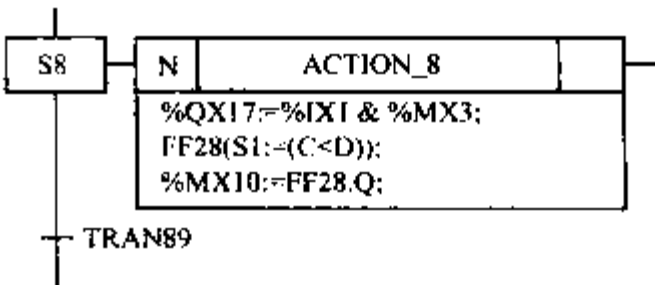


图 5-15 动作控制功能块示例

2. 动作控制功能块的应用规则

在功能上,动作控制功能块等效于下列规则的应用:

(1) 动作控制功能块是功能块

例如,它可表示为图 5-16 所示的功能块。分为带最终扫描逻辑(With Final Scan Logic)和不带最终扫描逻辑(Without Final Scan Logic)两类。其外部接口的数据类型,除了 T 是 TIME 数据类型外,其余都是 BOOL 数据类型。输出 Q 称为动作标志(Action Flag),其值用功能块输出的表示方法表示,例如,图 5-15 中可表示为 ACTION_8.Q。输出 A 称为动作(Activation)。

与动作连接的是动作控制功能块的实例。例如,图 5-15 中的 ACTION_8。它的输入是步标志变量 S8.X。当选用限定符为 N 时,表示将步标志变量连接到 N 变量。动作控制功能块的输出与动作本体连接。因此,当图 5-15 中的步标志变量为 1 时,表示步 S8 是活动步,与步 S8 连接的动作控制功能块实例 ACTION_8 的输出 Q 被置 1,这时,动作本体内的三组指令被执行。

当动作被定义为一个布尔变量时,动作控制功能块的输出 Q 是该布尔变量的状态。当动作被定义为一组语句、指令或网络时,则当动作控制功能块的输出 A 保持为 1 时,这些动作将被继续执行。这时,输出 Q 的状态可以通过在动作执行期间对只读布尔变量的读取来存取。例如,用 ACTION_8.Q 读取。

输出 Q 为 0(False),表示动作的执行过程已经到达最终时间,即该动作将不再被执行。因此,Q 被用于确定该动作执行时间是否已经结束。

在由限定符 P0 和 P1 调用的动作执行期间,动作控制功能块的输出 Q 应在 0。这时,才能够根据步的上升沿(P0)或下降沿(P1)确定输出 A 的脉冲信号。其他限定符的场合,输出 A 将根据输出 Q 或输出 Q 的下降沿信号确定其布尔值。当 Q 为 1 或在 Q 的下降沿时,输出 A 为 1。

对不同编程系统的产品,动作控制功能块的输出 A 和 Q 是与具体实现有关的。

(2) 与步或与功能块的结合

用限定符 N、R、S、L、D、P、P0、P1、SD、DS、SL 对动作控制功能块进行限时,动作控制功能块的输入连接到有关的步标志变量,称为与步结合(Association with a step)或与动作功能块结合(Association with an action block)。当与步结合的步是活动步或者与功能块结合的输入有值为 1 时,称该结合是活动的。

(3) 动作控制功能块的本体

动作控制功能块的本体可用图 5-17a 和图 5-17b 表示,其对应的外部接口见图 5-16a 和图 5-16b。

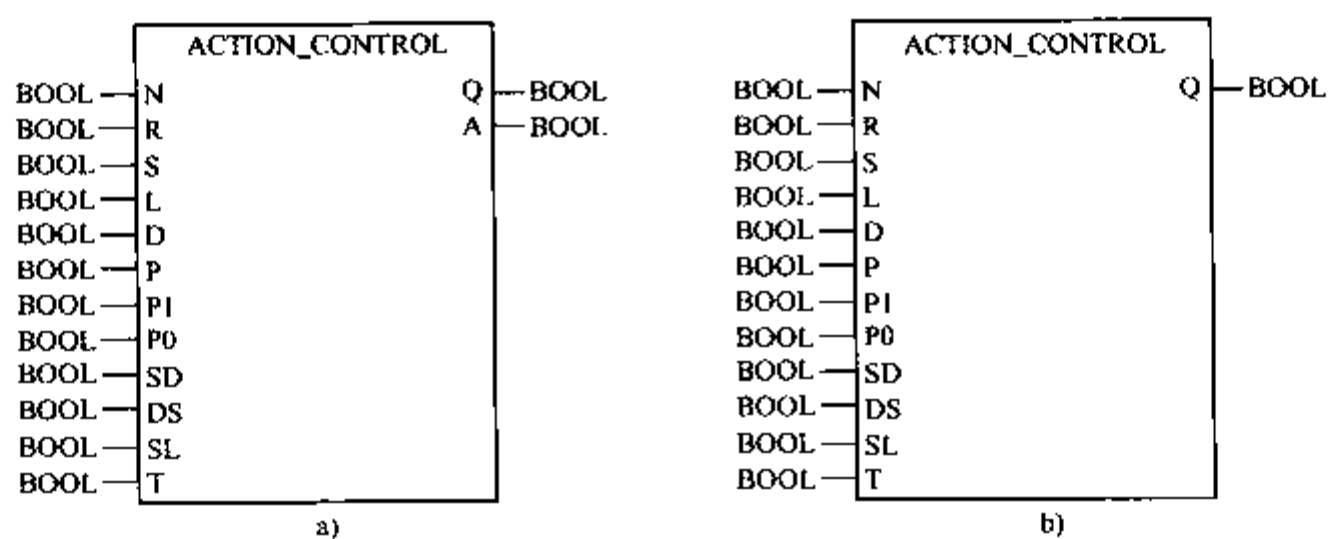


图 5-16 动作控制功能块的外部接口(对用户不可视)

a) 带最后扫描逻辑(见图 5-17a) b) 不带最后扫描逻辑(见图 5-17b)

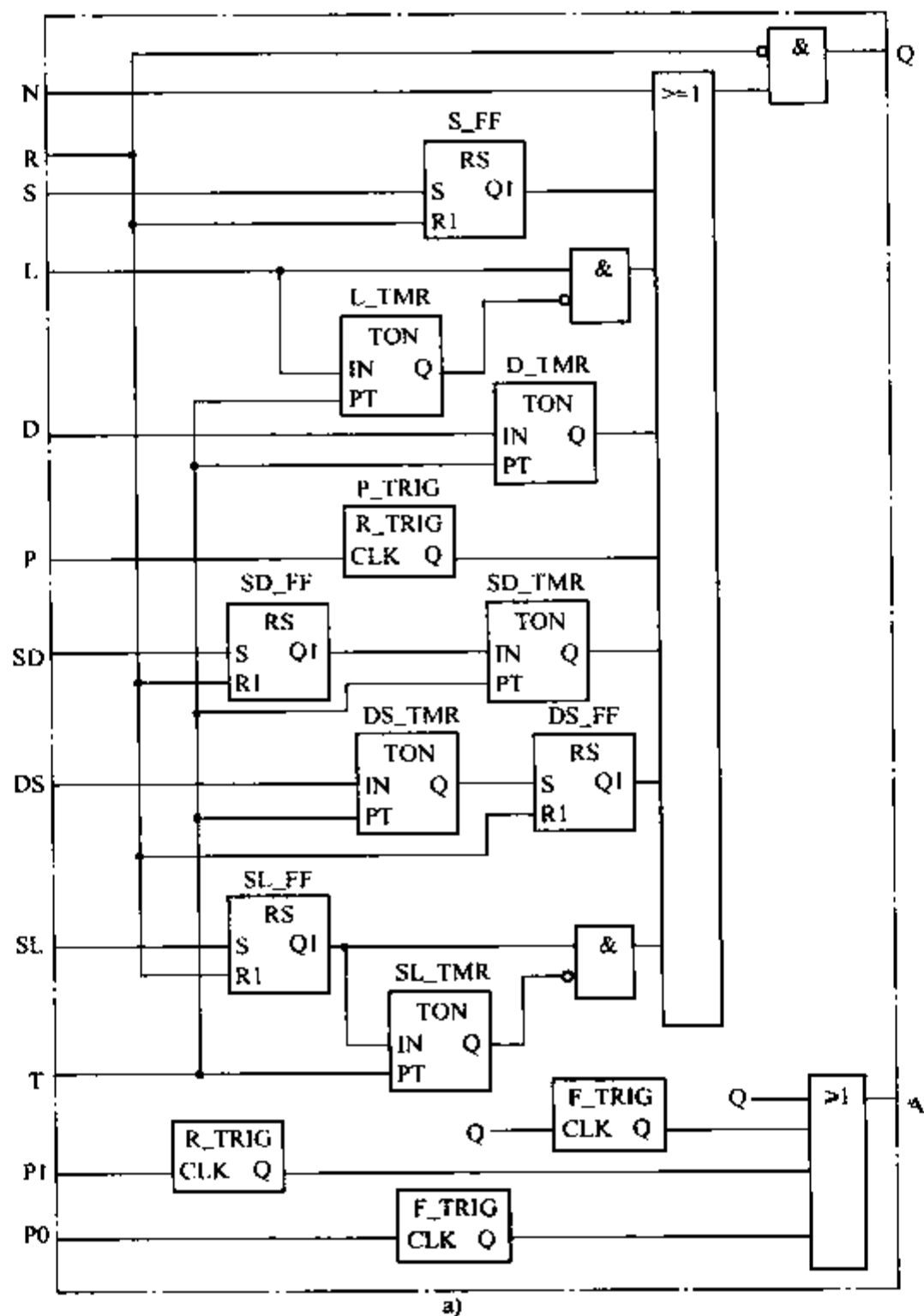


图 5-17 动作控制功能块

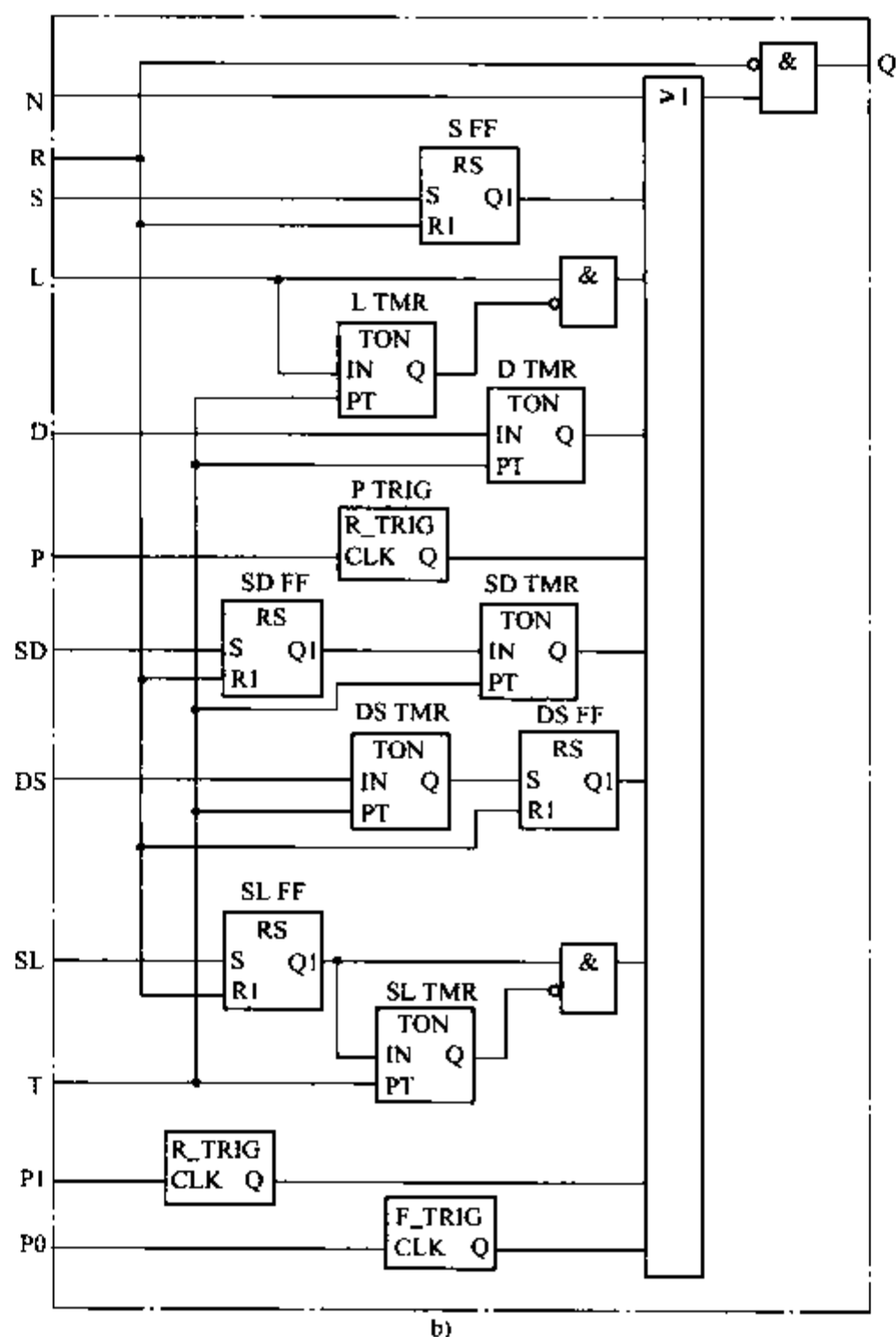


图 5-17 动作控制功能块(续)

a) 带最终扫描逻辑的动作控制功能块 b) 不带最终扫描逻辑的动作控制功能块

注:1. 该功能块对用户是不可见的。

2. 图 5-17a 所示功能块类型的外部接口在图 5-16a 给出;图 5-17b 所示功能块类型的外部接口在图 5-16b 给出。

(4) 动作控制功能块输入 T

它是一个 TIME 类型数据,仅当使用 L、D、SD、DS 和 SL 限定符时,才需要输入与动作持续时间有关的数据 T。

(5) 限定符

从图 5-17 可见,各种限定符定义不同输入和输出 Q 之间的关系。应根据限定符确定步标志变量与动作之间的逻辑关系。

- 无限定(N)表示不对动作控制功能块有任何的限定。当于其结合的步是活动步时,该动作控制功能块就被执行。

- 存储(S)和复位(R)限定符成对出现。复位(R)限定符用于对具有存储(S)限定符的动作控制功能块进行复位。当某动作控制功能块的限定符为R时,该动作被复位。如表5-8中S、SD、DS、SL限定符所定义的动作控制功能块要用R限定符的动作控制功能块来复位。
- SD和DS是不同的两种限定关系。图5-18显示了存储延迟(SD)和延迟存储(DS)限定符在逻辑关系上的区别。

图5-18a中,在步S2设置限定符SD,在步S6设置限定符R。当步S2是活动步时,经SR功能块存储,并经TON功能块延时T后输出F。因此,在步S2成为活动步后,只要步S2与步S6成为活动步的时间间隔大于T,就不必考虑步S2的持续时间S2.T是否大于T,系统就有输出F,即动作F被执行。

图5-18b中,在步S2设置限定符DS,在步S6设置限定符R。当步S2是活动步时,经TON功能块延时T后的信号才送SR功能块,因此,如果步S2的持续时间S2.T小于T,就不会有动作F被执行。

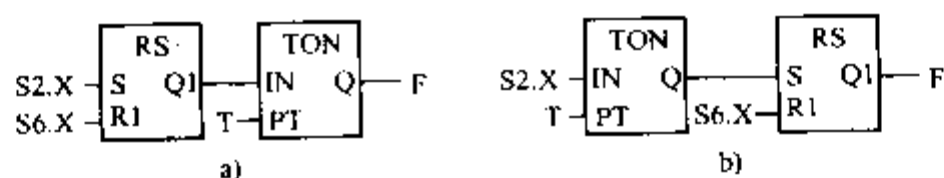


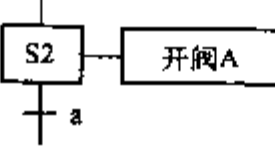

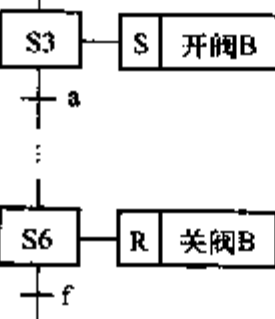
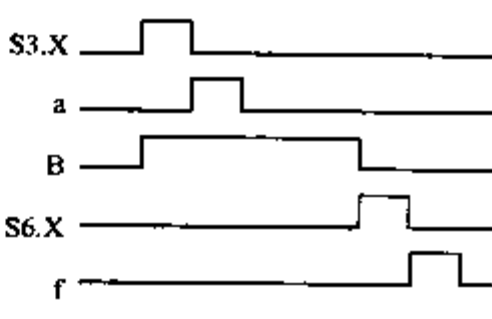
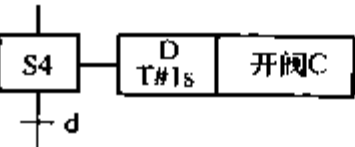

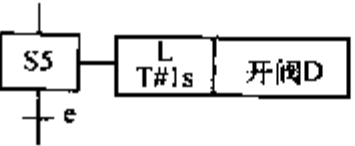

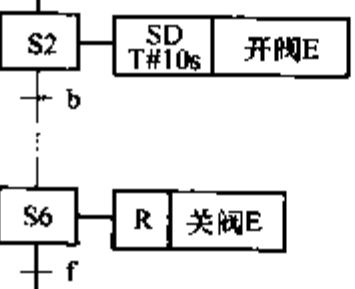
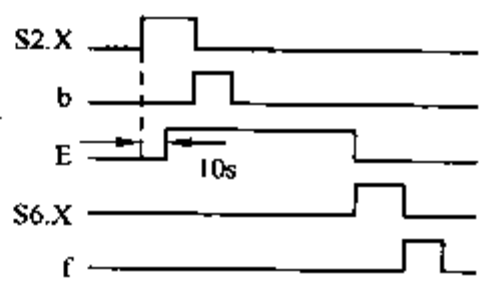
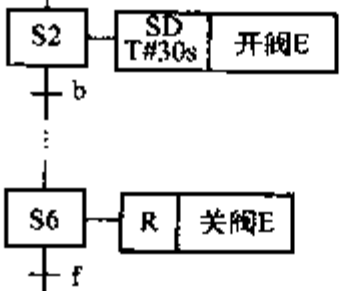
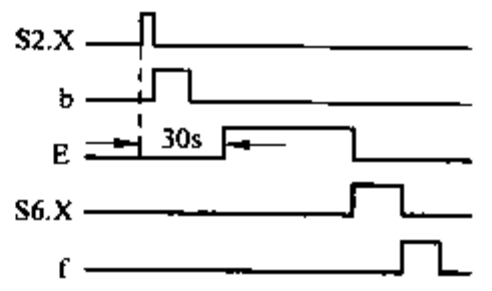
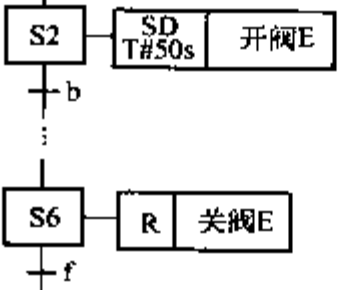
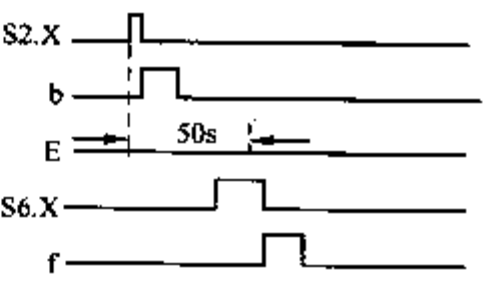
图5-18 SD和DS限定符的区别

a) SD限定符 b) DS限定符

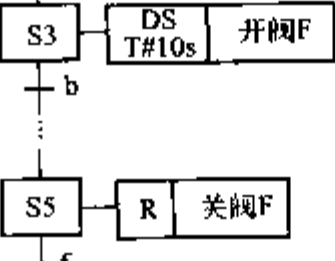
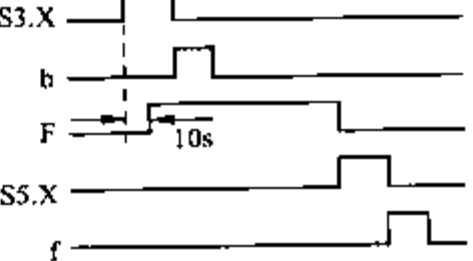
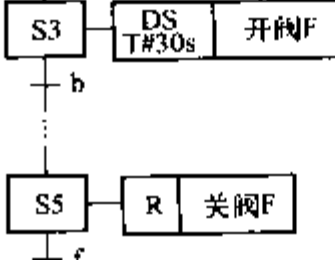
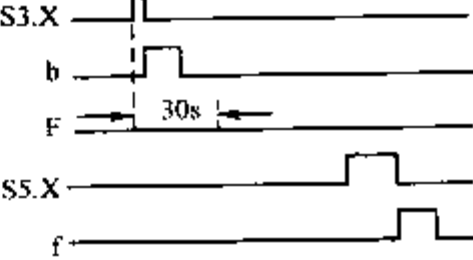
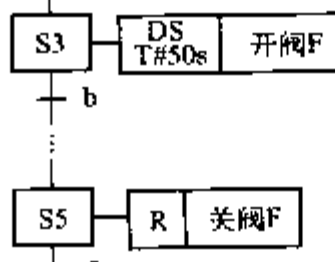
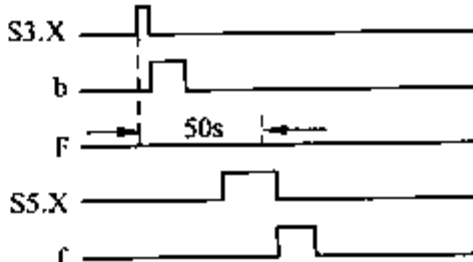
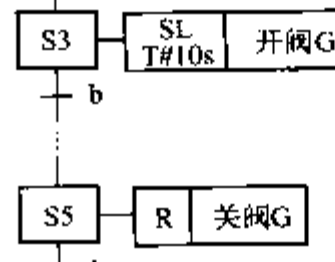
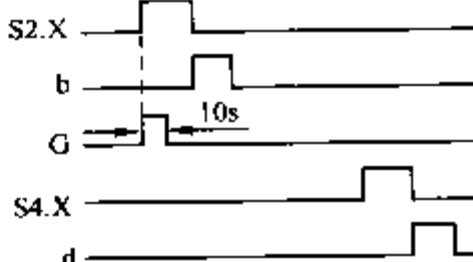
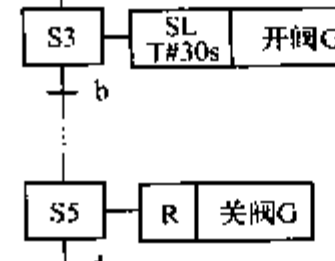
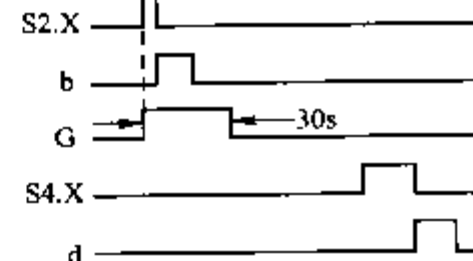
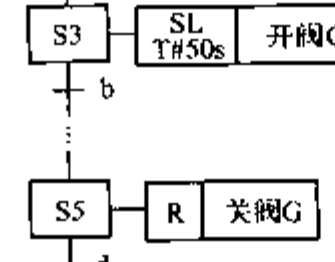
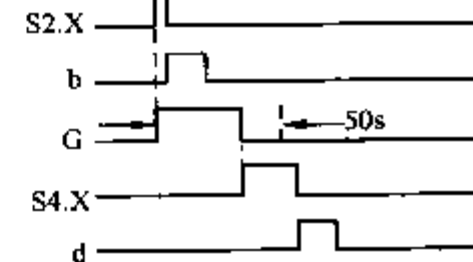
- 时限(L)限定符用于说明动作或命令执行时间的长短。例如,动作蒸汽控制阀打开40s,表示蒸汽阀打开的时间是40s。
- 延迟(D)限定符用于说明动作或命令在获得执行信号到执行操作之间的时间延滞,即所谓的时滞时间。例如,冷水泵运转后10s才打开泵下游的控制阀。这里,10s是延迟时间。
- 存储时限(SL)是存储和时限的结合。用于将信号先存储,再时限,因此,当步成为活动步时它被执行,并执行一个给定的时间。
- 脉冲(P)限定符用于提供一个脉冲触发,即当步成为活动步时,动作被执行,并执行到下一扫描周期。
- 带最后扫描逻辑的动作控制功能块和不带最后扫描逻辑的动作控制功能块中,脉冲P0和P1限定符的功能是不同的。不带最后扫描逻辑的动作控制功能块中,P1限定符的功能与P限定符具有相同的脉冲输出功能。它们都在步成为活动步(上升沿触发)的瞬间输出一个脉冲信号。P0限定符则在步从活动步成为非活动步时(下降沿触发)的瞬间输出一个脉冲信号。带最后扫描逻辑的动作控制功能块中,P0和P1限定符用于控制动作控制功能块的输出A,它不用于控制输出Q。此外,带最后扫描的动作控制功能块的输出会出现最后扫描造成的脉冲信号。

为说明限定符的作用,表5-8显示了顺序功能表图中说明限定符的时序图和功能说明。图中波形显示不带最后扫描的动作控制功能块执行的情况。

表 5-8 顺序功能表图中说明动作限定符的时序图和功能说明

功能表图限定符	时 序 图	功 能 说 明
		<p>非存储型(空或 N)</p> <p>当 S2 成为活动步时, 阀 A 打开</p> <p>当 S2 成为非活动步时, 阀 A 关闭</p>
		<p>存储型(S 和 R)</p> <p>当 S3 是活动步时, 阀 B 打开, 并保持在打开的状态</p> <p>当 S6 是活动步时, 阀 B 关闭, 并保持在关闭的状态</p>
		<p>延迟型(D)</p> <p>当 S4 是活动步后 1s 才打开阀 C</p> <p>当 S4 激活时间小于 1s 时, 阀 C 不能被打开</p>
		<p>时限型(L)</p> <p>当 S5 是活动步后, 阀 D 打开, 并打开 1 秒; 如果激活时间小于 1s, 则随 S5 成为非活动步而阀 D 关闭</p>
		<p>存储延迟型(SD 和 R)</p> <p>S2.X = 1 时, 延时 10s 开阀 E 并保持</p> <p>S6.X = 1 时, 阀 E 关闭并保持到下一次开阀信号</p> <p>S2.X = 1 的时间大于 10s</p> <p>步 S2 与步 S6 之间的时间大于 10s</p>
		<p>存储延迟型(SD 和 R)</p> <p>S2.X = 1 时, 延时 30s 开阀 E 并保持</p> <p>S6.X = 1 时, 阀 E 关闭并保持到下一次开阀信号</p> <p>虽然 S2.X = 1 的时间小于 30s, 但步 S2 与步 S6 之间的时间大于 30s, 因此, 阀 E 能打开</p>
		<p>存储延迟型(SD 和 R)</p> <p>S2.X = 1 的时间小于 50s, 而步 S2 与步 S6 之间的时间也小于 50s, 因此, 阀 E 不能打开, 并保持关闭, 直到下一次开阀信号</p>

(续)

功能表图限定符	时序图	功能说明
		延迟存储型(DS 和 R) S3. X = 1 时, 延时 10 s 开阀 F 并保持 S5. X = 1 时, 阀 F 关闭并保持到下一次开阀信号 S3. X = 1 的时间大于 10 s 步 S3 与步 S5 之间的时间大于 10 s
		延迟存储型(DS 和 R) S3. X = 1 时, 延时 30 s 后 S3. X 已 = 0, 因此不能打开阀 F 因 S3. X 的时间小于 30 s 虽然步 S3 与步 S5 之间的时间大于 30 s, 但是, 阀 F 不能打开, 它被保持关闭, 直到下一次开阀信号
		延迟存储型(DS 和 R) S3. X 的时间小于 50 s, 而步 S3 与步 S5 之间的时间小于 50 s, 因此, 阀 F 不能打开, 它保持关闭, 直到下一次开阀信号
		存储时限型(SL 和 R) S2. X = 1 时, 打开阀 G 保持打开 10 s S2. X = 1 的时间大于 10 s 但步 S2 与步 S4 之间的时间大于 10 s, 因此阀 G 保持打开 10 s
		存储时限型(SL 和 R) S2. X = 1 时, 打开阀 G 虽然 S2. X = 1 的时间小于 30 s, 但步 S2 与步 S4 之间的时间大于 30 s, 因此阀 G 保持打开 30 s
		存储时限型(SL 和 R) S2. X = 1 的时间小于 50 s, 而步 S2 与步 S4 之间的时间小于 50 s, 因此, 阀 G 保持打开, 直到 S4. X = 1 时关闭, 即阀 G 打开的时间小于 50 s

【例 5-15】 动作控制功能块示例。

图 5-19 显示了 SFC 程序中动作控制功能块和其等效的功能块图程序。

图中, 动作 HV_BREAKER 在步 S22 是活动步(S22. X = 1)时被执行, 它的布尔指示变量

就是 S22 步的转换条件 HV_BRKR_CLOSED。

动作控制功能 START_INDICATOR 是存储型动作,它的启动条件是 S22. X = 1,停止条件是 S27. X = 1。

动作控制功能块 RUNUP_MONITOR 是存储时限型。它的启动条件是 S23. X = 1,停止条件是动作执行时间到 1 s 或 S27. X = 1。

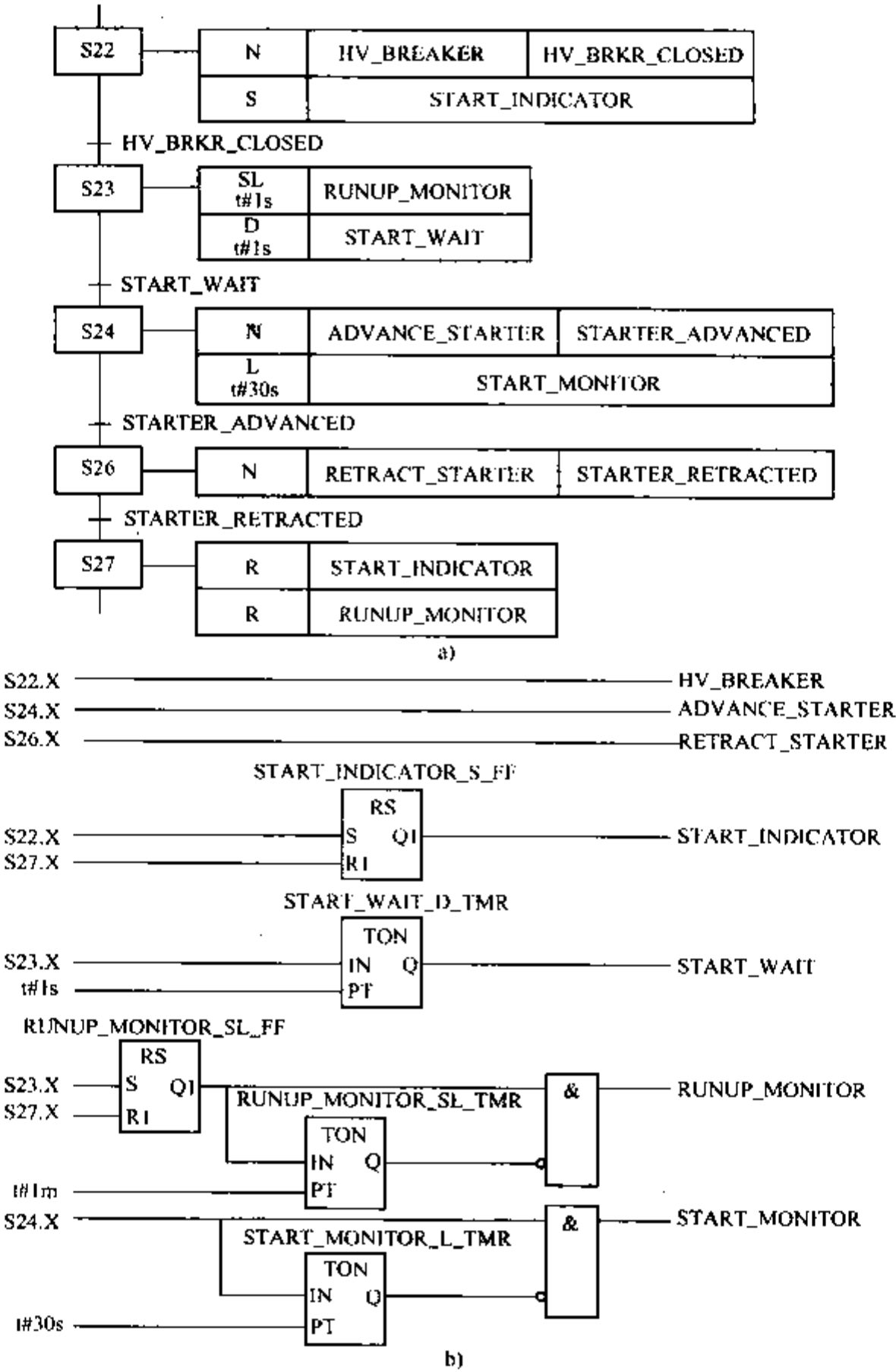


图 5-19

a) 动作控制功能块示例-SFC 表示 b) 动作控制功能块示例-等效功能块图

注:完整的 SFC 网络和它的有关说明未在示例中显示。

动作控制功能块 START_MONITOR 是时限型。它的启动条件是 S24. X = 1, 停止条件是动作执行时间到 30 s 或 S27. X = 1。

根据图 5 - 17b, 可方便地将图 5 - 19a 的 SFC 图转换为图 5 - 19b。它采用功能块描述各变量之间的关系。最上面的 3 条直接连线, 表示用 MOVE 功能块实现。

(6) 标准规定对带最后扫描的动作控制功能块, 当其输出 Q 为 1 (TRUE) 时, 与控制功能模块有关的动作被执行。同时, 当 Q 的下降沿时, 该动作被最后扫描, 从而再被执行一次。其结果是动作在动作有关的限定符被清除后再被执行一次。额外的执行发生在有关动作已经不被激活后, 会造成一些敏感和不可预料的副作用, 因此不被采用。

3. 演变(进展)规则

SFC 编程语言中, 步的进展和动作执行、转换实现有关。步的演变(进展)规则如下:

1) 步的演变(进展)从初始步开始, 因此初始步在程序执行开始时应是活动步。任何与初始步结合的动作被最先执行。

2) 步的演变(进展)过程中, 步的后继转换条件满足, 当该转换是使能转换时, 就发生转换的实现, 从而实现步的进展, 即该转换的所有前级步从活动步变为非活动步, 而该转换的所有后级步从非活动步变为活动步。

3) 与步结合的动作控制功能块只有在步是活动步时才被执行, 动作的执行根据动作控制功能块的输出 Q 确定, 当 Q 为 1 时, 这些动作被执行, 当 Q 为 0 时, 这些动作不被执行。

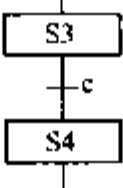
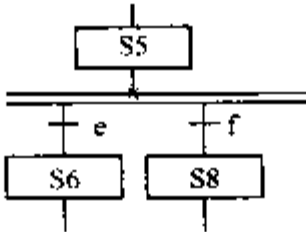
4) 在动作的执行过程中, 对后继转换进行求值, 当转换条件满足时, 发生转换的实现, 并实现步的进展, 从而使步根据 SFC 图的程序不断演变(进展)。

5) 选择序列的分支, 有多个转换条件, 它们是有优先级的。当不设置优先级时, 转换条件的判别是从左到右进行的。当设置优先级时, 用数字表示优先的等级, 数字越小优先级越高。表 5 - 9 显示了不同序列中, 步的演变(进展)过程。

6) 步的激活需要一定的时间, 同样, 转换的实现也需要一定的时间。不同的 PLC 产品, 所需时间不同。因此, 当活动步连接的动作具有延时限定功能时, 可能出现在活动步要启动动作时, 发生了实现转换的过程, 从而出现不活动的步中还有正在执行的动作这种现象。

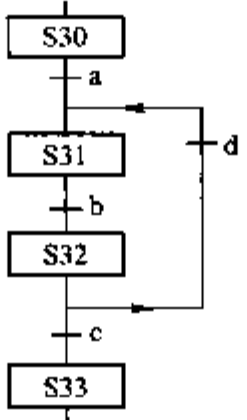
7) 根据限定符, 一些动作因其使能的条件已不满足时, 该动作就被标记为非活动。例如, 时限限定符标志的动作在时限规定的时间达到后就成为非活动的。它要到下一次 SFC 的求值时, 才再执行。

表 5 - 9 步的演变(进展)规则

序号	示 例	规 则
1		单序列 单序列中, 步和转换交替重复 仅在 S3 是活动步, 转换条件 c 为真时, 发生从步 S3 到步 S4 的进展
2a		选择序列: 分支 * 号表示转换进展的优先级: 从左到右 S5 为活动步, 转换条件 e 为真时, S5 进展到 S6, 转换条件 f 为真及 e 为假时, 才进展到 S8

(续)

序号	示 例	规 则
2b		<p>选择序列:分支</p> <p>* 号表示转换进展的优先级根据数字大小:从小到大</p> <p>S5 为活动步,转换条件 f 为真时,S5 进展到 S8,转换条件 e 为真及 f 为假时,才进展到 S6</p>
2c		<p>选择序列:分支</p> <p>S5 为活动步,转换条件 e 为真时,S5 进展到 S6,转换条件 e 为假及 f 为真时,才进展到 S8,应注意,分支时转换条件互斥</p>
3		<p>选择序列:合并</p> <p>S7 是活动步且 h 为真时,发生步 S7 到步 S10 的进展</p> <p>S9 是活动步且 j 为真时,发生步 S9 到步 S10 的进展</p>
4a		<p>并行序列:分支</p> <p>S11 是活动步,当 b 为真,同时发生到 S12、S14 的进展,然后 S12 和 S14 各自独立进行</p>
4b		<p>并行序列:合并</p> <p>连接到水平双线上面的各步为活动步且 d 为真时,才从步 S13 和 S15 进展到步 S16</p>
5a 5b 5c		<p>序列跨接(Sequence Skip)</p> <p>选择序列的特例,在分支中没有步</p> <p>图示是 5a,对应于 2a,当 S30 是活动步时,如果 a 为假且 d 为真,则发生从 S30 到 S33 的进展,使 S31 和 S32 被跨接(跳过)</p> <p>d 的有向连线方向向下,因此表示方向的箭头可不画出</p>
6a 6b 6c		<p>序列回路(Sequence Loop)</p> <p>选择序列的特例,分支的一个或几个返回到前级步</p> <p>图示为 6a,对应于 2a, S32 步的选择可到 S33,也可返回到 S31。而 S31 可从 S30 或 S32 的选择序列合并</p> <p>当 c 为假且 d 为真时,S32 返回到 S31,出现重复的循环回路</p> <p>d 的有向连线方向向上,因此表示方向的箭头必须画出</p>

序号	示 例	规 则
7		有向连线的方向 为清楚显示有向连线的方向,可用“<”字符和“>”字符表示有向连线的方向是从左到右和从右到左。使用时在两边添加水平连线符“-”,如图所示

4. 不安全顺序功能表图和不可达顺序功能表图

在设计时,需防止出现不安全顺序功能表图(Unsafe SFC)和不可达顺序功能表图(Unreachable SFC)。图 5-20 是错误的顺序功能表图设计示例。

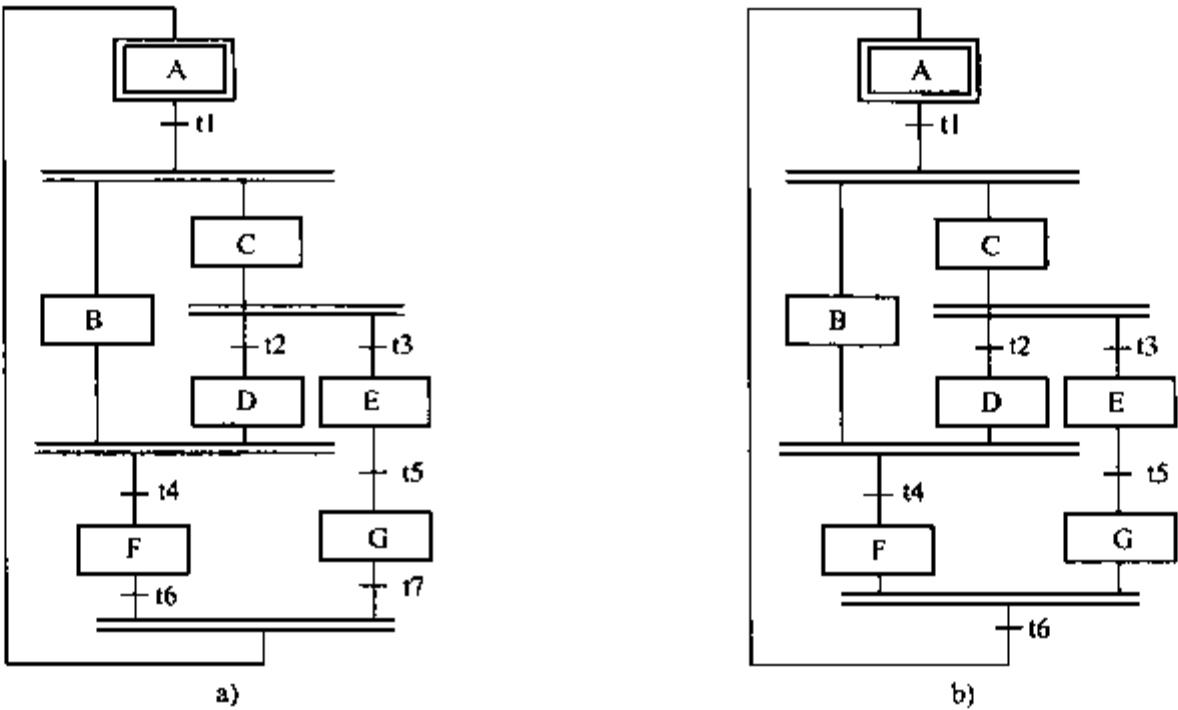


图 5-20 错误的 SFC 设计
a) 不安全 SFC 序列 b) 不可达 SFC 序列

图 5-20a 中,如果转换 t1 的转换条件满足,则步 B 和步 C 成为活动步,如果转换 t4 的转换条件为假,则步 B 会保持在活动步状态,但如果转换 t3 的转换条件满足,在步 E 成为活动步后,转换 t5 的转换条件满足,并经步 G,在转换 t7 的转换条件满足后仍使步 A 成为活动步,并在转换 t1 的转换条件满足时,使步 B 成为活动步。如果这时,步 B 由于转换 t4 的转换条件没有满足而仍处于活动步状态,那么,图 5-20a 设计的 SFC 会造成对步的不可控制。因此,这样的 SFC 网络是不安全的。

图 5-20b 中,如果转换 t1 的转换条件满足,则步 B 和步 C 成为活动步,当转换 t3 的转换条件满足时,使步 E 成为活动步,并在转换 t5 的转换条件满足时,使步 G 成为活动步。由于当步 E 成为活动步时,步 C 成为非活动步,这时,步 D 不能成为活动步,这表明步 D 是不可达的步。同样,由于步 D 不可达,则转换 t4 也不能成为使能转换,从而步 F 也成为不可达。因此,

图示的 SFC 网络是不可达的 SFC 网络。

为防止出现不安全和不可达 SFC 网络,需要保证在同步序列之外没有序列的跳转。

5.3.2 顺序功能表图的兼容

顺序功能表图编程语言既具有图形编程语言的特点,也具有文本编程语言的特点,因此,在 IEC 61131-3 标准中,它被作为公用元素定义。通常,由于它的图形功能强,常将它归为图形编程语言。顺序功能表图的兼容是指它可用图形编程语言的方式进行编程,也可用文本编程语言的方式进行编程。

例如,步既可用带步名的矩形框表示。也可用 STEP...END_STEP 结构的文本方式描述。表 5-6 的转换不仅可用图形符号表示,即可在 ST、LD、FBD、IL 编程语言中用图形符号表示,也可用文本形式表示。对动作和动作控制功能块等都具有这种兼容性。

5.4 示例

5.4.1 冲压机控制系统

1. 控制要求

冲压机用于对工件进行冲压成形。冲压机用液压控制系统驱动,采用电磁阀 DCV1 和 DCV2 进行换向,控制冲压头向下和向上的运动。图 5-21 是冲压机工作原理简图。

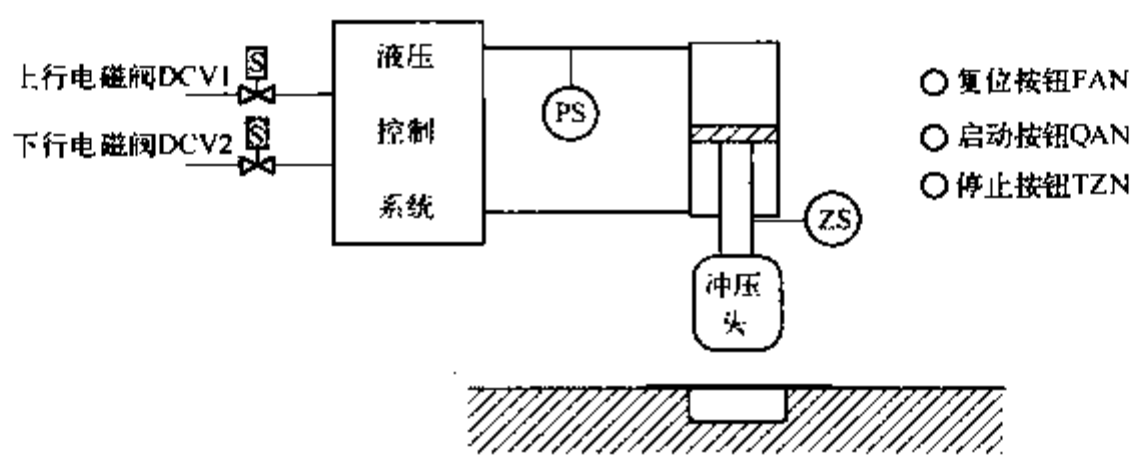


图 5-21 冲压机工作原理简图

冲压机工作过程如下:操作员按下复位按钮 FAN,电磁阀 DCV1 激励,使冲压头上移,直到位置开关 ZS 闭合。操作员将需冲压的工件放到冲压位置,并按下启动按钮 QAN,电磁阀 DCV2 激励,使冲压头下移,冲压和拉伸被加工工件,液压不断升高,到液位到达设定压力,压力开关 PS 闭合,进入保压,这时,DCV2 保持激励状态,定型时间为 5 s,然后电磁阀 DCV2 失励,电磁阀 DCV1 激励,冲压头上行,直到回复到位置开关 ZS 闭合。操作员取出已冲压的工件,如果需再冲压,则将需冲压工件放到冲压位置,准备下一次冲压。如果需停止,则按下停止按钮 TZN,停止冲压。

2. 编程

据冲压机的控制要求,编写图 5-22 所示的顺序功能表图。

图中,S001 是初始步,该步没有连接任何动作。S002 是复位步,用于冲压头的复位。S003

是冲压步,用于对工件进行冲压。S004 是回复步,用于下一次冲压或停止冲压。

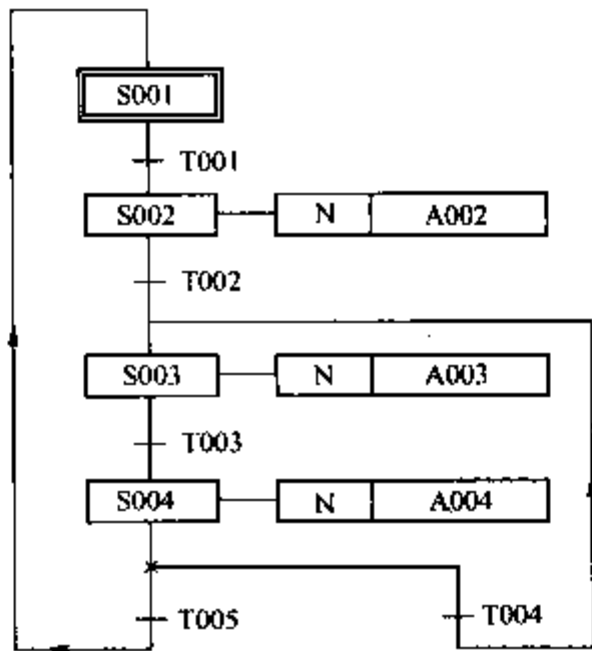


图 5-22 冲压机顺序功能表图程序

(1) 转换条件的编程

共有 5 个转换条件,编程如下:

1) 转换名 T001。T001 转换条件是按复位按钮。用 IL 编程语言编写转换条件如下:

```
LD    S001.X    (* 读取步 S001 的状态标志 *)
AND   FAN        (* 与复位按钮 FAN 信号进行与逻辑运算 *)
ST    T001       (* 如果按下复位按钮,则转换条件 T001 为真 *)
```

2) 转换名 T002。转换条件 T002 是位置开关 ZS 状态和起动按钮状态的与逻辑结果。用 LD 编程语言编写的程序如图 5-23 所示。

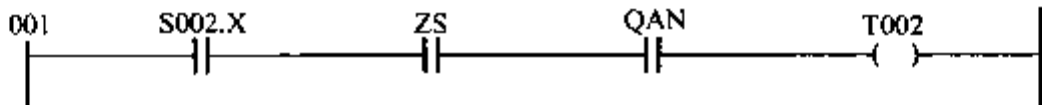


图 5-23 T002 转换条件的梯形图程序

程序中不仅使用 ZS 和 QAN 两个信号,还使用步 S002 的状态 S002.X,该信号是可有可无的。

3) 转换名 T003。T003 转换条件是压力开关 PS 为 1 后,延时 5 s 的信号,用功能块图编写的程序如图 5-24 所示。

4) 转换名 T004。T004 转换条件用于下一次冲压。转换条件是冲压头复位、起动信号和未按通知按钮的与逻辑运算结果。ST 编程语言编写的程序如下:

```
T004 := ZS AND QAN ANDN TZN; (* 冲压头复位和按下
                                起动按钮及未按停止按钮 *)
```

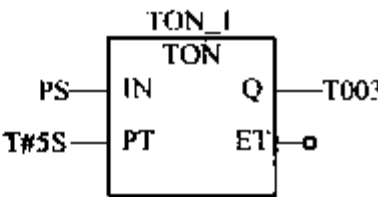


图 5-24 T003 的功能块图程序

5) 转换名 T005。T005 转换条件用于停止,它只需要按下停止按钮,对冲压头是否复位没有要求。但考虑到选择序列的互锁,因此,将 QAN 信号包含在程序中。IL 编程语言编写的程序如下:

```
LDN  QAN      (* 读取起动按钮的反相状态 *)
AND  TZN      (* 与停止按钮 TZN 信号进行与逻辑运算 *)
ST   T005     (* 运算结果作为转换条件 T005 *)
```

(2) 动作控制功能块的编程

该控制系统共有 3 个动作控制功能块。

1) A002。动作控制功能块 A002 完成冲压头复位操作。用 IL 编程语言编写程序如下：

```
LD    S002.X   (* 读取步 S002 的状态 *)
ST    DCV1     (* 打开电磁阀 DCV1 *)
```

复位按钮 FAN 是脉冲信号,因此,用步 S002 的状态作为电磁阀 DCV1 的激励信号,保证在步 S002 是活动步时,电磁阀 DCV1 都处于激励状态。

2) A003。完成冲压头下行进行冲压的操作,用 LD 编程语言编写的程序见图 5-25。

T003 转换条件和 A003 动作控制功能块的程序有一定联系,例如,在 A003 中用图 5-23 的程序外,再添加定时器对 PS 计时的程序,则 T003 转换条件是计时到的信号。这说明转换条件和动作控制功能块的程序并不是惟一的,在编程时应相互配合。



图 5-25 A003 动作控制功能块的程序

3) A004。完成冲压头上行复位的操作,ST 编程语言编写的程序如下：

```
DCV1 := S004.X;
```

(3) 注意事项

1) 用限定符 N 限定的动作控制功能块,可用所连接的步状态作为该动作控制功能块的操作信号,例如,用 S002、S003 和 S004 的步状态作为操作信号。

2) 动作控制功能块和转换条件是相互影响的。因此,各步连接的动作控制功能块和其后续的转换条件的程序可转换。例如,A003 和 T003 的程序。

3) 用户可用其熟悉的编程语言编写程序,提高程序的准确性和可靠性。

3. 调试

调试工作可在实际物理装置中进行,也可在仿真器中进行。对简单的程序可直接在物理装置中进行,对复杂的程序宜先在仿真器中进行。上述程序经物理装置调试合格。

5.4.2 交通信号控制系统

1. 控制要求

交警上班后,将起动开关 START 切换到自动(START = 1),交通信号灯根据下列控制要求自动切换。

1) 南北红灯点亮 13 s,同时,东西绿灯点亮 8 s,然后,东西绿灯闪烁 3 s,东西黄灯点亮 2 s。

2) 自动切换,东西红灯点亮 15 s,南北绿灯点亮 10 s,南北绿灯闪烁 3 s,然后,南北黄灯点亮

2 s。

图 5-26 显示了交通信号灯的控制时序。

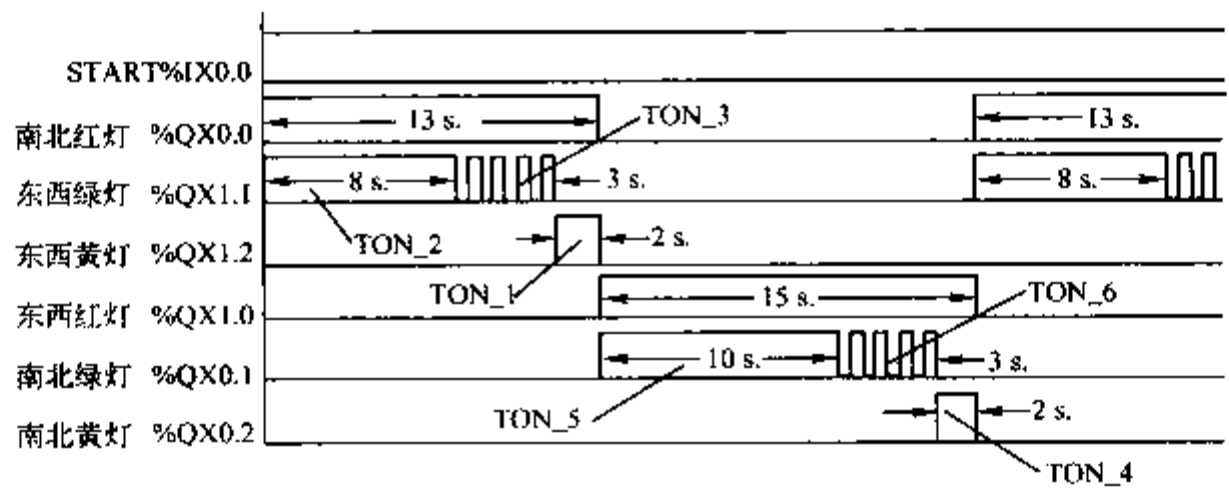


图 5-26 交通信号灯的控制时序图

交通信号控制系统有 1 个输入信号 START,地址为%IX0.0,有 6 个输出信号,即东西和南北向的红、绿和黄灯。示例用于说明如何用 SFC 编程语言编程,因此,对信号灯系统作了简化,例如,没有设置行人的交通信号灯控制程序,也没有设置在交警下班后的黄灯交替点亮的程序等。

2. 交通信号控制系统的 SFC 编程

(1) 确定 SFC 图

根据控制系统的控制要求确定 SFC 图,包括设置步名、转换名、动作名或动作控制功能块实例名及有向连线的走向,完成步与转换、转换与步之间的连接等。图 5-27 是交通信号控制系统的 SFC 图。

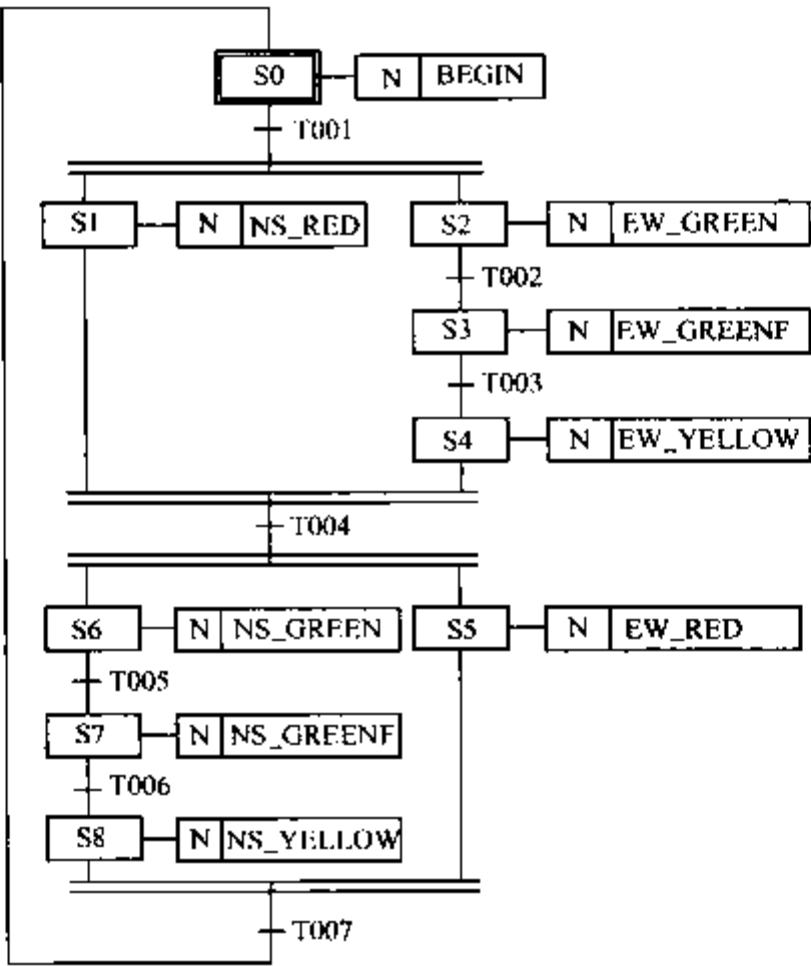


图 5-27 交通信号控制系统的 SFC 图

图中,用 S0 表示初始步,它用双线矩形框表示。S1 到 S8 分别是操作步,它与相应的动作连接。交通信号控制系统是简单控制系统,因此,各连接动作的限定符都设置为 N,表示不存储。连接的动作名分别是 BEGIN、NS_RED、EW_GREEN、EW_GREENF、EW_YELLOW、EW_RED、NS_GREEN、NS_GREENF、NS_YELLOW,用于表示东西、南北各信号灯的点亮和闪烁的控制要求。各步之间的转换名 T001、T002、T003、T004、T005、T006、T007 表示各转换条件。

(2) 转换条件的编程

转换条件编程时,可设置有关变量,并对转换条件进行编程,编程语言可采用 LD、ST、IL 或 FBD 等编程语言。如图 5-27 所示,共有 7 个转换条件,编程如下:

1) 转换名 T001。该转换条件是起动开关信号 START,因此,用 ST 编程语言编程如下:

```
T001 := START;
```

2) 转换名 T002。该转换条件用于计时,当步 S2 的消逝时间达到 8 s 时,T002 置 1。因此,用定时器 TON_2 实现消逝时间的计时。如果软件系统有消逝时间,也可直接使用,即 S2.T。用 IL 编程语言编程如下:

```
LD      S2.X          (* 读取步 S2 的步标志 S2.X *)
ST      TON_2.IN       (* 数据送定时器 TON_2 输入 IN *)
LD      T#8S          (* 设置定时器 TON_2 的设定时间为 8 秒 *)
ST      TON_2.PT       (* 数据送定时器 TON_2 输入 PT *)
CAL     TON_2          (* 调用定时器 TON_2 *)
LD      TON_2.Q        (* 取定时器 TON_2 输出 Q *)
ST      T002          (* 结果送转换条件的转换名 T002 *)
```

3) 转换名 T003。该转换条件用于计时,当步 S3 的消逝时间达到 3 s 时,T003 置 1。因此,用定时器 TON_3 实现消逝时间的计时。用 ST 编程语言编程如下:

```
TON_3(IN := S3.X, PT := T#3S); (* 调用 TON_3 *)
T003 := TON_3.Q;                (* 定时器输出送转换条件 *)
```

4) 转换名 T004。该转换条件用于计时,它用于对步 S4 的消逝时间进行计时。当消逝时间达到 2 s 时,转换条件 T004 置 1。因此,用定时器 TON_1 实现消逝时间的计时。用 FBD 编程语言编程的图形如图 5-28 所示。

T004 转换条件也可用步 S1 的消逝时间进行计时。这时的消逝时间为 8 s + 3 s + 2 s = 13 s。

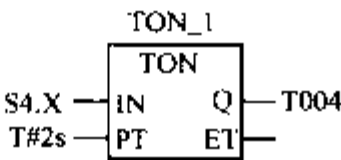


图 5-28 T004 转换条件

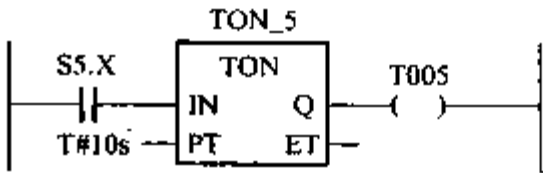


图 5-29 T005 转换条件

5) 转换名 T005、T006 和 T007。T005、T006 和 T007 转换条件的编程与 T002、T003 和 T004 转换条件的编程类似。为便于学习,用 LD 编程语言编程的 T005 转换条件图形如图 5-29 所示。用 ST 编程语言编程的 T006 转换条件如下:

```
TON_6(IN := S7. X, PT := T#3S); (* 调用 TON_6 *)
T006 := TON_6. Q; (* 定时器输出送转换条件 *)
```

T007 转换条件采用对步 S5 的消逝时间计时。因此,定时器设定值应为 $10\text{ s} + 3\text{ s} + 2\text{ s} = 15\text{ s}$ 。用 ST 编程语言编程如下:

```
TON_4(IN := S5. X, PT := T#15S); (* 调用 TON_4 *)
T007 := TON_4. Q; (* 定时器输出送转换条件 *)
```

注意,程序中的定时器 TON_4 与图 5-28 有区别,因为,图 5-28 中的 TON_1 对步 S4 的消逝时间计时,而上述程序对步 S5 的消逝时间进行计时。

(3) 动作或动作控制功能块的编程

动作或动作控制功能块的编程时,可设置有关变量,并对动作或动作控制功能块进行编程,编程语言可采用 LD、ST、IL 或 FBD 等编程语言。如图 5-27 所示,共有 9 个动作控制功能块需编程。

输出信号灯变量名称如表 5-10 所示。

表 5-10 输出信号灯变量名称

变量描述	南北红灯	南北绿灯	南北黄灯	东西红灯	东西绿灯	东西黄灯
变量名	NS_R	NS_G	NS_Y	EW_R	EW_G	EW_Y
变量地址	%QX0.0	%QX0.1	%QX0.2	%QX1.0	%QX1.1	%QX1.2

1) 动作控制功能块 BEGIN。动作控制功能块 BEGIN 是起动前的初始动作。在 SFC 程序中,通常在初始步设置一些初始值等。本示例不需要设置初始值,因此,该动作是一个空操作,程序中也可删除该块。

2) 动作控制功能块 NS_RED。该动作控制功能块用于点亮南北方向的红色信号灯。用 LD 编程语言编程,如图 5-30 所示。该动作采用 N 限定符,是非存储型。因此,当步 S1 成为非活动步时,动作 NS_RED 不再执行。为此,程序中用转换条件作为南北红灯的熄灭条件。可以看到,当步 S1 是活动步时,南北红灯 NS_R 点亮,当转换条件 T004 满足时,T004 常闭触点断开,S1 步成为非活动步,南北红灯 NS_R 熄灭。

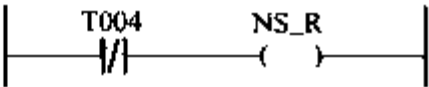


图 5-30 南北红灯
动作程序图

3) 动作控制功能块 EW_GREEN。该动作控制功能块用于点亮东西方向的绿色信号灯。用 IL 编程语言编程如下:

```
LD S2. X (* 读取步状态 S2. X *)
ST EW_G (* 存 EW_GREEN *)
```

与 NS_RED 的动作类似,但用步 S2 的状态 S2. X 来确定东西向绿灯的点亮。当状态 S2. X 满足时,表示步 S2 是活动步,因此,东西向绿灯 EW_G 点亮。当计时器 TON_2 的计时时间到,T002 为真,实现转换,并使其常闭触点断开,EW_G 东西向绿灯熄灭。

4) 动作控制功能块 EW_GREENF。该动作控制功能块用于东西方向的绿色信号灯的闪烁。为此采用振荡信号发生器线路。例如,采用图 4-22 所示的振荡信号发生器。图 5-31 是用 LD 编程语言编写的功能块体程序,它包含绿灯的闪烁线路。

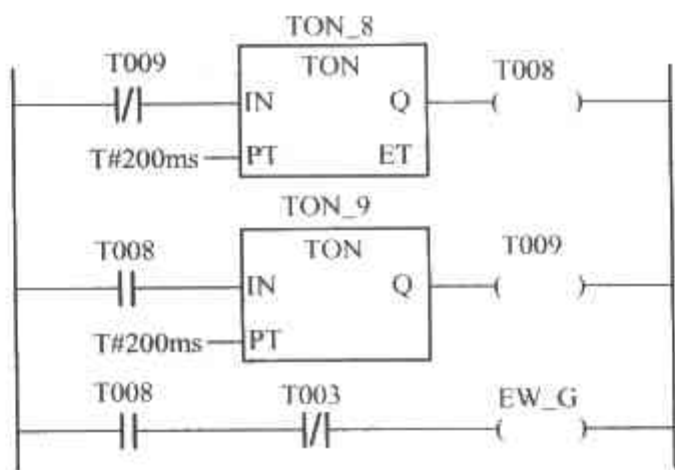


图 5-31 东西绿灯闪烁动作控制功能块图

由于步 S3 成为活动步后,步 2 成为非活动步,因此,与步 2 连接的动作块(采用限定符 N)不被存储,即东西向绿灯熄灭。为此,本动作控制功能块用于控制东西向绿灯闪烁。图中,TON_8 和 TON_9 是两个定时器功能块实例,采用相同计时设定值 200ms。转换条件 T003 计时到,使步 S3 成为非活动步,同时,使闪烁的绿灯熄灭。

5) 动作控制功能块 EW_YELLOW。该动作控制功能块用于点亮东西方向的黄色信号灯。用 ST 编程语言编写的功能块如下:

EW_Y := NOT T004; (* 用赋值语句将转换条件 T004 取反后送 EW_Y *)

6) 动作控制功能块 EW_RED、NS_GREEN、NS_GREENF、NS_YELLOW。这些动作控制功能块的编程方法与上述编程方法类似,不多述。

3. 调试

对编写的程序进行检查和调试,对出现的问题和故障进行分析,并排除,直到程序能够按控制系统控制要求进行控制。

用 MULTIPROG 对上述控制系统进行编程,并用该软件自带的仿真器 DEMOIO-DRIVER 进行仿真。图 5-32 是仿真器显示画面。

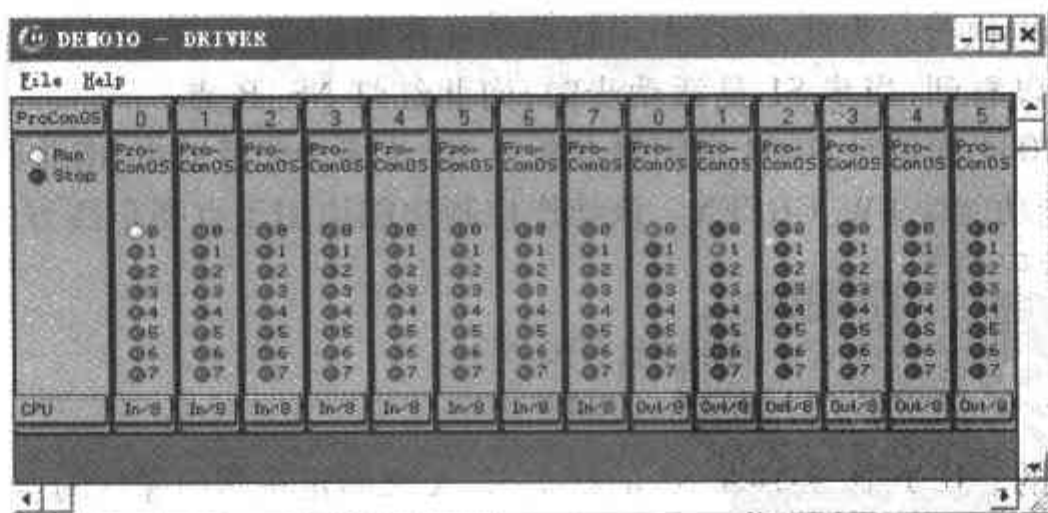


图 5-32 仿真器显示画面

仿真器画面显示仿真器有 8 块开关量输入卡和 8 块开关量输出卡(图中仅显示出 5 块)。每块卡有 8 个点。图中第 0 号输入卡第 0 点表示%IX0.0,第 0 号输出卡第 0 点表示%QX0.0,根据上面分配的地址,第 0 号输出卡第 0 点表示南北红灯,第 1 号输出卡第 1 点表示东西绿灯等。图中,当鼠标左键点击第 0 号输入卡第 0 点时,该信号灯点亮,表示 START = 1。因此,控

制系统运行,图中,输出卡的两个红灯分别表示南北红灯和东西绿灯点亮。

仿真结果显示程序运行正确。各信号灯能够根据控制要求正确点亮、闪烁和熄灭。因此,所编写程序正确,满足所需控制要求,可将程序下装 PLC 和实际运行。

4. 编程时的注意事项

- 1) 防止不安全和不可达序列出现。
- 2) 各转换名、动作控制功能块名是惟一的,应避免出现与变量名同名的现象。
- 3) 各转换条件、动作控制功能块内的本体程序可采用 IL、LD、ST、FBD 和 SFC 编程语言编写,编程人员可根据对编程语言的熟悉程度选用。
- 4) 变量的数据类型应匹配,防止因数据类型选择不当而出错。
- 5) 根据 IEC 61131-3 标准,可使用步的状态标志和步的消逝时间。当一些软件产品不提供消逝时间时,可在该步添加定时器,用于对该步进行计时,如示例所示。
- 6) 可采用各种限定符对动作控制进行限定。例如,可用存储 S、复位 R 等限定符使某一步中的动作延续到另一步成为活动步时才终止。

5.4.3 物料混合控制功能块

本示例用于说明如何用顺序功能表图来编写用户功能块。

1. 物料混合控制的要求

图 5-33 是物料混合生产过程的流程示意图。

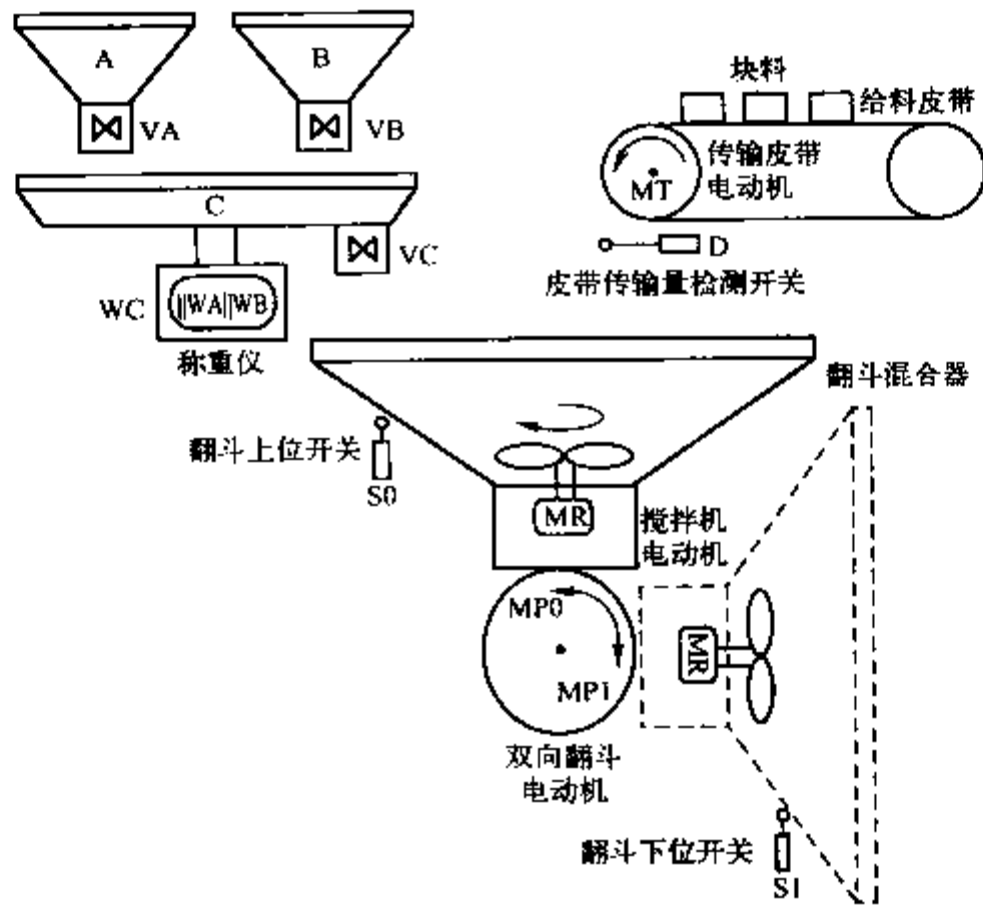


图 5-33 物料混合生产过程流程图

操作员先设置称重仪的皮重为 Z。检查翻斗混合器位于上位($S0 = 1$),称重料斗显示重量小于等于皮重时,如果控制系统接收到 STT(手动信号或来自其他系统的自动信号)信号时系统起动,料罐 A 出料阀 VA 打开,物料 A 送称重料斗 C,当称重仪称得重量到 $WA + Z$ 时,关闭 VA,打开料罐 B 出料阀 VB,物料 B 送称重料斗 C,当称重仪称得重量到 $WA + WB + Z$ 时,关闭

VB,并打开称重料斗的放料阀 VC。在添加液体物料的同时,给料皮带电动机 MT 运转,开始输送块料 1,当皮带传输量检测开关 D 为 1 时,停止块料 1 的添加,并开始添加块料 2,当皮带传输量检测开关 D 为 1 时,停止块料 2 的添加,并停止给料皮带电动机 MT。

称重料斗的放料阀 VC 打开或块料还未称重到量时,开始启动搅拌电动机 MR,搅拌电动机运转 T1 时间后,翻斗电动机反转,将混合物料放出,在反转过程中,搅拌电动机仍运转,到翻斗到下位(S1 =1)时,搅拌电动机才停止。然后,翻斗电动机正转信号 MP0 为 1,使翻斗电动机正转,翻斗到上位开关 S0 为 1 时,才停止反斗电动机 MP 的运转。

2. 程序设计

(1) 确定输入输出变量和数据类型

该控制系统有输入输出变量,见表 5 - 11。

表 5 - 11 变量表

输入变量名	数据类型	描 述	输入变量名	数据类型	描 述
STT	BOOL	启动指令	DONE	BOOL	工作状态
D	BOOL	皮带传输量检测开关	VA	BOOL	物料 A 出料阀
S0	BOOL	翻斗上位开关	VB	BOOL	物料 B 出料阀
S1	BOOL	翻斗下位开关	VC	BOOL	称重仪放料阀
WA	INT	物料 A 设定值	MT	BOOL	传输皮带电动机
WB	INT	物料 B 设定值	MR	BOOL	搅拌电动机
WC	WORD	BCD 表示的称重值	MP0	BOOL	翻斗电动机正转信号
Z	INT	皮重	MP1	BOOL	翻斗电动机反转信号
T1	TIME	混合时间			

(2) SFC

1) 变量声明。由于这样的物料混合系统有多套。因此,需编写用户功能块。这里,用 SFC 编程语言编写 MIX_2 功能块的功能块本体程序。功能块变量声明如下:

```
VAR_INPUT
    STT  : BOOL ; ( * 启动信号 * )
    D    : BOOL ; ( * 皮带传输量检测开关 * )
    S0   : BOOL ; ( * 混合翻斗上位开关 * )
    S1   : BOOL ; ( * 混合翻斗下位开关 * )
    WA   : INT ;   ( * 物料 A 计量设定值 * )
    WB   : INT ;   ( * 物料 B 计量设定值 * )
    Z    : INT ;   ( * 称重仪皮重 * )
    WC   : WORD  ( * BCD 表示的称重量 * )
    T1   : TIME ; ( * 搅拌混合时间 * )
END_VAR
VAR_OUTPUT
    DONE : BOOL ; ( * 工作状态 * )
    VA   : BOOL ; ( * 物料 A 进料阀 * )
    VB   : BOOL ; ( * 物料 B 进料阀 * )
```

VC : BOOL ; (* 称重仪放料阀 *)
 MT : BOOL ; (* 传输皮带电动机 *)
 MR : BOOL ; (* 搅拌电动机 *)
 MP0 : BOOL ; (* 翻斗电动机正转信号 *)
 MP1 : BOOL ; (* 翻斗电动机反转信号 *)

END_VAR

用图 5-34 表示该功能块的输入输出变量和数据类型。

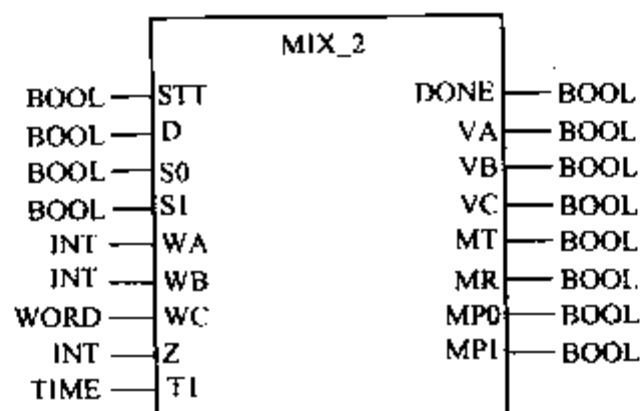


图5-34 物料混合控制功能块的图形符号

2) SFC 编程。

• 功能块 MIX_2 的本体。用 SFC 语言编写,如图 5-35 所示。

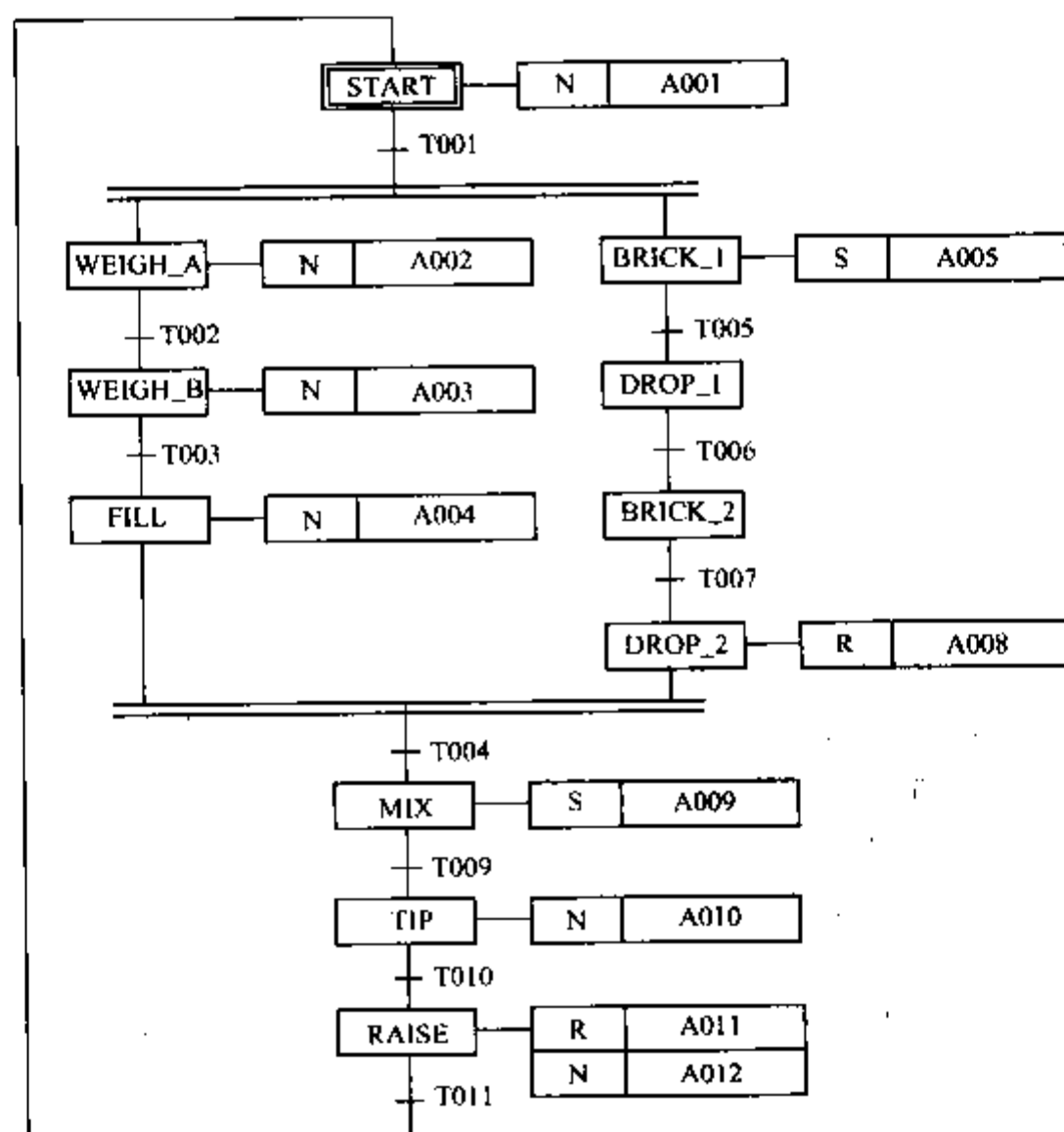


图 5-35 物料混合控制功能块 MIX_2 本体的 SFC 程序

- 转换条件。由于各转换条件比较简单,这里,用列表方式表示各步对应的转换条件。各步的转换条件见表 5-12。

表 5-12 转换条件表

转换名	转换条件	转换名	转换条件
T001	STT AND S0 AND (WORD_BCD_TO_INT(WC) <= Z)	T006	NOT D
T002	WORD_BCD_TO_INT(WC) >= (WA + Z)	T007	D
T003	WORD_BCD_TO_INT(WC) >= (WA + WB + Z)	T009	MIX. T >= T1
T004	(WORD_BCD_TO_INT(WC) <= Z) ANDN D	T010	SI
T005	D	T011	S0

- 动作控制功能块。与各步结合的动作控制功能块的动作比较简单,见表 5-13:

表 5-13 动作控制功能块

功能块实例名	限定符	完成的动作	功能块实例名	限定符	完成的动作
A001	N	DONE ; = 1	A008	R	MT ; = 0
A002	N	VA ; = 1	A009	S	MR ; = 1
A003	N	VB ; = 1	A010	N	MP1 ; = 1
A004	N	VC ; = 1	A011	R	MPO ; = 0
A005	S	MT ; = 1	A012	N	MR ; = 0

3) 转换条件和动作控制功能块编程时注意下列事项:

- 转换条件可用第 5.1.3 节介绍的各种方法编程。例如,直接列写有关转换条件在转换附近,也可用 LD、FBD、IL 和 ST 等编程语言编写。例如,可编写如图 5-36 所示的程序实现 T001 的转换条件。

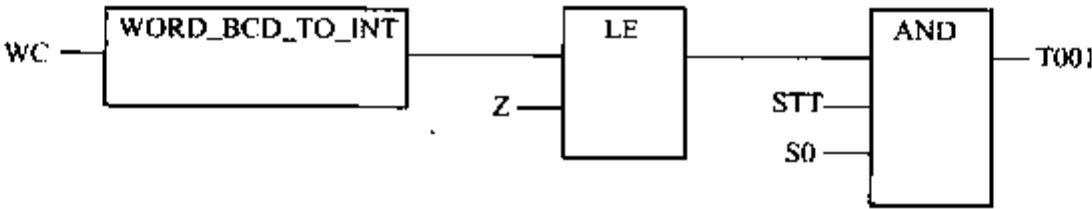


图 5-36 T001 转换条件的程序

- 当限定符为 S 或 R 时,表示与该步结合的动作具有记忆功能。因此,其动作将被保持。例如,与步 MIX 结合的动作 A009 完成起动搅拌电动机 MR 的动作,由于该动作是存储型动作,因此,电动机 MR 将一直运转,直到与步 RAISE 结合的动作 A011 执行。A011 是复位型动作,当步 RAISE 为活动步时,执行停止电动机 MR 的动作,它也同时使翻斗电动机正转信号为 1,使翻斗电动机正转。
- 限定符 N 不具有存储功能,因此,当该步为非活动步时,与该步连接的动作不再继续。例如,WEIGH_A 步为 1 时,执行打开 VA 阀的动作,当转换条件满足,即称重仪的称量达到 A 物料量时,VA 阀自动随着该步成为非活动步而关闭。

本示例说明功能块的程序也可用 SFC 编程语言编写。各种编程语言之间可相互结合,使编程环境更方便。

5.5 发挥 SFC 的潜力,改善当前 PLC 程序的开发实践

5.5.1 程序开发过程中必须遵循的基本方法

为了改善当前 PLC 程序的开发实践,保证程序开发的质量,提高程序开发的效率,实现软件的模块化是前提,即软件程序必须由若干被称为模块的部分所构成。这些部分之间的耦合关系相对较弱,而且每个部分都应该是独立开发和单独测试的。为此,程序开发过程中必须遵循一些基本方法,这些方法必须建立在下列基础上:

(1) 强调软件的总体设计

提高总体设计阶段在整个程序开发过程中的权重,即首先明确定义控制程序的结构,将各种控制和处理要求划分若干个模块,并确定这些模块之间的相互关系,然后才开始具体的程序开发。

软件总体设计阶段对该设计形成的软件程序产品的质量有重要影响。一个考虑周到、构思严谨的设计规范,对每一个程序开发人员所写代码的正确程度和开发效率有着相当大的影响。即使由于程序开发人员的不够成熟可能带来一些缺陷,往往也很容易在测试和调试阶段发现和改正,甚至完全消除。反之,如果设计规范不周全,甚至分析错误,对整个软件产品的损害必定相当严重,要解决它所付出的代价也是昂贵的。即使这个软件产品仍然可用,但软件维护花费时间、复用率低、难以客户化和难以扩展。

从经济因素看,一个有缺陷的设计使设计规范不确切、不准确,由此引起的程序毛病要克服起来代价不菲。如果消除一个程序缺陷的代价为 1,那么,要消除由于软件总体设计有错导致的程序缺陷所付出的代价可能是 10,而一个功能性的定义有问题,要找出问题的所在,再解决这个问题,所花费的代价就可能是 100。

(2) 强调标准

应该充分利用工控编程语言 IEC 61131-3 在程序设计阶段和开发阶段的潜力。

目前尚无一种专为 PLC 软件的程序设计和开发的方法,可以借鉴已经广泛支持传统(顺序)软件的开发生命周期的适合的方法,来寻求一种解决方案。不过这种解决方案应该是在围绕 IEC 61131-3 标准的开发环境下,与这种传统方法具有最佳的兼容性。

使用 IEC 61131-3 标准编程语言的优点之一是即使分属于不同单位的不同的程序开发人员,他们之间仍具有交换信息和程序库的可能性。因此,在程序开发外包、分包正在逐渐形成一种发展模式的时候,其重要性不言自明。虽然采用 IEC 61131-3 标准编程语言的方法,也许没有某种专用的设计方法效率高,但绝大多数程序开发人员都很熟悉,无须专门进行培训。

国际编程语言标准 IEC 61131-3 规定了软件模型、各种通用元素和 5 种编程语言,而且每种语言在表达控制功能的能力方面都具有一定的实际背景。这就是说,这 5 种编程语言都是依据工业控制的基本元器件及由其构成的网络或电路,采用某种在计算机上仿真它们的工作原理和功能而形成的。

梯形图(LD)语言是将并行动作的机电元件(诸如继电器触点和线圈、定时器、计数器等)

网络加以模型化。功能块图(FBD)语言是将并行动作的电子元件(诸如加法器、乘法器、移位寄存器、逻辑运算门等)的网络予以模型化。结构化文本(ST)语言将典型的信息处理任务(如在通用的高级语言 Pascal 中使用数值算法)予以模型化。指令表(IL)语言是将汇编语言中适用于控制系统的低层编程予以模型化。顺序功能表图(SFC)将时间驱动和事件驱动的顺序控制设备和算法模型化。

5.5.2 采用 SFC 作为设计系统控制的主要工具

国际编程语言标准并没有规定哪些语言可以用在 PLC 软件开发生命周期的哪个阶段。推荐采用 SFC 作为设计系统控制的主要工具,而其他的语言则用在控制软件代码开发阶段的主要原因如下:

(1) 表达控制要求的能力强

状态图和 Petri 网是公认的最适合表达模型动态过程的工具,广泛地被应用于许多领域。SFC 语言与状态图一样有着很强的表达控制功能的潜力,特别在面对并行控制问题时,较之采用 Petri 网,SFC 更显出其特色。SFC 在系统行为模型化的能力是其本质的一种体现。

(2) 图形的形式方法

SFC 不仅具有 IEC 61131-3 标准中规定的图形原语,而且与梯形图语言(LD)和功能块图语言(FBD)相比,还具有描述系统动态过程的高级特性。由于图形的语法相当简明,所以容易学习和使用。此外,值得注意的是,在过程表达时,SFC 可根据需要使用,既可以详尽,也可以简略。

(3) 支持初步设计

在设计初始阶段,由于许多方面尚未清晰,或者设计人员还未了解之前,仍可用 SFC 的图形形式方法进行系统行为的形式表达,因此,SFC 在初步设计阶段就是一种有价值的工具。它能够避免在用自然语言描述系统规范时可能出现的模糊不清,增强客户、设计者和编程人员之间的设计思想的沟通。

(4) 支持详细设计

早期设计阶段运用 SFC 得出的方案,可以在取得新的信息之后逐渐深化和细化,尤其是与状态步关联的动作块可进一步细化,使之成为新的嵌套的 SFC 方案。通过不断深化,获得所需设计深度。不论在初步设计阶段还是在详细设计阶段,SFC 在语义和语法上的连贯性,对编程人员来讲是一致的。

(5) 支持软件的分段执行

在处理控制软件时,SFC 语言可按不同的扫描执行周期简便地分割为多个程序段,为缩短最大扫描时间提供了先决条件。应该指出,在运用 SFC 语言表达程序段的分割时均是显性的,按执行条件是否满足来启动不同程序段的执行顺序,而执行条件都是显性条件,不存在隐含条件。

5.5.3 重视 SFC 语言与其他 4 种编程语言的互补关系

综上所述,在设计控制程序的结构时务必采用 SFC 语言,而只在后面程序代码的开发阶段才用其他 4 种编程语言。例如,基于 IEC 61131-3 标准的编程系统 ISaGRAF(就是采用上述

思路。

(1) 与 ST 编程语言的互补

运用 SFC 作为设计阶段的语言工具时,结构化文本语言 ST 作为它在写代码阶段的自然互补是较为合适的。因为在采用 SFC 描述控制结构时,用高级语言 ST 来表达与其每一步相关的算法显得有效和简洁,ST 语言在表达状态转移的条件有效的能力方面具有清晰的可读性。由于 ST 的通用形式与 C 语言和 Pascal 语言十分接近,在应用 SFC 时其每一步的动作和转移条件都可以用 ST 语言来写代码。

(2) 与 LD 编程语言的关系

20 世纪 80 年代 PLC 发展的鼎盛阶段,LD 语言广泛应用于北美和日本,而后再流传到全世界。由于 LD 语言十分接近于 PLC 的广大用户电气自动化专业所常用的电路原理图,特别是在表达继电逻辑方面,二者有着天然的联系。但在结构化的表达上,LD 语言不如 ST、IL 语言。此外,在大多数情况下用 LD 写的程序代码都很长。

(3) 与 IL 编程语言的关系

IL 编程语言与汇编语言很接近,其优点是 IL 语言编写的程序经过编译就得到 PLC 执行扫描运算的机器码。如果有一定的专门函数库,为表达某种算法,用 IL 来代替 ST 将提供很大的方便。相对于 ST 语言,如果有专家级的编程人员用 IL 语言编程,其程序的性能肯定好于用 ST 编写的程序。用 IL 语言的缺点是程序可读性差,形成的代码较长。

(4) 与 FBD 编程语言的关系

以 SFC 描述的控制结构中用功能块图语言 FBD 来控制模块十分合适,特别是复杂的控制功能尤其有效和方便。在处理由多个输入信号导出输出信号的这类问题时,FBD 语言可对关键要素给出极为简洁和明确的描述。在应用函数和功能块时,运算操作必须完全用精确的优先关系予以表达。

(5) LD 和 IL 编程语言替代 ST 语言的问题

由于历史的原因,用 LD 和 IL 编程语言编写的程序数量多,且都经过大量使用和相当深度的测试,只要这些程序可以复用,则在新的应用中仍然有使用的价值。但要很好地利用这些程序,需要 PLC 的编程系统能够将这些程序模块化,能按照准确的逆向工程判据将其分隔成一定的程序块。虽然得到的模块仍用原来的编程语言,但已经被组织到模块库中,程序开发人员只要调用,就可以构成新的应用程序。

模块化分为结构性的模块化和功能性的模块化。结构性的模块化是依据对控制系统的设计技术规范,通过对控制组成部分的分析进行的模块化。在进行结构性的模块化时,不仅要考虑满足本项目的控制要求,而且要考虑这些模块的复用性,即重复可使用性。结构性模块化时只能正确运用 SFC 语言作为最重要的工具。

功能性的模块化是正确表达一组输入参数与一个或多个输出参数之间特定的规律性关系,实现功能性的模块化运用的工具常常是 FBD 语言。

图 5-37 给出了 IEC 61131-3 标准规定的 5 种语言在 PLC 软件设计中的作用。横坐标表示不同的设计阶段,纵坐标表示语言的级别(与语言的结构完整性和表达的能力相关描述)。

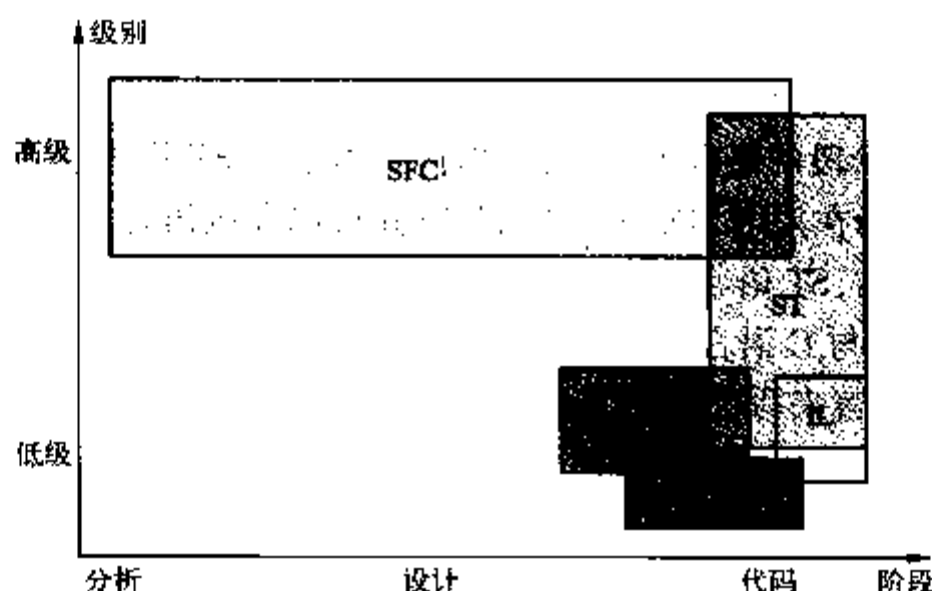


图 5-37 IEC 61131-3 标准规定的 5 种语言在 PLC 软件设计中的作用

5.5.4 SFC 与状态图的对应关系及异同

有限状态自动机是一类简明而有效的算法，状态图为它提供了图形化的表达。由于有限状态自动机是以非程序化的方式进行一种形式化的描述，所以特别容易处理。状态图作为一种有效的形式化工具，特别适于表达动态过程，因而被广泛地接受。

SFC 能够用来表达一个系统的控制具有状态图所有的特性，它与状态图之间最重要的区别是 SFC 中每个单步必须执行控制活动（即 SFC 中的动作）。

1. 状态图的有关概念

为了改善 PLC 的软件设计，便于与 SFC 语言比较，对状态图的有关概念介绍如下：

(1) 状态的有限集合 S

每个状态表示一个特定、有效的用来表征所描述的现象的状况。在描述一个系统的动态过程时，辨认此有特定含义的状况是将控制问题进行清晰分隔的基础。

(2) 事件的集合 E

有许多事件会引发由一个状态转换为另一个状态。在一个自动化系统中，这些事件可来自传感器的信号的组合，操作员发出的命令，以及某些内部变量达到给定的数值等。

(3) 初始状态 I

自动化的执行总是从状态集合中的某个被称为是初始状态的状态开始的。初始状态是惟一的。在一个自动化系统中，初始状态与一种恰当而又相关的初始阶段相对应。

(4) 最终状态的集合 F

最终状态集合 F 是状态集合 S 的一个子集。F 集合中的每个状态，对应于一个恰当的状况，系统就在此刻结束其执行过程。若系统停在某个不属于 F 的状态，这意味着发生了异常的终止。

(5) 转换规则的集合 T

转换规则用 $\langle s_i, e_{ij}, s_j \rangle$ （与当前状态 s_i 、使转换至一个新状态得以发生的事件 e_{ij} ，以及新达到状态 s_j 这 3 个元素相关）定义。T 集合中包含的转换规则的数量等同于自动化系统规定的状态转换的数量。

(6) 转换规则

使得离开每个状态的转换规则必须由不相交的事件所引发,以保证所达到的状态是惟一可辨认的。换言之,两个转换规则: $\langle si, ex, sj \rangle$ 和 $\langle si, ex, sk \rangle$ 不能属于同一个转换规则集 T 。

【例 5-3】 状态图表示的示例

图 5-38 给出了状态图表示的示例。

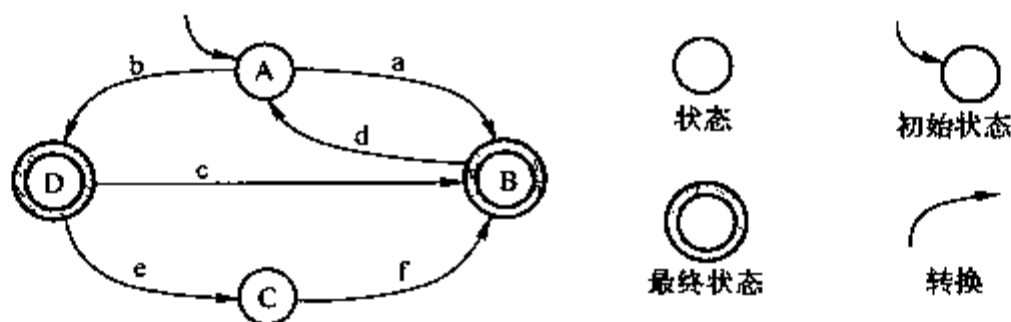


图 5-38 状态图表示的示例

根据图例,可知,示例有 A~D 等 4 个状态,其中,B 和 D 两个状态是最终状态,有 a~f 共 6 个转换规则。根据上述,可形式化定义如下:

$$\begin{aligned} S &= \{ A, B, C, D \} \\ E &= \{ a, b, c, d, e, f \} \\ I &= A \\ F &= \{ B, D \} \\ T &= \{ \langle A, a, B \rangle, \langle A, b, D \rangle, \langle D, c, B \rangle, \langle B, d, A \rangle, \langle D, e, C \rangle, \langle C, f, B \rangle \} \end{aligned}$$

2. 状态图转换为 SFC 时遵循的规则

状态图表示的系统的动态行为,必须用 SFC 方案将其转换为控制算法,即用与状态图中的状态数量相同的“步”,而每一步则包含“动作”(信号分析、比较、计算、命令发出等),来执行该系统其所对应的状态。

在设计 PLC 软件时,状态图非常有用,它所对应的 SFC 方案表现为一种理想的结构。但有两个概念应分清。

- 系统发生状态的改变是由控制发出的命令或其内部事件产生的结果。
- 用软件程序让系统的状态处于连续控制之下,以至于系统能以最短的时间做出反应。即使状态的改变是由控制所强制,这也要等到所发生的转换被证实是来自系统,才能够移至下一步。

程序编制人员在实现状态图到 SFC 转换时,必须遵循下列规则:

- 1) 步与转换是交错的;
- 2) 每个转换都对应于一个惟一的初始步(源点步)。
- 3) 每个转换都对应于一个惟一的目標步。
- 4) 步既可以是转换的源点,又可以转换的目标。
- 5) 每个转换都要定义一个使它得以进行的事件。
- 6) 可以通过多个转换,到达一个单一的步。
- 7) 可以通过多个转换,从一个单一的步离开。
- 8) 由不同的步离开的转换,可以采用同样的使转换得以发生的事件。

9) 到达给定步的转换,可以采用同样的使转换得以发生的事件。

10) 从一个给定步离开,必须采用不相容的转换事件。

状态图和 SFC 方案的语义差别,常常会导致一些非专业的编程者产生某些错误。

5.5.5 SFC 与 Petri 网的关系及异同

Petri 网是法国数学家 C. A. Petri 在 1962 年创立的一种描述条件和事件之间关系的数学工具,在控制领域被广泛采用。在描述离散动态事件系统控制时,它是一种很有效的工具。SFC 语言基于 Petri 网开发的编程语言 Grafcet。Grafcet 很早就成为国际标准 IEC 60848,它是主要针对顺序控制过程的控制功能和控制条件,按自顶而下的方法,以类似流程图的顺序,来进行设计和编程的通用化语言。

由于 Petri 网定义的行为规则,Petri 网是由处所(Places)和转换(Transitions)两种类型节点组成的有向图,节点由带箭头的弧线连接,连接规则如下:

1) 一根弧线可以将一个处所与一个转换连接,或者从一个转换连向一个处所,但不可以将一个处所和另一个处所连接,也不可以由一个转换连向另一个转换。

2) 一个处所可以由多个弧线所到达。

3) 一个转换可以由多个弧线到达。

4) 给定的瞬间每个处所被其自己的状态所表征,而系统的状态是其所有处所状态的总合。

5) 转换的发生由前一个处所和紧接着的处所的状态决定,其影响限于对这两个处所的状态产生作用。

因此,一个 Petri 网完全由下列各项确定:

1) 处所集合 P (每个处所有一个标识符,常以大写字母表示)。

2) 转换集合 T (每个转换有一个标识符,常以小写字母表示)。

3) 有向弧线集合 F 。

4) 权重函数 W (对网络的每个弧线赋以一个权重,权重为一正整数,它在发生转换中参与弧线到达或离开)。

5) 初始标志函数 M_0 (它对网络中的每个处所赋予一个初始状态,该标志是一个大于或等于 0 的整数,它在发生转换中跟着该处所作用;用图形表示的话,标志是处所内部的一个黑点,又称令牌)。

根据 Petri 网定义的行为规则,不能直接用 Petri 网来描述控制行为。即使是经过改进的预测/转换 Petri 网、有色 Petri 网、临时化 Petri 网等,与简单的 Petri 网比较,它们具有更强的表达能力,但也不适合于在总体上构建 SFC 方法所能达到的模型化。其关键是它们本质上做不到 SFC 方法的基本性能,

Petri 网及其改进方法仍存在下列问题:

1) 行为的不确定性。Petri 网在每一步能够决定转换可能发生,但不能确定实际上转换是否发生,在何时发生。

2) 缺乏定义起始步的原语。Petri 网没有起始处所,只有边界转换即没有前置处所的转换。

3) 处所作为目标的被动储存库存在。

4) 需要用流动的目标对控制逻辑模型化。

Petri 网主要通过仿真其行为来研究系统及其关键方面,因此,难以用于控制编程。

SFC 是按照自动化控制的要求,对 Petri 网方法加以适当简化和改进的一种程序设计方法。它有严格的数学基础,因此严密精确,但又简单易学。描述元素只有步、转换和有向连线等 3 种。描述方法直观,十分便于程序设计人员和其他专业人员在设计阶段进行沟通和交流,特别适合用户直接按工艺流程编制控制程序的应用。即使由不同的编程人员编制,其程序的差异也很少,因此,提高了编程的正确性和程序的可读性。SFC 表达控制内容直观,程序与实际工序一一对应,在控制工序发生异常时容易确认问题的所在,因而修改与维护均很方便。

总之,SFC 语言具有结构性的属性,其自顶向下的设计方法,以及可将步用作宏步,十分便于在初步设计时作为结构化的工具。进入详细设计阶段,根据已明确的控制步骤和要求细节,再将宏步扩展为每个具体的步,列出转换条件和与每一步相关联的动作,控制程序就在这样有条不紊的氛围中完成。PLC 软件开发生命周期的初始关键阶段,SFC 最适合于支持软件开发的,而其他 4 种语言,当然还有 SFC 语言本身,都适用于写程序代码的阶段。如何应用,则视具体情况而定。

对相当复杂的控制问题,可借助于有限状态自动机的方法,画出状态图,根据状态图与 SFC 语义的不同,将状态图转换为 SFC 程序。

目前,编程人员使用 SFC,仅把它作为一种编制顺序控制程序的工具。在讲解 IEC 61131-3 编程语言国际标准时,也没有突出 SFC 的结构化特性,因此,人们普遍对 SFC 的认识停留在表面。为此,本节比较 SFC 与状态图、SFC 与 Petri 网的关系和异同,希望促进和深化 SFC 的认识,有目的和方向明确地改善当前 PLC 程序的开发实践,保证程序开发的质量、提高程序开发的效率。本书最后用两个实际应用示例说明如何进行 SFC 设计。

5.6 顺序功能表图程序的转换

5.6.1 顺序功能表图程序中基本序列的转换

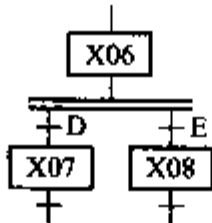
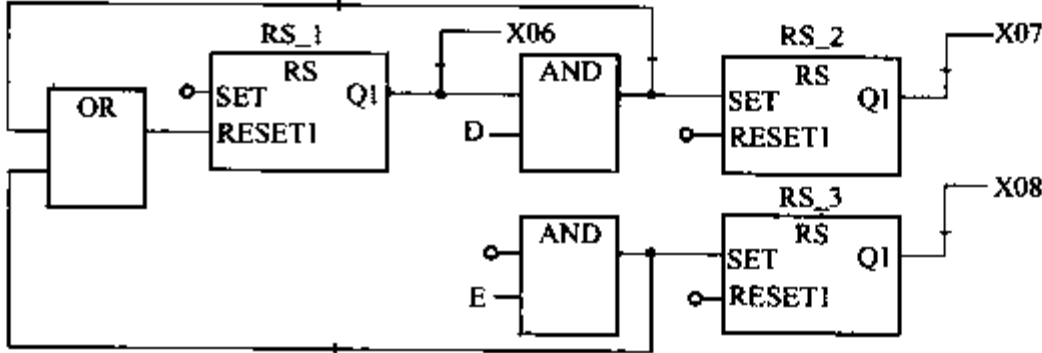
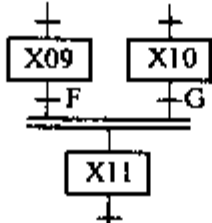
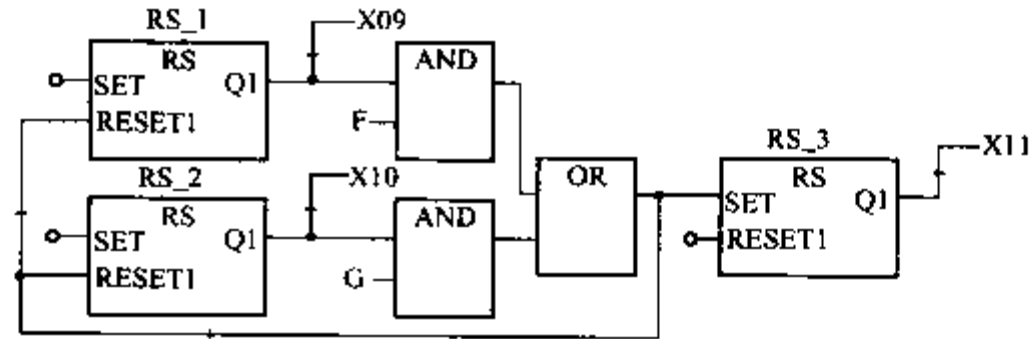
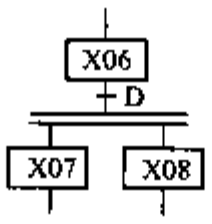
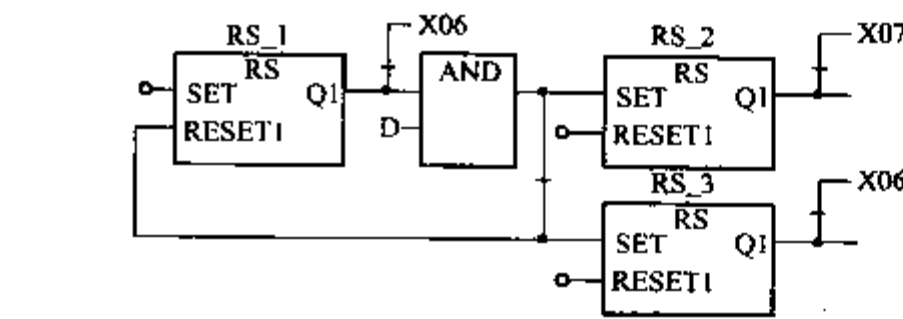
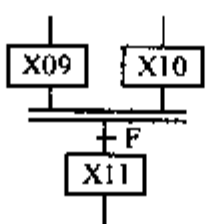
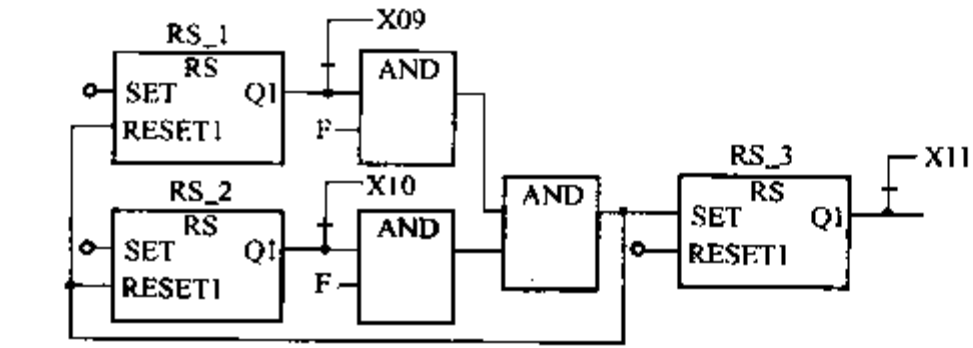
顺序功能表图编程语言具有缩短扫描时间、易于理解过程进展和易于相互思想的沟通等特点,对于不能提供顺序功能表图编程语言的可编程控制器,常采用与顺序功能表图相类似的编程方法实现基本序列的转换。

1. 采用具有保持功能的 RS 功能块

顺序功能表图的基本序列有单序列、选择序列和并行序列等。表 5 - 14 显示了功能表图基本序列转换为功能块图编程语言的方法。选用 RS 功能块的原因是使步的复位比步的置位具有更高的优先级,保证在转换条件满足时能够实现转换。

表 5 - 14 顺序功能表图基本序列转换为功能块图编程语言的程序

功能表图		功能块图编程语言的程序	
单序列			

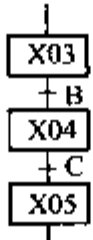
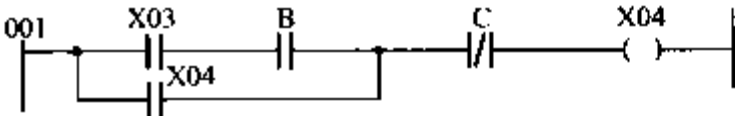
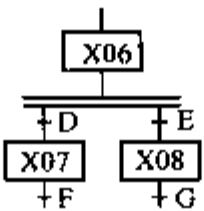
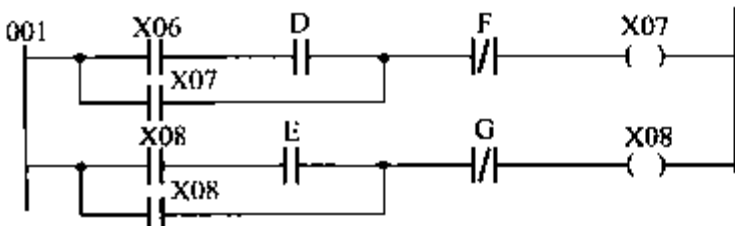
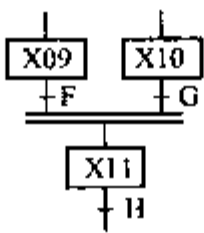
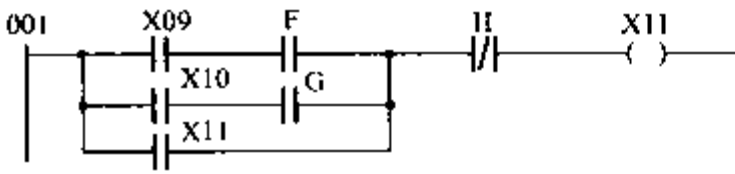
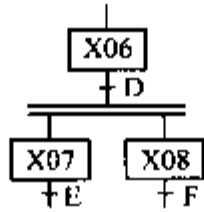
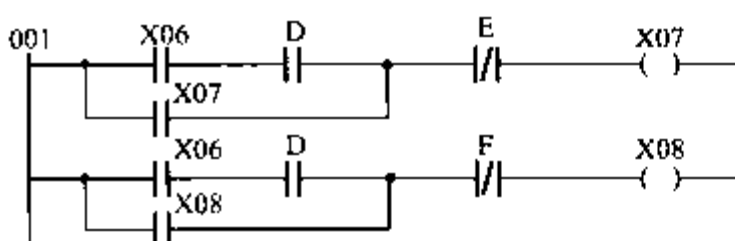
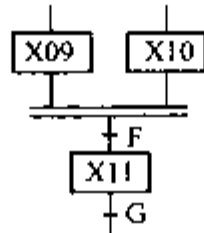

	功能表图	功能块图编程语言的程序
选择序列 (分支)		
选择序列 (合并)		
并行序列 (分支)		
并行序列 (合并)		

注:并行序列的合并功能块可将3个与函数合并为1个与函数,表中用3个与函数表示。

2. 用梯形图程序实现基本序列的转换

顺序功能表图的基本序列转换为梯形图程序时,有多种实现方法。表5-15显示了其中一种转换方法。

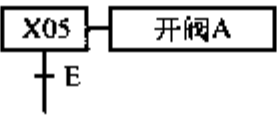
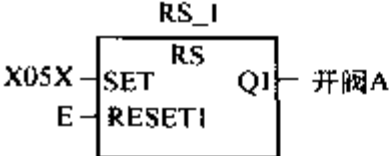
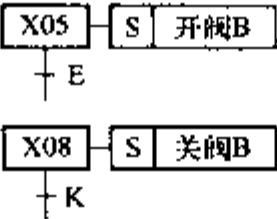
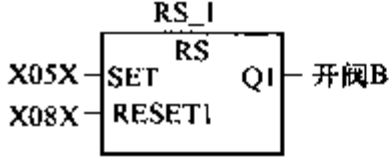
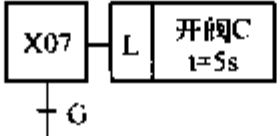
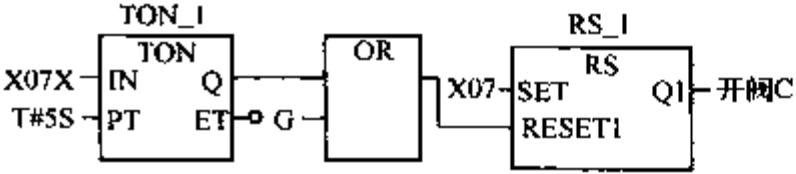
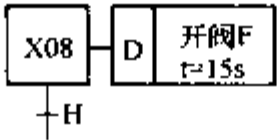
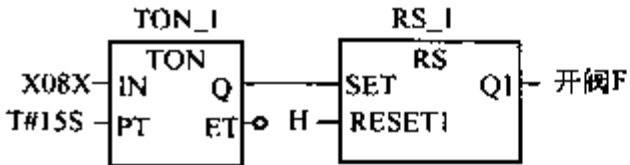
表 5-15 基本序列的转换

	功能表图	梯形图编程语言的程序
单序列		
选择序列 (分支)		
选择序列 (合并)		
并行序列 (分支)		
并行序列 (合并)		

5.6.2 顺序功能表图程序中动作控制功能块的转换

顺序功能表图程序中的动作控制功能块有不同类型,例如,存储型、时限型等。对这些动作控制功能块可采用图 5-17、图 5-18 规定的功能块实现转换。表 5-16 是动作控制功能块转换为功能块图程序的示例

表 5 - 16 顺序功能表图中动作控制功能块的转换示例

功能表图		梯形图编程语言的程序	
非存储型			
存储型			
时限型			
延迟型			

第6章 基本应用

6.1 电动机控制的编程

电动机控制在顺序控制中占重要位置,熟悉电动机控制的编程十分重要。

6.1.1 电动机起保停控制

电动机的控制有多种形式,编制的程序也各不相同。这里,列举部分电动机控制程序的编程。

1. 使用两个按钮实现电动机起保停控制

使用两个按钮控制电动机的起保停具有结构简单、易于编程的特点。根据起动优先或停止优先的不同,有两种不同的实现方法。

(1) 停止优先的电动机起保停控制

1) 图形类编程语言编程。采用双稳元素功能块 RS 可实现停止优先的电动机起保停控制。程序如图 6-1a 所示,也可用图 6-1b 所示梯形图程序实现。

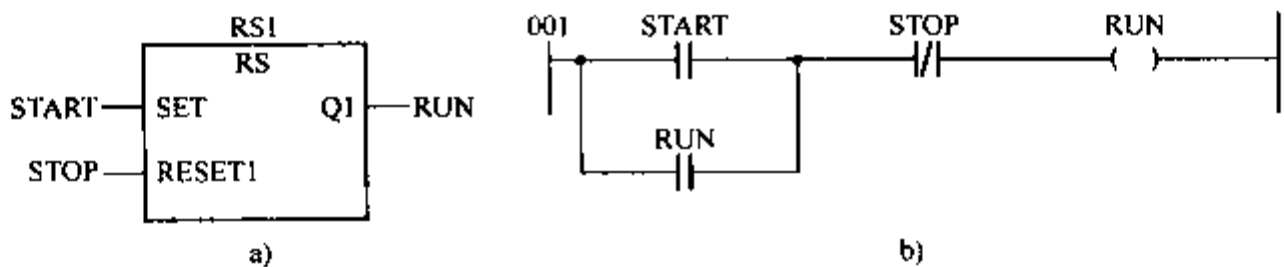


图 6-1 停止优先的起保停控制
a) 用功能块实现 b) 用梯形图实现

2) 文本类编程语言编程。用指令表编程语言编写的程序如下:

```
LD      START      (* 读取 START 状态 *)
OR      RUN         (* 与 RUN 进行或逻辑运算 *)
ANDN    STOP        (* 运算结果再与 STOP 取反的信号进行与逻辑运算 *)
ST      RUN         (* 最终结果存放在 RUN *)
```

用结构化文本编程语言编写的程序如下:

```
RUN := RS1 (SET := START, RESET1 := STOP);
```

(2) 起动优先的电动机起保停控制

一些紧急联锁用的设备,例如,消防水泵、紧急通风风机的起停控制采用起动优先的控制方式。

1) 图形类编程语言编程。采用双稳元素功能块 SR,起动优先的电动机起保停控制的功

能块图和梯形图编程语言编写的程序见图 6-2。

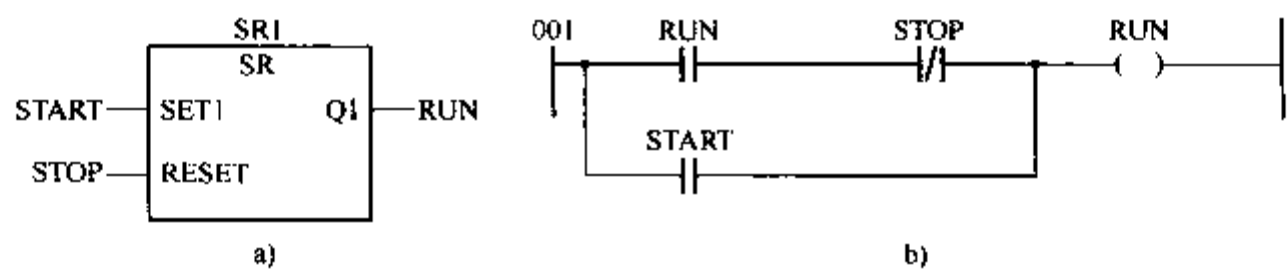


图 6-2 起动优先的起保停控制
a) 用功能块图实现 b) 用梯形图实现

2) 文本类编程语言编程。用指令表编程语言编写的程序如下：

```
LD      RUN      ( * 读取 RUN 状态 * )
ANDN    STOP     ( * 与 STOP 取反的信号进行与逻辑运算 * )
OR      START    ( * 运算结果再与 START 进行或逻辑运算 * )
ST      RUN      ( * 最终结果存放在 RUN * )
```

用结构化文本编程语言编写的程序如下：

```
RUN := SRI ( SET1 := START, RESET := STOP );
```

(3) 应用注意事项

- 1) 用功能块图和结构化文本编程语言编程时,START 和 STOP 采用常开触点。用梯形图和指令表编程语言编程时,START 是常开触点,STOP 是常闭触点。
- 2) 标准功能块中,停止优先控制规定用 S 和 R1 端表示置位端和复位端。为清楚表示其特性,一些软件采用 SET 和 RESET1 表示。同样,起动优先控制规定用 S1 和 R 端表示置位端和复位端。为清楚表示其特性,一些软件采用 SET1 和 RESET 表示。
- 3) 停止优先表示当同时按下起动(置位)和停止(复位)按钮时,输出 RUN 端为 0,即电动机停止。在图形上,输入端用 1 表示优先级高的端子,即 R1 或 RESET1 端的优先级高于 S 或 SET 端。起动优先表示当同时按下起动(置位)和停止(复位)按钮时,输出 RUN 端为 1,即电动机起动。
- 4) 大多数电动机和运转设备采用停止优先的起保停控制方式来控制其起停。

2. 使用一个按钮实现电动机起保停控制

使用一个按钮实现电动机起保停控制也称为单按钮起保停控制。常用于可编程控制器的输入点数较少,需要控制起停的设备较多的场合。

单按钮起保停控制的控制要求是当第一次按下起停按钮 START 时,电动机起动并保持,当第二次按下起停按钮 START 时,电动机停止并保持。有多种程序可实现所需控制要求。

(1) 上升沿触发的单按钮起保停控制

用按下 START 按钮时产生的脉冲信号实现起停控制,称为上升沿触发的单按钮起保停控制。图 6-3 是上升沿触发的单按钮起保停控制程序。

操作过程如下:当第一次按下 START 按钮时,经上升沿边沿检测功能块 R1 的检测,输出 L 有一个脉冲信号,在 003 梯级,使电动机 RUN 激励和运转,并自保。当第二次按下 START 按

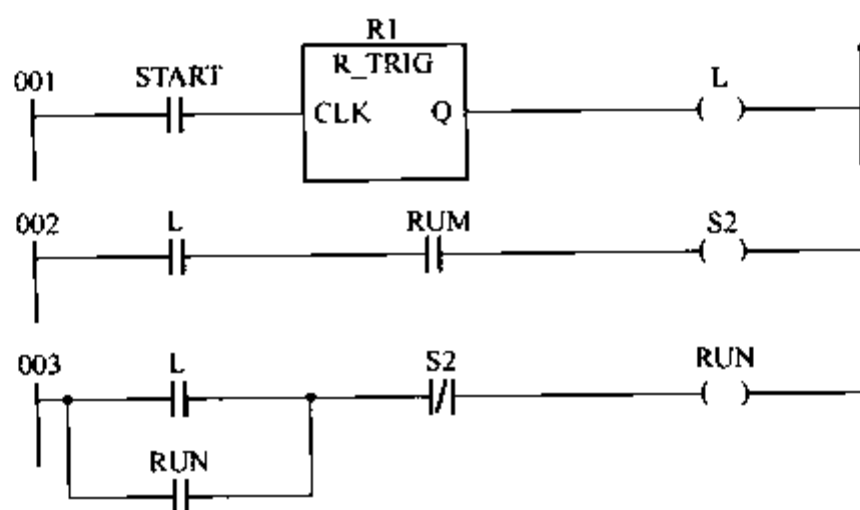


图 6-3 上升沿触发的单按钮起停控制

钮时,同样有一个脉冲信号 L 输出,因 RUN 自保,因此,在 002 梯级,使中间变量 S2 激励,从而在 003 梯级使输出 RUN 失励,即电动机停止运转。

用文本类编程语言的指令表编程语言编程如下:

```

LD    START
ST    R1. CLK
CAL   R1
LD    R1. Q
ST    L
LD    L
AND   RUN
ST    S2
LD    L
OR    RUN
ANDN  S2
ST    RUN

```

采用计数器 C1 对按钮按动的次数进行计数,也可实现单按钮起保停控制。图 6-4 是采用计数器实现单按钮起保停控制的程序。当第一次按下 START 按钮时,经上升沿边沿检测功能块 R1 的检测,输出 L 有一个脉冲信号,它作为计数器的输入脉冲,用于计数,计数设定值 PV 为 2,同时,脉冲信号 L 连接到 R1 双稳元素功能块的置位端,使其输出 RUN 激励。当第二次按下 START 按钮时,计数器 C1 计数到,它的输出 C1. Q 为 1。使 R1 的复位端置 1,因此,其输出 RUN 失励,表示电动机停止运转。

用文本类编程语言的指令表编程语言编程如下:

```

LD    START
ST    R1. CLK
CAL   R1
LD    R1. Q
ST    L
LD    L
ST    C1. CU

```

```

LD    C1. Q
ST    C1. RESET
LD    INT#2
ST    C1. PV
CAL   C1
LD    C1. Q
ST    C1Q
LD    L
ST    RS1. SET
LD    C1. Q
ST    RS1. RESET1
CAL   RS1
LD    RS1. Q1
ST    RUN

```

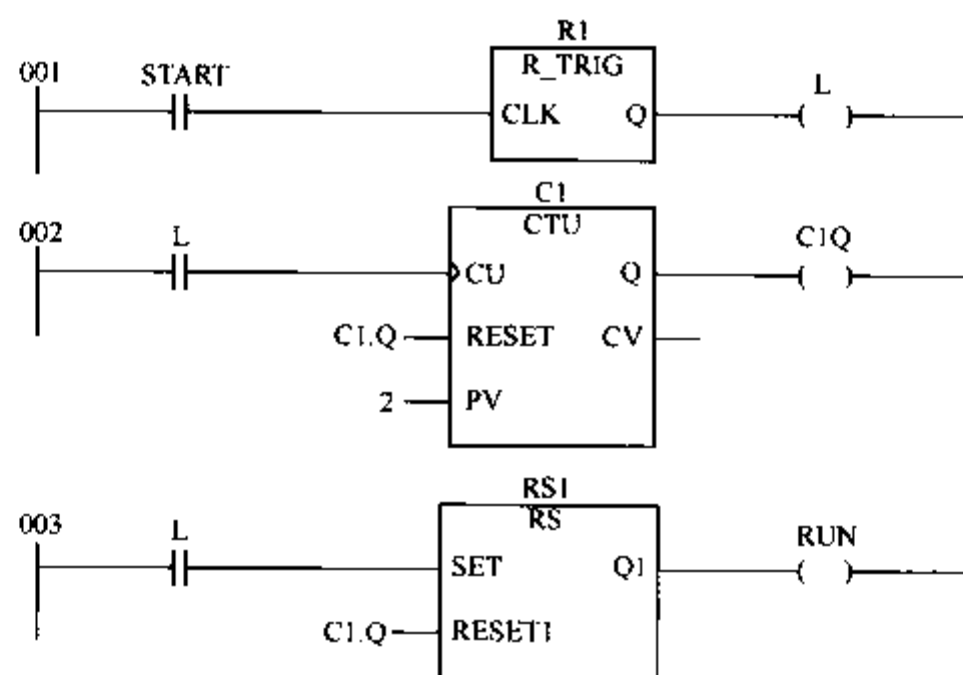


图 6-4 采用计数器实现单按钮起停控制的程序

(2) 下降沿触发的单按钮起保停控制

当上述程序的上升沿边沿检测功能块用下降沿边沿检测功能块替代时,可实现下降沿触发的单按钮起停控制。图 6-5 是常见的一种程序。

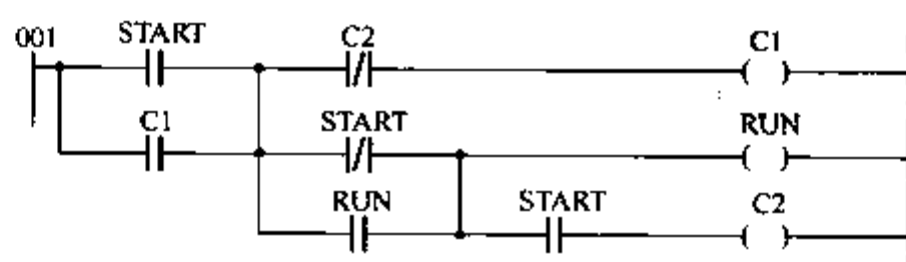


图 6-5 下降沿触发的单按钮起停控制程序

当第一次按下 START 按钮时,中间变量 C1 激励,并自保,但因串接在 RUN 梯级的是 START 的反相信号,因此,RUN 未激励,同样,C2 也未激励。当按钮释放时,START 取反信号

为 1,因此,RUN 激励,电动机运转。第二次按下 START 时,因 START、RUN 的线路仍接通,因此,RUN 保持激励状态,同时,C2 也激励。按钮 START 释放时,才使 RUN 失励,实现停止电动机运转的控制要求。

用文本类编程语言的指令表编程语言编程如下:

```
LD      START
OR      C1
ST      I_1
LD      I_1
AND     C2
ST      C1
LDN     START
OR      RUN
AND     I_1
ST      RUN
AND     START
ST      C2
```

程序中,用 I_1 表示中间变量,它存储 START 与 C1 的或逻辑运算结果。

6.1.2 电动机的正、反转控制

电动机的正反转切换可通过更换电动机的一组相线实现。电气控制线路中常用两个接触器 KM1 和 KM2 实现。如图 6-6 所示。图中,SB 2 和 SB3 分别是正转和反转的起动按钮,SB1 是停止按钮。

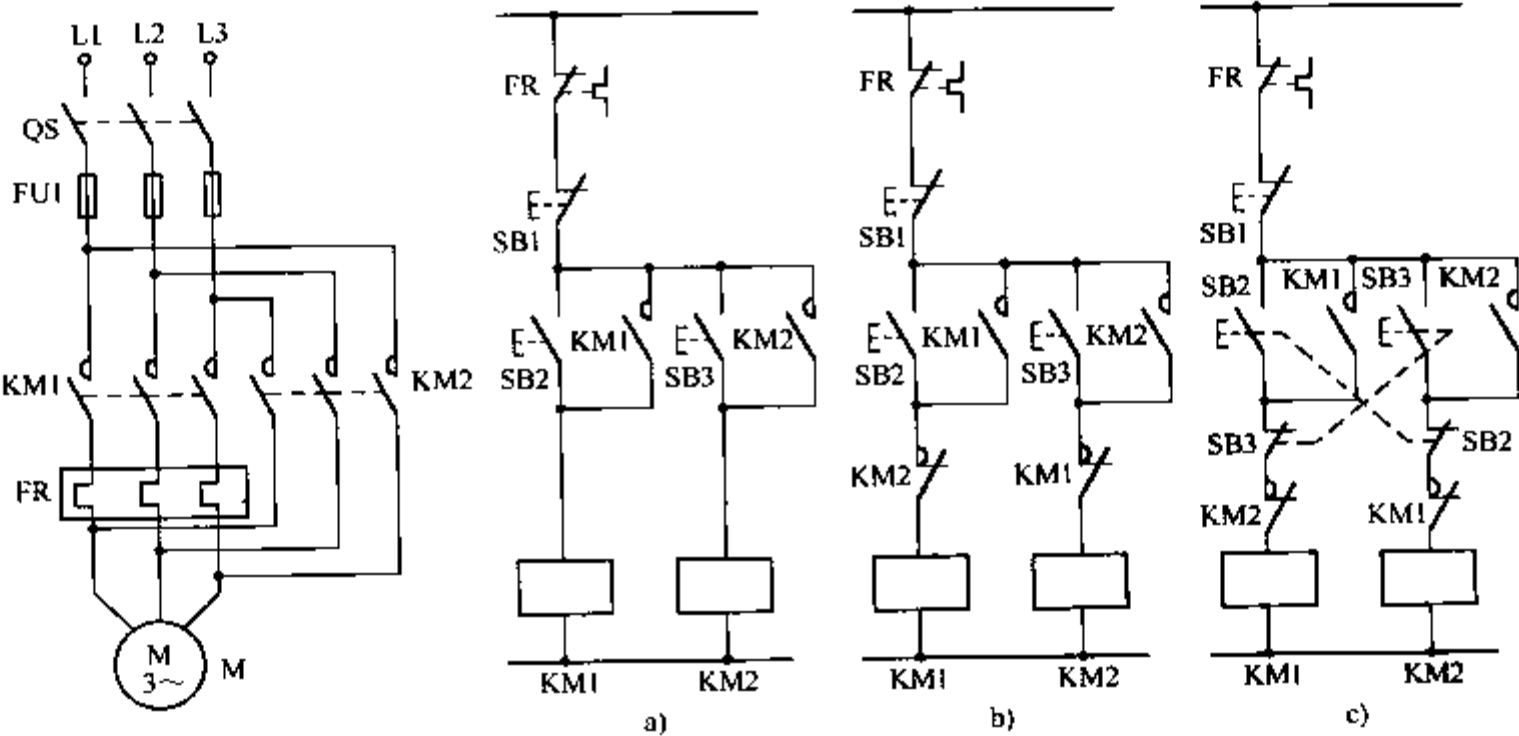


图 6-6 电动机正反转控制电路
a) 无互锁控制 b) 正-停-反转控制 c) 正-反-停控制

有两种实现电动机正、反转控制的方法。一种方法是电动机正、反转切换不经过停止的过程。另一种方法是电动机先停止,然后,根据切换要求切换到所需的运转方式。

1. 经停止过程的电动机正、反转切换控制

与图 6-6 对应,可采用两个起动按钮 ZQ 和 FQ,用于正转和反转的起动,一个按钮 TZ 用于停止运转。正转和反转的接触器用 ZZ 和 FZ 表示。程序如图 6-7 所示。

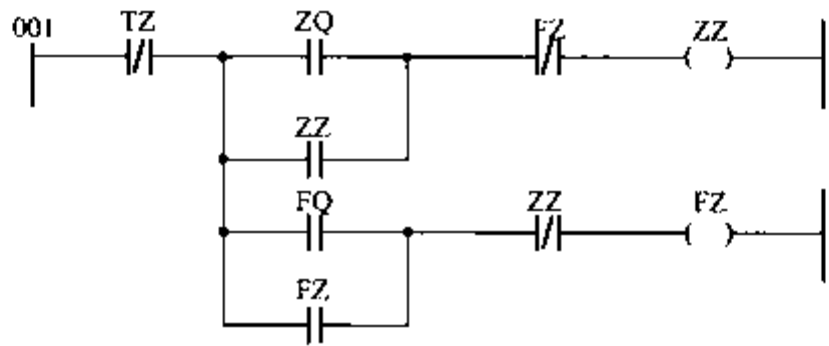


图 6-7 经停止过程的电动机正、反转切换控制程序

电动机在停运状态时,ZZ 和 FZ 都未激励。

当按下 ZQ 正转启动按钮时,输出 ZZ 激励,表示电动机正转。同时,它将反转控制线路切断,防止反转起动。只有按下 TZ 停止按钮,电动机才能停转。

当按下 FQ 反转启动按钮时,输出 FZ 激励,表示电动机反转。同时,它将正转控制线路切断,防止正转起动。只有按下 TZ 停止按钮,电动机才能停转。

这类程序和下面的程序可类似地用于抢答器类的应用,如第 4.2.3 节所示。

用功能块图编程语言编写的程序如图 6-8 所示。

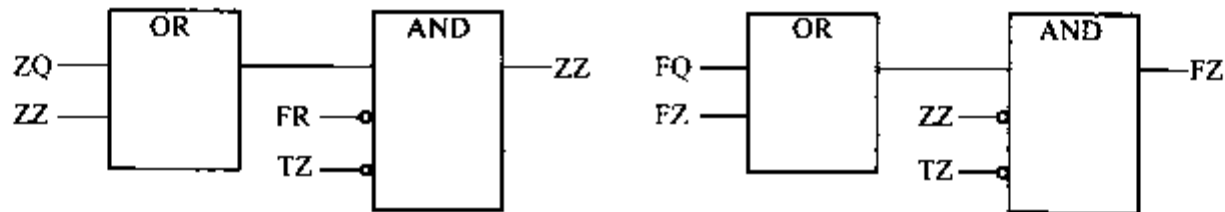


图 6-8 经停止过程的电动机正、反转切换的功能块图程序

2. 不经停止过程的电动机正、反转切换控制

与图 6-6 对应,采用两个起动按钮 ZQ 和 FQ 用于正转和反转的起动,一个按钮 TZ 用于停止运转。正转和反转的接触器为 ZZ 和 FZ。不经停止过程的电动机正、反转切换控制程序如图 6-9 所示。

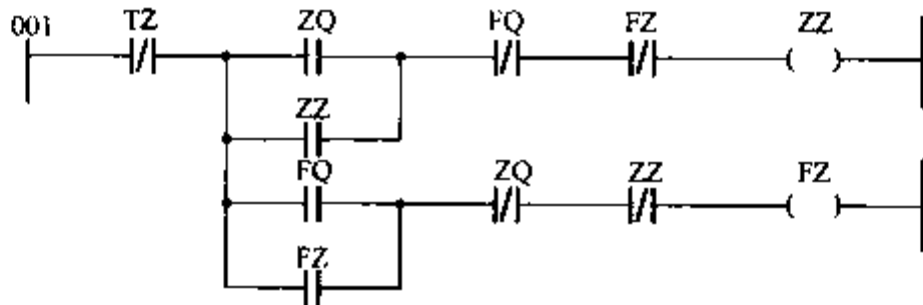


图 6-9 不经停止过程的电动机正、反转切换控制程序

与经停止过程的电动机正、反转切换控制程序类似,只在各梯级上增加如图 6-9 所示的

一个常闭触点,它用于停止电动机的运转。

当按下 ZQ 正转起动按钮时,输出 ZZ 激励,表示电动机正转。同时,它断开反转控制线路,使反转接触器触点断开。

当按下 FQ 反转起动按钮时,输出 FZ 激励,表示电动机反转。同时,它断开正转控制线路,使正转接触器触点断开。

按下 TZ 停止按钮,电动机停转。

为防止因接触器动作延迟可能发生相间短路。程序中除了用接触器触点进行互锁外,还用起动按钮反相触点断开另一控制回路,实现互锁。同时,在外部接线时采用图 6-10 所示互锁接线方式。

有时,也将电动机热继电器的常闭触点串联到各梯级中,实现当负载过大,热继电器激励时,能够及时使接触器输入信号断开。但实际应用时,常在外接线时串接接入热继电器触点,用于减少可编程控制器的输入点数,并用 KM1、KM2 的硬件互锁,如图 6-10 所示。

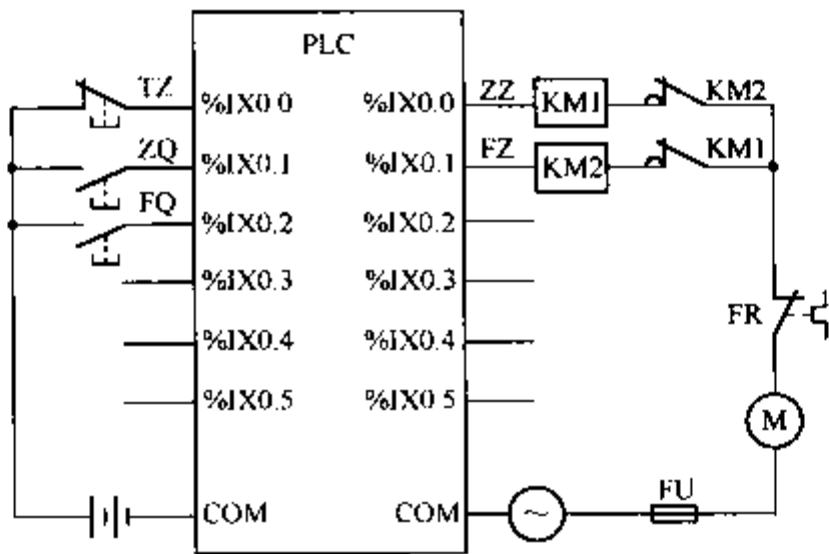


图 6-10 可编程控制器的外部接线图

6.1.3 电动机的星-三角转换控制

电动机的星-三角转换控制用于三相异步电动机的减压起动,即起动电动机时,采用星形连接,电动机运转后自动切换到三角形连接。

设电动机的 3 个绕组分别是 AX、BY 和 CZ,则星形连接是将 X、Y 和 Z 连接在一起,A、B 和 C 连接到三相输入端,三角形连接是将 A 与 Z、B 与 X、C 与 Y 连接在一起,并将 A、B 和 C 连接到三相输入端。

当输出接触器 KM_Y 激励, KM_Y 的各触点闭合,组成星形连接,当输出接触器 KM_Δ 激励, KM_Δ 的各触点闭合,组成三角形连接。当输出继电器 KM 激励,起动电动机。假设星形连接起动 5 s 后自动切换到三角形连接。电动机的星-三角转换控制程序见图 6-11。

程序转换过程如下:按下起动按钮 START 后,K1 激励并自保,同时使连接的接触器 KM 激励,相应触点闭合,003 梯级中,K2 激励,它使连接的接触器 KM_Y 激励,组成星形连接结构,并使电动机在低起动电流下运转,当延时时间 5 s 到,定时器 T1 的输出 T1.Q 变为 1,使 K2 失励,K3 激励,它使相应的接触器 KM_Y 失励,接触器 KM_Δ 激励,电动机组成三角形连接结构,并用三角形连接运转。按下停止按钮后,K1 失励,电动机停转。

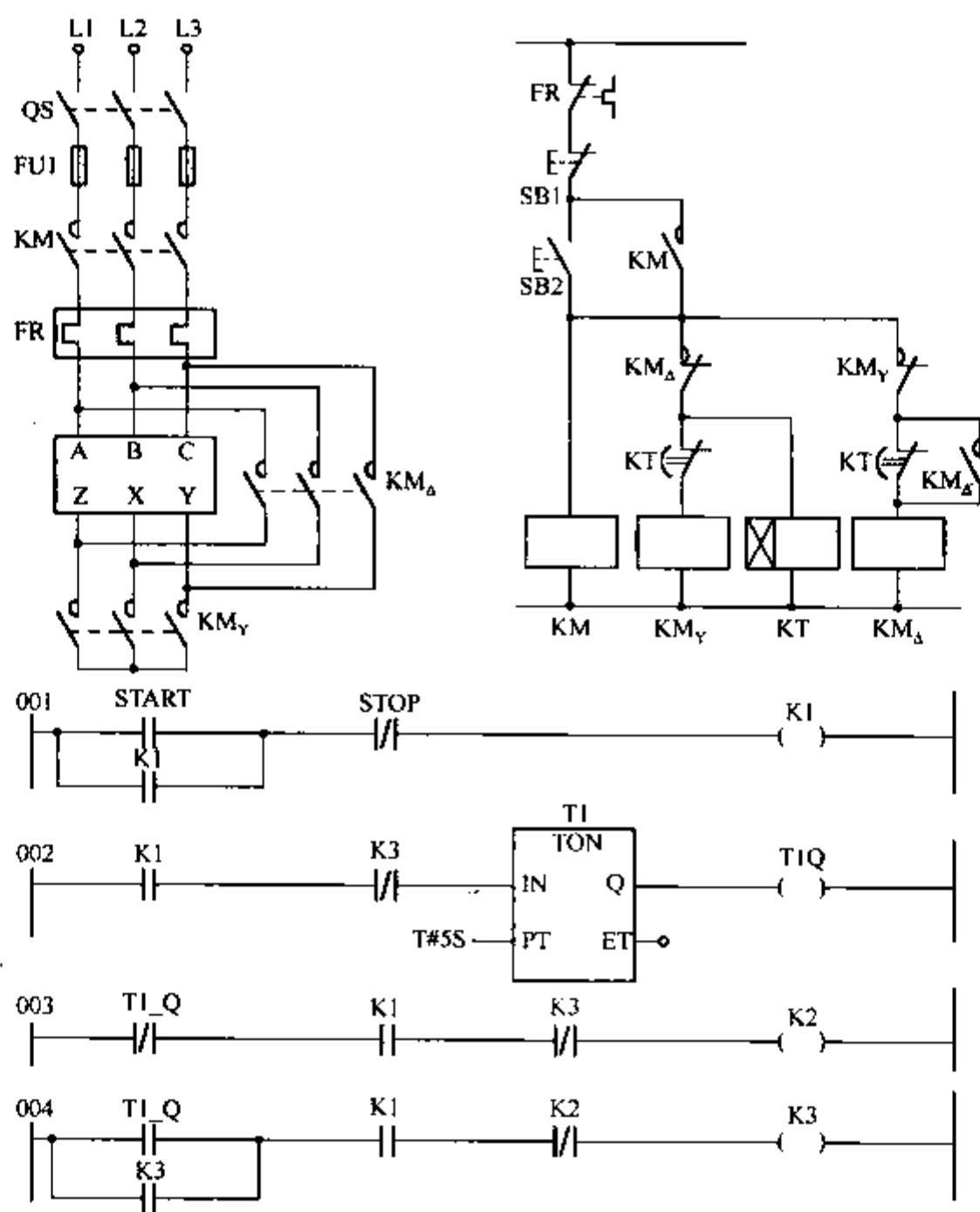


图 6-11 电动机星-三角转换控制电路和程序

程序中,星形连接和三角形连接的两个输出继电器已实现互锁,为防止转换过程中造成相间短路,硬件接线可用类似图 6-9 的形式实现硬件互锁。

星-三角起动的优点是星形起动电流只有原来三角形起动电流的三分之一。起动电流约为电动机额定电流的 2 倍。因此,起动特性好、结构简单、价格低。星形起动的缺点是起动转矩也下降到三角形直接起动转矩的三分之一,因此,其转矩特性差。

考虑到星形连接转换到三角形连接过程中可能造成相间短路,可在星形连接断开后,再延时一段较短时间,例如,0.5 s 后才转换到三角形连接。程序如图 6-12 所示。

程序增加一个定时器功能块,保证在星形连接断开信号发出后能够延时 0.5 s 才转换到三角形连接。如果能够直接采用星形连接的断开信号,则系统的转换会更可靠。

星-三角转换程序可用于各种需要转换的控制场合。一般用于电动机空载或轻载起动的场合。转换部分分别由 K2 和 K3 控制,而 K1 作为总起动信号使用,转换条件可以是延时信号或其他信号。工业过程中,通常采用星-三角起动器或软起动器实现。其工作原理与上述原理类似。

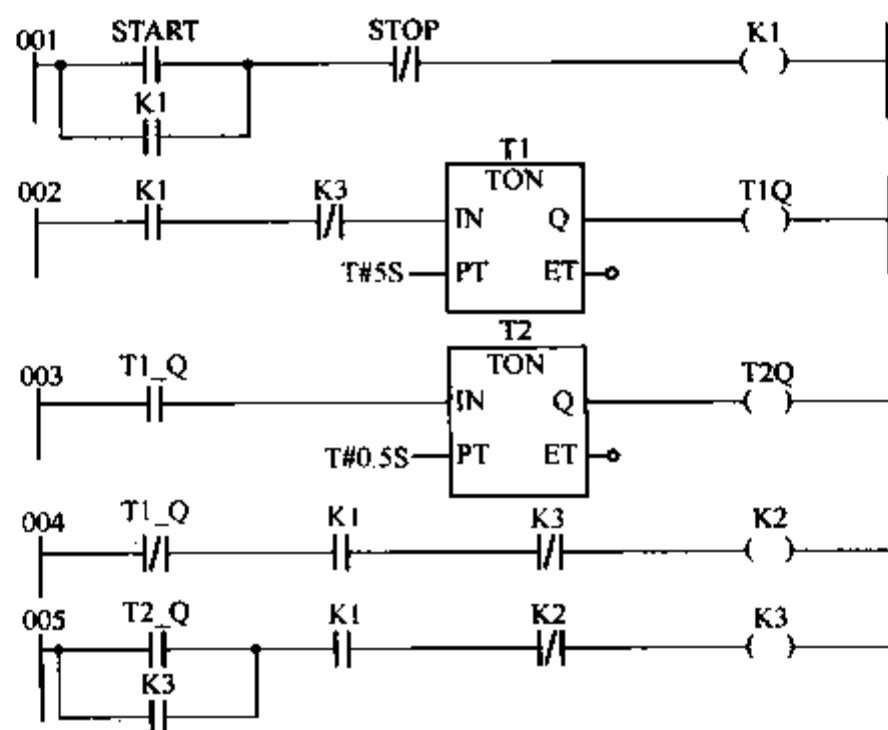


图 6-12 有延时的电动机星形 - 三角形转换程序

6.1.4 三相步进电动机的相序控制

三相步进电动机的相序控制包括转速控制、正反转控制、步数控制等。

1. 步进电动机的转速控制

三相步进电动机的转速控制分若干挡,某三相步进电动机的转速控制分 3 挡,即慢速(每 1 s 前进 1 步)、中速(每 0.5 s 前进 1 步)、高速(每 0.25 s 前进 1 步)。用 3 个开关 S1、S2 和 S3 控制,S1 闭合步进电动机转速为每秒前进一步,S2 开关闭合时,步进电动机转速为每秒前进 2 步,S3 开关闭合时,步进电动机转速为每秒前进 4 步。

为产生上述脉冲触发信号,需编制脉冲信号发生器程序。脉冲信号发生器可直接用定时器实现,如图 6-13 所示。

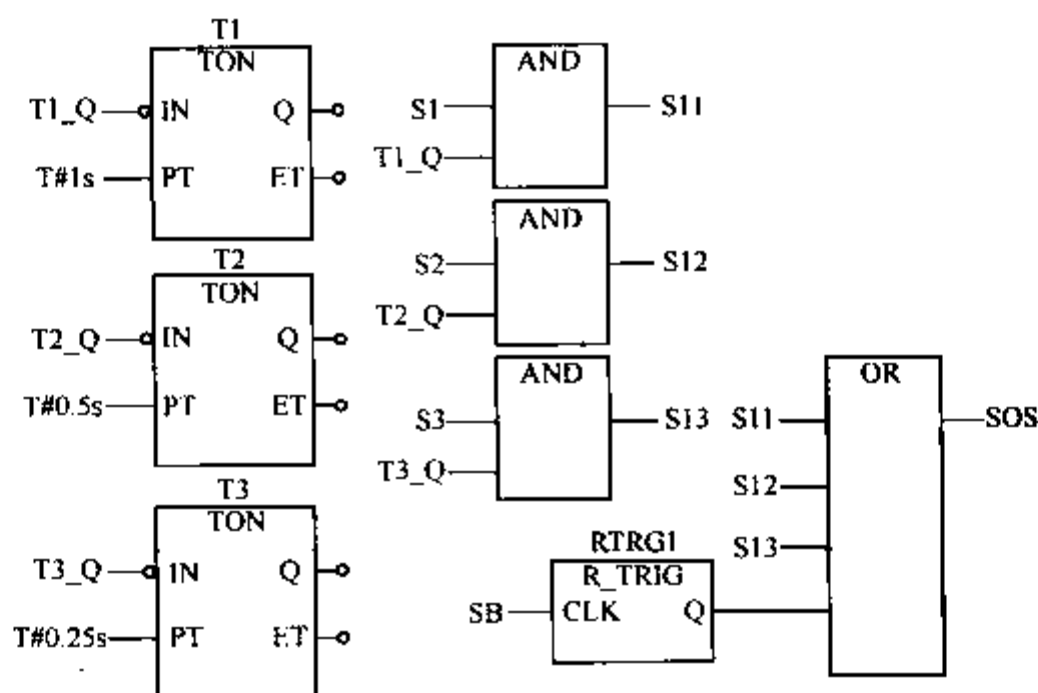


图 6-13 转速控制的脉冲信号发生器程序

图中,定时器 T1 功能块用于产生脉冲信号,其周期是 1 s,类似地,定时器 T2 和 T3 用于产生脉冲周期分别为 0.5 s 和 0.25 s 的脉冲信号。用 3 个与函数实现与各转速开关的与逻辑运算。因此,慢速开关 S1 闭合时,S11 有周期为 1 s 的脉冲信号输出;中速开关 S2 闭合时,S12 有周期为 0.5 s 的脉冲信号输出;高速开关 S3 闭合时,S13 有周期为 0.25 s 的脉冲信号输出。这些输出脉冲用于作为步进电动机的脉冲输入信号。

SB 是单步按钮信号,当操作人员希望步进电动机在步进过程中执行单步运行时,按下 SB 按钮,步进电动机在原有脉冲触发过程中添加一个单步。用上升沿检测功能块 RTRG1 检测按钮信号 SB,保证不管按钮按下时间的长短都只对步进电动机前进一步。脉冲信号输出是 SOS。

2. 步进电动机的步数控制

步进电动机的步数控制是切换到某种步数控制开关后,步进电动机在步进了规定的步数后自动停止的一类控制方法。示例采用两种步数控制,分别是 10 步和 100 步。当 10 步控制开关 S6 闭合时,步进电动机在步进 10 步后自动停止。同样,当 100 步控制开关 S7 闭合时,步进电动机在步进 100 步后自动停止。为每个步数开关设置一个计数器,用于对步数进行计数。示例中,采用两个计数器 C2 和 C3,分别设置计数设定值为 10 和 100,用其输出 C2. Q 和 C3. Q 和脉冲信号 SOS 进行与运算实现对步进信号的禁止。程序如图 6-14 所示。

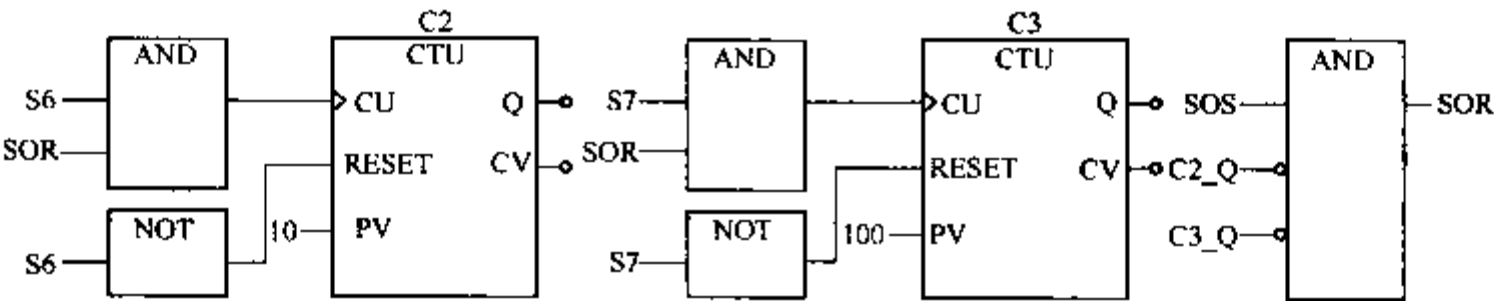


图 6-14 步进电动机步数控制程序

3. 步进电动机的三相六拍环行分配器控制程序

三相六拍环行分配器可以用硬件实现,也可用软件实现,示例采用软件实现三相六拍环行分配器的功能。表 6-1 是输出的三相和六拍之间的关系表。为实现上述控制要求,编制如图 6-15 所示程序。

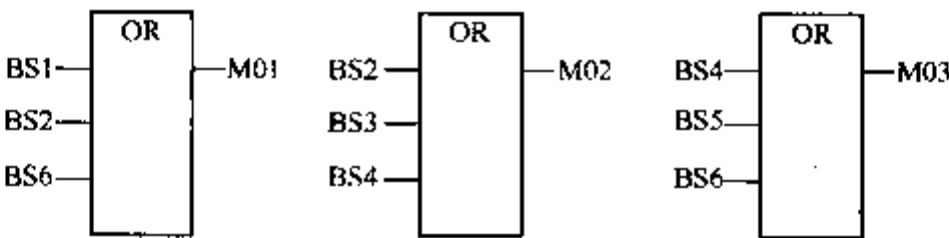


图 6-15 三相六拍环行分配器程序

表 6-1 三相输出与六拍之间的关系

	BS1	BS2	BS3	BS4	BS5	BS6
M01	1					
		1				
						1

(续)

	BS1	BS2	BS3	BS4	BS5	BS6
M02		1				
			1			
				1		
M03				1		
					1	
						1

4. 步进电动机的正反转控制

三相步进电动机的正反转控制与一般电动机的正反转控制类似。一般电动机的正反转控制采用调换相序的方法实现。步进电动机的正、反转通电顺序如图 6-16 所示。

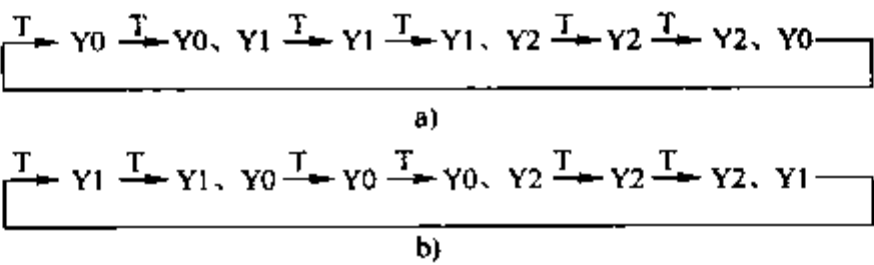


图 6-16 步进电动机正反转的通电顺序图

a) 步进电动机正转的通电顺序 b) 步进电动机反转的通电顺序

从通电顺序图可见,只需要将 Y0 和 Y1 调换相序就可实现正转和反转控制。用 S4 开关实现正反转控制。当 S4 开关断开时,M01 连接到 Y0,M02 连接到 Y1,M03 连接到 Y2,表示步进电动机正转。反之,当 S4 开关闭合时,M01 连接到 Y1,M02 连接到 Y0,M03 仍连接到 Y2,表示步进电动机反转。编制的程序如图 6-17 所示。

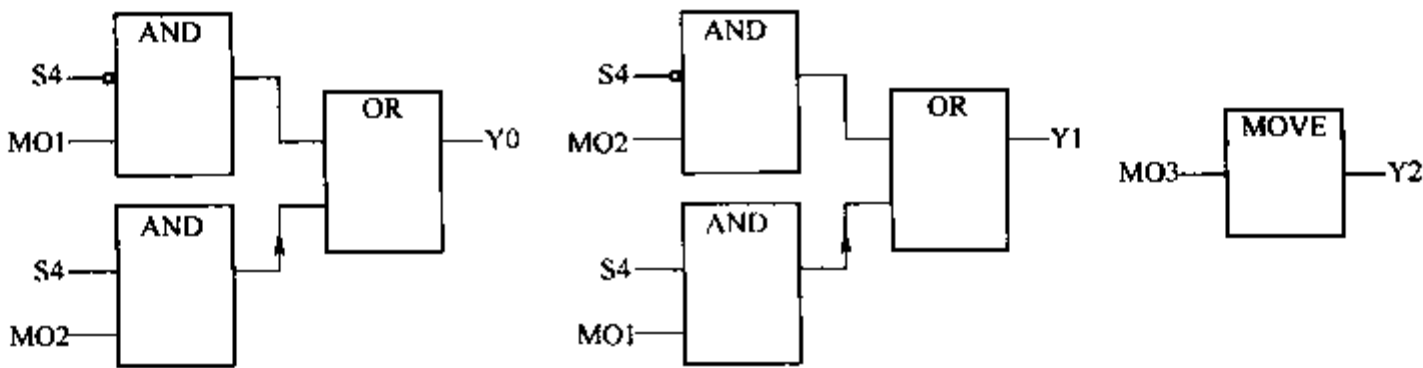
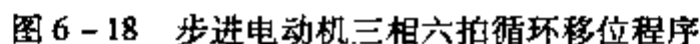


图 6-17 步进电动机正反转控制程序

5. 步进电动机三相六拍移位控制

IEC 61131-3 的标准函数中没有六拍循环移位函数,但提供左移循环移位函数 ROL。为实现六位的移位,可采用计数器 C1 对步进的移位进行计数,当步进移位 6 步后,重新开始从初始位置移位。图 6-18 是步进电动机三相六拍移位控制程序。



其他规定步数的循环移位也可采用上述程序,只需要改变其计数器的计数设定值。须注意,当移位个数大于 8 位时,程序中 SOR 数据类型应从字节改变为字或双字。

除星-三角转换降压起动外,还可串接电阻降低绕线式异步电动机电流的方式起动。此外,为使电动机快速制动,可采用将电动机的两相反接实现电动机的快速停运。

1. 电动机正转起动过程

242

其触点使接触器 KM3 激励,将起动电阻 R 短路,电动机转速上升到正常转速运转。

2. 电动机停止过程

按下停止按钮 SB1, %IX0.1 为 1,使接触器 KM1 失励,由于电动机转速还高于设定值,因此,瞬时接触继电器 KA1 仍处于激励状态,而接触器 KM1 的失励,使接触器 KM2 激励,电动机处于反接制动状态,电动机转速迅速下降。当转速低于速度开关设定的转速时,接触器 KM3 失励,电动机失电,达到迅速停转目的。

3. 电动机反转起停过程

与正转过程类似,按反转起动按钮 SB3,电动机串接制动电阻实现反转起动,转速达到设定值,KS2 动作,使制动电阻被 KM3 短接,实现反向运转。停止过程与正转停止过程类似,以 KM1 作为反接制动的接触器实现迅速停转目的。

4. 控制程序

输入信号 %IX0.0、%IX0.1 和 %IX0.2 分别连接正转起动按钮 QA1、反转起动按钮 QA2 和停止按钮 TA 信号,输入信号 %IX0.3 和 %IX0.4 分别连接正、反转转速开关 KS1 和 KS2。输出信号 %QX0.0、%QX0.1 和 %QX0.2 分别连接接触器 KM1、KM2 和 KM3,中间变量是 KA1~KA4。用结构化文本编程语言编写的控制程序如下:

```
KA3 := (QA1 OR KA3) AND (NOT TA) AND (NOT QA2) AND (NOT KA4);  
KM1 := ((NOT TA) AND KA3) OR KA2 AND (NOT KM2);  
KA4 := (QA2 OR KA4) AND (NOT TA) AND (NOT QA1) AND (NOT KA3);  
KM2 := ((NOT TA) AND KA4) OR KA1 AND (NOT KM1);  
KA1 := KS1 AND (KM1 OR KA1);  
KA2 := KS2 AND (KM2 OR KA2);  
KM3 := (KA1 AND KA3) OR (KA2 AND KA4);
```

须注意,结构化文本编程语言中,没有指令表编程语言的 ANDN、ORN 等操作符。但因 NOT 操作优先级高于 AND 操作,因此,一般不需要用圆括号表示其优先级操作。本示例使用圆括号的目的是增加程序的可读性。

采用可编程控制器控制电动机运行时,电动机的接线图没有变化,但控制电路由可编程控制器实现。中间继电器由内部软件实现,接触器由可编程控制器输出控制。

6.2 定时器、计数器的编程

使用定时器和计数器,可编写一些与时间和数量有关的应用程序。

6.2.1 电动机的定时运行

电动机的定时运行包括延时起动、延时停止,运行规定时间停止等。它们都需要采用定时器功能块。

1. 电动机延时起动控制

电动机延时起动控制与电动机的停止优先控制类似,但增加定时器用于延时。程序见图 6-20。程序中假设延时时间为 2 s。

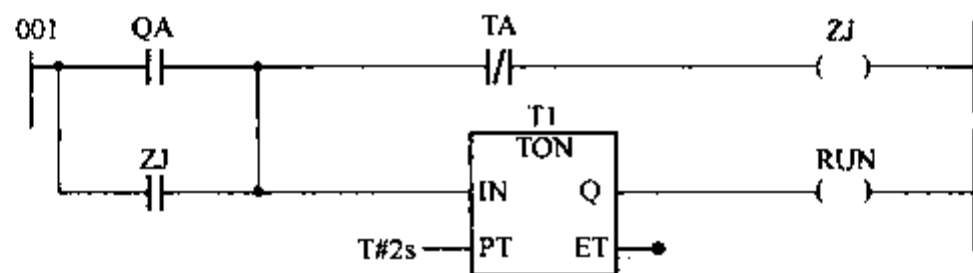


图 6-20 电动机延时起动控制程序

结构化文本编程语言编写的程序如下：

```
ZJ := (QA OR ZJ) AND NOT TA;
T1(IN := (QA OR ZJ), PT := T#2s);
RUN := T1.Q;
```

2. 电动机延时停止控制

电动机延时停止控制与电动机延时起动控制类似,只需要将程序中的延时接通定时器 TON 功能块改用延时断开定时器 TOF 功能块即可。

结构化文本编程语言编写的程序如下：

```
ZJ := (QA OR ZJ) AND NOT TA;
T1(IN := (QA OR ZJ), PT := T#2s);
RUN := T1.Q;
```

须注意,该程序与电动机延时起动程序相同,其区别是变量声明中,本程序中,T1 是 TOF 定时器的实例名,电动机延时起动程序中,T1 是 TON 定时器的实例名。

3. 电动机运行规定时间的控制

电动机起动后运行规定时间后自动停止的控制程序需要一个脉冲定时器 T1,程序与图 6.19 程序类似,只需要将 TON 定时器改为 TP 定时器。

4. 电动机延时起动,运行规定时间后停止的控制

电动机延时起动控制程序和运行规定时间的程序结合,可实现电动机延时起动,运行规定时间停止的控制要求。程序见图 6-21。假设延时时间(由 TON 功能块的设定设置)为 5 s,规定运行时间(由 TP 功能块的设定设置)为 15 s。

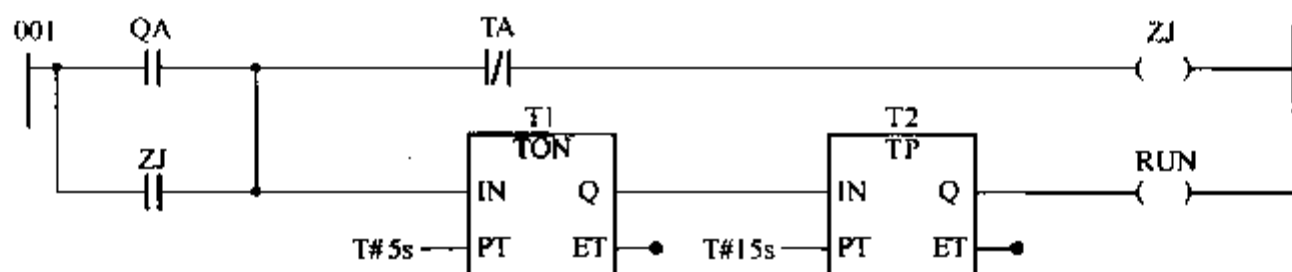


图 6-21 电动机延时起动,运行规定时间后停止的控制程序

5. 电动机延时起动,延时停止的控制

电动机延时起动和电动机延时停止的程序结合,可实现电动机延时起动和延时停止的控制要求。图 6-22 是控制程序。假设延时起动时间为 5 s,延时停止时间(由 TOF 功能块的设定设置)是 15 s。

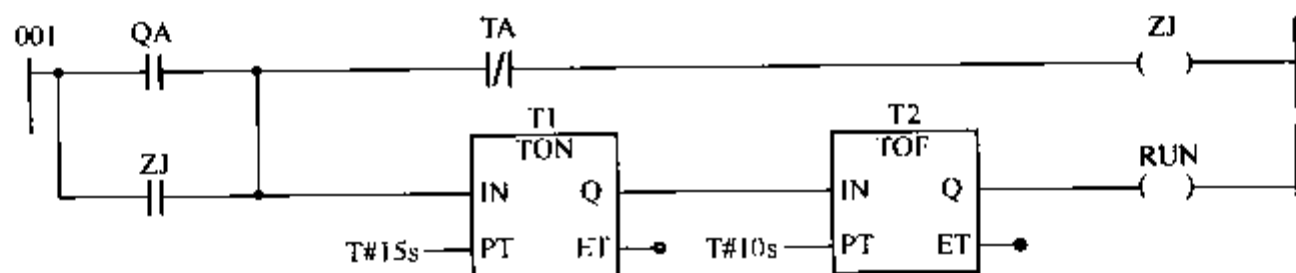


图 6-22 电动机延时启动和停止的控制程序

上面的示例说明对电动机的延时等与时间有关的程序具有类似的程序结构,根据不同的延时要求,采用不同的定时器或它们的组合即可。

6.2.2 长定时器的实现

IEC 61131-3 标准对定时器的定时时间没有规定其限度。但实际可编程控制器受存储器容量限制,不可能实现无限的定时时间。因此,对长时间的定时器,可采用有关程序实现。

1. 多定时器串联

多定时器串联程序可实现长定时。一个定时器计时到其设定值后,启动第 2 个定时器,然后,第 3 个定时器等。总延时时间是各定时器计时时间之和。

图 6-23 是一个定时器串联的程序示例。可编程控制器能够提供最长为 8 h 的定时器,为实现自动打印日报表,需要每隔 24 h 发送一个打印信号。当第一次闭合 START 开关后,图示程序能够每隔 24 h 给 PRINT 发送一个打印信号。

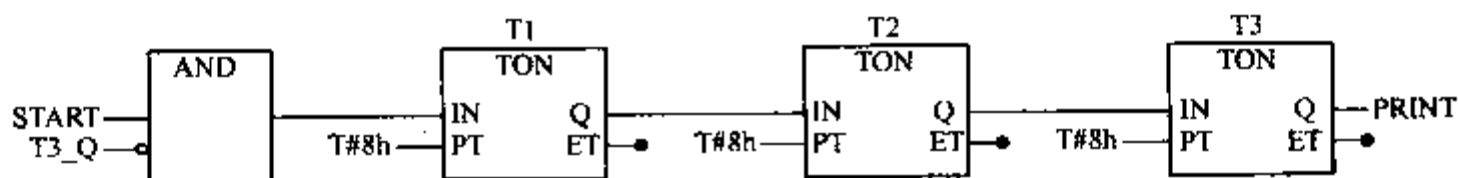


图 6-23 定时器串联程序示例

2. 定时器和计数器组合

定时器和计数器串联,可实现长定时功能。例如,图 6-23 的程序也可用图 6-24 所示定时器和计数器的串联程序实现。

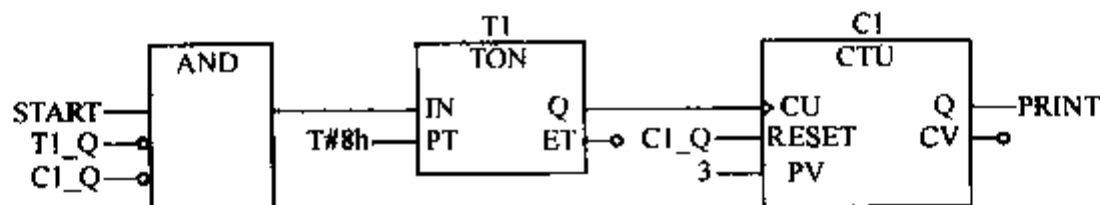


图 6-24 定时器和计数器串联程序示例

6.2.3 信号发生器的实现

脉冲信号发生器、方波信号发生器等已经在第 4.3.3 节介绍。

1. 锯齿波信号发生器

锯齿波信号发生器用于产生周期锯齿波,它被作为时间比例控制的信号源等。

假设锯齿波的周期为 10 s,幅值为 1,启动信号为 START,锯齿波以变量 OUT 输出。采用

延时接通定时器实现的程序如图 6-25 所示。信号波形如图 6-26 所示。

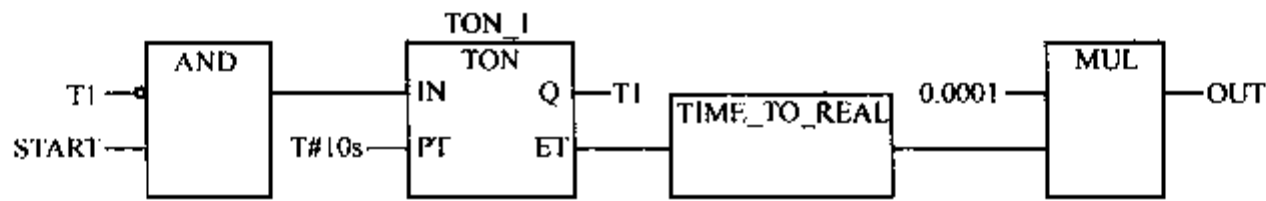


图 6-25 锯齿波信号发生器程序

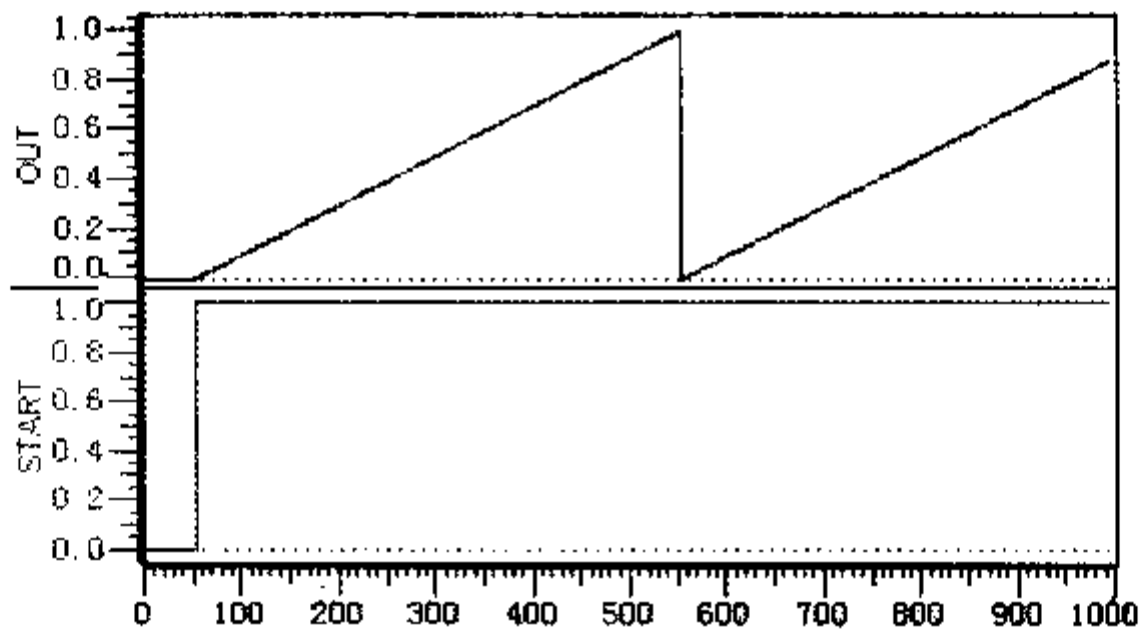


图 6-26 锯齿波信号波形图

须注意,将时间数据类型转换为实数数据类型时,时间数据以 ms 为单位,因此,锯齿波信号周期时间为 10000 ms,程序中乘以 0.0001 后,使锯齿波周期时间转换为实数 1.0。如果希望锯齿波的幅值为 K,应在乘法函数的乘数用 0.0001 与 K 之积表示。

锯齿波周期为 10 s,图中数据采样周期是 20 ms,因此,START 信号为 1 后,需采样 500 次后锯齿波才达到最大值,并在下一扫描周期返回到起点。

锯齿波信号发生器利用定时器的 ET 输出随时间增加的特性,再将时间数据转换为实数。这种方法也可应用于其他场合。例如,下面介绍的正弦波信号发生器等。可将它作为标准信号发生器使用。

应用锯齿波信号发生器可实现时间比例控制。图 6-27 是在锯齿波信号发生器程序中添加的有关程序和它的输出波形,该程序用于产生时间比例控制的输出。图中,AOUT 是经控制算法计算所得模拟量输出值,SCALE 是该模拟输出范围,TOUT 是对应的开关量输出。

时间比例控制中,打开的时间与总时间之比称为占空比。本程序中,打开的时间由 AOUT 决定,总时间为 10 s。

2. 三角函数信号发生器

锯齿波信号发生器和正弦函数等三角函数结合,可实现三角函数信号发生器。

图 6-28 是正弦波信号发生器程序。正弦信号以 10s 为周期,由于正弦函数输入是用弧度计算的,因此,需将乘数 0.0001 改为 0.000628,正弦波信号幅值是 OUT 输出乘以幅值 K 获得。

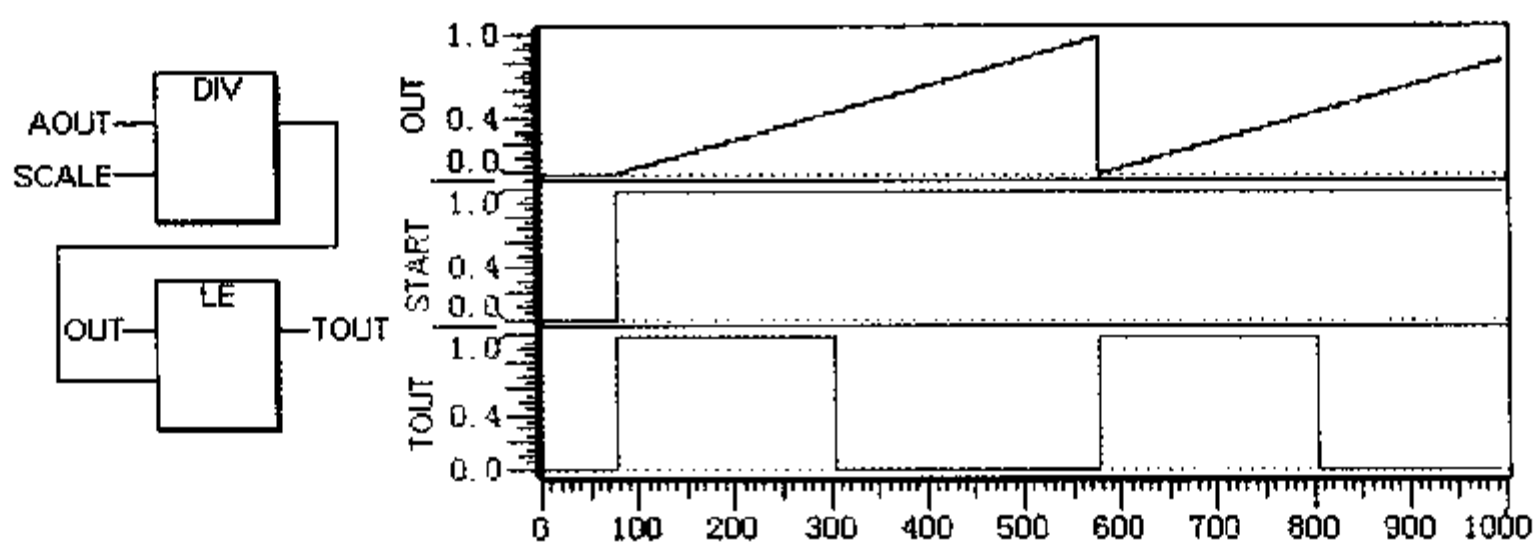


图 6-27 时间比例控制的程序和输出波形

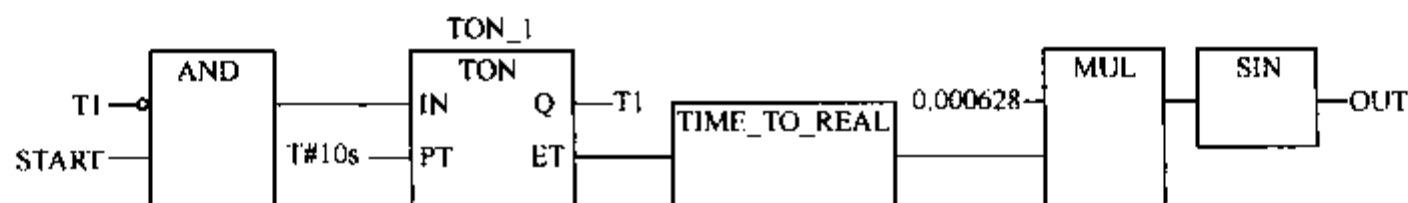


图 6-28 正弦信号发生器程序

如果在上述程序中将定时器设定时间由 10 s 改为 5 s, 则可获得以 10 s 为周期的全波整流输出信号。图 6-29 是输出的全波整流信号波形图。

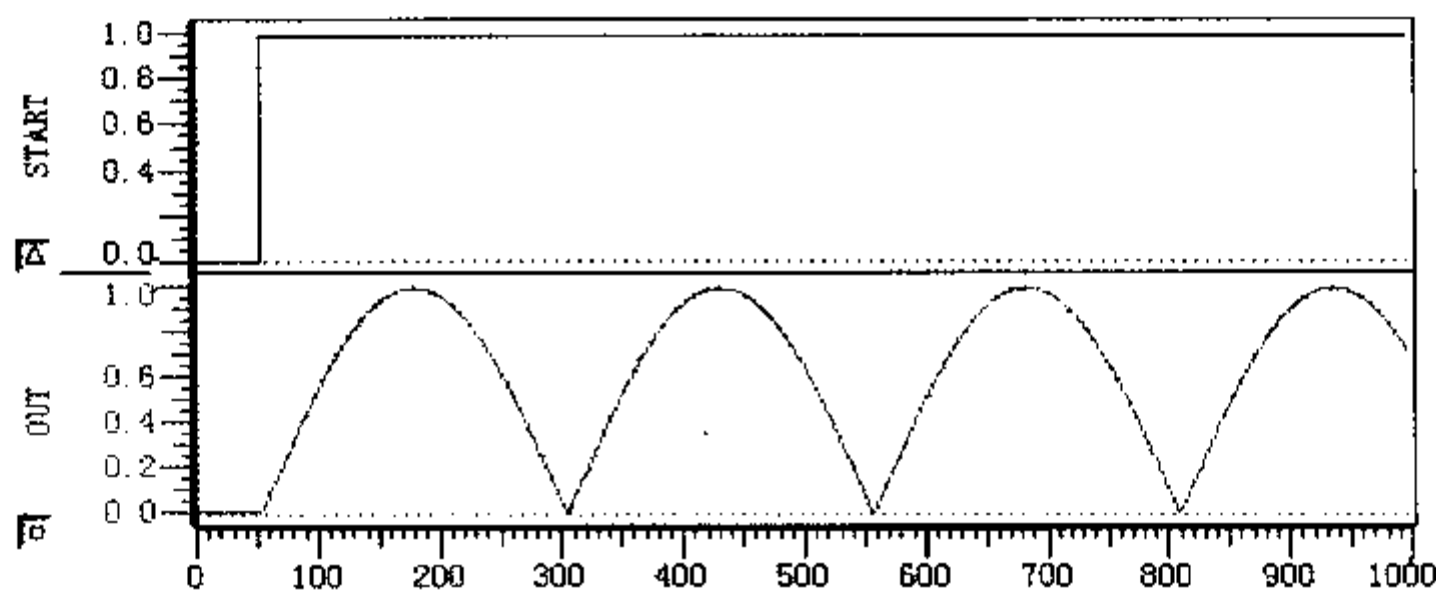


图 6-29 全波整流输出信号波形图

类似地, 半波整流信号也可用全波整流信号与方波信号进行相乘获得。其他三角函数信号, 例如, 余弦函数信号发生器等也可类似地发生。

在应用时须注意生成信号的周期由定时器确定, 信号幅值由输出乘幅值设置。

6.2.4 计数器的应用

1. 长计数器的实现

IEC 61131-3 标准对计数器的计数值没有规定其限度。但实际可编程控制器受存储器容量限制, 不可能实现无限的计数。因此, 对长计数的计数器可采用有关程序实现。

多计数器串联程序可实现长计数。一个计数器计数到其设定值后, 起动第 2 个计数器, 然

后,起动第 3 个计数器等。总计数次数是各计数器计数设定值之积。

图 6-30 显示了用两个加计数器 C1 和 C2 实现长计数功能的程序。START 给出计数脉冲输入信号。C1 计数器功能块计数设定值是 100,C2 计数器功能块计数设定值是 500,当计数值达到 50000 时,C2 计数器功能块输出一个脉冲信号。

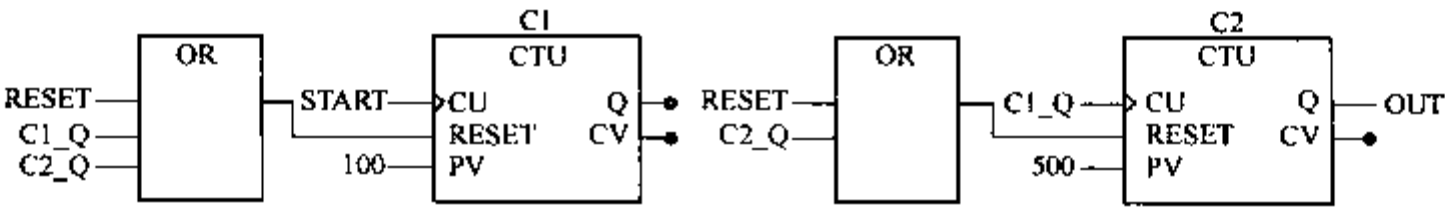


图 6-30 多计数器组成长计数功能的程序

2. 3 个按钮实现多台电动机的起保停控制

为减少可编程控制器的输入信号,可用 3 个按钮和计数器、比较函数等实现对多台电动机的起保停控制。图 6-31 显示对 4 台电动机的起保停控制程序。

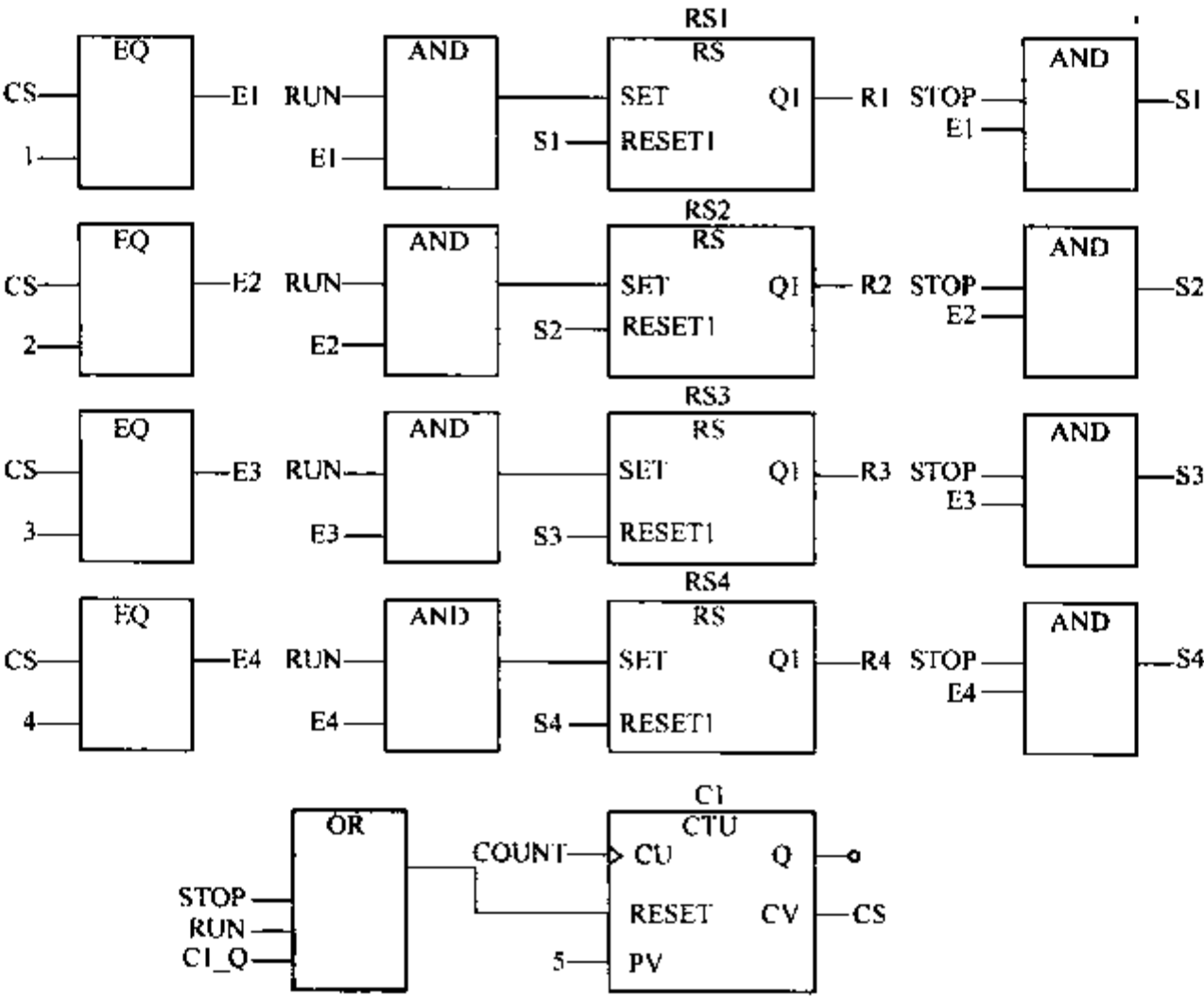


图 6-31 3 个按钮实现 4 台电动机的起保停控制程序

程序中,按钮 COUNT 用于输入电动机的编号,如果对第 1 台电动机操作按 1 次,对第 2 台电动机操作按 2 次,对第 3 台电动机操作按 3 次,对第 4 台电动机操作按 4 次。然后,按 RUN 按钮,对由 COUNT 选择的电动机执行起动操作,按 STOP 按钮,对由 COUNT 选择的电动机执行停止操作。程序用一个计数器选择所需电动机编号,从而可通过 RUN 和 STOP 按钮实现起动和停止的操作。

当按 COUNT 的次数不对时,可按足 5 次,消除所选的电动机编号。如果电动机的数量多

于 4 台,只需在第 4 行和第 5 行之间类似地插入有关电动机的程序,并对计数器设定值作更改即可,程序的其他部分不必改动。输出 CS 可连接数码显示管,显示对哪台电动机进行操作。

采用结构化文本编程语言编写的程序要简单得多,例如,不需要 4 个 EQ 函数,8 个 AND 函数等。下面是用结构化文本编程语言编写的程序:

```
C1(CU:=COUNT, RESET:=(STOP OR RUN OR C1.Q), PV:=5);
CS:=C1.CV;
CASE CS OF
1:   RS1(SET:=RUN, RESET1:=STOP);
      R1:=RS1.Q1;
2:   RS2(SET:=RUN, RESET1:=STOP);
      R2:=RS2.Q1;
3:   RS3(SET:=RUN, RESET1:=STOP);
      R3:=RS3.Q1;
4:   RS4(SET:=RUN, RESET1:=STOP);
      R4:=RS4.Q1;
ELSE CS:=0;
END_CASE;
```

如果希望每次操作后,CS 能够清零,以便后续操作,可在上述程序的 4 个 CASE 选项后添加 CS:=0,例如,R1:=RS1.Q1;CS:=0;。

3. 操作权限确认控制

实际操作过程中,可对一些重要操作设置操作权限,例如,操作员按击次数符合规定次数,就可进行有关操作。图 6-32 显示用两个计数器实现操作权限确认控制程序。

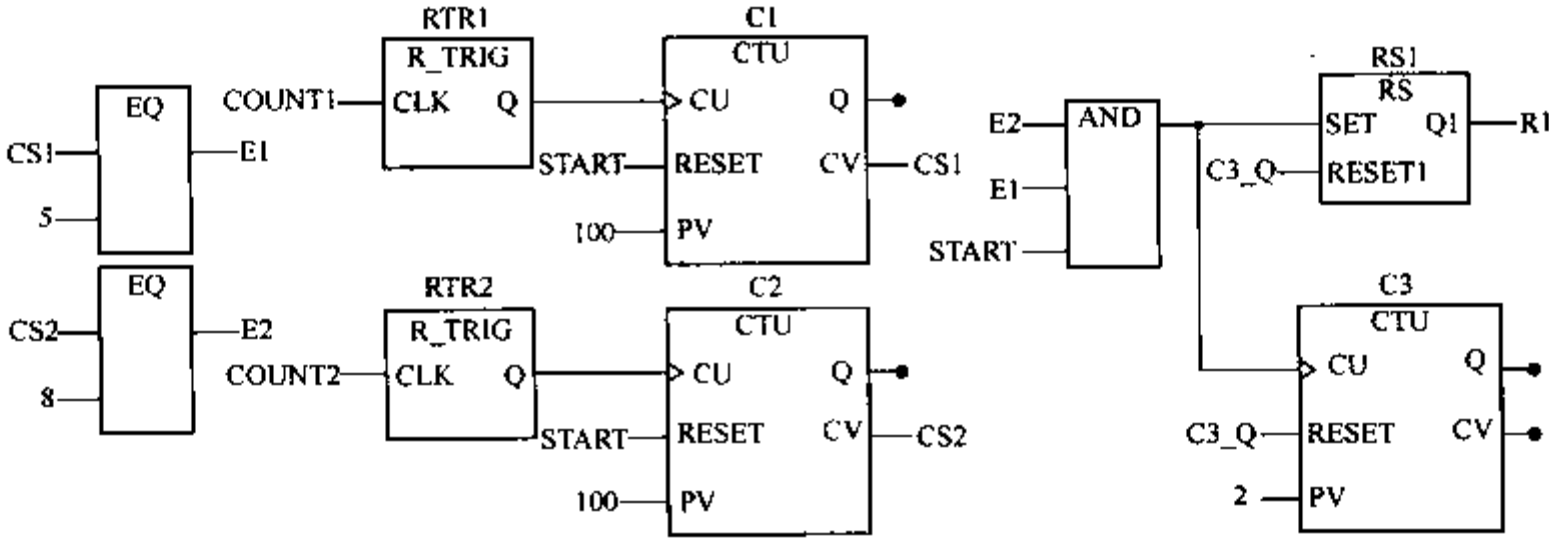


图 6-32 操作权限确认控制程序

假设操作权限设置如下:按钮 COUNT1 规定按击次数为 5 次,按钮 COUNT2 规定按击次数为 8 次。

操作权限确认控制程序的操作过程如下:按钮 COUNT1 按击 5 次,按钮 COUNT2 按击 8 次。再按 START 按钮可对 R1 执行起动操作。要使 R1 停止,同样要通过操作权限确认的过程,即按钮 COUNT1 按击 5 次,按钮 COUNT2 按击 8 次,再按 START 按钮可对 R1 执行停止操作。

如果按击次数不正确,则按 START 按钮后可重新进行操作权限确认的操作。

操作权限确认的规定按击次数也可经外部输入,例如,用操作拨盘输入数据,并传送到比较函数的两个输入端实现。

这里,R1 是软锁,有操作权限的操作员才能开启。

6.3 比较函数的应用

采用 3 个按钮和计数器实现对多台电动机的起停控制程序和操作权限确认控制程序等已经使用比较函数(等于)实现有关控制要求。下面介绍比较函数的其他应用。

6.3.1 旋转定位控制

组合机床的工作台在旋转过程中需要控制其转速。当工作台旋转到规定角度 T1 处需减速,旋转到 T2 处开始制动,旋转到 T3 处停止,停止处角度误差大于规定值 T4 则报警。

1. 格雷码转换为二进制码

采用 10 位绝对值编码器对旋转角度检测。格雷码转换为二进制码采用异或运算进行。因此,编码器的格雷码转换为二进制码可用图 6-33 所示程序实现,也可用异或函数实现。

2. 旋转定位控制

10 位绝对值编码器产生数字 0 ~ 1023,因此,可用一个字节的 BCD 码表示。将二进制数字转换为 BCD 码数值,可用数据类型转换函数。假设二进制数存放在 %IW0,用 W_BCD_TO_INT 函数将编码值转换为整数 ZJ。用 EQ 等于比较函数将该整数 ZJ 与规定的转角 T1 比较,如果相等则减速 JS。如果旋转的角度大于等于 T2,则用 GE 大于等于比较函数进行比较,满足条件后开始制动 ZD。如果旋转角度等于 T3,即用 EQ 等于比较函数比较 ZJ 与 T3,满足条件就停止 TZ。如果停止时旋转角度已超过 T4,即用 GE 大于等于比较函数进行比较,满足时发送报警信号 ALM。结构化文本编程语言的程序如下:

```
ZJ := W_BCD_TO_INT(%IW0);      (* BCD 转角值转换为整数 ZJ *)
JS := (ZJ = T1);                (* 如果转角等于 T1,则减速 JS 为真 *)
ZD := (ZJ >= T2);               (* 如果转角大于等于 T2,则制动 ZD 为真 *)
TZ := (ZJ = T3);                (* 如果转角等于 T3,则停止 TZ 为真 *)
ALM := (ZJ >= T4) AND NOT RUN;  (* 如已停止及转角大于等于 T4,则发报警 ALM *)
```

测量的旋转角度是模拟量,要转换为数字量,这称为量化。

设模拟量为 y ,量化后的数字量为 y^* ,则有

$$y = K_1 q y^* + K_2 \quad (6-1)$$

式中, K_1 ——变送器输出输入量程范围之比;

K_2 ——零点压缩;

q ——量化单位, $q = \frac{M}{2^N}$;

M ——模拟量全量程;

N ——寄存器位数。

y^* 是数字量,只能取整数。对 8 位编码器,转换精度为 0.5 级,对 10 位编码器,转换精度为 0.1 级,对 12 位编码器,转换精度可达 0.025 级。

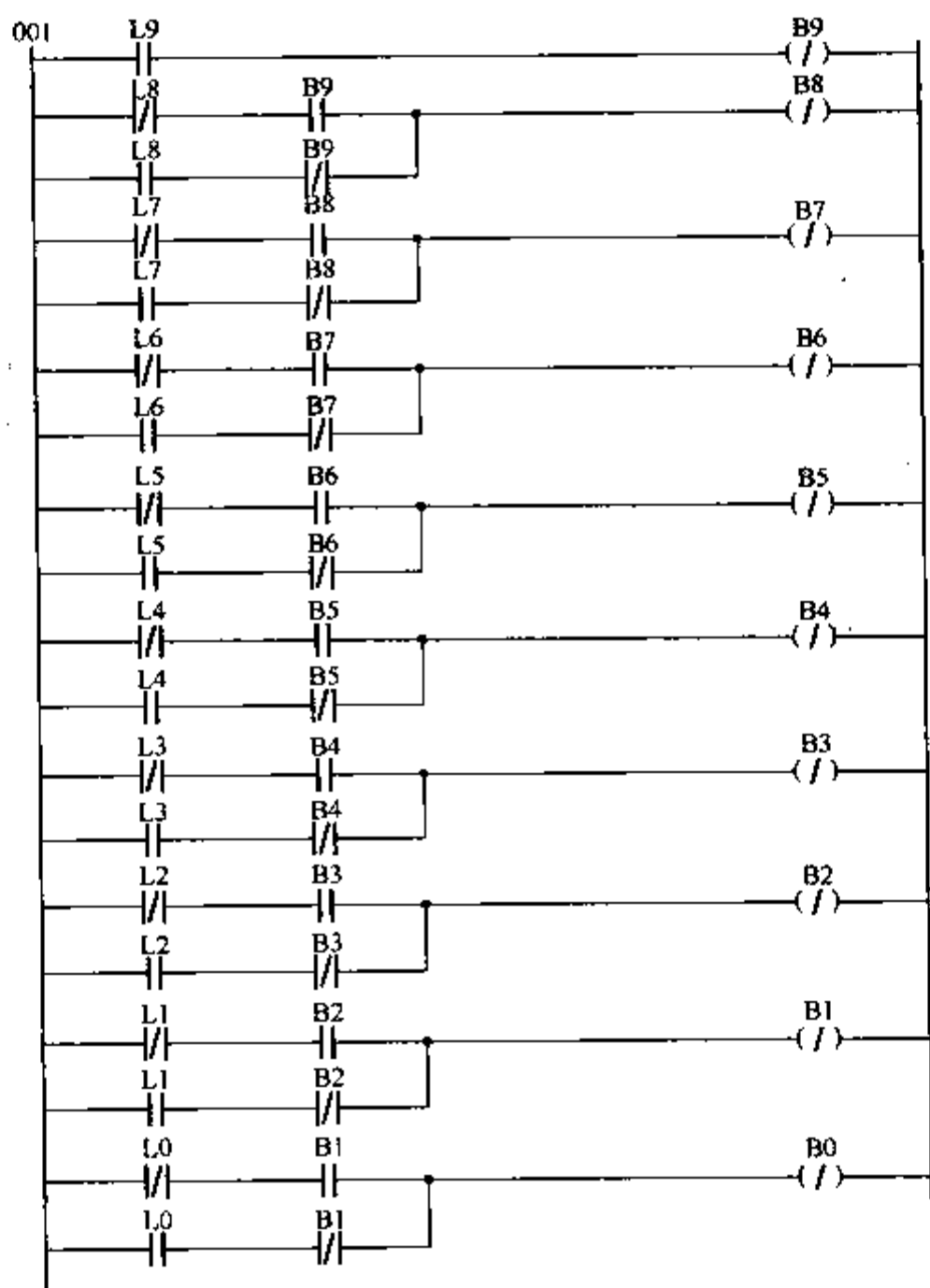


图 6-33 格雷码转换为二进制码程序

【例 6-1】 模拟量量化计算。

假设角度检测采用 10 位绝对值编码器,角度量程为 $0 \sim 360^\circ$,求 240° 和 270° 对应的数字量。

模拟量全量程

$$M = 360 - 0 = 360$$

$$N = 10, \text{ 则 } q = \frac{M}{2^N} = 360/1024$$

$$K_1 = \frac{360 - 0}{360 - 0} = 1, K_2 = 0^\circ, y = 240^\circ$$

因 $y = K_1 q y^* + K_2$, 得 $y^* = \frac{y - K_2}{K_1 q} = \frac{240 - 0}{360/1024} = 683$, 即 16#2AA。

同样, 270° 对应的数字量是 768, 即 16#300。

须注意, 转换为数字量后是整数, 因此, 采用四舍五入获得整数。

该旋转定位控制系统的 T1 是 240° , 程序中, 对应的数字应填 16#2AA, T2 是 270° , 程序中, 对应的数字应填 16#300, T3 是 360° , 表示旋转一周后停止, 因此, 程序中, 对应的数字应填

16#000。该系统中,数字 1 表示角度 0.35° ,该旋转定位控制系统设置 T4 为 2,即 0.7° 。如要提高旋转定位的控制精度,可改用 12 位绝对值编码器,这时,数字 1 表示 0.08789° 。

程序中,RUN 信号是运转状态信号,当旋转运行时,其值为 1。因此,只有在停止旋转后,如果超过 T4,才发送报警信号。

6.3.2 分度盘工位控制

分度盘每 18° 为一个工位,旋转一周有 20 个工位可进行控制。系统采用 8 位绝对值编码器检测旋转角度。为此,需要每隔 18° 有一个输出,用于作为该工位的操作信号。用等于比较函数实现旋转角度与不同工位对应角度的比较,并发送输出。此外,分度盘到达某工位后,需发送停止旋转信号,直到该工位的操作结束。控制系统要求在电源掉电后或紧急停车后能够保持停车前的位置。

根据该控制系统分析,可采用等于比较函数实现,功能块编程语言编写的程序见图 6-34。

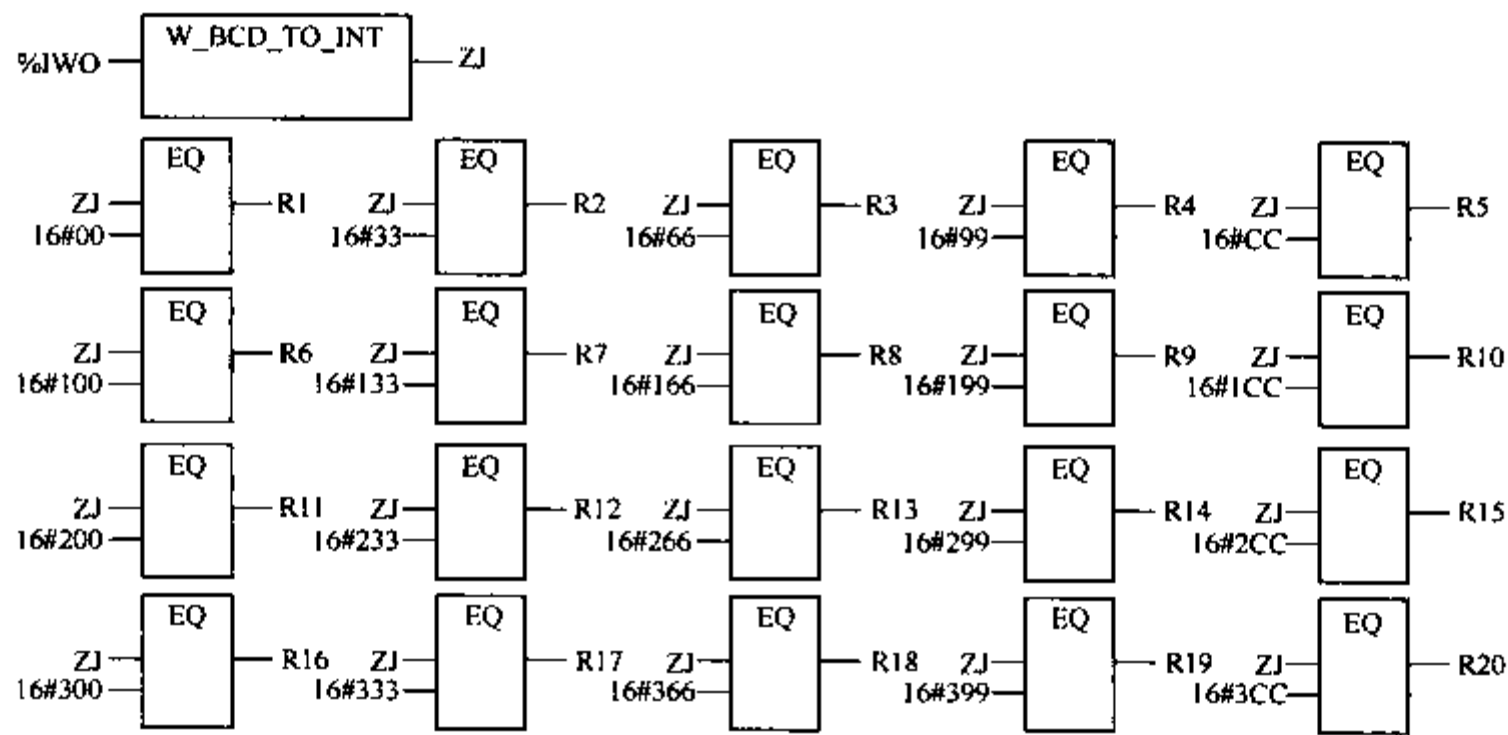


图 6-34 分度盘工位控制程序

分度盘的工位数据存放在 %IW0,它是 BCD 数,转换为整数 ZJ 后,与标准角度进行比较,一周的转角分为 20 等份,每个转角为 18° ,8 位绝对值编码的数据为 0 ~ 255。转换为十六进制是 0 ~ FF,每个转角为 16#33,因此,当 ZJ 等于 16#33 时,表示分度盘已经转过 18° ,对应的 R2 被置 1,类似地,当转过的角度为 54° 时,ZJ 与 16#99 比较结果为正,因此,对应的 R4 被置 1。

对多圈信号的转换,也可采用类型的程序。例如,采用模除函数将数据转换成小于 360° ,然后进行等于的比较操作。

须注意,用于检测分度盘角度的变量具有保持属性,即 RETAIN 属性。可保证在电源掉电时,该变量的值保持不变,在恢复电源后,仍能在掉电前的工位继续操作。

采用结构化文本编程语言编写的程序较简单,程序如下:

```
ZJ := W_BCD_TO_INT(%IW0);
FOR I := 0 TO 19
```

```

IF EQ(16#33 * I, ZJ) THEN R[1] := 1;
END_IF;

```

须注意,程序的变量声明中,R 是数组变量。

6.3.3 自动增益控制

自适应控制系统是一类能够适应过程特性或环境条件的变化,自动调整控制器参数的控制系统。简单自适应控制系统根据过程动态特性和扰动动态特性,调整控制器参数。通常,仅调整控制器的增益。因此,也称增益自适应控制系统。

依据偏差自动调整控制器增益的控制系统是一种简单自适应控制系统。假设采用 PI 控制算法,即

$$u = k_c ef(e) + \frac{k_c}{T_i} \int ef(e) dt \quad (6-2)$$

式中, $f(e)$ 是偏差 e 的函数。一种最简单的算法是 $f(e) = |e|$, 这表明,偏差大时控制作用增强,偏差小时控制作用缓和。这种系统在 pH 控制等工业过程控制中获得成功。

在可编程控制器中,设置一些偏差区间,在某一偏差区间,控制器用某一增益,不同的偏差区间有不同的增益,控制器根据偏差大小和对应的增益,计算出在该偏差下的控制器输出。

通常,偏差 e 较小,偏差函数 $f(e)$ 的值较小,控制器输出较小,系统响应较平缓。偏差 e 较大,偏差函数 $f(e)$ 的值较大,控制器输出较大,系统响应较激烈。偏差 e 和偏差函数 $f(e)$ 的关系类似模糊控制,需根据所采用的 A/D 卡转换精度等确定。一般,最大增益与最小增益之比不应大于 4。

采用 8 位输入 A/D 转换卡,测量信号 PV 与设定 SP 的差是偏差 e ,将偏差的上限和下限存放在 ES1 ~ EX1、ES2 ~ EX2、...、ES16 ~ EX16 等变量中,用大于等于和小于比较函数将偏差与这些限值比较,如果,偏差在某一偏差区间,则输出为对应的偏差函数值,偏差函数值存放在 K1 ~ K16 中。偏差值存放在 PC 变量内。程序见图 6-35。

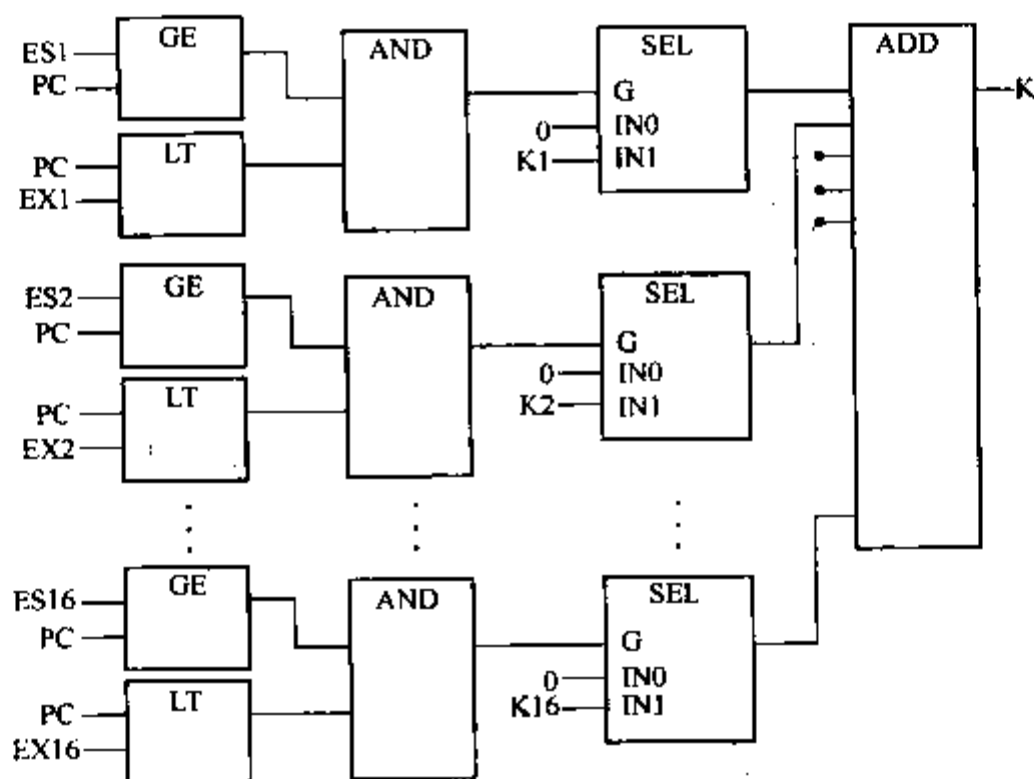


图 6-35 自动增益控制程序

这类程序也可用于非线性函数的折线近似。

6.4 运算函数的应用

逻辑运算可直接用可编程控制器的逻辑函数和功能块实现。代数运算通常要用运算类函数实现。例如,气体流量的温度、压力补偿运算,流量的开方运算,时间比例控制等。

6.4.1 控制算法

1. 时间比例控制

时间比例控制是将控制器的连续输出转换成与输出成比例的开关量输出的一类控制。例如,在一些精细化工生产过程中常采用时间比例控制。时间比例控制将连续的控制输出信号转换为占空比与输出成比例的开关量输出。通常,输出的接通时间与连续输出量成比例。因此,称为时间比例控制。接通时间与总周期之比称为占空比。

例如,8位模拟量输出对应数字量是0~255。假设总周期T₂是30s,即在30s内开关量有一次接通和断开过程。如果控制器连续输出是y,则接通时间应为

$$T_1 = \frac{30000}{255} \times y \text{ (单位:ms)} \quad (6-3)$$

用图6-32所示的程序组成方波信号发生器,定时器功能块T₂的设置即为总周期,本例可输入T#30s。时间比例控制的程序如图6-36所示。

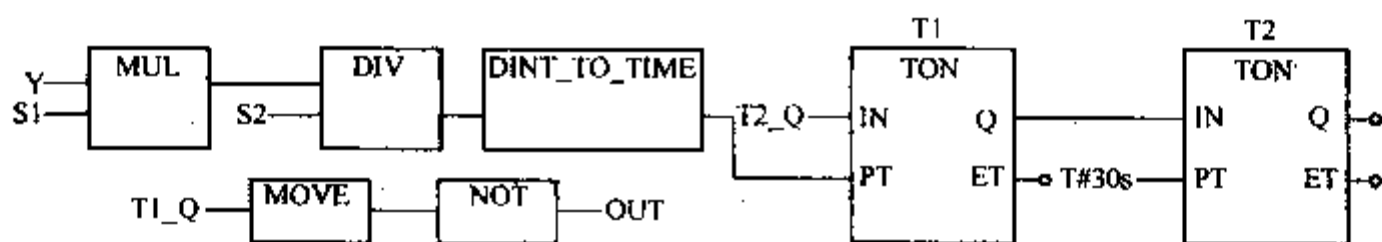


图 6-36 时间比例控制程序

程序中,MUL函数的输入S1,初始值设置为16#7530,即十进制数30000,变量数据类型是DINT。同样,DIV函数的输入S2,初始值设置为16#00FF,即十进制数255,数据类型也是DINT。虽然30000没有超过整数INT的范围32767,考虑应用的广泛性,仍用长整数数据类型。数据转换为时间的函数也采用长整数转换为时间的DINT_TO_TIME函数,转换后的数值以秒为单位。须注意,控制器输出Y也是长整数数据类型。

2. 比例积分微分控制

计算机控制系统中,采用比例积分微分控制算法。其位置算法是

$$u(k) = k_c e(k) + \frac{k_c}{T_i} \sum_{i=0}^k e(i) T_s + k_c T_d \frac{e(k) - e(k-1)}{T_s} \quad (6-4)$$

增量算法是

$$\Delta u(k) = k_c \Delta e(k) + \frac{k_c}{T_i} e(k) T_s + k_c T_d \frac{e(k) - 2e(k-1) + e(k-2)}{T_s} \quad (6-5)$$

(1) 积分控制功能块。

积分控制功能块实现积分运算。用于消除控制系统的余差,改善系统静态性能。积分控

制功能块的程序如下：

```

FUNCTION_BLOCK INTEGRAL1      (* 功能块名 INTEGRAL1 *)
VAR_INPUT                    (* 输入变量声明段开始 *)
    RUN : BOOL ;              (* 积分参数,1:积分;0:保持 *)
    R1 : BOOL ;               (* 超驰重定,1:超驰重定;0:积分或保持 *)
    XIN : REAL ;              (* 输入,通常是偏差信号 *)
    XO : REAL ;               (* 积分初始值 *)
    CYCLE : TIME ;            (* 采样周期 *)
END_VAR                      (* 变量声明段结束 *)
VAR_OUTPUT                   (* 输出变量声明段开始 *)
    Q : BOOL ;                (* 超驰状态说明,1:非超驰;0:超驰 *)
    XOUT : REAL ;             (* 积分项输出 *)
END_VAR                      (* 变量声明段结束 *)
Q := NOT R1 ;                (* 功能块本体开始, Q 等于非 R1 *)
IF R1 THEN XOUT := XO ;      (* 超驰状态,功能块输出等于积分初始值 *)
ELSIF RUN THEN XOUT := XOUT + XIN * TIME_TO_REAL (CYCLE) ;
                             (* 积分输出的计算公式 *)
END_IF;                      (* 功能块本体结束 *)
END_FUNCTION_BLOCK           (* 功能块结束 *)

```

程序中,积分控制算法采用累加方法计算积分增量输出

$$u_i(k) = \sum_{i=0}^k e(i)T, \quad (6-6)$$

图 6-37 是该功能块的变量表。

名称	类型	用法	描述	地址	初值	保持	run	inc
Q	BOOL	VAR_OUTPUT	功能块状态			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XOUT	REAL	VAR_OUTPUT	功能块输出			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RUN	BOOL	VAR_INPUT	功能块状态设置			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XIN	REAL	VAR_INPUT	输入			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XO	REAL	VAR_INPUT	初始值			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CYCLE	TIME	VAR_INPUT	采样周期			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
R1	BOOL	VAR_INPUT	超驰重定设置			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

图 6-37 积分功能块的变量表

(2) 微分控制功能块。

微分控制功能块实现微分运算,用于消除高频噪声的影响,改善系统动态性能。微分控制功能块的程序如下：

```

FUNCTION_BLOCK DERIVATIVE1    (* 功能块名 DERIVATIVE1 *)
VAR_INPUT                    (* 输入变量声明段开始 *)
    RUN : BOOL ;              (* 微分参数,1:加微分;0:不加微分 *)
    XIN : REAL ;              (* 输入,通常是偏差信号 *)
    CYCLE : TIME ;            (* 采样周期 *)

```



```

END_VAR                                ( * 变量声明段结束 * )
VAR_OUTPUT                             ( * 输出变量声明段开始 * )
    XOUT : REAL ;                      ( * 微分功能块输出 * )
END_VAR                                ( * 变量声明段结束 * )
VAR                                    ( * 变量段声明开始 * )
    X1, X2 : REAL ;                   ( * 中间变量,用于存放前两次的偏差 * )
END_VAR                                ( * 变量段声明结束 * )
IF RUN THEN                            ( * 功能块本体开始 * )
    XOUT := ( XIN + X1 - 2.0 * X2 ) / TIME_TO_REAL(CYCLE); ( * 计算微分功能块输出 * )
    X2 := X1; X1 := XIN;              ( * 偏差项替换 * )
END_IF;                               ( * 功能块本体结束 * )
END_FUNCTION_BLOCK                     ( * 功能块结束 * )

```

程序中,微分控制算法采用差分方法计算其增量输出为

$$u_D(k) = \frac{e(k) - 2e(k-1) + e(k-2)}{T_s} \quad (6-7)$$

实际应用时,为抑制高频噪声,可采用如下算法:

$$u_D(k) = \frac{e(k) + 3e(k-1) - 3e(k-2) - e(k-3)}{6T_s} \quad (6-8)$$

功能块本体的程序和变量声明需作相应改动。该功能块变量表与图 6-37 类似。

(3) 比例积分微分控制功能块

比例积分微分控制功能块实现比例积分微分运算,比例积分微分控制功能块调用积分功能块和微分功能块,其程序如下:

```

FUNCTION_BLOCK PID1                    ( * 功能块名 PID1 * )
VAR_INPUT                             ( * 输入变量声明段开始 * )
    AUTO : BOOL ;                     ( * 手动/自动参数,1:自动;0:手动 * )
    PV : REAL ;                       ( * 过程测量 * )
    PV : REAL ;                       ( * 过程测量 * )
    XO : REAL ;                       ( * 过程输出初值 * )
    KP : REAL ;                       ( * 控制器放大系数 * )
    TR : REAL ;                       ( * 控制器积分时间 * )
    TD : REAL ;                       ( * 控制器微分时间 * )
    CYCLE : TIME ;                   ( * 采样周期 * )
END_VAR                                ( * 变量声明段结束 * )
VAR_OUTPUT                             ( * 输出变量声明段开始 * )
    XOUT : REAL ;                     ( * PID 功能块输出 * )
END_VAR                                ( * 变量声明段结束 * )
VAR                                    ( * 变量段声明开始 * )
    XP, XI, XD : REAL ;              ( * 比例、积分、微分控制输出 * )
    INT1 : INTRGRAL1 ;                ( * 积分功能块实例名 INT1 * )
    DER1 : DERIVATIVE1 ;              ( * 微分功能块实例名 DER1 * )
    ERR : REAL ;                     ( * 偏差 * )

```

```

END_VAR                                ( * 变量段声明结束 * )
ERR := PV - SP ;                       ( * 计算偏差 * )
INT1( RUN := AUTO, R1 := NOT AUTO, XIN := ERR, X0 := TR * ( X0-ERR ), CYCLE := CY-
CLE );

                                ( * 调用积分功能块 * )
DER1( RUN := AUTO, XIN := ERR, CYCLE := CYCLE ); ( * 调用微分功能块 * )
XP := KP * ERR ;                      ( * 计算比例控制输出 * )
XI := KP * INT1. XOUT ;               ( * 计算积分控制输出 * )
XD := KP * DER1. XOUT * TD ;         ( * 计算微分控制输出 * )
XOUT := XP + XI + XD ;               ( * 计算 PID 功能块输出 * )
END_FUNCTION_BLOCK                    ( * 功能块结束 * )

```

比例积分微分控制功能块的变量声明见图 6-38。

名称	类型	用法	描述	地址	初值	保持	PDD	OPC
Default								
AUTO	BOOL	VAR_INPUT				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
PV	REAL	VAR_INPUT				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SP	REAL	VAR_INPUT				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
X0	REAL	VAR_INPUT				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
KP	REAL	VAR_INPUT				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
TR	REAL	VAR_INPUT				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
TD	REAL	VAR_INPUT				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CYCLE	TIME	VAR_INPUT				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XOUT	REAL	VAR_OUTPUT				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
INT1	integral1	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
DER1	DERIVATIVE1	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ERR	REAL	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XP	REAL	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XI	REAL	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
XD	REAL	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

图 6-38 PID1 功能块的变量表

(4) PID 应用实例

上述 3 个功能块是衍生功能块。其中, PID1 功能块调用积分和微分功能块。实际应用时,只需要调用 PID1 控制功能块即可计算 PID1 控制器输出。下面是一个示例程序:

```

PROGRAM EXAMPLE
VAR
    KP1: REAL ;
    XX : REAL ;
    PIDD : PID1 ;
END_VAR
KP1 := 0.01 ;
PIDD( AUTO := %IX3.0, PV := BYTE_TO_REAL(%IB0), SP := BYTE_TO_REAL(%IB1), KP :=
KP1, TR := 1000.0, TD := 0.1, X0 := 0.0, CYCLE := T#0.5s );
XX := PIDD. XOUT ;
END_PROGRAM

```

生产过程的测量值由 8 位输入卡输入,其值为 0 ~ 255。设定值也从外部输入,其值也是 0 ~ 255。程序中将测量值和设定值转换为实数,其他控制器参数可直接输入或从外部输入,示例中,KP1 由程序设置为 0.01,积分时间和微分时间由调用 PID1 时设置。控制运算的输出由 XX 端输出。

6.4.2 运算函数的应用

1. 流量的温度压力补偿

气体流量测量时,由于气体密度受到气体实际温度和压力的影响,因此,气体流量测量时要进行温度压力补偿。补偿公式如下:

$$F_n = F_1 \cdot \frac{p_1 + 1.0332}{p_n + 1.0332} \cdot \frac{t_n + 273.15}{t_1 + 273.15} \tag{6-9}$$

式中,下标 n 表示设计状况的数据;下标 1 表示实际工况下的数据; F 、 p 和 t 是气体的流量、表压(bar)和摄氏温度。

为在可编程控制器实现上述补偿运算,需要进行加、乘和除的代数运算。当流量测量采用孔板和差压变送器时,还需要开方运算。

假设,压力测量信号存放在%IW0,温度测量信号存放在%IW2,流量测量信号存放在%IW4,为减少可编程控制器的计算工作,可将设计工况气体的表压和温度先转换为绝压和开氏温度,即将设计工况气体的绝压值和开氏温度值直接作为被除数或除数。温度压力补偿程序如图 6-39。

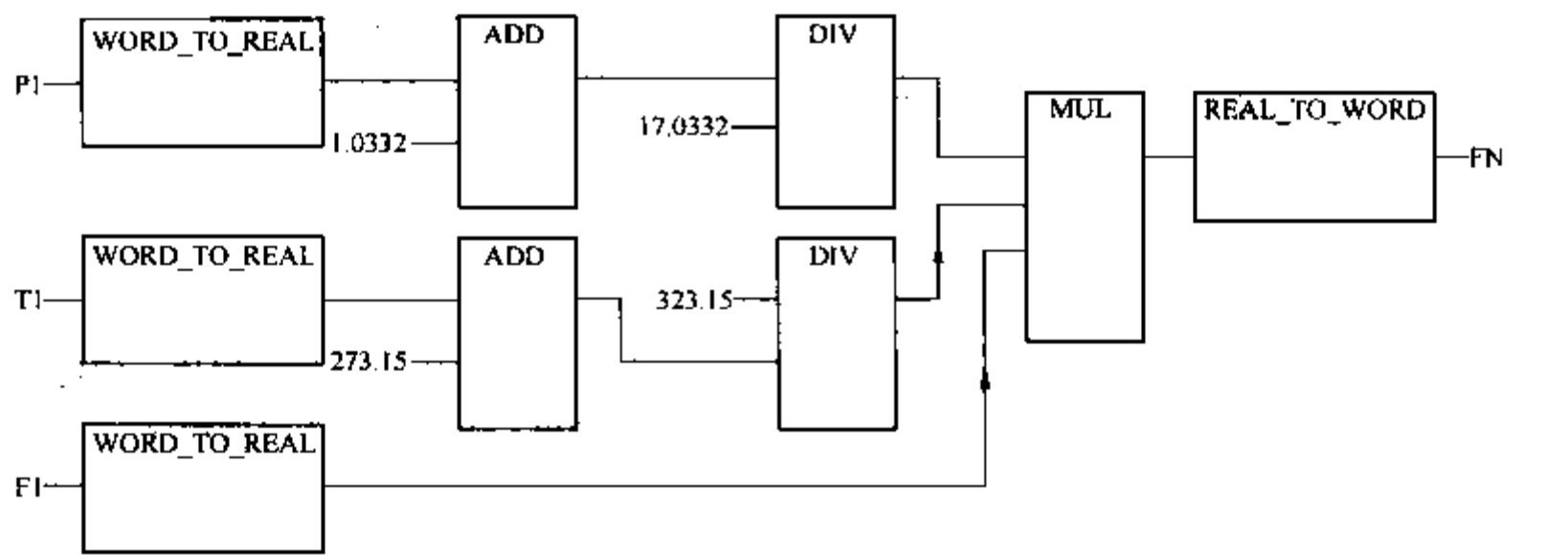


图 6-39 温度压力补偿程序

图示程序中,设计工况下,气体表压是 16bar,温度是 50℃。因此,直接用数据 17.0332 和 323.15 代入,以减少程序运算工作量。

考虑输入信号是 10 位 A/D 转换卡输入,因此,采用两个字节的字(WORD)存放,同样,输出数据也用字存放。图 6-40 是本程序的变量声明表。

2. 非线性补偿

非线性补偿环节可用于下列场合:

名称	类型	用法	描述	地址	初值	保持	PIB	OPC
Default								
F1	WORD	VAR	实际工况下测得的流量	%IW4		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
P1	WORD	VAR	实际工况下气体表压	%IW0		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
T1	WORD	VAR	实际工况下气体摄氏温度	%IW2		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
FN	WORD	VAR	折算到标准工况下的流量	%QW0		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

图 6-40 变量声明表

(1) 被控对象具有非线性特性

为此,采用非线性补偿环节,使合成后开环控制系统总特性成为线性特性或近似线性特性,满足控制系统稳定运行准则。例如,非线性的控制阀流量特性补偿被控对象的非线性特性。pH 控制系统中,用非线性的控制阀流量特性不能补偿 pH 的非线性特性,需要用非线性控制规律实现非线性补偿。

(2) 引入非线性控制规律

使控制系统变得简单或满足一定的控制要求。例如,采用位式控制,时间比例控制等。

非线性补偿环节的实现有多种方法,下面是常用的实现方法。

(1) 采用非线性函数

例如,采用等百分比特性函数、指数函数、对数函数等。也可将它们组合,形成新的非线性函数。可编程控制器中采用标准函数可方便地实现。

【例 6-2】 过热蒸汽温度和压力的非线性特性。

过热蒸汽温度和压力之间有下列非线性关系:

$$T_{sp} = 97.2334 + 65.53931 \lg p + 1.7403p \quad (6-10)$$

式中, p 是过热蒸汽的绝压 (10^5 kPa); T_{sp} 是饱和温度控制器设定 ($^{\circ}\text{C}$)。图 6-41 是饱和温度控制系统和过热蒸汽温度和压力模型曲线。该控制系统计算不同过热蒸汽压力下的饱和蒸汽温度设定值,并进行控制。

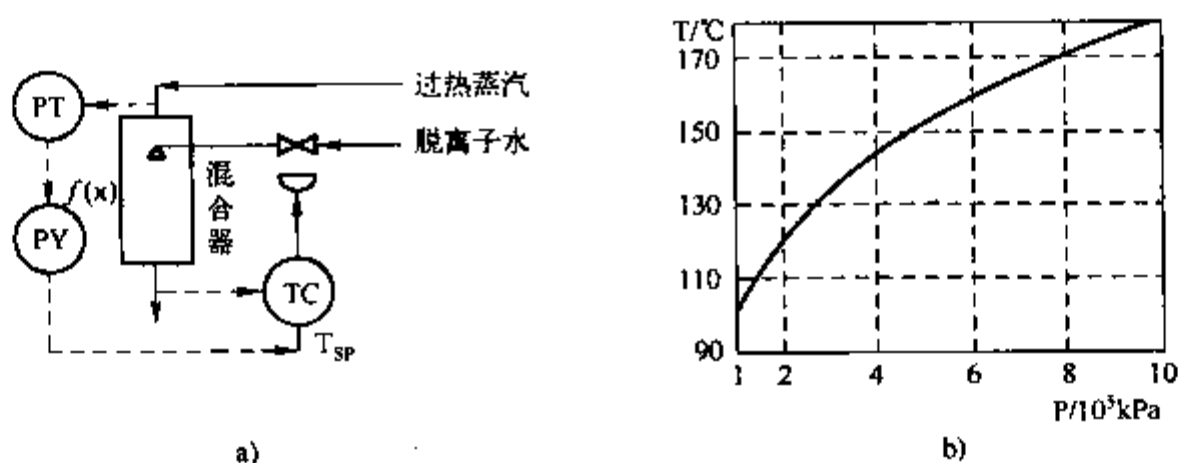


图 6-41 饱和蒸汽控制系统

为计算温度的设定值,采用可编程控制器的运算函数,组成非线性模型。非线性模型的程序见图 6-42。

压力变送器的量程为 $0 \sim 10 \times 10^5 \text{ kPa}$,采用 8 位输入卡,对应数字量 $-128 \sim 127$ 。程序中存放在 %IB0。如果对应数字量是 $0 \sim 255$,则将压力信号 P 经 WORD_TO_REAL 功能后,其返

回值作为除法功能的输入,除法功能的另一输入改为 25.5。运算后的返回值,即温度设定值存放在%QB0 中。程序见图 6-42。

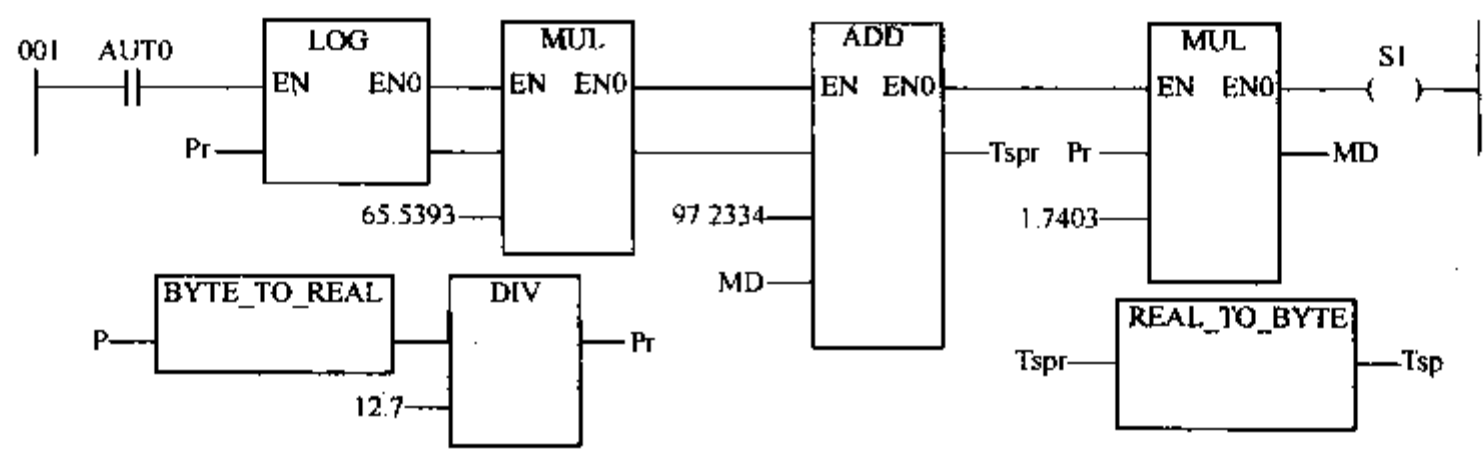


图 6-42 饱和蒸汽温度设定值计算程序

程序中, BYTE_TO_REAL 功能将压力值转换为数字量,由于有正负号,因此,转换后最大数字是 127,为对应压力的满量程值,应除以 12.7,才能获得实际的压力值 Pr。经运算后的温度设定值存放在 Tspr,当作为控制器设定值时,可直接用该值。

编制程序时须注意下列事项:

- 1) 变量的数据类型应一致,例如, LOG 函数的输入是实数,因此,不能直接用压力测量值 P(字节),必须经数据类型转换后才可使用。
- 2) 采用了 EN 和 ENO,它们可串联连接,程序中表示当控制器在自动状态才进行运算,从而降低可编程控制器的计算工作量。
- 3) 需要采用反馈变量进行连接,不能直接用连接线,例如,程序中的 MD。
- 4) 实际应用时,将 Tspr 直接作为温度控制器的设定(数据类型是实数),不需要转换为字节。
- 5) 虽然该程序是梯形图编程语言编写,但程序中,可采用功能块图表示有关运算,例如,转换压力测量值为实数,并进行除法运算来获得实际压力数据等。

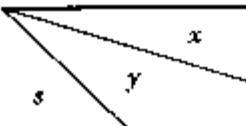
(2) 采用折线近似非线性特性

折线近似是实现非线性特性的常用方法。计算方法是将非线性特性的输入分隔成若干区间,可以平均分隔,也可以根据非线性特性进行不均匀分隔。根据输入分隔获得对应的输出值。各分隔区间内用线性插值描述该区间内输入和输出的关系,即输入信号落在某一区间内时,根据该区间内输入和输出满足线性关系,确定对应的输出。

【例 6-3】 控制阀流量特性的非线性补偿。

控制阀安装在实际生产过程的管道后,控制阀两端压降与系统总压降之比 $s < 1$,使控制阀理想流量特性畸变。对这种非线性特性可采用智能阀门定位器的非线性环节进行补偿,或在计算机控制装置中用非线性环节进行补偿。在可编程控制器中,为补偿因压降比造成的畸变,使流量特性产生非线性,通常可采用非线性环节进行补偿。例如,控制阀原流量特性是等百分比特性,因压降比不为 1,要实现等百分比流量特性,可用表 6-2 所示折线关系表进行非线性补偿。根据实际的压降比确定对应的折线点,并确定分隔区间,从而用可编程控制器实现非线性补偿。

表 6-2 非线性补偿用的折线关系表

	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0.1	0	0.0009	0.0026	0.0061	0.0138	0.0303	0.0644	0.1333	0.2679	0.5240	1
0.2	0	0.0022	0.0057	0.0120	0.0235	0.0449	0.0852	0.1602	0.2984	0.5496	1
0.3	0	0.0037	0.0093	0.0184	0.0337	0.0601	0.1061	0.1867	0.3278	0.5738	1
0.4	0	0.0052	0.0130	0.0250	0.0441	0.0752	0.1266	0.2122	0.3557	0.5965	1
0.5	0	0.0068	0.0167	0.0316	0.0543	0.0899	0.1463	0.2365	0.3820	0.6176	1
0.6	0	0.0083	0.0203	0.0380	0.0643	0.1042	0.1652	0.2596	0.4066	0.6372	1
0.7	0	0.0098	0.0239	0.0442	0.0739	0.1177	0.1831	0.2812	0.4296	0.6553	1
0.8	0	0.0113	0.0273	0.0501	0.0830	0.1306	0.2000	0.3016	0.4510	0.6720	1
0.9	0	0.0127	0.0305	0.0558	0.0917	0.1429	0.2159	0.3206	0.4709	0.6875	1
1.0	0	0.0140	0.0336	0.0612	0.0999	0.1544	0.2309	0.3384	0.4895	0.7017	1

采用这种方法也可补偿被控对象的非线性。需要注意,为补偿被控对象非线性和控制阀畸变的非线性,非线性补偿环节应串联在控制器与执行器之间。

假设 $s=0.5$,则从表 6-2 获得图 6-43 所示折线表示的非线性补偿用曲线。用 11 组 X, Y 数据,即 ES1 ~ ES11 和 EY1 ~ EY11,并采用线性插值方法,可获得某一输入 X0 对应的输出 Y0。

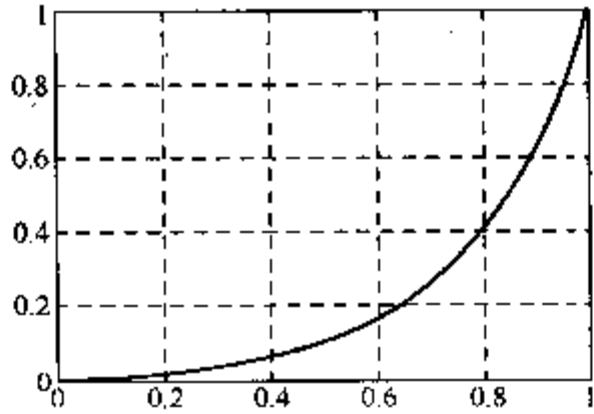


图 6-43 非线性补偿用曲线

1) 确定 X0 所在区间。采用与图 6-44 程序确定 X0 所在区间。程序中,ES1 ~ ES11 对应于输入区间的各分隔点数据,当输入 X0 在某一区间时,该区间的标志位置 1。例如,X0 在第 4 区间(ES4 ~ ES5 之间),则 K4 被置 1。

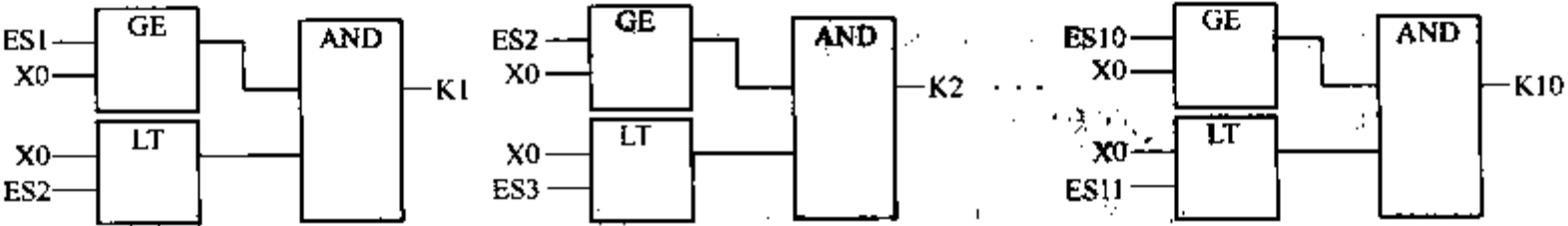


图 6-44 确定 X0 所在区间的程序

2) 线性插值确定输出 Y0。根据 X0 所在的区间,采用线性插值方法确定输出 Y0,计算公式如下:

$$Y_0 = \frac{EY_{i+1} - EY_i}{ES_{i+1} - ES_i} \times (X_0 - ES_i) + EY_i \quad (i = 1 \sim 10) \tag{6-11}$$

标志位 K1 的程序见图 6-45。

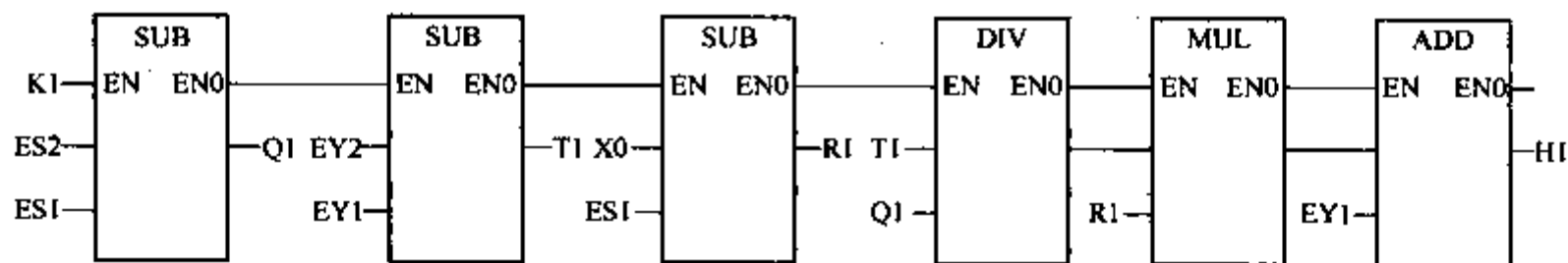


图 6-45 标志位 K1 的程序

其他标志位的程序可类似编制,不多述。程序采用 EN 和 ENO 的目的是用标志位作为 EN 的输入,使各函数的运算可实现。

3) 运算输出。各标志位作为选择功能的使能信号,各选择功能的输入有 H1 ~ H10 和零,选择函数的返回值作为加运算函数的输入,加函数的返回值即为 Y0。运算输出的程序类似图 6-36 的程序,不另画出。

程序编制时注意下列事项:

- 1) 输入信号 X0 是实数,而测量值根据所用 A/D 卡精度,可以存放在字节或字中,为此,应将测量值转换为实数,即用数据类型转换函数实现。
- 2) 非线性特性的输入分隔区间,应使在分隔区间内输入和输出呈现线性关系,因此,区间的分割应根据非线性特性的严重程度分割,非线性越严重,分割区间越密。
- 3) 有多种提高运算精度的方法。例如,可采用非线性插值的计算方法,如采用抛物线插值算法,但因计算工作量大,一般可采用增加分隔区间的方法来提高精度。例如,示例中可在输入 0.5 ~ 0.8 之间增加分隔区间。
- 4) 分隔区间越多,精度越高,但计算工作量越大。因此,在满足精度要求下,应尽量减少分隔区间。
- 5) 采用结构化文本编程语言编写运算类功能的程序具有编程简单的特点,但结构化文本编程语言编制的程序要消耗较多的 CPU 开销,应权衡利弊,统筹考虑选用合适的编程语言进行编程。

6.5 移位函数的应用

移位类函数在时间逻辑控制系统和顺序逻辑控制系统中有大量应用。

6.5.1 霓虹灯顺序点亮程序

时间逻辑控制系统的输出与时间有关,该控制系统根据时间的先后次序执行有关操作。下面以霓虹灯点亮和熄灭的控制系统为例说明时间逻辑控制系统的设计方法。

某霓虹灯点亮和熄灭的控制系统有 8 个霓虹灯,霓虹灯的点亮和熄灭按图 6-46 所示循

环变化。

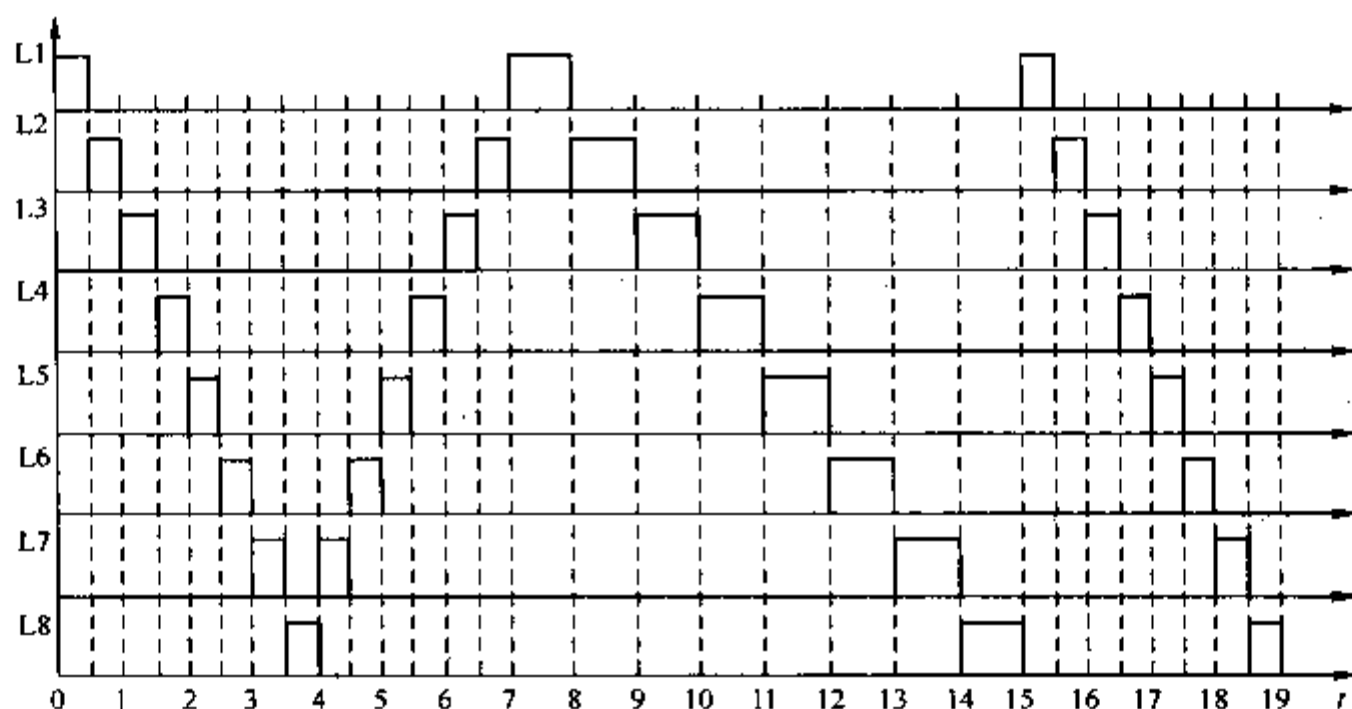


图 6-46 霓虹灯控制系统时序图

根据图示时间,每隔 0.5 s 有一个信号灯的变化,共有 30 步,循环周期为 15 s。循环周期开始的 7 s 内信号灯点亮时间为 0.5 s,以后点亮时间为 1 s

1. 脉冲信号发生器程序

由于该控制系统的时间间隔为 0.5 s,因此,用脉冲信号发生器程序产生 0.5 s 脉冲信号,它用于对移位函数进行移位操作。程序如下:

```
LD      START      (* 读取 START 开关的状态 *)
ANDN    T1. Q       (* 与 T1 定时器输出取反信号进行与逻辑运算 *)
ST      T1. IN      (* 运算结果作为定时器 T1 的输入 *)
LD      T#0.5S      (* 读取时间数据 *)
ST      T1. PT       (* 作为定时器 T1 的设定 *)
CAL      T1          (* 调用定时器 T1 功能块 *)
LD      T1. Q        (* 读取调用结果 *)
ST      L0           (* 存放到 L0,作为移位脉冲信号 *)
```

2. 移位控制程序

采用移位函数 ROL 实现移位。根据控制要求,需 30 步(计 15 s)实现一个循环过程,每步时间 0.5 s。程序分 3 部分,第 1 部分程序用计数器确定步数,程序如下:

```
LD      L0           (* 取移位脉冲信号 *)
ST      C1. CU        (* 作为计数器的脉冲信号 *)
LD      C1. Q          (* 取计数器输出信号 *)
ST      C1. RESET     (* 作为计数器的自复位信号 *)
LD      30             (* 计数器计数设定 *)
ST      C1. PV         (* 存放在计数器设定 *)
CAL      C1            (* 调用计数器 *)
LD      C1. Q          (* 取计数器输出 *)
```


ST C1Q (* 存放在 C1Q *)

第 2 部分程序是选择功能,用于在移位 30 步后进行复位操作,程序如下:

LD C1.Q (* 取计数器输出作为复位信号 *)
 SEL SF1, SF2 (* 选择的信号来自 SF1 和 SF2 *)
 ST SF1 (* 选择功能的返回值存放在 SF1 *)

第 3 部分程序用功能块图表示,第 3 部分程序有 EN 功能,用于确定何时进行移位操作。因此,可采用 EN 和 ENO 功能实现。移位功能是循环左移,移位位数是 1 位。被移位的数据存放在 SF1 中。程序如图 6-47 所示。

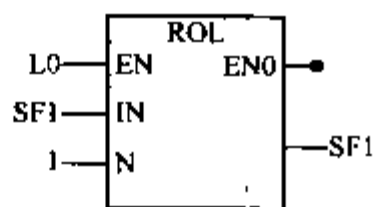


图 6-47 移位程序

3. 霓虹灯顺序点亮程序

根据图 6-47 所示霓虹灯点亮的步位置,可编写各霓虹灯点亮的步号。示例中,SF1 的数据存放在 %QW0 开始的双字(DWORD)中,程序如下:

LD %QX0.0 (* 取移位寄存器的第 1 步 *)
 OR %QX1.6 (* 或第 15 步 *)
 OR %QX1.7 (* 或第 16 步 *)
 AND START (* 只有 START 闭合时,才点亮 *)
 ST L1 (* 在上述步,点亮霓虹灯 L1 *)
 LD %QX0.1 (* 取移位寄存器的第 2 步 *)
 OR %QX1.5 (* 或第 14 步 *)
 OR %QX2.0 (* 或第 17 步 *)
 OR %QX2.1 (* 或第 18 步 *)
 ST L2 (* 在上述步,点亮霓虹灯 L2 *)
 LD %QX0.2 (* 取移位寄存器的第 3 步 *)
 OR %QX1.4 (* 或第 13 步 *)
 OR %QX2.2 (* 或第 19 步 *)
 OR %QX2.3 (* 或第 20 步 *)
 ST L3 (* 在上述步,点亮霓虹灯 L3 *)
 LD %QX0.3 (* 取移位寄存器的第 4 步 *)
 OR %QX1.3 (* 或第 12 步 *)
 OR %QX2.4 (* 或第 21 步 *)
 OR %QX2.5 (* 或第 22 步 *)
 ST L4 (* 在上述步,点亮霓虹灯 L4 *)
 LD %QX0.4 (* 取移位寄存器的第 5 步 *)
 OR %QX1.2 (* 或第 11 步 *)
 OR %QX2.6 (* 或第 23 步 *)
 OR %QX2.7 (* 或第 24 步 *)
 ST L5 (* 在上述步,点亮霓虹灯 L5 *)
 LD %QX0.5 (* 取移位寄存器的第 6 步 *)
 OR %QX1.1 (* 或第 10 步 *)
 OR %QX3.0 (* 或第 25 步 *)

```
OR      %QX3.1      ( * 或第 26 步 * )
ST      L6          ( * 在上述步,点亮霓虹灯 L6 * )
LD      %QX0.6      ( * 取移位寄存器的第 7 步 * )
OR      %QX1.0      ( * 或第 9 步 * )
OR      %QX3.2      ( * 或第 27 步 * )
OR      %QX3.3      ( * 或第 28 步 * )
ST      L7          ( * 在上述步,点亮霓虹灯 L7 * )
LD      %QX0.7      ( * 取移位寄存器的第 7 步 * )
OR      %QX3.4      ( * 或第 29 步 * )
OR      %QX3.5      ( * 或第 30 步 * )
ST      L8          ( * 在上述步,点亮霓虹灯 L8 * )
```

4. 变量声明

霓虹灯控制系统的变量表见图 6-48。SF1 变量选用双字数据类型,共可移位 32 位。为保证移位 0 位就进行循环,采用计数器 C1。定时器产生周期 0.5 s 的脉冲信号。

需说明,本示例采用两种编程语言编制程序,系统有两个任务,一个任务由指令表编程语言编写,另一个任务由功能块图编程语言编写。每个任务有一个变量表,图 6-48 用于指令表编程语言编写的程序。功能块图编程语言编写的程序只有 L0 和 SF1 两个变量,其数据类型和地址等与图 6-48 中的 L0 和 SF1 保持一致。

名称	类型	用法	描述	地址	初值	保持	PBD	OPC
Default								
START	BOOL	VAR		%IX0.0				
L0	BOOL	VAR		%QX7.0				
T1	TON	VAR						
C1	CTU	VAR						
C1Q	BOOL	VAR						
SF1	DWORD	VAR		%QD0	16#0001			
SF2	DWORD	VAR			16#0001			
L1	BOOL	VAR		%QX5.0				
L2	BOOL	VAR		%QX5.1				
L3	BOOL	VAR		%QX5.2				
L4	BOOL	VAR		%QX5.3				
L5	BOOL	VAR		%QX5.4				
L6	BOOL	VAR		%QX5.5				
L7	BOOL	VAR		%QX5.6				
L8	BOOL	VAR		%QX5.7				

图 6-48 霓虹灯控制系统的变量表

5. 时间循环功能块

用图 4.27 的功能块可直接编写。共有 8 组信号灯,每组信号灯是由两个 CYCTIME 功能块输出的或逻辑运算结果控制。例如,L1 信号灯的程序如下:

```
LD      START      ( * START 闭合时,才执行点灯程序 * )
CAL     CYCTIME_1( T1:= T#0S,T2:= T#0.5s,T3:= T#14.5s) ( * 调用 CYCTIME_1 * )
LD      CYCTIME_1.Q  ( * 读 CYCTIME_1 输出 * )
ST      C1          ( * 存放在 C1 * )
LD      START      ( * START 闭合时,才执行点灯程序 * )
CAL     CYCTIME_2( T1:= T#7S,T2:= T#1s,T3:= T#14s)  ( * 调用 CYCTIME_2 * )
```

LD	CYCTIME_2.Q	(* 读 CYCTIME_2 输出 *)
OR	C1	(* 与 C1 进行或运算 *)
ST	L1	(* 结果用于控制信号灯 L1 *)

其他信号灯的程序与上述程序类似,只需要改变有关时间设置即可。

6.5.2 操作权限确认控制

操作权限确认控制也可用移位函数实现。示例用 4 个按钮作为操作权限确认按钮,3 个按钮 S1、S2 和 S3 用于输入操作权限口令数值,第 4 个按钮 S4 用于确认输入的数值。如果输入的口令数值符合设置的数值,则输出 LS 置 1。程序如图 6-49 所示。

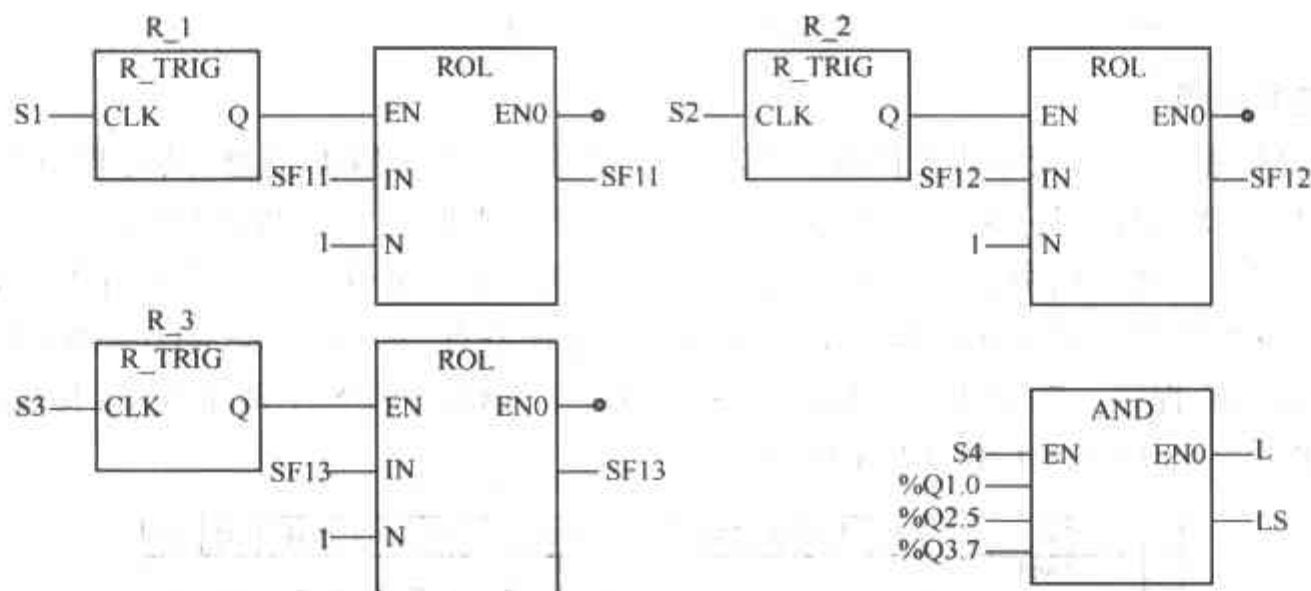


图 6-49 操作权限确认程序

程序中,与函数的 3 个输入数值是设置的权限数值,程序中为 1,6,8,即当 S1 按 1 次,S2 按 6 次,S3 按 8 次后,再按 S4 按钮,就能使输出 LS 置 1,并用于操作权限确认,如按的次数不符合上述数值,则表示操作权限不符合,不能使 LS 置 1。

示例的变量表见图 6-50。用于移位单元的变量 SF1、SF2 和 SF3,它们的数据类型是字节,因此,每个按钮最多可按 8 次,按的次数超过 8 次,表示重新开始计数。如果采用的数据类型是字,则每个按钮最多按动次数可达 16 次,它使操作权限确认的安全性大大提高。须注意,设置的权限数值由字的某位确定。例如,权限数值是 13,15,8,SF1、SF2 和 SF3 分别是 %QW0,%QW2,%QW4,则与逻辑函数的输入地址应设置为 %QX1.4,%QX3.6 和 %QX5.7。

名称	类型	用法	描述	地址	初值	保持	PDI	OPC
Default								
R_1	R_TRIG	VAR						
R_2	R_TRIG	VAR						
R_3	R_TRIG	VAR						
S1	BOOL	VAR		%I0.0				
S2	BOOL	VAR		%I0.1				
S3	BOOL	VAR		%I0.2				
S4	BOOL	VAR		%I0.3				
L	BOOL	VAR						
LS	BOOL	VAR		%Q0.0				
SF11	BYTE	VAR		%QB1	16#80			
SF12	BYTE	VAR		%QB2	16#80			
SF13	BYTE	VAR		%QB3	16#80			

图 6-50 操作权限控制程序的变量表

须注意,用于移位的变量是 SF1、SF2 和 SF3,其初始值需设置为 16#80,使系统启动后,3 个移位变量的初始值都是第 8 位为 1。该设置能保证按下的次数等于计数数值,即按第 1 次,计数数值为 1,按第 2 次,计数数值为 2 等。

6.5.3 长循环移位控制

需要移位的步数很大时,需要多个字、双字或长字之间移位。示例以两个字的移位为例说明编程方法。该控制系统要求移位 21 步,用两个字可最大移位 32 步,程序用计数器计数到 21 步后进行循环。程序见图 6-51,程序中所用变量表见图 6-52。

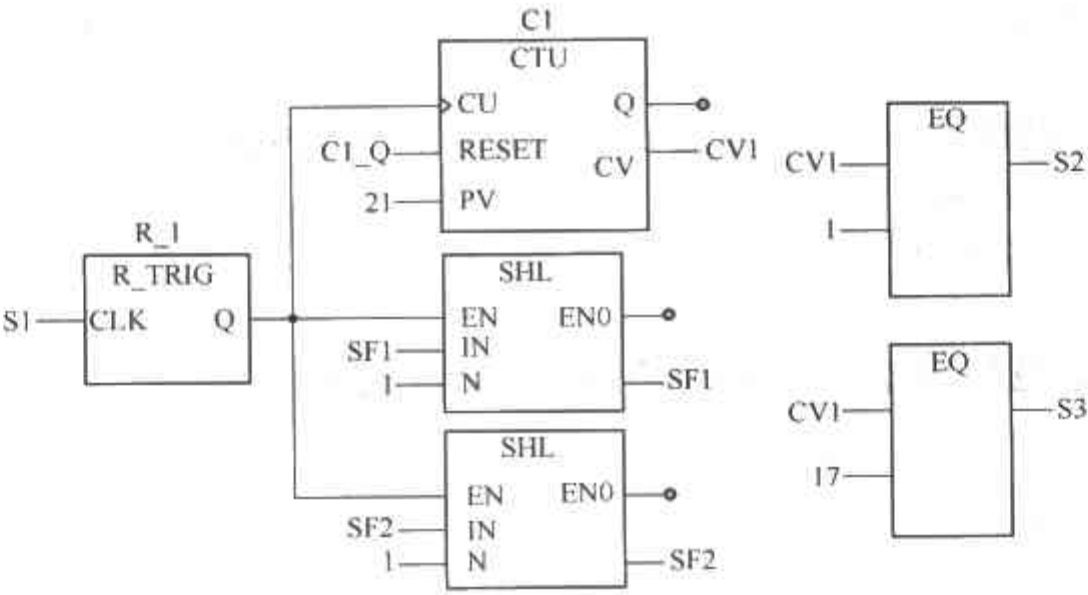


图 6-51 长循环移位控制程序

名称	类型	用法	描述	地址	初值	保持	PDI	OPC
Default								
S1	BOOL	VAR		%IX0.0				
R_1	R_TRIG	VAR						
SF1	WORD	VAR		%QW0				
SF2	WORD	VAR		%QW2				
S2	BOOL	VAR		%QX0.0				
S3	BOOL	VAR		%QX2.0				
C1	CTU	VAR						
CV1	INT	VAR						

图 6-52 长循环移位控制程

程序用 SHL 实现字为单位的不循环左移。用计数器 C1 对移位步数计数。当计数步数为 1 时,用比较函数将第一个字的低位置 1,从而实现初始值的置位。当计数步数为 17 时,将后续字的低位置 1,实现从上一字到下一字的移位操作。当计数值达到循环设置值 21 时,用计数器 C1 的输出进行自复位,实现循环操作。

上述程序也可用于字和双字的移位。例如,当 SF2 采用双字时,就可实现最大 48 步的循环左移位。此外,前后字之间可以不连续。例如,SF1 的地址是 %QW0,SF2 的地址可在 %QW6 等。

类似地,将左移函数改为右移,并对初始值进行更改,可用于长循环的右移。

S2 和 S3 分别是 SF1 和 SF2 的低位地址。例如,SF1 地址是 %QW0,则 S2 的地址是 %QX0.0,SF2 的地址是 %QW2,则 S3 的地址是 %QX4.0。

S1 是移位信号,当上述程序用于时间顺序控制系统时,S1 是各个时间脉冲信号。当上述程序用于顺序功能表图的应用时,S1 是各转换条件信号。

第7章 编程举例

本章着重于应用 IEC 61131-3 去完成工业控制问题的编程。特别是在设计 IEC 61131-3 的基本系统时,应该注意的事项,具体如下:

- 1) 输入和输出接口定义。
- 2) 对控制问题进行分析,并将其分解为较易管理的组件,用于确定采用程序块或采用功能块。
- 3) 如何把控制软件加以划分,使之可在不同的 PLC 资源中运行。
- 4) 选择最适当的编程语言。
- 5) 确定总的配置和组态。

7.1 火力发电厂蒸汽轮机驱动给水泵的控制

7.1.1 过程简介

大型火力发电厂的给水泵用来保持锅炉汽包的水位,使它控制在安全的工作区间。图 7-1 中由锅炉产生的过热蒸汽供给驱动发电机的蒸汽轮机和其他以蒸汽作为动力的厂用设备,譬如驱动锅炉的主给水泵的蒸汽轮机。若汽包的水位太低,会使锅炉水冷壁钢管从炉膛中吸收过多热量,导致过热甚至损坏;若汽包的水位太高,会使过热蒸汽中含有水分,以致损坏主蒸汽轮机的叶片。驱动给水泵的小型汽轮机也从主蒸汽管中获取过热蒸汽作为动力,将从凝汽器回流的脱去气泡的水加压泵至汽包,以维持汽包水位。

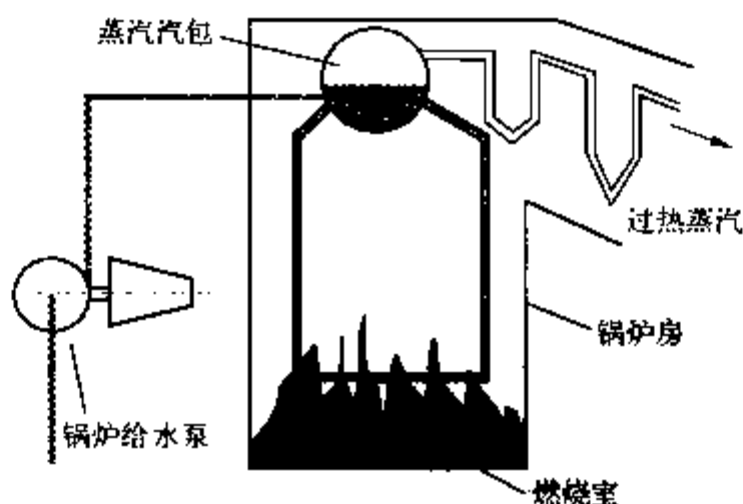


图 7-1 锅炉给水系统概貌

本章叙述的应用举例就是考虑给水泵用蒸汽轮机驱动要解决的控制问题,以及控制系统与给水泵装置中的内其他设备的相互影响和作用的问题。

图 7-2 描述了锅炉给水泵系统的构成。蒸汽轮机的主轴通过联轴节与给水泵的主轴连接,向给水泵提供驱动动力,同时,它还驱动一台油泵,给水泵轴承的润滑油和液压调速器的动

力由该油泵供给。

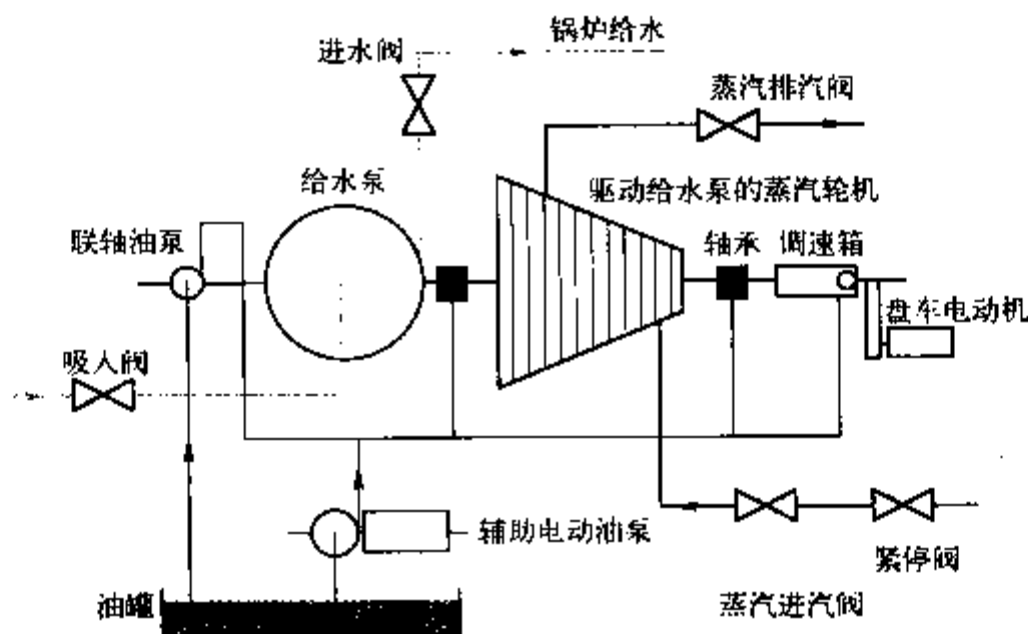


图 7-2 锅炉给水泵系统的构成

操作人员可在中央控制室内完成给水泵起动的初始化,设定给水泵由静止到满速运行的升速时间,以及初始运行速度。在打开蒸汽进汽阀和排汽阀后,蒸汽轮机按指定的斜率升速。为使蒸汽轮机的转子和外壳的温度均衡,汽轮机先升速至某中间转速,并保持此转速运转一段时间,直到蒸汽轮机的外壳温度与进汽温度相适应。然后慢慢开大蒸汽进气阀,让蒸汽量逐步加大,直至蒸汽轮机到达运行速度。

当给水泵处于运行速度(约 3000 转/min)时,打开油泵的负压阀和给水泵的进水阀,将给水输至锅炉的汽包。此时控制系统切换至“随动控制”,由一种专门的算法按照一定的关键的过程参数值调节蒸汽轮机的速度。这些参数值包括汽包的水位、给水流量和进水阀的阀位。该调节控制算法计算出所要求的泵速,通过液压调速器来微调蒸汽轮机的转速。

为了让给水泵停车,操作人员按下停车按钮或处于紧急停车条件时,先关闭供汽管道的蒸汽进汽阀和排汽阀。当蒸汽轮机减速并停转时,起动蒸汽轮机的盘车电动机,让蒸汽轮机慢速旋转,以保证蒸汽轮机的轴在冷却过程中不致变形。

当蒸汽轮机轴承的液压低到最低压力时,或当蒸汽轮机的转速过低,以至不能带动油泵的轴转动时,为维持轴承的液压,要起用辅助电动油泵。

7.1.2 设计方法

可以把给水泵控制系统的设计考虑分为下列阶段:

- 1) 定义控制系统的输入和输出,即控制系统与外部之间的接口,包括来自传感器的输入和连接至执行器(诸如阀门和齿轮箱)的输出。
- 2) 定义控制系统与蒸汽轮机拖动给水泵装置的其他部分需要交换的主要信号。
- 3) 定义与操作人员的所有进行交互的接口、人控功能和监控数据。
- 4) 采用由上至下的方法分析控制问题,由此可得到按 IEC 61131-3 标准所要求的程序组织单元,即 IEC 程序和功能块。
- 5) 定义所要求的任意低级功能块。
- 6) 定义不同程序和功能块所要求的扫描循环时间。

7) 进行程序和功能块的详细设计。

图 7-3 表示控制系统与蒸汽轮机拖动给水泵装置的主要接口。基本的功能是调节给水泵的转速,以保持锅炉汽包的水位在给定的数值。不过,整个控制系统还需要监控 100 多个输入信号(包括阀位传感器、温度、油压等),以确定整个装置的状态。另外还有许多输出,用来调整蒸汽阀门的开度,以及辅助电气装置。

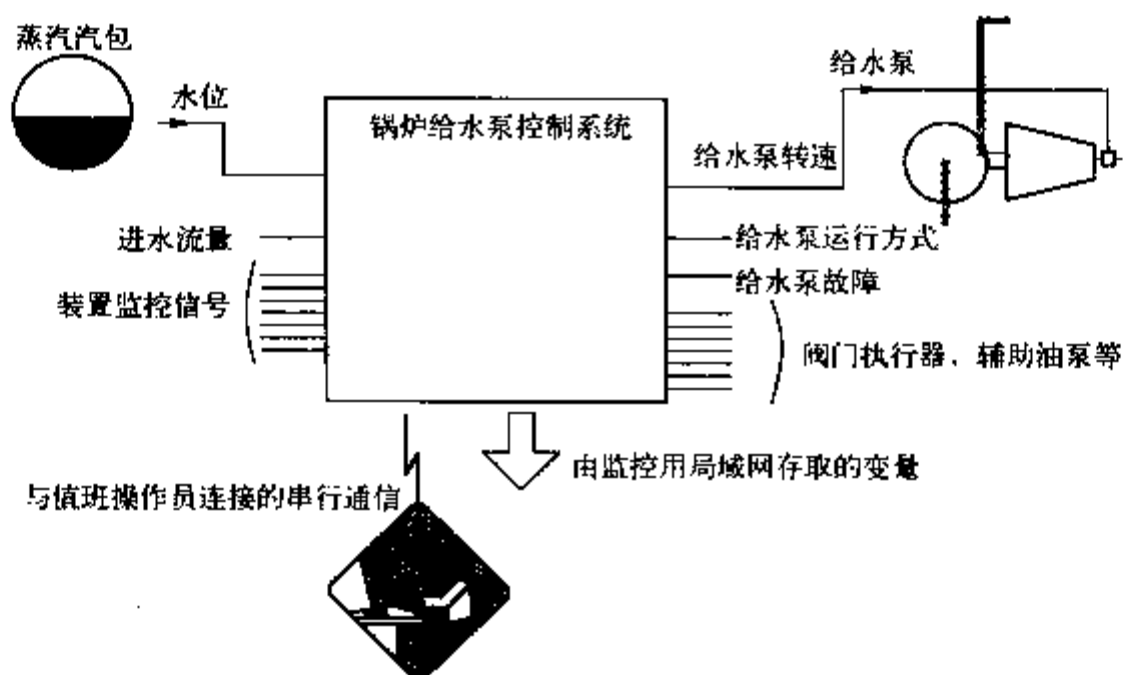


图 7-3 锅炉给水泵控制系统概貌

1. 数据类型的定义

在每一个项目中都会要求指定数据类型,以某种格式来表示与局部的问题范畴相兼容的数据。因此一定要定义数据类型,使这些数据可以在整个项目中加以应用。

【例 7-1】 定义给水泵运行模式的数据类型。

```
TYPE PumpMode :(  
    NotAvailable,      (* 给水泵不可运行 *)  
    Stopped,           (* 给水泵停车 *)  
    Barring,           (* 给水泵以盘车转速旋转 *)  
    Ramping,           (* 给水泵转速正在上升 *)  
    Running            (* 给水泵以运行转速运转 *)  
)  
END_TYPE
```

【例 7-2】 定义给水泵转速提升的数据类型。

```
TYPE RampSpec  
    Target : REAL;      (* 给水泵提升的目标转速 *)  
    Duration : TIME;    (* 给水泵提升的时间间隔 *)  
END_TYPE
```

2. 定义给水泵装置的输入和输出

所有与给水泵运行有关的输入传感器的信号(包括蒸汽阀的阀位、油压、齿轮变速器和温

度等)都需要定义。同样,还应定义所有的驱动执行机构(诸如阀门控制器的执行机构和齿轮变速箱的执行机构,在起动辅助油泵时的输出)的主要输出信号。虽然在项目的起始阶段只能定义主要的 I/O 信号,但在项目的控制要求已很清晰的时候,还需要增添附加的 I/O 信号。

理想的策略是规定一个命名规则,使从信号名字的本身就可以看出每个信号的来源和目的。每个信号都应该定义其数据类型以及能清楚地分辨出是系统的输入还是输出。在一个大型项目中要建立一个数据库用来保存和存储所有的 I/O 信号的信息。

信号值读入或向外部设备发送的输出信号的方式因 PLC 而异。在本例中假定 PLC 的硬件配置是将 I/O 值存放在已预先确定的存储空间。

【例 7-3】 定义系统输入。

```
P1_Local AT %IX1.0 ; BOOL;      ( * 就地控制 * )
P1_PumpSpeed AT %ID50 ; REAL;    ( * 给水泵转速 * )
P1_FlowRate AT %ID51 ; REAL;    ( * 给水流量 * )
P1_DrumLevel AT %ID52 ; REAL;    ( * 蒸汽汽包水位 * )
P1_CasingTemp AT %ID53 ; REAL;  ( * 蒸汽轮机机壳温度 * )
.....
```

【例 7-4】 定义系统输出。

```
P1_SteamIV AT %QD80 ; REAL;     ( * 蒸汽进汽阀阀位 * )
P1_SteamEV AT %QX81 ; BOOL;     ( * 排汽阀 * )
P1_SuctionV AT %QX10 ; BOOL;    ( * 虹吸抽水阀 * )
P1_DischV AT %QX11 ; BOOL;      ( * 锅炉进水阀 * )
.....
```

前缀“P1_”表示与 1 号泵有关的信号。这是为今后可能的扩建考虑的,一旦要增加 2 号泵的控制系统,那么与之相关的信号均可以“P2_”为前缀。

3. 外部接口的定义

虽然前面所举的例子着重于专门对给水泵进行控制,不过也不能忽略对装置的其他部分的接口加以定义,其他控制系统也是系统设计的重要部分。给水泵的控制系统一般都有来自其他系统的硬接线信号,还有一个甚至多个通信接口。必须定义这些信号和所有通信接口的相互作用信号。

允许经由装置的局域网进行存取的变量,可以用 VAR_ACCESS 结构予以说明。

【例 7-5】 定义直接主控制输入。

```
P1_StartPemit AT %IX200 ;
                BOOL;      ( * 允许给水泵起动的主信号 * )
P1_Local AT %IX201 ;
                BOOL;      ( * 给水泵紧急停车信号 * )
.....
```

【例 7-6】 定义主控制输出。

```
P1_PumpMode AT %QW400 ;
PumpMode;      ( * 给水泵运行方式 * )
```



```

PI_Local AT %QW401 ;
                                BOOL;                ( * 给水泵运行故障 * )
.....

```

用于通信网络的存取路径变量也需要定义。

【例 7-7】 定义至锅炉 1 号泵转速的存取路径变量

```

AI_BFPSpeed  : PI_PumpSpeed; REAL  READ_ONLY
( * 至锅炉 1 号泵运行速度的存取路径变量,只读属性 * )
AI_BFPMode   : PI_PumpMode; PumpMode READ_ONLY
( * 至锅炉 1 号泵运行模式的存取路径变,只读属性 * )
.....

```

4. 人机交互作用的定义

操作人员通过直接接线的开关和指示灯、指示器之间的交互作用,可以用一般的输入和输出信号定义。

但是通过通信接口的交互作用可能要求采用特定的通信功能块。例如,如果采用专用的串行通信接口,可能要专门开发一个用 C 语言编写的低层次的功能块。在 IEC 61131-3 标准中不涉及串口通信问题。

上面的例子中假定已经开发了一个名为“Operator_Data_Request”的低层次功能块。它仅为操作员提供一种远程面板或显示站的接口。它向操作员发送消息,在得到响应后再发送信号。该功能块有两个标志:“Message_Ready”和“Data_Valid”,在操作员的消息可被刷新和当操作员送入的数据为有效后,这两个标志作为一种信号标志发出。图 7-4 是该功能块的图形符号。

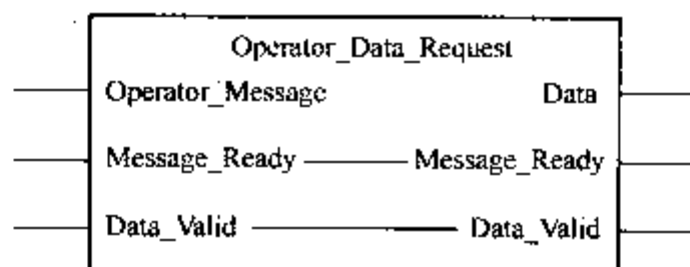


图 7-4 Operator_Data_Request 功能块的图形符号

该模块用于蒸汽轮机升速运行顺序中请求蒸汽轮机按线性比例升速的速度和升速过程所需的时间。

注:“Message_Ready”和“Data_Valid”是既提供输入、又提供输出的 VAR_IN_OUT 变量的例子。

7.1.3 控制问题分解

在上述给水泵的例子中,主要控制功能可分为两部分:顺序控制和辅助控制;随动控制。这里假定 PLC 提供两个处理资源。主资源 Main 运行汽轮机升速的主顺序控制,以及控制所有的辅助功能。第二个资源 ModulatingControl 对模拟量输入信号进行扫描采样,并运行随动控制算法。

在这两个资源内的控制软件均封装在程序块 MainControl1 和 ModControl1 中。整个控制软件均包含在配置 BFPI_Control 内,如图 7-5 所示。

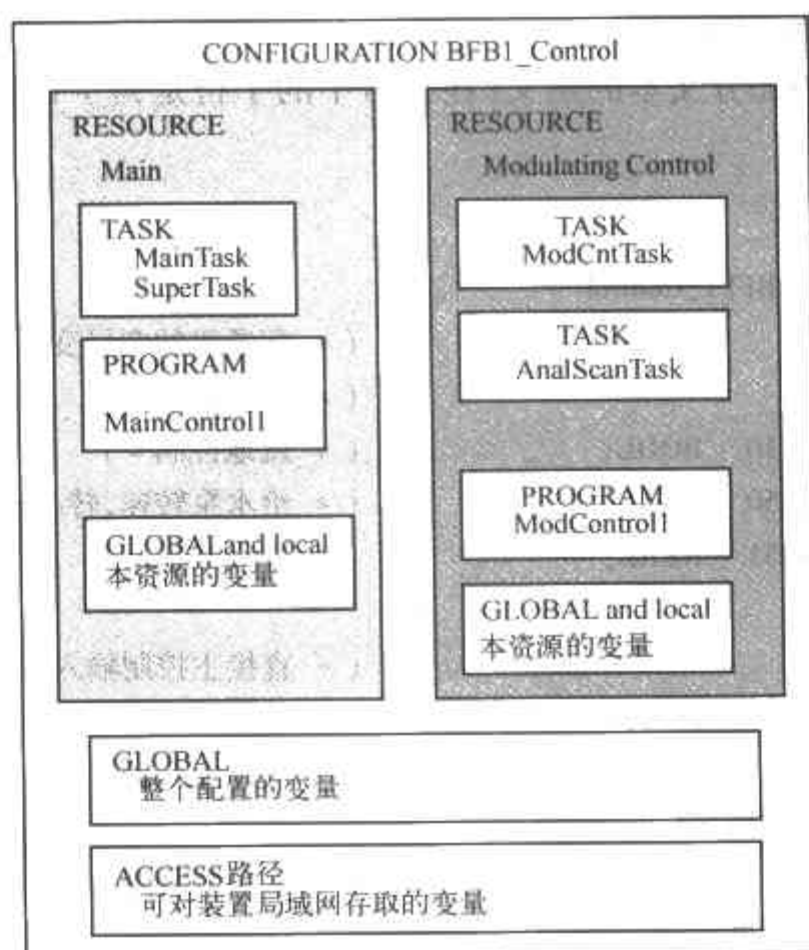


图 7-5 系统配置的结构层次

在两个资源中所用到的全局变量在配置这一级定义。这些全局变量既有在两个资源都要求、且表示系统的输入和输出,也有那些作为两个资源之间交换的信息。例如,当给水泵已升至设定的运转速度时,MainControl1 发出信号立即起动 ModControl1。该信号是由在主资源 MainControl1 程序中设置的一个全局布尔变量 PI_Auto 实现。

至于那些仅在一个特定资源内所要求的全局变量,则在资源的级别中加以说明。例如,常数 MaxPumpSpeed 仅与 MainControl1 程序有关,因此只要在主资源 Main 中作为全局变量来定义。在那些仅在资源内用到的全局变量,就在该资源内定义,这将有助于规划整个设计。

由于对所有低级别的软件存取全局变量并无限制,所以,应让全局变量尽可能少。在本例中,所有关键的变量都经由正式的输入和输出进入和导出程序。这样做的结果使软件的设计具有良好的结构性,但也会导致在主程序中存在大量的输入/输出变量。

说明存取路径是为了对所选择的变量提供外部路径。在这里,通信存取要限于少数主要变量,诸如给水泵的转速和运行方式。为了监控起见,对另外一些变量也可设置只读路径,诸如报警监控的变量。不应提供对内部控制变量的直接存取方式,因为偶尔的突发因素有可能会对控制系统的完整性产生危险。

此外,还假定,超出 IEC 61131-3 范围以外的诊断手段,在系统调试期间为建立各种整定常数也是很有用的,理当予以考虑。

每个资源包含一个或多个任务定义。任务 MainTask 运行主程序块和大多数在资源 MainControl1 内的功能块。另一个任务 Supertask 是与给水泵运行的监控功能块有关的。由于这个功能块都是用来监控慢速变化的输入数据,诸如温度和振动幅度等,因此运行时采取慢扫描的速率。

在 ModulatingControl 资源中有一个名为 ModCntTask 的任务,由它承担随动控制功能块和

模拟量输入信号调理功能块的运行。

下面给出这一配置的部分文本的定义(在括号中的字符是为了让读者能清晰地了解而附加的说明语句)。

【例7-8】 定义配置。

```
CONFIGURATION BFP1_Control
  VAR_GLOBAL
    (* 配置级的全局变量 *)
    (* 系统输入 *)
    P1_Local AT %IX10 : BOOL;      (* 就地控制 *)
    P1_Local AT %ID50 : REAL;      (* 给水泵转速,转/分 *)
    P1_Local AT %ID51 : REAL;      (* 进水流量 *)
    .....
    (* 直接主控制输入 *)
    P1_StartPermit AT %IX200 :
    BOOL;                          (* 允许主起动 *)
    P1_PumpTrip AT %IX201 :
    BOOL;                          (* 给水泵紧停 *)
    P1_StartUp AT %IX202 :
    BOOL;                          (* 给水泵运行起动 *)
    .....
    (* 系统输出 *)
    P1_SteamIV AT %QD80 : REAL;    (* 蒸汽进气阀阀位 *)
    .....
    (* 主控制输出 *)
    P1_PumpMode AT %QW400 :
    PumpMode;                      (* 给水泵运行模式 *)
    P1_PumpFault AT %QW4010 :
    BOOL;                          (* 给水泵运行故障 *)
    .....
    (* 配置级,共享全局变量 *)
    P1_Auto : BOOL;                (* 随动控制自动运行 *)
  END_VAR
  RESOURCE MainControl
    VAR_GLOBAL CONSTANT
      (* 定义所有的全局常数 *)
      Seq_Scan_Period : TIME := T#100ms;
      MaxPumpSpeedI : REAL := 4000.0;
      .....
    END_VAR
  VAR
    (* 资源级变量 *)
    (* 系统输入 *)
    P1_AOP_Press AT %ID533 : REAL; (* 辅助油压泵压力 *)
```

```

.....
( * 系统输出 * )
P1_SteamEV AT % QX81 : BOOL;          ( * 排气阀 * )
P1_SuctionV AT % QX10 : BOOL;          ( * 虹吸抽水阀 * )
P1_DischgV AT % QX81 : BOOL;          ( * 锅炉进水阀 * )
.....
END_VAR
( * 任务定义 * )
TASK MainTask( INTERVAL := Seq_Scan_Period,
               PRIORITY := 5)
TASK SuperTask( INTERVAL := T#500ms,
               PRIORITY := 10)

( * 调用 MainControl1 程序 * )
PROGRAM MainControl1 WITH MainTask :
    MainControl(
    ( * 输入变量连接清单 * )
    StartUp := P1_StartUp,              ( * 给水泵运行 * )
    Local := P1_Local,                  ( * 就地控制方式 * )
    PumpSpeed := P1_PumpSpeed,
    FlowRate := P1_FlowRate,
    PumpTrip := P1_PumpTrip,
    .....
    ( * 输出变量连接清单 * )
    Auto => P1_Auto,
    PumpMode => P1_PumpMode,
    PumpFault => P1_PumpFault,
    SteamIV => P1_SteamIV,              ( * 蒸汽进气阀 * )
    SteamEV => P1_SteamEV,              ( * 蒸汽排气阀 * )
    SuctionV => P1_SuctionV,            ( * 虹吸抽水阀 * )
    P1_DischgV => ( P1_DischgV,          ( * 锅炉进水阀 * )
    .....
    ( * 将监控功能块赋予任务 SuperTask * )
    PumpSupervision With SuperTask;
    );
END_RESOURCE
RESOURCE ModulatingControl
.....( * 从略 * )
END_RESOURCE
VAR_ACCESS
( * 通过通信网络可存取的存取变量 * )
A_BFP1Speed : P1_PumpSpeed : REAL READ_ONLY;
A_BFP1Mode : P1_PumpMode : PumpMode READ_ONLY;
A_BFP1Suction :

```

```
MainControl. P1_SuctionV ; BOOL READ_ONLY;  
.....  
END_VAR  
END_CONFIGURATION
```

注：一个大型系统的配置可能有几百行。但许多符合 IEC 61131-3 的编程系统，常可用编程站自动完成。

7.1.4 程序分解

本节将以更近的视角来分析给水泵控制系统中用到的一个程序块。注意到 MainControl1 是程序类型 MainControl 的一个实例。这样一来，今后如果要进行系统扩建，例如增加第 2 台给水泵的控制系统时，只要简单地建立第 2 个程序实例 MainControl2，并将这个新的实例与第 2 台给水泵相关的输入和输出链接起来即可。

程序类型 MainControl 包括许多功能块实例，如图 7-6 所示。

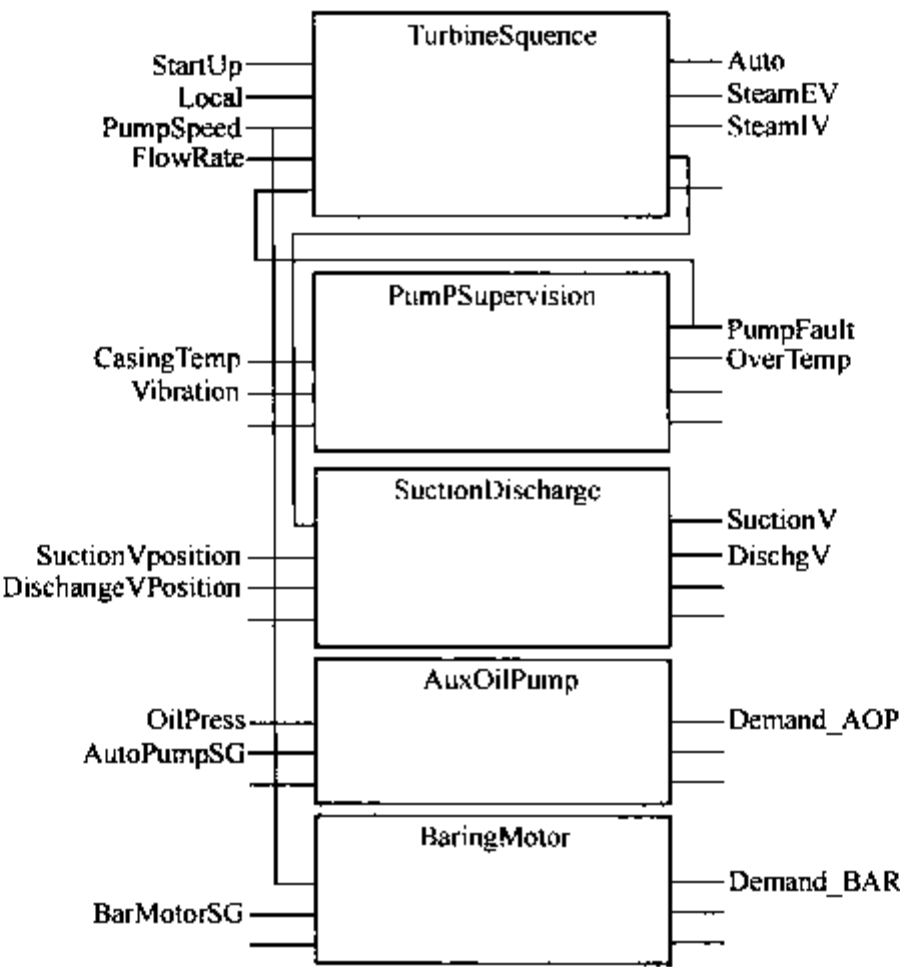


图 7-6 MainControl 程序的分解

注：在表达程序结构时，采用功能块图语言 FBD 是很理想的。为表达主程序的结构，在此图中仅标出了最基本的信号。为避免造成不必要的混乱，在功能块中也未给出输入和输出。在实际系统中每个功能块可能有 30 至 40 个输入输出。

在图 7-6 中的每个功能块都与锅炉给水泵的某个特定部分的控制有关。只要可能，都应该把这个特定部分的行为特性的控制逻辑包含在这个特定的功能块内。

在 MainControl 程序内的每个顶层功能块的行为特性说明如下。

1. 蒸汽轮机顺序起动功能块 TurbineSequence

此模块包含蒸汽轮机的主升速顺序。在收到 StartUp 信号，即作出相应响应，按功能块所

设定的起动顺序,经过几个步序使给水泵升速至运转速度。这些步序是:向操作员请求升速的时间和目标运转速度;打开蒸汽进汽阀、排汽阀;在升速过程中让汽轮机保持在某个中间转速,直到蒸汽轮机的外壳温度稳定在一定数值;继续升速,当给水泵达到给定的转速时发出 Auto 信号。该 Auto 信号触发随动控制程序来承担蒸汽轮机的速度控制。

本蒸汽轮机顺序起动功能块 TurbineSequence 基本包括将蒸汽轮机的转速升至其运转速度所需要的主要步序。其行为特性可以用顺序功能图 SFC 给以最适当的描述。图 7-7 是蒸汽轮机起动升速顺序的一个部分。

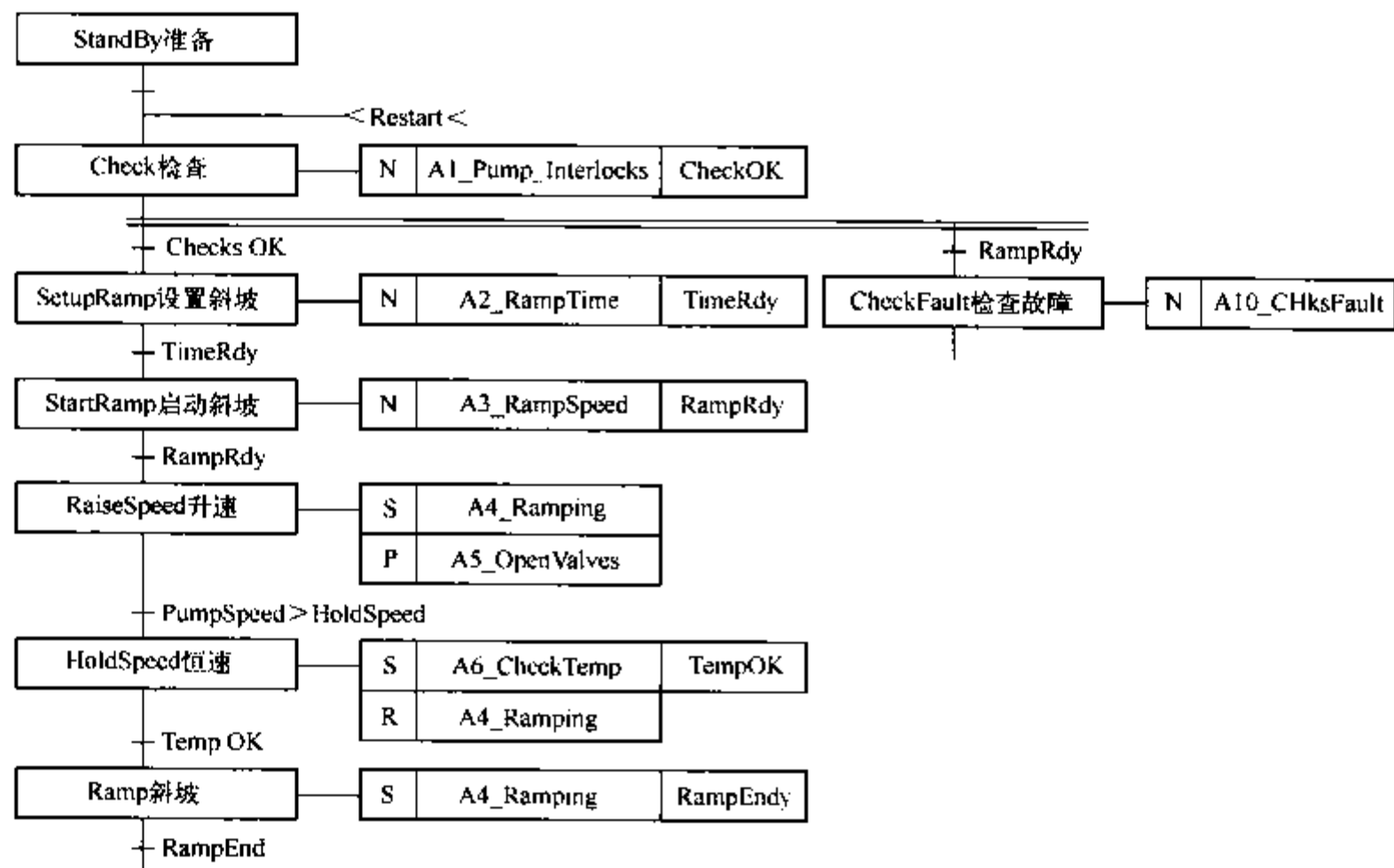


图 7-7 蒸汽轮机顺序起动功能块的顺序功能表图

由图 7-7 可以清晰地了解,从蒸汽轮机处于备用状态 StandBy(这时其初始态)开始,当接到起动信号 StartUp 时,立即转至检查状态 Checks。在此状态执行动作块用来检查给水泵的各种连锁情况,如果一切正常即发出 ChecksOK 信号,转移至下一个状态 SetupRamp;否则发出 NOT ChecksOK 信号,转移至 ChksFault 状态。在转移至 SetupRamp 状态时执行动作块 A2_RampTime,当升速时间设置好即发出 TimeRdy 信号,用以表示转移至下一个状态的条件成立。下一个状态是 StartRamp,对应的动作块则是 A3_RampSpeed,当升速目标转速设置好,即发出 RampRdy 信号,用作向再下一个状态 RaiseSpeed 转移的条件。在此状态下,执行的动作块 A5_OpenValves 开启相应的阀门(蒸汽紧停阀和排气阀),进行蒸汽轮机的升速(A4_Ramping 置位)。当转速达到预先设定好的中间转速 HoldSpeed,即发出相应的信号向下面的状态 Hold-Speed 转移。在这个状态下,蒸汽轮机一直以这个转速运行(即不执行升速,动作块 A4_Ramping 复位),直到此状态下的另一个动作块 A6_CheckTemp 得到蒸汽轮机外壳温度已趋于稳定的结果,发出 TempOK 信号。以此作为向下一个状态 Ramp 转移的条件成立。在 Ramp 状态下执行的动作块又将 A4_Ramping 置位,蒸汽轮机继续升速,一直到转速达到所设定的运转速

度,即发出升速顺序结束的信号 RampEnd。

要注意的是,如果接到 Restart 信号,直接进入初始状态后的检查状态 Checks。

顺序功能图 SFC 中所要求的动作块可以用 IEC 61131-3 中规定的任一种语言来编写。例如,在图 7-8 中给出的梯形图有关动作块 A3_RampSpeed 的控制逻辑。为向值班操作人员提示,要键入蒸汽轮机起动升速所需的目标转速,将它用作功能块 Operator_Data_Request 的一个实例。当 StartRamp. X 信号置“1”则 StartRamp 步被激活,而 StartRamp. X 信号的上升沿用来使消息准备好信号 Msg_Rdy 置“1”。这触发 Operator_Data_Request 功能块向值班操作员发出提示“Ramp Speed”(“升速目标转速”)。

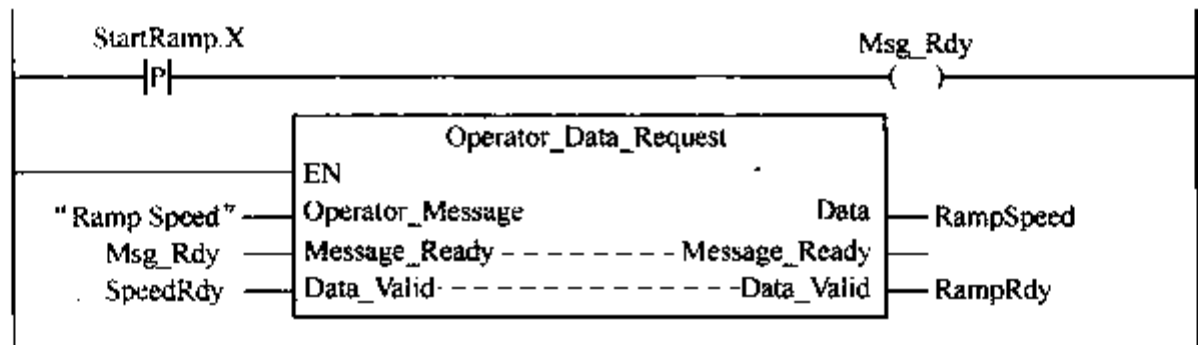


图 7-8 A3_RampSpeed 动作控制功能模块

功能块设置了斜坡准备信号 RampRdy,一个指示器变量在下面步的转换时被测试。同样,在 A5_OpenValves 的动作控制功能模块可采用下列 ST 语句描述:

```
ACTION A5_OpenValves
  StEmergStopCtl ( Position := OPEN, Time := T#15s );
  StExhaustCtl ( Position := OPEN, Time := T#15s );
END_ACTION
```

此动作块调用两个功能块 StEmergStopCtl 和 StExhaustpCtl,由它们将蒸汽紧停阀和蒸汽排气阀打开,为蒸汽轮机进行起动升速做好准备。这两个功能块都是阀门控制功能块 Valva-Control(在下一节有进一步的叙述)的两个实例。

2. 给水泵监控功能块 PumpSupervision

本功能块涉及对蒸汽轮机和给水泵进行检查,以监控它们是否运行在正常的限定范围之内。模块要连续地检测轴承温度、振动幅度,以及蒸汽轮机外壳的温度,以确保这些参数均处在可接受的范围。任意一个参数若偏离正常范围,立即发出给水泵故障输出信号 PumpFault。这一例行检查由蒸汽轮机顺序起动功能块 TurbineSequence 进行,若发生故障可以暂停蒸汽轮机的升速顺序。

这个模块连续检测给水泵和蒸汽轮机的温度和振动测量值,并检查是否处在运行要求的范围之内。图 7-9 中示出了用功能块图 FBD 语言编写的这个功能块的程序的一部分。不过,如果用梯形图语言 LD 或结构文本语言 ST 编写的程序,同样也可以适用。

由于此模块仅涉及检测变化相当慢的输入,所以它可以用比主控制程序 MainControl 中的其他模块要慢的扫描速率运行。为此,专门在本例的配置定义中,把与给水泵监控功能块 PumpSupervision 相关联的任务 SuperTask 调用的扫描周期指定为 500 ms。

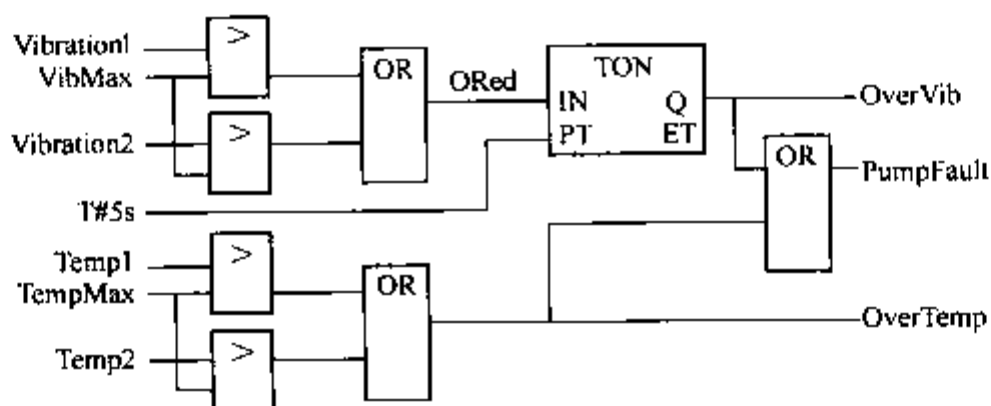


图 7-9 用功能块图描述 PumpSupervision 功能块

3. 给水泵抽水给水功能块 SuctionDischarge

在给水泵转速接近其运转速度时,锅炉给水的虹吸抽水阀和进水阀打开。此模块确保这些阀门仅在达到规定的状态下开启;而且也要检查这些阀门的阀位传感器,以确保阀门已开到所要求的位置。

图 7-9 虽然只表示了整个给水泵监控功能块 PumpSupervision 程序的一部分,但却是最核心的部分。该程序实现了当有两个输入振动信号时,先分别与振动信号容许的上限值比较,只要其中有一个大于上限值(两个作大于比较的函数的输出进行或运算),即发给接通延迟定时器,这确保振动超限保持了 5 s 后立即引发给水泵故障信号。对于温度信号超限,也是同样方法处理。

4. 辅助油泵功能块 AuxOilPump

让蒸汽轮机和给水泵的轴承内的油压一直维持一个良好的压力,是保证它们处于正常状态的关键。本模块连续的检测油压,检测它是否总高于规定的最小临界值。如果油压跌至临界值以下,且持续了几秒钟,辅助的电动油泵就立刻上电起动。模块还要检查辅助油泵的减速齿轮箱,以确保它按要求运转,并且辅助油泵以规定的压力向轴承供油。

5. 盘车电动机功能块 BarringMotor

当给水泵停止运转时,蒸汽轮机要按一定规程减速,直至停转。此时必须让盘车电动机与蒸汽轮机的转轴啮合,使汽轮机以一个适当的低速旋转,以避免蒸汽轮机轴变形。本功能块要确保在收到来自功能块 TurbineSequence 的信号的请求时,让盘车电动机与蒸汽轮机的轴啮合,并起动电动机。此外,本模块还要执行当蒸汽轮机升速以致进入正常的升速顺序后,切断盘车电动机的电源,让该电动机与汽轮机的轴断开啮合。

7.1.5 低层次功能块

在前面所举的例子中,有许多阀门要控制,除了蒸汽进气阀是按比例开启之外,其他阀门都可以用同样的阀门控制器功能块予以控制。

在由来自控制系统的信号驱动阀门执行器去打开或关闭阀门时,必须在预定的时间内让阀门到位。每个阀门上都安装了限位开关,当阀门达到全开或全关位置时,这些限位开关都会发出信号。

功能块 ValveControl 提供阀门移动至所要求位置的全部逻辑控制,检查阀门是否到位。如果没有到位,立即产生一个故障信号指示所要求的阀门移动未曾检查到。

图 7-10 示出了功能块 ValveControl 与阀门执行器的连接。阀门减速箱上的限位开关将

所检测到的阀门位置反馈给功能块 ValveControl。

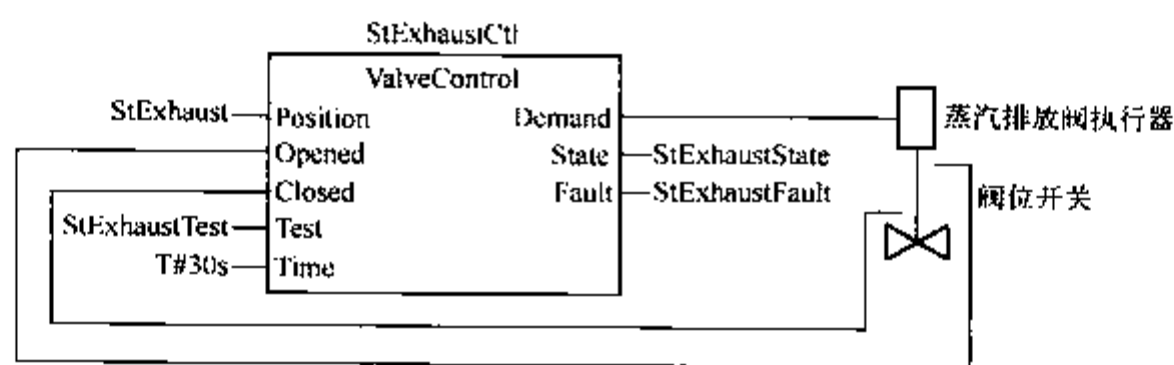


图 7 - 10 阀门控制功能块 ValveControl

对给水泵控制系统还要考虑许多其他的功能块,诸如起动辅助油泵或盘车电动机等电气装置都会用到齿轮箱离合控制的功能块。可以事先设计好类似的模块,以便于今后可用在其他的系统中。

7.1.6 信号流

因为用基于 IEC 61131-3 编程语言标准的编程系统所编制的软件可以深度嵌套,所以信号值从源头发至目的地,也许会经过多个层次的流动。这样,一个特定信号的数值就有可能在不同的层次以不同的变量名命名。

下面的例子是考虑蒸汽排气阀所要求的信号,从其源头(阀门控制功能块 ValveControl 名为 StExhaustpCtl 的实例)的流动。此功能块存在于 TurbineSequence 功能块内,并直接从其输出请求 Demand 驱动 TurbineSequence 的输出 SteamEV。而按层次的次序 TurbineSequence 是程序 MainControl1 内的一个实例, TurbineSequence 功能块的输出 SteamEV 直接驱动程序 MainControl1 的输出 SteamEV。结果,程序 MainControl1 的输出 SteamEV 的值写入资源级变量 P1_SteamEV 中。图 7 - 11 示出了从功能块 StExhaustpCtl. Demand 流出的信号到达了资源级的变量 P1_SteamEV。

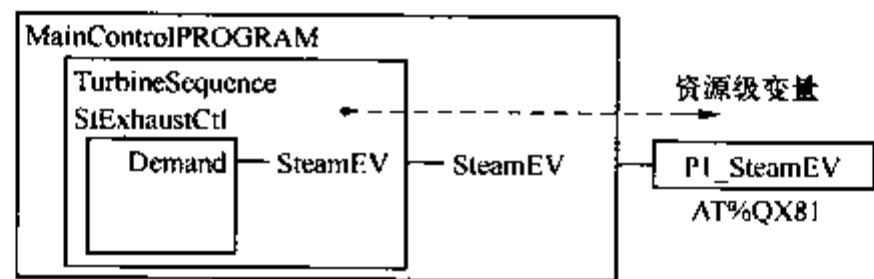


图 7 - 11 信号流示例

在调试基于 IEC 61131-3 标准的程序时,要注意对每个变量的前后(或上下)关系和作用域。例如,变量 SteamEV 作为 TurbineSequence 功能块的输出,不应与程序 MainControl1 的输出 P1_SteamEV 混淆。

同样也存在由系统的输入项内部功能块流动的信号流。

虽然在大系统中也可能存在许多软件层次,但这些层次不应造成使系统性能变差的影响。一种良好的编译设计足以保证在信号值经过软件模块的界面传递时,不会附加任何显著的处理开销。

7.2 分选器控制

本示例用于说明如何进行顺序功能表图的编写。

7.2.1 分选器过程简介

为分选不同大小的工件,可采用分选器。它从大的工件中选出小工件,将大工件送到右侧,小工件送到左侧。从结构观点看,它是由两个集成的基本单元——皮带传送机和旋转平台组成。如图 7-12 所示。

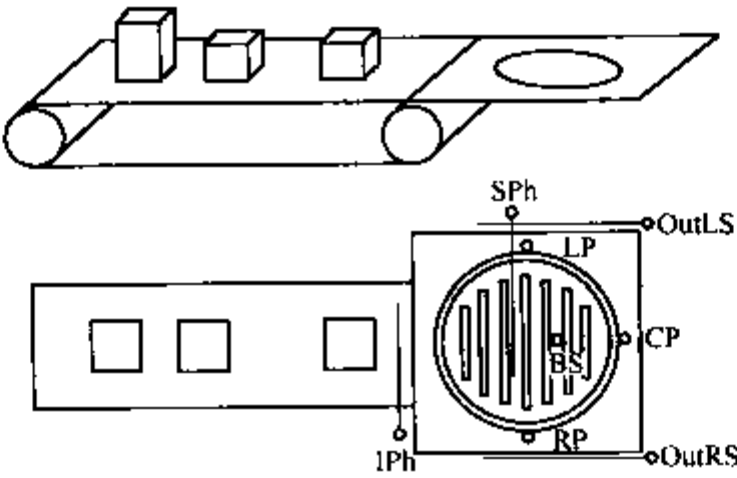


图 7-12 分选器工艺过程示意图

电动机驱动传送皮带,将被检盒状工件前移至旋转平台。该电动机停转,直到旋转平台准备对下一个工件进行分拣。旋转平台由两个电动机交替操作:一个电动机带动平台旋转,使该旋转平台接收被拣工件和选择工件往哪个出口方向送出。另一个传送电动机则移动辊道确保送入工件定位在平台的中间位置,并在确定工件的出口方向后将其推至已选定的出口。当旋转平台上无工件,或工件越出平台进给边界时,这两个电动机停转,并将旋转平台的方向对准于中间位置,准备接收新的工件。

工件的大小和位置选用适当的光电传感器来识别。左向出口和右向出口分别设有检测传感器 LP 和 RP,用来确认工件是否到达所选定的方向。

当工件从传送带进入旋转平台的边界,光电检测元件(IPh)检测到工件进入旋转平台的事件,接着工件被送入旋转平台,开始加载。这时旋转平台上的辊道传送电动机启动,把工件移至平台中间。定位信号由位置传感器 BS 检测。当该传感器确认工件已处在中间位置,立即让旋转平台上的辊道传送电动机停转。

由光电检测元件 SPh 检测工件的高度,并按此信号选择出口方向。此时启动平台旋转电动机,根据被选择的方向电机正转或反转,当到达所选的方向,并由传感器(LP 或 RP)检测到,立即停止旋转电动机,启动辊道传送电动机,把工件推移至所选择的出口方向。当相关的出口限位开关(OutLS 或 OutRS)动作,表示工件离开平台,卸载完成,平台的辊道传送电动机停止,而旋转电动机重新启动,将平台旋转到中央位置,中央位置由 CP 传感器检测。

7.2.2 程序设计和优化

1. 定义输入输出变量和数据类型

需要用于描述选择性分选器控制的变量列于表 7-1 中,它包含变量的名称、数据类型和

变量描述。

表 7-1 变量表

变量名	数据类型	描 述	变量名	数据类型	描 述
IPh	BOOL	检测有无工件的光电传感器	OutLS	BOOL	工件已移出左出口的位置传感器
SPh	BOOL	检测工件大小(高度)的光电传感器	OutRS	BOOL	工件已移出右出口的位置传感器
BS	BOOL	工件处于平台中间的位置传感器	输出变量		
LP	BOOL	工件左出口位置传感器	ConvM	BOOL	皮带传送机电动机运行控制信号
RP	BOOL	工件右出口的位置传感器	PlatTM	BOOL	平台辊道传送电动机运行控制信号
CP	BOOL	平台转向中间位置的位置传感器	PlatRM	{-1,0,1}	平台旋转电动机反、停和正转控制信号

对输出变量 PlatRM,假设有 3 个值: -1,0 和 1,分别表示电动机向左转(逆时针),停止和向右转(顺时针)。实际应用时,采用两个布尔变量 PlatRML、PlatRMR,这两个变量为 1 分别表示左转和右转,同时为 0 表示停转。

为清晰表达工件已移出旋转平台,特采用两个信号传送单元,分别用来表示已接收并传送了高和低的工件,并用准备(真)和咬合(假)命名。它们是分拣器的输入信号。

2. 控制系统设计

(1) 状态图

为简化,这里仅考虑分拣器行为的主要概念。

根据上述,分拣器行为表示为下列两个阶段的交替:

- 1) 向前的阶段:当工件跨越传送带与旋转平台的边界,它的高度被检测并被推向合适的出口方向。
- 2) 向后的阶段:旋转平台返回到中央位置,传送带再次运行。

图 7-13 表示该分选过程的状态图和两个阶段,其中,实线表示向前的阶段,虚线表示向后的阶段。

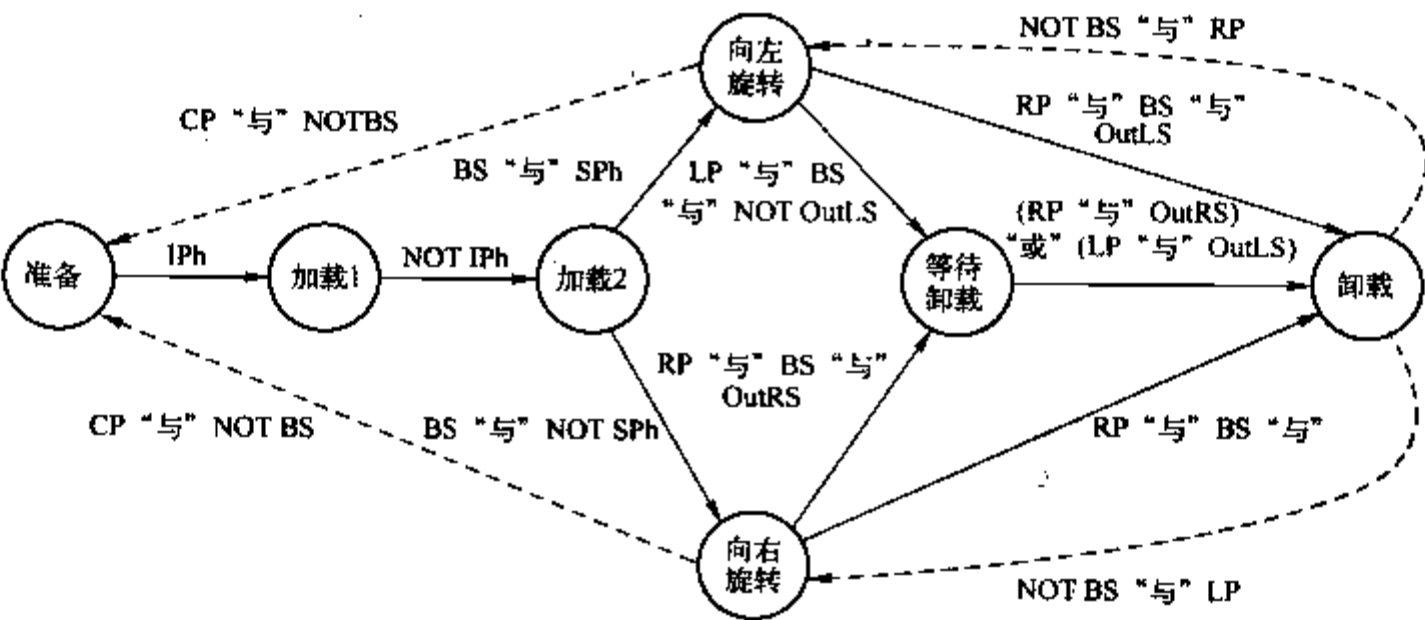


图 7-13 分拣器状态图和两个阶段

状态图确定 7 个有意义的系统状态,用圆表示,其主要意义如下:

- 1) Ready:该状态时,皮带输送机开始运行,而旋转平台电动机停止。
- 2) Loading1:该状态表示工件跨越传送带与旋转平台时的状态,为了将工件送到平台,传送电动机运转是必要的。
- 3) Loading2:一旦工件已经完成装载到旋转平台,皮带输送机电动机就应停止直到平台的滚辊继续工作。
- 4) LRotation:左转,如果是大工件,则在工件达到平台中央和传送电动机停止后起动,另一个左转是在反向阶段,它移回平台从右面到中央位置(准备位置)。
- 5) RRotation:右转,如果工件是小的,在工件到平台中央和传送电动机停止后起动,另一个右转是在反向阶段,它移回平台从左面到中央位置(准备位置)。
- 6) Unloading:工件已经到达最终传送的位置。因此,平台旋转电动机停止,传送电动机开始运转,系统保持该状态直到工件完全离开平台。
- 7) WaitUnloading:该状态在工件已经到达最终传送方向,但是信号显示下一单元尚未到达可卸载状态。系统将等待直到信号变化,然后才能将工件转入卸载状态。

(2) 状态、转换条件和动作块的确定

根据事件确定状态的传递。一般可用合适的输入变量(和可能的工作)组合来表示。表7-2说明了状态和转换条件的关系。

表 7-2 分选器状态和转换条件的关系

原 状 态	新 状 态	转 换 条 件	原 状 态	新 状 态	转 换 条 件
Ready	Loading1	IPh	Loading1	Loading2	NOT IPh
Loading2	LRotation	BS AND SPh	LRotation	Unloading	LP AND BS AND OutLS
Loading2	RRotation	BS AND NOT SPh	RRotation	Unloading	RP AND BS AND OutRS
LRotation	WaitUnloading	LP AND BS AND NOT OutLS	Unloading	LRotation	NOT BS AND RP
RRotation	WaitUnloading	RP AND BS AND NOT OutRS	Unloading	RRotation	NOT BS AND LP
LRotation	Ready	CP AND NOT BS	RRotation	Ready	CP AND NOT BS
WaitUnloading	Unloading	(RP AND OutRS) OR (LP AND OutLS)			

为完整表示系统的行为特征,表7-3 表示了在各步连接的动作和其意义。

表 7-3 分选器状态和动作关系

状 态	转移条件成立时	动 作	意 义
Ready	E	PlatRM := 0	平台旋转电动机停止
		ConVM := T	皮带输送机电动机运转
Loading1	E	PlatTM := T	平台传送电动机运转
Loading2	E	ConVM := F	皮带输送机电动机停止
LRotation	E	PlatTM := F	平台传送电动机停止
		PlatRMR := 1	平台旋转电动机向右旋转
RRotation	E	PlatTM := F	平台传送电动机停止
		PlatRML := 1	平台旋转电动机向左旋转

(续)

状 态	转移条件成立时	动 作	意 义
Unloading	E	PlatRM : = 0	平台旋转电动机停止
		PlatTM : = T	平台传送电动机停止
WaitUnloading	E	PlatRM : = 0	平台旋转电动机停止

(3) SFC 图

分选器的控制过程如图 7-13 所示,结合表 7-2 和表 7-3,可编写分选器的顺序功能表图,见图 7-14,转换条件表见表 7-4,功能块见表 7-5。

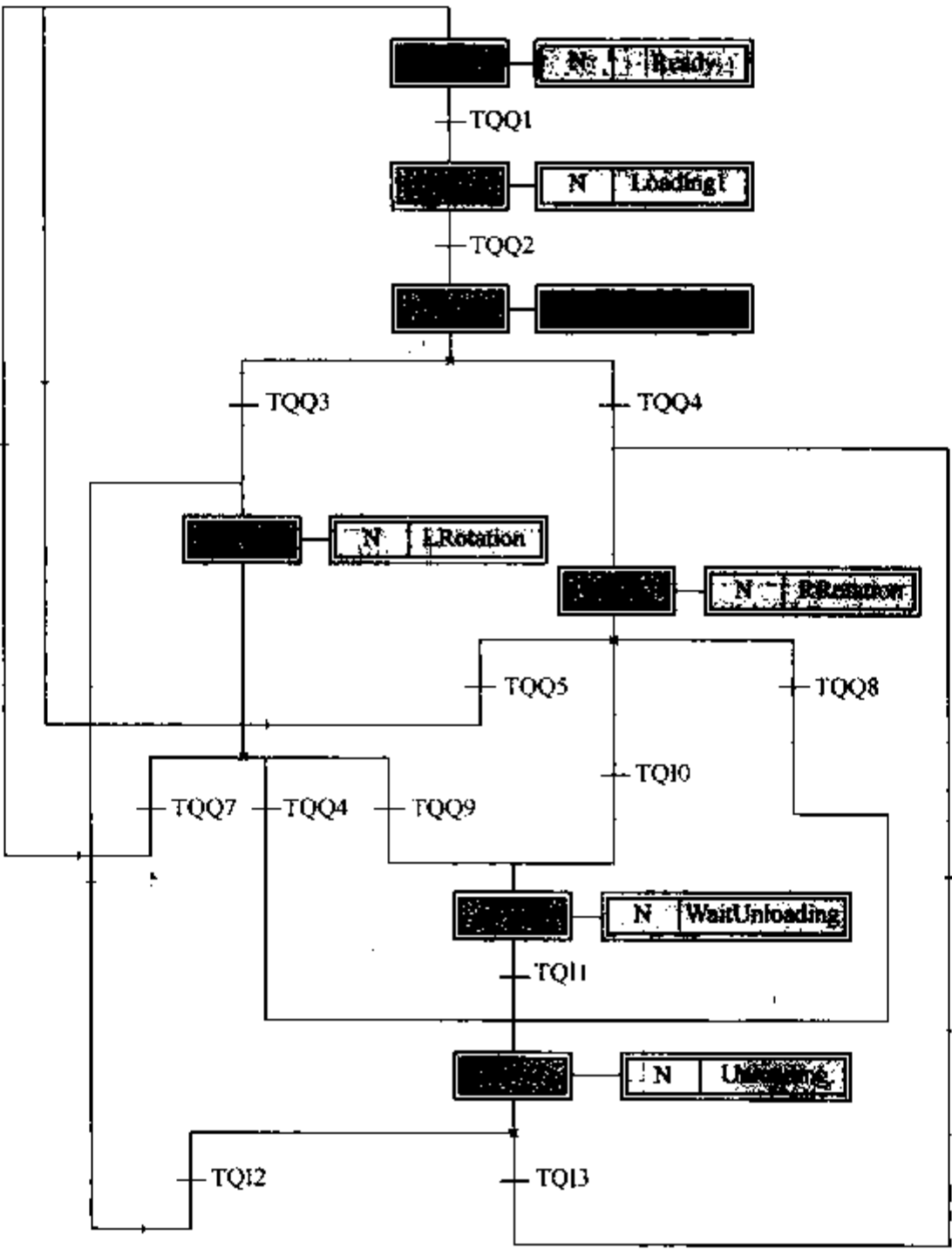


图 7-14 分选器控制的 SFC 图

表 7-4 分选器的转换条件

原 状 态	新 状 态	转 换 名	抓 取 感 觉 条 件
Ready	Loading1	T001	IPh
Loading1	Loading2	T002	NOT IPh
Loading2	LRotation	T003	BS AND SPh
Loading2	RRotation	T004	BS AND NOT SPh
RRotation	Ready	T005	CP AND NOT BS
RRotation	Unloading	T006	RP AND BS AND OutRS
LRotation	Ready	T007	CP AND NOT BS
LRotation	Unloading	T008	LP AND BS AND OutLS
LRotation	WaitUnloading	T009	LP AND BS AND NOT OutLS
RRotation	WaitUnloading	T010	RP AND BS AND NOT OutRS
WaitUnloading	Unloading	T011	(RP AND OutRS) OR (LP AND OutLS)
Unloading	LRotation	T012	NOT BS AND RP
Unloading	RRotation	T013	NOT BS AND LP

表 7-5 动作控制功能块

步或状态	皮带输送机电动机	平台传送电动机	平台旋转电动机	
	ConVM	PlatTM	PlatRMR	PlatRML
Ready	1	0	0	0
Loading1	1	1	0	0
Loading2	0	1	0	0
LRotation	0	0	0	1
RRotation	0	0	1	0
Unloading	0	0	0	0
WaitUnloading	0	1	0	0

可以方便地用 IEC61131-3 标准的编程语言编写转换条件和动作控制功能块的程序。

3. SFC 图的优化

实际应用时,可考虑对分选器控制系统进行优化,以提高系统效率。例如,可考虑当工件在平台的旋转步时,皮带输送机就可将下一个工件传送到使 IPh 为 1 的位置。这样,当旋转平台回复到 CP 位置时,就能立即进行下一次分选,而不必等待传送皮带将下一个工件送来。这种优化可缩短操作时间,提高分选效率。仅当两个工件之间没有足够的距离确保可对它们进行正确分选时,才应停止皮带机的传送。因此,当平台仍在处理前一个工件时,下一个工件是否到达 IPh 检测位置,可用 IF 语句进行识别。为此,要增加一些步或状态,定义一个新的状态图。也可采用原状态图,但增加新的有关其状态的动作。图 7-13 是经改进后的状态图。

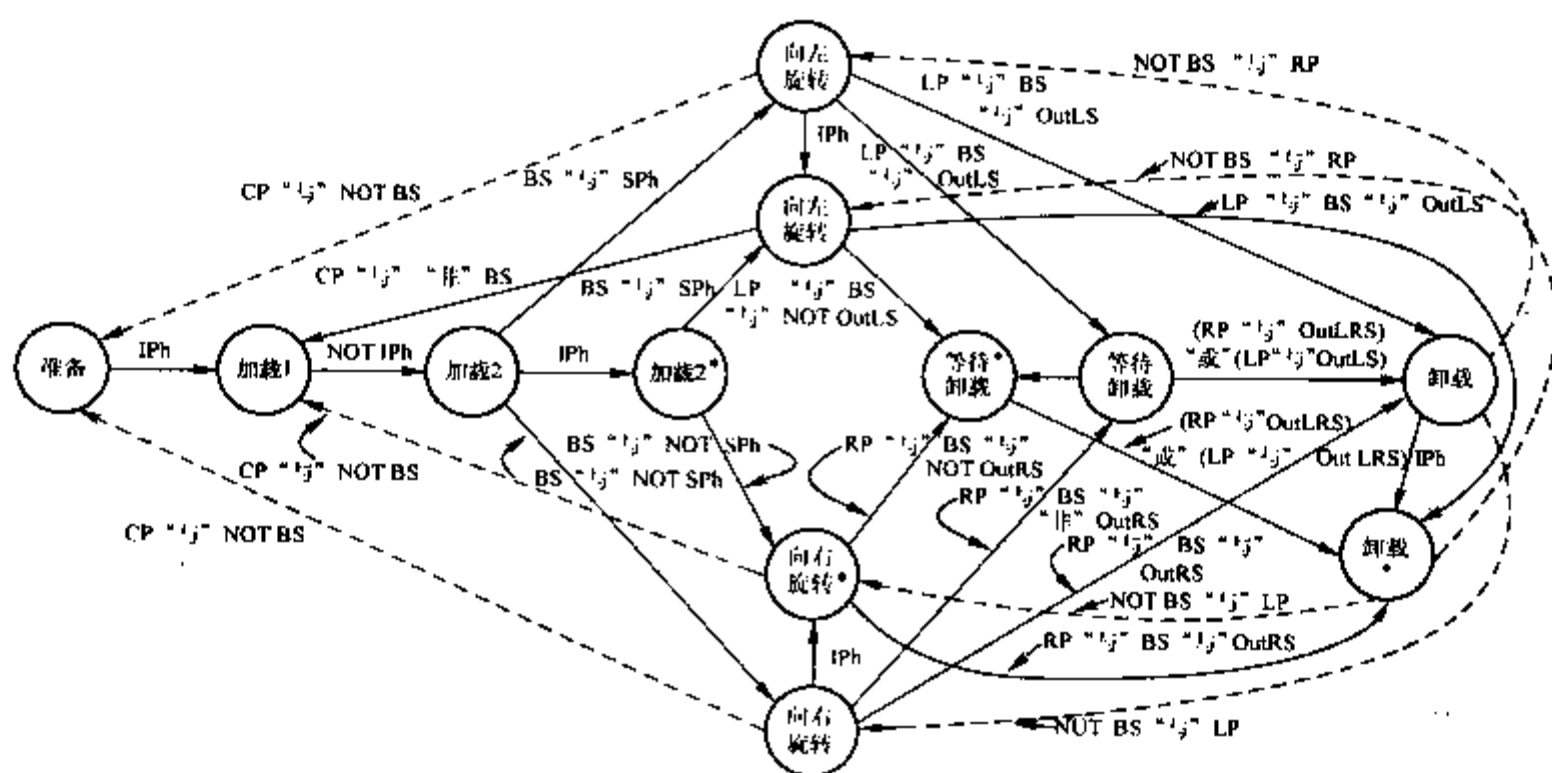


图 7-15 经改进后的状态图

改进后的动作控制功能块见表 7-6。转换条件可从图 7-15 直接列出,不多述。

表 7-6 经改进后的动作控制功能块

步	传送皮带电动机	平台传送电动机	平台旋转电动机	
	ConVM	PlatTM	PlatRMR	PlatRML
Ready	1	0	0	0
Loading1	1	1	0	0
Loading2	0	0	0	0
Loading2 *	0	1	0	0
LRotation	0	0	0	1
LRotation *	0	0	0	1
RRotation	0	0	1	0
RRotation *	0	0	1	0
Unloading	0	1	0	0
Unloading *	0	1	0	0
WaitUnloading	0	0	0	0
WaitUnloading *	0	0	0	0

实际应用时,可增加判别语句实现。如在 S004、S005、S006 和 S007 步增加下列语句:

IF IPh THEN ConVM := 0 ; END_IF;

第 8 章 实 验

8.1 用梯形图和功能块图编程语言编写程序

实验目的

- 1) 标准编程软件的基本操作方法。
- 2) 梯形图和功能块图编程语言的使用。
- 3) 标准函数和标准功能块的使用。

实验设备

计算机,带因特网接口。

实验步骤

- 1) 开机,点击科维软件公司的 MULTIPROG 软件的图标,进入桌面。该软件介绍见附录。
- 2) 用梯形图编写程序。程序的变量表如图 8-1 所示。

	名称	类型	用法	描述	地址	初值	保持	PBD	OPC
	Default								
	START1	BOOL	VAR	一楼启动按钮	%IX0.0		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	START2	BOOL	VAR	二楼启动按钮	%IX0.1		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	STOP1	BOOL	VAR	一楼停止按钮	%IX0.2		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	STOP2	BOOL	VAR	二楼停止按钮	%IX0.3		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	RUN	BOOL	VAR	电动机	%QX0.0		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

图 8-1 变量表

梯形图程序见图 8-2。

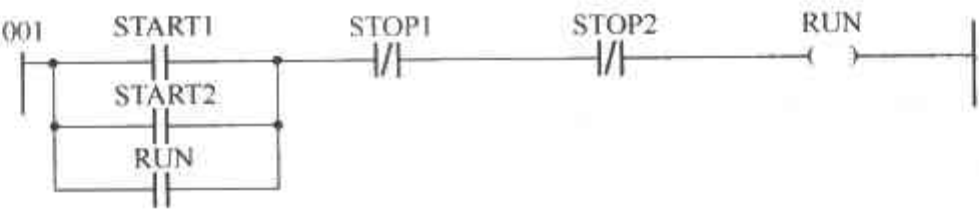


图 8-2 梯形图程序

- 3) 检查程序,并下载到仿真器,运行后观测信号灯的变化情况,如图 8-3 所示。

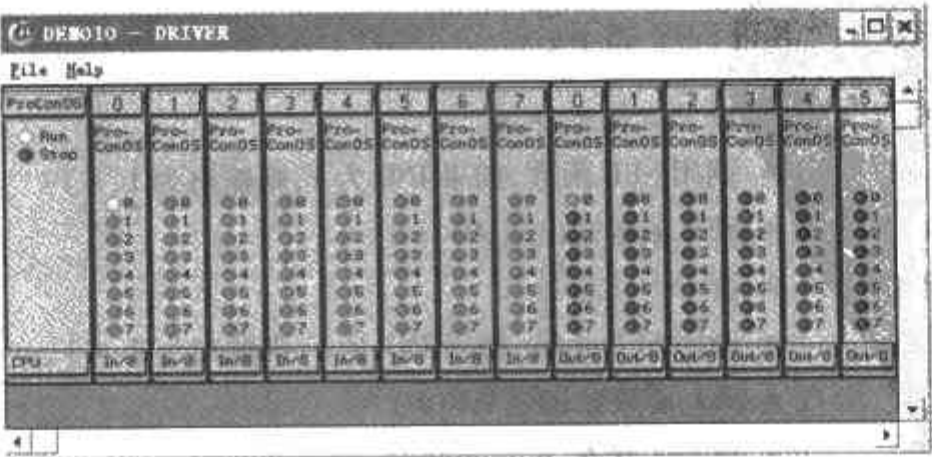


图 8-3 仿真器显示的画面

- 4) 用功能块图编程语言编写上述程序。先建立功能表图的程序,并用类似方法编写变量表。
- 5) 编写功能块图程序,见图 8-4,并用仿真器进行仿真。

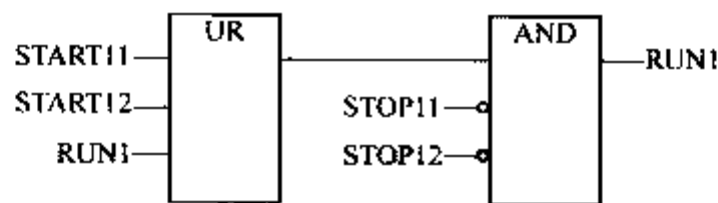


图 8-4 功能块图的程序 1

- 6) 编写功能块图程序,用 RS 标准功能块实现上述控制要求,程序见图 8-5。

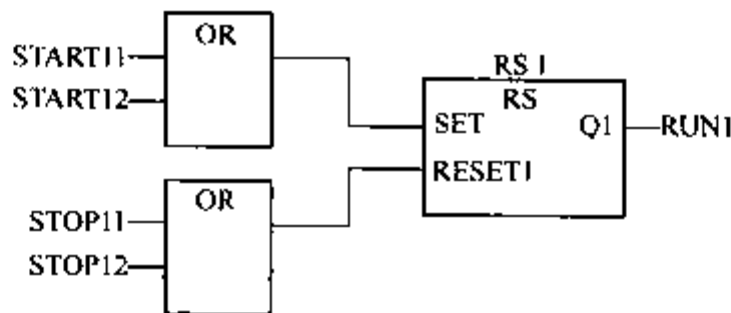


图 8-5 功能块图的程序 2

- 7) 比较功能块图程序中有关变量的连接有什么不同,梯形图和功能块图程序实现与、或逻辑运算的方法,说明本程序的功能。

8.2 用指令表和结构化文本编程语言编写程序

实验目的

- 1) 标准编程软件的基本操作方法。
- 2) 指令表和结构化文本编程语言的使用。
- 3) 标准函数和衍生函数的使用。

实验设备

计算机,带编程软件。

实验步骤

- 1) 输入的指令表编程语言程序如下:

```
LD      IN0      (* 读 IN0 状态 *)
SEL     0, 1      (* 调用 SEL 函数,如果 IN0 为 1,则输出为 1 *)
ST      A0        (* SEL 函数输出送 A0 *)
LD      IN1      (* 读 IN1 状态 *)
SEL     0, 2      (* 调用 SEL 函数,如果 IN1 为 1,则输出为 2 *)
ST      A1        (* SEL 函数输出送 A1 *)
LD      IN2      (* 读 IN2 状态 *)
SEL     0, 4      (* 调用 SEL 函数,如果 IN2 为 1,则输出为 4 *)
```

ST A2 (* SEL 函数输出送 A2 *)
ADD A1 (* 结果加 A1 *)
ADD A0 (* 结果加 A0 *)
ST OUT (* 输出到 OUT *)

2) 变量表见图 8-6。

名称	类型	用法	描述	地址	初值	保持	PDD	OPC
Default								
IN0	BOOL	VAR		%IX0.0		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
IN1	BOOL	VAR		%IX0.1		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
IN2	BOOL	VAR		%IX0.2		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A0	INT	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A1	INT	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A2	INT	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
OUT	INT	VAR				<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

图 8-6 变量表

3) 运行结果见图 8-7。将调试开关选中。

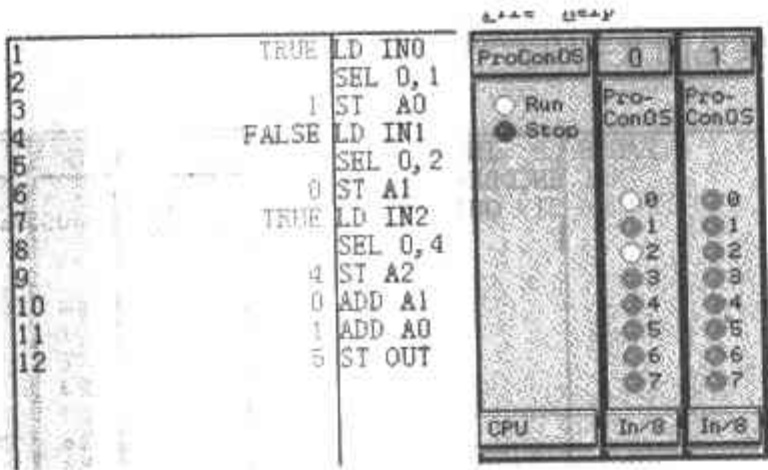


图 8-7 运行结果显示画面

4) 编写结构化文本编程语言程序如下：

A0:=SEL(IN0,0,1);
A1:=SEL(IN1,0,2);
A2:=SEL(IN2,0,4);
OUT:=A0+A1+A2;

5) 运行结果见图 8-8。

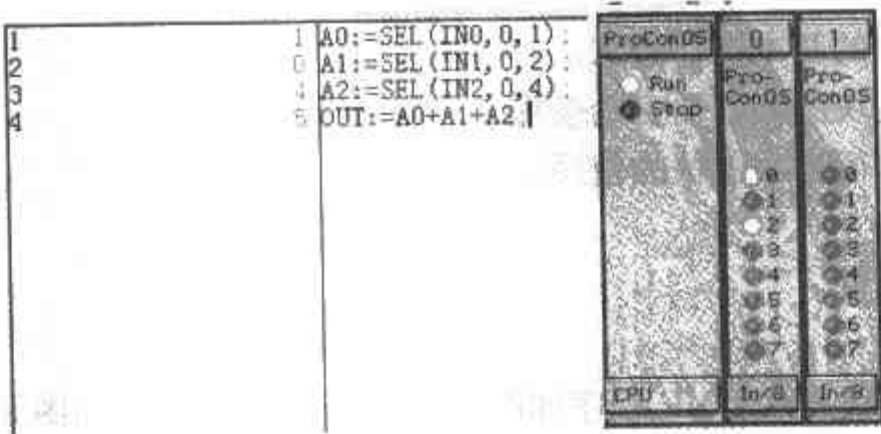


图 8-8 运行结果显示画面

6) 编写衍生函数 ENCODER,用于上述的译码。函数本体程序如下:

```
ENCODER:=SEL(IN0,0,1)+SEL(IN1,0,2)+SEL(IN2,0,4);
```

函数的变量表见图 8-9,注意与原变量表的区别。

名称	类型	用法	描述	地址	初值	保持	PDO	OPC
Default								
IN0	BOOL	VAR_INPUT						
IN1	BOOL	VAR_INPUT						
IN2	BOOL	VAR_INPUT						

图 8-9 衍生函数的变量表

7) 编写程序用于调用函数 ENCODER。用指令表编程语言编写的程序如下:

```
LD IN0          (* 读取 ENCODER 的第一个输入 IN0 *)
ENCODER IN1,IN2  (* 读取其他参数,并调用 ENCODER 函数 *)
ST OUT          (* 函数运算结果送 OUT *)
```

8) 调试运行后的结果见图 8-10。

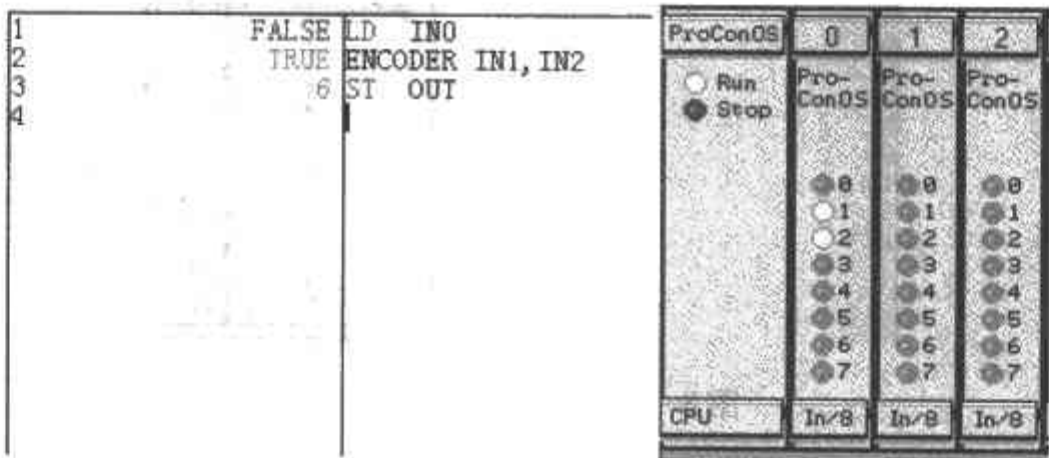


图 8-10 调用衍生函数和程序执行的显示画面

9) 比较衍生函数和函数调用及程序、变量表的不同,熟悉衍生函数的编写。

8.3 编写衍生功能块和调用功能块的程序

实验目的

- 1) 标准编程软件的基本操作方法。
- 2) 梯形图和功能块图编程语言的使用。
- 3) 标准功能块和衍生功能块的使用。

实验设备

计算机,带编程软件。

实验步骤

1) 编写衍生功能块 CYCTIME 程序如图 8-11 所示。变量表如图 8-12 所示,它调用标准功能块。

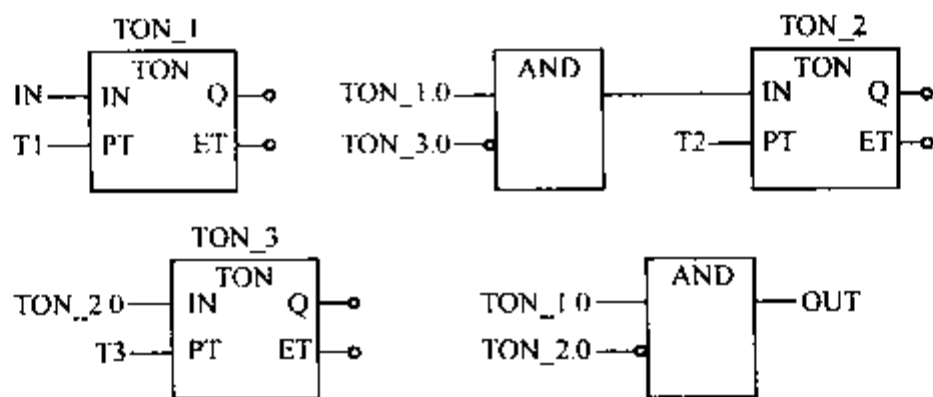


图 8-11 衍生功能块 CYCTIME 程序

名称	类型	用法	描述	地址	初值	保持	PBD	OPC
Default								
TON_1	TON	VAR						
TON_2	TON	VAR						
TON_3	TON	VAR						
IN	BOOL	VAR_INPUT						
T1	TIME	VAR_INPUT						
T2	TIME	VAR_INPUT						
T3	TIME	VAR_INPUT						
OUT	BOOL	VAR_OUTPUT						

图 8-12 衍生功能块 CYCTIME 的变量表

2) 编写调用衍生功能块的程序,如图 8-13 所示。

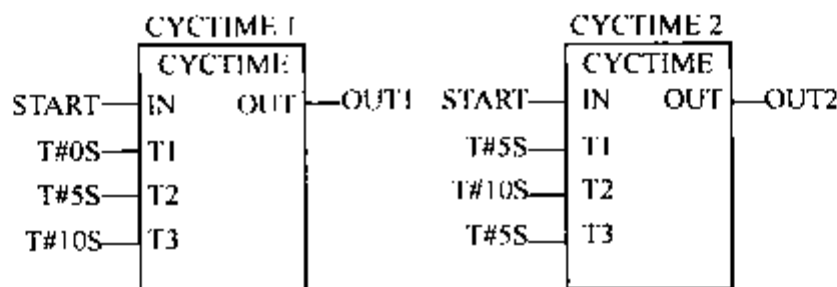


图 8-13 调用衍生功能块的程序

程序的变量表如图 8-14 所示。

名称	类型	用法	描述	地址	初值	保持	PBD	OPC
Default								
CYCTIME_1	CYCTIME	VAR						
START	BOOL	VAR		%IX0.0				
CYCTIME_2	CYCTIME	VAR						
OUT1	BOOL	VAR		%QX0.0				
OUT2	BOOL	VAR		%QX0.1				

图 8-14 调用衍生功能块的程序变量表

3) 运行程序,观测信号灯 OUT1 和 OUT2 的变化情况。

8.4 用顺序功能表图编程语言编写程序

实验目的

1) 标准编程软件的基本操作方法。

- 2) 顺序功能表图编程语言的使用。
- 3) 动作控制功能块和转换条件的编写。

实验设备

计算机,带编程软件。

实验步骤

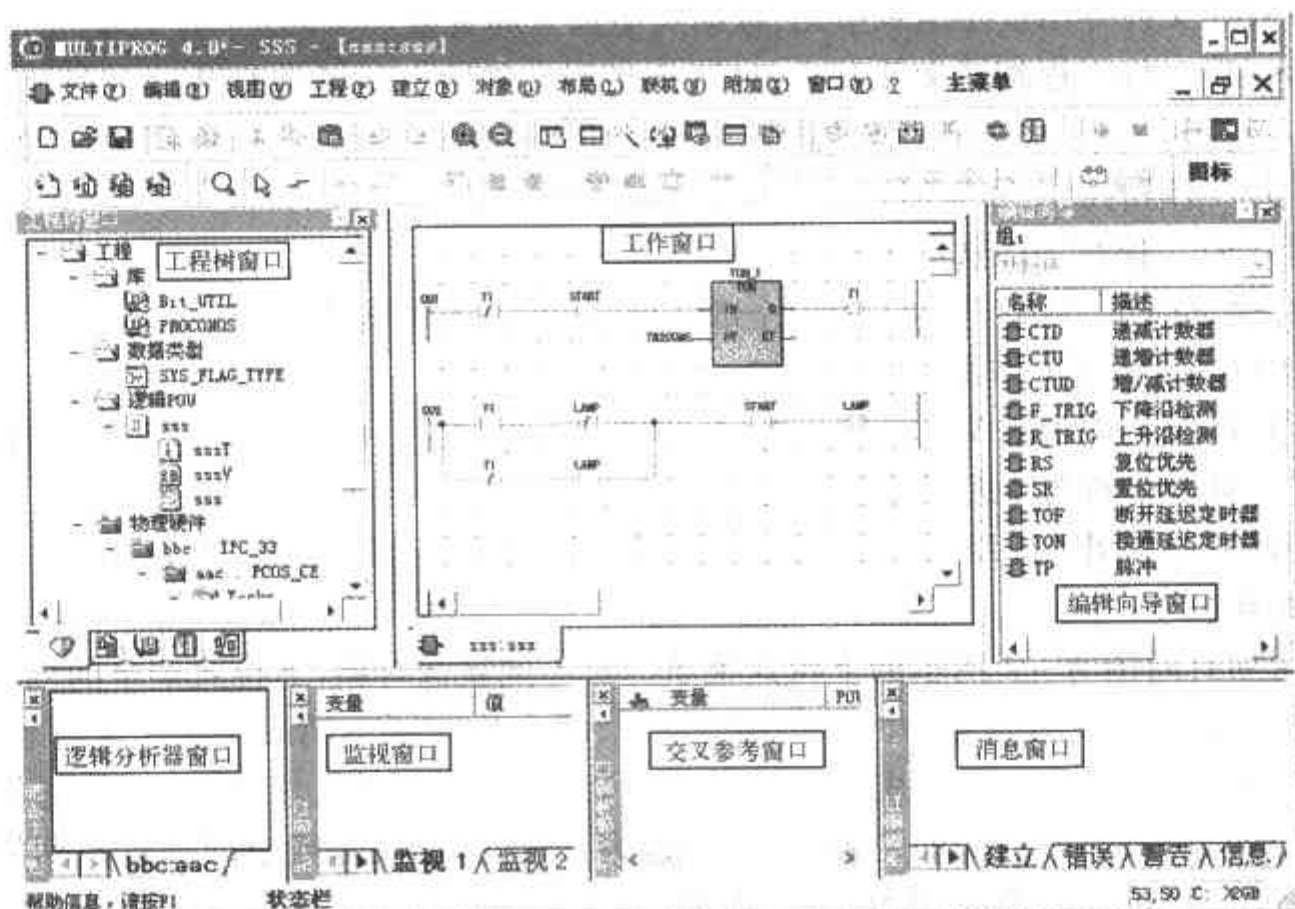
- 1) 编写交通信号控制系统程序,顺序功能表图程序见图 5-27。
- 2) 编写各转换条件的程序,详见教材。
- 3) 编写各动作控制功能块的程序,详见教材,部分程序需要读者编写。
- 4) 执行顺序功能表图程序,观测信号灯的变化情况。
- 5) 与梯形图等其他编程语言编写的程序比较。

附录

附录 A MULTIPROG 软件的使用

1. 编程语言的操作桌面

MULTIPROG 的操作桌面如附图 A-1 所示。图中除主菜单和有关图标外,共有 7 个窗口,它们是工程树窗口、编辑向导窗口、逻辑分析窗口、工作窗口、监视窗口、交叉变量窗口和消息窗口。这些窗口可以通过点击主菜单“视图”的下拉式菜单选项或有关图标打开或关闭。



附图 A-1 MULTIPROG 的操作桌面

操作桌面的图标与 Windows 操作桌面类似。一些常用图标也与 Windows 图标相同或类似。例如,建立空白文档、打开和保存;打印预览和打印;剪切、复制和粘贴;撤消和恢复等。此外,该编程系统也建立了专用图标,它们将在下面各节介绍。操作桌面的操作也与 Windows 操作系统相同,例如,下拉式菜单、点击、拖曳、双击等。

用户应用的主要窗口是工作窗口。该窗口根据所选用的编程语言可输入文本类编程语言,也可绘制图形类编程语言的图形元素。其他窗口也可作为独立窗口显示。

2. 主菜单

各主菜单下拉式菜单选项前的图标与操作桌面的图标一致,因此,可直接点击相应图标执行该命令。MULTIPROG 主菜单有下列选项。

(1) 文件(F)

文件选项有附图 A-2 所示的下拉式菜单。下拉式菜单前的图标与操作桌面的图标一致。用户可直接使用操作桌面的图标实现相同的操作。快捷键的使用方法与 Windows 类似。例如,按 <Alt + F> 键用于打开文件菜单,按 <Ctrl + W> 键用于新建工程等。

- 新建工程(W)。用于建立新的工程项目。在编程时,应先建立新工程项目。
- 打开工程/解压缩工程(O)。用于打开和解压缩已建立的工程。
- 工程另存为/压缩为(A)。将已编制的工程文件另存或压缩为其他文件名的文件。
- 关闭工程(R)。关闭已打开的工程项目。
- 删除工程(D)。将工程项目文件删除。
- 另存为模板(M)。将建立的工程项目作为模板保存。
- 删除模板(T)。将建立的工程项目模板删除。
- 导出(E)。将当前工程的翻译文件、交叉参考文件的加载项导出,或用扩展的 IEC 61131-3 文件导出程序组织单元、数据类型和全局变量。
- 导入(I)。将工程翻译文件 IEC 61131-3 扩展文件导入到当前工程。
- 输入密码(N)。为防止无权限的人员对已建工程的误操作,该选项用于设置工程密码,并用于密码的激活和撤消。
- 保存(S)。保存当前选中的文件,不退出操作桌面。
- 保存全部(L)。保存当前工程的全部文件,不退出操作桌面。
- 关闭(C)。关闭工作窗。
- 打印(P)、打印预览(V)、打印设置(U)、打印工程(J)。用于打印有关文件、预览需打印文件、对打印的文件进行设置和文件打印。
- 退出(X)。退出 MULTIPROG 操作桌面,回到 Windows 操作桌面。








(2) 编辑(E)

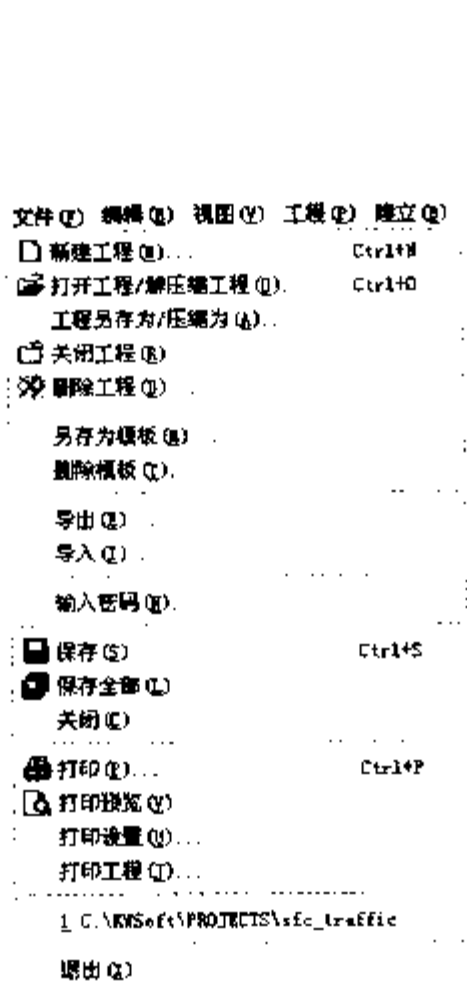
编辑选项有附图 A-2 所示的下拉式菜单。对不同被选对象,下拉式菜单的内容会不同。其操作功能与 Windows 有关选项的操作功能类似,但名称有所不同或对功能进行了细分。例如,“查找”和“全局查找”,分别用于局部变量和全局变量的查找。“查找”功能还细分为“查找上一个”和“查找下一个”功能,“替换”功能分为替换变量(局部或全局变量)、功能(该编程系统软件将功能 Function 译为函数)和功能块。

在附图 A-3 中,“模式”选项用于在 LD 编程语言编制的程序中标记对象、插入分支、连接对象等。在 SFC 编程语言编制的程序中还用于插入 SFC 分支。同样,“伸展/压缩”选项用于在 LD 编程语言编制的程序中插入或删除行,插入或删除列,排列电源轨线等。在 SFC 编程语言编制的程序中还用于优化 SFC 的有向连线。

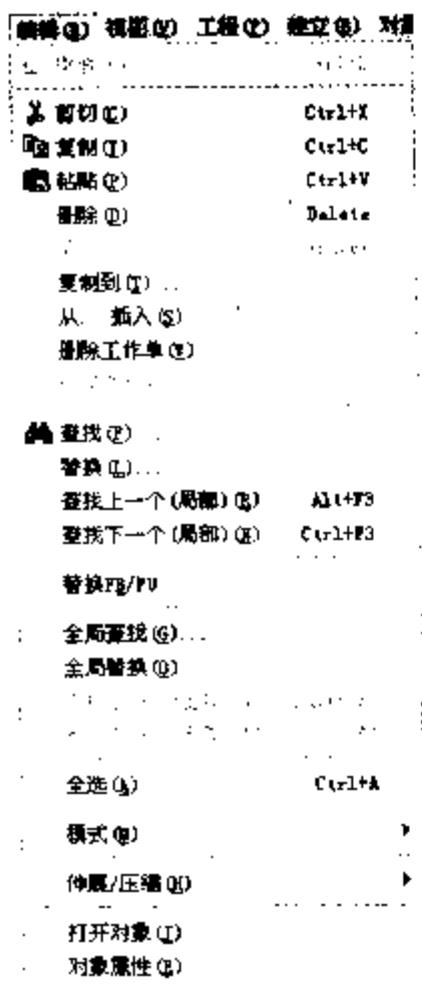
根据所选对象的不同,该下拉式菜单有不同的可编辑选项。例如,对 ST 编程语言编制的程序段,下拉式菜单如附图 A-4 所示。图中的“列出所声明的变量”用于将该 ST 编程语言编制程序中说明的变量全部列出。

(3) 视图(V)

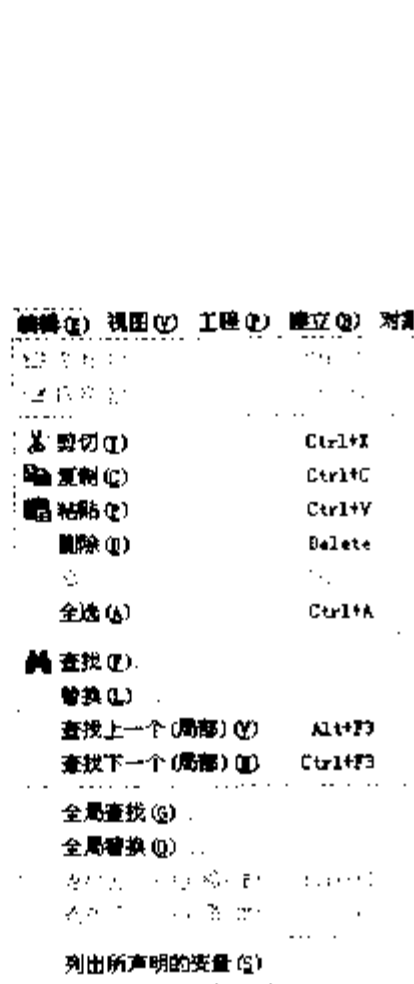
视图选项有附图 A-5 所示的下拉式菜单。每个选项都有对应的图标可使用。分别对应工程树窗、消息窗、编辑向导窗、交叉参考窗、监视窗、逻辑分析窗和打开变量工作单。



附图 A-2 “文件”的下拉式菜单



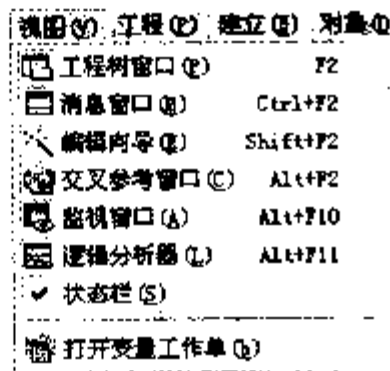
附图 A-3 “编辑”的下拉式菜单



附图 A-4 “编辑”的不同下拉式菜单

在操作桌面的最下面有一条状态栏,用于显示当前工作状态。选中“视图”下拉式菜单的“状态栏(S)”选项,用于显示(√)或不显示该状态栏。

- 工程树窗口,即工程树管理器。它包括工程树编辑器和实例树。工程树编辑器用于编辑用户的工程结构(例如,说明库、添加工作单用来编辑数据类型、添加新的程序组织单元和工作单),用于编辑变量和代码本体工作单。在物理硬件目录下用于建立工程名称和类型、程序组织单元名称和类型、配置名和类型、资源名和类型、任务名和类型及所选编程语言等。实例树显示资源的可用任务和有关程序,及所有功能和实例化的功能块。它可通过该窗口下部的图标选择显示实例树的工程、程序组织单元(含数据类型和逻辑程序组织单元)、库、所用硬件和实例等。
- 消息窗口。用于显示编译过程中的不同步骤、编译错误和警告,以及一些其他信息,例如,工程中已编译的 POU 实例树等。
- 编辑向导窗口。分为“组”列表框和选择区域等两个区域。它用于选择程序所需的功能或功能块等。先在工作窗口点击,然后通过该窗口“组”的下拉式菜单选择所需的选项,在选择区域将显示该选项所有的可选项。选择区域可提供标准功能或功能块,也可提供用户的功能或功能块。可提供库内存储的选项或全部选项等。附图 A-6 显示标准功能块的编辑向导窗口。



附图 A-5 “视图”的下拉式菜单



附图 A-6 显示标准功能块的编辑向导窗口

- 交叉参考窗口。用于在调试和出错隔离时显示各编程语言编制程序中使用的变量、程序组织单元或工作单的位置、对该变量的访问方式(读或写等)、所用命令、输入输出地址、全局路径、变量的数据类型、初始值、变量在程序组织单元中显示的位置等,它也显示所有变量、功能块、动作、转换、步、跳转、标号以及在当前工程内所使用的连接符。
- 监视窗口。用于搜集来自不同工作单的变量,获得这些变量一起工作的评价报告。它可用于插入和删除变量,在调试时强制设置和清除变量。监视窗口包括变量、变量的当前值、约定值和数据类型、实例等。可多个监视窗口,通过窗口下部的标志选择所需监视的窗口。
- 逻辑分析窗口。最多显示 10 条需分析变量的记录曲线。
- 打开变量工作单。用于将所用变量工作单显示在工作窗口,也可点击工程树窗口的最后字母为 V 的文件来显示变量工作单。

(4) 工程(P)

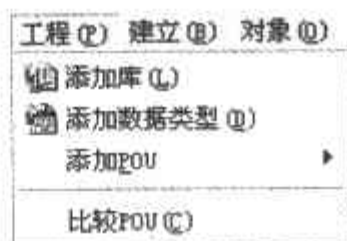
工程选项有附图 A-7 所示的下拉式菜单。

- 添加库。主要用于在工程树的库目录下添加用户的库文件(后缀为 .mwt 的文件)。
- 添加数据类型。与操作桌面上添加对象、添加程序、添加功能块和添加函数(即功能,下同)的图标功能类似,用于添加工作单和数据类型到用户的工程项目。
- 添加 POU(程序组织单元)。包括添加功能块、程序和功能(函数)等选项。
- 比较 POU。用于比较两个工程项目中的 POU。比较前用“制作”或“重建工程”命令先对 POU 进行编译。通常用于比较 PLC 的工程项目中 POU 与 PC 硬盘上保存的工程项目中 POU 是否一致。它也检测代码本体工作单和变量工作单内的不同点,及“逻辑 POU”子树结构中的不同点。

(5) 建立(B)

建立选项有附图 A-8 所示的下拉式菜单。

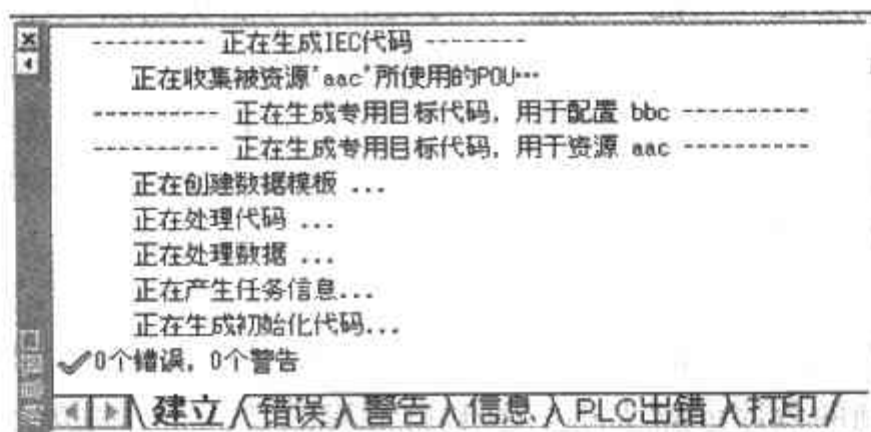
- 制作(M)。当程序更改后,需要用“制作”选项对更改后的程序进行编译。编译时在消息窗口显示编译过程和编译结果。附图 A-9 显示某工程经制作后在消息窗显示的消息。



附图 A-7 “工程”的下拉式菜单



附图 A-8 “建立”的下拉式菜单



附图 A-9 某工程经制作后在消息窗口显示的画面

- 修补 POU(P)。当工程的结构,包括所使用(空的)程序、功能及功能块等 POU 已经定义,当前工程已经用“制作”命令进行编译,并下装到 PLC 系统和执行冷启动,为 POU 定义的存储器保留区域(工厂约定为 500B)有足够空间用于存放修补 POU 的用户数据时,可以用“修补 POU”命令对该工程进行修补,并将修补内容补充到已在 PLC 运行的系统中。
- 编译工作单(W)。系统会在工作单关闭或保存时自动对工作单和变量工作单进行编译。
- 重建工程(R)。在第一次编译或所说明的用户库已被更改时需要重建工程。该命令将编译和连接所有工作单,并将编译错误显示在消息窗口。
- 停止编译(S)。用于停止当前正在编译的工程。
- 转到下一个错误(G)和转到上一个错误(V)。在消息窗口的错误页显示下一个和上一个错误。
- 建立交叉参考(C)。用于自动建立交叉参考的变量列表。

(6) 对象(O)

对象选项有附图 A-10 所示的下拉式菜单。除使



附图 A-10 “对象”的下拉式菜单

用 IL、ST 编程语言时,下拉式菜单只有变量选项外,其他编程语言时,下拉式菜单内容如图示,它分为 5 组。

- 第 1 组主要用于 SFC 编程语言。实现添加步或转换,添加动作控制功能块和组成并行和选择分支序列。
- 第 2 组主要用于 FBD 编程语言。实现添加变量、添加连接符、标号、跳转符,添加返回符和设置文字注释。
- 第 3、4 组主要用于 LD 编程语言。实现添加梯级、添加触点、添加线圈、添加串接触点、添加并接触点、添加电源轨线等。还可用于将触点或线圈属性切换(例如,从常开切换到常闭)。
- 第 5 组用于复制功能和功能块的输入,将功能或功能块输出或输入信号取反连接。

使用 IL、ST 编程语言时,对象的下拉式菜单只有“变量”选项,用于添加变量。

这些命令前的图标与操作桌面的图标一致。因此,可以用操作桌面的图标直接进行操作。

针对不同的被选对象,例如,触点,功能或功能块的输入或输出,下拉式菜单中显示可执行的命令。不能对被选对象执行的命令用灰色显示。

(7) 布局(L)

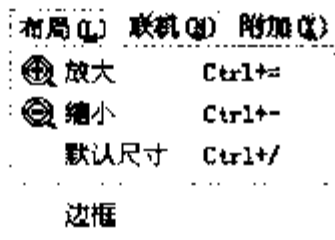
除采用图形类编程语言外,布局选项有附图 A - 11 所示的下拉式菜单。布局用于设置工作窗口中编程语言编制的程序元素大小。图形类编程语言的布局选项的下拉式菜单有不同的命令项。

- 放大和缩小。用于将整个工作窗内容放大和缩小,其目的是使在有效画面(最大 999 × 999 像素)上能清晰显示编制的程序。
- 默认尺寸。使工作窗恢复到系统约定的大小显示(250 × 999 像素)。
- 边框。用于注释时设置或不设置边框。

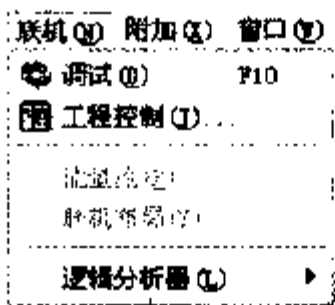
图形类编程语言的布局选项有其他选项,可参考有关资料。

(8) 联机(N)

联机选项有附图 A - 12 所示的下拉式菜单。该选项用于联机调试和工程控制,也可用于仿真器运行。



附图 A - 11 “布局”的下拉式菜单



附图 A - 12 “联机”的下拉式菜单

- 调试(D)。在系统软件下装后,可执行调试命令,这时,在工作窗可观测各变量的变化情况。
- 工程控制(J)。用于将已编译的程序下装到 PLC 系统。
- 能量流(P)。能量流用于在联机模式下检查哪些程序被实际执行(用竖线表示),哪些程序未被执行(用横线表示)。执行该命令时,从工作单切换到带能量流的地址状态。

- 联机布局(Y)。用于改变联机模式下变量值的显示方式。例如,只显示变量值不显示变量名,显示所有变量值,显示所有形参值等。
- 逻辑分析器(L)。它是用特定时间间隔记录变量值的工具。包括开始记录(S)、停止记录(P)、触发条件(T)、已连接变量(V)、捕获曲线(A)、清除曲线(C)、窗口宽度(W)和导出数据(E)等选项。

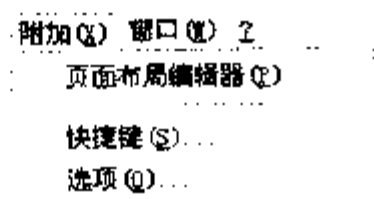
(9) 附加(X)

附加选项有附图 A-13 所示的下拉式菜单。

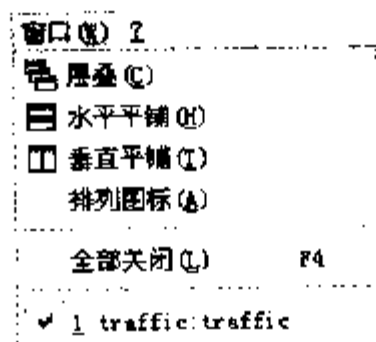
- 页面布局编辑器(P)。它是创建新页面布局或编辑原有页面布局的工具。用于标记对象、移动对象和删除对象。
- 快捷键(S)。用于定义用户的快捷键或定制的默认快捷键。
- 选项(O)。从弹出的对话框来设置菜单、工具栏、图形编辑器、逻辑分析器颜色、文本编辑器和文本颜色等选项的功能。

(10) 窗口(W)

窗口选项有附图 A-14 所示的下拉式菜单。用于整理桌面上的窗口和符号。例如,将多个窗口按层叠、水平平铺或垂直平铺方式排列、排列桌面上图标,或全部关闭窗口。



附图 A-13 “附加”的下拉式菜单



附图 A-14 “窗口”的下拉式菜单

(11) 求助选项

该求助选项的下拉式菜单用于提供电子教材、说明什么是 IEC 61131、提供对功能、功能块的帮助、提供对 PLC 的帮助、提供该软件的信息,也可输入注册信息。

3. 编程操作

MULTIPROG 编程系统软件的操作简单,有中文提示和图标快捷键,使用方便,容易掌握和学习。

(1) 建立新的工程项目

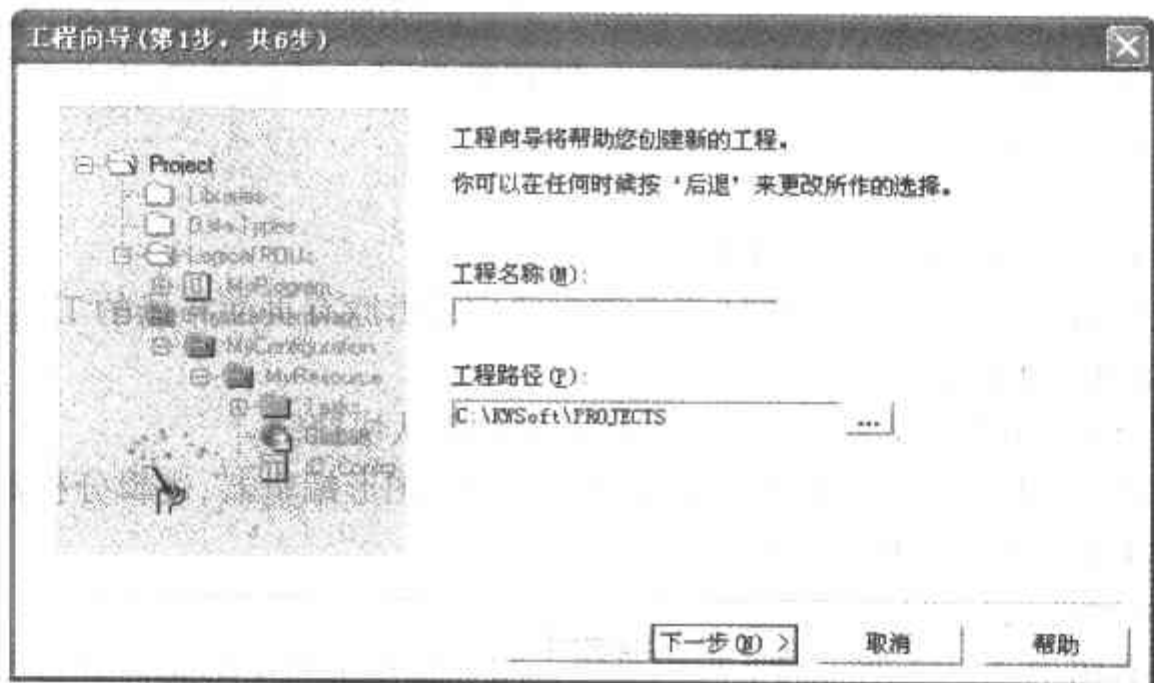
点击 KW-Software 目录下如附图 A-15 所示的图标,弹出附图 A-1 所示的操作桌面。显示窗口可通过点击有关图标或下拉式菜单打开或关闭。



附图 A-15 系统启动图标

点击主菜单“文件”选项的下拉式菜单,选择“新建工程”选项,从弹出的对话框中选择“工程向导”选项,弹出如附图 A-16 所示的对话框,也可根据所使用的模板选择,并输入新的工程项目名称,工程路径,并点击“下一步”按钮,依次弹出有关对话框,并输入程序组织单元(POU)名,及在该程序组织单元需使用的编程语言;配置名和类型(说明连接的 PLC 特性);资源名和类型(说明 PLC 中处理器类型和特性);任务名和类型(说

明 POU 用什么方式工作,例如,周期,事件,系统或约定方式)。最后点击“完成”按钮来完成工程向导的对话输入。









附图 A - 16 “新建工程”时的“工程向导”对话框

新建工程完成后,在工程树窗可见如附图 A - 17 所示的新建工程结构。



附图 A - 17 工程树窗显示的新建工程结构

程序组织单元 1、配置 1、资源 1、任务 1 及其后列出的类型都可从图中浏览。在逻辑 POU 树的分支“程序组织单元 1”下有 3 个分支。它们的最后字母分别表示表(T)、变量(V)和程序工作单。“程序组织单元 1”是输入的 POU 名。选用不同编程语言时,程序工作单前的图标不同。选中有关选项时,在工作窗显示相应的内容。选用不同编程语言时,操作桌面显示的可用图标不同,工程树上各分支前的图标也不同。

工作表(图标)包含了可选的用于文档资料目的的描述。变量表(图标,上部浅黄色)包含该 POU 中的所有局部变量说明。程序工作单图标根据所用编程语言不同而变化(ST 和 IL 编程语言图标,上部浅蓝色;LD、FBD 和 SFC 编程语言图标分别为、、,上部红色)。SFC 编程语言中的转换和动作也有对应的图标。

(2) 编程




















程序所用的变量可以在编程前说明,也可在编制过程中说明。变量说明在上述变量表显示。双击该选项可在工作窗显示变量表。附图 A-18 显示了某工程项目的变量表。

名称	类型	用法	描述	地址	初值	保持	POU	QPC
delay	BOOL	VAR	delay before next step		TRUE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
right	BOOL	VAR	turn right or left		TRUE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

附图 A-18 变量表


1) 变量说明。将光标移到变量表上 Default 位置,点击鼠标右键,从弹出对话框中选择“插入变量”选项,在该行下出现 NewVar 变量,可输入所插入变量的名称、类型、用法、描述、地址、初始值及属性标志(例如,保持)等。

2) 编程。根据所选编程语言,双击工程树的程序工作单,可在弹出的工作窗进行编程。不同编程语言显示的图标不同,编程方法也不同。

- LD 编程语言 光标点击工作窗,操作桌面显示可使用的编程元素图标。LD 编程语言包括插入梯级()、插入触点()、插入线圈()、并接触点( 、  和 )等操作,在触点两侧可以插入电源轨线,也可插入注释()、插入连接符、跳转或标号()或插入返回符()等。触点和线圈可以连接变量(),或通过双击所选触点或线圈的图形元素,从弹出对话框选用变量。当编程前没有完成变量说明表时,可在编程的同时完成对变量的说明。触点或线圈的取反既可从弹出对话框中选择类型来改变,也可点击图标()实现。
- IL 编程语言和 ST 编程语言 光标点击工作窗,操作桌面显示可使用的编程元素图标。IL 和 ST 编程语言是文本编程语言,因此,只需输入符合编程语言的指令或语句即可。
- FBD 编程语言 光标点击工作窗,操作桌面显示可使用的编程元素图标。可从编辑向导窗的组 and 选择区域选择所需的功能或功能块,或用户定义的功能或功能块。双击该功能或功能块,在弹出对话框中输入有关信息,经确定后,该功能或功能块将在工作窗显示。在功能或功能块的形参位置双击,从弹出对话框选择应连接的变量。两个功能或功能块之间的连接可直接点击连接对象图标(),再点击起点和终点来实现。操作桌面的大小可用缩放图标()实现。该编程语言还提供复制()和取反()功能。
- SFC 编程语言 SFC 编程语言提供创建步转换序列()、创建动作()和插入并行和选择分支()图标,也提供插入 SFC 分支()的图标。

建立 SFC 程序时,在工程树结构中显示转换和动作两个分支路径。每建立一个转换或动作都会根据所使用的编程语言,建立相应分支。

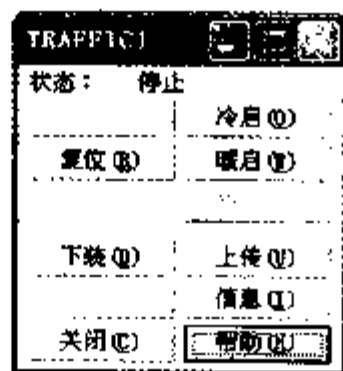
转换(条件)和动作(控制功能块)的编程方法与上述编程方法相同,不多述。

3) 程序编译。点击操作桌面主菜单“建立”的下拉式菜单选项“制作”,或直接点击图标()对所编程序进行编译。编辑信息在消息窗显示。当显示出错时,可点击错误、警告等页面查找编译时的错误或警告信息,并在程序中更改。例如,某些变量没有被使用、语法错误等。

(3) 运行

程序编译正确后,需下装到实际的 PLC 系统或下装到仿真器。可点击主菜单“联机”的下拉式菜单选项“工程控制”,或直接点击图标(国)来下装程序。从弹出对话框选择“下装”,并从弹出的下装对话框选择“下装”,状态栏将显示程序下装的过程。

下装后,弹出如附图 A-19 所示对话框。点击“冷启”按钮可执行仿真器的启动,或调用图形编辑器来显示 LD 编程语言工作单中的联机值。可从操作桌面下部的缩小图标,选择“DEMOIO - DRIVER”,恢复仿真器,即在桌面显示如图 4-17 所示的仿真器画面。其中,In/8 的 8 个面板用于点击输入信号的开和关,Out/8 的 8 个面板用于程序输出信号的显示。



附图 A-19 下装后显示画面

点击主菜单“联机”下拉式菜单选项“调试”或直接点击桌面图标(●)进入调试状态。在工作窗显示运行状态。

输入和输出面板号 and 对应点号的地址是 0~7,例如,In/8 的第 0 输入板卡第 1 点的地址是%IX0.1,Out/8 第 1 输出板卡第 0 点的地址是%QX1.0。

运行过程中,点击附图 A-19 所示显示画面的“停止”按钮,可停止程序运行。

注意,“暖启”、“冷启”和“热启”的区别:“热启”时不初始化任何数据,“冷启”时初始化所有变量,“暖启”时只初始化非保持型变量,可保持属性的变量采用其最新值。在更改“结束用户/起始系统”的地址后,系统只能“冷启”。

为修补 POU,可关闭“调试”开关(点击该图标),并点击修补 POU 图标(幽),或从主菜单“建立”下拉式菜单选择“修补 POU”实现。

附录 B 习题和思考题

B.1 填空题

1. 标准化编程语言来自_____方面的发展,即_____。
2. 编程语言与_____无关是十分重要的。为便于程序移植,标准编程语言提供_____机制。
3. 一个资源中可以有_____个任务。一旦任务被设置,则它就可控制_____执行,或者根据_____来执行。
4. 存取路径用_____开始,用_____结束,中间是_____,它由_____等项组成。
5. 采用标准编程语言,在一台 PLC 中可同时_____独立程序,实现对_____的完全控制能力。
6. 资源具有 IEC 61131-1 标定义的_____功能。
7. 枚举类型的系统初始约定值是_____。
8. 想从字符串'abcdef'中截取最左面的 4 个字符,则可用_____函数。
9. 时间文字外部表示有_____,_____,_____,_____。例如,用_____表示持续时间 15 时 3 秒。

10. C := DELETE('abcdefg',2,3); D := REPLACE('abcd','123',1,2); 运算结果,C 和 D 的内容分别是_____和_____。

B.2 选择题

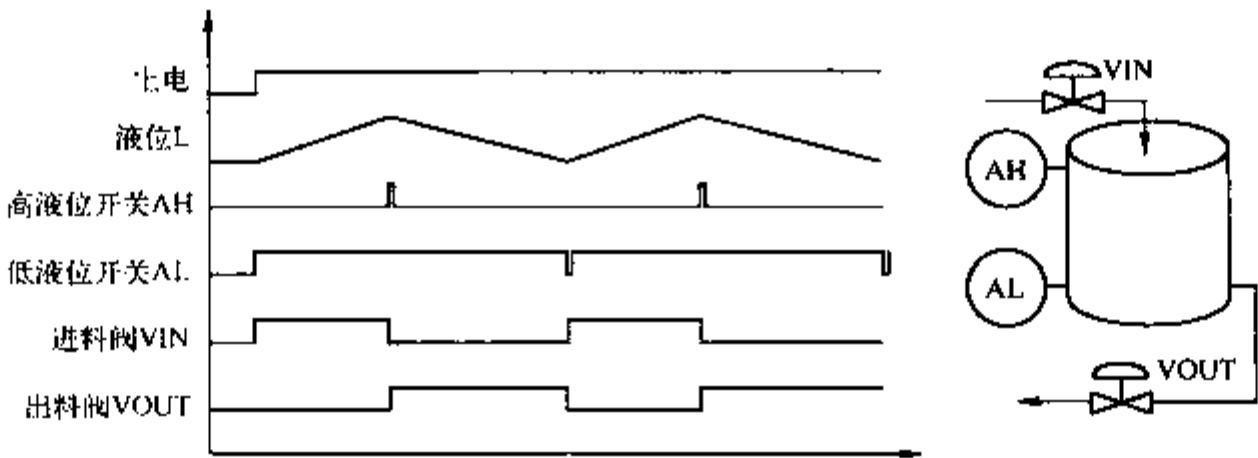
- 读写和只读属性对()变量有效。
(A) 输入 (B) 输出 (C) 存取 (D) 全局
- 功能块调用相同的输入变量时,其输出与()有关,因此,结果可以不相同。
(A) 内部变量 (B) 外部变量 (C) 输入变量 (D) 输出变量
- START 输入 10 秒后输出 OUTPUT,START 输入恢复时,OUTPUT 恢复,应采用()语句。
(A) TP1(IN:=START,PT:=T#10s);OUTPUT:=TP1.Q;
(B) TOF1(IN:=START,PT:=T#10s);OUTPUT:=TOF1.Q;
(C) TON1(IN:=START,PT:=T#10s);OUTPUT:=TON1.Q;
(D) OUTPUT:=R_TRIG1(CLK:=START);
- TASK SLOW_1(INTERVAL:=t#20ms,PRIORITY:=2);表示它是一个()任务。
(A) 间隔时间 20 ms 的周期 (B) 扫描周期为 20 ms 的周期
(C) 优先级为 2 (D) 事件触发
- ()不是标识符。
(A) I_am_a_student (B) _Abc_ (C) T_101_1 (D) T#12s
- ()是错误的分隔符号使用。
(A) TON_1(IN:=START,PT:=T#25h); (B) ANA;INT:=I2;
(C) J:=J; (D) D#2007_10_1;
- 不正确的数据外部表示是()。
(A) INT#40000 (B) REAL#5 (C) Tod#12;36 (D) 'I am a student'
- 当 START 输入信号计数 10 次,输出一个脉冲信号,应采用()语句。
(A) CTU1(CU:=START,RESET:=CTU1.Q,PV:=10);OUTPUT:=CTU1.Q;
(B) CTD1(CD:=START,LOAD:=CTD1.Q,PV:=10);OUTPUT:=CTD1.Q;
(C) CTU1(CU:=START,RESET:=CTU1.Q,PV:=DINT#10);OUTPUT:=CTU1.Q;
(D) CTD1(CD:=START,LOAD:=CTD1.Q,PV:=DINT#10);OUTPUT:=CTD1.Q;
- REAL_TO_SINT(2.5)和 REAL_TO_USINT(-3.6)的运算结果分别是()
(A) 3,-3 (B) 2,-4 (C) 3,0 (D) 2,-3
- 变量声明:VAR TT:ARRAY[1...2,1...4] OF INT:=[1,2,3,3(4),6,1];END_VAR;因此,TT[2,2]和 TT[2,3]的初始值分别是()。
(A) 3,4 (B) 3,6 (C) 4,6 (D) 4,4

B.3 编程和思考题

- 先编写用户函数 AREA,用于根据圆的半径 RR 计算圆面积 AREA,然后编写程序,计算直径为 5.2 cm 的圆面积。
- 编写用户功能块,要求输入信号 IN1 与输入信号 IN2 比较,如果 IN1 大于 IN2,则输出

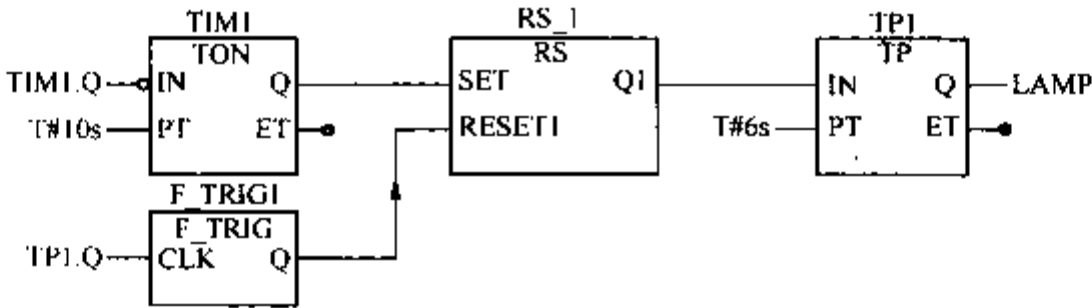
Q 是 IN1-IN2 的值,输出 Q1 为 1。反之,输出等于 IN2-IN1 的值,Q1 为 -1。

- 3. 编写程序,控制要求如下:将开关 START1 合上后,先延时 5 s,然后绿灯 GREEN 点亮 3 s,然后熄灭,并每隔 6 s,再点亮 3 s,循环点亮和熄灭。
- 4. 编写用户函数,输入信号与 20 比较,如果大于 20,则输出 20,反之,输出等于输入。
- 5. 编写程序,实现如附图 B - 1 所示输入输出信号的波形。



附图 B - 1 题 B. 3 - 5 图

- 6. 画出附图 B - 2 所示程序执行后的输出 LAMP 的波形。



附图 B - 2 题 B. 3 - 6 图

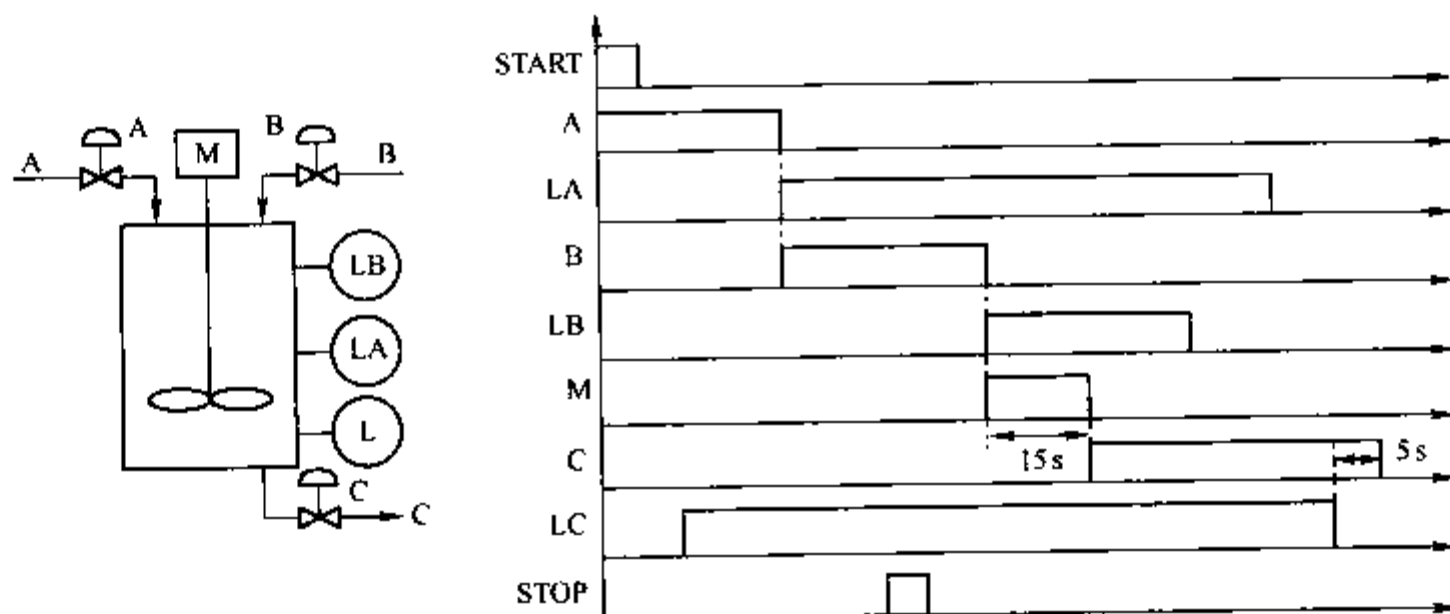
- 7. 结构化文本编程语言程序如下,说明该程序的功能。C1 是 CTU 实例名,RUN 和 STOP 是起动和停止按钮,COUNT 是设置按钮,用于设置电动机号。

```
C1(CU:=COUNT, RESET:=(STOP OR RUN OR C1.Q), PV:=5);
CS:=C1.CV;
CASE CS OF
1: RS1(SET:=RUN,RESET1:=STOP);
   R1:=RS1.Q1;
2: RS2(SET:=RUN,RESET1:=STOP);
   R2:=RS2.Q1;
3: RS3(SET:=RUN,RESET1:=STOP);
   R3:=RS3.Q1;
4: RS4(SET:=RUN,RESET1:=STOP);
   R4:=RS4.Q1;
ELSE CS:=0;
END_CASE;
```

8. 指令表编程语言程序如下,说明每条指令执行后,结果存储器的内容,并说明该程序的功能。

LD	RUN
ANDN	STOP
OR	START
ST	RUN

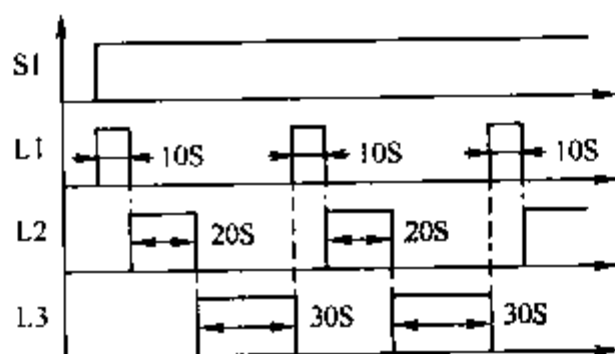
9. 用功能块图或顺序功能表图编程语言编写程序,实现物料的混合控制。生产过程和信号波形如附图 B-3 所示。



附图 B-3 题 B.3-9

操作过程说明如下:操作人员检查混合罐液位是否已排空,已排空后由操作人员按下 START 起动按钮,自动开物料 A 的进料阀 A,当液位升到 LA 时,自动关进料阀 A,并自动开物料 B 的进料阀 B。当液位升到 LB 时,关进料阀 B,并起动搅拌电动机 M,搅拌持续 15 s 后停止,并开出料阀 C。当液位降到 L 时,表示物料已达下限,再持续 5 s 后,物料可全部排空,自动关出料阀 C。整个物料混合和排放过程结束进入下次混合过程,如此循环。当按下 STOP 停止按钮时,在排空过程后关闭出料阀 C。

10. 编写程序实现 3 个信号灯的点亮和熄灭控制。信号灯的时序如附图 B-4 所示。图中, S1 是开关信号,当 S1 闭合后,信号灯 L1 点亮 10 s 并熄灭,然后信号灯 L2 点亮 20 s 并熄灭,最后信号灯 L3 点亮 30 s 并熄灭,该循环过程在 S1 断开时终止。



附图 B-4 题 B.3-10 图

参考文献

- [1] Lewis R W. Programming industrial control systems using IEC 1131-3 (Revised edition)[M]. London. 1998.
- [2] Bonfatti F, Monari P D, Sampieri U. IEC 61131-3 Programming Methodology(software engineering methods for industrial automated systems)[M]. ICS Triplex ISaGRAF Inc. 2003.
- [3] Lucas M R. Understanding and Assessing Logic Control Design Methodologies[D]. University of Michigan. 2003.
- [4] International Electrotechnical Commission. IEC 61131-1 Programmable controllers - Part1 : General information[S]. 2nd ed. 2003.
- [5] International Electrotechnical Commission. IEC 61131-2 Programmable controllers - Part2 : Equipment requirements and tests[S]. 2nd ed. 2003.
- [6] International Electrotechnical Commission. IEC 61131-3 Programmable controllers - Part3 : Programming language[S]. 2nd ed. 2003.
- [7] International Electrotechnical Commission. IEC 61131-8 Guidelines for the application and implementation of programming languages[S]. 2003.
- [8] 中国国家标准管理委员会. GB/T 15969.1—1995 可编程序控制器—第1部分:通用信息[S].
- [9] 中国国家标准管理委员会. GB/T 15969.2—1995 可编程序控制器—第2部分:设备特性[S].
- [10] 中国国家标准管理委员会. GB/T 15969.3—2006 可编程序控制器—第3部分:编程语言[S].
- [11] 中国国家标准管理委员会. GB/T 15969.5—2002 可编程序控制器—第5部分:通信[S].
- [12] 中国国家标准管理委员会. HG/T 20700-2000 可编程控制器系统设计规范[S].
- [13] KW. MULTIPROG Quick start guide[G]. KW-Software GmbH. 2003.
- [14] 施耐德电气(中国)投资有限公司. Concept 2.6 用户手册[G]. 2003.
- [15] Sixnet. ISaGRAF Version 3.47 user's guide[G]. AlterSys Inc. 2003.
- [16] 一方梯队. OpenPCS V5.0.1 用户手册[G]. Infoteam Software GmbH. 2003.
- [17] 彭瑜. IEC 61131-3 的现状与发展[J]. 世界仪表与自动化. 2002(6):2-3
- [18] 彭瑜. 工控编程语言 IEC 61131-3 的现状与发展[J]. 国内外机电一体化技术. 2004(7):1.
- [19] 彭瑜. 充分发挥 SFC 的潜力,改善当前 PLC 程序的开发实践[J]. 国内外机电一体化技术. 2007(8).
- [20] 彭瑜. 工控编程语言国际标准 IEC 61131-3 及其影响[J]. 国内外机电一体化技术. 2006(4).
- [21] 彭瑜. 基于多平台编程系统的工控程序移植及其实现途径[J]. 国内外机电一体化技术. 2005(1).
- [22] 彭瑜. 关于 IEC 61131、PLC 和软 PLC 的一些观点——在中国的过去、现在和未来趋势[J]. 国内外机电一体化技术. 2003(1).
- [23] 何衍庆,戴自祥,等. 可编程序控制器原理及应用技巧[M]2版. 北京:化学工业出版社,2003.
- [24] 何衍庆,黎冰,黄海燕. 可编程控制器编程语言及应用[M]. 北京:电子工业出版社,2006.
- [25] 何衍庆,何乙平,王朋. 常用 PLC 应用手册[M]. 北京:电子工业出版社,2008.
- [26] <http://bbs.zidonghua.com.cn>. GE Fanuc 90-70 PLC 在宝钢初轧厂生产线改造中的应用[EB]. 自动化网论坛. 2005.11.
- [27] 胡学林. 可编程控制器教程(提高篇)[M]. 北京:电子工业出版社,2005.
- [28] 胡学林. 可编程控制器教程(实训篇)[M]. 北京:电子工业出版社,2004.
- [29] 高鸿斌,孔美静,赫孟合. 西门子 PLC 与工业控制网络应用[M]. 北京:电子工业出版社,2006.
- [30] 齐蓉编. 最新可编程控制器教程[M]. 西安:西安工业大学出版社,2000.
- [31] 宋伯生. PLC 编程理论、算法及技巧[M]. 北京:机械工业出版社,2005.
- [32] 陈在平,赵相宾. 可编程序控制器技术与应用系统设计[M]. 北京:机械工业出版社,2005.

[G e n e r a l I n f o r m a t i o n]

书名= I E C 6 1 1 3 1 - 3 编程语言及应用基础

作者=

页数=

出版日期=

出版社=

S S 号= 1 2 1 3 7 9 4 4

D X 号=

封面
书名
版权
前言
目录
第 1 章概述

1 . 1 P L C o p e n
1 . 1 . 1 P L C o p e n 组织
1 . 1 . 2 认证等级
1 . 1 . 3 P L C o P e n 的丁作
1 . 2 I E C 6 1 1 3 1 标准
1 . 2 . 1 I E C 6 1 1 3 1 的基本情况
1 . 2 . 2 I E C 6 1 1 3 1 - 3 编程语言
1 . 2 . 3 标准编程语言的特点

第 2 章公用元素和程序组织单元

2 . 1 软件模型、编程模型
2 . 1 . 1 软件模型
2 . 1 . 2 编程模型
2 . 2 公用元素
2 . 2 . 1 字符集
2 . 2 . 2 标识符
2 . 2 . 3 分界符
2 . 2 . 4 关键字
2 . 2 . 5 空格和注释
2 . 3 数据外部表示
2 . 3 . 1 数值文字
2 . 3 . 2 字符串文字
2 . 3 . 3 时间文字
2 . 4 数据类型
2 . 4 . 1 基本数据类型
2 . 4 . 2 一般数据类型
2 . 4 . 3 衍生数据类型
2 . 4 . 4 数据类型的允许取值范围和初始化
2 . 4 . 5 衍生数据类型的应用准则
2 . 5 变量
2 . 5 . 1 变量的表示
2 . 5 . 2 变量的属性和初始化
2 . 6 程序组织单元
2 . 6 . 1 函数
2 . 6 . 2 功能块
2 . 6 . 3 程序
2 . 6 . 4 程序组织单元

第 3 章文本类编程语言

3 . 1 文本类编程语言及其公用元素
3 . 1 . 1 文本类编程语言概述
3 . 1 . 2 文本类编程语言的公用元素
3 . 2 指令表编程语言
3 . 2 . 1 指令
3 . 2 . 2 函数和功能块
3 . 2 . 3 示例
3 . 3 结构化文本编程语言
3 . 3 . 1 结构化文本的表示
3 . 3 . 2 语句
3 . 3 . 3 示例

第 4 章图形类编程语言

- 4 . 1 图形类编程语言的公用元素
 - 4 . 1 . 1 线、模块和流向
 - 4 . 1 . 2 网络求值和执行控制元素
- 4 . 2 梯形图编程语言
 - 4 . 2 . 1 传统梯形图编程语言的缺点
 - 4 . 2 . 2 梯形图的组成元素
 - 4 . 2 . 3 梯形图的执行
 - 4 . 2 . 4 示例
- 4 . 3 功能块图编程语言
 - 4 . 3 . 1 功能块图图形符号和功能块的组合 - -
 - 4 . 3 . 2 功能块图的编程和执行
 - 4 . 3 . 3 示例

第 5 章顺序功能表图编程语言

- 5 . 1 顺序功能表图的三要素
 - 5 . 1 . 1 基本概念
 - 5 . 1 . 2 步
 - 5 . 1 . 3 转换
 - 5 . 1 . 4 有向连线
- 5 . 2 顺序功能表图的程序结构
 - 5 . 2 . 1 单序列结构
 - 5 . 2 . 2 选择序列结构
 - 5 . 2 . 3 并行序列结构
 - 5 . 2 . 4 不安全序列和不可达序列结构
- 5 . 3 顺序功能表图编程语言
 - 5 . 3 . 1 顺序功能表图的进展
 - 5 . 3 . 2 顺序功能表图的兼容
- 5 . 4 示例
 - 5 . 4 . 1 冲压机控制系统
 - 5 . 4 . 2 交通信号控制系统
 - 5 . 4 . 3 物料混合控制功能块
- 5 . 5 发挥 S F C 的潜力，改善当前 P L C 程序的开发实践
 - 5 . 5 . 1 程序开发过程中必须遵循的基本方法
 - 5 . 5 . 2 采用 S F C 作为设计系统控制的主要工具
 - 5 . 5 . 3 重视 S F C 语言与其他 4 种编程语言的互补关系
 - 5 . 5 . 4 S F C 与状态图的对应关系及异同
 - 5 . 5 . 5 S F C 与 P e t r i 网的关系及异同
- 5 . 6 顺序功能表图程序的转换
 - 5 . 6 . 1 顺序功能表图程序中基本序列的转换
 - 5 . 6 . 2 顺序功能表图程序中动作控制功能块的转换

第 6 章基本应用

- 6 . 1 电动机控制的编程
 - 6 . 1 . 1 电动机起保停控制
 - 6 . 1 . 2 电动机的正、反转控制
 - 6 . 1 . 3 电动机的星—三角转换控制
 - 6 . 1 . 4 三相步进电动机的相序控制
 - 6 . 1 . 5 电动机带反接制动电阻的可逆运转反接制动控制
- 6 . 2 定时器、计数器的编程
 - 6 . 2 . 1 电动机的定时运行
 - 6 . 2 . 2 长定时器的实现
 - 6 . 2 . 3 信号发生器的实现
 - 6 . 2 . 4 计数器的应用
- 6 . 3 比较函数的应用
 - 6 . 3 . 1 旋转定位控制
 - 6 . 3 . 2 分度盘工位控制
 - 6 . 3 . 3 自动增益控制

- 6 . 4 运算函数的应用
 - 6 . 4 . 1 控制算法
 - 6 . 4 . 2 运算函数的应用
- 6 . 5 移位函数的应用
 - 6 . 5 . 1 霓虹灯顺序点亮程序
 - 6 . 5 . 2 操作权限确认控制
 - 6 . 5 . 3 长循环移位控制

第 7 章编程举例

- 7 . 1 火力发电厂蒸汽轮机驱动给水泵的控制
 - 7 . 1 . 1 过程简介
 - 7 . 1 . 2 设计方法
 - 7 . 1 . 3 控制问题分解
 - 7 . 1 . 4 程序分解
 - 7 . 1 . 5 低层次功能块
 - 7 . 1 . 6 信号流
- 7 . 2 分选器控制
 - 7 . 2 . 1 分选器过程简介
 - 7 . 2 . 2 程序设计和优化

第 8 章实验

- 8 . 1 用梯形图和功能块图编程语言编写程序
- 8 . 2 用指令表和结构化文本编程语言编写程序
- 8 . 3 编写衍生功能块和调用功能块的程序
- 8 . 4 用顺序功能表图编程语言编写程序

附录

- 附录 A M U L T I P R O G 软件的使用
- 附录 B 习题和思考题
 - B . 1 填空题
 - B . 2 选择题
 - B . 3 编程和思考题

参考文献