Allen-Bradley

Основные инструкции программируемых контроллеров Logix5000™

1756 ControlLogix®,
1769 CompactLogix™,
1789 SoftLogix™,
1794 FlexLogix™, PowerFlex
700S with DriveLogix

Справочное руководство

Rockwell Automation

Важная информация для пользователя

В связи с большим разнообразием применений продуктов, описанных в данном документе, лица, ответственные за использование этих продуктов, должны обеспечить принятие всех необходимых мер по соблюдению всех эксплуатационных требований и требований техники безопасности для каждого такого использования, включая выполнение требований всех применимых законов, правил, норм и стандартов. Rockwell Automation ни в коем случае не отвечает за косвенный ущерб, связанный с использованием данных продуктов.

Все иллюстрации, схемы, примеры программ и примеры компоновки, приведенные в данном документе, используются лишь в качестве примера. Поскольку каждое конкретное оборудование характеризуется множеством специфических параметров и требований, Rockwell Automation не берет на себя обязательства или ответственность (включая ответственность по интеллектуальной собственности) за практическое использование продуктов на основе приведенных в данном документе примеров.

В публикации SGI-1.1 фирмы Allen-Bradley *«Руководство по обеспечению безопасности при использовании, установке и обслуживании полупроводниковых устройств управления»* (имеющуюся в вашем местном представительстве Rockwell Automation) описываются важные различия между полупроводниковым оборудованием и электромеханическими устройствами, которые необходимо учитывать при применении продуктов, описанных в данной публикации.

В настоящем документе используются примечания, обращающие ваше внимание на вопросы безопасности. Примечания, примеры обозначения которых приводятся ниже, помогают вам определить потенциальную опасность, избежать потенциальную опасность и понять последствия потенциальной опасности:

ВНИМАНИЕ

Обозначает информацию о способах действий или обстоятельствах, которые могут привести к травмам или смерти людей, материальному ущербу или экономическим потерям.

ВАЖНО!

Обозначает информацию, имеющую критическое значение для успешного применения и понимания продукта.

Введение

В этой версии данного документа содержится новая и обновленная информация. Эта информация обозначается вертикальной полосой, как показано для данного абзаца.

Обновленная информация

Данный документ содержит следующие изменения:

Изменение:	**Глава/ приложение:**
Указатель инструкций теперь содержит имя каждой инструкции	Указа-тель инструкций
Таблица, содержащая изменения в кодах ошибок PLC/SLC с R9.x и ниже по R10.x и выше.	3
Уточнения, касающиеся выбора опции кэширования для сообщения. Изменения включают количество соединений, которое вы можете поместить в кэш-память.	3
Дополнительные коды продукта для атрибута ProductCode объекта CONTROLLERDEVICE.	3
Новые атрибуты для объекта TASK: • DisableUpdateOutputs • EnableTimeOut • InhibitTask • OverlapCount • Status	3
Инструкция задания системного значения (Set System Value - SSV) теперь позволяет вам программно изменять приоритетность и частоту (периодичность) выполнения задачи.	3
Чтобы получить или установить значение тайм-аута для задачи обработки событий, используйте атрибут Rate (Частота) объекта TASK.	3
Инструкция немедленного вывода (Immediate Output - IOT). Используйте инструкцию IOT для немедленного обновления выходных данных или запуска задачи обработки событий в другом контроллере.	3
Объяснение влияния инструкции сброса SFC (SFC Reset - SFR) на хранимые операции.	10
Инструкция запуска задачи обработки событий (Trigger Event Task - EVENT). Используйте инструкцию EVENT для запуска выполнения задачи обработки событий.	10
Информация о том, как следует выбирать элементы функционального блока, включая такие элементы как IREF, OREF, ICON и OCON.	B
Уточненная информация по использованию инструкций функционального блока в режиме периодической синхронизации.	B
Пояснение различия между элементом CASE структурированного текста и оператором-переключателем C/C++.	C

Примечания:

Как найти инструкцию

Для поиска справочной информации по инструкциям Logix используйте нижеследующий указатель (закрашенные серым цветом инструкции описываются в других руководствах). В указателе также содержится информация об имеющихся для данной инструкции языках программирования.

Если указатель ссылается на:	Данная инструкция описывается в:
номер страницы	данном руководстве
управление процессом	Справочном руководстве по набору инструкций для приводов и управления процессом с помощью программируемых контроллеров Logix5000, публикация 1756-RM006
перемещение	Справочном руководстве по набору инструкций для управления перемещением с помощью программируемых контроллеров Logix5000, публикация 1756-RM007

Инструкция:	Местонахождение:	Языки:
ABL ASCII Test for Buffer Line (Проверка буфера на наличие символа завершения)	16-5	релейная логика структурированный текст
ABS Absolute Value (Абсолютная величина)	5-29	релейная логика структурированный текст функциональный блок
ACB ASCII Chars in Buffer (Подсчет числа символов ACSII в буфере)	16-10	релейная логика структурированный текст
ACL ASCII Clear Buffer (Очистка буфера/удаление инструкций ASCII)	16-10	релейная логика структурированный текст
ACOS Arc Cosine (Арккосинус)	13-14	структурированный текст
ACS Arc Cosine (Арккосинус)	13-14	релейная логика функциональный блок
ADD Add (Сложение)	5-6	релейная логика структурированный текст функциональный блок
AFI Always False Instruction (Всегда ложная инструкция)	10-23	релейная логика
AHL ASCII Handshake Lines (Квитирование линий ASCII)	16-12	релейная логика структурированный текст

Инструкция:	Местонахождение:	Языки:
ALM Alarm (Аварийная сигнализация)	управление процессом	структурированный текст функциональный блок
AND Bitwise AND (Поразрядное "И")	6-23	релейная логика структурированный текст функциональный блок
ARD ASCII Read (Чтение фиксированного количества символов ASCII) релейная логика	16-16	релейная логика структурированный текст
ARL ASCII Read Line (Чтение переменного количества символов ASCII)	16-19	релейная логика структурированный текст
ASIN Arc Sine (Арксинус)	13-11	структурированный текст
ASN Arc Sine (Арксинус)	13-11	релейная логика функциональный блок
ATAN Arc Tangent (Арктангенс)	13-17	структурированный текст
ATN Arc Tangent (Арктангенс)	13-17	релейная логика функциональный блок
AVE File Average (Файловое усреднение)	7-38	релейная логика
AWA ASCII Write Append (Присоединение к ASCII при записи)	16-23	релейная логика структурированный текст

Инструкция:	Местонахождение:	Языки:
AWT ASCII Write (Запись ASCII)	16-28	релейная логика структурированный текст
BAND Boolean AND (Булево "И")	6-35	структурированный текст функциональный блок
BNOT Boolean NOT (Булево "НЕ")	6-44	структурированный текст функциональный блок
BOR Boolean OR (Булево "ИЛИ")	6-38	структурированный текст функциональный блок
BRK Break (Прерывание выполнения)	11-5	релейная логика
BSL Bit Shift Left (Сдвиг бита влево)	8-2	релейная логика
BSR Bit Shift Right (Сдвиг бита вправо)	8-5	релейная логика
BTD Bit Field Distribute (Распределение битовых колец)	6-11	релейная логика
BTDT Bit Field Distribute with Target (Перемещение битов в функциональном блоке)	6-14	структурированный текст функциональный блок
BTR Message (Сообщение)	3-2	релейная логика структурированный текст
BTW Message (Сообщение)	3-2	
BXOR Boolean Exclusive OR (Булево "исключающее ИЛИ")	6-41	структурированный текст функциональный блок
CLR Clear (Очистка)	6-17	релейная логика структурированный текст
CMP Compare (Сравнение)	4-2	релейная логика
CONCAT String Concatenate (Присоединение к строке)	17-3	релейная логика структурированный текст
COP Copy File (Копирование файла)	7-28	релейная логика структурированный текст
COS Cosine (Косинус)	13-5	релейная логика структурированный текст
CPS Synchronous Copy File (Синхронное копирование файла)	7-28	релейная логика структурированный текст
CPT Compute (Вычисление)	5-2	релейная логика
CTD Count Down (Обратный счет)	2-28	релейная логика
CTU Count Up (Прямой счет)	2-24	релейная логика
CTUD Count Up/Down (Прямой/обратный счет)	2-32	структурированный текст функциональный блок
D2SD Дискретное устройство с 2 состояниями управление процессом	управление процессом	структурированный текст функциональный блок
D3SD Дискретное устройство с 3 состояниями управление процессом	управление процессом	
DDT Diagnostic Detect (Диагностика)	12-10	релейная логика
DEDT Время простоя управление процессом	управление процессом	структурированный текст функциональный блок
DEG Degrees (Градусы)	15-2	релейная логика структурированный текст функциональный блок
DELETE String Delete (Удаление строки)	17-5	релейная логика структурированный текст
DERV Derivative (Производная)	управление процессом	управление процессом структурированный текст функциональный блок
DFF D Flip-Flop (D-триггео)	управление процессом	управление процессом структурированный текст функциональный блок
DIV Divide (Деление)	5-15	релейная логика структурированный текст функциональный блок
DTOS DINT to String (Преобразование значения DINT в строку ASCII)	18-8	релейная логика структурированный текст

Инструкция:	Местонахождение:	Языки:
DTR Data Transitional (Изменение данных)	12-18	релейная логика
EOT End of Transition (Завершение перехода)	10-25	релейная логика структурированный текст
EQU Equal to (Равно)	4-7	релейная логика структурированный текст функциональный блок
ESEL Enhanced Select (Улучшенный выбор)	управление процессом	управление процессом структурированный текст функциональный блок
EVENT Trigger Event Task (Запуск задачи обработки событий)	10-31	релейная логика структурированный текст
FAL File Arithmetic and Logic (Файловая арифметика и логика)	7-7	релейная логика
FBC File Bit Comparison (Файловое сравнение битов)	12-2	релейная логика
FFL FIFO Load (Загрузка FIFO)	8-8	релейная логика
FFU FIFO Unload (Выгрузка FIFO)	8-14	релейная логика
FGEN Function Generator (Генератор функций)	управление процессом	управление процессом структурированный текст функциональный блок
FIND Find String (Поиск строки)	17-7	релейная логика структурированный текст
FIL File Fill (Заполнение массива данными)	7-34	релейная логика
FOR For (Цикл For)	11-2	релейная логика
FRD Convert to Integer (Преобразование в целое число)	15-9	релейная логика функциональный блок
FSC File Search and Comapre (Файловый поиск и сравнение)	7-19	релейная логика

Инструкция:	Местонахождение:	Языки:
GEQ Greater than or Equal to (Больше или равно)	4-11	релейная логика структурированный текст функциональный блок
GRT Greater Than (Больше)	4-15	релейная логика структурированный текст функциональный блок
GSV Get System Value (Получить системное значение)	3-34	релейная логика структурированный текст
HLL High/Low Limit (Высокий/низкий предел)	управление процессом	структурированный текст функциональный блок
HPF High Pass Filter (Фильтр верхних частот)	управление процессом	структурированный текст функциональный блок
ICON Input Wire Connector (Входной соединитель)	B-1	функциональный блок
INSERT Insert String (Вставить строку)	17-9	релейная логика структурированный текст
INTGR Integrator (Интегратор)	управление процессом	структурированный текст функциональный блок
IOT Immediate Output (Немедленный вывод)	3-57	релейная логика структурированный текст
IREF Input Reference (Входная ссылка)	B-1	функциональный блок
JKFF JK Flip-Flop (JK-триггер)	управление процессом	структурированный текст функциональный блок
JMP Jump to Label (Переход к метке)	10-2	релейная логика
JSR Jump to Subroutine (Переход к подпрограмме)	10-4	релейная логика структурированный текст функциональный блок
JXR Jump to External Routine (Переход к внешней процедуре)	10-14	релейная логика
LBL Label (Метка)	10-2	релейная логика

Инструкция:	Местонахождение:	Языки:
LDL2 Second Order Lead Lag (Опережение/задержка второго порядка)	управление процессом	структурированный текст функциональный блок
LDLG Lead-Lag (Опережение/задержка)	управление процессом	структурированный текст функциональный блок
LEQ Less Than or Equal to (Меньше или равно)	4-19	релейная логика структурированный текст функциональный блок
LES Less Than (Меньше)	4-23	релейная логика структурированный текст функциональный блок
LFL LIFO Load (Загрузка в LIFO)	8-20	релейная логика
LFU LIFO Unload (Выгрузка в LIFO)	8-26	релейная логика
LIM Limit (Предел)	4-27	релейная логика функциональный блок
LN Natural Log (Натуральный логарифм)	14-2	релейная логика структурированный текст функциональный блок
LOG Log Base 10 (Десятмчный логарифм)	14-4	релейная логика структурированный текст функциональный блок
LOWER Lower case (Нижний регистр)	18-14	релейная логика структурированный текст
LPF Low Pass Filter (Фильтр нижних частот)	управление процессом	структурированный текст функциональный блок
MAAT Motion Apply Axis Tuning (Настройка оси при применении перемещения)	перемещение	релейная логика структурированный текст
MAFR Motion Axis Fault Reset (Сброс ошибки оси перемещения)	перемещение	релейная логика структурированный текст
MAG Motion Axis Gear (Привод оси перемещения)	перемещение	релейная логика структурированный текст
MAH Motion Axis Home (Начало оси перемещения)	перемещение	релейная логика структурированный текст
MAHD Motion Apply Hookup Diagnostics (Подключение диагностики при применении перемещения)	перемещение	релейная логика структурированный текст
MAJ Motion Axis Jog (Отрывистое перемещение оси перемещения)	перемещение	релейная логика структурированный текст
MAM Motion Axis Mov (Перемещение оси перемещения)	перемещение	релейная логика структурированный текст
MAOC Motion Arm Output Cam (Разрешение выходного кулачка перемещения)	перемещение	релейная логика структурированный текст
MAPC Motion Axis Position Cam (Кулачок положения оси перемещения)	перемещение	релейная логика структурированный текст
MAR Motion Arm Registration (Разрешение регистрации перемещения)	перемещение	релейная логика структурированный текст
MAS Motion Axis Stop (Останов оси перемещения)	перемещение	релейная логика структурированный текст
MASD Motion Axis Shutdown (Выключение оси перемещения)	перемещение	релейная логика структурированный текст
MASR Motion Axis Shutdown Reset (Сброс выключения оси перемещения)	перемещение	релейная логика структурированный текст
MATC Motion Axis Time Cam (Кулачок времени оси перемещения)	перемещение	релейная логика структурированный текст
MAVE Moving Average (Перемещение среднего)	управление процессом	структурированный текст функциональный блок
MAW Motion Arm Watch (Разрешение слежения за перемещением)	перемещение	релейная логика структурированный текст
MAXC Maximum Capture (Максимальный захват)	управление процессом	структурированный текст

Инструкция:	Местонахождение:	Языки:
MCCD Motion Coordinated Change Dynamics (Динамика скоординированных изменений при перемещении)	перемещение	релейная логика структурированный текст
MCCM Motion Coordinated Circular Move (Скоординированное круговое движение при перемещении)	перемещение	релейная логика структурированный текст
MCCP Motion Calculate Cam Profile (Расчет профиля кулачка при перемещении)	перемещение	релейная логика структурированный текст
MCD Motion Change Dynamics (Динамика изменений при перемещении)	перемещение	релейная логика структурированный текст
MCLM Motion Coordinated Linear Move (Скоординированное линейное движение при перемещении)	перемещение	релейная логика структурированный текст
MCR Master Control Reset (Сброс управляющей программы)	10-19	релейная логика
MCS Motion Coordinated Stop (Скоординированный останов перемещения)	перемещение	релейная логика структурированный текст
MCSD Motion Coordinated Shutdown (Скоординированное выключение перемещения)	перемеще-ие	релейная логика структурированный текст
MCSR Motion Coordinated Shutdown Reset (Сброс скоординированного выключения перемещения)	перемещение	релейная логика структурированный текст
MDF Motion Direct Drive Off (Отключение непосредственного привода перемещения)	перемещение	релейная логика структурированный текст
MDO Motion Direct Drive On (Включение непосредственного привода перемещения)	перемещение	релейная логика структурированный текст
MDOC Motion Disarm Output Cam (Запрещение выходного кулачка при перемещении)	перемещение	релейная логика структурированный текст
MDR Motion Disarm Registration (Запрещение регистрации перемещения)	перемещение	релейная логика структурированный текст
MDW Motion Disarm Watch (Запрещение слежения за перемещением)	перемещение	релейная логика структурированный текст
MEQ Mask Equal to (Маска равна)	4-33	релейная логика структурированный текст функциональный блок
MGS Motion Group Stop (Групповой останов перемещения)	перемещение	релейная логика структурированный текст
MGSD Motion Group Shutdown (Групповое выключение перемещения)	перемещение	релейная логика структурированный текст
MGSP Motion Group Strobe Position (Стробирование положения при групповом перемещении)	перемещение	релейная логика структурированный текст
MGSR Motion Group Shutdown Reset (Сброс выключения группового перемещения)	перемещение	релейная логика структурированный текст
MID Middle String (Средняя строка)	17-11	релейная логика структурированный текст
MINC Minimum Capture (Минимальный захват)	управление процессом	структурированный текст функциональный блок
MOD Module (Модуль)	5-19	релейная логика структурированный текст функциональный блок
MOV Move (Перемещение)	6-3	релейная логика
MRAT Motion Run Axis Tuning (Настройка оси при выполнении перемещения)	перемещение	релейная логика структурированный текст
MRHD Motion Run Hookup Diagnostics (Подключение диагностики при выполнении перемещения)	перемещение	релейная логика структурированный текст

Инструкция:	Местонахождение:	Языки:
MRD Motion Redefine Position (Переопределение положения при перемещении)	перемещение	релейная логика структурированный текст
MSF Motion Servo Off (Отключение сервопривода перемещения)	перемещение	релейная логика структурированный текст
MSG Message (Сообщение)	3-2	релейная логика структурированный текст
MSO Motion Servo On (Включение сервопривода перемещения)	перемещение	релейная логика структурированный текст
MSTD Moving Standard Deviation (Перемещение стандартного отклонения)	управление процессом	структурированный текст функциональный блок
MUL Multiply (Умножение)	5-12	релейная логика структурированный текст функциональный блок
MUX Multiplexer (Мультиплексор)	управление процессом	функциональный блок
MVM Masked Moved (Маскированное перемещение)	6-5	релейная логика
MVMT Masked Move with Target (Маскированное перемещение с местом назначения)	6-8	структурированный текст функциональный блок
NEG Negate (Отрицание)	5-26	релейная логика структурированный текст функциональный блок
NEQ Not Equal to (Не равно)	4-38	релейная логика структурированный текст функциональный блок
NOP No Operation (Пустая операция)	10-24	релейная логика
NOT Bitwise NO (Побитовое "НЕ")	6-32	релейная логика структурированный текст функциональный блок
NITCH Notch Filter (Измерительный фильтр)	управление процессом	структурированный текст функциональный блок

Инструкция:	Местонахождение:	Языки:
OCON Output Wire Connector (Выходной соединитель)	B-1	функциональный блок
ONS One Shot (Одно включение)	1-12	релейная логика
OR Bitwise OR (Разрядное "ИЛИ")	6-26	релейная логика структурированный текст функциональный блок
OREF Output Reference (Выходная ссылка)	B-1	функциональный блок
OSF One Shot Falling (Одно включение по заднему фронту)	1-17	релейная логика
OSFI One Shot Falling with Input (Одно включение по заднему фронту от входа)	1-22	структурированный текст функциональный блок
OSR One Shot Rising (Одно включение по переднему фронту)	1-15	релейная логика
OSRI One Shot Rising with Input (Установка выходного бита при переключении входного бита)	1-19	структурированный текст функциональный блок
OTE Output Energize (Выход включить)	1-6	релейная логика
OTL Output Latch (Фиксация выхода)	1-8	релейная логика
OTU Output Unlatch (Расфиксация выхода)	1-10	релейная логика
PI Proportional + Integral (Пропорциональный + интегральный)	управление процессом	структурированный текст функциональный блок
PID Proportional Integral Derivative (ПИД регулятор)	12-21	релейная логика структурированный текст
PIDE Enhanced PID (Улучшенный ПИД)	управление процессом	структурированный текст функциональный блок
PMUL Pulse Multiplier (Умножитель импульсов)	управление процессом	структурированный текст функциональный блок

Инструкция:	Местонахождение:	Языки:
POSP Position Proportional (Позиционно-пропорциональный)	управление процессом	структурированный текст функциональный блок
RAD Radians (Радианы)	15-4	релейная логика структурированный текст функциональный блок
RES Reset (Сброс)	2-36	релейная логика
RESD Reset Dominant (Доминанта сброса)	управление процессом	структурированный текст функциональный блок
RET Return (Возврат)	10-4 и 11-6	релейная логика структурированный текст функциональный блок
RLIM Rate Limiter (Ограничитель расхода)	управление процессом	структурированный текст функциональный блок
RMPS Ramp/Soak (Линейное изменение/выдержка)	управление процессом	структурированный текст функциональный блок
RTO Retentive Timer On (Таймер с сохранением времени включения)	2-10	релейная логика
RTOR Retentive Timer On with Reset (Таймер с сохранением времени включения со сбросом)	2-20	структурированный текст функциональный блок
RTOS REAL to String (Преобразование данных типа REAL в строку ASCII)	18-10	релейная логика структурированный текст
SBR Subroutine (Подпрограмма)	10-4	релейная логика структурированный текст функциональный блок
SCL Scale (Масштаб)	управление процессом	структурированный текст функциональный блок
SCRV S-Curve (S-образная кривая)	управление процессом	структурированный текст функциональный блок
SEL Select (Выбрать)	управление процессом	функциональный блок
SETD Set Dominant (Задать доминанту)	управление процессом	структурированный текст функциональный блок
SFP SFC Pause (Пауза SFC)	10-27	релейная логика структурированный текст
SFR SFC Reset (Сброс SFC)	10-29	релейная логика структурированный текст
SIN Sine (Синус)	13-2	релейная логика структурированный текст функциональный блок
SIZE Size In Elements (Размер в элементах)	7-53	релейная логика структурированный текст
SNEG Selected Negate (Выбранное отрицание)	управление процессом	структурированный текст функциональный блок
SOC Second-Order Controller (Контроллер второго порядка)	управление процессом	структурированный текст функциональный блок
SQI Sequencer Input (Секвенсер входа)	9-2	релейная логика
SQL Sequencer Load (Загрузка секвенсера)	9-10	релейная логика
SQO Sequencer Output (Секвенсер выхода)	9-6	релейная логика
SQR Square Root (Квадратный корень)	5-23	релейная логика функциональный блок
SQRT Square Root (Квадратный корень)	5-23	структурированный текст
SRT File Sort (Сортировка файла)	7-43	релейная логика структурированный текст
SRTP Split Range Time Proportional (Разбиение диапазона пропорционально времени)	управление процессом	структурированный текст функциональный блок
SSUM Selected Summer (Выбранный сумматор)	управление процессом	структурированный текст функциональный блок
SSV Set System Value (Задать системное значение)	3-34	релейная логика структурированный текст

Инструкция:	Местонахождение:	Языки:
STD File Standard Deviation (Стандартное отклонение для массива)	7-48	релейная логика
STOD String to DINT (Преобразование строки ASCII в DINT)	18-4	релейная логика структурированный текст
STOR String to REAL (Преобразование строки ASCII в REAL)	18-6	релейная логика структурированный текст
SUB Subtract (Вычитание)	5-9	релейная логика структурированный текст функциональный блок
SWPB Swap Byte (Переставить байт)	6-19	релейная логика структурированный текст
TAN Tangent (Тангенс)	13-8	релейная логика структурированный текст функциональный блок
TND Temporary End (Временный конец)	10-17	релейная логика
TOD Convert to BCD (Преобразование в код)	15-6	релейная логика функциональный блок
TOF Timer Off Delay (Таймер с выдержкой на отключение)	2-6	релейная логика
TOFR Timer Off Delay with Reset (Таймер с выдержкой на отключение со сбросом)	2-17	структурированный текст функциональный блок
TON Timer On Delay (Таймер с выдержкой на включение)	2-2	релейная логика
TONR Timer On Delay with Reset (Таймер с выдержкой на включение со сбросом)	2-14	структурированный текст функциональный блок
TOT Totalizer (Суммирующее устройство)	управление процессом	структурированный текст функциональный блок
TRN Truncate (Усечение)	15-11	релейная логика функциональный блок
TRUNC Truncate (Усечение)	15-11	структурированный текст
UID User Interrupt Disable (Запрещение прерывания))	10-21	релейная логика структурированный текст
UIE User Interrupt Enable (Разрешение прерывания)	10-21	релейная логика структурированный текст
UPDN Up/Down Accumulator (Сумматор в прямом/обратном порядке)	управление процессом	структурированный текст функциональный блок
UPPER Upper Case (Верхний регистр)	18-12	релейная логика структурированный текст
XIC Examine if Closed (Проверить на состояние ВКЛ)	1-2	релейная логика
XID Examine if Open (Проверить на состояние ОТКЛ)	1-4	релейная логика
XOR Bitwise Exclusive OR (Поразрядное "исключающее ИЛИ")	6-29	релейная логика структурированный текст функциональный блок
XPY X to the Power of Y (X в степени Y)	14-6	релейная логика структурированный текст функциональный блок

Введение

Настоящее руководство является одним из нескольких документов по инструкциям Logix5000.

Задача:	Документы:
Программирование контроллера для приложений с последовательным доступом. Вы находитесь здесь 	*Справочное руководство по основным инструкциям контроллеров Logix 5000*, публикация 1756-RM003
Программирование контроллера для приложений по управлению процессом или приводами	*Справочное руководство по инструкциям для приводов и управления процессом с помощью контроллеров Logix5000*, публикация 1756-RM006
Программирование контроллера для приложений по перемещению	*Справочное руководство по набору инструкций для управления перемещением с помощью контроллеров Logix5000*, публикация 1756-RM007
Импорт текстового файла или тегов в проект	*Справочное руководство по импорту/экспорту для контроллеров Logix5000*
Экспорт проекта или тегов в текстовый файл	
Преобразование приложения для PLC-5 или SLC 500 в приложение для Logix5000	*Справочное руководство по преобразованию логики PLC-5 или SLC 500 в логику Logix5550 для контроллеров Logix5000*

Для кого предназначено это руководство

Этот документ предоставляет программисту подробную информацию по каждой имеющейся инструкции для контроллера на основе Logix. Необходимо предварительно ознакомиться с тем, как контроллер на основе Logix хранит и обрабатывает данные.

Начинающим программистам перед использованием какой-либо инструкции следует прочитать всю информацию о ней. Опытные программисты могут обращаться к описанию инструкций для уточнения деталей.

Назначение данного руководства

В этом руководстве каждая инструкция описывается в следующем формате:

В этом разделе:	Содержится информация следующего рода:
Имя инструкции	идентифицирует инструкцию определяет, является ли данная инструкция инструкцией входа или выхода
Операнды	указываются все операнды данной инструкции  если она имеется в релейной логике, описываются соответствующие операнды  если она имеется в структурированном тексте, описываются соответствующие операнды  если она имеется в функциональном блоке, описываются соответствующие операнды. Показанные в функциональном блоке по умолчанию выводы используются лишь по умолчанию. В таблице операндов перечисляются все возможные выводы для функционального блока.
Структура инструкции	указываются управляющие биты состояния и значения инструкции, если они имеются
Описание	описывается использование инструкции даются различия для разрешенного и запрещенного состояния инструкции, где это применимо
Арифметические флаги состояния	указывается, влияет ли данная инструкция на арифметические флаги состояния
Условия ошибки	указывается, генерирует ли инструкция неосновные или основные ошибки если да, указывается тип ошибки и ее код
Выполнение	указываются подробности выполнения инструкции
Пример	дается как минимум один пример программы на каждом из имеющихся языков каждый пример сопровождается пояснением

Следующие иконки помогают идентифицировать информацию, относящуюся к конкретному языку программирования:

Эта иконка:	Указывает на следующий язык программирования:
	релейная логика
	структурированный текст
	функциональный блок

Общая информация для всех инструкций

Набор инструкций Logix5000 имеет ряд общих атрибутов:

За этой информацией:	Обращайтесь к следующему приложению:
общие атрибуты	приложение "Общие атрибуты" определяет: арифметические флаги состояния · типы данных · ключевые слова
атрибуты функционального блока	приложение "Атрибуты функционального блока" определяет: · программный и операторский контроль · режимы задания времени

Используемые условные обозначения и терминология

Установлен и сброшен

В этом руководстве термины "установлен" и "сброшен" используются для определения состояния битов (булевых переменных) и значений (небулевых переменных):

Этот термин:	Означает следующее:
установлен	**бит установлен на 1 (ВКЛ)** **значение установлено на любое ненулевое число**
сброшен	**бит установлен на 0 (ВЫКЛ)** **все биты значения установлены на 0**

Если операнд или параметр поддерживает несколько типов данных, оптимальные типы данных выделяются **жирным шрифтом**. Инструкция выполняется быстрее и требует меньший объем памяти, если все операнды инструкции используют один и тот же оптимальный тип данных, обычно DINT или REAL.

Состояние цепочки релейной логики

Контроллер анализирует инструкции релейной логики, исходя из состояния цепочки перед данной инструкцией (входного условия цепочки). На основе инструкции и входного условия цепочки контроллер устанавливает условие цепочки после инструкции (выходное условие цепочки), которое, в свою очередь, влияет на всякую последующую инструкцию.

Если условие цепочки перед инструкцией - "истина", контроллер анализирует инструкцию и устанавливает выходное состояние цепочки на основе результатов ее выполнения. Если результатом инструкции является "истина", то выходное состояние цепочки - "истина", если же результатом инструкции является "ложь", то выходное состояние цепочки - "ложь".

Также контроллер выполняет предварительное сканирование инструкций. Предварительное сканирование - это специальное сканирование всех процедур в контроллере. В процессе предварительного сканирования контроллер сканирует все главные процедуры и подпрограммы, но игнорирует переходы, при которых может быть пропущено выполнение инструкций. Контроллер выполняет все циклы FOR и вызовы подпрограмм. Если какая-либо подпрограмма вызывается более одного раза, она выполняется при каждом вызове. Контроллер использует предварительное сканирование инструкций релейной логики для сброса не сохраняемого ввода-вывода и внутренних значений.

При предварительном сканировании входные значения не являются текущими, а выходные данные не записываются. Предварительное сканирование вызывается следующими условиями:

- Переключение из режима Program (Программирование) в режим Run (Выполнение)
- Автоматический переход в режим Run после включения питания.

Предварительное сканирование для программы не выполняется в следующих случаях:

- Эта программа становится запланированной в процессе работы контроллера.
- Эта программа является не запланированной при переходе контроллера в режим Run.

Состояния функционального блока

ВАЖНО!

При программировании в функциональном блоке ограничивайте диапазон для технических единиц значениями в $+/-10^{+/-15}$, так как внутренние вычисления с плавающей точкой выполняются с использованием плавающей точки одинарной точности. Выход технических единиц за пределы этого диапазона может привести к потере точности, если результаты приблизятся к ограничениям плавающей точки одинарной точности ($+/-10^{+/-38}$).

Контроллер оценивает инструкции функционального блока на основе состояния различных условий.

Возможное состояние:	Описание:
предварительное сканирование	Предварительное сканирование процедур функционального блока аналогично предварительному сканированию процедур релейной логики. Единственным различием является то, что в процессе предварительного сканирования происходит сброс параметра EnableIn для каждой инструкции функционального блока.
первое сканирование инструкции	Первым сканированием инструкции называется первое выполнение инструкции после предварительного сканирования. Контроллер использует первое сканирование инструкции для считывания текущих входных данных и определения соответствующего состояния, в котором он должен находиться.
первое выполнение инструкции	Первым выполнением инструкции называется первое выполнение инструкции с новым экземпляром структуры данных. Контроллер использует первое выполнение инструкции для генерации коэффициентов и других хранимых данных, которые остаются неизменными для функционального блока после первоначальной загрузки.

Каждая инструкция функционального блока также включает параметры EnableIn и EnableOut:

- при установленном параметре EnableIn инструкции функционального блока выполняются нормальным образом.
- при сброшенном параметре EnableIn инструкция функционального блока выполняет либо предварительное сканирование логики, либо постсканирование логики, либо просто пропускает нормальное выполнение алгоритма.
- EnableOut является зеркальным отражением EnableIn, однако, если при выполнении функционального блока обнаруживается состояние переполнения, то EnableOut также сбрасывается.
- выполнение функционального блока возобновляется с того места, где оно было остановлено, когда EnableIn переключается со сброшенного состояния на установленное. Однако имеется ряд инструкций функционального блока, реализующих специальные функции, такие как повторная инициализация, когда EnableIn переключается со сброшенного состояния на установленное. Инструкции функционального блока с

временными параметрами в режиме задания времени Oversample всегда возобновляют выполнение с того места, где оно было оставлено, когда EnableIn переключается со сброшенного состояния на установленное.

Если параметр EnableIn не подключен, то инструкция всегда будет выполняться нормальным образом, а EnableIn останется установленным. Если вы сбросите EnableIn, он перейдет в установленное состояние при очередном выполнении данной инструкции.

Примечания:

Глава 1

Битовые инструкции (XIC, XIO, OTE, OTL, OUT, ONS, OSR, OSF, OSRI, OSFI)

Глава 2

Инструкции таймера и счетчика (TON, TOF, RTO, TONR, TOFR, RTOR, CTU, CTD, CTUD, RES)

Глава 3

Инструкции ввода/вывода (MSG, GSV, SSV, IOT)

Глава 4

Инструкции сравнения (CMP, EQU, GEQ, GRT, LEQ, LES, LIM, MEQ, NEQ)

Глава 5

Инструкции вычислений/ математических операций (CPT, ADD, SUB, MUL, DIV, MODE, SQR, SQRT, NEG, ABS)

Глава 6

Инструкции перемещения/ логические инструкции (MOV, MVM, BTD, MVMT, BTDT, CLR, SWPB, AND, OR, XOR, NOT, BAND, BOR, BXOR, BNOT)

Глава 7

Инструкции Массива (Файла)/Прочие (FAL, FSC, COP, CPS, FLL, AVE, SRT, STD, SIZE)

Глава 8

Инструкции Массива (Файла)/Сдвига (BSL, BSR, FFL, FFU, LFL, LFU)

Глава 9

Инструкции секвенсеров (SQI, SQO, SQL)

Глава 10

Инструкции программного управления (JMP, LBL, JSR, RET, SBR, JXR, TND, MCR, UID, UIE, AFI,NOP, EOT, SFP, SFR, EVENT)

Глава 11

Инструкции FOR/ BREAK (FOR, FOR...DO, BRK, EXIT, RET)

Глава 12

Специальные инструкции (FBC, DDT, DTR, PID)

Глава 13

Тригонометрические инструкции (SIN, COS, TAN, ASN, ASIN, ACS, ACOS, ATN, ATAN)

Глава 14

Научные математические инструкции (LN, LOG, XPY)

Глава 15

Математические инструкции преобразования (DEG, RAD, TOD, FRD, TRN, TRUNC)

Глава 16

Инструкции ASCII последовательного порта (ABL, ACB, ACL, AHL, ARD, ARL, AWA, AWT)

Глава 17

Строковые инструкции ASCII (CONCAT, DELETE, FIND, INSERT, MID)

Глава 18

Инструкции преобразования ASCII (STOD, STOR, DTOS, RTOS, UPPER, LOWER)

Приложение A

Общие атрибуты

Приложение B

Атрибуты функционального блока

Приложение C

Программирование структурированного текста

Битовые инструкции (XIC, XIO, OTE, OTL, OUT, ONS, OSR, OSF, OSRI, OSFI)

Введение

Используйте битовые (релейные) инструкции для контроля и управления состоянием битов.

Если вы хотите:	Используйте инструкцию:	Имеющуюся в языках:	См. страницу:
разрешить выходы при установленном бите	XIC	релейная логика структурированный текст (1)	1-2
разрешить выходы при сброшенном бите	XIO	релейная логика структурированный текст (1)	1-4
установить бит	OTE	релейная логика структурированный текст (1)	1-6
установить бит (с сохранением)	OTL	релейная логика структурированный текст (1)	1-8
сбросить бит (с сохранением)	OTU	релейная логика структурированный текст (1)	1-10
разрешить выходы для одного сканирования всякий раз, когда цепочка получает значение "истина"	ONS	релейная логика структурированный текст (1)	1-12
устанавливать бит для одного сканирования всякий раз, когда цепочка получает значение "истина"	OSR	релейная логика	1-15
устанавливать бит для одного сканирования всякий раз, когда цепочка получает значение "ложь"	OSF	релейная логика	1-17
устанавливать бит для одного сканирования всякий раз, когда в функциональном блоке устанавливается входной бит	OSRI	структурированный текст функциональный блок	1-19
устанавливать бит для одного сканирования всякий раз, когда в функциональном блоке сбрасывается входной бит	OSFI	структурированный текст функциональный блок	1-22

(1) Эквивалентная инструкция структурированного текста отсутствует. Для достижения того же результата используйте другие средства программирования структурированного текста. См. описание соответствующей инструкции.

Examine If Closed (XIC) (Проверить на состояние ВКЛ)

Инструкция XIC проверяет, установлен ли бит данных.

Операнды:

Релейная логика

Операнд:	Тип:	Формат:	Описание:
бит данных	BOOL	тег	проверяемый бит

Структурированный текст

В структурированном тексте инструкция XIC отсутствует, но можно получить тот же результат с помощью конструкции IF...THEN.

```
IF  data_bit  THEN
     <statement>;
END_IF;
```

Информацию о синтаксисе конструкций структурированного текста можно найти в Приложении C.

Описание:

Инструкция XIC проверяет, установлен ли бит данных.

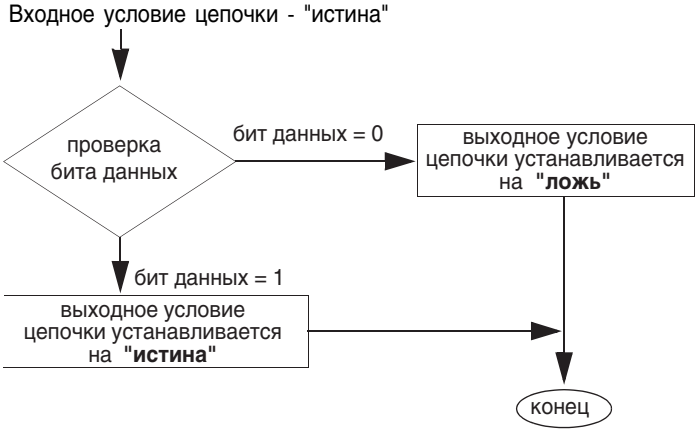
Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

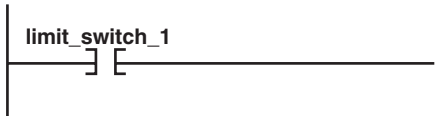
отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Выходное условие цепочки устанавливается на "ложь"
входное условие цепочки - "ложь"	Выходное условие цепочки устанавливается на "ложь"
Входное условие цепочки - "истина"	
постсканирование	Выходное условие цепочки устанавливается на "ложь"

Пример 1: Если установлен *limi_switch_1*, то это разрешает следующую по порядку инструкцию (выходное условие цепочки - "истина").

Релейная логика

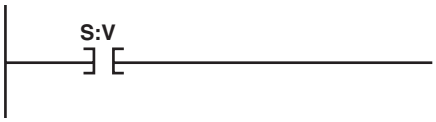


Структурированный текст

```
IF  limit_switch  THEN
     <statement>;
END_IF;
```

Пример 2: Если установлен S:V (что указывает на переполнение), то разрешается следующая по порядку инструкция (выходное условие цепочки - "истина").

Релейная логика



Структурированный текст

```
IF  S:V  THEN
     <statement>;
END_IF;
```

Examine if Open (XIO) (Проверить на состояние ОТКЛ)

Инструкция XIO проверяет, сброшен ли бит данных.

Операнды:

Релейная логика

Операнд:	Тип:	Формат:	Описание:
бит данных	BOOL	тег	проверяемый бит

Структурированный текст

В структурированном тексте инструкция XIO отсутствует, но можно получить тот же результат с помощью конструкции IF...THEN.

```
IF  NOT  data_bit  THEN
     <statement>;
END_IF;
```

Информацию о синтаксисе конструкций структурированного текста можно найти в Приложении C.

Описание:

Инструкция XIO проверяет, установлен ли бит данных.

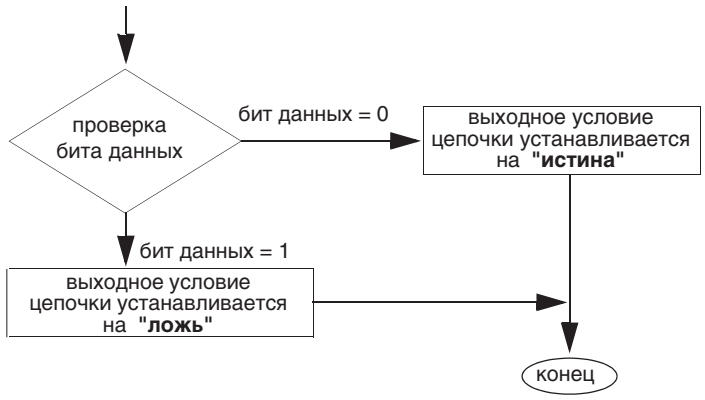
Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

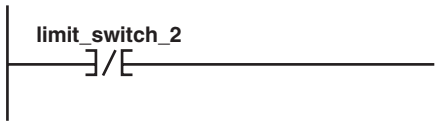
отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Выходное условие цепочки устанавливается на "ложь"
входное условие цепочки - "ложь"	Выходное условие цепочки устанавливается на "ложь"
входное условие цепочки - "истина"	
постсканирование	Выходное условие цепочки устанавливается на "ложь"

Пример 1: Если сброшен *limi_switch_2*, то это разрешает следующую по порядку инструкцию (выходное условие цепочки - "истина").

Релейная логика

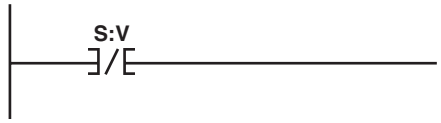


Структурированный текст

```
IF  NOT  limit_switch_2  THEN
     <statement>;
END_IF;
```

Пример 2: Если сброшен *S:V* (что указывает на отсутствие переполнения), то разрешается следующая по порядку инструкция (выходное условие цепочки - "истина").

Релейная логика



Структурированный текст

```
IF  NOT  S:V  THEN
     <statement>;
END_IF;
```

Output Energize (OTE) (Выход включить)

Инструкция OTE устанавливает или сбрасывает бит данных.

Операнды:

Релейная логика

Операнд:	Тип:	Формат:	Описание:
бит данных	BOOL	тег	устанавливаемый или сбрасываемый бит

Структурированный текст

В структурированном тексте инструкция OTE отсутствует, но можно получить тот же результат с помощью присваивания без сохранения.

```
data_bit [:=] BOOL_expression;
```

Информацию о синтаксисе операций присваивания и выражений в структурированном тексте можно найти в Приложении C.

Описание:

Когда инструкция OTE разрешена, контроллер устанавливает бит данных.
Когда инструкция OTE запрещена, контроллер сбрасывает бит данных.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Бит данных сбрасывается. Выходное условие цепочки устанавливается на "ложь"
входное условие цепочки - "ложь"	Бит данных сбрасывается. Выходное условие цепочки устанавливается на "ложь"
входное условие цепочки - "истина"	Бита данных устанавливается. Выходное условие цепочки устанавливается на "истина".
постсканирование	Бит данных сбрасывается. Выходное условие цепочки устанавливается на "ложь".

Пример: Когда *switch* установлен, то инструкция OTE устанавливает (включает) *light_1*. Когда *switch* сброшен, то инструкция OTE сбрасывает (выключает) *light_1*.

Релейная логика

Структурированный текст

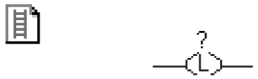
```
light_1  [:=]  switch;
```

Output Latch (OTL) (Фиксация выхода)

Инструкция OTL устанавливает (фиксирует) бит данных.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
бит данных	BOOL	тег	устанавливаемый бит

Структурированный текст

В структурированном тексте инструкция OTL отсутствует, но можно получить тот же результат с помощью конструкции IF...THEN и операции присваивания.

```
IF BOOL_expression THEN
     data_bit := 1;
END_IF;
```

Информацию о синтаксисе конструкций, выражений и операций присваивания в структурированном тексте можно найти в Приложении C.

Описание:

Когда инструкция OTL разрешена, она устанавливает бит данных. Этот бит данных остается установленным до тех пор, пока он не будет сброшен, обычно с помощью инструкции OTU. Когда инструкция OTL запрещена, она не меняет состояние бита данных.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

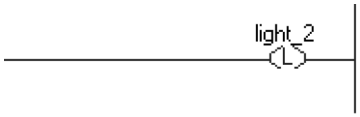
отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Бит данных не изменяется. Выходное условие цепочки устанавливается на "ложь"
входное условие цепочки - "ложь"	Бит данных не изменяется. Выходное условие цепочки устанавливается на "ложь"
входное условие цепочки - "истина"	Бита данных устанавливается. Выходное условие цепочки устанавливается на "истина".
постсканирование	Бит данных не изменяется. Выходное условие цепочки устанавливается на "ложь".

Пример: Когда инструкция OTL разрешена, она устанавливает light_2. Этот бит остается установленным пока он не будет сброшен, как правило с помощью инструкции OTU.

Релейная логика



Структурированный текст

```
IF  BOOL_expression  THEN
     light_2  :=  1;
END_IF;
```

Output Unlatch (OTU) (Расфиксации выхода)

Инструкция OTU сбрасывает бит данных (снимает фиксацию).

Операнды:

Релейная логика

Операнд:	Тип:	Формат:	Описание:
бит данных	BOOL	тег	сбрасываемый бит

Структурированный текст

В структурированном тексте инструкция OTU отсутствует, но можно получить тот же результат с помощью конструкции IF..THEN и операции присваивания.

```
IF BOOL_expression THEN
     data_bit := 0;
END_IF;
```

Информацию о синтаксисе конструкций, выражений и операций присваивания в структурированном тексте можно найти в Приложении C.

Описание:

Когда инструкция OTU разрешена, она сбрасывает бит данных. Когда инструкция OTU запрещена, она не меняет состояние бита данных.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Бит данных не изменяется. Выходное условие цепочки устанавливается на "ложь"
входное условие цепочки - "ложь"	Бит данных не изменяется. Выходное условие цепочки устанавливается на "ложь"
входное условие цепочки - "истина"	Бита данных сбрасывается. Выходное условие цепочки устанавливается на "истина".
постсканирование	Бит данных не изменяется. Выходное условие цепочки устанавливается на "ложь".

Пример: Когда инструкция OTL разрешена, она сбрасывает light_2.

Релейная логика

Структурированный текст

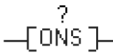
```
IF BOOL_expression THEN
     light_2 := 0;
END_IF;
```

One Shot (ONS) (Одно включение)

Инструкция ONS разрешает или запрещает оставшуюся часть цепочки в зависимости от состояния бита памяти.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
бит памяти	BOOL	тег	бит внутренней памяти сохраняет входное условие цепочки с последнего выполнения инструкции

Структурированный текст

В структурированном тексте инструкция ONS отсутствует, но можно получить тот же результат с помощью конструкции IF...THEN.

```
IF BOOL_expression AND NOT storage_bit THEN
     <statement>;
END_IF;
storage_bit := BOOL_expression;
```

Информацию о синтаксисе конструкций и выражений в структурированном тексте можно найти в Приложении C.

Описание:

Когда инструкция ONS разрешена и бит памяти сброшен, эта инструкция разрешает оставшуюся часть цепочки. Когда инструкция ONS запрещена или когда бит памяти установлен, эта инструкция запрещает оставшуюся часть цепочки.

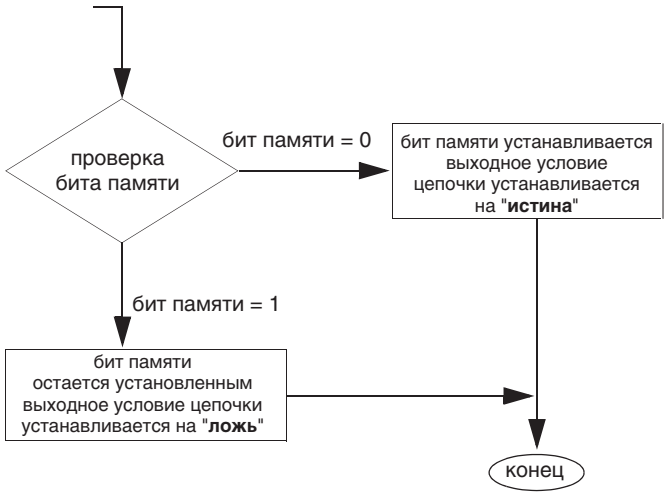
Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Бит памяти устанавливается, чтобы предотвратить неверное срабатывание во время первого сканирования Выходное условие цепочки устанавливается на "ложь"
входное условие цепочки - "ложь"	Бит памяти сбрасывается. Выходное условие цепочки устанавливается на "ложь"
входное условие цепочки - "истина"	
постсканирование	

Пример: Обычно вы предваряете инструкцию ONS инструкцией ввода, так как для правильной работы инструкции ONS вы сканируете ее и когда она разрешена, и когда она запрещена. Когда инструкция ONS разрешается, входное условие цепочки должно сброситься или бит памяти должен быть сброшен, чтобы инструкция ONS вновь разрешилась.

При всяком сканировании, когда сбрасывается *limit_switch_1* или устанавливается *storage_1*, эта цепочка не затрагивается. При всяком сканировании, когда *limit_switch_1* устанавливается, а *storage_1* сбрасывается, инструкция ONS устанавливает *storage_1*, а инструкция ADD увеличивает *sum* на 1. Все время, пока *limit_switch_1* остается установленным, значение *sum* не изменяется. Чтобы значение *sum* вновь увеличилось на единицу, *limit_switch_1* вновь должен перейти из сброшенного состояния в установленное.

Релейная логика

Структурированный текст

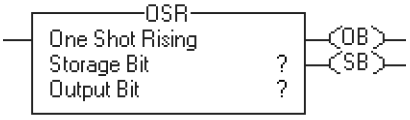
```
IF limit_switch_1 AND NOT storage_1 THEN
     sum := sum + 1;
END_IF;
storage_1 := limit_switch_1;
```

One Shot Rising (OSR) (Одно включение по переднему фронту)

Инструкция OSR устанавливает или сбрасывает выходной бит в зависимости от состояния бита памяти.

В структурированном тексте и функциональном блоке этой инструкции соответствует инструкция OSRI (см. стр. 1-19).

Операнды:

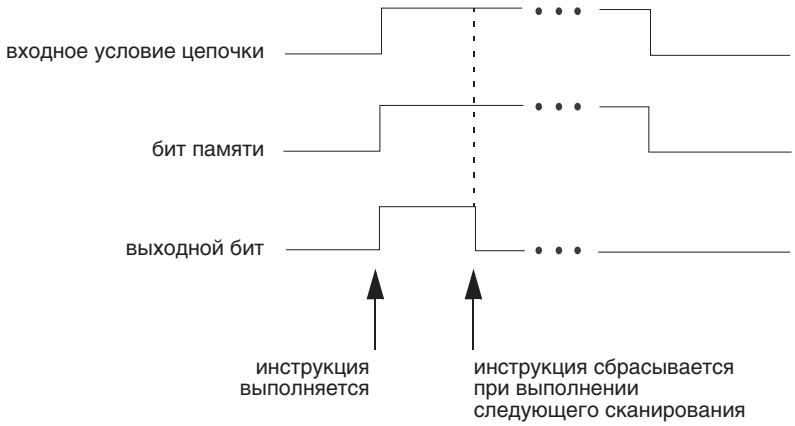


Релейная логика

Операнд:	Тип:	Формат:	Описание:
бит памяти	BOOL	тег	бит внутренней памяти сохраняет входное условие цепочки с последнего выполнения инструкции
выходной бит	BOOL	тег	устанавливаемый бит

Описание:

Когда инструкция OSR разрешена и бит памяти сброшен, эта инструкция устанавливает выходной бит. Когда инструкция OSR разрешена и бит памяти установлен, или когда эта инструкция запрещена, она сбрасывает выходной бит.



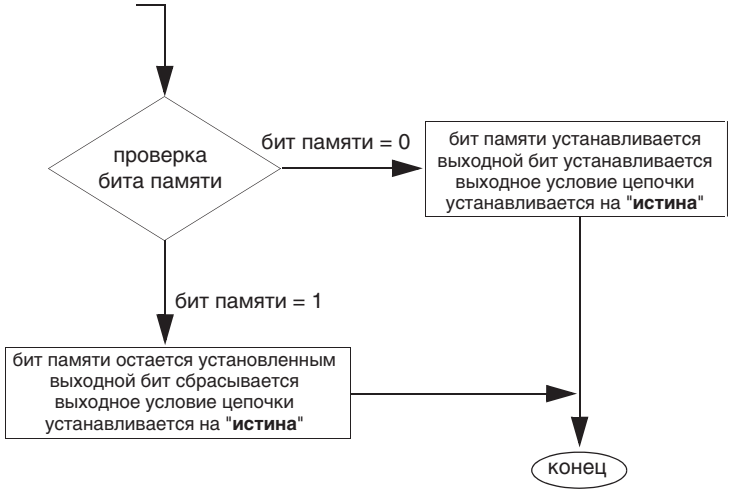
Арифметические флаги состояния:

не затрагиваются

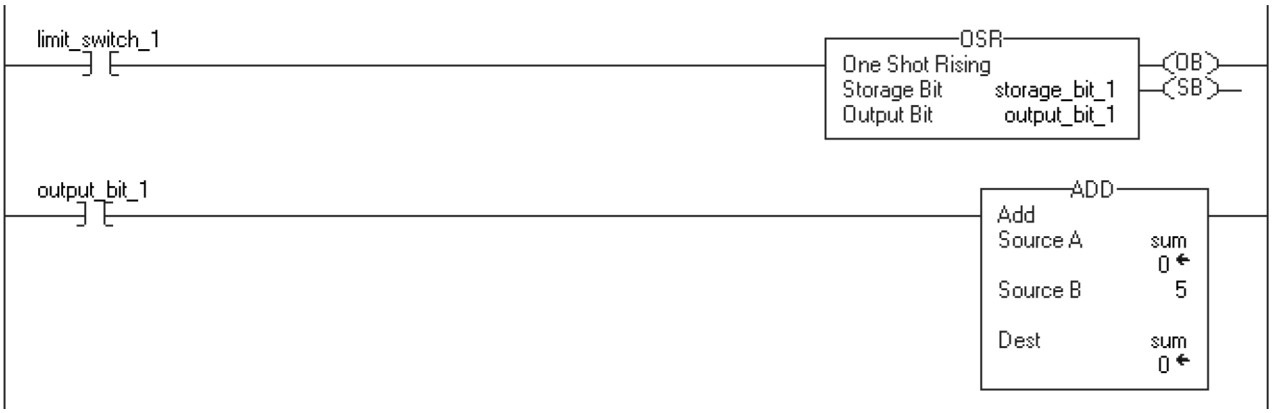
Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Бит памяти устанавливается, чтобы предотвратить неверное срабатывание во время первого сканирования. Выходной бит сбрасывается. Выходное условие цепочки устанавливается на "ложь"
входное условие цепочки - "ложь"	Бит памяти сбрасывается. Выходной бит не изменяется. Выходное условие цепочки устанавливается на "ложь"
входное условие цепочки - "истина"	
постсканирование	Бит памяти сбрасывается. Выходной бит не изменяется. Выходное условие цепочки устанавливается на "ложь"

Пример: Всякий раз когда *limit_switch_1* переходит из сброшенного состояния в установленное, инструкция OSR устанавливает *output_bit_1*, а инструкция ADD увеличивает *sum* на 5. Все время, пока *limit_switch_1* остается установленным, значение *sum* не изменяется. Чтобы значение *sum* вновь увеличилось, *limit_switch_1* вновь должен перейти из сброшенного состояния в установленное. Вы можете использовать *output_bit_1* в нескольких цепочках для запуска других операций.

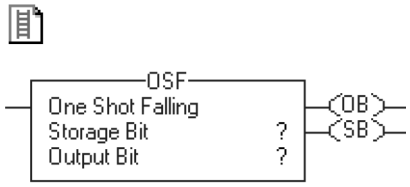


One Shot Falling (OSF) (Одно включение по заднему фронту)

Инструкция OSF устанавливает или сбрасывает выходной бит в зависимости от состояния бита памяти.

В структурированном тексте и функциональном блоке этой инструкции соответствует инструкция OSFI (см. стр. 1-22).

Операнды:

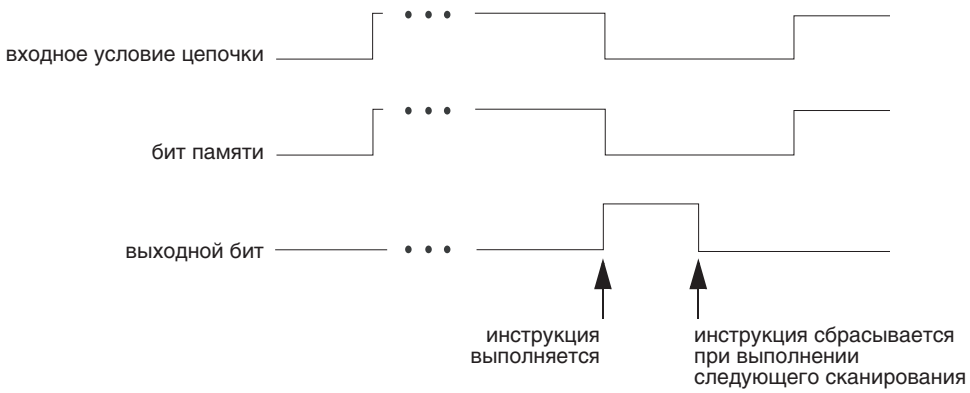


Операнды релейной логики

Операнд:	Тип:	Формат:	Описание:
бит памяти	BOOL	тег	бит внутренней памяти сохраняет входное условие цепочки с последнего выполнения инструкции
выходной бит	BOOL	тег	устанавливаемый бит

Описание:

Когда инструкция OSF запрещена и бит памяти установлен, эта инструкция устанавливает выходной бит. Когда инструкция OSF запрещена и бит памяти сброшен, или когда эта инструкция разрешена, она сбрасывает выходной бит.



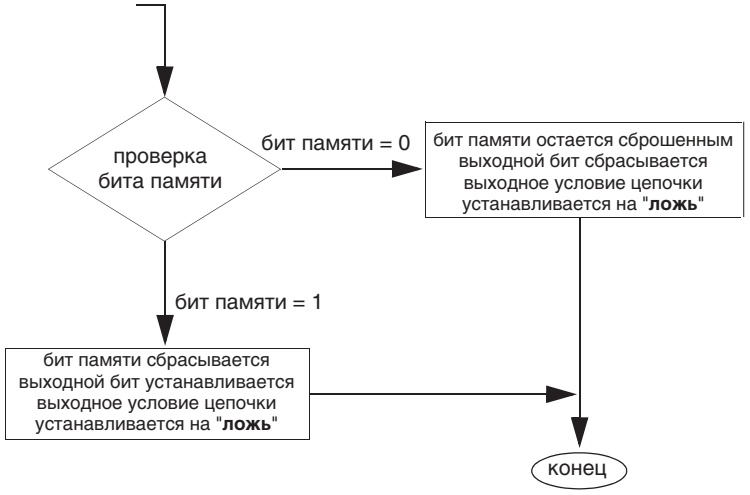
Арифметические флаги состояния:

не затрагиваются

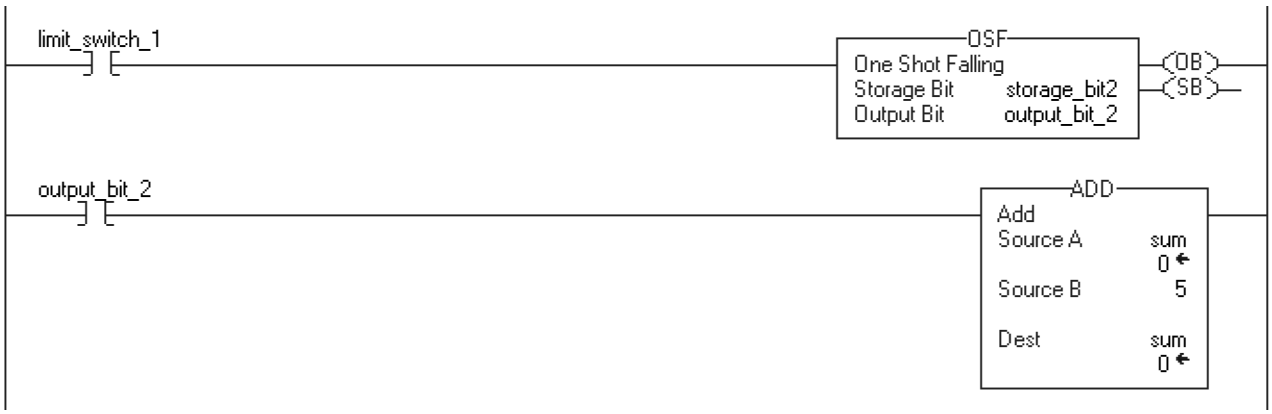
Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Бит памяти устанавливается, чтобы предотвратить неверное срабатывание во время первого сканирования. Выходной бит сбрасывается. Выходное условие цепочки устанавливается на "ложь"
входное условие цепочки - "ложь"	
входное условие цепочки - "истина"	Бит памяти устанавливается. Выходной бит сбрасывается. Выходное условие цепочки устанавливается на "истина"
постсканирование	См. действия для входного условия цепочки "ложь".

Пример: Всякий раз, когда *limit_switch_1* переходит из установленного состояния в сброшенное, инструкция OSF устанавливает *output_bit_2*, а инструкция ADD увеличивает *sum* на 5. Все время, пока *limit_switch_1* остается сброшенным, значение *sum* не изменяется. Чтобы значение *sum* вновь увеличилось, *limit_switch_1* вновь должен перейти из установленного состояния в сброшенное. Вы можете использовать *output_bit_2* в нескольких цепочках для запуска других операций.



One Shot Rising with Input (OSRI) (Одно включение по переднему фронту от входа)

Инструкция OSRI устанавливает выходной бит на один цикл выполнения при переключении входного бита со сброшенного состояния на установленное.

В релейной логике ей соответствует инструкция OSR (см. стр. 1-15).

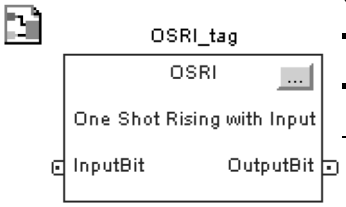
Операнды:

Структурированный текст

```
OSRI(OSRI_tag);
```

Операнд:	Тип:	Формат:	Описание:
тег OSRI	FBD_ONESHOT	структура	структура OSRI

Функциональный блок

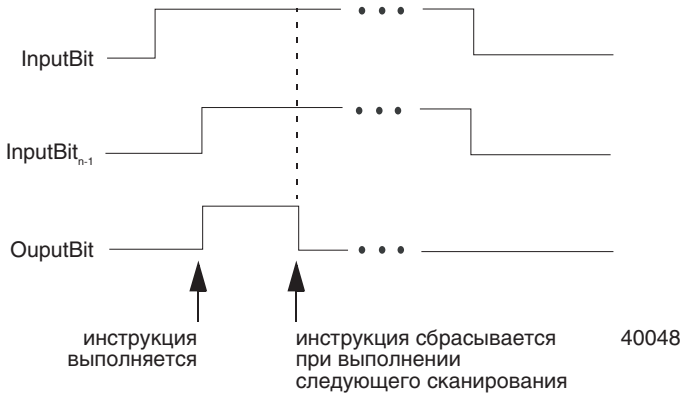


Операнд:	Тип:	Формат:	Описание:
тег OSRI	FBD_ONESHOT	структура	структура OSRI

Структура FBD_ONESHOT

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	**Функциональный блок:** Если этот параметр сброшен, то инструкция не выполняется, а выходные данные не обновляются. Если он установлен, то инструкция выполняется. По умолчанию параметр установлен. **Структурированный текст:** Не оказывает влияния. Инструкция выполняется.
InputBit	BOOL	Входной бит. Это эквивалентно условию цепочки для инструкции релейной логики OSR. По умолчанию сброшен.
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
OutputBit	BOOL	Выходной бит

Описание: Когда InputBit установлен, а InputBitn-1 сброшен, инструкция OSRI устанавливает OuputBit. Когда InputBitn-1 установлен или когда InputBit сброшен, инструкция OSRI сбрасывает OuputBit.



Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

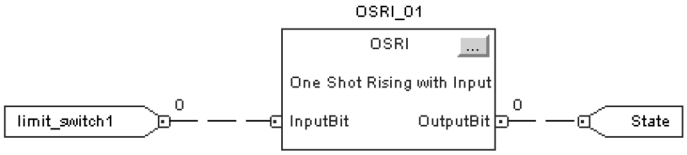
Условие:	Действие функционального блока:	Действие структурированного текста:
предварительное сканирование	Ничего не происходит.	Ничего не происходит.
первое сканирование инструкции	Устанавливается InputBitn-1	Устанавливается InputBitn-1
первое выполнение инструкции	Устанавливается InputBitn-1	Устанавливается InputBitn-1
EnableIn сброшен	EnableOut сбрасывается, инструкция ничего не выполняет, а выходные данные не обновляются.	неприменимо
EnableIn установлен	При переходе InputBit со сброшенного состояния в установленное инструкция устанавливает InputBitn-1. Инструкция выполняется. Устанавливается EnableOut.	При переходе InputBit со сброшенного состояния в установленное инструкция устанавливает InputBitn-1. EnableIn всегда установлен. Инструкция выполняется.
постсканирование	Ничего не происходит.	Ничего не происходит.

Пример: Когда *limit_switch_1* переходит из сброшенного состояния в установленное, инструкция OSRI устанавливает OutputBit на одно сканирование.

Структурированный текст

```
OSRI_01.InputBit := limit_switch1;

OSRI(OSRI_01);

State := OSRI_01.OutputBit;
```

Функциональный блок



One Shot Falling with Input (OSFI) (Одно включение по заднему фронту от входа)

Инструкция OSFI устанавливает выходной бит на один цикл выполнения при переключении входного бита с установленного состояния на сброшенное.

В релейной логике ей соответствует инструкция OSF (см. стр. 1-17).

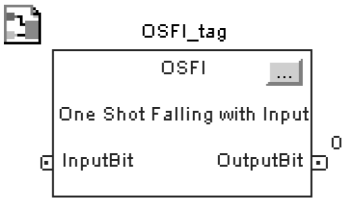
Операнды:

Структурированный текст

```
OSFI(OSFI_tag);
```

Операнд:	Тип:	Формат:	Описание:
тег OSFI	FBD_ONESHOT	структура	структура OSFI

Функциональный блок

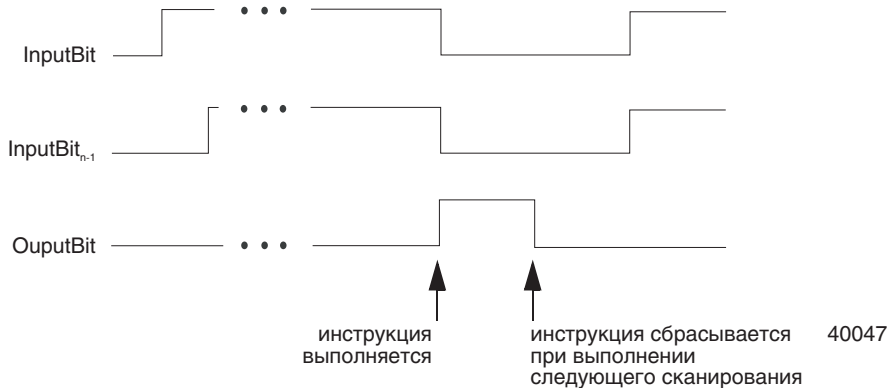


Операнд:	Тип:	Формат:	Описание:
тег OSFI	FBD_ONESHOT	структура	структура OSFI

Структура FBD_ONESHOT

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	**Функциональный блок:** Если этот параметр сброшен, то инструкция не выполняется, а выходные данные не обновляются. Если он установлен, то инструкция выполняется. По умолчанию параметр установлен. **Структурированный текст:** Не оказывает влияния. Инструкция выполняется.
InputBit	BOOL	Входной бит. Это эквивалентно условию цепочки для инструкции релейной логике OSF. По умолчанию сброшен.
Выходной параметр:	Тип данных:	Описание:
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
OutputBit	BOOL	Выходной бит

Описание: Когда InputBit сброшен, а InputBitn-1 установлен, инструкция OSRI устанавливает OuputBit. Когда InputBitn-1 сброшен или когда InputBit установлен, инструкция OSRI сбрасывает OuputBit.



Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Условие:	Действие функционального блока:	Действие структурированного текста:
предварительное сканирование	Ничего не происходит.	Ничего не происходит.
первое сканирование инструкции	Сбрасывается InputBitn-1	Сбрасывается InputBitn-1
первое выполнение инструкции	Сбрасывается InputBitn-1	Сбрасывается InputBitn-1
EnableIn сброшен	EnableOut сбрасывается, инструкция ничего не выполняет, а выходные данные не обновляются.	неприменимо
EnableIn установлен	При переходе InputBit со сброшенного состояния в установленное инструкция сбрасывает InputBitn-1. Инструкция выполняется. Устанавливается EnableOut.	При переходе InputBit со сброшенного состояния в установленное инструкция сбрасывает InputBitn-1. EnableIn всегда установлен. Инструкция выполняется.
постсканирование	Ничего не происходит.	Ничего не происходит.

Пример: Когда *limit_switch_1* переходит из установленного состояния в сброшенное, инструкция OSFI устанавливает OutputBit на одно сканирование.

Структурированный текст

```
OSFI_01.InputBit := limit_switch1;

OSFI(OSFI_01);

Output_state := OSFI_01.OutputBit;
```

Функциональный блок

Инструкции таймера и счетчика

(TON, TOF, RTO, TONR, TOFR, RTOR, CTU, CTD, CTUD, RES)

Введение Таймеры и счетчики управляют операциями на основе времени или количества событий.

Если вы хотите:	Используйте эту инструкцию:	Имеющуюся в языках:	См. стр.:
получить время, в течение которого таймер включен	TON	релейная логика	2-2
получить время, в течение которого таймер выключен	TOF	релейная логика	2-6
просуммировать время	RTO	релейная логика	2-10
получить время, в течение которого таймер включен со встроенным сбросом в функциональном блоке	TONR	структурированный текст функциональный блок	2-14
получить время, в течение которого таймер отключен со встроенным сбросом в функциональном блоке	TOFR	структурированный текст функциональный блок	2-17
просуммировать время со встроенным сбросом в функциональном блоке	RTOR	структурированный текст функциональный блок	2-20
выполнить прямой счет	CTU	релейная логика	2-24
выполнить обратный счет	CTD	релейная логика	2-28
выполнить прямой и обратный счет в функциональном блоке	CTUD	структурированный текст функциональный блок	2-32
обнулить таймер или счетчик	RES	релейная логика	2-36

Масштаб времени для всех таймеров составляет 1 мс.

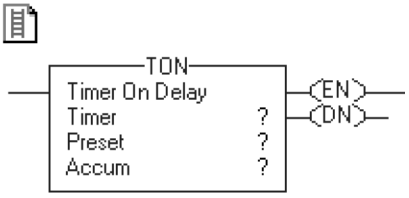
Timer On Delay (TON) (Таймер с выдержкой на включение)

Инструкция TON представляет собой таймер без сохранения, суммирующий время, в течение которого данная инструкция разрешена (входное условие цепочки "истина").

В структурированном тексте и функциональном блоке этой инструкции соответствует инструкция TONR (см. стр. 2-14).

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Timer	TIMER	тег	структура timer
Preset	DINT	непосредственный	длительность задержки (суммирования времени)
Accum	DINT	непосредственный	общее количество подсчитанных таймером миллисекунд начальным значением обычно является 0

Структура TIMER

Мнемоника:	Тип данных:	Описание:
.EN	BOOL	Бит разрешения указывает на то, что инструкция TON разрешена.
.TT	BOOL	Бит таймирования указывает на то, что выполняется операция отсчета времени.
.DN	BOOL	Бит выполнения устанавливается, когда ACC ⩾ .PRE.
.PRE	DINT	Заданное значение - это значение в миллисекундах, которого должно достичь накопленное значение, чтобы данная инструкция установила бит .DN.
.ACC	DINT	Накопленное значение соответствует количеству миллисекунд, прошедших после разрешения инструкции TON.

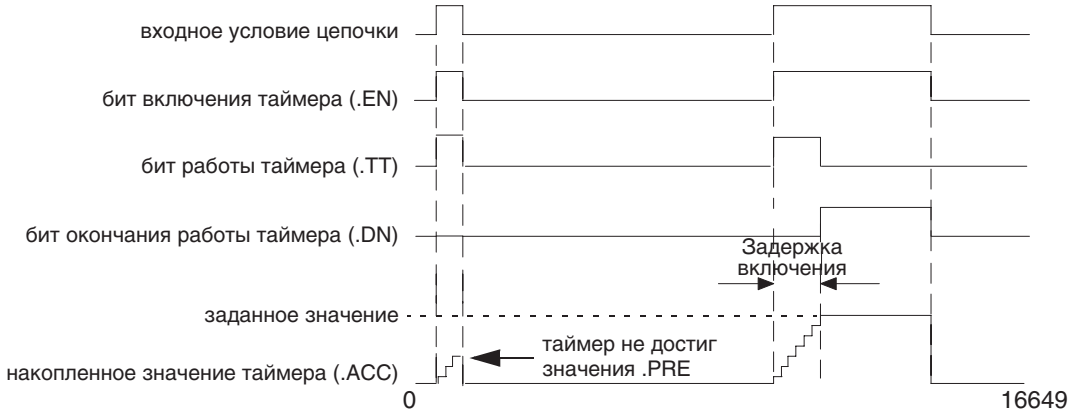
Описание:

Инструкция TON суммирует время до тех пор, пока:

- инструкция TON не будет запрещена
- не будет выполнено условие .ACC ⩾ .PRE

Временной масштаб всегда 1 мс. Например, для 2-секундного таймера в качестве значения .PRE введите 2000.

Когда инструкция TON запрещена, значение .ACC обнуляется.



Арифметические флаги состояния:

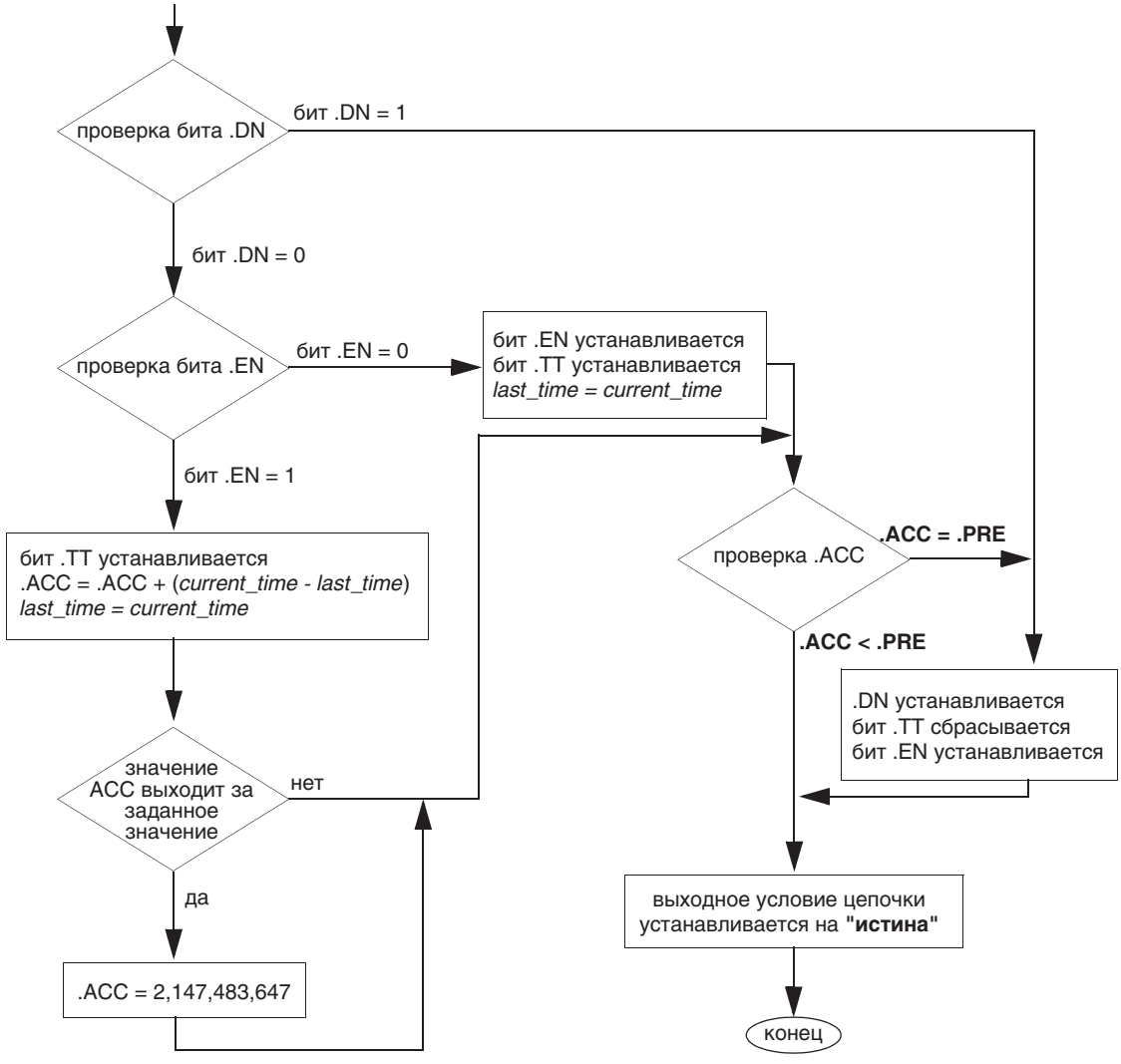
не затрагиваются

Условия ошибки:

Основная ошибка произойдет при:	Тип ошибки:	Код ошибки:
PRE < 0	4	34
.ACC < 0	4	34

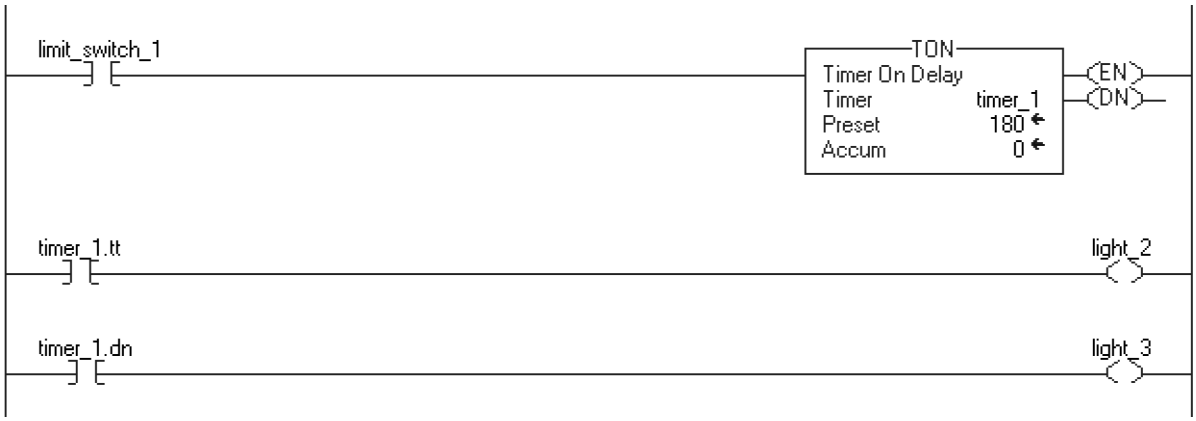
Выполнение:

Условие:	**Действие релейной логики:**
предварительное сканирование	Биты .EN, .TT и .DN сбрасываются. Значение .ACC обнуляется. Выходное условие цепочки устанавливается на "ложь".
входное условие цепочки "ложь"	Биты .EN, .TT и .DN сбрасываются. Значение .ACC обнуляется. Выходное условие цепочки устанавливается на "ложь".
входное условие цепочки "истина"	



постсканирование	Выходное условие цепочки устанавливается на "ложь"

Пример: Когда устанавливается *limit_switch_1*, *light_2* включается на 180 мс (*timer_1* ведет отсчет времени). Когда значение *timer_1.acc* достигает 180, *light_2* выключается, а *light_3* включается. *Light_3* остается включенным все время, пока инструкция TON запрещена. Если *limit_switch_1* сбрасывается в то время, когда *timer_1* ведет отсчет времени, *light_2* выключается.



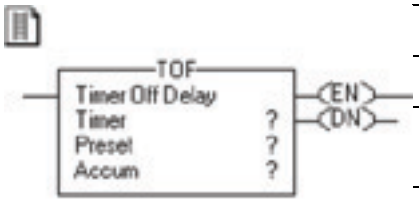
Timer Off Delay (TOF) (Таймер с выдержкой на отключение)

Инструкция TOF представляет собой таймер без сохранения, суммирующий время, в течение которого данная инструкция разрешена (входное условие цепочки "ложь").

В структурированном тексте и функциональном блоке этой инструкции соответствует инструкция TOFR (см. стр. 2-17).

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Timer	TIMER	тег	структура timer
Preset	DINT	непосредственный	длительность задержки (суммирования времени)
Accum	DINT	непосредственный	общее количество подсчитанных таймером миллисекунд начальным значением обычно является 0

Структура TIMER

Мнемоника:	Тип данных:	Описание:
.EN	BOOL	Бит разрешения указывает на то, что инструкция TOF разрешена.
.TT	BOOL	Бит таймирования указывает на то, что выполняется операция отсчета времени.
.DN	BOOL	Бит выполнения сбрасывается, когда .ACC ⩾.PRE.
.PRE	DINT	Заданное значение - это значение в миллисекундах, которого должно достичь накопленное значение, чтобы данная инструкция сбросила бит .DN.
.ACC	DINT	Накопленное значение соответствует количеству миллисекунд, прошедших после разрешения инструкции TOF.

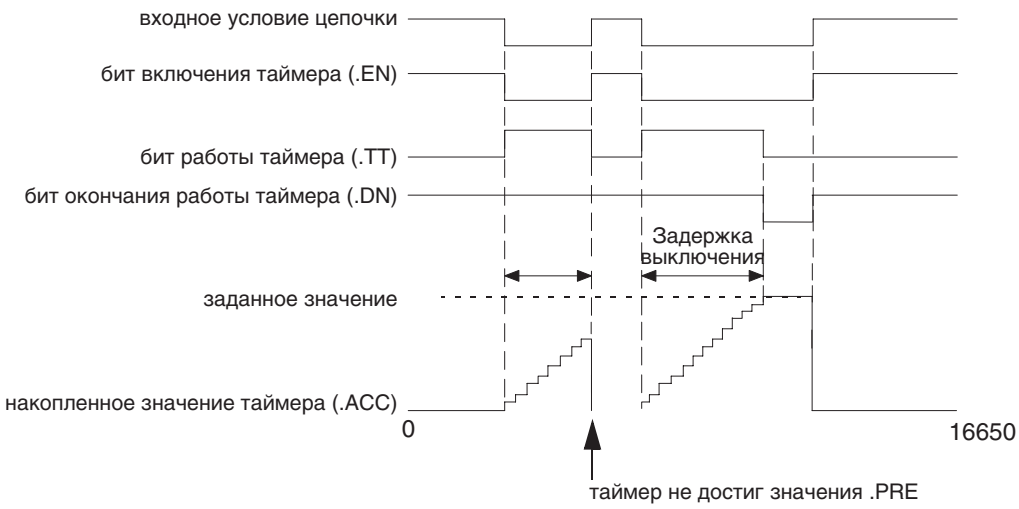
Описание:

Инструкция TOF суммирует время до тех пор, пока:

- инструкция TOF не будет запрещена
- не будет выполнено условие .ACC ⩾ .PRE

Временной масштаб всегда 1 мс. Например, для 2-секундного таймера в качестве значения .PRE введите 2000.

Когда инструкция TOF запрещена, значение .ACC обнуляется.



Арифметические флаги состояния:

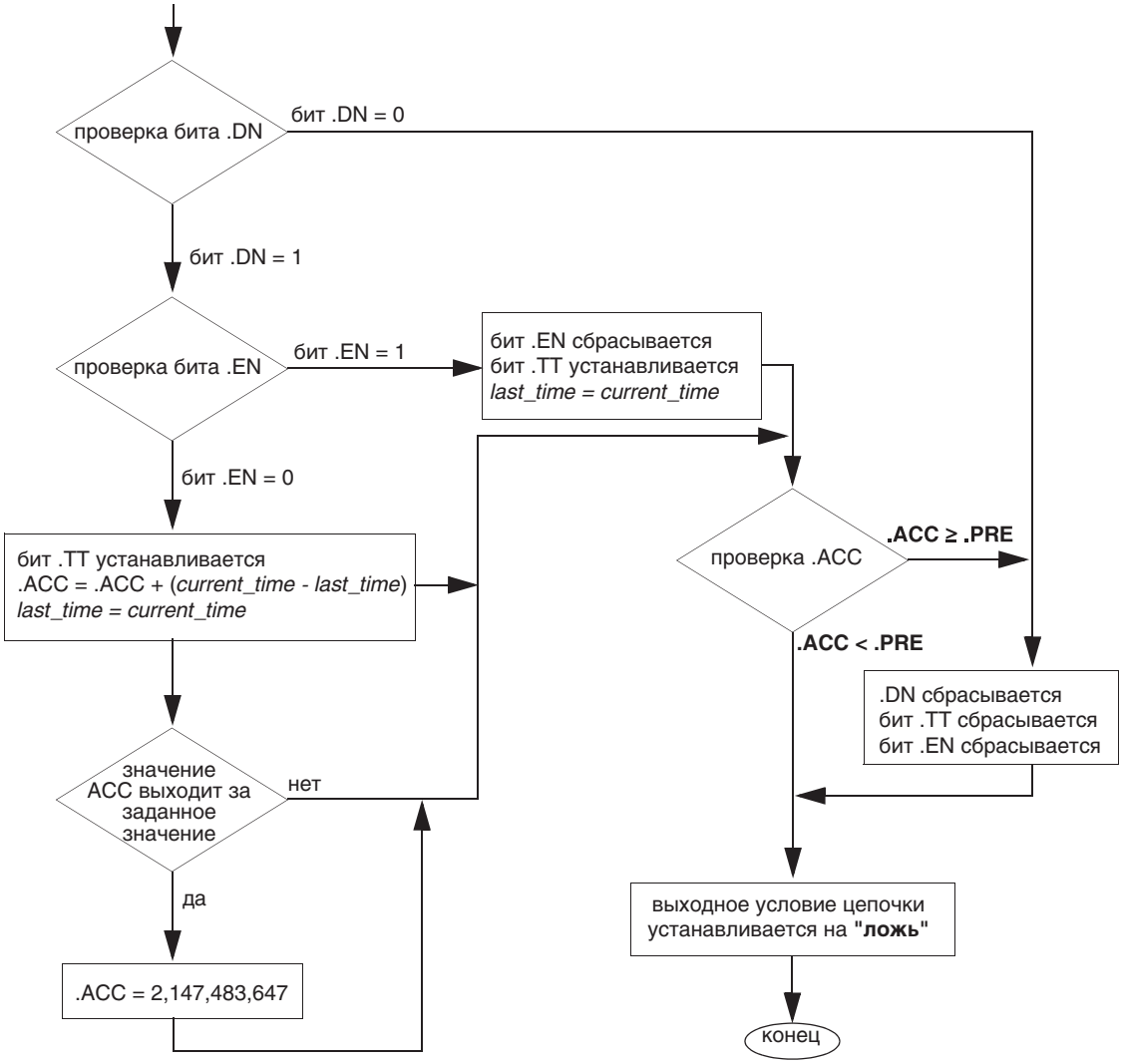
не затрагиваются

Условия ошибки:

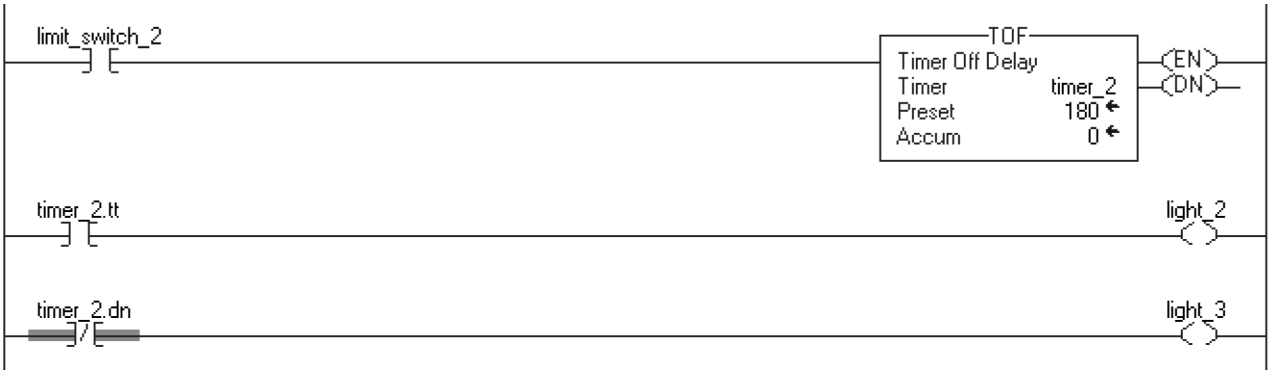
Основная ошибка произойдет при:	Тип ошибки:	Код ошибки:
.PRE < 0	4	34
.ACC < 0	4	34

Выполнение:

Условие:	**Действие релейной логики:**
предварительное сканирование	Биты .EN, .TT и .DN сбрасываются. Значение .ACC устанавливается равным значению .PRE. Выходное условие цепочки устанавливается на "ложь".
входное условие цепочки "ложь"	
входное условие цепочки "истина"	Биты .EN, .TT и .DN устанавливаются. Значение .ACC обнуляется. Выходное условие цепочки устанавливается на "истина"
постсканирование	Выходное условие цепочки устанавливается на "ложь"



Пример: Когда сбрасывается *limit_switch_2*, *light_2* включается на 180 мс (*timer_2* ведет отсчет времени). Когда значение *timer_2*.acc достигает 180, *light_2* выключается, а *light_3* включается. *Light_3* остается включенным до тех пор, пока инструкция TOF не будет разрешена. Если *limit_switch_2* устанавливается в то время, когда *timer_2* ведет отсчет времени, *light_2* выключается.

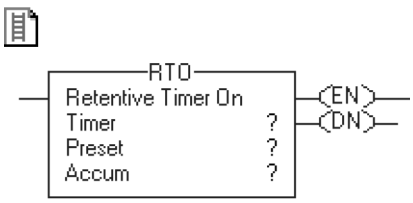


Retentive Timer On (RTO) (Таймер с сохранением времени включения)

Инструкция RTO представляет собой таймер с сохранением, суммирующий время, в течение которого данная инструкция разрешена.

В структурированном тексте и функциональном блоке этой инструкции соответствует инструкция RTOR (см. стр. 2-20).

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Описание:
Timer	TIMER	тег	структура timer
Preset	DINT	непосредственный	длительность задержки (суммирования времени)
Accum	DINT	непосредственный	общее количество подсчитанных таймером миллисекунд начальным значением обычно является 0

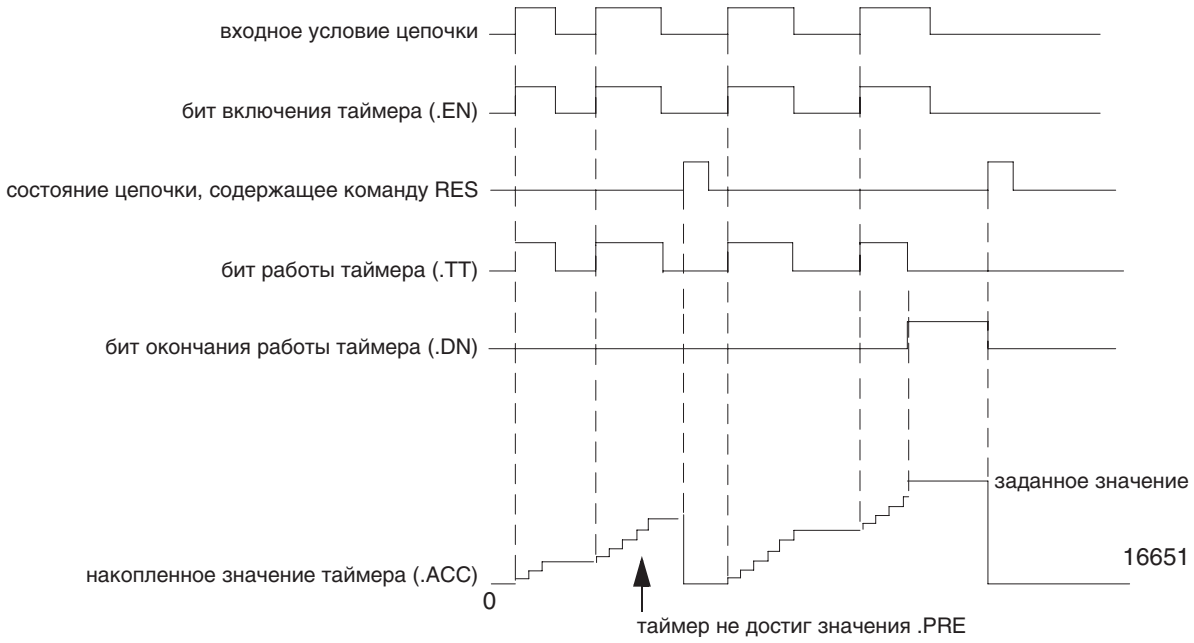
Структура TIMER

Мнемоника:	Тип данных:	Описание:
.EN	BOOL	Бит разрешения указывает на то, что инструкция RTO разрешена.
.TT	BOOL	Бит таймирования указывает на то, что выполняется операция отсчета времени.
.DN	BOOL	Бит выполнения указывает на то, что .ACC ⩾ .PRE.
.PRE	DINT	Заданное значение - это значение в миллисекундах, которого должно достичь накопленное значение, чтобы данная инструкция установила бит .DN.
.ACC	DINT	Накопленное значение соответствует количеству миллисекунд, прошедших после разрешения инструкции RTO.

Описание:

Инструкция RTO суммирует время до тех пор, пока она не будет запрещена. После запрещения инструкции RTO значение .ACC сохраняется. Вы должны обнулить значение .ACC. Обычно для этого используется инструкция RES со ссылкой на ту же самую структуру TIMER.

Временной масштаб всегда 1 мс. Например, для 2-секундного таймера в качестве значения .PRE введите 2000.



Арифметические флаги состояния:

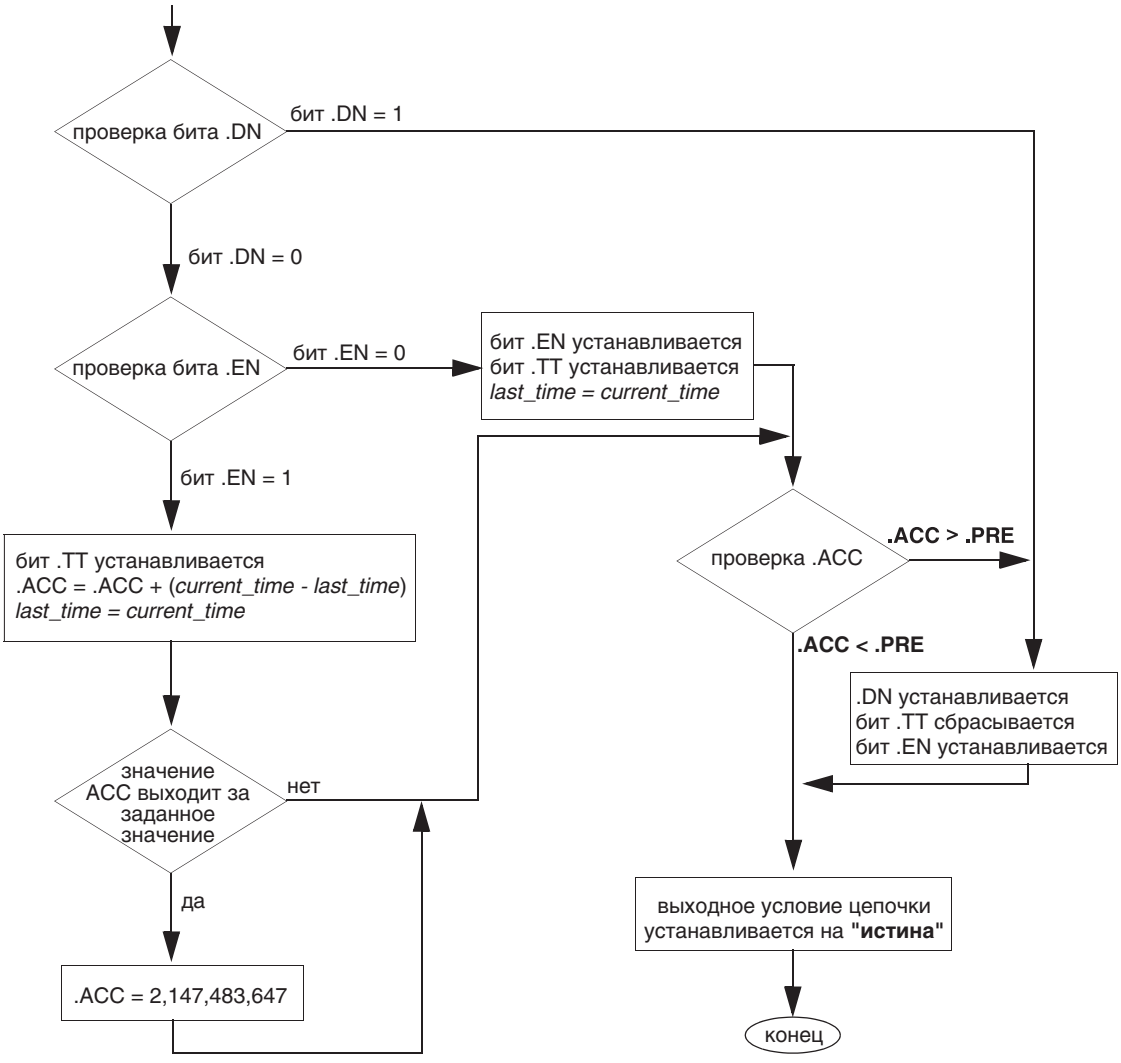
не затрагиваются

Условия ошибки:

Основная ошибка произойдет при:	Тип ошибки:	Код ошибки:
.PRE < 0	4	34
.ACC < 0	4	34

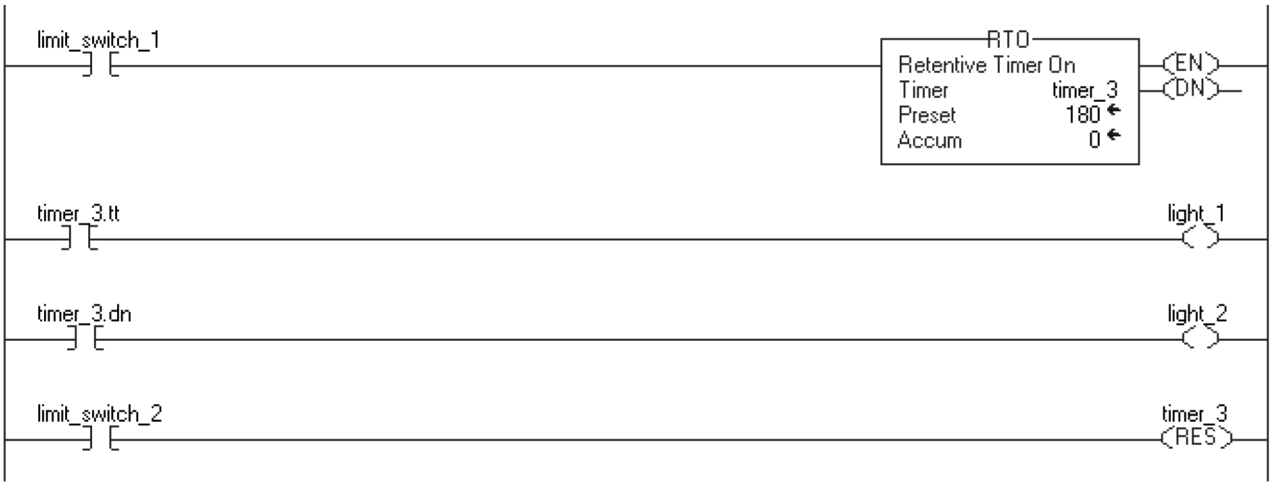
Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Биты .EN, .TT и .DN сбрасываются. Значение .ACC не изменяется. Выходное условие цепочки устанавливается на "ложь".
входное условие цепочки "ложь"	Биты .EN и .TT сбрасываются. Бит .DN не изменяется. Значение .ACC не изменяется. Выходное условие цепочки устанавливается на "ложь".
входное условие цепочки "истина"	



постсканирование	Выходное условие цепочки устанавливается на "ложь"

Пример: Когда устанавливается *limit_switch_1*, *light_1* включается на 180 мс (*timer_3* ведет отсчет времени). Когда значение *timer_3*.acc достигает 180, *light_1* выключается, а *light_2* включается. *Light_2* остается включенным до тех пор, пока не будет сброшен *timer_3*. Если *limit_switch_2* сбрасывается в то время, когда *timer_3* ведет отсчет времени, *light_1* остается включенным. Когда устанавливается *limit_switch_2*, инструкция RES сбрасывает *timer_3* (сбрасывает бит состояния и обнуляет значение .ACC).



Timer On Delay with Reset (TONR) (Таймер с выдержкой на включение со сбросом)

Инструкция TONR представляет собой таймер без сохранения, суммирующий время, когда установлен параметр TimerEnable.

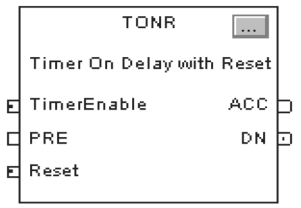
Эта инструкция имеется в релейной логике в виде двух отдельных инструкций: TON (см. стр. 2-2) и RES (см. стр. 2-36).

Операнды:

```
TONR(TONR_tag);
```

Структурированный текст

Переменная:	Тип:	Формат:	Описание:
тег TONR	FBD_TIMER	структура	структура TONR



Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег TONR	FBD_TIMER	структура	структура TONR

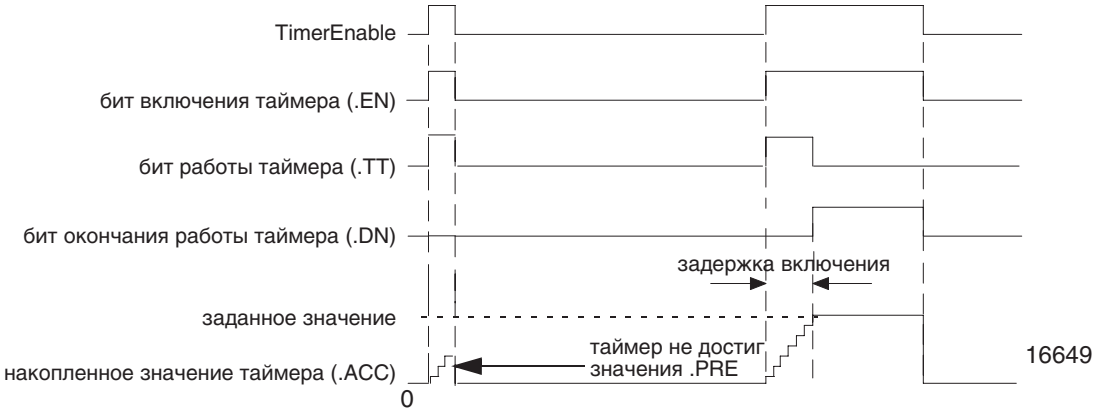
Структура FBD_TIMER

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	**Функциональный блок:** Если этот параметр сброшен, данная инструкция не выполняется, и выходы не обновляются. Если он установлен, инструкция выполняется. По умолчанию параметр установлен. **Структурированный текст:** Никак не влияет. Инструкция выполняется.
TimerEnable	BOOL	Если этот параметр установлен, таймер запускается и суммирует время. По умолчанию параметр сброшен.
PRE	DINT	Заданное значение для таймера. Это значение в миллисекундах, которого должен достичь параметр ACC, чтобы отсчет времени завершился. Если это значение является недопустимым, инструкция устанавливает соответствующий бит в параметре Status, и таймер не запускается. Допустимое значение находится в диапазоне от 0 до максимального положительного целого числа.
Reset	BOOL	Запрос на сброс таймера. Когда этот параметр установлен, таймер сбрасывается. По умолчанию параметр сброшен.
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	В результате выполнения инструкции получен допустимый результат.
ACC	BOOL	Суммарное время в миллисекундах.
EN	BOOL	Разрешение таймера. Указывает на то, что инструкция таймера разрешена.
TT	BOOL	Работа таймера. Когда этот параметр установлен, таймер выполняет отсчет времени.
DN	BOOL	Завершение работы таймера. Указывает на то, что суммарное время превысило заданное значение или сравнялось с ним.
Status	DINT	Состояние функционального блока.
InstructFault (Status.0)	BOOL	Инструкция обнаружила одну из следующих ошибок выполнения. Это не является основной или неосновной ошибкой контроллера. Чтобы понять, что именно произошло, проверьте остальные биты состояния.
PresetInv (Status.1)	BOOL	Заданное значение является недопустимым.

Описание: Инструкция TONR суммирует время до тех пор, пока:

- инструкция TONR не будет запрещена
- не будет выполнено условие ACC ⩾ PRE

Временной масштаб всегда 1 мс. Например, для 2-секундного таймера в качестве значения PRE введите 2000.



Для сброса данной инструкции установите входной параметр Reset. Если параметр TimerEnable устанавлен при установленном параметре Reset, инструкция TONR вновь начнет выполнять отсчет времени после сброса параметра Reset.

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

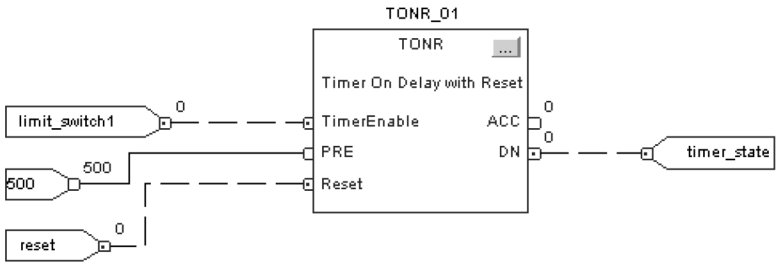
Условие:	Действие функционального блока:	Действие структурированного текста:
предварительное сканирование	Ничего не происходит.	Ничего не происходит.
первое сканирование инструкции	EN, TT и DN сбрасываются. Значение ACC обнуляется.	EN, TT и DN сбрасываются. Значение ACC обнуляется.
первое выполнение инструкции	EN, TT и DN сбрасываются. Значение ACC обнуляется.	EN, TT и DN сбрасываются. Значение ACC обнуляется.
EnableIn сбрасывается	EnableOut сбрасывается, инструкция ничего не выполняет, а выходы не обновляются.	не применимо
EnableIn устанавливается	Когда EnableIn переходит из сброшенного состояния в установленное, инструкция инициализируется, как описано для первого сканирования инструкции. Инструкция выполняется. EnableOut устанавливается.	EnableIn всегда установлен. Инструкция выполняется.
сброс	Когда устанавливается входной параметр Reset, инструкция сбрасывает EN, TT и DN и обнуляет ACC.	Когда устанавливается входной параметр Reset, инструкция сбрасывает EN, TT и DN и обнуляет ACC.
постсканирование	Ничего не происходит.	Ничего не происходит.

Пример: При каждом сканировании, когда установлен *limit_switch1*, инструкция TONR увеличивает значение ACC на величину истекшего времени до тех пор, пока значение ACC не достигнет значения PRE. При выполнении условия ACC ⩾ PRE устанавливается параметр DN, а также устанавливается *timer_state*.

Структурированный текст

```
TONR_01.Preset := 500;

TONR_01.Reset : = reset;

TONR_O1.TimerEnable := limit_switch1;


TONR(TONR_01);


timer_state := TONR_01.DN;
```

Пример функционального блока



Timer Off Delay with Reset (TOFR) (Таймер с выдержкой на отключение со сбросом)

Инструкция TOFR представляет собой таймер без сохранения, суммирующий время, когда сброшен параметр TimerEnable.

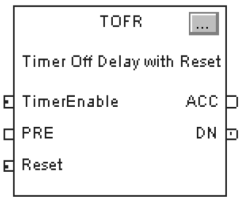
Эта инструкция имеется в релейной логике в виде двух отдельных инструкций: TOF (см. стр. 2-6) и RES (см. стр. 2-36).

Операнды:

```
TOFR(TOFR_tag);
```

Структурированный текст

Переменная:	Тип:	Формат:	Описание:
тег TOFR	FBD_TIMER	структура	структура TOFR



Операнды функционального блока

Операнд:	Тип:	Формат:	Описание:
тег TOFR	FBD_TIMER	структура	структура TOFR

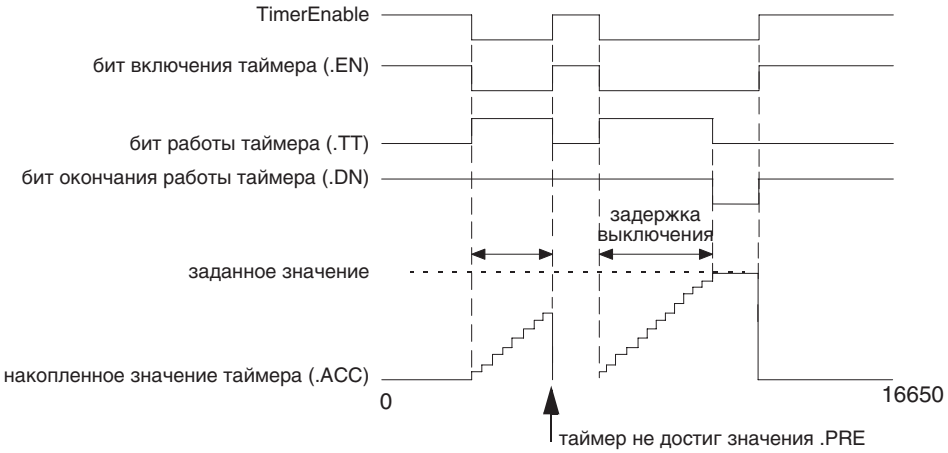
Структура FBD_TIMER

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	**Функциональный блок:** Если этот параметр сброшен, данная инструкция не выполняется, и выходы не обновляются. Если он установлен, инструкция выполняется. По умолчанию параметр установлен. **Структурированный текст:** Никак не влияет. Инструкция выполняется.
TimerEnable	BOOL	Если этот параметр установлен, таймер запускается и суммирует время. По умолчанию параметр сброшен.
PRE	DINT	Заданное значение для таймера. Это значение в миллисекундах, которого должен достичь параметр ACC, чтобы отсчет времени завершился. Если это значение является недопустимым, инструкция устанавливает соответствующий бит в параметре Status, и таймер не запускается. Допустимое значение находится в диапазоне от 0 до максимального положительного целого числа.
Reset	BOOL	Запрос на сброс таймера. Когда этот параметр установлен, таймер сбрасывается. По умолчанию параметр сброшен.
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	В результате выполнения инструкции получен допустимый результат.
ACC	BOOL	Суммарное время в миллисекундах.
EN	BOOL	Разрешение таймера. Указывает на то, что инструкция таймера разрешена.
TT	BOOL	Работа таймера. Когда этот параметр установлен, таймер выполняет отсчет времени.
DN	BOOL	Завершение работы таймера. Указывает на то, что суммарное время превысило заданное значение или сравнялось с ним.
Status	DINT	Состояние функционального блока.
InstructFault (Status.0)	BOOL	Инструкция обнаружила одну из следующих ошибок выполнения. Это не является основной или неосновной ошибкой контроллера. Чтобы понять, что именно произошло, проверьте остальные биты состояния.
PresetInv (Status.1)	BOOL	Заданное значение является недопустимым.

Описание: Инструкция TOFR суммирует время до тех пор, пока:

- инструкция TOFR не будет запрещена
- не будет выполнено условие ACC ⩾ PRE

Временной масштаб всегда 1 мс. Например, для 2-секундного таймера в качестве значения PRE введите 2000.



Для сброса данной инструкции установите входной параметр Reset. Если параметр TimerEnable сброшен при установленном параметре Reset, инструкция TOFR вновь начнет выполнять отсчет времени после сброса параметра Reset.

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

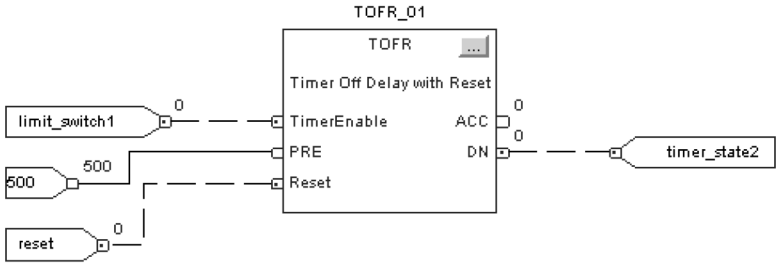
Условие:	Действие функционального блока:	Действие структурированного текста:
предварительное сканирование	Ничего не происходит.	Ничего не происходит.
первое сканирование инструкции	EN, TT и DN сбрасываются. Значение ACC устанавливается на PRE.	EN, TT и DN сбрасываются. Значение ACC устанавливается на PRE.
первое выполнение инструкции	EN, TT и DN сбрасываются. Значение ACC устанавливается на PRE.	EN, TT и DN сбрасываются. Значение ACC устанавливается на PRE.
EnableIn сбрасывается	EnableOut сбрасывается, инструкция ничего не выполняет, а выходы не обновляются.	не применимо
EnableIn устанавливается	Когда EnableIn переходит из сброшенного состояния в установленное, инструкция инициализируется, как описано для первого сканирования инструкции. Инструкция выполняется. EnableOut устанавливается.	EnableIn всегда установлен. Инструкция выполняется.
сброс	Когда устанавливается входной параметр Reset, инструкция сбрасывает EN, TT и DN и устанавливает ACC = PRE. Обратите внимание на отличие от использования инструкции RES в случае инструкции TOF.	Когда устанавливается входной параметр Reset, инструкция сбрасывает EN, TT и DN и устанавливает ACC = PRE. Обратите внимание на отличие от использования инструкции RES в случае инструкции TOF.
постсканирование	Ничего не происходит.	Ничего не происходит.

Пример: При каждом сканировании, когда установлен *limit_switch1*, инструкция TOFR увеличивает значение ACC на величину истекшего времени до тех пор, пока значение ACC не достигнет значения PRE. При выполнении условия ACC і PRE сбрасывается параметр DN, а также устанавливается *timer_state2*.

Структурированный текст

```
TOFR_01.Preset := 500

TOFR_01.Reset := reset;

TOFR_O1.TimerEnable := limit_switch1;


TOFR(TOFR_01);


timer_state2 := TOFR_01.DN;
```

Функциональный блок



Retentive Timer On with Reset (RTOR) (Таймер с сохранением времени включения со сбросом)

Инструкция RTOR представляет собой таймер с сохранением, суммирующий время, когда установлен параметр TimerEnable.

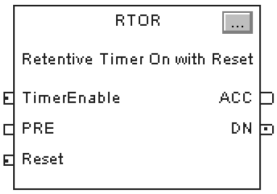
Эта инструкция имеется в релейной логике в виде двух отдельных инструкций: RTO (см. стр. 2-10) и RES (см. стр. 2-36).

Операнды:

```
RTOR(RTOR_tag);
```

Структурированный текст

Переменная:	Тип:	Формат:	Описание:
тег RTOR	FBD_TIMER	структура	структура RTOR



Функциональный блок

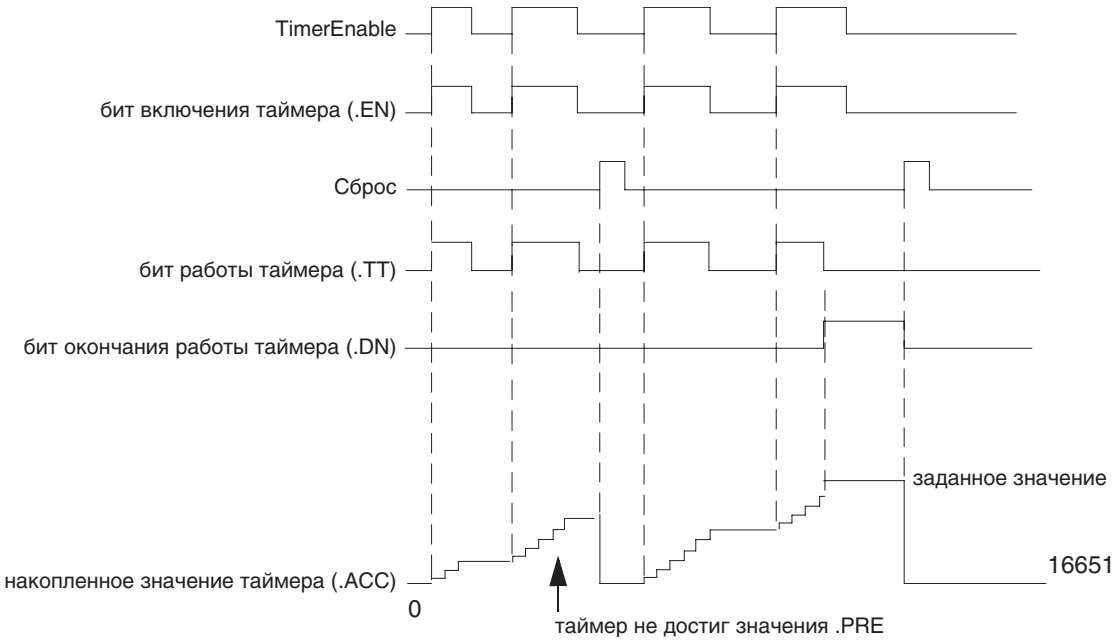
Операнд:	Тип:	Формат:	Описание:
тег RTOR	FBD_TIMER	структура	структура RTOR

Структура

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	**Функциональный блок:** Если этот параметр сброшен, данная инструкция не выполняется, и выходы не обновляются. Если он установлен, инструкция выполняется. По умолчанию параметр установлен. **Структурированный текст:** Никак не влияет. Инструкция выполняется.
TimerEnable	BOOL	Если этот параметр установлен, таймер запускается и суммирует время. По умолчанию параметр сброшен.
PRE	DINT	Заданное значение для таймера. Это значение в миллисекундах, которого должен достичь параметр ACC, чтобы отсчет времени завершился. Если это значение является недопустимым, инструкция устанавливает соответствующий бит в параметре Status, и таймер не запускается. Допустимое значение находится в диапазоне от 0 до максимального положительного целого числа.
Reset	BOOL	Запрос на сброс таймера. Когда этот параметр установлен, таймер сбрасывается.
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	В результате выполнения инструкции получен допустимый результат.
ACC	BOOL	Суммарное время в миллисекундах. Это значение сохраняется даже при сброшенном входном параметре TimerEnable. Этим поведение данного блока отличается от блока TONR.
EN	BOOL	Разрешение таймера. Указывает на то, что инструкция таймера разрешена.
TT	BOOL	Работа таймера. Когда этот параметр установлен, таймер выполняет отсчет времени.
DN	BOOL	Завершение работы таймера. Указывает на то, что суммарное время превысило заданное значение или сравнялось с ним.
Status	DINT	Состояние функционального блока.
InstructFault (Status.0)	BOOL	Инструкция обнаружила одну из следующих ошибок выполнения. Это не является основной или неосновной ошибкой контроллера. Чтобы понять, что именно произошло, проверьте остальные биты состояния.
PresetInv (Status.1)	BOOL	Заданное значение является недопустимым.

Описание: Инструкция RTOR суммирует время до тех пор, пока она не будет запрещена. При запрещении инструкции RTOR она сохраняет значение ACC. Вы должны обнулить значение ACC с помощью входного параметра Reset.

Временной масштаб всегда 1 мс. Например, для 2-секундного таймера в качестве значения PRE введите 2000.



Для сброса данной инструкции установите входной параметр Reset. Если параметр TimerEnable установлен при установленном параметре Reset, инструкция RTNR вновь начнет выполнять отсчет времени после сброса параметра Reset.

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

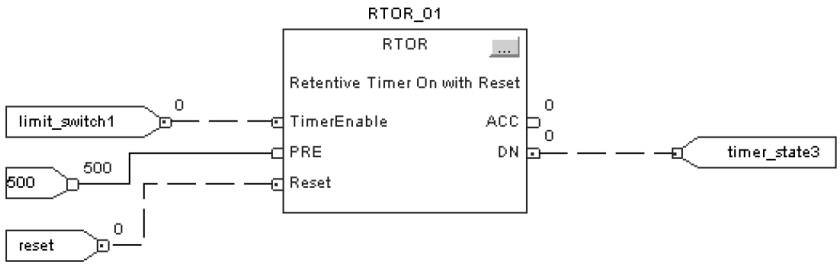
Условие:	Действие функционального блока:	Действие структурированного текста:
предварительное сканирование	Ничего не происходит.	Ничего не происходит.
первое сканирование инструкции	EN, TT и DN сбрасываются. Значение ACC не изменяется.	EN, TT и DN сбрасываются. Значение ACC не изменяется.
первое выполнение инструкции	EN, TT и DN сбрасываются. Значение ACC не изменяется.	EN, TT и DN сбрасываются. Значение ACC не изменяется.
EnableIn сбрасывается	EnableOut сбрасывается, инструкция ничего не выполняет, а выходы не обновляются.	не применимо
EnableIn устанавливается	**Функциональный блок:** Когда EnableIn переходит из сброшенного состояния в установленное, инструкция инициализируется, как описано для первого сканирования инструкции. Инструкция выполняется. EnableOut устанавливается.	EnableIn всегда установлен. Инструкция выполняется.
сброс	Когда устанавливается входной параметр Reset, инструкция сбрасывает EN, TT и DN и обнуляет ACC.	Когда устанавливается входной параметр Reset, инструкция сбрасывает EN, TT и DN и обнуляет ACC.
постсканирование	Ничего не происходит.	Ничего не происходит.

Пример: При каждом сканировании, когда установлен *limit_switch1*, инструкция RTOR увеличивает значение ACC на величину истекшего времени до тех пор, пока значение ACC не достигнет значения PRE. При выполнении условия ACC ⩾ PRE устанавливается параметр DN, а также устанавливается *timer_state3*.

Структурированный текст

```
RTOR_01.Preset := 500

RTOR_01.Reset := reset;

RTOR_O1.TimerEnable := limit_switch1;


RTOR(RTOR_01);


timer_state3 := RTOR_01.DN;
```

Функциональный блок



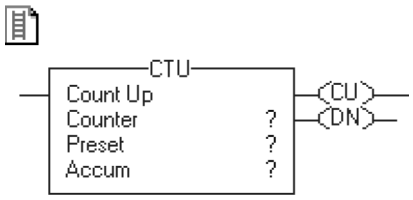
Count Up (CTU) (Прямой счет)

Инструкция CTU выполняет прямой счет.

В структурированном тексте и функциональном блоке этой инструкции соответствует инструкция CTUD (см. стр. 2-32).

Операнды:

Релейная логика



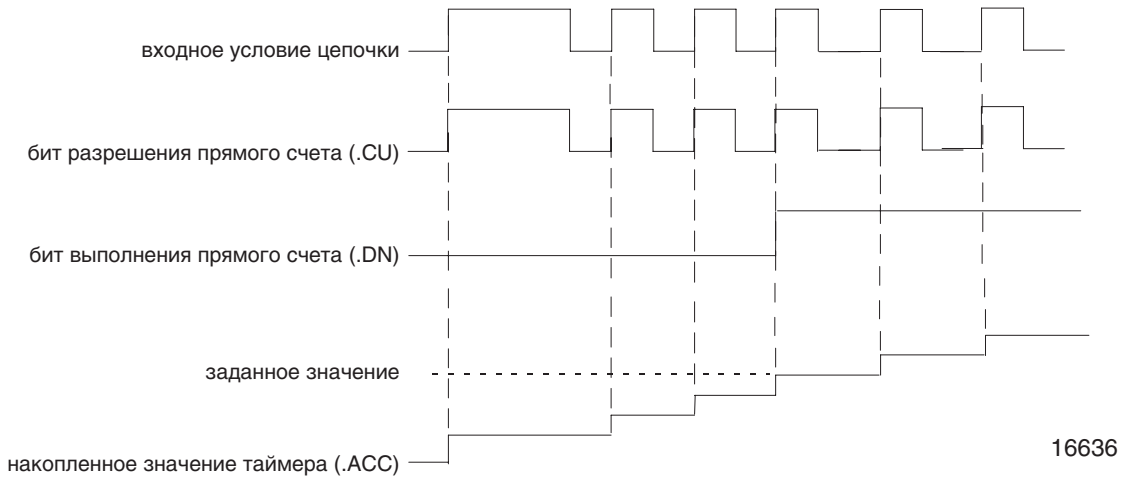
Операнд:	Тип:	Формат:	Описание:
Counter	COUNTER	тег	структура COUNTER
Preset	DINT	непосредственный	до какого верхнего значения выполнять счет
Accum	DINT	непосредственный	количество сделанных счетчиком отсчетов исходное значение обычно равно 0

Структура COUNTER

Мнемоника:	Тип данных:	Описание:
.CU	BOOL	Бит разрешения прямого счета указывает на то, что инструкция CTU разрешена.
.DN	BOOL	Бит выполнения указывает на выполнение условия .ACC і .PRE.
.OV	BOOL	Бит переполнения указывает на то, что счетчик вышел за верхнее предельное значение 2147483647. При этом счетчик переходит на -2147483648 и вновь начинает выполнять прямой счет.
.UN	BOOL	Бит отрицательного переполнения указывает на то, что счетчик вышел за нижнее предельное значение -2147483648. При этом счетчик переходит на 2147483647 и вновь начинает выполнять обратный счет.
.PRE	DINT	При достижении накопленным значением этого заданного значения инструкция устанавливает бит .DN.
.ACC	DINT	Накопленное значение соответствует количеству подсчитанных данной инструкцией переходов.

Описание: Когда инструкция CTU разрешена, а бит .CU сброшен, эта инструкция увеличивает значение счетчика на единицу. Когда инструкция разрешена, а бит .CU установлен, или когда инструкция запрещена, она сохраняет значение .ACC.

Накопленное значение продолжает увеличиваться даже после установки бита .DN. Для сброса накопленного значения используйте инструкцию RES со ссылкой на структуру COUNTER или запишите 0 в накопленное значение.

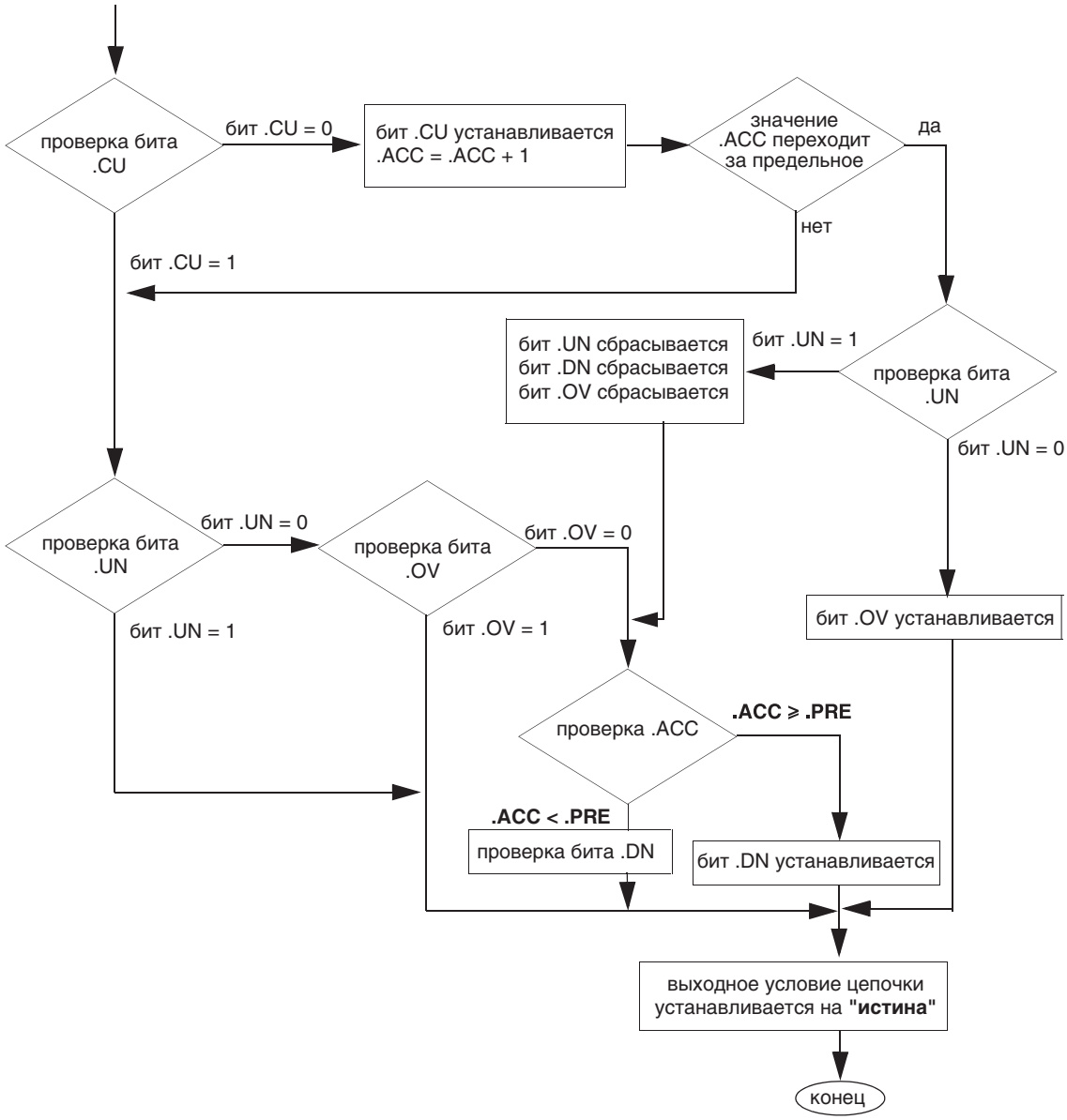


Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

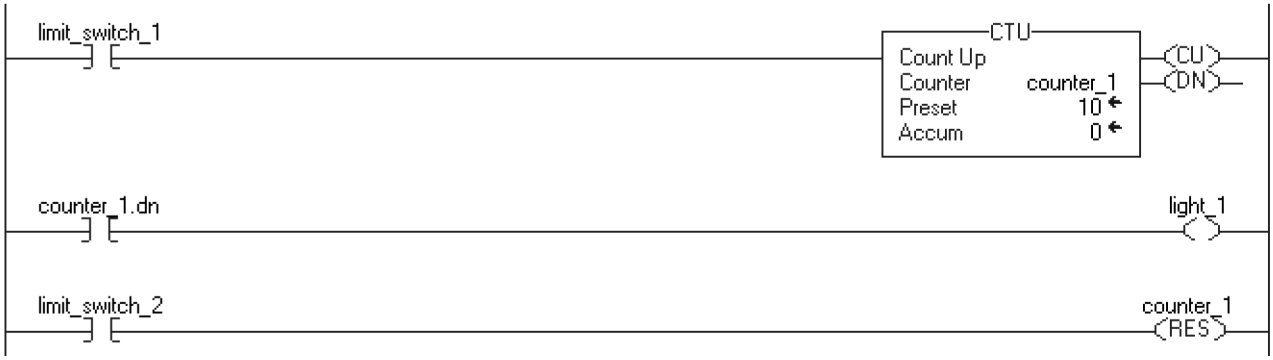
Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Бит .CU устанавливается, чтобы предотвратить неверное приращение значения во время первого сканирования программы. Выходное условие цепочки устанавливается на "ложь".
входное условие цепочки "ложь"	Бит .CU сбрасывается. Выходное условие цепочки устанавливается на "ложь".
входное условие цепочки "истина"	



постсканирование	Выходное условие цепочки устанавливается на "ложь".

Пример: После 10 переходов *limit_switch_1* из отключенного состояния во включенное устанавливается бит .DN и включается *light_1*. Если *limit_switch_1* продолжает переходить из отключенного состояния во включенное, *counter_1* продолжает увеличивать счет, а бит .DN остается установленным. Когда включается *limit_switch_2*, инструкция RES сбрасывает *counter_1* (сбрасывает биты состояния и значение .ACC), а *light_1* выключается.



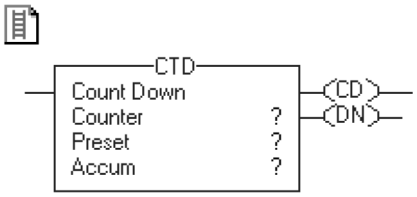
Count Down (CTD) (Обратный счет)

Инструкция CTD выполняет обратный счет.

В структурированном тексте и функциональном блоке этой инструкции соответствует инструкция CTUD (см. стр. 2-32).

Операнды:

Релейная логика



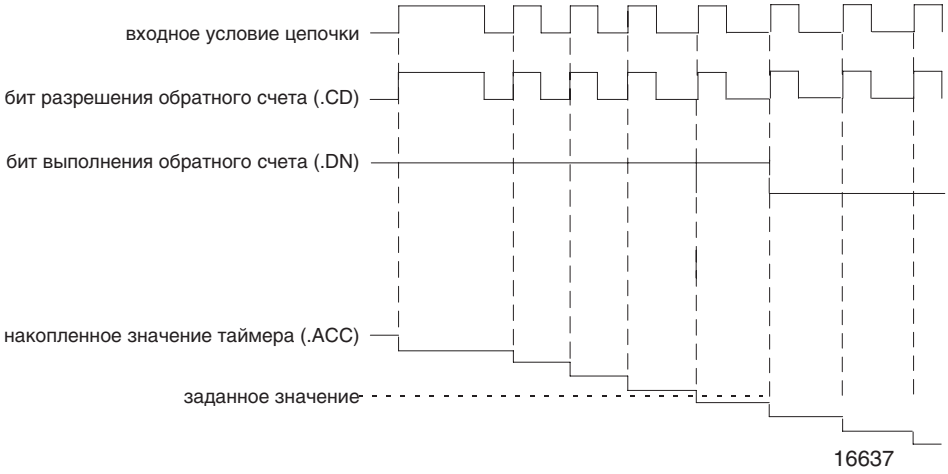
Операнд:	Тип:	Формат:	Описание:
Counter	COUNTER	тег	структура COUNTER
Preset	DINT	непосредственный	до какого нижнего значения выполнять счет
Accum	DINT	непосредственный	количество сделанных счетчиком отсчетов исходное значение обычно равно 0

Структура COUNTER

Мнемоника:	Тип данных:	Описание:
.CD	BOOL	Бит разрешения обратного счета указывает на то, что инструкция CTD разрешена.
.DN	BOOL	Бит выполнения указывает на выполнение условия .ACC i .PRE.
.OV	BOOL	Бит переполнения указывает на то, что счетчик вышел за верхнее предельное значение 2147483647. При этом счетчик переходит на -2147483648 и вновь начинает выполнять прямой счет.
.UN	BOOL	Бит отрицательного переполнения указывает на то, что счетчик вышел за нижнее предельное значение -2147483648. При этом счетчик переходит на 2147483647 и вновь начинает выполнять обратный счет.
.PRE	DINT	При достижении накопленным значением этого заданного значения инструкция устанавливает бит .DN.
.ACC	DINT	Накопленное значение соответствует количеству подсчитанных данной инструкцией переходов.

Описание: Инструкция CTD обычно используется вместе с инструкцией CTU, ссылающейся на ту же самую структуру COUNTER.

Когда инструкция CTD разрешена, а бит .CD сброшен, эта инструкция уменьшает значение счетчика на единицу. Когда инструкция разрешена, а бит .CD установлен, или когда инструкция запрещена, она сохраняет значение .ACC.



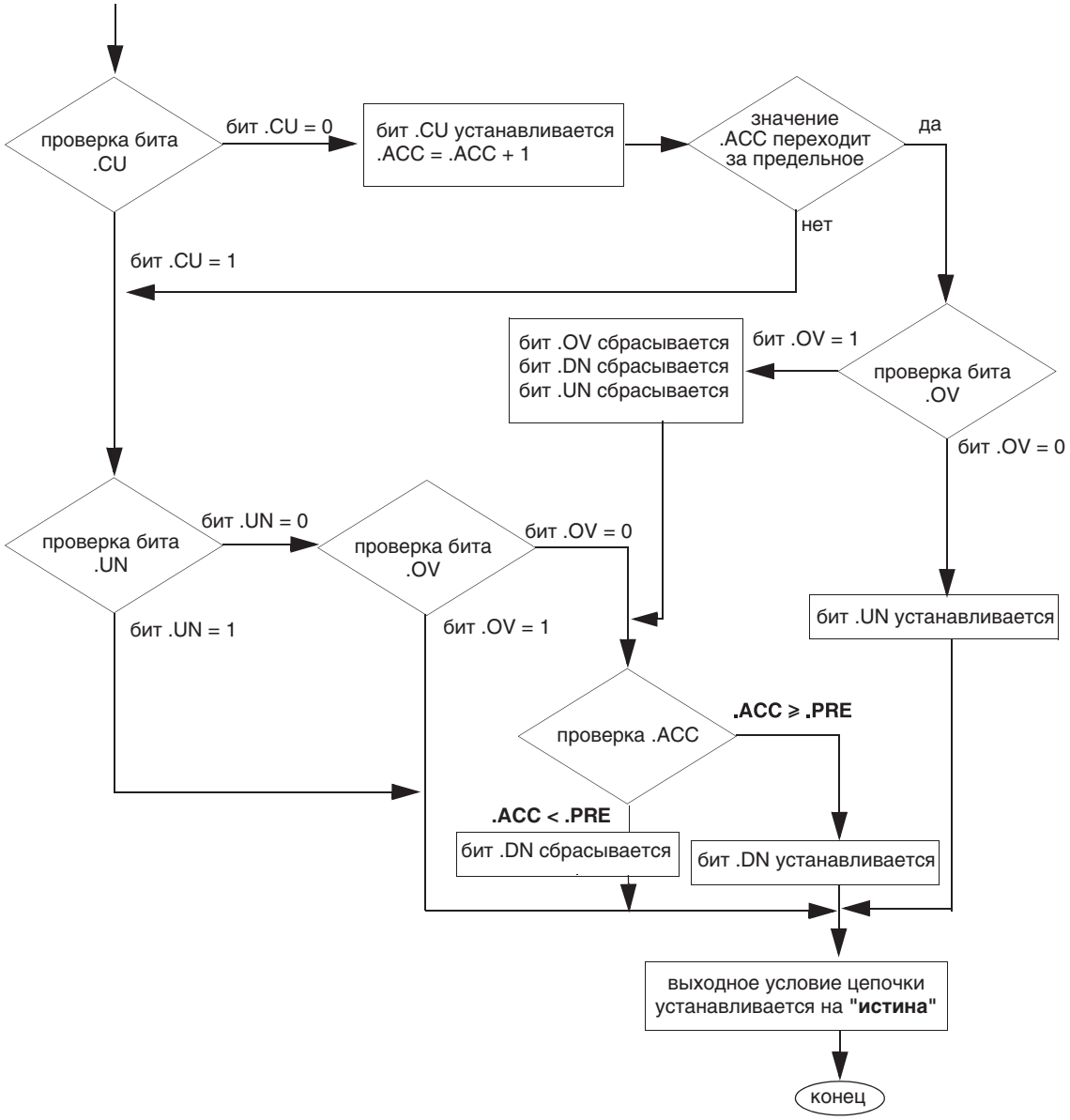
Накопленное значение продолжает уменьшаться даже после установки бита .DN. Для сброса накопленного значения используйте инструкцию RES со ссылкой на структуру COUNTER или запишите 0 в накопленное значение.

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

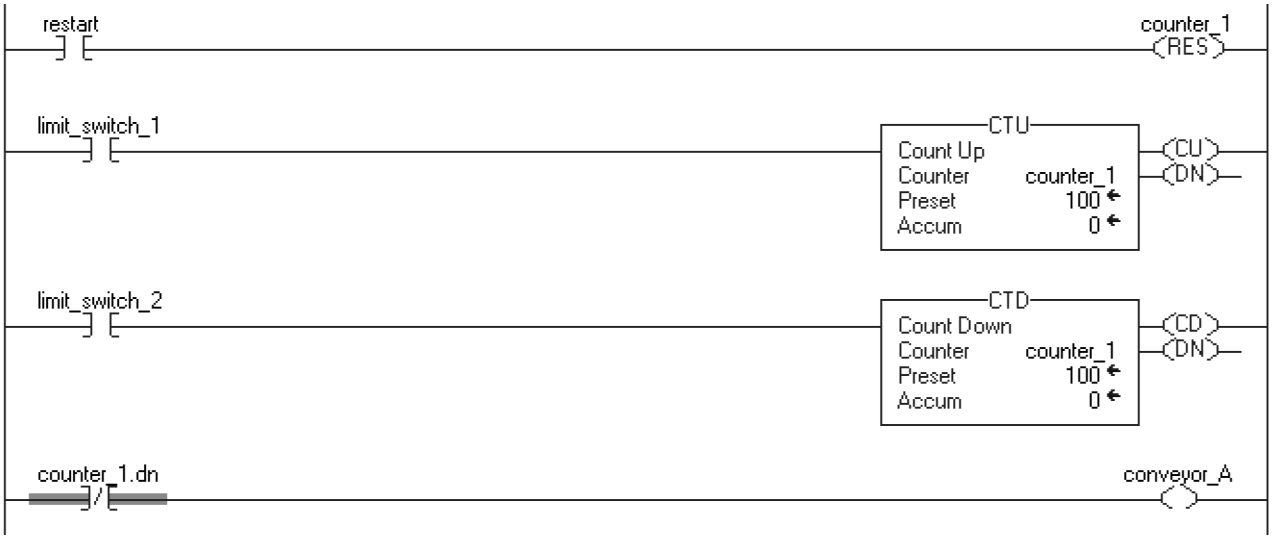
Выполнение:

Условие:	**Действие релейной логики:**
предварительное сканирование	Бит .CD устанавливается, чтобы предотвратить неверное уменьшение значения во время первого сканирования программы. Выходное условие цепочки устанавливается на "ложь".
входное условие цепочки "ложь"	Бит .CD сбрасывается. Выходное условие цепочки устанавливается на "ложь".
входное условие цепочки "истина"	



постсканирование	Выходное условие цепочки устанавливается на "ложь".

Пример: Конвейер доставляет детали в буферную зону. При каждом поступлении детали *limit_switch_1* включается, а *counter_1* увеличивается на единицу. При каждом убытии детали *limit_switch_2* включается, а *counter_1* уменьшается на единицу. Если в буферной зоне находится 100 деталей (установлен counter_1.dn), то *conveyor_a* включается и останавливает доставляющий детали конвейер до тех пор, пока в буфере не появится свободное место для новых деталей.



Count Up/Down (CTUD) (Прямой/обратный счет)

Инструкция CTUD увеличивает счет на единицу, когда параметр CUEnable переходит из сброшенного состояния в установленное. Инструкция уменьшает счет на единицу, когда параметр CDEnable переходит из сброшенного состояния в установленное.

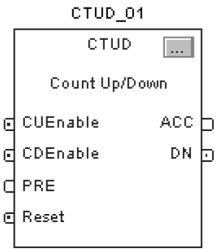
В релейной логике этой инструкции соответствуют три отдельные инструкции: CTU (см. стр. 2-24), CTD (см. стр. 2-28) и RES (см. стр. 2-36).

Операнды:

```
CTUD(CTUD_tag);
```

Структурированный текст

Переменная:	Тип:	Формат:	Описание:
тег CTUD	FBD_COUNTER	структура	структура CTUD



Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег CTUD	FBD_COUNTER	структура	структура CTUD

Структура FBD_COUNTER

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	**Функциональный блок:** Если этот параметр сброшен, инструкция не выполняется, и выходы не обновляются. Если он установлен, инструкция выполняется. По умолчанию параметр установлен. **Структурированный текст:** Никак не влияет. Инструкция выполняется.
CUEnable	BOOL	Разрешает прямой счет. Когда вход переходит из сброшенного состояния в установленное, сумматор увеличивается на единицу. По умолчанию параметр сброшен.
CDEnable	BOOL	Разрешает обратный счет. Когда вход переходит из сброшенного состояния в установленное, сумматор уменьшается на единицу. По умолчанию параметр сброшен.
PRE	DINT	Заданное значение счетчика. Это значение, которого должно достичь накопленное значение, чтобы установился параметр DN. Допустимым является любое целое число. По умолчанию равно 0.
Reset	BOOL	Запрос на сброс счетчика. Когда этот параметр установлен, счетчик сбрасывается. По умолчанию параметр сброшен.

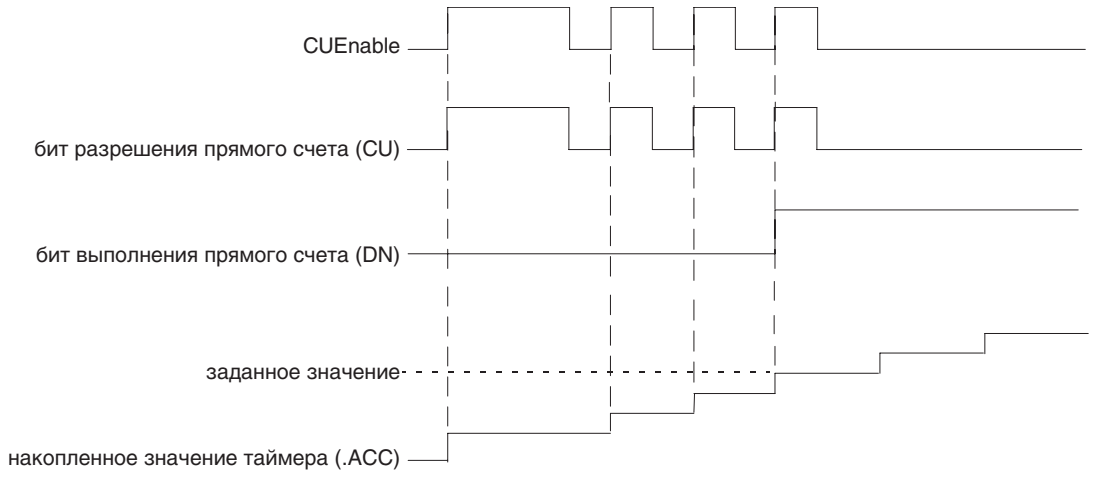
Выходной параметр:	Тип данных:	Описание:
EnableOut	BOOL	В результате выполнения инструкции получен допустимый результат.
ACC	DINT	Накопленное значение
CU	BOOL	Прямой счет разрешен.
CD	BOOL	Обратный счет разрешен.
DN	BOOL	Счет выполнен. Устанавливается, когда накопленное значение становится больше или равно заданному значению.
OV	BOOL	Переполнение счетчика. Указывает на то, что счетчик вышел за верхнее предельное значение 2147483647. При этом счетчик переходит на -2147483648 и вновь начинает выполнять прямой счет.
UN	BOOL	Отрицательное переполнение счетчика. Указывает на то, что счетчик вышел за нижнее предельное значение -2147483648. При этом счетчик переходит на 2147483648 м вновь начинает выполнять обратный счет.

Описание

Когда инструкция CTUD разрешена и установлен параметр CUEnable, инструкция увеличивает значение счетчика на единицу. Когда инструкция разрешена и установлен параметр CDEnable, эта инструкция уменьшает значение счетчика на единицу.

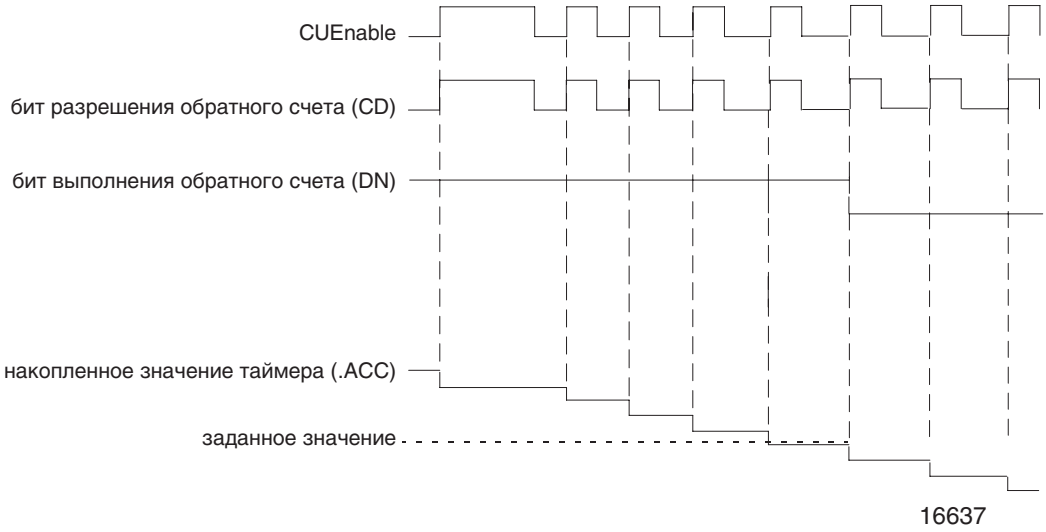
Оба входных параметра CUEnable и CDEnable могут переключаться во время одного и того же сканирования. При этом инструкция сначала выполняет прямой счет, а затем обратный счет.

Прямой счет



16636

Обратный счет



Когда инструкция CTUD запрещена, она сохраняет накопленное значение.. Для сброса инструкции установите входной параметр Reset структуры FBD_COUNTER.

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

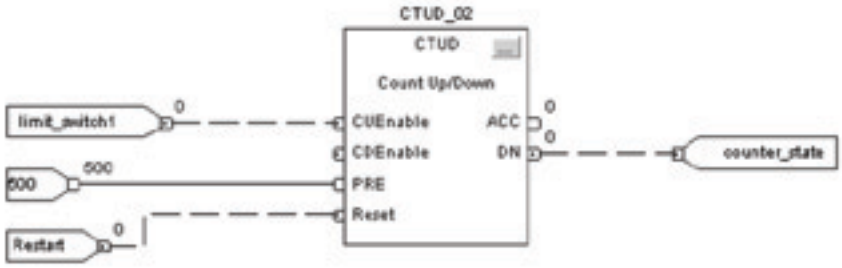
Условие:	Действие функционального блока:	Действие структурированного текста:
предварительное сканирование	Инициализация не требуется.	Инициализация не требуется.
первое сканирование инструкции	Устанавливаются CUEnable n-1 и CDEnable n-1.	Устанавливаются CUEnable n-1 и CDEnable n-1.
первое выполнение инструкции	Устанавливаются CUEnable n-1 и CDEnable n-1.	Устанавливаются CUEnable n-1 и CDEnable n-1.
EnableIn сбрасывается	EnableOut сбрасывается, инструкция ничего не выполняет, а выходы не обновляются.	не применимо
EnableIn устанавливается	Инструкция устанавливает CUEnable n-1 и CDEnable n-1. При переходе EnableIn со сброшенного состояния в установленное: • Инструкция выполняется. • EnableOut устанавливается.	Инструкция устанавливает CUEnable n-1 и CDEnable n-1. EnableIn всегда установлен. Инструкция выполняется.
сброс	Когда инструкция установлена, она сбрасывает CUEnable n-1, CDEnable n-1, CU, CD, DN, OV и UN и обнуляет ACC.	Когда инструкция установлена, она сбрасывает CUEnable n-1, CDEnable n-1, CU, CD, DN, OV и UN и обнуляет ACC.
постсканирование	Ничего не происходит.	Ничего не происходит.

Пример: Когда *limit_switch1* переходит из сброшенного состояния в установленное, CUEnable устанавливается на одно сканирование, а инструкция CTUD увеличивает значение ACC на 1. При выполнении условия ACC ⩾ PRE устанавливается параметр DN, что разрешает инструкцию функционального блока, следующую за инструкцией CTUD.

Структурированный текст

```
CTUD_01.Preset  :=  500;

CTUD_01.Reset  :=  Restart;

CTUD_O1.CUEnable  :=  limit_switch1;


CTUD(CTUD_01);


counter_state  :=  CTUD_01.DN;
```

Функциональный блок

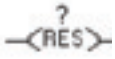


Reset (RES) (Сброс)

Инструкция RES сбрасывает структуру TIMER, COUNTER или CONTROL.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
структура	TIMER CONTROL COUNTER	тег	сбрасываемая структура

Описание:

Когда инструкция RES разрешена, она сбрасывает следующие

При использовании инструкции RES для:	Инструкция сбрасывает:
TIMER	значение .ACC биты состояния управления
COUNTER	значение .ACC биты состояния управления
CONTROL	значение .POS биты состояния управления

ВНИМАНИЕ

Поскольку инструкция RES сбрасывает значение .ACC, бит .DN и бит .TT, не используйте инструкцию RES для сброса таймера TOF.

Арифметические флаги состояния:

не затрагиваются

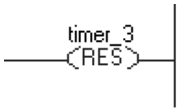
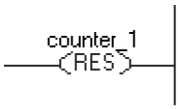
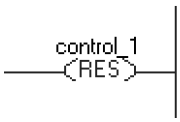
Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Выходное условие цепочки устанавливается на "ложь".
входное условие цепочки "ложь"	Выходное условие цепочки устанавливается на "ложь".
входное условие цепочки "истина"	Инструкция RES сбрасывает указанную структуру. Выходное условие цепочки устанавливается на "истина".
постсканирование	Выходное условие цепочки устанавливается на "ложь".

Примеры:

Пример:	Описание:
	Когда инструкция разрешена, *timer_3* сбрасывается.
	Когда инструкция разрешена, *counter_1* сбрасывается.
	Когда инструкция разрешена, *control_1* сбрасывается.

Примечания:

Инструкции ввода/вывода

(MSG, GSV, SSV, IOT)

Введение

Инструкции ввода/вывода считывают или записывают данные из контроллера или в него, или блок данных из другого модуля в другой сети или в другой модуль.

Если вы хотите:	Используйте эту инструкцию:	Имеющуюся в языках:	См. стр.:
отправить данные в другой модуль или из другого модуля	MSG	релейная логика структурированный текст	3-2
получить информацию о состоянии контроллера	GSV	релейная логика структурированный текст	3-34
задать информацию о состоянии контроллера	SSV	релейная логика структурированный текст	3-34
• отправить выходные значения в модуль ввода/вывода или контроллер-потребитель в конкретной точке вашей логики • запустить задачу обработки событий в другом контроллере	IOT	релейная логика структурированный текст	3-57

Message (MSG) (Сообщение)

Инструкция MSG осуществляет асинхронное чтение или запись блока данных в другой модуль в сети.

Операнды: Релейная логика

Операнд:	Тип:	Формат:	Описание:
Message control	MESSAGE	тег	структура message

Структурированный текст

Операнды те же самые, что и в инструкции MSG релейной логики.

Структура MESSAGE

Мнемоника:	Тип данных:	Описание:	
.FLAGS	INT	Член .FLAGS обеспечивает доступ к членам (битам) состояния в одном 16-разрядном слове.	
		Этот бит:	**Является следующим членом :**
		2	.EW
		4	.ER
		5	.DN
		6	.ST
		7	.EN
		8	.TO
		9	.EN_CC
		Внимание: Сброс каких-либо битов состояния MSG при разрешенной инструкции MSG может привести к нарушению связи.	
.ERR	INT	Если установлен бит .ER, слово кода ошибки указывает на коды ошибки для инструкции MSG.	
.EXERR	INT	Расширенный код ошибки указывает на дополнительную информацию по коду ошибки для некоторых кодов ошибки.	
.REQ_LEN	INT	Запрошенная длина указывает, сколько слов попытается передать инструкции MSG.	
.DN_LEN	INT	Выполненная длина указывает на фактически переданное количество слов.	
.EW	BOOL	Бит разрешения ожидания устанавливается, когда контроллер обнаруживает поступление запроса на сообщение в очередь. Контроллер сбрасывает бит .EW, когда устанавливается бит .ST.	
.ER	BOOL	Бит ошибки устанавливается, когда контроллер обнаруживает неудавшуюся попытку передачи данных. Бит .ER сбрасывается при следующем переходе входного условия цепочки из «ложь» в «истина».	
.DN	BOOL	Бит выполнения устанавливается после успешной передачи последнего пакета сообщения. Бит .DN сбрасывается при следующем переходе входного условия цепочки из «ложь» в «истина».	
.ST	BOOL	Стартовый бит устанавливается, когда контроллер начинает выполнять инструкцию MSG. Бит .ST сбрасывается при установке бита .DN или .ER.	

Мнемоника:	Тип данных:	Описание:
.EN	BOOL	Бит разрешения устанавливается, когда входное условие цепочки переходит на «истина», и остается установленным до тех пор, пока не будет установлен бит .DN или .ER и входное условие цепочки не перейдет на «ложь». Если входное условие цепочки перейдет на «ложь», а биты .DN и .ER сброшены, бит .EN останется установленным.
.TO	BOOL	Если вы вручную установите бит .TO, контроллер остановит обработку сообщения и установит бит .ER.
.EN_CC	BOOL	Бит разрешения кэширования определяет, каким образом обращаться с соединением MSG. См. раздел «Выбор опции кэширования» на стр. 3-31. Соединения для инструкций MSG, исходящих из последовательного порта, не кэшируются, даже если установлен бит .EN_CC.
.ERR_SRC	SINT	Используется программным обеспечением RSLogix 5000 для указания пути ошибки в диалоговом окне Message Configuration (Конфигурирование сообщения).
.DestinationLink	INT	Чтобы изменить Destination Link (Ссылка на приемник) сообщения DH+ или CIP with Source ID, установите этот член на требуемое значение.
.DestinationNode	INT	Чтобы изменить Destination Node (Узел-приемник) сообщения DH+ или CIP with Source ID, установите этот член на требуемое значение.
.SourceLink	INT	Чтобы изменить Source Link (Ссылка на источник) сообщения DH+ или CIP with Source ID, установите этот член на требуемое значение.
.Class	INT	Чтобы изменить параметр Class (Класс) сообщения CIP Generic, установите этот член на требуемое значение.
.Attribute	INT	Чтобы изменить параметр Attribute (Атрибут) сообщения CIP Generic, установите этот член на требуемое значение.
.Instance	DINT	Чтобы изменить параметр Instance (Экземпляр) сообщения CIP Generic, установите этот член на требуемое значение.
.LocalIndex	DINT	Если вы использует звездочку [*] для обозначения номера элемента для локального массива, LocalIndex соответствует номеру элемента. Чтобы изменить номер элемента, установите этот член на требуемое значение.
		Если сообщение: / **То локальный массив является:**
		считывает данные / элементом Destination (приемник)
		записывает данные / элементом Source (источник)
.Channel	SINT	Чтобы отправить сообщение из другого канала модуля 1756-DHRIO, установите этот член на требуемое значение. Используйте символ ASCII A или B.
.Rack	SINT	Чтобы изменить номер стойки для сообщения с поблочной передачей, установите этот член на требуемый номер стойки (восьмеричный).
.Group	SINT	Чтобы изменить номер группы для сообщения с поблочной передачей, установите этот член на требуемый номер группы (восьмеричный).
.Slot	SINT	Чтобы изменить номер слота для сообщения с поблочной передачей, установите этот член на требуемый номер слота.
		Если сообщение идет по следующей сети: / **Укажите номера слота как:**
		универсальный удаленный ввод/вывод / восьмеричное число
		ControlNet / десятичное число (0-15)
.Path	STRING	Чтобы отправить сообщение в другой контроллер, установите этот член на новый путь. · Введите путь в виде шестнадцатеричных значений. · Опустите запятые [,] Например, для пути 1, 0, 2, 42, 1, 3 введите $01$00$02$2A$01$03. Чтобы перейти к просмотру какого-либо устройства и автоматически создать всю новую строку или ее часть, нажмите на тег строки правой кнопкой мыши и выберите *Go to Message Path Editor* (Перейти в редактор пути сообщения). .Path

Мнемоника:	Тип данных:	Описание:
.RemoteIndex	DINT	Если вы использует звездочку [*] для обозначения номера элемента для удаленного массива, RemoteIndex соответствует номеру элемента. Чтобы изменить номер элемента, установите этот член на требуемое значение. **Если сообщение:** — **То удаленный массив является:** считывает данные — элементом Source (источник) записывает данные — элементом Destination (приемник)
.RemoteElement	STRING	Для задания другого тега или адреса в контроллере, в который направляется сообщение, установите этот член на требуемое значение. Введите тег или адрес в виде символов ASCII. **Если сообщение:** — **То удаленный массив является:** считывает данные — элементом Source (источник) записывает данные — элементом Destination (приемник)
.UnconnectedTimeout	DINT	Тайм-аут для сообщений без соединения. Значение по умолчанию - 30 секунд.
.ConnectionRate	DINT	Умножение ConnectionRate (частота соединения) на TimeoutMultiplier (коэффициент тайм-аутов) дает тайм-аут для сообщений с установленным соединением. · Значение ConnectionRate по умолчанию - 7,5 секунд. · Значение TimeoutMultiplier по умолчанию - 0 (что соответствует множителю 4). · Тайм-аут для сообщений с установленным соединением по умолчанию составляет 30 секунд (7,5 секунд x 4 = 30 секунд). · Для изменения значения тайм-аута измените ConnectionRate, сохранив значение по умолчанию TimeoutMultiplier.
.TimeoutMultiplier	SINT	

ВНИМАНИЕ

Контроллер обрабатывает биты .ST, .EW, .DN и .ER асинхронно по отношению к сканированию программы. Для проверки этих битов в релейной логике скопируйте слово .FLAGS в тег INT и проверьте эти биты оттуда. В противном случае проблемы синхронизации могут нарушить работу вашего приложения с возможным повреждением оборудования и причинением травм персоналу.

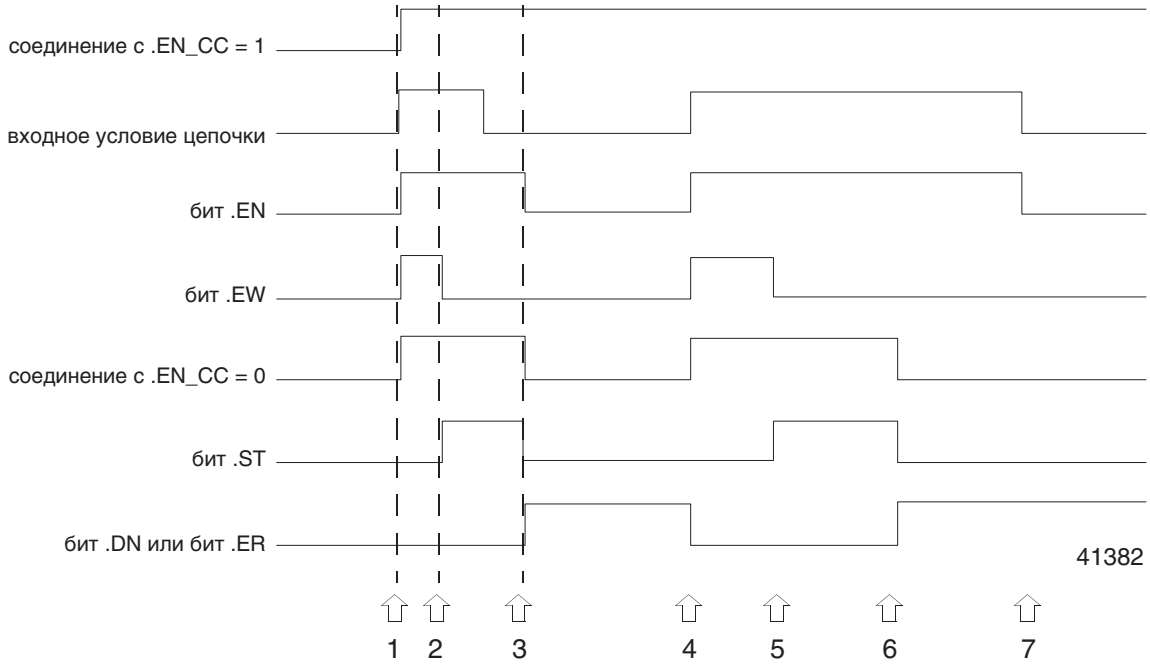
Описание

Инструкция MSG передает элементы данных.

Это инструкция перехода:

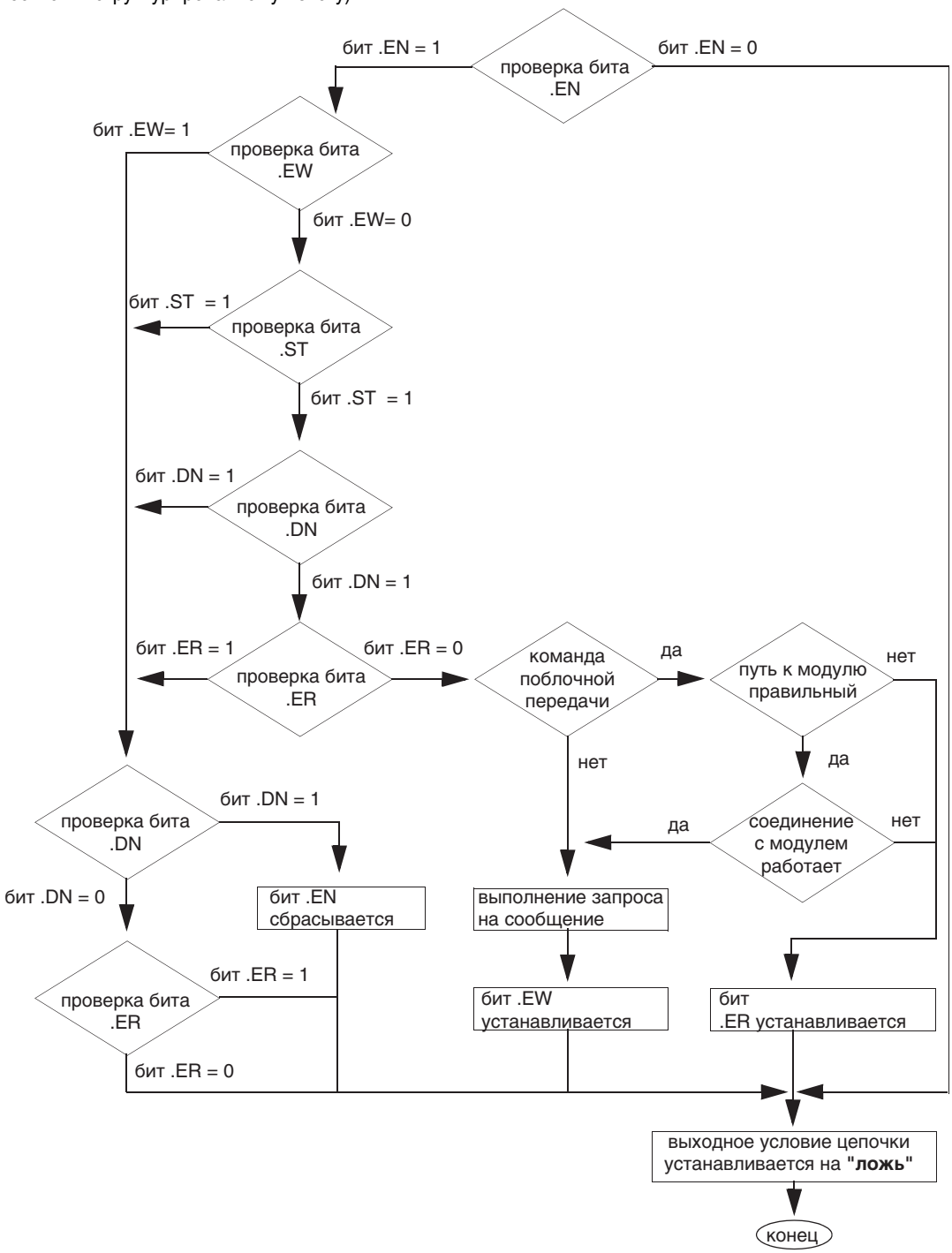
- В релейной логике переключайте входное условие цепочки со сброшенного в установленное всякий раз, когда должна быть выполнена эта инструкция.
- В структурированном тексте обусловьте эту инструкцию таким образом, чтобы она выполнялась лишь при переходе. См. Приложение C.

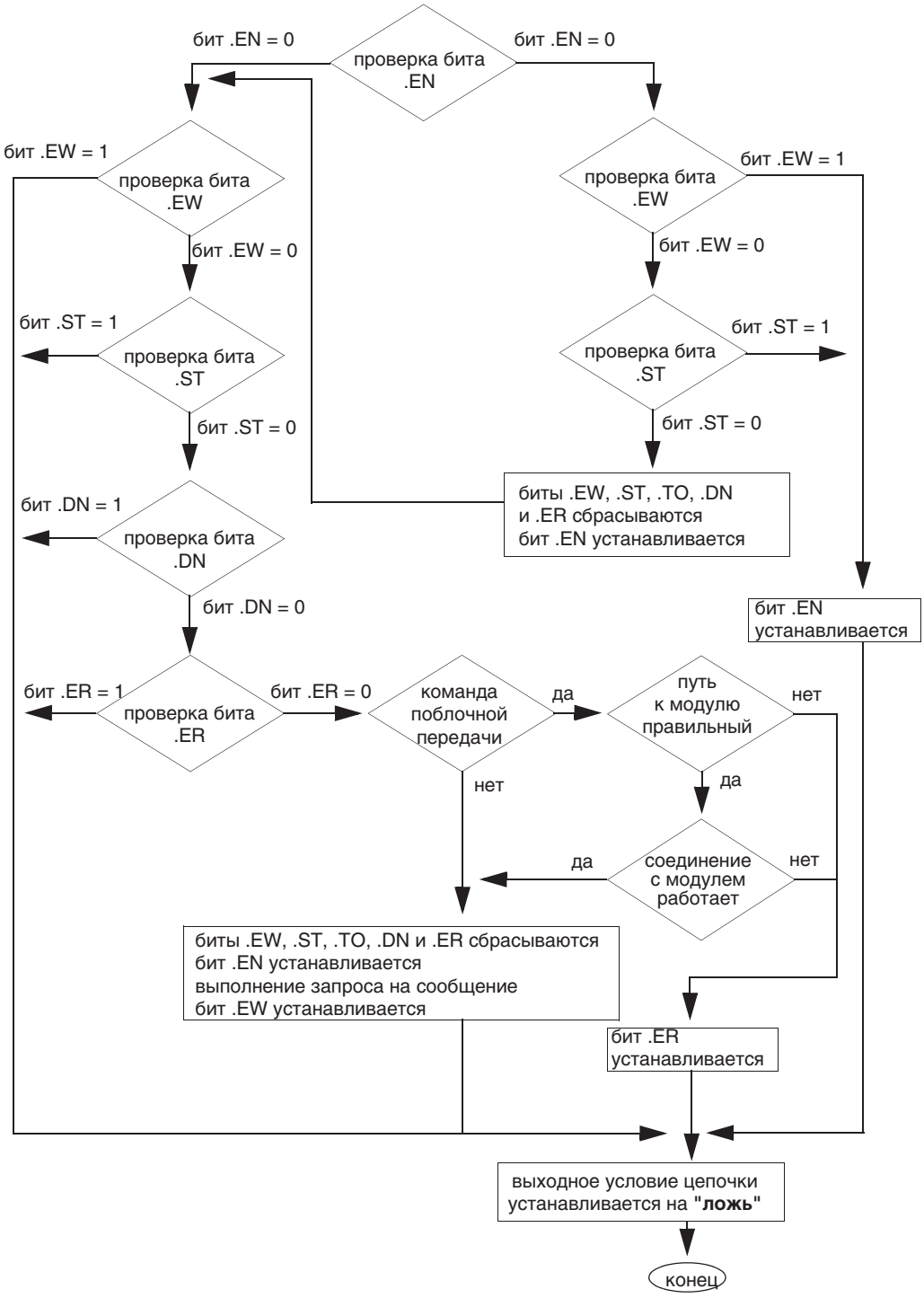
Размер каждого элемента зависит от заданных вами типов данных, а также используемого вами типа инструкции передачи сообщения.



Где:	Описание:	Где:	Описание:
1	входное условие цепочки «истина» .EN устанавливается .EW устанавливается соединение открывается*	5	сообщение отправляется .ST устанавливается .EW сбрасывается
2	сообщение отправляется .ST устанавливается .EW сбрасывается	6	сообщение выполнено или произошла ошибка входное условие цепочки по-прежнему «истина» .DN или .ER устанавливается .ST сбрасывается соединение закрывается (если .EN_CC= 0)
3	сообщение выполнено или произошла ошибка входное условие цепочки «ложь» .DN или .ER устанавливается .ST сбрасывается соединение закрывается (если .EN_CC= 0) .EN сбрасывается (входное условие цепочки «ложь»)	7	входное условие цепочки переходит на «ложь», и устанавливается .DN или .ER .EN сбрасывается
4	входное условие цепочки «истина» .DN или .ER был устновлен ранее .EN устанавливается .EW устанавливается соединение открывается* .DN или .ER сбрасывается		

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Ничего не происходит.
входное условие цепочки «ложь» (не относится к структурированному тексту)		
входное условие цепочки "истина"	Инструкция выполняется. Выходное условие цепочки устанавливается на "истина".	не применимо

Условие:	**Действие релейной логики:**	**Действие структурированного текста:**
EnableIn установлен	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции		
постсканирование	Выходное условие цепочки устанавливается на "ложь".	Ничего не происходит.

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Коды ошибки для инструкции MSG

Коды ошибки зависят от типа инструкции MSG.

Коды ошибки

Программное обеспечение RSLogix 5000 не всегда выводит на экран полное описание

Код ошибки (шестнадца-теричный):	Описание:	Сообщение программного обеспечения:
0001	Connection failure (Нарушение соединения) (см. расширенные коды ошибки)	совпадает с описанием
0002	Insufficient resource (Не хватает ресурса)	совпадает с описанием
0003	Invalid value (Недопустимое значение)	совпадает с описанием
0004	IOI syntax error (Ошибка в синтаксисе инструкции ввода/вывода) (см. расширенные коды ошибки)	совпадает с описанием
0005	Destination unknown, class unsupported, instance undefined or structure element undefined (Приемник не известен, класс не поддерживается, экземпляр не определен или элемент структуры не определен) (см. расширенные коды ошибки)	совпадает с описанием
0006	Insufficient packet space (Недостаточно места для пакета)	совпадает с описанием
0007	Connection lost (Потеря соединения)	совпадает с описанием
0008	Service unsupported (Сервис не поддерживается)	совпадает с описанием
0009	Error in data segment or invalid attribute value (Ошибка в сегменте данных или неверное значение атрибута)	совпадает с описанием
000A	Attribute list error (Ошибка в списке атрибутов)	совпадает с описанием
000B	State already exists (Состояние уже существует)	совпадает с описанием
000C	Object model conflict (Конфликт объектной модели)	совпадает с описанием
000D	Object already exists (Объект уже существует)	совпадает с описанием
000E	Attribute not settable (Атрибут не задается)	совпадает с описанием
000F	Permission denied (Разрешение не дано)	совпадает с описанием
0010	Device state conflict (Конфликт состояния устройства)	совпадает с описанием
0011	Reply will not fit (Ответ не подойдет)	совпадает с описанием
0012	Fragment primitive (Примитив фрагмента)	совпадает с описанием
0013	Insufficient command data (Недостаточные данные инструкции)	совпадает с описанием
0014	Attribute not supported (Атрибут не поддерживается)	совпадает с описанием

Код ошибки (шестнадца-теричный):	**Описание:**	**Сообщение программного обеспечения:**
0015	Too much data (Слишком много данных)	совпадает с описанием
001A	Bridge request too large (Слишком большой запрос к мосту)	совпадает с описанием
001B	Bridge response too large (Слишком большой ответ от моста)	совпадает с описанием
001C	Attribute list shortage (Нехватка списка атрибутов)	совпадает с описанием
001D	Invalid attribute list (Неверный список атрибутов)	совпадает с описанием
001E	Embedded service error (Ошибка встроенного сервиса)	совпадает с описанием
001F	Connection related failure (Отказ, связанный с соединением) (см. расширенные коды ошибки)	совпадает с описанием
0022	Invalid reply received (Получен недопустимый ответ)	совпадает с описанием
0025	Key segment error (Ошибка ключевого сегмента)	совпадает с описанием
0026	Invalid IOI error (Недопустимая ошибка инструкции ввода/вывода)	совпадает с описанием
0027	Unexpected attribute in list (Непредвиденный атрибут в списке)	совпадает с описанием
0028	DeviceNet error – invalid member ID (Ошибка DeviceNet – неверный ID члена)	совпадает с описанием
0029	DeviceNet error – member not settable (ошибка DeviceNet – член не задается)	совпадает с описанием
00D1	Module not in run state (Модуль не в рабочем состоянии)	unknown error (неизвестная ошибка)
00FB	Message port not supported (Порт сообщения не поддерживается)	unknown error (неизвестная ошибка)
00FC	Message unsupported data type (Не поддерживаемый сообщением тип данных)	unknown error (неизвестная ошибка)
00FD	Message uninitialized (Сообщение не инициализировано)	unknown error (неизвестная ошибка)
00FE	Message timeout (Тайм-аут сообщения)	unknown error (неизвестная ошибка)
00FF	General error (Общая ошибка) (см. расширенные коды ошибки)	unknown error (неизвестная ошибка)

Расширенные коды ошибки

Для расширенных кодов ошибки программное обеспечение RSLogix 5000 не выводит на экран никакой текст.

Для кода ошибки **0001** имеются следующие расширенные коды ошибки.

Расширенный код ошибки (шестнадцате-ричный):	Описание:
0100	Соединение занято
0103	Передача данных не поддерживается
0106	Конфликт принадлежности
0107	Соединение не найдено
0108	Недопустимый тип соединения
0109	Недопустимый размер соединения
0110	Модуль не сконфигурирован
0111	EPR не поддерживается
0114	Неверный модуль
0115	Неверный тип устройства
0116	Неверная версия
0118	Недопустимый формат конфигурации
011A	Приложение без соединения

Расширенный код ошибки (шестнадцате-ричный):	Описание:
0203	Тайм-аут соединения
0204	Тайм-аут сообщения без соединения
0205	Ошибка параметра передачи без соединения
0206	Слишком большое сообщение
0301	Отсутствует буферная память
0302	Отсутствует полоса пропускания
0303	Отсутствуют скринеры
0305	Совпадение сигнатуры
0311	Порт отсутствует
0312	Адрес связи отсутствует
0315	Недопустимый тип сегмента
0317	Соединение не запланировано

Для кода ошибки **001F** имеются следующие расширенные коды ошибки.

Расширенный код ошибки (шестнадцатеричный):	Описание:
0203	Тайм-аут соединения

Для кодов ошибки **0004** и **0005** имеются следующие расширенные коды ошибки.

Расширенный код ошибки (шестнадцатеричный):	Описание:
0000	расширенному состоянию не хватает памяти
0001	расширенному состоянию не хватает экземпляров

Для кода ошибки 00FF имеются следующие расширенные коды ошибки.

Расширенный код ошибки (шестнадцате-ричный):	**Описание:**
2001	Излишняя команда ввода/вывода
2002	Неверное значение параметра
2018	Отказ от семафора
201B	Слишком маленький размер
201C	Недопустимый размер
2100	Сбой привилегии
2101	Недопустимое положение кнопочного переключателя
2102	Недопустимый пароль
2103	Пароль не введен
2104	Адрес за пределами допустимого диапазона
2105	Адрес и количество за пределами допустимого диапазона
2106	Данные используются
2107	Тип недопустим или не поддерживается

Расширенный код ошибки (шестнадцате-ричный):	**Описание:**
2108	Контроллер находится в режиме загрузки
2109	Попытка изменения количества размерностей массива
210A	Недопустимое символическое имя
210B	Символ не существует
210E	Поиск завершился неудачно
210F	Задача не может запуститься
2110	Невозможно записать
2111	Невозможно считать
2112	Совместно используемая процедура не может редактироваться
2113	Контроллер неисправен
2114	Режим выполнения запрещен

Коды ошибки PLC и SLC (.ERR)

Микропрограммное обеспечение Logix версии 10.x и выше содержит новые коды для ошибок, связанных с сообщениями типа PLC и SLC (сообщения PCCC).

- Это изменение позволяет программному обеспечению RSLogix 5000 выводить на экран более содержательное описание для многих ошибок. Ранее программное обеспечение не давало описания никаких ошибок, относящихся к коду ошибки 00F0.
- Это изменение также увеличивает согласованность кодов ошибки с ошибками, возвращаемыми другими контроллерами, например, PLC-5.

В следующей таблице отражены изменения в кодах ошибки между версиями R9.x и более ранними и R10.x и более поздними. В результате этих изменений член .ERR возвращает уникальное значение для каждой ошибки PCCC. .EXERR для таких ошибок больше не требуется.

Таблица 3.1 Коды ошибки PLC и SLC (шестнадцатеричные)

Версия R9.x и предшествующие		Версия R10.x и последующие		Описание:
.ERR	.EXERR	.ERR	.EXERR	
0010		1000		Illegal command or format from local processor (Недопустимая команда или формат от локального процессора)
0020		2000		Communication module not working (Модуль связи не работает)
0030		3000		Remote node is missing, disconnected or shut down (Удаленный узел отсутствует, отсоединен или запрещен)
0040		4000		Processor connected but faulted (hardware) (Процессор подключился, но дал сбой (аппаратный))
0050		5000		Wrong station number (Неверный номер станции)
0060		6000		Requested function is not available (Запрашиваемая функция отсутствует)
0070		7000		Processor is in Program mode (Процессор в режиме программирования)
0080		8000		Processor's compatibility file does not exist (Файла совместимости процессора не существует)
0090		9000		Remote node cannot buffer command (Удаленный узел не может буферизовать команду)
00B0		B000		Processor is downloading so it is not accessible (Процессор загружает данные и поэтому недоступен)
00F0	0001	F001		Processor incorrectly converted the address (Процессор неверно преобразовал адрес)
00F0	0002	F002		Incomplete address (Неполный адрес)
00F0	0003	F003		Incorrect address (Неверный адрес)
00F0	0004	F004		Illegal address format - symbol not found (Недопустимый формат адреса - символ не найден)
00F0	0005	F005		Illegal address format - symbol has 0 or greater than the maximum number of characters supported by the device (Недопустимый формат адреса - символ содержит 0 или длиннее поддерживаемого устройством количества знаков)

Версия R9.x и предшествующие		Версия R10.x и последующие		Описание:
.ERR	**.EXERR**	**.ERR**	**.EXERR**	
00F0	0006	F006		Address file does not exist in target processor (Файл адресов отсутствует в целевом процессоре)
00F0	0007	F007		Destination file is too small for the number of words requested (Файл назначения слишком мал для запрашиваемого количества слов)
00F0	0008	F008		Cannot complete request Situation changed during multipacket operation (Невозможно выполнить запрос Во время многопакетной операции изменилась ситуация)
00F0	0009	F009		Data or file too large Memory unavailable
00F0	000A	F00A		Target processor cannot put requested information in packets (Целевой процессор не может разместить запрашиваемую информацию в пакеты)
00F0	000B	F00B		Privilege error; access denied (Ошибка привилегии; в доступе отказано)
00F0	000C	F00C		Requested function is not available (Запрашиваемая функция отсутствует)
00F0	000D	F00D		Request is redundant (Избыточный запрос)
00F0	000E	F00E		Command cannot be executed (Команда не может быть выполнена)
00F0	000F	F00F		Overflow; histogram overflow (Переполнение; переполнение гистограммы)
00F0	0010	F010		No access (Нет доступа)
00F0	0011	F011		Data type requested does not match data available (Запрашиваемый тип данных не соответствует имеющимся данным)
00F0	0012	F012		Incorrect command parameters (Неверные параметры команды)
00F0	0013	F013		Address reference exists to deleted area (Существует адресная ссылка на удаленную область)
00F0	0014	F014		Command execution failure for unknown reason PLC-3 histogram overflow (Переполнение гистограммы PLC-3)
00F0	0015	F015		Data conversion error (Ошибка преобразования данных)
00F0	0016	F016		The scanner is not available to communicate with a 1771 rack adapter (Сканер не доступен для коммуникации с адаптером стойки 1771)
00F0	0017	F017		The adapter is not available to communicate with the module (Адаптер не доступен для коммуникации с модулем)
00F0	0018	F018		The 1771 module response was not valid (Неверный ответ модуля 1771)
00F0	0019	F019		Duplicate label (Продублированная метка)
00F0	001A	F01A		File owner active - the file is being used (Владелец файла активен - файл используется)
00F0	001B	F01B		Program owner active - someone is downloading or editing online (Владелец программы активен - кто-то загружает данные или редактирует он-лайн)
00F0	001C	F01C		Disk file is write protected or otherwise not accessible (offline only) (Диск защищен от записи или недоступен по иной причине (только оффлайн))
00F0	001D	F01D		Disk file is being used by another application (Дисковый файл используется другим приложением) Update not performed (offline only) (Обновление не выполняется (только оффлайн)

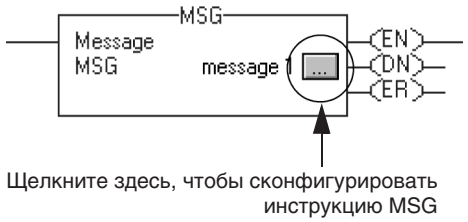
Коды ошибки при поблочной передаче

В Logix5000 имеются следующие коды ошибки, относящиеся к поблочной передаче.

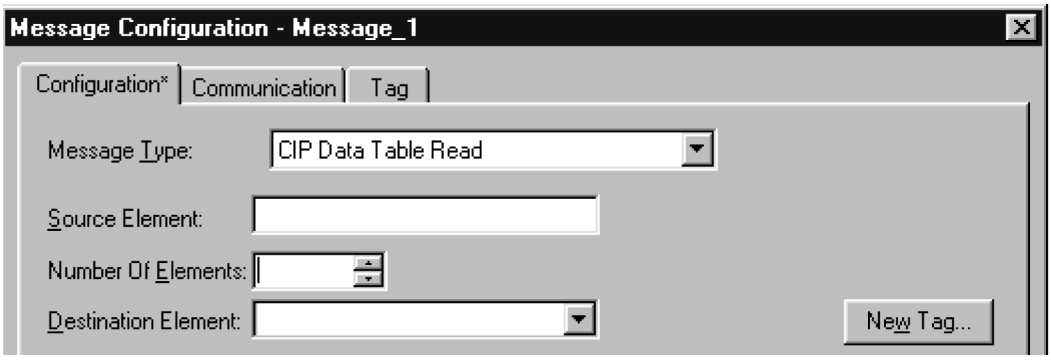
Код ошибки (шестнадца-теричный):	Описание:	Сообщение программного обеспечения:
00D0	Сканер не получил ответ на запрос поблочной передачи от модуля поблочной передачи через 3,5 секунды после запроса	unknown error (неизвестная ошибка)
00D1	Контрольная сумма ответа на запрос чтения не соответствует контрольной сумме потока данных	unknown error (неизвестная ошибка)
00D2	Сканер запросил чтение или запись, но модуль поблочной передачи ответил противоположным образом	unknown error (неизвестная ошибка)
00D3	Сканер запросил длину, а модуль поблочной передачи дал ответ с другой длиной	unknown error (неизвестная ошибка)
00D6	Сканер получил ответ от модуля поблочной передачи, говорящий о невыполнении запроса записи	unknown error (неизвестная ошибка)
00EA	Сканер не был сконфигурирован для коммуникации со стойкой, содержащей данный модуль поблочной передачи	unknown error (неизвестная ошибка)
00EB	Указанный логический слот отсутствует для данного размера стойки	unknown error (неизвестная ошибка)
00EC	Идет обработка запроса на поблочную передачу, и требуется ответ до начала обработки следующего запроса	unknown error (неизвестная ошибка)
00ED	Размер запроса на поблочную передачу не соответствует запросам на поблочную передачу допустимого размера	unknown error (неизвестная ошибка)
00EE	Тип запроса на поблочную передачу не соответствует ожидаемому BT_READ или BT_WRITE	unknown error (неизвестная ошибка)
00EF	Сканер не смог найти доступный слот в таблице поблочной передачи для выполнения запроса на поблочную передачу	unknown error (неизвестная ошибка)
00F0	Сканер получил запрос на сброс удаленных каналов ввода/вывода при наличии невыполненных пересылок блоков	unknown error (неизвестная ошибка)
00F3	Очереди на удаленные пересылки блоков заполнены	unknown error (неизвестная ошибка)
00F5	Нет сконфигурированных каналов связи для запрашиваемой стойки или слота	unknown error (неизвестная ошибка)
00F6	Нет сконфигурированных каналов связи для дистанционного ввода/вывода	unknown error (неизвестная ошибка)
00F7	Указанное в инструкции время ожидания для поблочной передачи истекло до завершения передачи	unknown error (неизвестная ошибка)
00F8	Ошибка в протоколе поблочной передачи – незатребованная пересылка блоков	unknown error (неизвестная ошибка)
00F9	Данные поблочной передачи утеряны из-за неисправного канала связи	unknown error (неизвестная ошибка)
00FA	Модуль поблочной передачи запросил другую длину, чем соответствующая инструкция поблочной передачи	unknown error (неизвестная ошибка)
00FB	Контрольная сумма считанных данных при поблочной передаче была неверной	unknown error (неизвестная ошибка)
00FC	Была выполнена недопустимая передача данных между адаптером и модулем поблочной передачи при поблочной записи	unknown error (неизвестная ошибка)
00FD	Размер передаваемого блока данных плюс размер индекса в таблице поблочно пересылаемых данных был больше размера файла таблицы данных для поблочной передачи	unknown error (неизвестная ошибка)

Задание деталей конфигурации

После ввода инструкции MSG и задания структуры MESSAGE определите детали сообщения в диалоговом окне Message Configuration (Конфигурация сообщения).



Задаваемые вами детали зависят от выбранного вами типа сообщения.



Если целевым устройством является:	Выберите один из следующих типов сообщения:	См. стр.
Контроллер Logix5000	CIP Data Table Read (Чтение таблицы данных CIP)	3-16
	CIP Data Table Write (Запись таблицы данных CIP)	
Модуль ввода/вывода, который вы конфигурируете с помощью ПО RSLogix 5000	Module Reconfigure (Реконфигурирование модуля)	3-17
	CIP Generic (Общее CIP)	3-18
Контроллер PLC-5	PLC5 Typed Read (Типовое чтение из PLC-5)	3-19
	PLC5 Typed Write (Типовая запись в PLC-5)	
	PLC5 Word Range Read (Чтение слов из PLC-5)	
	PLC5 Word Range Write (Запись слов в PLC-5)	
Контроллер SLC Контроллер MicroLogix	SLC Typed Read (Типовое чтение из SLC	3-20
	SLC Typed Write (Типовая запись в SLC)	
Модуль поблочной передачи	Block-Transfer read (Чтение с поблочной передачей)	3-21
	Block-Transfer Write (Запись с поблочной передачей)	
Процессор PLC-3	PLC3 typed read (Типовое чтение из PLC-3)	3-22
	PLC3 typed write (Типовая запись в PLC-3)	
	PLC3 word range read (Чтение слов из PLC-3)	
	PLC3 word range write (Запись слов в PLC-3)	
Процессор PLC-2	PLC2 unprotected read (Незащищенное чтение из PLC-2)	3-23
	PLC2 unprotected write (Незащищенная запись в PLC-2)	

Вы должны указать следующую информацию по конфигурации:

Для данного свойства:	Укажите:
Source Element (Элемент-источник)	· Если вы выбрали тип сообщения - «чтение» (read), Source Element представляет собой адрес данных, которые вы хотите считать в целевое устройство. Используйте синтаксис адресации целевого устройства. · Если вы выбрали тип сообщения – «запись» (write), то Source Element представляет собой первый элемент тега, который вы хотите отправить в целевое устройство.
Number of Elements (Количество элементов)	Количество считываемых/записываемых вами элементов зависит от типа используемых вами данных. Элемент соответствует одной «порции» соответствующих данных. Например, *timer1* – это один элемент, состоящий из одной структуры управления таймером.
Destination Element (Целевой элемент)	· Если вы выбрали тип сообщения - «чтение» (read), Destination Element представляет собой первый элемент тега в контроллере Logix5000, где вы хотите сохранить данные, считанные из целевого устройства. · Если вы выбрали тип сообщения – «запись» (write), то Destination Element представляет собой адрес ячейки в целевом устройстве, куда вы хотите записать данные.

Задание сообщений типа CIP Data Table Read и CIP Data Table Write (чтения и записи таблиц данных CIP)

Сообщения типа CIP Data Table Read и CIP Data Table Write осуществляют передачу данных между контроллерами Logix5000.

Выберите эту инструкцию:	Если вы хотите:
CIP Data Table Read	считать данные из другого контроллера. Типы источника (Source) и приемника (Destination) должны совпадать.
CIP Data Table Write	записать данные в другой контроллер. Типы источника (Source) и приемника (Destination) должны совпадать.

Реконфигурирование модуля ввода/вывода

Используйте сообщение типа Module Reconfigure для отправки информации о новой конфигурации в модуль ввода/вывода. Во время реконфигурирования:

- Модули ввода продолжают отправлять входные данные в контроллер.
- Модули вывода продолжают управлять своими устройствами вывода.

Для сообщения типа Module Reconfigure требуются следующие свойства конфигурации:

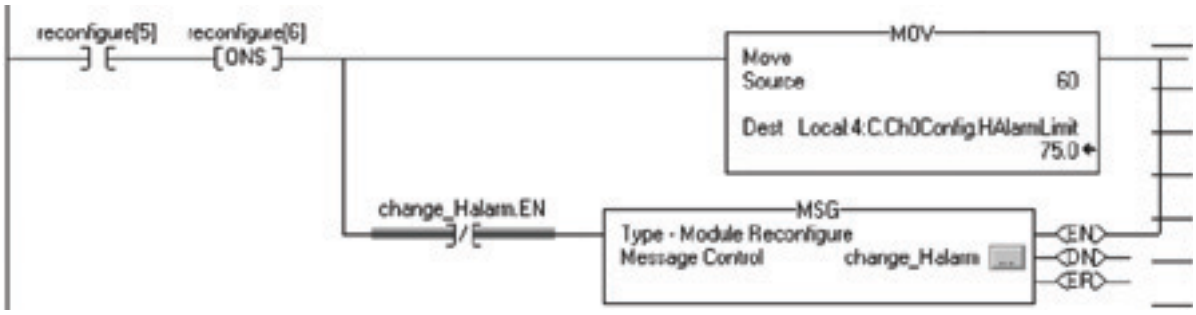
В этом свойстве:	Выберите:
Message Type (Тип сообщения)	Module Reconfigure

Пример: Чтобы реконфигурировать модуль ввода/вывода:

1. Установите соответствующий член тега конфигурации модуля на новое значение.
2. Направьте в реконфигурируемый модуль сообщение Module Reconfigure.

При установленном *reconfigure*[5] установите сигнализацию высокого значения на 60 для локального модуля в слоте 4. После этого сообщение Reconfigure Module направит в модуль новое значение сигнализации. Команда ONS предотвращает отправку цепочкой в модуль множества сообщений, пока *reconfigure*[5] остается установленным.

Релейная логика



Структурированный текст

```
IF reconfigure[5] AND NOT reconfigure[6]THEN
     Local:4:C.Ch0Config.HAlarmLimit := 60;
     IF NOT change_Halarm.EN THEN
          MSG(change_Halarm);
     END_IF;
END_IF;
     reconfigure[6] := reconfigure[5];
```

Задание сообщений типа CIP Generic (Общие CIP)

Сообщение типа CIP Generic выполняет конкретное действие над модулем ввода/вывода.

Если вы хотите:	В следующем свойстве:	Введите или выберите:	
Выполнить импульсный тест на модуле цифрового вывода	Message Type (Тип сообщения)	CIP Generic	
	Service Type (Тип сервиса)	Pulse Test	
	Source (Источник)	*tag_name* типа INT [5]	
		Этот массив содержит:	
		tag_name[0]	битовая маска тестируемых точек (поточечное тестирование)
		tag_name[1]	резервное, оставьте 0
		tag_name[2]	длительность импульса (в сотнях мкс, обычно 20)
		tag_name[3]	задержка перехода через нуль для ввода/вывода ControlLogix (в сотнях мкс, обычно 40)
		tag_name[4]	проверка задержки
	Destination (Приемник)	оставьте пустым	
Сбросить электронные предохранители на модуле цифрового вывода	Message Type	CIP Generic	
	Service Type	Reset Electronic Fuse	
	Source	*tag name* типа DINT Этот тег представляет собой битовую маску точек, для которых сбрасываются предохранители.	
	Destination	оставьте пустым	
Сбросить зафиксированную защелкой диагностику на модуле цифрового ввода	Message Type	CIP Generic	
	Service Type	Reset Latched Diagnostics (I)	
	Source	*tag name* типа DINT Этот тег представляет собой битовую маску точек, для которых сбрасывается диагностика.	
Сбросить зафиксированную защелкой диагностику на модуле цифрового вывода	Message Type	CIP Generic	
	Service Type	Reset Latched Diagnostics (O)	
	Source	*tag name* типа DINT Этот тег представляет собой битовую маску точек, для которых сбрасывается диагностика.	
Снять защелку для сигнализации модуля аналогового ввода	Message Type	CIP Generic	
	Service Type	Выберите, для какой сигнализации вы хотите снять защелку: · Unlatch All Alarms (I) (вся сигнализация) · Unlatch Analog High Alarm (I) (сигнализация высокого аналогового значения) · Unlatch Analog High High Alarm (I) (сигнализация высокого-высокого аналогового значения) · Unlatch Analog Low Alarm (I) · (сигнализация низкого аналогового значения) · Unlatch Analog Low Low Alarm (I) (сигнализация низкого-низкого аналогового значения) · Unlatch Rate Alarm (I) (сигнализация скорости)	
	Instance (Экземпляр)	Канал сигнализации, для которой вы хотите снять защелку	

Если вы хотите:	В следующем свойстве:	Введите или выберите:
Снять защелку для сигнализации модуля аналогового вывода	Message Type	CIP Generic
	Service Type	Выберите, для какой сигнализации вы хотите снять защелку: · Unlatch All Alarms (O) (вся сигнализация) · Unlatch High Alarm (O) (сигнализация высокого значения) · Unlatch Low Alarm (O) (сигнализация низкого значения) · Unlatch Ramp Alarm (O) (сигнализация линейного нарастания)
	Instance (Экземпляр)	Канал сигнализации, для которой вы хотите снять защелку

Задание сообщений PLC-5

Используйте различные типы сообщений PLC-5 для обмена данными с контроллерами PLC-5.

Выберите следующую команду:	Если вы хотите:
PLC5 Typed Read	Считать16-разрядные целочисленные данные, данные с плавающей точкой или строкового типа с обеспечением целостности данных. См. Таблицу 3.2 на стр. 3-19.
PLC5 Typed Write	Записать 16-разрядные целочисленные данные, данные с плавающей точкой или строкового типа с обеспечением целостности данных. См. Таблицу 3.2 на стр. 3-19.
PLC5 Word Range Read	Считать непрерывный ряд 16-разрядных слов из памяти PLC-5 независимо от типа данных. Эта команда начинает работать с адреса, указанного в качестве Source Element, и последовательно считывает запрошенное количество 16-разрядных слов. Данные из Source Element сохраняются начиная с адреса, указанного в качестве Destination Tag.
PLC5 Word Range Write	Записать непрерывный ряд 6-разрядных слов из памяти Logix5000 в память PLC-5 независимо от типа данных. Эта команда начинает работать с адреса, указанного в качестве Source Tag, и последовательно считывает запрошенное количество 16-разрядных слов. Данные из Source Tag сохраняются в процессоре PLC-5, начиная с адреса, указанного в качестве Destination Element.

В следующей таблице показаны типы данных, которые должны использоваться с сообщениями типа PLC5 Typed Read и PLC5 Typed Write.

Таблица 3.2. Типы данных для сообщений PLC5 Typed Read и PLC5 Typed Write

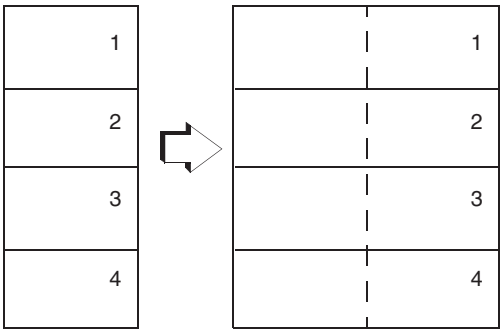
Для этого типа данных PLC-5:	Используйте этот тип данных Logix5000:
B	INT
F	REAL
N	INT
	DINT (Записывайте значения DINT в контроллер PLC-5, только если значение $\geqslant$ -32,768 и $\leqslant$ 32,767.)
S	INT
ST	STRING

Команды Typed Read и Typed Write также работают с процессорами SLC 5/03 (OS303 и выше), SLC 5/04 (OS402 и выше) и SLC 5/05.

Нижеследующие рисунки показывают, чем отличаются типовые инструкции и инструкции чтения/записи диапазона слов. В качестве примера используется инструкция чтения из процессора PLC-5 в контроллер Logix5000.

Типовая команда чтения

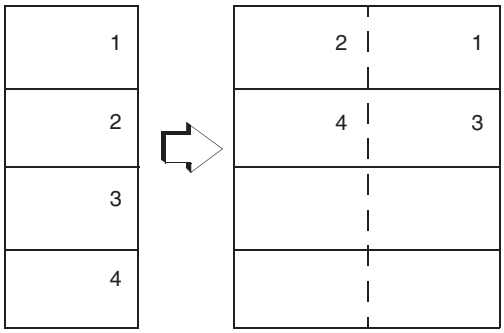
16-разрядные слова в процессоре PLC-5
32-разрядные слова в контроллере Logix5000



Типовые команды сохраняют структуру данных и значение.

Команда чтения диапазона слов

16-разрядные слова в процессоре PLC-5
32-разрядные слова в контроллере Logix5000



Команды, работающие с диапазоном слов, осуществляют последовательное заполнение целевого тега. Структура данных и значение изменяются в зависимости от целевого типа данных.

Задание сообщений SLC

Используйте различные типы сообщений SLC для обмена данными с контроллерами SLC и MicroLogix. В таблице показаны типы данных, к которым обеспечивает доступ эта инструкция, а также соответствующие типы данных Logix5000.

Для этого типа данных SLC или Micrologix:	Используйте этот тип данных Logix5000:
F	REAL
L (контроллеры Micrologix 1200 и 1500)	DINT
N	INT

Задание сообщений с поблочной передачей

Различные типы сообщений с поблочной передачей используются для обмена данными с модулями поблочной передачи по сети универсального дистанционного ввода/вывода (Universal Remote I/O).

Если вы хотите:	Выберите эту инструкцию:
считать данные из модуля поблочной передачи. Этот тип сообщения заменяет инструкцию BTR.	Block-Transfer Read
записать данные в модуль поблочной передачи. Этот тип сообщения заменяет инструкцию BTW.	Block-Transfer Write

Чтобы сконфигурировать сообщение с поблочной передачей, следуйте следующим инструкциям:

- Исходные теги (для BTW) и целевые теги (для BTR) должны иметь достаточно большой размер, позволяющий принять запрашиваемые данные (кроме структур MESSAGE, AXIS и MODULE).
- Задайте количество 16-разрядных целых чисел (INT) для отправки или получения. Вы можете задать от 0 до 64 целых чисел.

Если вы хотите, чтобы:	Задайте:
Модуль поблочной передачи определял количество отправляемых 16-разрядных целых чисел (BTR).	0 в качестве количества элементов
Контроллер отправлял 64 целых числа (BTW).	

Задание сообщений PLC-3

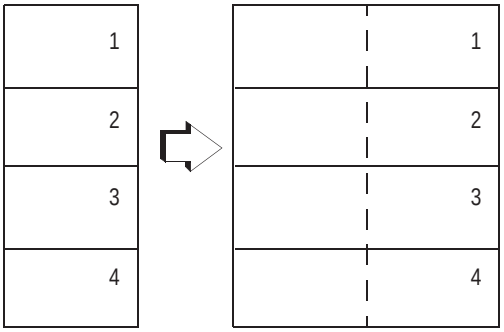
Различные типы сообщений PLC-3 предназначены для процессоров PLC-3.

Выберите эту команду:	Если вы хотите:
PLC3 Typed Read	считать целочисленные данные или данные типа REAL. В случае целых чисел эта команда считывает 16-разрядные целые числа из процессора PLC-3 и записывает их в массивы данных типа SINT, INT или DINT в контроллере Logix5000 с сохранением целостности данных. Также эта команда осуществляет чтение данных с плавающей точкой из PLC-3 и записывает их в тег с типом данных REAL в контроллере Logix5000.
PLC3 Typed Write	записать целочисленные данные или данные типа REAL. Эта команда записывает данные типа SINT или INT в целочисленный файл PLC-3 с сохранением целостности данных. Вы можете записать данные типа DINT только в том случае, если они вписываются в тип данных INT (-32768 *Љ* данные *J* 32767). Также эта команда осуществляет запись данных типа REAL из контроллера Logix5000 в файл данных с плавающей точкой PLC-3.
PLC3 Word Range Read	считать непрерывный ряд 16-разрядных слов из памяти PLC-3 независимо от типа данных. Эта команда начинает работать с адреса, указанного в качестве Source Element, и последовательно считывает запрошенное количество 16-разрядных слов. Данные из Source Element сохраняются начиная с адреса, указанного в качестве Destination Tag.
PLC3 Word Range Write	записать непрерывный ряд 16-разрядных слов из памяти Logix5000 в память PLC-3 независимо от типа данных. Эта команда начинает работать с адреса, указанного в качестве Source Tag, и последовательно считывает запрошенное количество 16-разрядных слов. Данные из Source Tag сохраняются в процессоре PLC-3, начиная с адреса, указанного в качестве Destination Element.

Нижеследующие рисунки показывают, чем отличаются типовые команды и команды чтения/записи диапазона слов. В качестве примера используется команда чтения из процессора PLC-3 в контроллер Logix5000.

Типовая команда чтения

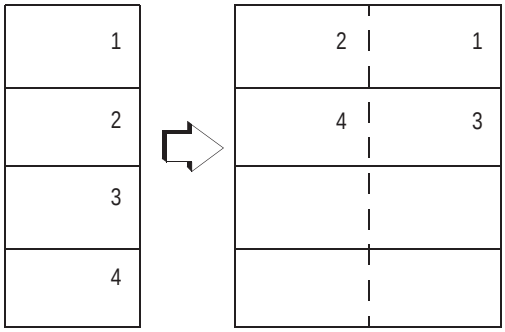
16-разрядные слова в процессоре PLC-3
32-разрядные слова в контроллере Logix5000



Типовые команды сохраняют структуру данных и значение

Команда чтения диапазона слов

16-разрядные слова в процессоре PLC-5
32-разрядные слова в контроллере Logix5000



Команды, работающие с диапазоном слов, осуществляют последовательное заполнение целевого тега. Структура данных и значение изменяются в зависимости от целевого типа данных.

Задание сообщений PLC-2

Различные типы сообщений PLC-2 предназначены для процессоров PLC-2.

Выберите эту команду:	**Если вы хотите:**
PLC2 Unprotected Read	считать 16-разрядные слова из любой области таблицы данных PLC-2 или файла совместимости с PLC-2 другого процессора.
PLC2 Unprotected Write	записать 16-разрядные слова в любую область таблицы данных PLC-2 или файла совместимости с PLC-2 другого процессора.

При передаче сообщения используются 16-разрядные слова, поэтому убедитесь в том, что Logix5000 правильно сохраняет передаваемые данные (обычно в виде массива INT).

Примеры конфигурации инструкции MSG

В следующих примерах представлены исходные и целевые теги, а также элементы для различных сочетаний контроллеров.

Для инструкций MSG, исходящих из контроллера Logix5000 и осуществляющих запись в другой контроллер:

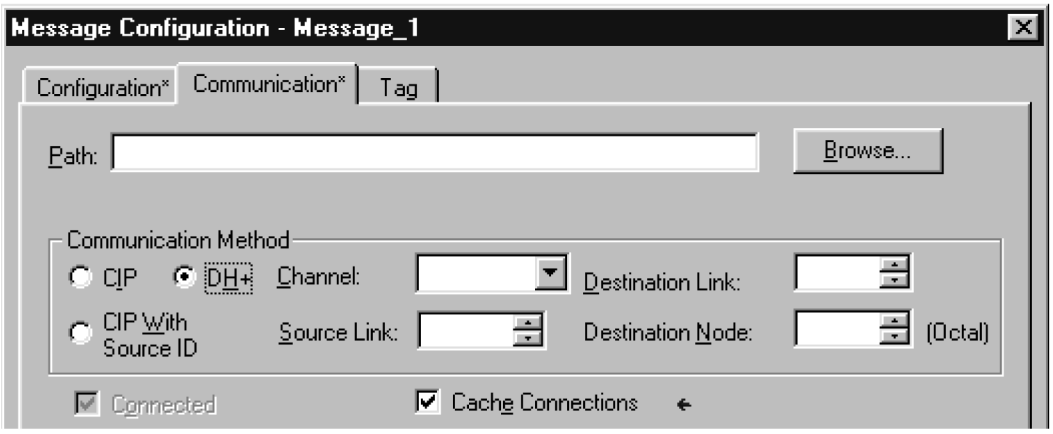
Путь сообщения:	Пример источника и приемника:	
Logix5000 → Logix5000	исходный тег	*array_1[0]*
	целевой тег	*array_2[0]*
	В качестве исходного тега вы можете использовать тег-псевдоним (в выдающем инструкцию контроллере Logix5000).	
	Нельзя использовать псевдоним для целевого тега. Приемником должен быть базовый тег.	
Logix5000 → PLC-5 Logix5000 → SLC	исходный тег	*array_1[0]*
	целевой элемент	*N7:10*
	Вы можете использовать тег-псевдоним в качестве исходного тега (в выдающем инструкцию контроллере Logix5000).	
Logix5000 → PLC-2	исходный тег	*array_1[0]*
	целевой элемент	*010*

Для инструкций MSG, исходящих из контроллера Logix5000 и осуществляющих чтение из другого контроллера:

Путь сообщения:	Пример источника и приемника:	
Logix5000 → Logix5000	исходный тег	*array_1[0]*
	целевой тег	*array_2[0]*
	Нельзя использовать псевдоним для исходного тега. Источником должен быть базовый тег.	
	Вы можете использовать тег-псевдоним в качестве целевого тега (в выдающем инструкцию контроллере Logix5000)	
Logix5000 → PLC-5 Logix5000 → SLC	исходный элемент	*N7:10*
	целевой тег	*array_1[0]*
	Вы можете использовать тег-псевдоним в качестве целевого тега (в выдающем инструкцию контроллере Logix5000).	
Logix5000 → PLC-2	исходный элемент	*010*
	целевой тег	*array_1[0]*

Задание деталей передачи данных

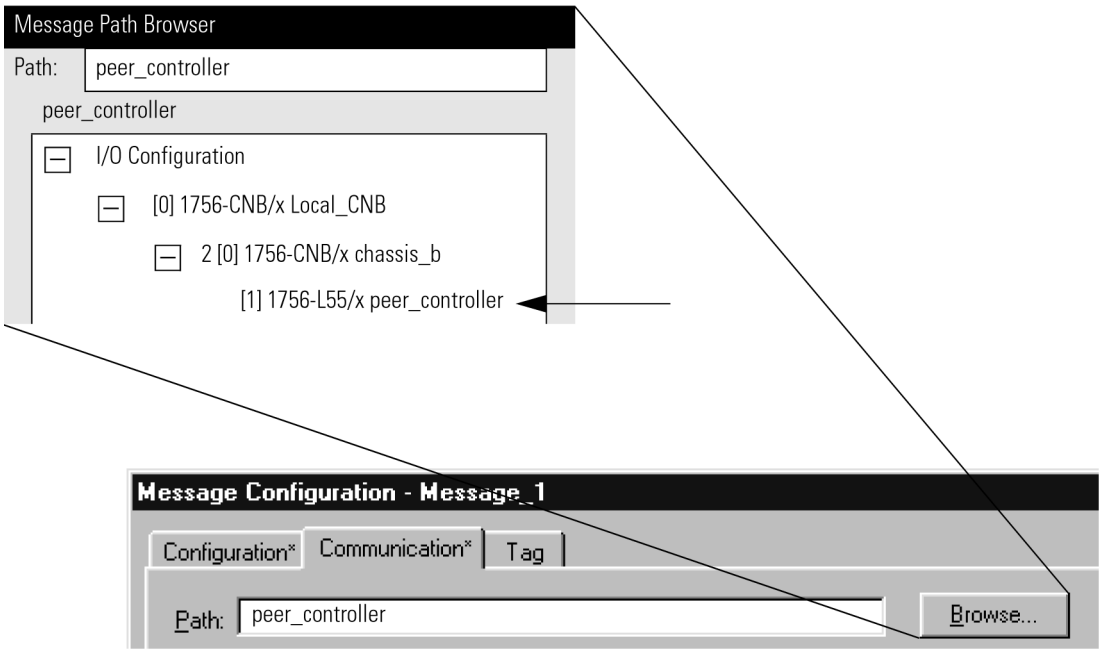
После того, как вы сконфигуруете инструкцию MSG, задайте следующие детали на вкладке Communication (Коммуникации).



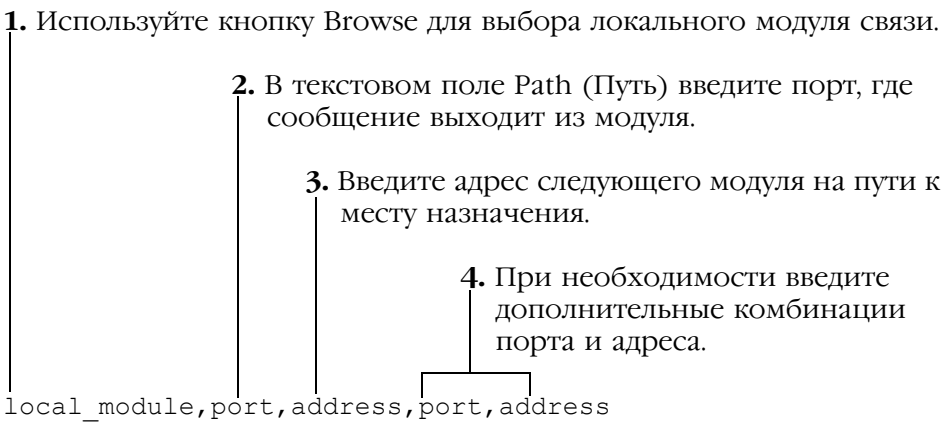
Задание пути

Путь представляет собой маршрут, по которому сообщение попадает к месту назначения.

- Если вы добавили локальный модуль связи, удаленный модуль связи, а также контроллер или устройство назначения в конфигурацию ввода/вывода контроллера, вы можете выбрать приемник сообщения с помощью кнопки Browse (Просмотр).



- Некоторые удаленные модули или устройства связи недоступны для конфигурации ввода/вывода контроллера. В таких ситуациях задайте путь следующим образом:

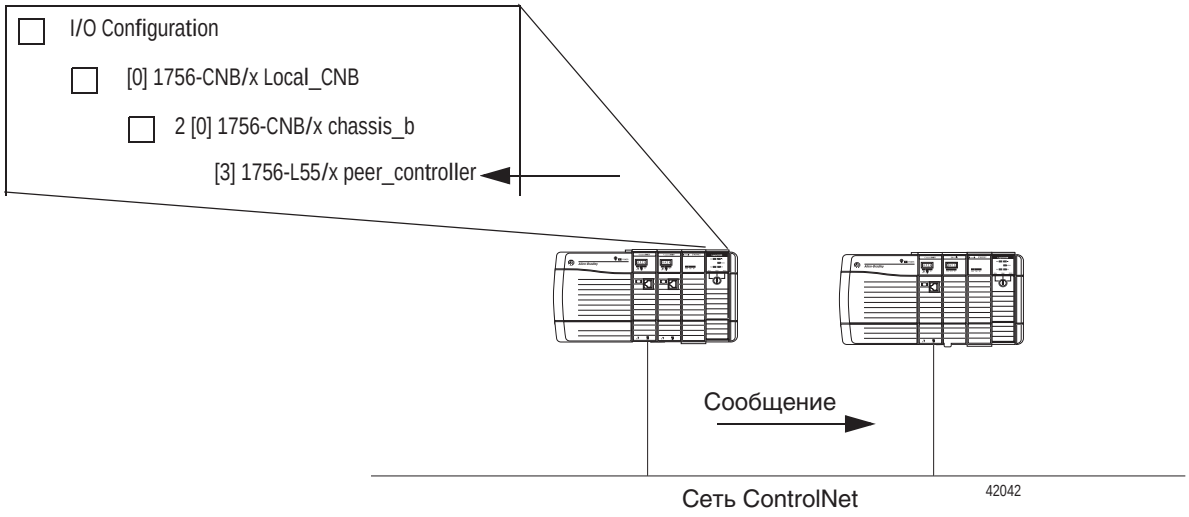


Где:	Для:	Это:
port	системной платы любого контроллера или модуля 1756	1
	порта DF1 из контроллера Logix5000	2
	порта ControlNet из модуля 1756-CNB	
	порта EtherNet из модуля 1756-ENBx или –ENET	
	порта DH+ канала A из модуля 1756-DHRIO	
	порта DH+ канала B из модуля 1756-DHRIO	3
address	системной платы ControlLogix	номер слота
	сети DF1	адрес станции (0-254)
	сети ControNet	номер узла (десятичное число в диапазоне 1-99)
	сети DH+	8#, за которым следует номер узла (восьмеричное число в диапазоне 1-77). Например, для задания восьмеричного адреса узла 37 введите 8#37.
	сети EtherNet/IP	Вы можете задать модуль в сети EtherNet/IP, используя один из следующих форматов: IP-адрес (например, 130.130.130.5) IP-адрес:Порт (например, 130.130.130.5:24) Имя DNS (например, tanks) Имя DNS:Порт (например, tanks:24)

- Для сообщений с поблочной передачей добавьте следующие модули к конфигурации ввода/вывода контроллера:

Для поблочной передачи по этой сети:	**Добавьте следующие модули к конфигурации вода/вывода:**
ControlNet	· локальный модуль связи (например, модуль 1756-CNB) · удаленный адаптерный модуль (например, модуль 1771-CAN)
универсальный дистанционный ввод/вывод	· локальный модуль связи (например, модуль 1756-DHRIO) · один удаленный адаптерный модуль (например, модуль 1771-ASB) для каждой стойки или части стойки в шасси · модуль поблочной передачи (не обязательно)
	Примеры пути приводятся на следующих страницах: · для ControlNet – стр. 3-27 · для EtherNet/IP – стр. 3-28 · для сообщения DH+ - стр. 3-29

ПРИМЕР Задание пути по ControlNet

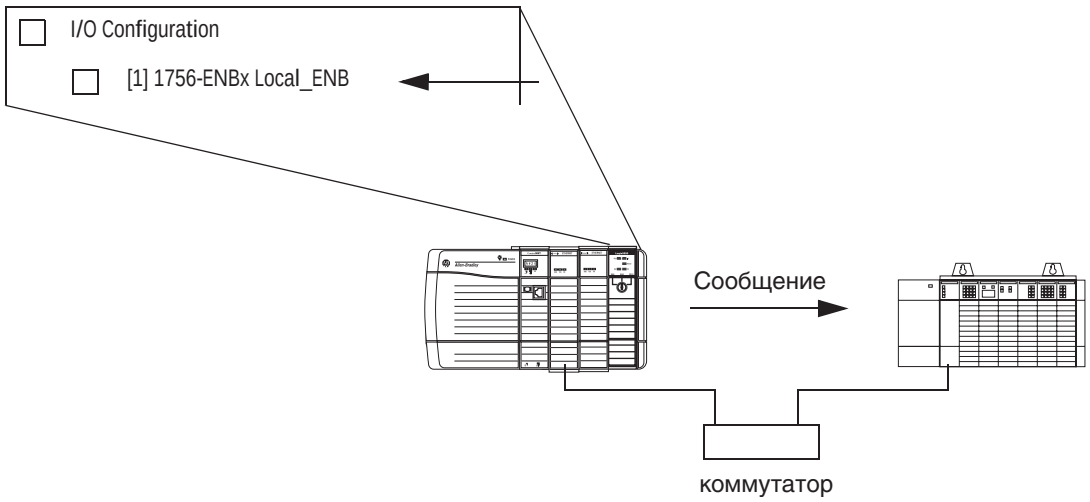


Путь: peer_controller

где:

peer_controller – имя контроллера, получающего сообщение.

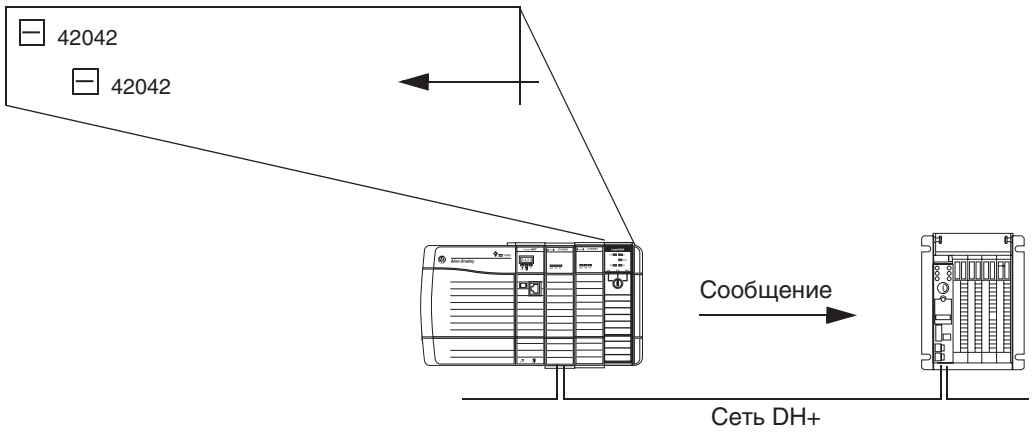
ПРИМЕР Задание пути по EtherNet/IP



Путь: Local_ENB,2,127.127.127.12

Где:	Это:
Local_ENB	имя модуля 1756-ENBx в локальном шасси
2	порт Ethernet модуля 1756-ENBx в локальном шасси
127.127.127.12	IP-адрес контроллера SLC 5/05

ПРИМЕР Задание пути по DH+



Путь: Local_DHRIO

где:

Local_DHRIO - это имя в модуле 1756-DHRIO, находящемся в том же шасси, что и отправляющий сообщение контроллер.

Задание способа передачи данных или адреса модуля:

Для выбора способа передачи данных или адреса модуля для сообщения используйте следующую таблицу.

Если целевым устройством является:	Выберите:	И задайте:	
Контроллер Logix5000	**CIP**	больше ничего задавать не требуется	
Контроллер PLC-5 по сети EtherNet/IP			
Контроллер PLC-5 по сети ControlNet			
Контроллер SLC 5/05			
Контроллер PLC-5 по сети DH+	**DH+**	Channel (Канал):	Канал A или B модуля1756-DHRIO, подключенный к сети DH+
Контроллер SLC по сети DH+		Source Link (Канал связи с источником):	ID канала связи, назначенный объединительной плате контроллера в маршрутной таблице модуля 1756-DHRIO. (Исходным узлом в маршрутной таблице автоматически является номер слота контроллера.)
Процессор PLC-3		Destination Link (Канал связи с приемником):	ID канала связи для удаленной сети DH+, в которой находится целевое устройство
Процессор PLC-2		Destination Node (Целевой узел):	Адрес станции целевого устройства в восьмеричном представлении
		Если существует лишь один канал связи с DH+ и вы не использовали программное обеспечение RSLinx для конфигурирования модуля DH/RIO для связи с удаленными устройствами, задайте 0 как для Source Link, так и для Destination Link.	
Приложение на рабочей станции, получающее незатребованное сообщение, направленное через сеть EtherNet/IP или ControlNet посредством RSLinx	**CIP with Source ID** (Это позволяет приложению получать данные из контроллера.)	Source Link:	Удаленный ID раздела в программном обеспечении RSLinx
		Destination Link:	ID виртуального канала связи, заданный в RSLinx (0-65535)
		Destination Node:	ID приемника (восьмеричное число в диапазоне 0-77), переданный приложением в RSLinx. Для раздела DDE в RSLinx используйте 77.
		В качестве исходного узла (Source Node) используется номер слота контроллера ControlLogix.	
Модуль поблочной передачи по универсальной удаленной сети ввода/вывода	RIO	Channel (Канал):	Канал A или B модуля1756-DHRIO, подключенный к универсальной удаленной сети ввода/вывода.
		Rack (Стойка):	Номер стойки модуля (восьмеричный)
		Group (Группа):	Номер группы модуля
		Slot (Слот):	Номер слота, в котором находится модуль
Модуль поблочной передачи по сети ControlNet	ControlNet	Slot (Слот):	Номер слота, в котором находится модуль

Выбор опции кэширования

В зависимости от того, как вы сконфигурировали инструкцию MSG, она может использовать соединение для отправки или получения данных.

Этот тип сообщения:	И этот способ передачи данных:	Использует соединение:
CIP Data Table Read и Write	→	✓
PLC2, PLC3, PLC5 и SLC (все типы)	CIP	
	CIP with Source ID	
	DH+	✓
CIP Generic	→	на ваш выбор (1)
Block-transfer Read и Write	→	✓

(1) Вы можете создавать соединения для сообщений типа CIP Generic. Однако для большинства приложений мы рекомендуем не использовать соединения для сообщений CIP Generic.

Если инструкция MSG использует соединение, вы имеете возможность оставить соединение открытым (кэшировать его) или закрыть соединение после того, как сообщение завершит передачу данных.

Если вы:	То:
Кэшируете соединение	Соединение остается открытым после выполнения инструкции MSG. Это позволяет оптимизировать время выполнения. Открытие соединения при каждом выполнении инструкции MSG увеличивает время работы.
Не кэшируете соединение	Соединение закрывается после выполнения инструкции MSG. Это освобождает соединение для использования в других целях.

В контроллере имеются следующие ограничения по количеству соединений, которые можно кэшировать:

Если у вас следующая версия программного и микропрограммного обеспечения:	То вы можете кэшировать:
11.x и ниже	· сообщения с поблочной передачей – до 16 соединений · другие типы сообщений – до 16 соединений
12.x и выше	до 32 соединений

Если в одно и то же устройство отправляется несколько сообщений, то такие сообщения могут совместно использовать соединение.

Если инструкции MSG передают данные:	**И они:**	**То:**
в различные устройства	→	Каждая инструкция MSG использует одно соединение
в одно и то же устройство	разрешены одновременно	Каждая инструкция MSG использует одно соединение
	разрешены НЕ одновременно	Инструкция MSG совместно используют соединение (т.е. вместе они считаются за 1 соединение.)

ПРИМЕР

Совместное использование соединения
Если контроллер попеременно отправляет в один и тот же модуль сообщение чтения с поблочной передачей и сообщение записи с поблочной передачей, то эти два сообщения вместе считаются за 1 соединение. Кэширование обоих сообщений засчитывается за 1 в списке кэша.

Методические рекомендации

При планировании и программировании инструкций MSG придерживайтесь следующих рекомендаций:

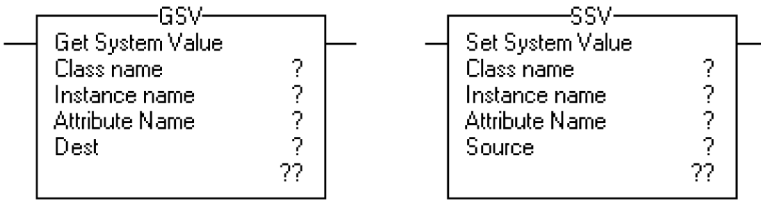
Рекомендация:	**Подробное описание:**
1. Для каждой инструкции MSG создайте управляющий тег.	Для каждой инструкции MSG требуется собственный управляющий тег. · Data type (тип данных) = MESSAGE · Scope (область действия) = controller · Этот тег *не может* входить в массив или иметь определенный пользователем тип данных
2. Храните исходные и/или целевые данные в области действия контроллера.	Инструкция MSG может обращаться лишь к тегам, находящимся в папке Controller Tags (область действия контроллера).
3. Если ваша инструкция MSG предназначена для устройства, использующего 16-разрядные целые числа, используйте буфер INT в MSG и DINT во всем проекте.	Если ваше сообщение предназначается устройству, использующему 16-разрядные целые числа, такому как контроллер PLC-5® или SLC 500‘, и оно передает целочисленные данные (не REAL), используйте буфер из INT в сообщении и DINT во всем проекте. Это повысит эффективность вашего проекта, так как контроллеры Logix имеют более высокую производительность и используют меньше памяти при работе с 32-разрядными целыми числами (DINT). Преобразование INT в DINT см. *Общие процедуры контроллеров Logix5000*, публикация 1756-PM001.
4. Кэшируйте наиболее часто выполняемые инструкции MSG с соединением.	Кэшируйте соединения для наиболее часто выполняемых инструкций MSG до максимально допустимого для вашей версии контроллера количества. Это позволяет оптимизировать время выполнения за счет того, что контроллеру не придется открывать соединение при каждом выполнении сообщения.
5. Если вы хотите разрешить одновременно более 16 инструкций MSG, используйте определенную стратегию управления.	Если вы одновременно разрешаете более 16 инструкций MSG, некоторые из них могут попадать в очередь с задержкой. Чтобы гарантировать выполнение каждого сообщения, используйте один из следующих вариантов действий: · разрешайте сообщения одно за другим. · разрешайте сообщения группами. · Запрограммируйте сообщение таким образом, чтобы происходил обмен данными с несколькими устройствами. Дополнительную информацию см. *Общие процедуры контроллеров Logix5000*, публикация 1756-PM001. · Запрограммируйте логику таким образом, чтобы она координировала выполнение сообщений. Дополнительную информацию см. *Общие процедуры контроллеров Logix5000*, публикация 1756-PM001.
6. Следите за тем, чтобы количество инструкций MSG без соединения плюс количество некэшированных инструкций MSG было меньше количества буферов без соединения.	Контроллер может иметь от 10 до 40 буферов без соединения. Значение по умолчанию – 10. · Если все буферы без соединения заняты в тот момент, когда инструкция покидает очередь сообщений, инструкция выдает ошибку и не осуществляет передачу данных. · Вы можете увеличить количество буферов без соединения (до 40 максимум), но при этом выполняйте рекомендацию 5. · Информацию по увеличению количества буферов без соединения см. *Общие процедуры контроллеров Logix5000*, публикация 1756-PM001.

Get System Value (GSV) (Получить системное значение) и Set System Value (SSV) (Установить системное значение)

Инструкции GSV/SSV получают и устанавливают системные данные, хранимые в объектах.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Class name		имя	имя объекта
Instance name		имя	имя конкретного объекта, когда для объекта требуется имя
Attribute name		имя	атрибут объекта тип данных зависит от выбранного вами атрибута
Destination (GSV)	SINT INT DINT REAL	тег	приемник данных по атрибуту
Source (SSV)	SINT INT DINT REAL	тег	тег, содержащий данные, которые вы хотите скопировать в атрибут

Структурированный текст

```
GSV(ClassName,InstanceName,AttributeName,Dest);
SSV(ClassName,InstanceName,AttributeName,Source);
```

Операнды совпадают с операндами инструкций GSV и SSV для релейной логики.

Описание:

Инструкции GSV/SSV получают и устанавливают системные данные контроллера, хранимые в объектах. Контроллер хранит системные данные в объектах. Здесь нет файла состояния как в процессоре PLC-5.

Когда инструкция GSV разрешена, она находит указанную информацию и помещает ее в место назначения. Когда инструкция SSV разрешена, она устанавливает указанный атрибут, используя данные из источника.

При вводе команды GSV/SSV, программное обеспечение програмирования выводит на экран допустимые классы объектов, имена объектов и имена атрибутов для каждой из этих команд. В случае команды GSV вы можете получить значения для всех имеющихся атрибутов. В случае команды SSV программное обеспечение отображает лишь те атрибуты, которые можно задать.

ВНИМАНИЕ

Используйте команды GSV/SSV с осторожностью. Внесение изменений в объекты может привести к непредвиденной работе контроллера или причинению травм персоналу.

Если размер источника (Source) или приемника (Destination) слишком мал, то команда не будет выполняться и будет зарегистрирована неосновная ошибка. В следующем разделе «Объекты GSV/SSV» указываются атрибуты каждого объекта и соответствующие им типы данных. Например, для атрибута MajorFaultRecord объекта Program требуется тип данных DINT[11].

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

Неосновная ошибка произойдет в случае:	Тип ошибки:	Код ошибки:
неверного адреса объекта	4	5
задания объекта, не поддерживающего GSV/SSV	4	6
неверного атрибута	4	6
недостаточной информации для команды SSV	4	6
недостаточного размера приемника для размещения требуемых данных для команды GSV	4	7

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Ничего не происходит
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки «истина»	Команда выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn установлен	не применимо	EnableIn всегда установлен. Команда выполняется.
команда выполняется	Получение и задание указанного значения.	Получение или задание указанного значения.
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Ничего не происходит.

Объекты GSV/SSV

При вводе инструкции GSV/SSV вы задаете объект и его атрибут, к которому вы хотите обратиться. В некоторых случаях может быть несколько объектов одного типа, тогда надо будет дополнительно указать имя объекта. Например, в вашем приложении может быть несколько задач. Каждой задаче соответствует свой объект TASK, к которому вы можете обратиться по имени задачи.

ВНИМАНИЕ

В случае инструкции GSV в приемник копируется лишь заданный объем данных. Например, если атрибут задан как SINT, а приемник – DINT, то обновятся лишь младшие 8 битов приемника DINT, а остальные 24 бита не изменятся.

Вы можете обращаться к следующим объектам:

За информацией по этому объекту:	Обращайтесь к следующей странице данного документа:
AXIS	*ControlLogix Motion Module Setup and Configuration Manual, publication 1756-UM006*
CONTROLLER	3-37
CONTROLLERDEVICE	3-37
CST	3-39
DF1	3-40
FAULTLOG	3-43
MESSAGE	3-44
MODULE	3-46
MOTIONGROUP	3-47
PROGRAM	3-48
ROUTINE	3-49
SERIALPORT	3-49
TASK	3-51
WALLCLOCKTIME	3-53

Обращение к объекту CONTROLLER

Объект CONTROLLER предоставляет информацию о состоянии работы контроллера.

Атрибут:	Тип данных:	Инструкция:	Описание:
TimeSlice	INT	GSV SSV	Процент CPU, выделенный для обмена данными. Допустимыми являются значения от 10 до 90. Это значение нельзя изменять, когда кнопочный переключатель контроллера находится в рабочем положении.

Обращение к объекту CONTROLLERDEVICE

Объект CONTROLLERDEVICE идентифицирует физическое оборудование контроллера.

Атрибут:	Тип данных:	Инструкция:	Описание:	
DeviceName	SINT[33]	GSV	Строка ASCII, идентифицирующая каталожный номер контроллера и платы памяти. Первый байт представляет собой счетчик количества символов ASCII, возвращенных в строке массива.	
ProductCode	INT	GSV	Идентифицирует тип контроллера	
			Контроллер Logix:	**Код продукта:**
			CompactLogix5320	43
			CompactLogix5330	44
			CompactLogix5335E	65
			ControlLogix5550	3
			ControlLogix5553	50
			ControlLogix5555	51
			ControlLogix5561	54
			ControlLogix5562	55
			ControlLogix5563	56
			DriveLogix5720	48
			FlexLogix5433	41
			FlexLogix5434	42
			SoftLogix5860	15
ProductRev	INT	GSV	Идентифицирует текущую ревизию продукта. Должен отображаться в шестнадцатеричном виде. Младший байт соответствует основной ревизии; старший байт – неосновной ревизии.	

Атрибут:	Тип данных:	Инструкция:	Описание:
SerialNumber	DINT	GSV	Серийный номер устройства. Серийный номер присваивается при изготовлении устройства.
Status	INT	GSV	Биты, указывающие на состояние: Биты 3-0 зарезервированы **Биты состояния устройства** **Биты 7-4: Означает:** 0000 в резерве 0001 идет обновление флэш-памяти 0010 в резерве 0011 в резерве 0100 неисправная флэш-память 0101 ошибка 0110 выполнение 0111 программа **Биты состояния ошибки** **Биты 11-8: означает:** 0001 исправимая неосновная ошибка 0010 неисправимая неосновная ошибка 0100 исправимая основная ошибка 1000 неисправимая основная ошибка **Биты состояния самого Logix5000** **Биты 13-12: Означает:** 01 переключатель в рабочем положении 10 переключатель в положении программирования 11 переключатель в положении дистанционной работы **Биты 15-14 Означает:** 01 контроллер меняет режим 10 режим отладки, если контроллер находится в рабочем режиме
Type	INT	GSV	Идентифицирует устройство как контроллер. Контроллер = 14
Vendor	INT	GSV	Идентифицирует поставщика устройства. Allen-Bradley = 0001

Обращение к объекту CST

Объект CST (coordinated system time - согласованное системное время) предоставляет согласованное системное время для устройств, находящихся в одном шасси.

Атрибут:	Тип данных:	Инструкция:	Описание:	
CurrentStatus	INT	GSV	Текущее состояние согласованного системного времени. Биты указывают на следующее:	
			Бит	Означает:
			0	сбой аппаратуры таймера: аппаратура встроенного таймера устройства находится в неисправном состоянии
			1	разрешено линейное увеличение: текущее значение младших 16+ битов таймера линейно увеличивается до требуемого значения, а не мгновенно принимает более низкое значение Манипуляции с этими битами производятся с помощью метода тактовой синхронизации, определяемого конкретной сетью.
			2	задатчик системного времени: объект CST является источником задающего времени в системе ControlLogix
			3	синхронизирован: 64-разрядный атрибут CurrentValue объекта CST синхронизируется главным объектом CST посредством обновления системного времени
			4	задатчик времени локальной сети: объект CST является источником задающего времени для локальной сети
			5	в режиме реле: объект CST работает в режиме реле времени
			6	обнаружен дубликат задатчика времени: обнаружен дубликат задатчика времени локальной сети. Для зависящих от времени узлов этот бит всегда 0.
			7	Не используется
			8-9	00 = зависящий от времени узел 01 = узел задатчика времени 10 = узел реле времени 11 = не используется
			10-15	не используются
CurrentValue	DINT[2]	GSV	Текущее значение таймера. DINT[0] содержит младшие 32 бита, а DINT[1] – старшие 32 бита. Источник таймера корректируется для приведения его в соответствие со значением, переданным сервисами обновления и от синхронизации локальной сети связи. Такой корректировкой является либо линейное увеличение до требуемого значения, либо мгновенная установка требуемого значения, как описывается для атрибута CurrentStatus.	

Обращение к объекту DF1

Объект DF1 обеспечивает интерфейс с драйвером связи DF1, который вы можете сконфигурировать для последовательного порта

Атрибут:	Тип данных:	Инструкция:	Описание:
ACKTimeout	DINT	GSV	Время ожидания подтверждения передачи сообщения (только для двухточечной связи и главного устройства). Допустимые значения находятся в диапазоне от 0 до 32767. Задержка измеряется периодами по 20 мс. Значение по умолчанию – 50 (1 с).
DiagnosticCounters	INT[19]	GSV	Массив диагностических счетчиков для драйвера связи DF1.

смещение слова	двухточечный DF1	подчиненный DF1	главный
0	сигнатура (0x0043)	сигнатура (0x0042)	сигнатура (0x0044)
1	биты модема	биты модема	биты модема
2	отправленные пакеты	отправленные пакеты	отправленные пакеты
3	полученные пакеты	полученные пакеты	полученные пакеты
4	не доставленные пакеты	не доставленные пакеты	не доставленные пакеты
5	не используется	повторные сообщения	повторные сообщения
6	полученные NAK	полученные NAK	не используется
7	полученные ENQ	полученные пакеты опроса	не используется
8	неверные не подтвержденные пакеты	неверные не подтвержденные пакеты	неверные не подтвержденные пакеты
9	не подтвержденное сообщение по отсутствию памяти	не подтвержденное сообщение по отсутствию памяти	не используется
10	полученные дублированные пакеты	полученные дублированные пакеты	полученные дублированные пакеты
11	полученные неверные символы	не используется	не используется
12	количество восстановлений DCD	количество восстановлений DCD	количество восстановлений DCD
13	количество потерянных модемов	количество потерянных модемов	количество потерянных модемов
14	не используется	не используется	максимальное время приоритетного сканирования
15	не используется	не используется	последнее время приоритетного сканирования
16	не используется	не используется	максимальное время нормального сканирования
17	не используется	не используется	последнее время нормального сканирования
18	отправленные ENQ	не используется	не используется

DuplicateDetection	SINT	GSV	разрешает обнаружение дублированных сообщений.

Значение:	Означает:
0	обнаружение дублированных сообщений запрещено
ненулевое	обнаружение дублированных сообщений разрешено

EmbeddedResponseEnable	SINT	GSV	разрешение функции встроенного ответа (только для двухточечной связи)

Значение:	Означает:
0	инициируется только после получения единицы (по умолчанию)
1	разрешена безусловно

ENQTransmitLimit	SINT	GSV	Количество запросов (ENQ), которые должны быть отправлены по тайм-ауту подтверждения (ACK) (только для двухточечной связи). Допустимыми являются значения от 0 до 127. Настройка по умолчанию – 3.

EOTSuppression	SINT	GSV	Разрешение подавления передач EOT в ответ на пакеты опросов (только для подчиненных устройств). **Значение:** **Означает:** 0 подавление EOT запрещено ненулевое подавление EOT разрешено
ErrorDetection	SINT	GSV	Задает схему обнаружения ошибок. **Значение:** **Означает:** 0 BCC (по умолчанию) 1 CRC
MasterMessageTransmit	SINT	GSV	Текущее значение передачи главного сообщения (только для главного устройства) **Значение:** **Означает:** 0 между опросами станций 1 в процессе опроса (вместо номера станции главного устройства) Значение по умолчанию 0.
NAKReceivedLimit	SINT	GSV	Количество NAK, полученных в ответ на сообщение до прекращения передачи данных (только для двухточечной связи). Допустимыми являются значения от 0 до 127. Значение по умолчанию 0.
NormalPollGroupSize	INT	GSV	Количество станций, которые должны опрашиваться в массиве узла нормального опроса после опроса всех станций в массиве узла приоритетного опроса (только для главного устройства). Допустимыми являются значения от 0 до 255. Значение по умолчанию 0.
PollingMode	SINT	GVS	Текущий режим опроса (только для главного устройства). **Значение:** **Означает:** 0 основан на обмене сообщениями, но не позволяет подчиненным устройствам инициировать сообщения 1 основан на обмене сообщениями, но позволяет подчиненным устройствам инициировать сообщения (по умолчанию) 2 стандартный, передача одного сообщения при одном сканировании узла 3 стандартный, передача нескольких сообщений при одном сканировании узла Значение по умолчанию 1.
ReplyMessageWait	DINT	GSV	Время (действующее как задающее) ожидания после получения ACK до опроса подчиненного устройства (только для главного устройства). Допустимыми являются значения от 0 до 65535. Задержка измеряется периодами по 20 мс. Значение по умолчанию – 5 периодов (100 мс).
StationAddress	INT	GSV	Текущий адрес станции последовательного порта. Допустимыми являются значения от 0 до 254. Значение по умолчанию 0.

SlavePollTimeout	DINT	GSV	Время в мс, которое подчиненное устройство ждет опроса от главного, после чего объявляет о невозможности передачи данных из-за того, что главное устройство неактивно (только для подчиненного устройства). Допустимыми являются значения от 0 до 32767. Задержка измеряется периодами по 20 мс. Значение по умолчанию 3000 периодов (1 минута).
TransmitRetries	SINT	GSV	Количество повторных отправок сообщения без подтверждения его получения (только для главного и подчиненного устройства). Допустимыми являются значения от 0 до 127. Значение по умолчанию 3.
PendingACKTimeout	DINT	SSV	Ожидание значения для атрибута ACKTimeout.
PendingDuplicateDetection	SINT	SSV	Ожидание значения для атрибута DuplicateDetection.
PendingEmbedded-ResponseEnable	SINT	SSV	Ожидание значения для атрибута EmbeddedResponse.
PendingENQTransmitLimit	SINT	SSV	Ожидание значения для атрибута ENQTransmitLimit.
PendingEOTSuppression	SINT	SSV	Ожидание значения для атрибута EOTSuppression.
PendingErrorDetection	SINT	SSV	Ожидание значения для атрибута ErrorDetection.
PendingNormalPollGroupSize	INT	SSV	Ожидание значения для атрибута NormalPollGroupSize.
PendingMasterMessage-Transmit	SINT	SSV	Ожидание значения для атрибута MasterMessageTransmit.
PendingNAKReceiveLimit	SINT	SSV	Ожидание значения для атрибута NAKReceivedLimit.
PendingPollingMode	SINT	SSV	Ожидание значения для атрибута PollingMode.
PendingReplyMessageWait	DINT	SSV	Ожидание значения для атрибута ReplyMessageWait.
PendingStationAddress	INT	SSV	Ожидание значения для атрибута StationAddress.
PendingSlavePollTimeout	DINT	SSV	Ожидание значения для атрибута SlavePollTimeout.
PendingTransmitRetries	SINT	SSV	Ожидание значения для атрибута TransmitRetries.

Для применения значений любого из находящихся в ожидании атрибутов DF1:

1. Используйте инструкцию SSV для задания значения находящегося в ожидании атрибута.

   Вы можете задать значения для любого количества ожидающих атрибутов, используя инструкцию SSV для каждого находящегося в ожидании атрибута.

2. Используйте инструкцию MSG для использования заданного значения. Инструкция MSG применяет все заданные вами ожидающие атрибуты. Сконфигурируйте инструкцию MSG следующим образом:

Вкладка MSG Configuration:	**Поле:**	**Значение:**
Configuration (Конфигурация)	Message Type (Тип сообщения)	CIP Generic
	Service Code (Код сервиса)	Od hex
	Object Type	a2
	Object ID (ID объекта)	1
	Object Attribute (Атрибут объекта)	оставить пустым
	Source (Источник)	оставить пустым
	Number of Elements (Количество элементов)	0
	Destination (Приемник)	оставить пустым
Communication (Связь)	Path (Путь)	путь передачи данных к самому себе (1,s где s – номер слота для контроллера)

Обращение к объекту FAULTLOG

Объект FAULTLOG предоставляет информацию о сбоях контроллера.

Атрибут:	Тип данных:	Инструкция:	Описание:
MajorEvents	INT	GSV	Количество основных сбоев с момента последнего сброса данного счетчика.
MinorEvents	INT	GSV SSV	Количество неосновных сбоев с момента последнего сброса данного счетчика.
MajorFaultBits	DINT	GSV SSV	Биты указывают на причину текущего основного сбоя. **Бит:** **Означает:** 1 отключение питания 3 ввод/вывод 4 выполнение инструкции (программа) 5 обработчик ошибок 6 сторожевой таймер 7 стек 8 изменение режима 11 перемещение
MinorFaultBits	DINT	GSV SSV	Биты указывают на причину текущего неосновного сбоя. **Бит:** **Означает:** 4 выполнение инструкции (программа) 6 сторожевой таймер 9 последовательный порт 10 батарея

Обращение к объекту MESSAGE

Вы можете обратиться к объекту MESSAGE с помощью инструкций GSV/SSV. Укажите имя тега сообщения, чтобы определить нужный вам объект MESSAGE. Объект MESSAGE обеспечивает интерфейс для организации и запуска одноранговой передачи данных. Этот объект заменяет тип данных MG процессора PLC-5.

Атрибут:	Тип данных:	Инструкция:	Описание:	
ConnectionPath	SINT[130]	GSV SSV	Данные для задания пути соединения. Первые два байта (младший и старший) соответствуют длине пути соединения в байтах.	
ConnectionRate	DINT	GSV SSV	Требуемая скорость передачи пакетов для данного соединения.	
MessageType	SINT	GSV SSV	Задает тип сообщения.	
			Значение:	**Означает:**
			0	не инициализировано
Port	SINT	GSV SSV	Указывает порт, в который должно быть направлено сообщение.	
			Значение:	**Означает:**
			1	объединительная плата
			2	последовательный порт
TimeoutMultiplier	SINT	GSV SSV	Определяет, кода соединение должно быть закрыто по превышению времени ожидания.	
			Значение:	**Означает:**
			0	соединение будет закрываться по тайм-ауту через время, равное 4-кратному периоду обновления (по умолчанию)
			1	соединение будет закрываться по тайм-ауту через время, равное 8-кратному периоду обновления
			2	соединение будет закрываться по тайм-ауту через время, равное 16-кратному периоду обновления
UnconnectedTimeout	DINT	GSV SSV	Тайм-аут в микросекундах для всех сообщений без соединения. Значение по умолчанию 30000000 микросекунд (30 секунд).	

Значение какого-либо атрибута объекта MESSAGE можно изменить следующим образом:

1. Используйте инструкцию GSV для получения атрибута MessageType и его сохранения в теге.

2. Чтобы установить атрибут MessageType на 0, используйте инструкцию SSV.

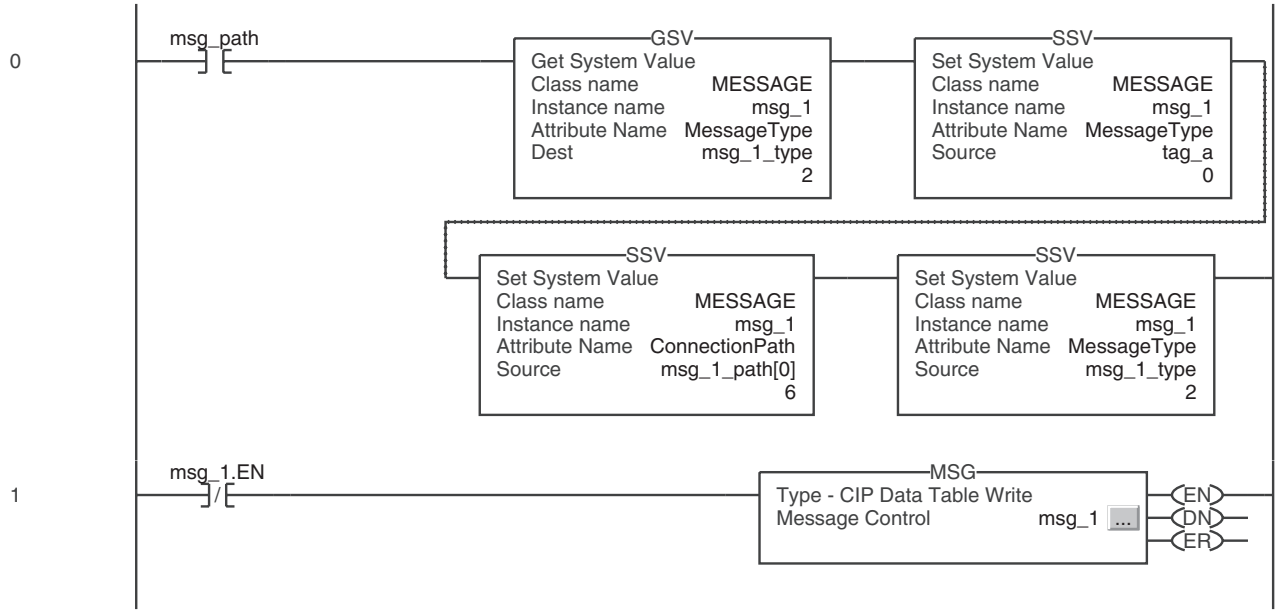
3. Используйте инструкцию SSV для задания атрибута объекта MESSAGE, который вы хотите изменить.

4. Для установки атрибута MessageType на его первоначальное значение, полученное в п.1, используйте инструкцию SSV.

Пример: В следующем примере изменяется атрибут ConnectionPath таким образом, что сообщение направляется в другой контроллер. Когда *msg_path* разрешен, путь для сообщения *msg_1* устанавливается на значение *msg_1_path*. Это направляет сообщение в другой контроллер.

Где:	Это:
msg_1	Сообщение, атрибуты которого вы хотите изменить
msg_1_type	Тег, в котором хранится значение атрибута MessageType
tag_a	Тег, в котором хранится 0.
msg_1_path	Тег массива, в котором хранится новый путь соединения для данного сообщения

Релейная логика



Структурированный текст

```
IF  msg_path  THEN
     GSV(MESSAGE,msg_1,MessageType,msg_1_type);
     SSV(MESSAGE,msg_1,MessageType,tag_a);
     SSV(MESSAGE,msg_1,ConnectionPath,msg_1_path[0]);
     SSV(MESSAGE,msg_1,MessageType,msg_1_type);
END_IF;
IF  NOT  msg_1.EN  THEN
     MSG(msg_1);
END_IF;
```

Обращение к объекту MODULE

Объект MODULE предоставляет информацию о состоянии модуля. Чтобы выбрать конкретный объект MODULE, установите операнд Object Name (Имя объекта) инструкции GSV/SSV на имя модуля. При этом указанный модуль должен присутствовать в разделе I/O Configuration (Конфигурация ввода/вывода) организатора контроллера и должен иметь имя устройства.

Атрибут:	Тип данных:	Инструкция:	Описание:	
EntryStatus	INT	GSV	Указывает текущее состояние записи таблицы соответствий. Младшие 12 битов должны быть маскированы при выполнении операции сравнения. Действительны только биты 12-15.	
			Значение:	**Означает:**
			16#0000	Режим простоя: контроллер подключается к источнику питания.
			16#1000	Сбой: все соединения объекта MODULE с соответствующим модулем дают сбой. Это значение не должно использоваться для определения того, связан ли отказ модуля с тем, что объект MODULE периодически выходит из этого состояния, вновь пытаясь соединиться с модулем. Вместо этого проверьте наличие рабочего состояния (16#4000). Поверьте FaultCode на неравенство нулю, чтобы определить, имеется ли неисправность модуля. Если имеется неисправность, то атрибуты FaultCode и FaultInfo сохраняют свои значения до устранения неисправности.
			16#2000	**Проверка**: объект MODULE проверяет целостность объекта MODULE перед установлением соединений с модулем.
			16#3000	**Соединение**: объект MODULE инициирует соединения с модулем.
			16#4000	**Работа**: все соединения с модулем установлены, и данные успешно передаются.
			16#5000	**Отключение**: объект MODULE запрещает все соединения с модулем.
			16#6000	**Запрещение**: объект MODULE запрещен (в атрибуте Mode установлен бит запрета)
			16#7000	**Ожидание**: родительский объект MODULE, от которого зависит данный объект MODULE, не работает
FaultCode	INT	GSV	Число, идентифицирующее неисправность модуля, если она имеет место.	
FaultInfo	DINT	GSV	Содержит конкретную информацию о коде ошибки объекта MODULE.	
ForceStatus	INT	GSV	Указывает состояние форсировки.	
			Бит:	**Значение:**
			0	форсировка установлена (1=да, 0=нет)
			1	форсировка разрешена (1=да, 0=нет)
			2-15	не используется

Атрибут:	Тип данных:	Инструкция:	Описание:
Instance	DINT	GSV	Указывает номер экземпляра для объекта MODULE.
LEDStatus	INT	GSV	Указывает на текущее состояние светодиодного индикатора ввода/вывода, находящегося на передней панели контроллера **Значение:** **Означает:** 0 **Индикатор выключен**: Для данного контроллера не сконфигурировано ни одного объекта MODULE (нет никаких модулей в разделе I/O Configuration организатора контроллера). 1 Мигает красным: Ни один из объектов MODULE не работает. 2 Мигает зеленым: Хотя бы один объект MODULE не работает. 3 Постоянный зеленый: Все объекты MODULE работают. **Примечание**: Для этого атрибута не нужно вводить имя объекта, так как он относится ко всей совокупности модулей.
Mode	INT	GSV SSV	Указывает на текущий режим объекта MODULE. **Бит:** **Означает:** 0 Если этот бит установлен, генерируется основная ошибка в том случае, когда какие-либо из соединений объекта MODULE дают сбой, в то время как контроллер находится в рабочем режиме (Run mode). 2 Если этот бит установлен, объект MODULE переходит в состояние запрета после закрытия всех соединений с данным модулем.

Обращение к объекту MOTIONGROUP

Объект MOTIONGROUP предоставляет информацию о состоянии группы осей для сервомодуля. Чтобы определить нужный вам объект MOTIONGROUP, задайте имя тега группы перемещения.

Атрибут:	Тип данных:	Инструкция:	Описание:
Instance	DINT	GSV	Указывает номер экземпляра для данного объекта MOTION_GROUP.

Обращение к объекту PROGRAM

Объект PROGRAM предоставляет информацию о состоянии программы. Чтобы определить нужный вам объект PROGRAM, задайте имя программы.

Атрибут:	Тип данных:	Инструкция:	Описание:
DisableFlag	SINT	GSV SSV	Управляет выполнением данной программы. **Значение:** **Означает:** 0 выполнение разрешено 1 выполнение запрещено
Instance	DINT	GSV	Указывает номер экземпляра для данного объекта PROGRAM.
LastScanTime	DINT	GSV SSV	Время, которое выполнялась программа в последний раз. Время указывается в микросекундах.
MajorFaultRecord	DINT[11]	GSV SSV	Регистрирует основные ошибки для данной программы Для упрощения доступа к атрибуту MajorFaultReord мы рекомендуем создать пользовательскую структуру:
Имя:	**Тип данных:**	**Стиль:**	**Описание:**
TimeLow	DINT	Decimal	младшие 32 бита значения временной отметки ошибки
TimeHigh	DINT	Decimal	старшие 32 бита значения временной отметки ошибки
Type	INT	Decimal	тип ошибки (программа, ввод/вывод и т.д.)
Code	INT	Decimal	уникальный код ошибки (зависит от типа ошибки)
Info	DINT[8]	Hexadecimal	информация по данной ошибке (зависит от типа и кода ошибки)
MaxScanTime	DINT	GSV SSV	Максимальное зарегистрированное время выполнения данной программы. Время указывается в микросекундах.
MinorFaultRecord	DINT[11]	GSV SSV	Регистрирует неосновные ошибки для данной программы Для упрощения доступа к атрибуту MinorFaultReord мы рекомендуем создать пользовательскую структуру:
Имя:	**Тип данных:**	**Стиль:**	**Описание:**
TimeLow	DINT	Decimal	младшие 32 бита значения временной отметки ошибки
TimeHigh	DINT	Decimal	старшие 32 бита значения временной отметки ошибки
Type	INT	Decimal	тип ошибки (программа, ввод/вывод и т.д.)
Code	INT	Decimal	уникальный код ошибки (зависит от типа ошибки)
Info	DINT[8]	Hexadecimal	информация по данной ошибке (зависит от типа и кода ошибки)
SFCRestart	INT	GSV SSV	не используется – зарезервирован на будущее

Обращение к объекту ROUTINE

Объект ROUTINE предоставляет информацию о состоянии подпрограммы. Чтобы определить нужный вам объект ROUTINE, задайте имя подпрограммы.

Атрибут:	Тип данных:	Инструкция:	Описание:
Instance	DINT	GSV	Указывает номер экземпляра для данного объекта ROUTINE. Допустимыми являются значения от 0 до 65535.

Обращение к объекту SERIALPORT

Объект SERIALPORT обеспечивает интерфейс с последовательным коммуникационным портом.

Атрибут:	Тип данных:	Инструкция:	Описание:	
BaudRate	DINT	GSV	Задает скорость передачи данных в бодах. Допустимыми значениями являются 110, 300, 600, 1200, 2400, 4800, 9600 и 19200 (по умолчанию).	
DataBits	SINT	GSV	Задает количество битов данных на один символ.	
			Значение:	**Означает:**
			7	7 битов данных (только ASCII)
			8	8 битов данных (по умолчанию)
Parity	SINT	GSV	Задает контроль по четности	
			Значение:	**Означает:**
			0	отсутствие контроля по четности (по умолчанию)
			1	контроль по нечетности (только ASCII)
			2	контроль по четности
RTSOffDelay	INT	GSV	Время задержки запрещения линии RTS (запроса на пересылку) после передачи последнего символа. Допустимыми являются значения от 0 до 32767. Задержка определяется в периодах по 20 мс. Значение по умолчанию – 0 мс.	
RTSSendDelay	INT	GSV	Время задержки передачи первого символа сообщения после включения линии RTS. Допустимыми являются значения от 0 до 32767. Задержка определяется в периодах по 20 мс. Значение по умолчанию – 0 мс.	
StopBits	SINT	GSV	Задает количество стоп-битов.	
			Значение:	**Означает:**
			1	1 стоп-бит (по умолчанию)
			2	2 стоп-бита (только ASCII)
PendingBaudRate	DINT	SSV	Ожидание значения для атрибута BaudRate	
PendingDataBits	SINT	SSV	Ожидание значения для атрибута DataBits.	
PendingParity	SINT	SSV	Ожидание значения для атрибута Parity.	

Атрибут:	Тип данных:	Инструкция:	Описание:
PendingRTSOffDelay	INT	SSV	Ожидание значения для атрибута RTSOffDelay.
PendingRTSSendDelay	INT	SSV	Ожидание значения для атрибута RTSSendDelay.
PendingStopBits	SINT	SSV	Ожидание значения для атрибута StopBts.

Для применения значений любого из находящихся в ожидании атрибутов SERIALPORT:

1. Используйте команду SSV для задания значения находящегося в ожидании атрибута.

   Вы можете задать значения для любого количества ожидающих атрибутов, используя команду SSV для каждого находящегося в ожидании атрибута.

2. Используйте команду MSG для использования заданного значения. Команда MSG применяет все заданные вами ожидающие атрибуты. Сконфигурируйте команду MSG следующим образом:

Вкладка MSG Configuration:	Поле:	Значение:
Configuration (Конфигурация)	Message Type Тип сообщения)	CIP Generic
	Service Code (Код сервиса)	Od hex
	Object type (Тип объекта)	6f hex
	Object ID (ID объекта)	1
	Object Attribute (Атрибут объекта)	оставить пустым
	Source (Источник)	оставить пустым
	Number of Elements (Количество элементов)	0
	Destination (Приемник)	оставить пустым
Communication (Связь)	Path (Путь)	Путь передачи данных к самому себе (1,s где s – номер слота для контроллера)

Обращение к объекту TASK

Объект TASK обеспечивает информацию о состоянии задачи. Чтобы определить нужный вам объект TASK, задайте имя задачи.

Атрибут:	Тип данных:	Инструкция:	Описание:
DisableUpdateOutputs	DINT	GSV SSV	Разрешает или запрещает обработку выходов в конце выполнения задачи **Чтобы:** — **Установите этот атрибут на:** разрешать обработку выходов в конце выполнения задачи — 0 запретить обработку выходов в конце выполнения задачи — 1 (или любое ненулевое значение)
EnableTimeOut	DINT	GSV SSV	Разрешает или запрещает функцию тайм-аута задачи события. **Чтобы:** — **Установите этот атрибут на:** запретить функцию тайм-аута — 0 разрешать функцию тайм-аута — 1 (или любое ненулевое значение)
InhibitTask	DINT	GSV SSV	Предотвращает выполнение задачи. Если задача запрещена, контроллер все равно выполняет предварительное сканирование задачи при переходе контроллера из режима программирования в режим работы или тестирования. **Чтобы:** — **Установите этот атрибут на:** разрешать задачу — 0 (по умолчанию) запретить (запретить) задачу — 1 (или любое ненулевое значение)
Instance	DINT	GSV	Указывает номер экземпляра для данного объекта TASK. Допустимыми являются значения от 0 до 31.
LastScanTime	DINT	GSV SSV	Время последнего выполнения этой задачи. Это время указывается в микросекундах.
MaxInterval	DINT[2]	GSV SSV	Максимальный временной интервал между последовательными выполнениями задачи. DINT[0] содержит младшие 32 бита этого значения; DINT[1] содержит старшие 32 бита значения. Значение 0 означает, что задача выполнялась не более 1 раза.
MaxScanTime	DINT	GSV SSV	Максимальное зарегистрированное время выполнения этой задачи. Это время указывается в микросекундах.

Атрибут:	Тип данных:	Инструкция:	Описание:
MinInterval	DINT[2]	GSV SSV	Минимальный временной интервал между последовательными выполнениями задачи. DINT[0] содержит младшие 32 бита этого значения; DINT[1] содержит старшие 32 бита значения. Значение 0 означает, что задача выполнялась не более 1 раза.
OverlapCount	DINT	GSV SSV	Количество запусков задачи в то время, как она выполнялась. Допустим для задачи события или периодической задачи. Для сброса счетчика установите этот атрибут на 0.
Priority	INT	GSV SSV	Относительная приоритетность задачи по сравнению с другими задачами. Допустимыми являются значения от 0 до 15.
Rate	DINT	GSV SSV	**Если тип этой задачи:** / **То атрибут Rate задает:** периодическая: Период выполнения задачи. Время указывается в микросекундах. событие: Значение тайм-аута для данной задачи. Время указывается в микросекундах.
StartTime	DINT[2]	GSV SSV	Значение WALLCLOCKTIME на начало последнего выполнения задачи. DINT[0] содержит младшие 32 бита этого значения; DINT[1] содержит старшие 32 бита значения.
Status	DINT	GSV SSV	Предоставляет информацию о состоянии задачи. После установки контроллером одного из этих битов вы должны вручную сбросить этот бит. **Чтобы определить:** / **Проверьте бит:** Запустила ли инструкция EVNT задачу (только для задачи события): 0 Запустил ли тайм-аут задачу (только для задачи обработки события): 1 Произошло ли перекрытие для данной задачи: 2
Watchdog	DINT	GSV SSV	Временной предел выполнения всех программ, связанных с данной задачей. Время указывается в микросекундах. Если вы введете 0, присваиваются следующие значения: **Время:** / **Тип задачи:** 0,5 с: периодическая или события 5,0 с: непрерывная

Обращение к объекту WALLCLOCKTIME

Объект WALLCLOCKTIME предоставляет временную отметку, которую контроллер может использовать для планирования.

Атрибут:	Тип данных:	Инструкция:	Описание:
CSTOffset	DINT[2]	GSV SSV	Положительное смещение по отношению к значению CurrentValue объекта CST (согласованное системное время, см. стр. 3-39). DINT[0] содержит младшие 32 бита этого значения; DINT[1] содержит старшие 32 бита значения. Значение указывается в микросекундах. По умолчанию принимается 0.
CurrentValue	DINT[2]	GSV SSV	Текущее значение времени по часам. DINT[0] содержит младшие 32 бита этого значения; DINT[1] содержит старшие 32 бита значения. Значение представляет собой время в микросекундах, прошедшее с 00 часов 00 минут 1 января 1972 г. Объекты CST и WALLCLOCKTIME в контроллере связаны математической зависимостью. Например, если вы сложите CurrentValue объекта CST и CSTOffset объекта WALLCLOCKTIME, вы получите CurrentValue объекта WALLCLOCKTIME.
DateTime	DINT[7]	GSV SSV	Дата и время в удобочитаемом формате. DINT[0] год DINT[1] целочисленное представление месяца (1-12) DINT[2] целочисленное представление дня (1-31) DINT[3] час (0-23) DINT[4] минута (0-59) DINT[5] секунды (0-59) DINT[6] микросекунды (0-999999)

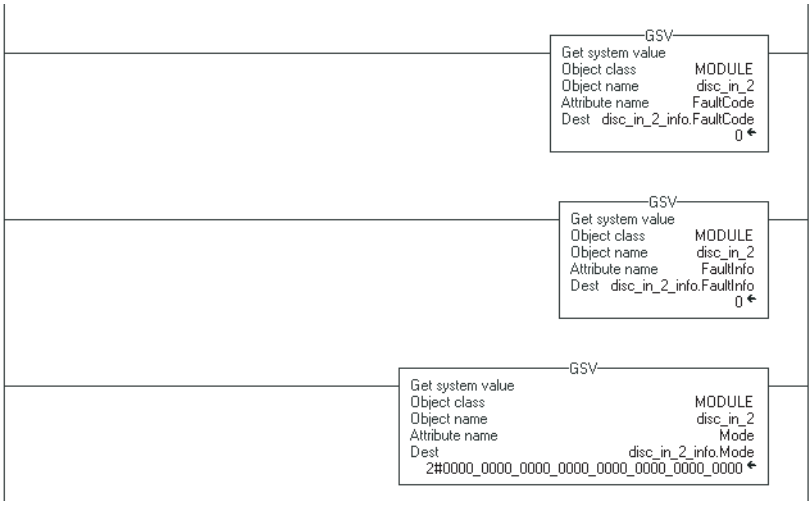
Пример программирования с использованием команд GSV/SSV

Получение информации об ошибках

В следующем примере команды GSV используются для получения информации об ошибках.

Пример 1: В этом примере информация об ошибках извлекается из модуля ввода/вывода *disc_in_2* и помещается в заданную пользователем структуру *disc_in_2_info*.

Релейная логика

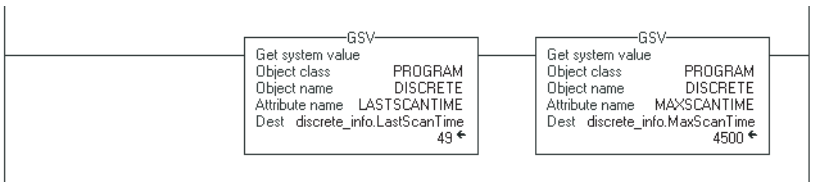


Структурированный текст

```
GSV(MODULE,disc_in_2,FaultCode,disc_in_2_info.FaultCode);

GSV(MODULE,disc_in_2,FaultInfo,disc_in_2_info.FaultInfo);

GSV(MODULE,disc_in_2,Mode,disc_in_2_info.Mode)
```

Пример 2: В этом примере извлекается информация о состоянии программы *discrete*, которая затем помещается в заданную пользователем структуру *discrete_info*.

Релейная логика

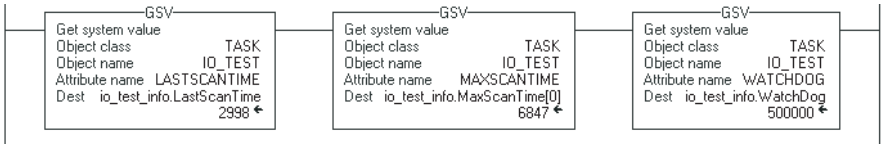


Структурированный текст

```
GSV(PROGRAM,DISCRETE,LASTSCANTIME,
     discrete_info.LastScanTime);

GSV(PROGRAM,DISCRETE,MAXSCANTIME,discrete_info.MaxScanTime);
```

Пример 3: В этом примере извлекается информация о состоянии задачи *IO_test*, которая затем помещается в заданную пользователем структуру *to_test_info*.

Релейная логика



Структурированный текст

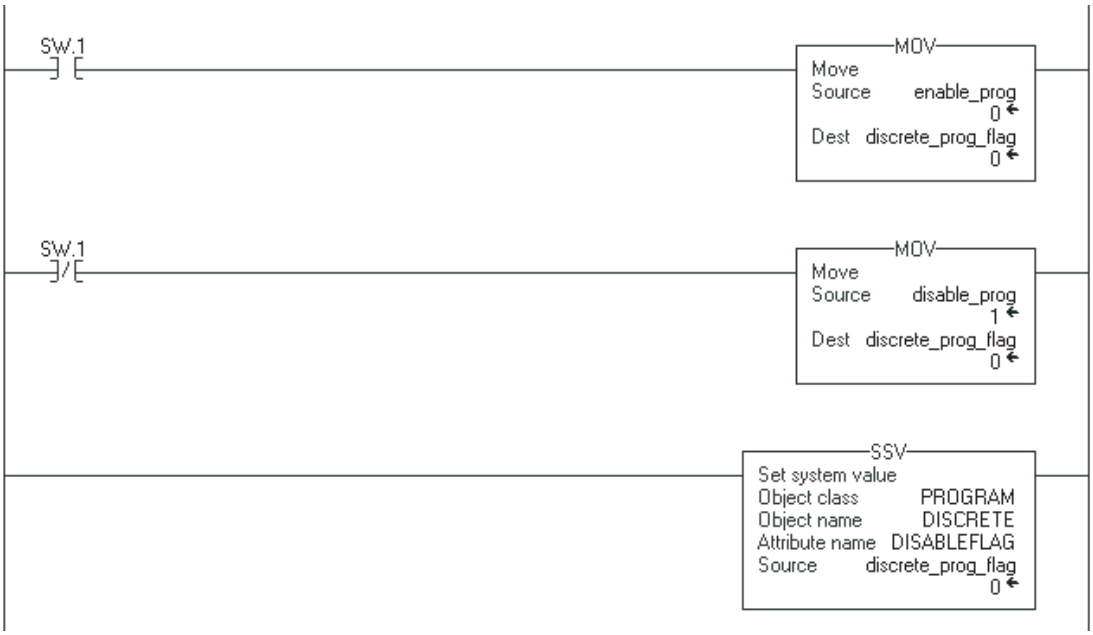
```
GSV(TASK,IO_TEST,LASTSCANTIME,io_test_info.LastScanTime);
GSV(TASK,IO_TEST,MAXSCANTIME,io_test_info.MaxScanTime);
GSV(TASK,IO_TEST,WATCHDOG,io_test_info.WatchDog);
```

Задание флагов разрешения и запрещения

В следующих примерах инструкция SSV используется для активизации или блокирования программы. Этот способ также можно использовать для включения или выключения модуля ввода/вывода аналогично использованию битов запрета для процессора PLC-5.

Пример: Значение, соответствующее состоянию SW.1, помещается в атрибут *disableflag* программы discrete.

Релейная логика



Структурированный текст

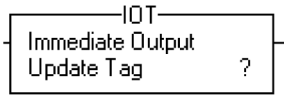
```
IF  SW.1  THEN
     discrete_prog_flag  :=  enable_prog;
ELSE
     discrete_prog_flag  :=  disable_prog;
END_IF;
SSV(PROGRAM,DISCRETE,DISABLEFLAG,discrete_prog_flag);
```

Immediate Output (IOT) (Немедленный вывод)

Инструкция IOT осуществляет немедленное обновление указанных выходных данных (выходного тега или производимого тега).

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:
Update Tag		тег	тег, который вы хотите обновить: · выходной тег модуля ввода/вывода или · производимый тег *Не* выбирайте член или элемент тега. Например, Local:5:0 годится, а Local:5:0:Data *не* годится.

Структурированный текст

```
IOT(output_tag);
```

Операнды те же, что и для инструкции IOT релейной логики.

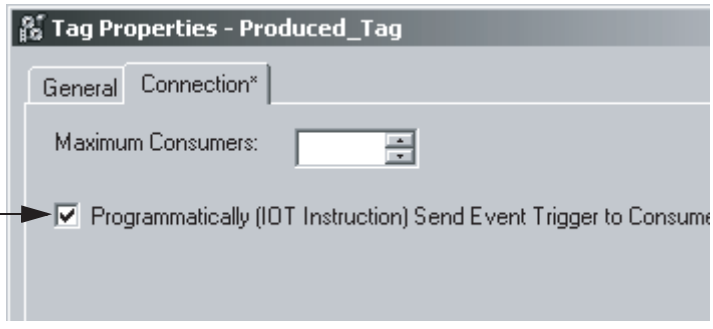
Описание:

Инструкция IOT отменяет запрошенный интервал передачи пакетов (RPI) выходного соединения и направляет по этому соединению свежие данные.

- Выходным соединением является соединение, относящееся к выходному тегу модуля ввода/вывода или к производимому тегу.
- Если соединение предназначено для производимого тега, инструкция IOT также направляет триггер события в потребляющий контроллер. Это позволяет инструкции IOT запустить задачу события в потребляющем контроллере.

Чтобы использовать инструкцию IOT и производимый тег для запуска задачи события в потребляющем контроллере, сконфигурируйте производимый тег следующим образом:

Установите этот флажок. Это конфигурирует тег таким образом, чтобы он обновлял свой триггер события только посредством команды IOT.

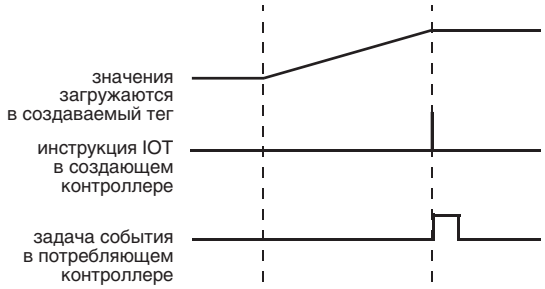


Тип сети между контроллерами определяет время получения новых данных и триггера события потребляющим контроллером при помощи инструкции IOT.

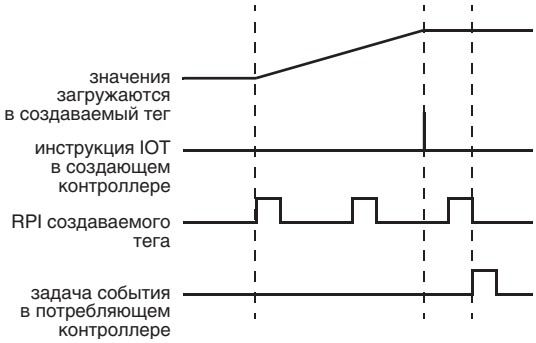
При использовании этого контроллера:	В этой сети:	Потребляющее устройство получит данные и триггер события:
ControlLogix	объединительная плата	немедленно
	сеть EtherNet/IP	немедленно
	сеть ControlNet	в течение фактического интервала передачи пакета (API) для потребляемого тега (соединение)
SoftLogix5800	Вы можете создавать и потреблять теги только в сети ControlNet.	в течение фактического интервала передачи пакета (API) для потребляемого тега (соединение)

На следующих графиках сравнивается получение данных с помощью инструкции IOT по сетям EtherNet/IP и ControlNet.

Сеть EtherNet/IP



Сеть ControlNet



Арифметические флаги состояния: не затрагиваются

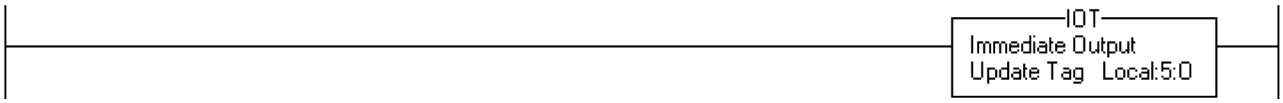
Условия ошибки: отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Ничего не происходит.
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn установлен	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция: · обновляет соединение заданного тега. · сбрасывает таймер RPI данного соединения.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Ничего не происходит.

Пример 1: При выполнении инструкции IOT она немедленно отправляет значения тега *Local:5:0* в модуль вывода.

Релейная логика

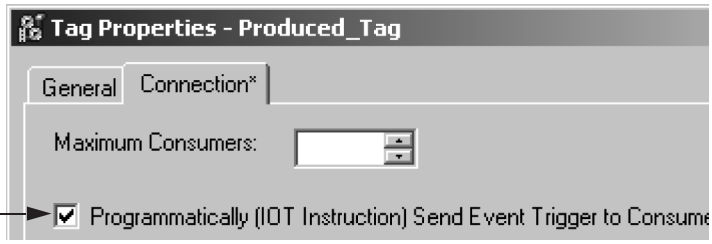


Структурированный текст

```
IOT  (Local:5:O);
```

Пример 2: Контроллер управляет станцией 24 и создает данные для следующее станции (станции 25). Чтобы использовать инструкцию IOT для подачи сигнала о передаче новых данных, производимый тег конфигурируется следующим образом:

Produced_Tag конфигурируется для обновления своего триггера события посредством команды IOT.

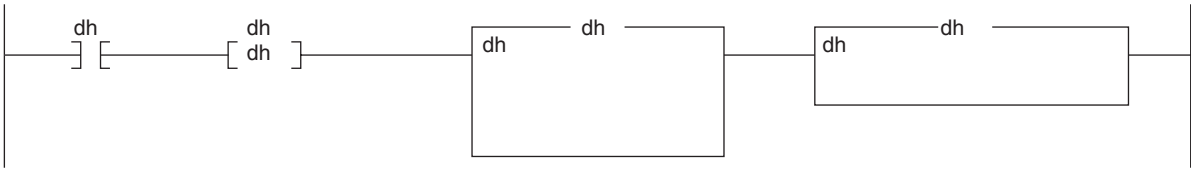


Релейная логика

Если *New_Data* = on, то для одного сканирования происходит следующее:

Инструкция CPS устанавливает *Produced_Tag* = *Source_Tag*.

Инструкция IOT обновляет *Produced_Tag* и направляет обновленные данные в потребляющий контроллер (станция 25). Когда потребляющий контроллер получает эти данные, он запускает соответствующую задачу события в этом контроллере.



Структурированный текст

```
IF  New_Data  AND  NOT  Trigger_Consumer  THEN
     CPS  (Source_Tag,Produced_Tag,1);
     IOT  (Produced_Tag);
END_IF;
Trigger_Consumer  :=  New_Data;
```

Инструкции сравнения

(CMP, EQU, GEQ, GRT, LEQ, LES, LIM, MEQ, NEQ)

Введение

Инструкции сравнения позволяют сравнивать значения при помощи выражения или специальной инструкции сравнения.

Если вы хотите:	Используйте инструкцию:	Имеющуюся в этих языках:	См. стр.
сравнить значения на основе выражения	CMP	релейная логика структурированный текст(1)	4-2
проверить, равно ли одно значение другому значению	EQU	релейная логика структурированный текст(2) функциональный блок	4-7
проверить, больше ли или равно одно значение другому значению	GEQ	релейная логика структурированный текст(1) функциональный блок	4-11
проверить, больше ли одно значение другого значения	GRT	релейная логика структурированный текст(1) функциональный блок	4-15
проверить, меньше ли или равно одно значение другому значению	LEQ	релейная логика структурированный текст(1) функциональный блок	4-19
проверить, меньше ли одно значение другого значения	LES	релейная логика структурированный текст(1) функциональный блок	4-23
проверить, находится ли одно значение между двумя другими значениями	LIM	релейная логика структурированный текст(1) функциональный блок	4-27
пропустить два значения через маску и проверить, равны ли они	MEQ	релейная логика структурированный текст(1) функциональный блок	4-33
проверить, не равно ли одно значение другому значению	NEQ	релейная логика структурированный текст(1) функциональный блок	4-38

(1) В структурированном тексте нет эквивалентной инструкции. Используйте другие средства программирования структурированного текста для получения того же результата. См. описание данной инструкции.

(2) В структурированном тексте нет эквивалентной инструкции. Используйте соответствующий оператор в выражении.

Вы можете сравнивать значения различных типов данных, например, с плавающей точкой и целочисленные.

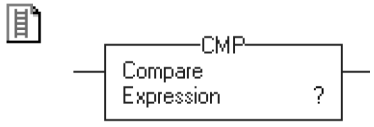
Оптимальные типы данных для инструкций релейной логики выделены **жирным** шрифтом. Инструкция выполняется быстрее и требует меньше памяти, если для всех операндов инструкции используется один и тот же оптимальный тип данных, обычно DINT или REAL.

Compare (CMP) (Сравнение)

Инструкция CMP выполняет сравнение для заданных в выражении арифметических операций.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Expression	SINT INT DINT REAL string	непосредственный	выражение, состоящее из тегов и/или непосредственных значений, разделенных операторами
	Тег SINT или INT преобразуется в значение DINT посредством дополнения знаком.		

Структурированный текст

В структурированном тексте нет инструкции CMP, но вы можете получить тот же результ с помощью конструкции IF..THEN и выражения.

```
IF  BOOL_expression  THEN
     <statement>;
END_IF;
```

Информацию о синтаксисе конструкций и выражений структурированного текста можно найти в Приложении C.

Описание:

Задайте выражение CMP с помощью операторов, тегов и непосредственных значений. Части более сложных выражений заключайте в скобки ().

Инструкция CMP выполняется несколько медленнее и использует больше памяти, чем другие инструкции сравнения. Преимущество инструкции CMP заключается в том, что она позволяет вводить сложные выражения в одной инструкции.

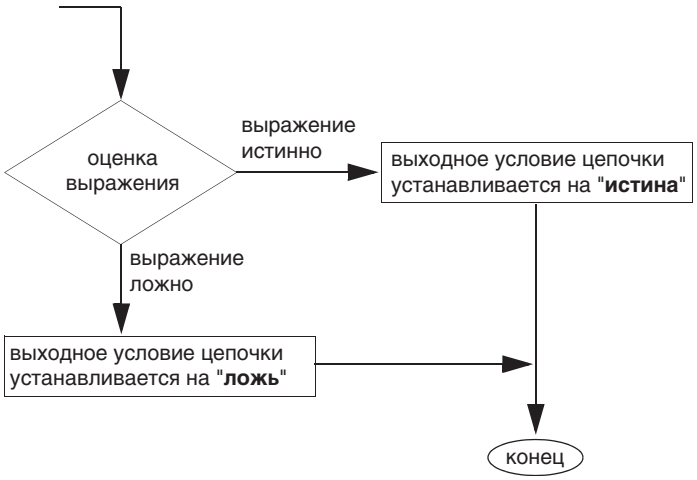
Арифметические флаги состояния:

Инструкция CMP влияет на арифметические флаги состояния лишь в том случае, если выражение содержит оператор (например, +, -, *, /), влияющий на арифметические флаги состояния.

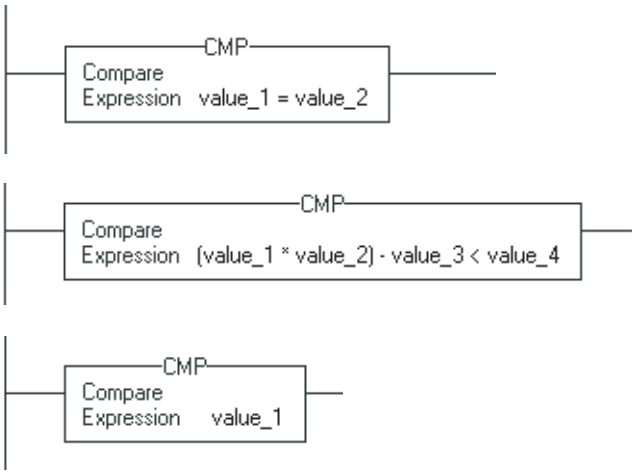
Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «истина»	
постсканирование	Выходное условие цепочки устанавливается на "ложь".

Примеры: Если инструкция CMP обнаружит, что выражение истинно, выходное условие цепочки установится на «истина».



Если вы введете выражение без оператора сравнения, например, *value_1 + value_2* или *value_1*, то эта инструкция будет оценивать выражение следующим образом:

Если выражение:	Выходное условие цепочки устанавливается на:
не равно нулю	истина
равно нулю	ложь

Выражения инструкции CMP

Выражение инструкции CMP программируются так же, как выражения инструкции FSC. В следующих разделах содержится информация по допустимым операторам, формату и порядку действий, общим для обеих инструкций.

Допустимые операторы

Оператор:	Описание:	Оптимальный тип данных:
+	сложение	DINT, REAL
-	вычитание/ отрицание	DINT, REAL
*	умножение	DINT, REAL
/	деление	DINT, REAL
=	равно	DINT, REAL
<	меньше	DINT, REAL
<=	меньше или равно	DINT, REAL
>	больше	DINT, REAL
>=	больше или равно	DINT, REAL
<>	не равно	DINT, REAL
**	степень (x в y)	DINT, REAL
ABS	абсолютное значение	DINT, REAL
ACS	арккосинус	REAL
AND	побитовое И	DINT
ASN	арксинус	REAL
ATN	арктангенс	REAL

Оператор:	Описание:	Оптимальный тип данных:
COS	косинус	REAL
DEG	радианы в градусы	DINT, REAL
FRD	BCD в целое	DINT
LN	натуральный логарифм	REAL
LOG	логарифм с основанием 10	REAL
MOD	деление по модулю	DINT, REAL
NOT	побитовое дополнение	DINT
OR	побитовое ИЛИ	DINT
RAD	градусы в радианы	DINT, REAL
SIN	синус	REAL
SQR	корень квадратный	DINT, REAL
TAN	тангенс	REAL
TOD	целое в BCD	DINT
TRN	усечение	DINT, REAL
XOR	побитовое исключающее ИЛИ	DINT

Форматирование выражений

Для каждого используемого в выражении оператора вы должны задать один или два операнда (теги или непосредственные значения). Для форматирования операторов и операндов в выражении используйте следующую таблицу:

Для операторов, оперирующих с:	Используйте этот формат:	Примеры:
одним операндом	оператор(операнд)	ABS(*tag_a)*
двумя операндами	операнд_a оператор операнд_b	*tag_b* + 5 *tag_c* AND *tag_d* (*tag_e* ** 2) MOD (*tag_f* / *tag_g*)

Задание порядка выполнения операций

Операции, которые вы записываете в выражение, выполняются инструкцией в установленном порядке, не обязательно в том порядке, в котором вы их записываете. Вы можете изменить порядок выполнения операций, сгруппировав члены выражения в скобках и принудив тем самым инструкцию выполнять операцию в скобках раньше других операций.

Операции одного порядка выполняются слева направо.

Порядок:	Операция:
1.	()
2.	ABS, ACS, ASN, ATN, COS, DEG, FRD, LN, LOG, RAD, SIN, SQR, TAN, TOD, TRN
3.	**
4.	- (отрицание), NOT
5.	*, /, MOD
6.	<, <=, >, >=, =
7.	- (вычитание), +
8.	AND
9.	XOR
10.	OR

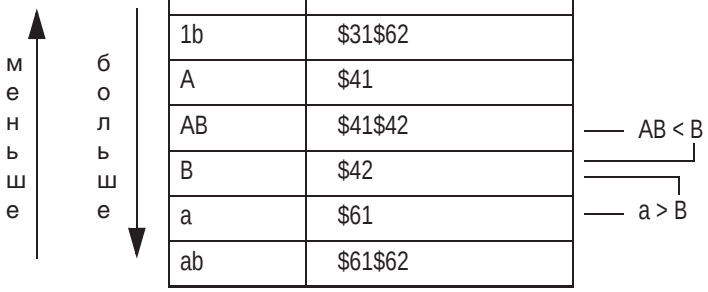
Использование строк в выражении

Для выполнения сравнения строковых данных используйте релейную логику или выражение структурированного текста. Используйте строки в выражении в соответствии со следующими инструкциями:

- Выражение позволяет вам сравнивать два строковых тега.
- Вы *не можете* вводить символы ASCII непосредственно в выражение.
- Разрешается использование только следующих операторов

Оператор:	Описание:
=	равно
<	меньше
<=	меньше или равно
>	больше
>=	больше или равно
<>	неравно

- Строки равны, если их символы совпадают.
- Символы ASCII чувствительны к регистру. Заглавная «A» ($41) *не* равна строчной «a» ($61).
- Шестнадцатеричные значения символов определяют, больше или меньше одна строка другой строки. Шестнадцатеричные коды символов приводятся на задней обложке этого руководства.
- Когда две строки осортированы как в телефонном справочнике, порядок следования символов в строках определяет, какая строка больше.



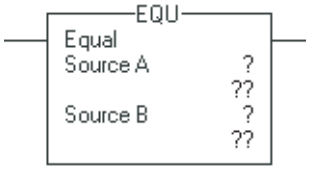
Символы ASCII	Шестнадцатеричный код
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

Equal to (EQU) (Равно)

Инструкция EQU проверяет, равно ли значение Source A значению Source B.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Source A	SINT INT **DINT** **REAL** **строка**	непосредственный тег	значение, проверяемое против Source B
Source B	SINT INT **DINT** **REAL** **строка**	непосредственный тег	значение, проверяемое против Source A

- Если вы ввели тег SINT или INT, то это значение преобразуется в DINT посредством дополнения знаком.
- Значения типа REAL редко бывают абсолютно равными. Если вам требуется определить равенство двух значений типа REAL, используйте инструкцию LIM.
- Строковыми типами данных являются:
  - используемый по умолчанию тип данных STRING
  - все создаваемые вами новые типы строковых данных
- Чтобы выполнить проверку для символов строки, введите строковый тег как для Source A, так и для Source B.

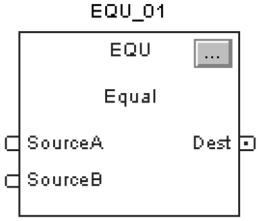
Структурированный текст

```
IF sourceA = sourceB THEN
    <statements>;
```

Используйте знак равно «=» в качестве оператора в выражении. Это выражение оценивает, равно ли значение *sourceA* значению *sourceB.*

Информация о синтаксисе выражений структурированного текста содержится в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег EQU	FBD_COMPARE	структура	структура EQU

Структура FBD_COMPARE

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, инструкция не выполняется, и выходы не обновляются. По умолчанию установлен.
SourceA	REAL	Значение, проверяемое против SourceB. Допустимым значением является любое число с плавающей точкой.
SourceB	REAL	Значение, проверяемое против SourceA. Допустимым значением является любое число с плавающей точкой.
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	В результате выполнения инструкции получен допустимый результат.
Dest	BOOL	Результат выполнения инструкции. Это эквивалентно выходному условию цепочки для инструкции EQU релейной логики.

Описание: Используйте инструкцию EQU для сравнения двух чисел или двух строк символов ASCII. При сравнении строк:

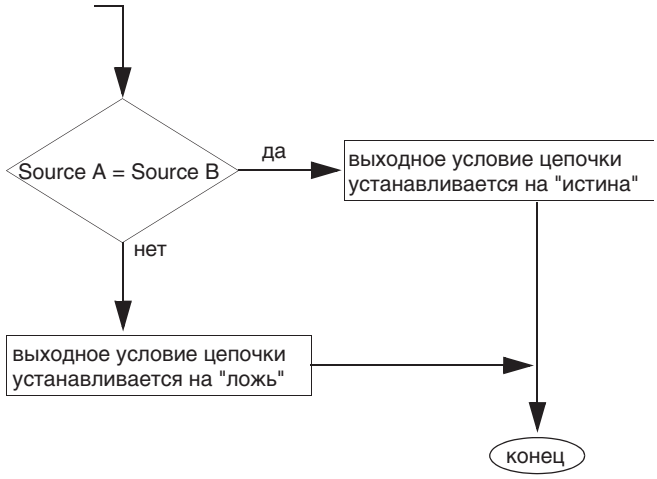
- Строки равны, если их символы совпадают.
- Символы ASCII чувствительны к регистру. «A» верхнего регистра ($41) *не* равно «a» нижнего регистра ($61).

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «истина»	
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Ничего не происходит.
первое сканирование инструкции	Ничего не происходит.
первое выполнение инструкции	Ничего не происходит.
EnableIn сброшен	EnableOut сбрасывается
EmableIn установлен	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Ничего не происходит.

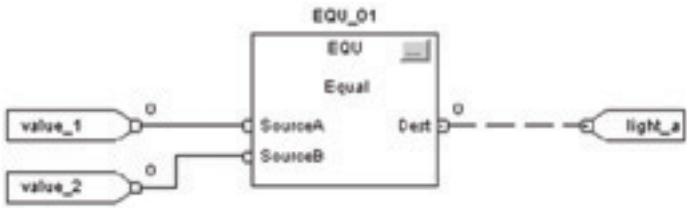
Пример: Если *value_1* равно *value_2*, *light_a* устанавливается. Если *value_1* не равно *value_2*, *light_a* сбрасывается.

Релейная логика

Структурированный текст

```
light_a := (value_1 = value_2);
```

Функциональный блок

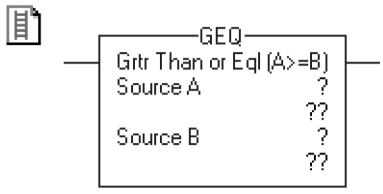


Greater than or Equal to (GEQ) (Больше или равно)

Инструкция GEQ проверяет, больше ли или равно ли значение Source A значению Source B.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Source A	SINT INT **DINT** **REAL** **строка**	непосредственный тег	значение, проверяемое против Source B
Source B	SINT INT **DINT** **REAL** **строка**	непосредственный тег	значение, проверяемое против Source A

- Если вы ввели тег SINT или INT, то это значение преобразуется в DINT посредством дополнения знаком.
- Строковыми типами данных являются:
  - используемый по умолчанию тип данных STRING
  - все создаваемые вами новые типы строковых данных
- Чтобы выполнить проверку для символов строки, введите строковый тег как для Source A, так и для Source B.

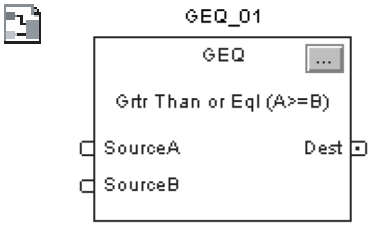
Структурированный текст

```
IF sourceA >= sourceB THEN
    <statements>;
```

Используйте расположенные подряд знаки больше и равно «>=» в качестве оператора в выражении. Это выражение оценивает, больше ли или равно ли значение *sourceA* значению *sourceB*.

Информация о синтаксисе выражений структурированного текста содержится в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег GEQ	FBD_COMPARE	структура	структура GEQ

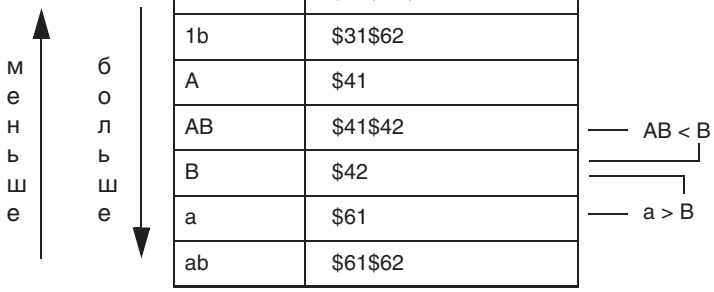
Структура FBD_COMPARE

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, инструкция не выполняется, и выходы не обновляются. По умолчанию установлен.
SourceA	REAL	Значение, проверяемое против SourceB. Допустимым значением является любое число с плавающей точкой.
SourceB	REAL	Значение, проверяемое против SourceA. Допустимым значением является любое число с плавающей точкой.
Выходной параметр:	Тип данных:	Описание:
EnableOut	BOOL	В результате выполнения инструкции получен допустимый результат.
Dest	BOOL	Результат выполнения инструкции. Это эквивалентно выходному условию цепочки для инструкции GEQ релейной логики.

Описание: Используйте инструкцию GEQ для проверки, больше ли или равно ли значение Source A значению Source B.

При сравнении строк:

- Шестнадцатеричные значения символов определяют, больше или меньше одна строка другой строки. Шестнадцатеричные коды символов приводятся на задней обложке этого руководства.
- Когда две строки отсортированы как в телефонном справочнике, порядок следования символов в строках определяет, какая строка больше



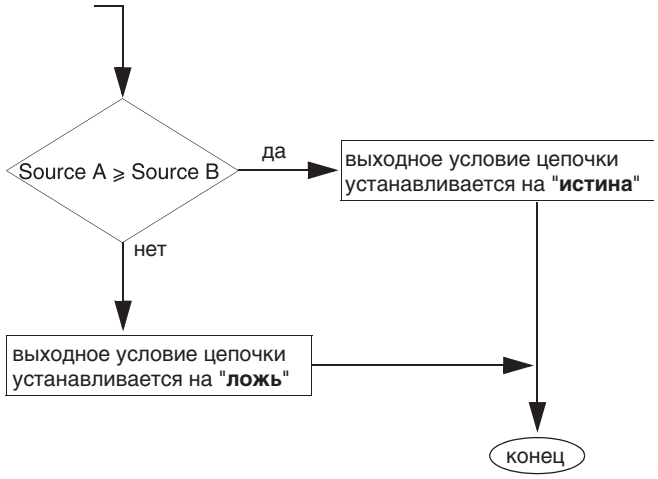
Символы ASCII	Шестнадцатеричный код
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

Условие:	**Действие:**
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «истина»	
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	**Действие:**
предварительное сканирование	Ничего не происходит.
первое сканирование инструкции	Ничего не происходит.
первое выполнение инструкции	Ничего не происходит.
EnableIn сброшен	EnableOut сбрасывается
EmableIn установлен	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Ничего не происходит.

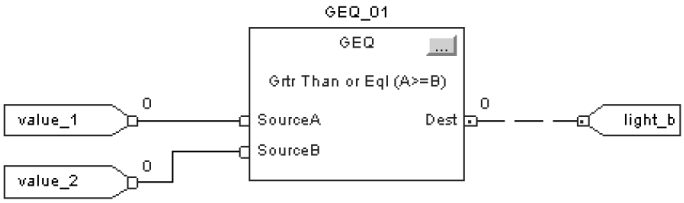
Пример: Если *value_1* больше или равно *value_2*, *light_b* устанавливается. Если *value_1* меньше *value_2*, *light_b* сбрасывается.

Релейная логика

Структурированный текст

```
light_b := (value_1 >= value_2);
```

Функциональный блок

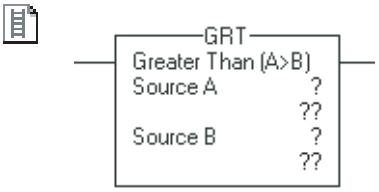


Greater Than (GRT) (Больше)

Инструкция GRT проверяет, больше ли значение Source A значения Source B.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Source A	SINT INT **DINT** **REAL** **строка**	непосредственный тег	значение, проверяемое против Source B
Source B	SINT INT **DINT** **REAL** **строка**	непосредственный тег	значение, проверяемое против Source A

- Если вы ввели тег SINT или INT, то это значение преобразуется в DINT посредством дополнения знаком.
- Строковыми типами данных являются:
  - используемый по умолчанию тип данных STRING
  - все создаваемые вами новые типы строковых данных
- Чтобы выполнить проверку для символов строки, введите строковый тег как для Source A, так и для Source B.

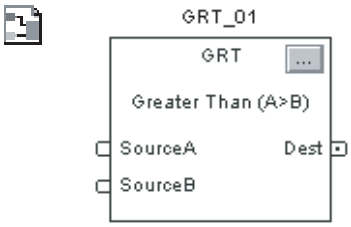
Структурированный текст

```
IF sourceA > sourceB THEN
    <statements>;
```

Используйте знак больше «>» в качестве оператора в выражении. Это выражение оценивает, больше ли значение *sourceA* значения *sourceB.*

Информация о синтаксисе выражений структурированного текста содержится в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег GRT	FBD_COMPARE	структура	структура GRT

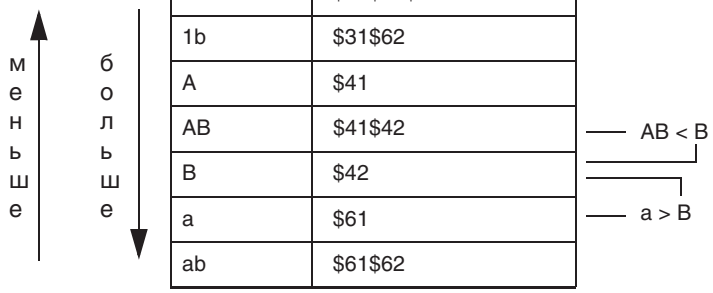
Структура FBD_COMPARE

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, инструкция не выполняется, и выходы не обновляются. По умолчанию установлен.
SourceA	REAL	Значение, проверяемое против SourceB. Допустимым значением является любое число с плавающей точкой.
SourceB	REAL	Значение, проверяемое против SourceA. Допустимым значением является любое число с плавающей точкой.
Выходной параметр:	Тип данных:	Описание:
EnableOut	BOOL	В результате выполнения инструкции получен допустимый результат.
Dest	BOOL	Результат выполнения инструкции. Это эквивалентно выходному условию цепочки для инструкции GRT релейной логики.

Описание: Используйте инструкцию GRT для проверки, больше ли значение Source A значения Source B.

При сравнении строк:

- Шестнадцатеричные значения символов определяют, больше или меньше одна строка другой строки. Шестнадцатеричные коды символов приводятся на задней обложке этого руководства.
- Когда две строки отсортированы как в телефонном справочнике, порядок следования символов в строках определяет, какая строка больше.



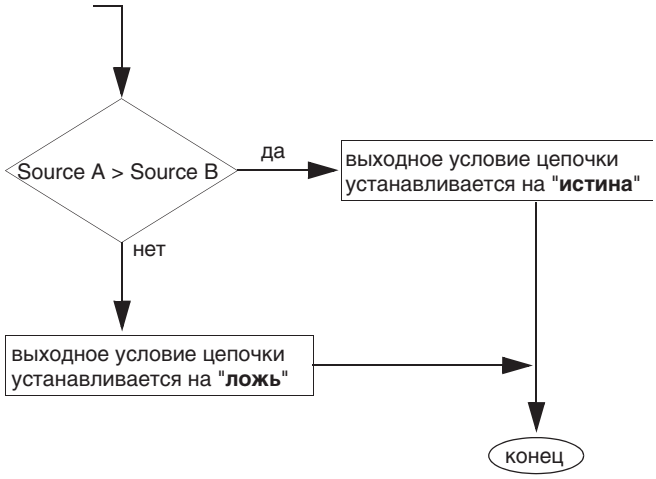
Символы ASCII	Шестнадцатеричный код
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

Условие:	**Действие:**
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «истина»	
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	**Действие:**
предварительное сканирование	Ничего не происходит.
первое сканирование инструкции	Ничего не происходит.
первое выполнение инструкции	Ничего не происходит.
EnableIn сброшен	EnableOut сбрасывается
EmableIn установлен	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Ничего не происходит.

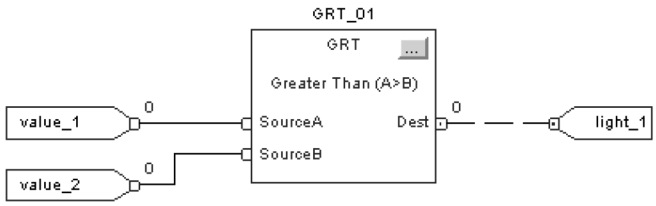
Пример: Если *value_1* больше *value_2*, *light_1* устанавливается. Если *value_1* меньше или равно *value_2*, *light_1* сбрасывается.

Релейная логика

Структурированный текст

```
light_1 := (value_1 > value_2);
```

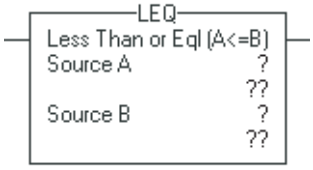
Функциональный блок



Less than or Equal to (LEQ) (Меньше или равно)

Инструкция LEQ проверяет, меньше ли или равно ли значение Source A значению Source B.

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Описание:
Source A	SINT INT **DINT** **REAL** **строка**	непосредственный тег	значение, проверяемое против Source B
Source B	SINT INT **DINT** **REAL** **строка**	непосредственный тег	значение, проверяемое против Source A

- Если вы ввели тег SINT или INT, то это значение преобразуется в DINT посредством дополнения знаком.
- Строковыми типами данных являются:
  - используемый по умолчанию тип данных STRING
  - все создаваемые вами новые типы строковых данных
- Чтобы выполнить проверку для символов строки, введите строковый тег как для Source A, так и для Source B.

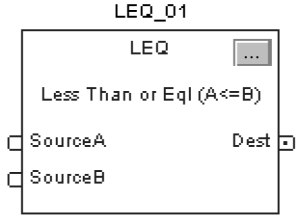
Структурированный текст

```
IF sourceA <= sourceB THEN
    <statements>;
```

Используйте расположенные подряд знаки меньше и равно «<=» в качестве оператора в выражении. Это выражение оценивает, меньше ли или равно ли значение *sourceA* значению *sourceB*.

Информация о синтаксисе выражений структурированного текста содержится в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег LEQ	FBD_COMPARE	структура	структура LEQ

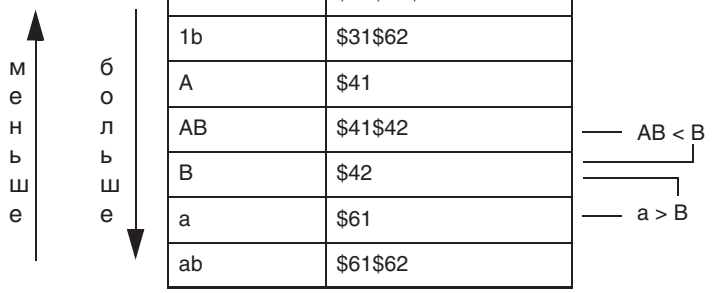
Структура FBD_COMPARE

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, инструкция не выполняется, и выходы не обновляются. По умолчанию установлен.
SourceA	REAL	Значение, проверяемое против SourceB. Допустимым значением является любое число с плавающей точкой.
SourceB	REAL	Значение, проверяемое против SourceA. Допустимым значением является любое число с плавающей точкой.
Выходной параметр:	Тип данных:	Описание:
EnableOut	BOOL	В результате выполнения инструкции получен допустимый результат.
Dest	BOOL	Результат выполнения инструкции. Это эквивалентно выходному условию цепочки для инструкции LEQ релейной логики.

Описание: Используйте инструкцию LEQ для проверки, меньше ли или равно ли значение Source A значению Source B.

При сравнении строк:

- Шестнадцатеричные значения символов определяют, больше или меньше одна строка другой строки. Шестнадцатеричные коды символов приводятся на задней обложке этого руководства.
- Когда две строки отсортированы как в телефонном справочнике, порядок следования символов в строках определяет, какая строка больше.



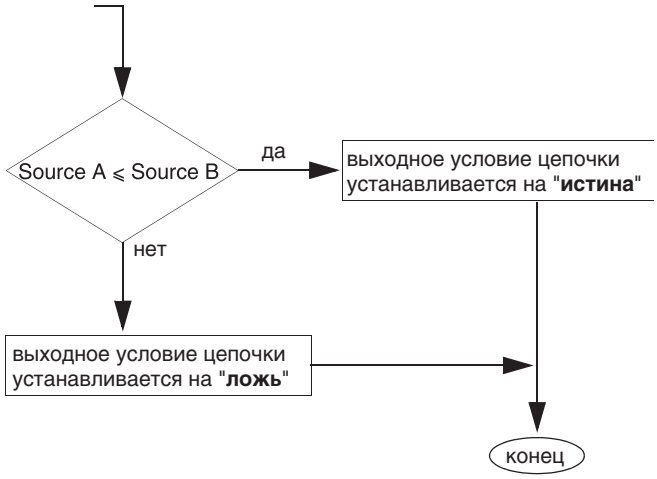
Символы ASCII	Шестнадцатеричный код
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

Условие:	**Действие:**
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «истина»	
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	**Действие:**
предварительное сканирование	Ничего не происходит.
первое сканирование инструкции	Ничего не происходит.
первое выполнение инструкции	Ничего не происходит.
EnableIn сброшен	EnableOut сбрасывается
EmableIn установлен	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Ничего не происходит.

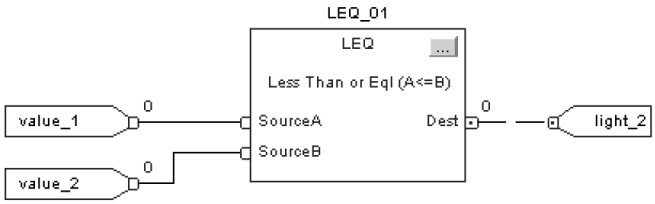
Пример: Если *value_1* меньше или равно *value_2*, *light_2* устанавливается. Если *value_1* больше *value_2*, *light_2* сбрасывается.

Релейная логика

Структурированный текст

```
light_2 := (value_1 <= value_2);
```

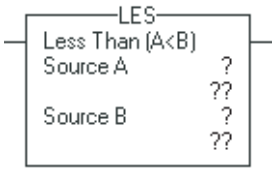
Функциональный блок



Less Than (LES) (Меньше)

Инструкция LES проверяет, меньше ли значение Source A значения Source B.

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Описание:
Source A	SINT INT **DINT** **REAL** **строка**	непосредственный тег	значение, проверяемое против Source B
Source B	SINT INT **DINT** **REAL** **строка**	непосредственный тег	значение, проверяемое против Source A

- Если вы ввели тег SINT или INT, то это значение преобразуется в DINT посредством дополнения знаком.
- Строковыми типами данных являются:
  - используемый по умолчанию тип данных STRING
  - все создаваемые вами новые типы строковых данных
- Чтобы выполнить проверку для символов строки, введите строковый тег как для Source A, так и для Source B.

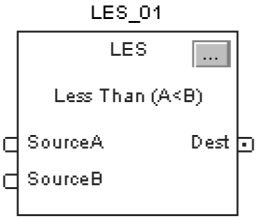
Структурированный текст

```
IF sourceA < sourceB THEN
    <statements>;
```

Используйте знак меньше «<» в качестве оператора в выражении. Это выражение оценивает, меньше ли значение *sourceA* значения *sourceB*.

Информация о синтаксисе выражений структурированного текста содержится в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег LES	FBD_COMPARE	структура	структура LES

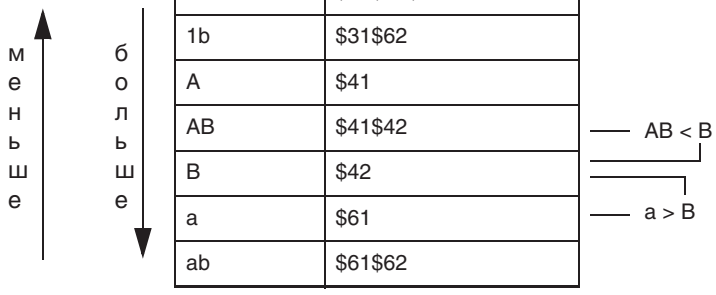
Структура FBD_COMPARE

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, инструкция не выполняется, и выходы не обновляются. По умолчанию установлен.
SourceA	REAL	Значение, проверяемое против SourceB. Допустимым значением является любое число с плавающей точкой.
SourceB	REAL	Значение, проверяемое против SourceA. Допустимым значением является любое число с плавающей точкой.
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	В результате выполнения инструкции получен допустимый результат.
Dest	BOOL	Результат выполнения инструкции. Это эквивалентно выходному условию цепочки для инструкции LES релейной логики.

Описание: Используйте инструкцию LES для проверки, меньше ли значение Source A значения Source B.

При сравнении строк:

- Шестнадцатеричные значения символов определяют, больше или меньше одна строка другой строки. Шестнадцатеричные коды символов приводятся на задней обложке этого руководства.
- Когда две строки отсортированы как в телефонном справочнике, порядок следования символов в строках определяет, какая строка больше.



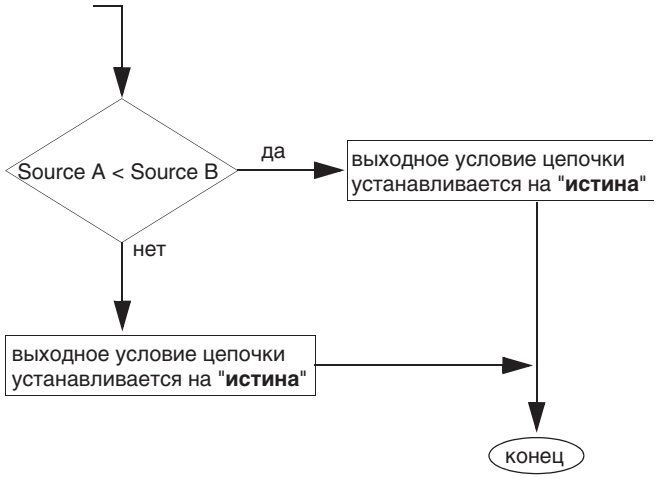
Символы ASCII	Шестнадцатеричный код
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

Условие:	**Действие:**
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «истина»	
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	**Действие:**
предварительное сканирование	Ничего не происходит.
первое сканирование инструкции	Ничего не происходит.
первое выполнение инструкции	Ничего не происходит.
EnableIn сброшен	EnableOut сбрасывается
EmableIn установлен	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Ничего не происходит.

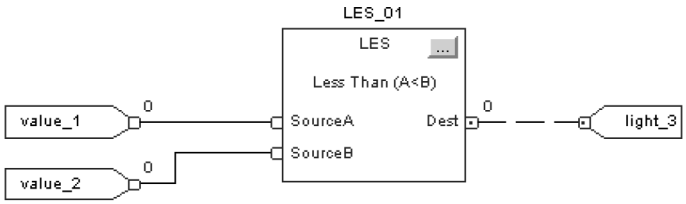
Пример: Если *value_1* меньше *value_2*, *light_3* устанавливается. Если *value_1* больше или равно *value_2*, *light_3* сбрасывается.

Релейная логика

Структурированный текст

```
light_3 := (value_1 < value_2);
```

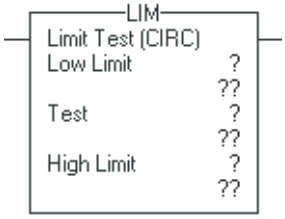
Функциональный блок



Limit (LIM) (Предел)

Инструкция LIM проверяет, находится ли значение Test в диапазоне между значениями Low Limit и High Limit.

Операнды:



Релейная логика

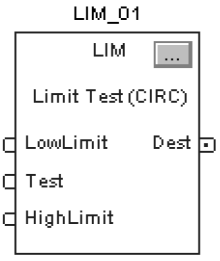
Операнд:	Тип:	Формат:	Описание:
Low limit	SINT INT **DINT** **REAL**	непосредственный тег	значение нижнего предела
	Тег SINT или INT преобразуется в DINT посредством дополнения знаком.		
Test	SINT INT **DINT** **REAL**	непосредственный тег	проверяемое значение
	Тег SINT или INT преобразуется в DINT посредством дополнения знаком.		
High limit	SINT INT **DINT** **REAL**	непосредственный тег	значение верхнего предела
	Тег SINT или INT преобразуется в DINT посредством дополнения знаком.		

Структурированный текст

В структурированном тексте инструкция LIM отсутствует, но вы можете получить тот же результат с помощью следующей конструкции структурированного текста.

```
IF  (LowLimit  <=  HighLimit  AND
     (Test  >=  LowLimit  AND  Test  <=  HighLimit))  OR
     (LowLimit  >=  HighLimit  AND
     (Test  <=  LowLimit  OR  Test  >=  HighLimit))  THEN

          <statement>;

END_IF;
```

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег LIM	FBD_LIMIT	структура	структура LIM

Структура FBD_LIMIT

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Если этот параметр сброшен, инструкция не выполняется, и выходы не обновляются. Если он установлен, инструкция выполняется, как описано в разделе «Выполнение». По умолчанию установлен.
LowLimit	REAL	Значение нижнего предела. Допустимым значением является любое число с плавающей точкой.
Test	REAL	Значение, проверяемое против пределов. Допустимым значением является любое число с плавающей точкой.
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	В результате выполнения инструкции получен допустимый результат.
Dest	BOOL	Результат выполнения инструкции. Это эквивалентно выходному условию цепочки для инструкции LIM релейной логики.
HighLimit	REAL	Значение верхнего предела. Допустимым значением является любое число с плавающей точкой.

Описание:

Инструкция LIM проверяет, находится ли значение Test в диапазоне между Low Limit и High Limit.

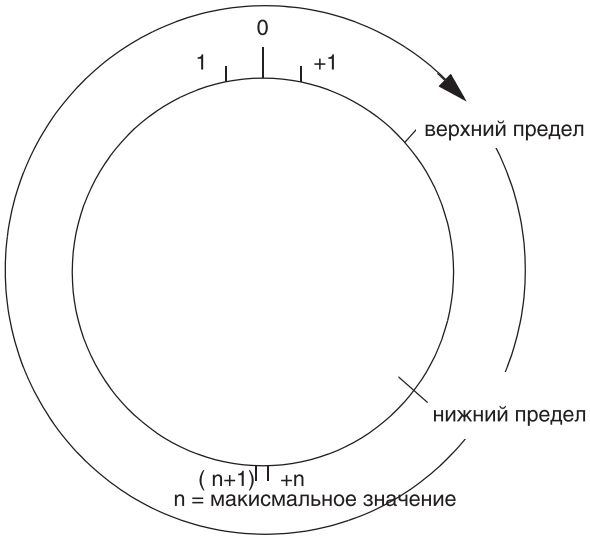
Если Low Limit:	И значение Test:	То выходное условие цепочки:
≤ High Limit	равно одному из пределов или находится между ними	истина
	не равно ни одному из пределов или находится вне ограниченного ими диапазона	ложь
≥ High Limit	равно одному из пределов или находится вне ограниченного ими диапазона	истина
	не равно ни одному из пределов или находится между ними	ложь

Целые числа со знаком переходят от максимального положительного числа к максимальному отрицательному числу, когда устанавливается самый старший значащий бит. Например, для 16-разрядных целых чисел (типа INT) максимальным положительным целым числом является 32767, которое записывается в шестнадцатеричном виде как 16#7FFF (биты с 0 по 14 установлены). Если вы увеличите это число на единицу, то получится 16#8000 (бит 15 установлен). Для целых чисел со знаком шестнадцатеричному 16#8000 соответствует десятичное число –32768. Увеличивая это число на единицу до тех пор, пока не будет установлены все 16 битов, получим 16#FFFF, что соответствует десятичному числу –1.

Это можно показать на круговой числовой оси (см. следующие рисунки). Инструкция LIM начинает работать от нижнего предела (Low Limit) и увеличивает значение в направлении по часовой стрелке, пока не будет достигнут верхний предел (High Limit). Всякое проверяемое значение (Test), находящееся в диапазоне между нижним и верхним пределом по часовой стрелке, устанавливает выходное условие цепочки на "истина". Всякое проверяемое значение, находящееся в диапазоне между верхним и нижним пределом по часовой стрелке, устанавливает выходное условие цепочки на "ложь".

Low Limit ⩽ High Limit
Результатом инструкции является "истина", если проверяемое значение равно одному из пределов или находится между нижним и верхним пределами

Low Limit ⩾ High Limit
Результатом инструкции является "истина", если проверяемое значение равно одному из пределов или находится вне диапазона, ограниченного нижним и верхним пределами

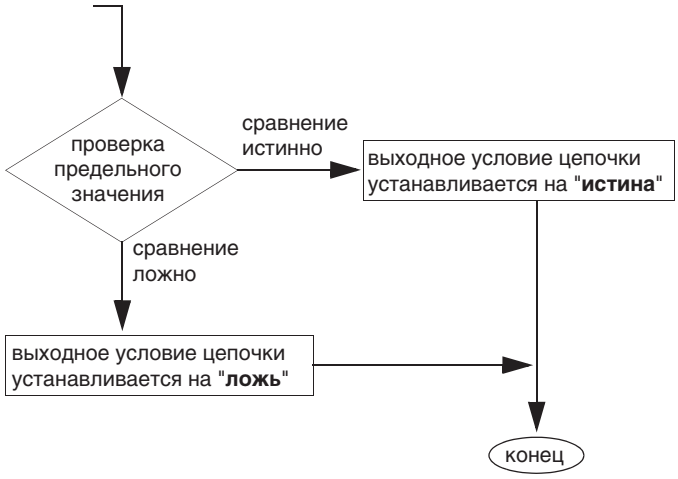


Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «истина»	
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Ничего не происходит.
первое сканирование инструкции	Ничего не происходит.
первое выполнение инструкции	Ничего не происходит.
EnableIn сброшен	EnableOut сбрасывается, инструкция ничего не делает, и выходы не обновляются.
EmableIn установлен	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Ничего не происходит.

Пример 1: Low Limit ⩽ High Limit

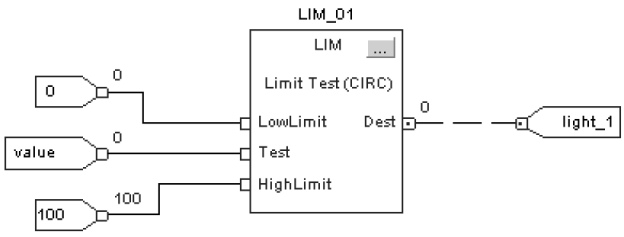
Когда 0 ⩽ *value* ⩾ 100, устанавливается *light_1*. Если *value* < 0 или *value* > 100, *light_1* сбрасывается.

Релейная логика

Структурированный текст

```
IF (value <= 100 AND(value >= 0 AND value <= 100)) OR
     (value >= 100 AND value <= 0 OR value >= 100)) THEN

          light_1 := 1;

ELSE

          light_1 := 0;

END_IF;
```

Функциональный блок



Пример 2: Low Limit ⩾ High Limit

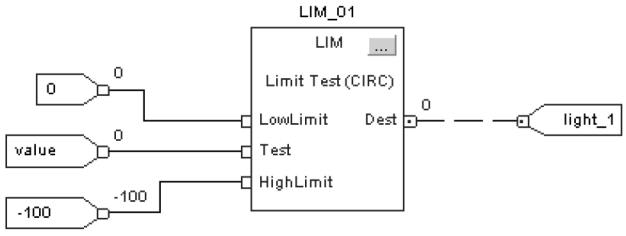
Когда value ⩾ 0 или *value* ⩽ -100, устанавливается *light_1*.
Если *value* < 0 или *value* > - 100, *light_1* сбрасывается.

Релейная логика

Структурированный текст

```
IF (0 <= -100 AND value >= 0 AND value <= -100)) OR
     (0 >= -100 AND(value <= 0 OR value >= -100)) THEN
          light_1 := 1;
ELSE
          light_1 := 0;
END_IF;
```

Функциональный блок

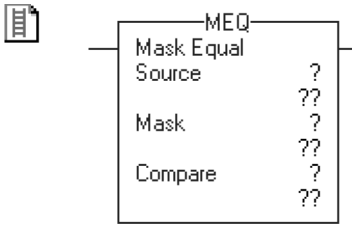


Mask Equal to (MEQ)

Инструкция MEQ пропускает значения Source и Compare через Mask и выполняет сравнение полученных результатов.

Операнды:

Релейная логика



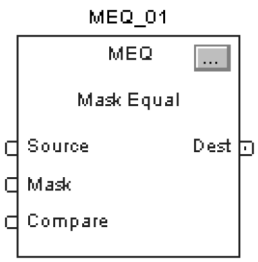
Операнд:	Тип:	Формат:	Описание:
Source	SINT INT **DINT**	непосредственный тег	значение, проверяемого против Compare
	Тег SINT или INT преобразуется в DINT посредством дополнения знаком.		
Mask	SINT INT **DINT**	непосредственный тег	определяет, какие биты блокировать, а какие пропускать
	Тег SINT или INT преобразуется в DINT посредством дополнения знаком.		
Compare	SINT INT **DINT**	непосредственный тег	значение верхнего предела
	Тег SINT или INT преобразуется в DINT посредством дополнения знаком.		

Структурированный текст

В структурированном тексте инструкция MEQ отсутствует, но вы можете получить тот же результат с помощью следующей конструкции структурированного текста.

```
IF (Source AND Mask) = (Compare AND Mask) THEN
     <statement>;
END_IF;
```

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег MEQ	FBD_MASK_EQUAL	структура	структура MEQ

Структура FBD_MASK_EQUAL

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Если этот параметр сброшен, инструкция не выполняется, и выходы не обновляются. Если он установлен, инструкция выполняется, как описано в разделе «Выполнение». По умолчанию установлен.
Source	DINT	Значение, проверяемое против Compare. Допустимым значением является любое целое число.
Mask	DINT	Задает, какие биты блокировать (маскировать). Допустимым значением является любое целое число.
Compare	DINT	Значение для сравнения. Допустимым значением является любое целое число.
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	В результате выполнения инструкции получен допустимый результат.
Dest	BOOL	Результат выполнения инструкции. Это эквивалентно выходному условию цепочки для инструкции MEQ релейной логики.

Описание: «1» в маске означает, что бит данных пропускается для сравнения. «0» в маске означает, что бит данных блокируется. Как правило, Source, Mask и Compare имеют один тип данных.

Если вы смешиваете типы целочисленных данных, инструкция будет заполнять старшие биты меньших значений целочисленных типов данных нулями так, чтобы они имели тот же размер, что и наибольшее значение этого типа данных.

Ввод непосредственного значения маски

Когда вы вводите значение маски, программное обеспечение по умолчанию воспринимает ее как десятичное значение. Если вы хотите ввести маску, используя другой формат, снабдите значение соответствующим префиксом.

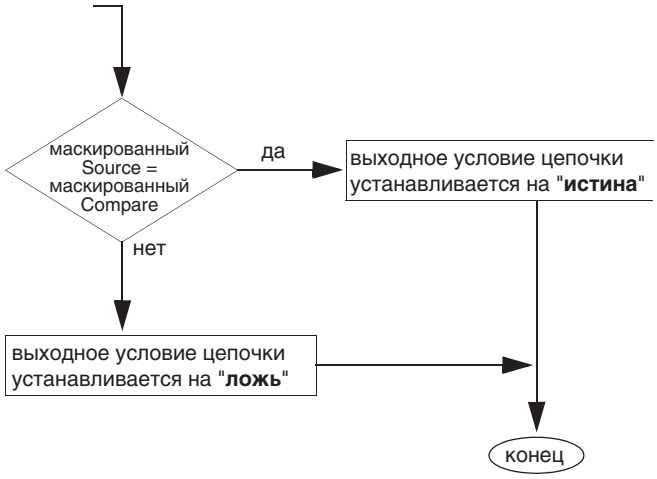
Префикс:	Описание:
16#	шестнадцатеричный например, 16#0F0F
8#	восьмеричный например, 8#16
2#	двоичный например, 2#00110011

Арифметические флаги состояния: не затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «истина»	
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Ничего не происходит.
первое сканирование инструкции	Ничего не происходит.
первое выполнение инструкции	Ничего не происходит.
EnableIn сброшен	EnableOut сбрасывается, инструкция ничего не делает, и выходы не обновляются.
EmableIn установлен	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Ничего не происходит.

Пример 1: Если маскированное *value_1* равно маскированному *value_2*, *light_1* устанавливается. Если маскированное *value_1* не равно маскированному *value_2*, *light_1* сбрасывается. В этом примере показано, что маскированные значения равны. 0 в маске запрещает инструкции сравнивать соответствующий бит (в примере помечены x).

value_1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1
mask_1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
Masked *value_1*	0	1	0	1	0	1	0	1	1	1	1	1	x	x	x	x

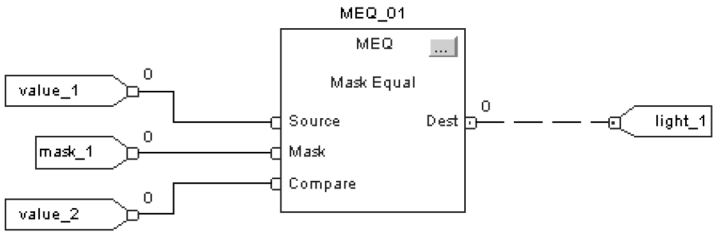
value_2	0	1	0	1	0	1	0	1	1	1	1	1	0	0	0	0
mask_1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
Masked *value_2*	0	1	0	1	0	1	0	1	1	1	1	1	x	x	x	x

Релейная логика

Структурированнный текст

```
light_1 := ((value_1 AND mask_1)=(value_2 AND mask_2));
```

Функциональный блок

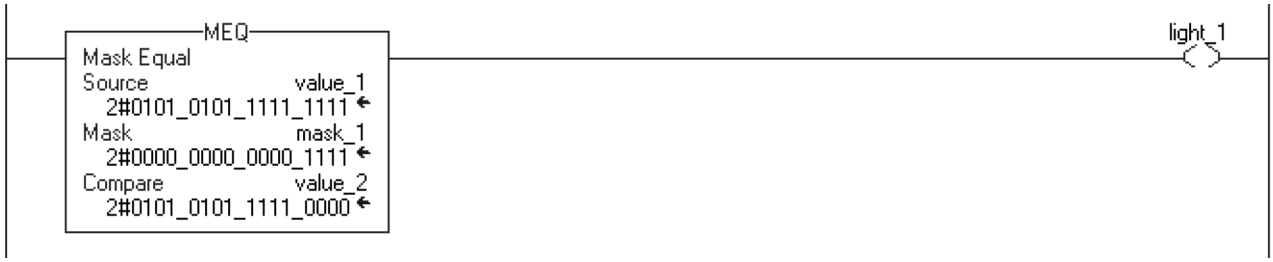


Пример 2: Если маскированное *value_1* равно маскированному *value_2*, *light_1* устанавливается. Если маскированное *value_1* не равно маскированному *value_2*, *light_1* сбрасывается. В этом примере показано, что маскированные значения не равны. 0 в маске запрещает инструкции сравнивать соответствующий бит (в примере помечены x).

value_1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1
mask_1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
Masked *value_1*	x	x	x	x	x	x	x	x	x	x	x	x	1	1	1	1

value_2	0	1	0	1	0	1	0	1	1	1	1	1	0	0	0	0
mask_1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
Masked *value_2*	x	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0

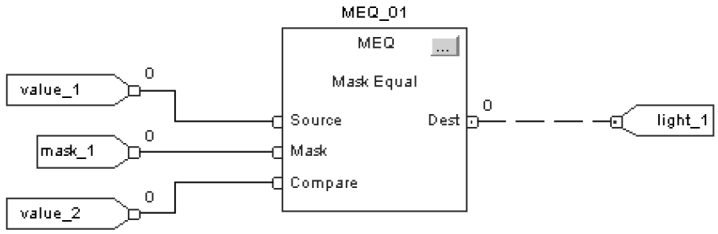
Релейная логика



Структурированнный текст

```
light_1 := ((value_1 AND mask_1)=(value_2 AND mask_2));
```

Функциональный блок

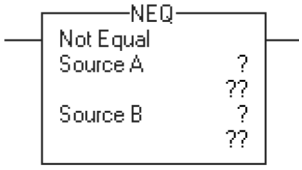


Not Equal to (NEQ) (Не равно)

Инструкция NEQ проверяет, не равно ли значение Source A значению Source B.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Source A	SINT INT **DINT** **REAL** **строка**	непосредственный тег	значение, проверяемое против Source B
Source B	SINT INT **DINT** **REAL** **строка**	непосредственный тег	значение, проверяемое против Source A

- Если вы ввели тег SINT или INT, то это значение преобразуется в DINT посредством дополнения знаком.
- Строковыми типами данных являются:
  - используемый по умолчанию тип данных STRING
  - все создаваемые вами новые типы строковых данных
- Чтобы выполнить проверку для символов строки, введите строковый тег как для Source A, так и для Source B.

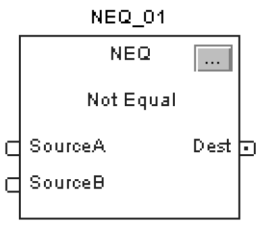
Структурированный текст

```
IF sourceA <> sourceB THEN
    <statements>;
```

Используйте расположенные подряд знаки меньше и больше «<>» в качестве оператора в выражении. Это выражение оценивает, не равно ли значение *sourceA* значению *sourceB*.

Информация о синтаксисе выражений структурированного текста содержится в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег NEQ	FBD_COMPARE	структура	структура NEQ

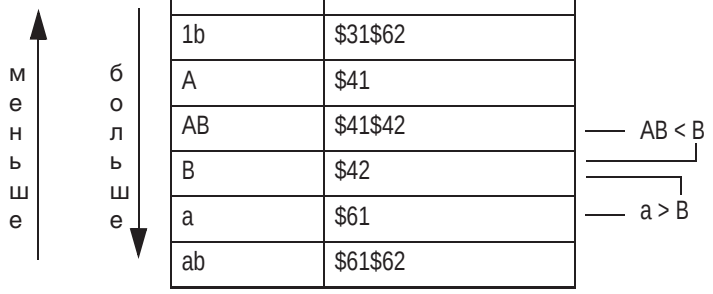
Структура FBD_COMPARE

Входной параметр:	**Тип данных:**	**Описание:**
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, инструкция не выполняется, и выходы не обновляются. По умолчанию установлен.
SourceA	REAL	Значение, проверяемое против SourceB. Допустимым значением является любое число с плавающей точкой.
SourceB	REAL	Значение, проверяемое против SourceA. Допустимым значением является любое число с плавающей точкой.
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	В результате выполнения инструкции получен допустимый результат.
Dest	BOOL	Результат выполнения инструкции. Это эквивалентно выходному условию цепочки для инструкции NEQ релейной логики.

Описание: Инструкция NEQ проверяет, не равно ли значение Source A значению Source B.

При сравнении строк:

- Строки не равны, если их символы не совпадают.
- Символы ASCII чувствительны к регистру. «A» верхнего регистра ($41) *не* равно «a» нижнего регистра ($61).



Символы ASCII	**Шестнадцатеричный код**
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

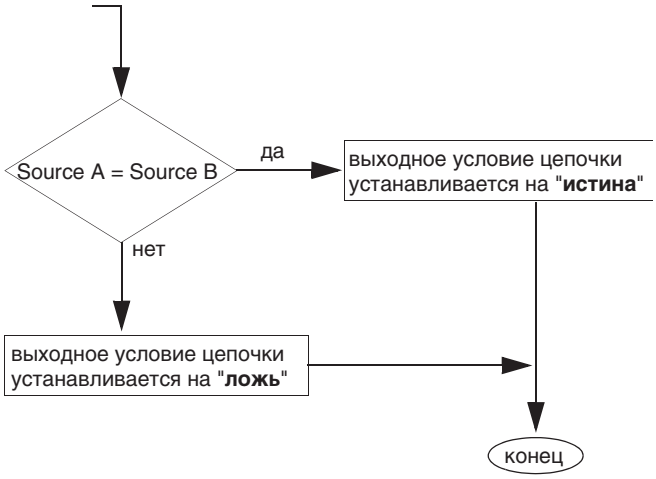
Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки «истина»	



постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Ничего не происходит.
первое сканирование инструкции	Ничего не происходит.
первое выполнение инструкции	Ничего не происходит.
EnableIn сброшен	EnableOut сбрасывается
EmableIn установлен	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Ничего не происходит.

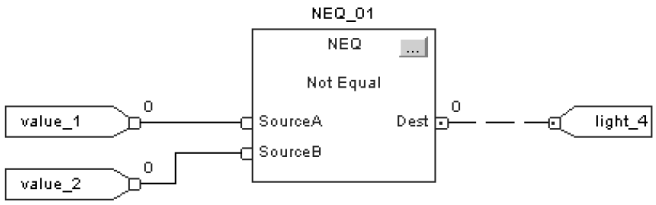
Пример: Если *value_1* не равно *value_2*, *light_4* устанавливается. Если *value_1* равно *value_2*, *light_4* сбрасывается.

Релейная логика

Структурированный текст

```
light_4 := (value_1 <> value_2);
```

Функциональный блок



Примечания:

Инструкции вычислений/ математических операций

(CPT, ADD, SUB, MUL, DIV, MODE, SQR, SQRT, NEG, ABS)

Введение

Инструкции вычислений/математических операций анализируют арифметические операции посредством выражения или определенной арифметической инструкции.

Если вы хотите:	Исполь-зуйте эту инструкцию:	Имеющуюся в этих языках:	См. стр.
проанализировать выражение	CPT	релейная логика структурированный текст(1)	5-2
скложить два значения	ADD	релейная логика структурированный текст(2) функциональный блок	5-6
вычесть одно значение из другого	SUB	релейная логика структурированный текст(2) функциональный блок	5-9
перемножить два значения	MUL	релейная логика структурированный текст(2) функциональный блок	5-12
разделить одно значение на другое	DIV	релейная логика структурированный текст(2) функциональный блок	5-15
определить остаток после деления одного значения на другое	MOD	релейная логика структурированный текст(2) функциональный блок	5-19
вычислить квадратный корень из значения	SQR SQRT(3)	релейная логика структурированный текст функциональный блок	5-23
присвоить значению противоположный знак	NEG	релейная логика структурированный текст(2) функциональный блок	5-26
определить абсолютную величину выражения	ABS	релейная логика структурированный текст функциональный блок	5-29

(1) Не существует эквивалентной инструкции для структурированного текста. Используйте другие средства программирования структурированного текста для достижения таких же результатов. Смотрите описание инструкции.

(2) Не существует эквивалентной инструкции для структурированного текста. Используйте оператор в выражении.

(3) Только структурированный текст.

Вы можете смешивать типы данных, но это может привести к потере точности и ошибке округления, и для выполнения инструкции потребуется больше времени. Проверьте бит S:V, чтобы убедиться, отброшена ли у полученного результата дробная часть.

Для инструкций релейной логики, **жирный** шрифт типов данных означает, что это оптимальные типы данных. Инструкция выполняется быстрее и требует меньший объем памяти, если все операнды инструкции используют один и тот же оптимальный тип данных, обычно DINT или REAL.

Compute (CPT) (Вычисление)

Инструкция CPT выполняет арифметические операции, которые вы задаете в выражении.

Операнды: Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Destination	SINT INT **DINT** **REAL**	тег	тег для хранения результата
Выражение	SINT INT **DINT** **REAL**	непосредственный тег	выражение, состоящее из тегов и/или непосредственных значений, разделенных операторами
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		

Структурированный текст

В структурированном тексте инструкция CPT отсутствует, но можно получить тот же результат посредством присваивания и выражений.

Информацию о синтаксисе операций присваивания и выражений в структурированном тексте можно найти в Приложении C.

```
destination := numeric_expresion;
```

Описание:

Инструкция CPT выполняет арифметические операции, которые вы задаете в выражении. Когда инструкция CPT разрешена, она анализирует выражение и помещает результат в Destination (приемник).

Инструкция CPT выполняется немного медленнее и использует больше памяти, чем остальные инструкции вычислений/ математических операций. Преимущество инструкции CPT в том, что она позволяет вам вводить сложные выражения с помощью одной инструкции.

ПОДСКАЗКА Для выражения *не существует предела* по длине.

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

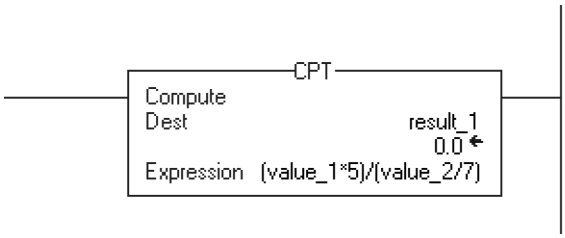
Условия ошибки:

отсутствуют

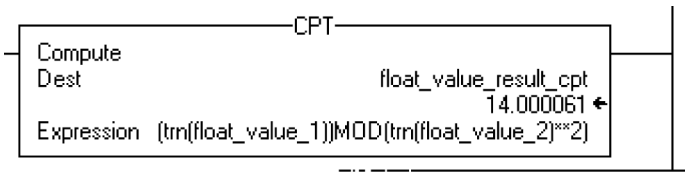
Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Инструкция анализирует Expression (выражение) и помещает результат в Destination (приемник). Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример 1: Когда инструкция CPT разрешена, она анализирует *value_1*, умноженное на 5, делит этот результат на результат деления *value_2* на 7, и помещает конечный результат в *result_1*.



Пример 2: Когда инструкция CPT разрешена, она отбрасывает дробную часть *float_value_1* и *float_value_2*, возводит усеченное *float_value_2* в квадрат, делит усеченное *float_value_1* на этот результат и сохраняет остаток от деления в *float_value_result_cpt*.



Допустимые операторы

Оператор:	Описание:	Оптимальный:
+	сложение	DINT, REAL
-	вычитание/отрицание	DINT, REAL
*	умножение	DINT, REAL
/	деление	DINT, REAL
**	степень (x в степени y)	DINT, REAL
ABS	абсолютная величина	DINT, REAL
ACS	арккосинус	REAL
AND	побитовое «И»	DINT
ASN	арксинус	REAL
ATN	арктангенс	REAL
COS	косинус	REAL
DEG	преобразование радиан в градусы	DINT, REAL
FRD	преобразование BCD в целые числа	DINT

Оператор:	Описание:	Оптимальный:
LN	натуральный логарифм	REAL
LOG	логарифм с основанием 10	REAL
MOD	деление по модулю	DINT, REAL
NOT	побитовое дополнение	DINT
OR	побитовое «ИЛИ»	DINT
RAD	преобразование градусов в радианы	DINT, REAL
SIN	синус	REAL
SQR	квадратный корень	DINT, REAL
TAN	тангенс	REAL
TOD	преобразование целых чисел в BCD	DINT
TRN	отбросить дробную часть	DINT, REAL
XOR	побитовое «исключающее ИЛИ»	DINT

Форматирование выражений

Для каждого оператора, который вы используете в выражении, вы должны предусмотреть один или два операнда (теги или непосредственные значения). Используйте следующую таблицу, чтобы отформатировать операторы и операнды внутри выражения:

Для операторов, которые работают:	Используйте этот формат:	Примеры:
с одним операндом	оператор(операнд)	ABS(*tag_a*)
с двумя операндами	операнд_a оператор операнд_b	· *tag_b* + 5 · *tag_c* AND *tag_d* · (*tag_e* ** 2) MOD · (*tag_f* / *tag_g*)

Задание порядка операции

Операции, которые вы записываете в выражение, выполняются инструкцией в установленном порядке, не обязательно в том порядке, в котором вы их записываете. Вы можете изменить порядок операции, поместив элементы в скобки и принудив тем самым инструкцию выполнять операцию в скобках раньше других операций.

Операции, имеющие один и тот же порядок, выполняются слева направо.

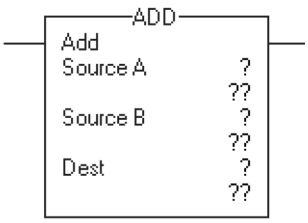
Порядок:	Операция:
1.	( )
2.	ABS, ACS, ASN, ATN, COS, DEG, FRD, LN, LOG, RAD, SIN, SQR, TAN, TOD, TRN
3.	**
4.	– (отрицание), NOT
5.	*, /, MOD
6.	– (вычитание), +
7.	AND
8.	XOR
9.	OR

Add (ADD) (Сложение)

Инструкция ADD складывает Source A (источник A) и Source B (источник B) и помещает результат в Destination (приемник).

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:
Source A	SINT INT **DINT** **REAL**	непосредственный тег	значение для сложения с Source B
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Source B	SINT INT **DINT** **REAL**	непосредственный тег	значение для сложения с Source A
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Destination	SINT INT **DINT** **REAL**	тег	тег для хранения результата

Структурированный текст

```
dest := sourceA + sourceB;
```

Используйте знак плюс «+» в качестве оператора в выражении. Это выражение складывает *sourceA* и *sourceB* и сохраняет результат в *dest*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег ADD	FBD_MATH	структура	структура ADD

Структура FBD_MATH

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
SourceA	REAL	Значение для сложения с SourceB. Допустимое значение = любое значение с плавающей точкой
SourceB	REAL	Значение для сложения с SourceA. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание: Инструкция ADD складывает Source A (источник A) и Source B (источник B) и помещает результат в Destination (приемник).

Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

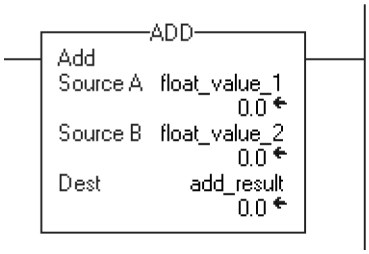
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Destination = Source A + Source B Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Сложение *float_value_1* и *float_value_2* и помещение результата в *add_result*.

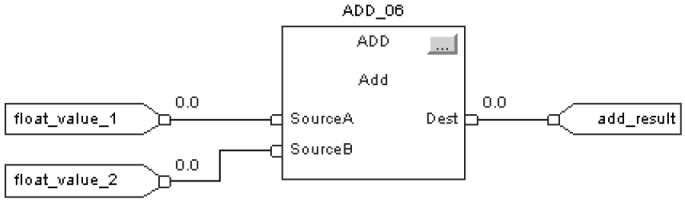
Релейная логика



Структурированный текст

```
add_result := float_value_1 + float_value_2;
```

Функциональный блок

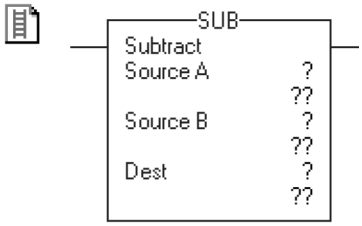


Subtract (SUB) (Вычитание)

Инструкция SUB вычитает Source B (источник B) из Source A (источника A) и помещает результат в Destination (приемник).

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:
Source A	SINT INT **DINT** **REAL**	непосредственный тег	значение, из которого вычитается Source B
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Source B	SINT INT **DINT** **REAL**	непосредственный тег	значение, которое вычитается из Source A
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Destination	SINT INT **DINT** **REAL**	тег	тег для хранения результата

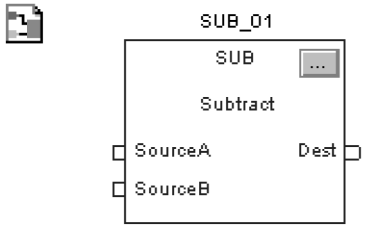
Структурированный текст

```
dest := sourceA - sourceB;
```

Используйте знак минус «–» в качестве оператора в выражении. Это выражение вычитает *sourceB* из *sourceA* и сохраняет результат в *dest.*

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег SUB	FBD_MATH	структура	структура SUB

Структура FBD_MATH

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
SourceA	REAL	Значение, из которого вычитается SourceB. Допустимое значение = любое значение с плавающей точкой
SourceB	REAL	Значение для сложения с SourceA. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание: Инструкция SUB вычитает Source B (источник B) из Source A (источника A) и помещает результат в Destination (приемник).

Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

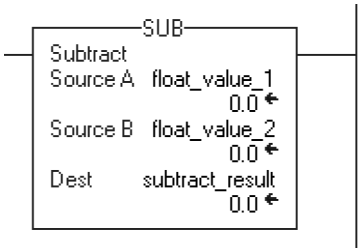
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Destination = Source B – Source A Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Вычитание *float_value_2* из *float_value_1* и помещение результата в *substract_result*.

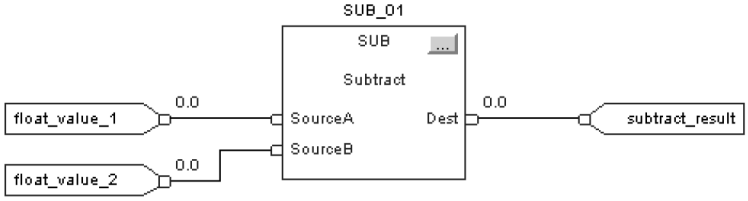
Релейная логика



Структурированный текст

```
subtract_result := float_value_1 - float_value_2;
```

Функциональный блок

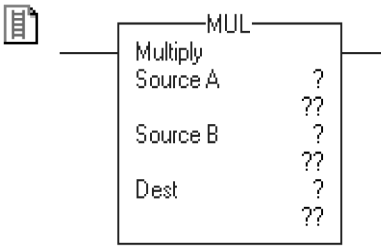


Multiply (MUL) (Умножение)

Инструкция MUL умножает Source A (источник A) на Source B (источник B) и помещает результат в Destination (приемник).

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:
Source A	SINT INT **DINT** **REAL**	непосредственный тег	значение множимого
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Source B	SINT INT **DINT** **REAL**	непосредственный тег	значение множителя
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Приемник Destination	SINT INT **DINT** **REAL**	тег	тег для хранения результата

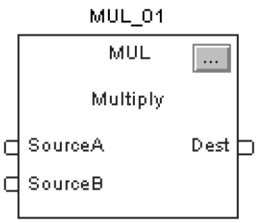
Структурированный текст

```
dest := sourceA * sourceB;
```

Используйте знак умножения «*» в качестве оператора в выражении. Это выражение умножает *sourceA* на *sourceB* и хранит результат в *dest*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег MUL	FBD_MATH	структура	структура MUL

Структура FBD_MATH

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
SourceA	REAL	Значение множимого. Допустимое значение = любое значение с плавающей точкой
SourceB	REAL	Значение множителя. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание: Инструкция MUL умножает Source A (источник A) на Source B (источник B) и помещает результат в Destination (приемник).

Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

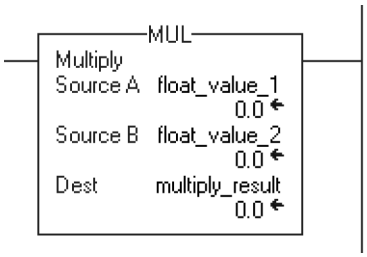
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Destination = Source B Ч Source A Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Умножение *float_value_1* на *float_value_2* и помещение результата в *multiply_result*.

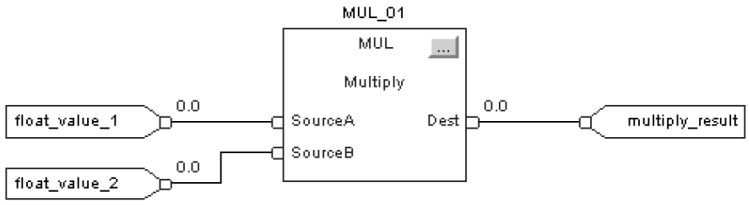
Релейная логика



Структурированный текст

```
multiply_result := float_value_1 * float_value_2;
```

Функциональный блок

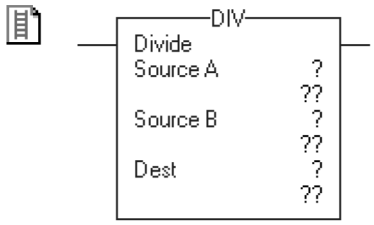


Divide (DIV) (Деление)

Инструкция DIV делит Source A (источник A) на Source B (источник B) и помещает результат в Destination (приемник).

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:
Source A	SINT INT **DINT** **REAL**	непосредственный тег	значение делимого
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Source B	SINT INT **DINT** **REAL**	непосредственный тег	значение делителя
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Destination	SINT INT **DINT** **REAL**	тег	тег для хранения результата

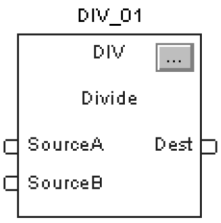
Структурированный текст

```
dest := sourceA / sourceB;
```

Используйте знак деления «/» в качестве оператора в выражении. Это выражение делит *sourceA* на *sourceB* и хранит результат в *dest*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег DIV	FBD_MATH	структура	структура DIV

Структура FBD_MATH

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
SourceA	REAL	Значение делимого. Допустимое значение = любое значение с плавающей точкой
SourceB	REAL	Значение делителя. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание: Если Destination (приемник) *не* использует тип данных REAL, инструкция обращается с дробной частью результата следующим образом:

Если Source A:	То дробная часть результата:	Пример:		
и Source B не используют REAL	отбрасывается	Source A	DINT	5
		Source B	DINT	3
		Destination	DINT	1
или Source B использует REAL	округляется	Source A	REAL	5.0
		Source B	DINT	3
		Destination	DINT	2

- Если Source B (делитель) равен нулю:
- происходит неосновная ошибка
  - Тип 4: программная ошибка
  - Код 4: арифметическое переполнение
- приемник устанавливается следующим образом:

Если Source B равен 0 и:	И приемник:	А результат:	То приемник устанавливается на:
все операнды целые числа (SINT, INT или DINT)	→	→	Source A
хотя бы один операнд REAL	SINT, INT или DINT	положительный	-1
		отрицательный	0
	REAL	положительный	1.$ (плюс бесконечность)
		отрицательный	-1.$ (минус бесконечность)

Чтобы выявить возможность операции деления на 0, проверьте бит неосновной ошибки (S:MINOR). Обратитесь к *Logix5000 Controllers Common Procedures* (*Logic5000 Общие процедуры контроллеров*), публикация 1756-PM001.

Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

Условия ошибки:

Неосновная ошибка произойдет, если	Тип ошибки:	Код ошибки:
делитель равен нулю	4	4

Выполнение:

Релейная логика

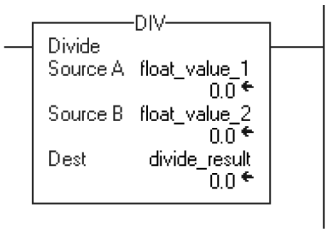
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Destination = Source A / Source B Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Деление *float_value_1* на *float_value_2* и помещение результата в *divide_result*.

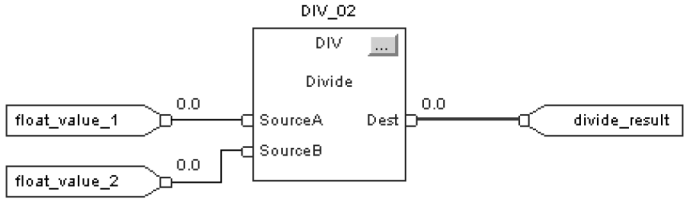
Релейная логика



Структурированный текст

```
divide_result := float_value_1 / float_value_2;
```

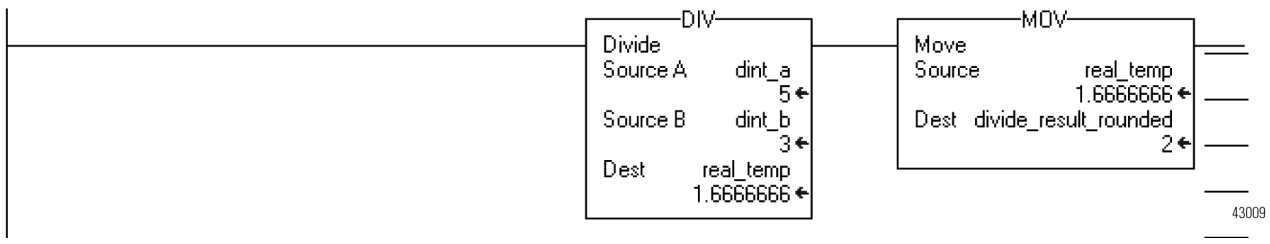
Функциональный блок



Пример 2: Инструкции DIV и MOD работают вместе при делении двух целых чисел, округлении результата и помещении результата в целочисленный тег:

- Инструкция DIV делит *dint_a* на *dint_b*.
- Чтобы округлить результат, Destination (приемник) должен быть тегом REAL. (Если бы приемник был целочисленным тегом (SINT, INT или DINT), инструкция отбрасывала бы дробную часть результата.)
- Инструкция MOV перемещает округленный результат (*real_temp*) из DIV в *divide_result_rounded*.
- Поскольку *divide_result_rounded* - тег DINT, значение из *real_temp* округляется и помещается в приемник DINT.

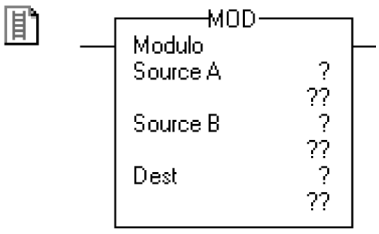
Релейная логика



Modulo (MOD) (Остаток от деления)

Инструкция MOD делит Source A (источник A) на Source B (источник B) и помещает остаток в Destination (приемник).

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Source A	SINT INT DINT REAL	непосредственный тег	значение делимого
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Source B	SINT INT DINT REAL	непосредственный тег	значение делителя
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Destination	SINT INT DINT REAL	тег	тег для хранения результата

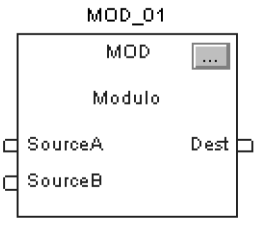
Структурированный текст

```
dest := sourceA MOD sourceB;
```

Используйте MOD в качестве оператора в выражении. Это выражение делит *sourceA* на *sourceB* и сохраняет остаток в *dest*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег MOD	FBD_MATH	структура	структура MOD

Структура FBD_MATH

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
SourceA	REAL	Значение делимого. Допустимое значение = любое значение с плавающей точкой
SourceB	REAL	Значение делителя. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание: Если Source B (делитель) равен нулю:

- происходит неосновная ошибка
  - Тип 4: программная ошибка
  - Код 4: арифметическое переполнение
- приемник устанавливается следующим образом:

Если Source B равен 0 и:	И приемник:	А результат:	То приемник устанавливается на:
все операнды целые числа (SINT, INT или DINT)	→	→	Source A
хотя бы один операнд REAL	SINT, INT или DINT	положительный	-1
		отрицательный	0
	REAL	положительный	1.$ (плюс бесконечность)
		отрицательный	-1.$ (минус бесконечность)

Чтобы выявить возможную операцию деления на 0, проверьте бит неосновной ошибки (S:MINOR). Обратитесь к *Logix5000 Controllers Common Procedures* (*Logic5000 Общие процедуры контроллеров*), публикация 1756-PM001.

Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

Условия ошибки:

Неосновная ошибка произойдет, если	Тип ошибки:	Код ошибки:
делитель равен нулю	4	4

Выполнение:

Релейная логика

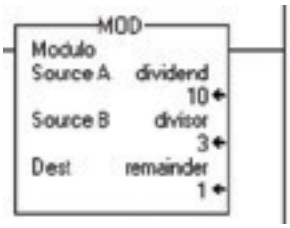
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Destination = Source A – (TRN (Source A / Source B) * Source B) Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Деление *dividend* на *divisor* и помещение остатка в *reminder*. В этом примере 10 содержит три раза по три, остаток – 1.

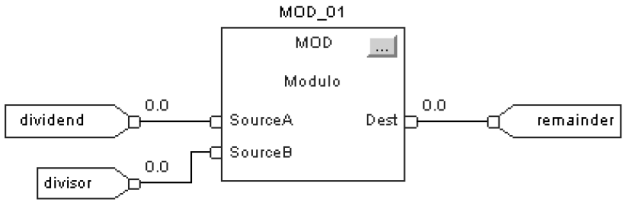
Релейная логика



Структурированный текст

```
remainder := dividend MOD divisor;
```

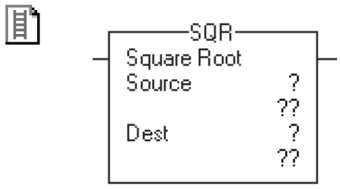
Функциональный блок



Square Root (SQR) (Квадратный корень)

Инструкция SQR извлекает квадратный корень из Source (источника) и помещает результат в Destination (приемник).

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Source	SINT INT **DINT** **REAL**	непосредственный тег	извлечение квадратного корня из этого значения
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Destination	SINT INT **DINT** **REAL**	тег	тег для хранения результата

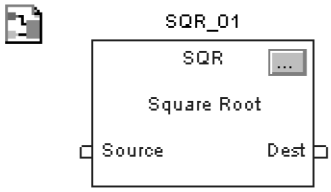
Структурированный текст

```
dest := SQRT(source);
```

Используйте SQRT в качестве функции. Это выражение извлекает квадратный корень из *source* и сохраняет результат в *dest.*

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег SQR	FBD_MATH_ADVANCED	структура	структура SQR

Структура FBD_MATH_ADVANCED

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Извлечение квадратного корня из этого значения. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание: Если Destination (приемник) *не* используют тип данных REAL, инструкция обращается с дробной частью результата следующим образом:

Если Source:	То дробная часть результата:	Пример:		
не используют REAL	отбрасывается	Source	DINT	3
		Destination	DINT	1
использует REAL	округляется	Source	REAL	3.0
		Destination	DINT	2

Если Source (источник) имеет отрицательное значение, инструкция берет абсолютную величину Source (источника) перед извлечением квадратного корня.

Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

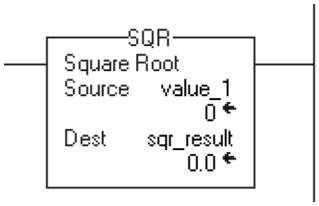
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	$Destination = \sqrt{Source}$ Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Извлечение квадратного корня из *value_1* и помещение результата в *sqr_result.*

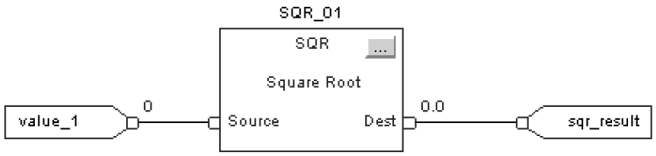
Релейная логика



Структурированный текст

```
sqr_result := SQRT(value_1);
```

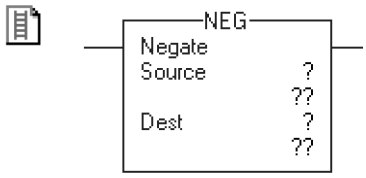
Функциональный блок



Negate (NEG) (Смена знака)

Инструкция NEG изменяет знак Source (источника) и помещает результат в Destination (приемник).

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Source	SINT	непосредственный	значение для отрицания
	INT	тег	
	DINT		
	REAL		
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Destination	SINT	тег	тег для хранения результата
	INT		
	DINT		
	REAL		

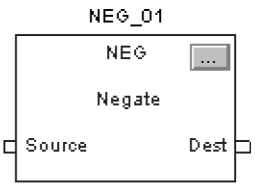
Структурированный текст

```
dest := -source;
```

Используйте знак минус «–» в качестве оператора в выражении. Это выражение изменяет знак *source* и сохраняет результат в *dest*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег NEG	FBD_MATH_ADVANCED	структура	структура NEG

Структура FBD_MATH_ADVANCED

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Значение для отрицания. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание:

Если вы отрицаете отрицательное значение, результат будет положительным. Если вы отрицаете положительное значение, результат будет отрицательным.

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

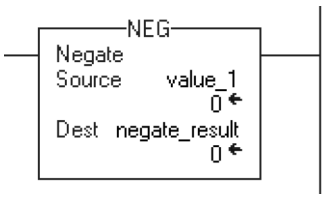
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Destination = 0 – Source Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Изменение знака *value_1* и помещение результата в *negate_result*.

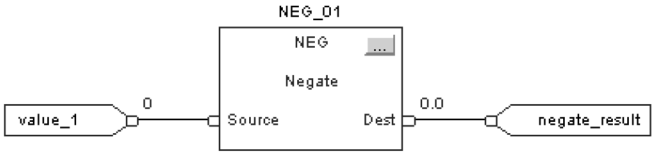
Релейная логика



Структурированный текст

```
negate_result := -value_1;
```

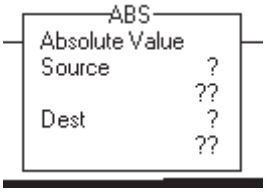
Функциональный блок



Absolute Value (ABS) (Абсолютная величина)

Инструкция ABS берет абсолютную величину Source (источника) и помещает результат в Destination (приемник).

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Source	SINT INT DINT REAL	непосредственный тег	значение, для которого берется абсолютная величина
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Destination	SINT INT DINT REAL	тег	тег для хранения результата

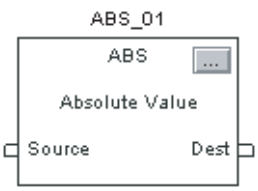
Структурированный текст

```
dest := ABS(source)
```

Используйте инструкцию ABS в качестве функции. Это выражение вычисляет абсолютную величину *source* и сохраняет результат в *dest*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег ABS	FBD_MATH_ADVANCED	структура	структура ABS

Структура FBD_MATH

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Значение, для которого берется абсолютная величина. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание:

Инструкция ABS берет абсолютную величину Source (источника) и помещает результат в Destination (приемник).

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

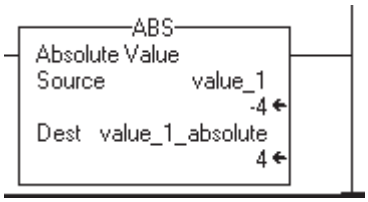
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Destination = | Source | Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Помещение абсолютного значения *value_1* в *value_1_absolute*. В этом примере абсолютным значением «–4» является положительное значение «4».

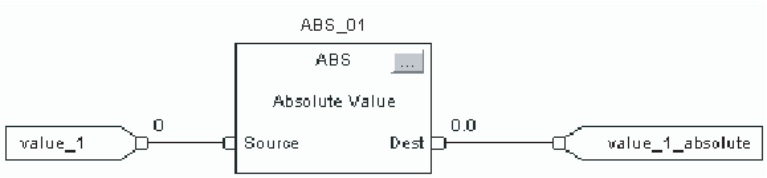
Релейная логика



Структурированный текст

```
value_1_absolute := ABS(value_1);
```

Функциональный блок



Примечания:

Инструкции перемещения/ логические инструкции

(MOV, MVM, BTD, MVMT, BTDT, CLR, SWPB, AND, OR, XOR, NOT, BAND, BOR, BXOR, BNOT)

Введение

Вы можете смешивать типы данных, но это может привести к потере точности и ошибке округления, и для выполнения инструкции потребуется больше времени. Проверьте бит S:V, чтобы убедиться, отброшена ли у полученного результата дробная часть.

Для инструкций релейной логики, **жирный** шрифт типов данных означает, что это оптимальные типы данных. Инструкция выполняется быстрее и требует меньший объем памяти, если все операнды инструкции используют один и тот же оптимальный тип данных, обычно DINT или REAL.

Инструкции перемещения модифицируют и перемещают биты.

Если вы хотите:	Используйте эту инструкцию:	Имеющуюся в этих языках:	См. стр.
скопировать значение	MOV	релейная логика структурированный текст(1)	6-3
скопировать определенную часть целого числа	MVM	релейная логика	6-5
копировать определенную часть целого числа в функциональном блоке	MVMT	структурированный текст функциональный блок	6-8
переместить биты внутри целого числа или между целыми числами	BTD	релейная логика	6-11
переместить биты внутри целого числа или между целыми числами в функциональном блоке	BTDT	структурированный текст функциональный блок	6-14
обнулить значение	CLR	структурированный текст(1) функциональный блок	6-17
перегруппировать байты тега INT, DINT или REAL	SWPB	релейная логика структурированный текст	6-19

(1) Не существует эквивалентной инструкции для структурированного текста. Используйте другие средства программирования структурированного текста для достижения таких же результатов. Смотрите описание инструкции.

Логические инструкции выполняют операции с битами.

Если вы хотите:	**Используйте эту инструкцию:**	**Имеющуюся в этих языках:**	**См. стр.**
выполнить операцию «побитовое И»	Bitwise AND &(1)	релейная логика структурированный текст(2) функциональный блок	6-23
выполнить операцию «побитовое ИЛИ»	Bitwise OR	релейная логика структурированный текст(2) функциональный блок	6-26
выполнить операцию «побитовое исключающее ИЛИ»	Bitwise XOR	релейная логика структурированный текст(2) функциональный блок	6-29
выполнить операцию «побитовое НЕ»	Bitwise NOT	релейная логика структурированный текст(2) функциональный блок	6-32
выполнить операцию «логическое И» для максимум восьми булевых входов.	Boolean AND (BAND)	структурированный текст(2) функциональный блок	6-35
выполнить операцию «логическое ИЛИ» для максимум восьми булевых входов.	Boolean OR (BOR)	структурированный текст(2) функциональный блок	6-38
выполнить операцию «исключающее ИЛИ» для максимум двух булевых входов.	Boolean Exclusive OR (BXOR)	структурированный текст(2) функциональный блок	6-41
дополнить булев вход.	Boolean NOT (BNOT)	структурированный текст(2) функциональный блок	6-44

(1) Только структурированный текст.

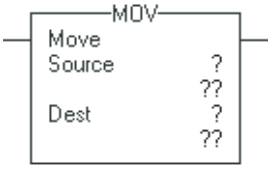
(2) В структурированном тексте операции AND, OR, XOR и NOT могут быть побитовыми или логическими.

Move (MOV) (Перемещение)

Инструкция MOV копирует Source (источник) в Destination (приемник). Source сохраняется без изменений.

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:
Source	SINT INT **DINT** **REAL**	непосредственный тег	значение для перемещения (копирования)
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Destination	SINT INT **DINT** **REAL**	тег	тег для хранения результата

Структурированный текст

```
dest := source;
```

Используйте операцию присваивания ":=" с выражением. Эта операция присваивания перемещает значение *source* в *dest*.

Информацию о синтаксисе операций присваивания и выражений в структурированном тексте можно найти в разделе «Программирование структурированного текста».

Описание:

Инструкция MOV копирует Source (источник) в Destination (приемник). Source остается без изменений.

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

Условия ошибки:

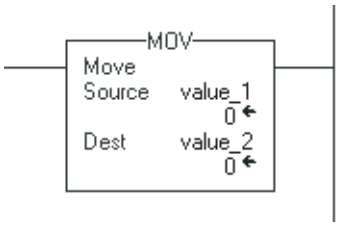
отсутствуют

Выполнение:

Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Инструкция копирует Source в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример: Перемещение данных *value_1* в *value_2*.

Релейная логика



Структурированный текст

```
value_2 := value _1;
```

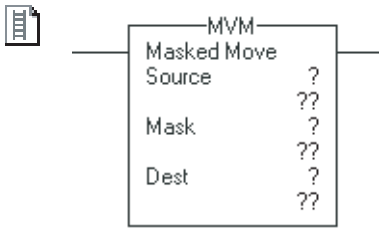
Masked Move (MVM) (Маскированное перемещение)

Инструкция MVM копирует Source (источник) в Destination (приемник), разрешая маскировать определенные части данных.

В структурированном тексте и функциональном блоке этой инструкции соответствует инструкция MVMT (см. стр. 6-8).

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:
Source	SINT INT **DINT**	непосредственный тег	значение для перемещения
	Тег SINT или INT преобразуется в значение DINT посредством заполнения нулями.		
Mask	SINT INT **DINT**	непосредственный тег	показывает, какие биты блокировать, а какие пропускать
	Тег SINT или INT преобразуется в значение DINT посредством заполнения нулями.		
Destination	SINT INT **DINT** REAL	тег	тег для хранения результата

Структурированный текст

```
dest := (Dest AND NOT (Mask))
    OR (Source AND Mask);
```

Эта инструкция имеется в структурированном тексте в виде MVMT. Или вы можете скомбинировать побитовую логику внутри выражения и присвоить результат приемнику. Это выражение выполняет маскированное перемещение для *Source*.

Информацию о синтаксисе операций присваивания и выражений в структурированном тексте можно найти в разделе «Программирование структурированного текста».

Описание:

Инструкция MVM использует Mask (маску), чтобы блокировать или пропускать биты данных Source (источника). «1» в маске означает, что бит данных пропускается. «0» в маске означает, что бит данных блокируется.

Если вы смешиваете типы целочисленных данных, инструкция будет заполнять старшие биты меньших значений целочисленных типов данных нулями так, чтобы они имели тот же размер, что и наибольшее значение этого типа данных.

Ввод непосредственного значения маски

Когда вы вводите значение маски, программное обеспечение по умолчанию воспринимает ее как десятичное значение. Если вы хотите ввести маску, используя другой формат, снабдите значение соответствующим префиксом.

Префикс:	Описание:
16#	шестнадцатеричный например, 16#OFOF
8#	восьмеричный например, 8#16
2#	двоичный например, 2#00110011

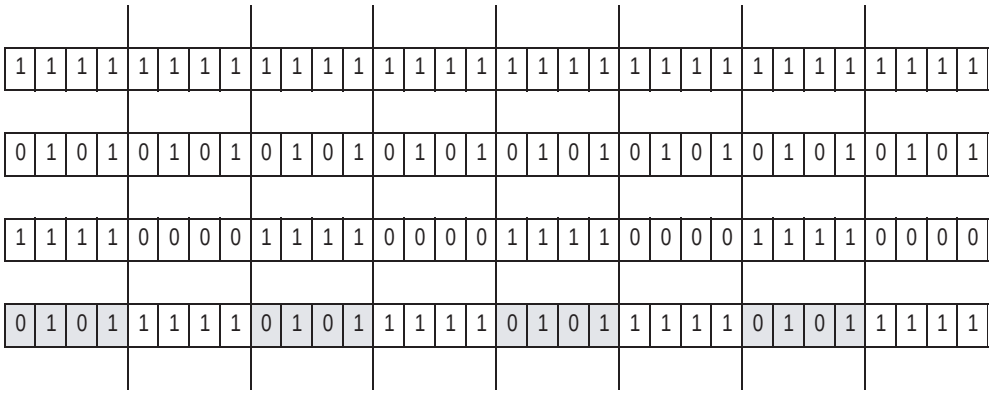
Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

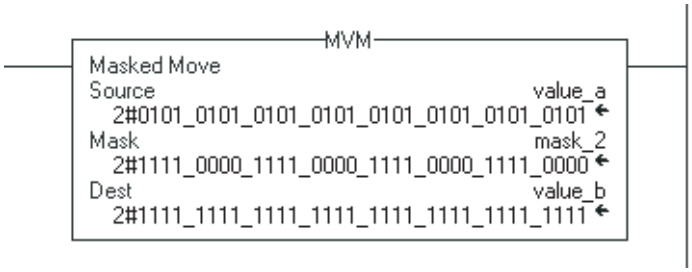
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Инструкция пропускает Source через Mask и копирует результат в Destination. Немаскированные биты в Destination остаются без изменений. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример: Копирование данных из *value_a* в *value_b* с разрешением маскирования данных (0 маскирует данные в *value_a*).



Закрашенные ячейки показывают биты, которые изменились в *value_b*.

Релейная логика



Структурированный текст

```
value_b := (value_b AND NOT (mask_2)) OR
     (value_a AND mask_2);
```

Masked Move with Target (MVMT) (Маскированное перемещение с целевым значением)

Инструкция MVMT сначала копирует Target (целевое значение) в Destination (приемник). Затем эта инструкция сравнивает маскированный Source (источник) с Destination и вносит требуемые изменения в Destination. Target и Source сохраняются без изменений.

В релейной логике этой инструкции соответствует инструкция MVM (см. стр. 6-5).

Операнды:

```
MVMT(MVMT_tag);
```

Структурированный текст

Переменная:	Тип:	Формат:	Описание:
Тег MVMT	FBD_MASKED_MOVE	структура	структура MVMT

Функциональный блок

Операнд:	Тип:	Формат:	Описание:
Тег MVMT	FBD_MASKED_MOVE	структура	структура MVMT

Структура FDB_MASKED_MOVE

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	**Функциональный блок:** Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. Если этот параметр установлен, инструкция выполняется. По умолчанию параметр установлен. **Структурированный текст:** Ничего не происходит. Инструкция выполняется.
Source	DINT	Входное значение для перемещения в Destination на основе значения Mask. Допустимое значение = любое целое число
Mask	DINT	Маска битов для перемещения из Source в Dest. Все биты, установленные на единицу, вызывают перемещение соответствующих битов из Source в Dest. Все биты, устанавленные на ноль, вызывают отсутствие перемещения соответствующих битов из Source в Dest. Допустимое значение = любое целое число
Target	DINT	Входное значение для перемещения в Dest, предшествующего перемещению битов Source посредством Mask. Допустимое значение = любое целое число
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	DINT	Результат инструкции маскированного перемещения. Арифметические флаги состояния устанавливаются для этого выхода.

Описание: Когда инструкция MVMT разрешена, она использует Mask (маску), чтобы блокировать или пропускать биты данных Source (источника). «1» в маске означает, что бит данных пропускается. «0» в маске означает, что бит данных блокируется.

Если вы смешиваете типы целочисленных данных, инструкция будет заполнять старшие биты меньших значений данных целочисленного типа нулями таким образом, чтобы они имели тот же размер, что и наибольшее значение этого типа данных.

Ввод непосредственного значения маски посредством Input Reference (входной ссылки)

При вводе значения маски программное обеспечение по умолчанию воспринимает его как десятичное. Если вы хотите ввести маску, используя другой формат, снабдите значение соответствующим префиксом.

Префикс:	Описание:
16#	шестнадцатеричный например, 16#OFOF
8#	восьмеричный например, 8#16
2#	двоичный например, 2#00110011

Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

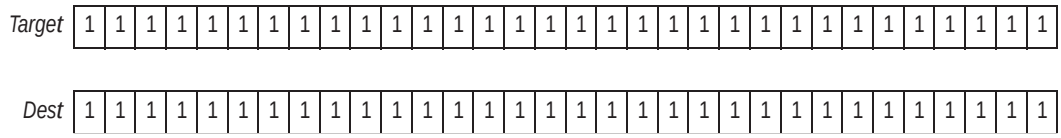
Условия ошибки: отсутствуют

Выполнение:

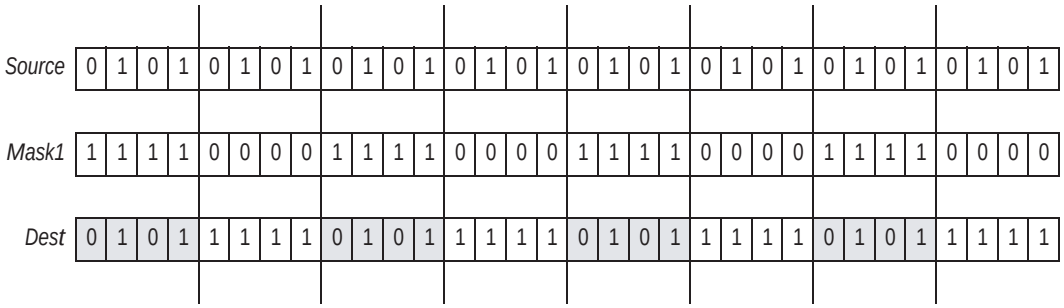
Условие:	Действие функционального блока:	Действие структурированного текста:
предварительное сканирование	Никакого действия не производится.	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается, инструкция не выполняется, а выходные данные не обновляются.	не применимо
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.	EnableIn всегда установлен. Инструкция выполняетсяю
постсканирование	Никакого действия не производится.	Никакого действия не производится.

Пример:

1. Копирование Target (целевого значения) в Destination (приемник).



2. Маскирование Source (источника) и сравнение его с Dest. В Dest вносятся все необходимые изменения. Source и Target сохраняются без изменений. 0 в маске запрещает инструкции выполнять сравнение этого бита (в примере показано как x).

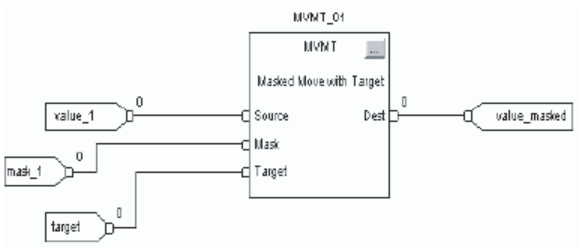


Закрашенные ячейки показывают измененные биты

Структурированный текст

```
MVMT_01.Source := value_1;
MVMT_01.Mask := mask1;
MVMT_01.Target := target;

MVMT(MVMT_01);

value_masked := MVMT_01.Dest;
```

Функциональный блок



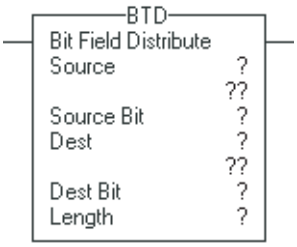
Bit Field Distribute (BTD) (Распределение битовых полей)

Инструкция BTD копирует заданные биты из Source (источника), осуществляет сдвиг этих битов на соответствующую позицию и записывает их в Destination (приемник).

В структурированном тексте и функциональном блоке этой инструкции соответствует инструкция BTDT (см. стр. 6-14).

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:
Source	SINT INT **DINT**	непосредственный тег	тег, содержащий биты для перемещения
	Тег SINT или INT преобразуется в значение DINT посредством заполнения нулями.		
Source bit	DINT	непосредственный (0-31 DINT) (0-15 INT) (0-7 SINT)	номер бита (наименьший номер бита), с которого начинается перемещение должен находиться в пределах допустимого диапазона для типа данных Source
Destination	SINT INT **DINT**	тег	тег, куда перемещаются биты
Destination bit	DINT	непосредственный (0-31 DINT) (0-15 INT) (0-7 SINT)	номер бита (наименьший номер бита), с которого начинается копирование битов из Source должен находиться в пределах допустимого диапазона для типа данных Destination
Length	DINT	непосредственный (1-32)	число битов для перемещения

Описание:

Когда инструкция BTD разрешена, она копирует группу битов из Source (источника) в Destination (приемник). Группа битов задается битом Source (наименьший номер бита в группе) и Length (длиной) (число битов для копирования). Бит Destination определяет бит с наименьшим номером, с которого необходимо начать копирование в Destination. Source остается без изменений.

Если длина битового поля превышает Destination, инструкция не сохраняет лишние биты. Никакие лишние биты не переносятся в следующее слово.

Если вы смешиваете типы целочисленных данных, инструкция будет заполнять старшие биты меньших значений данных целочисленного типа нулями таким образом, чтобы они имели тот же размер, что и наибольшее значение этого типа данных.

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

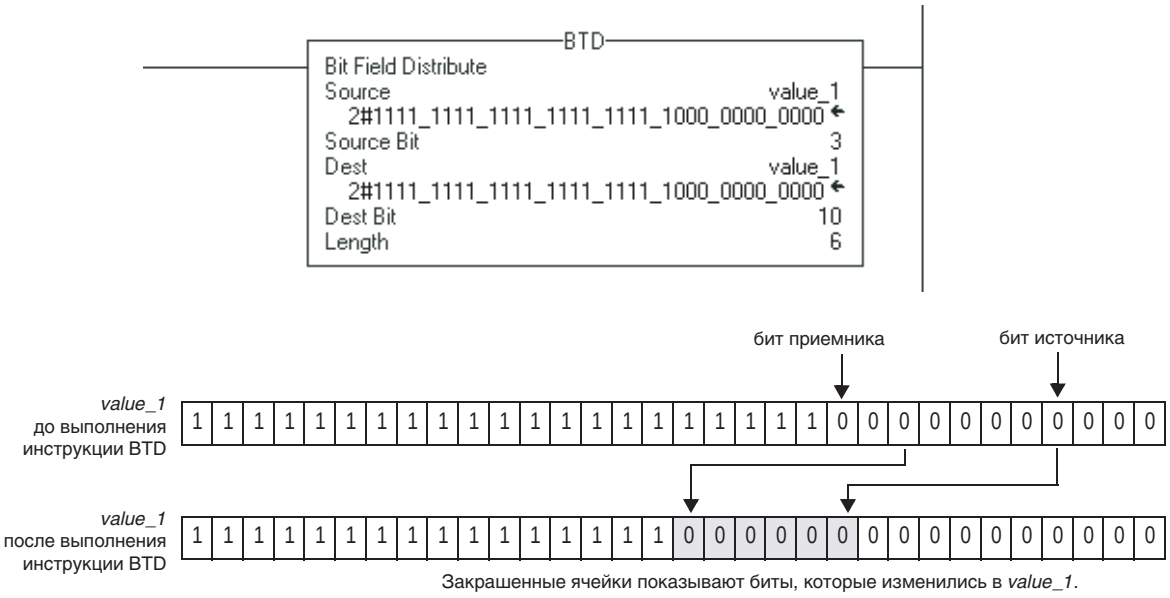
Условия ошибки:

отсутствуют

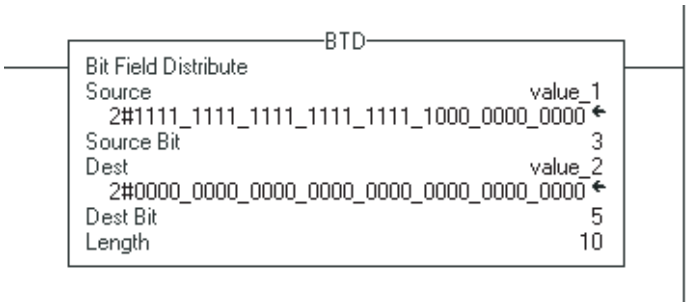
Выполнение:

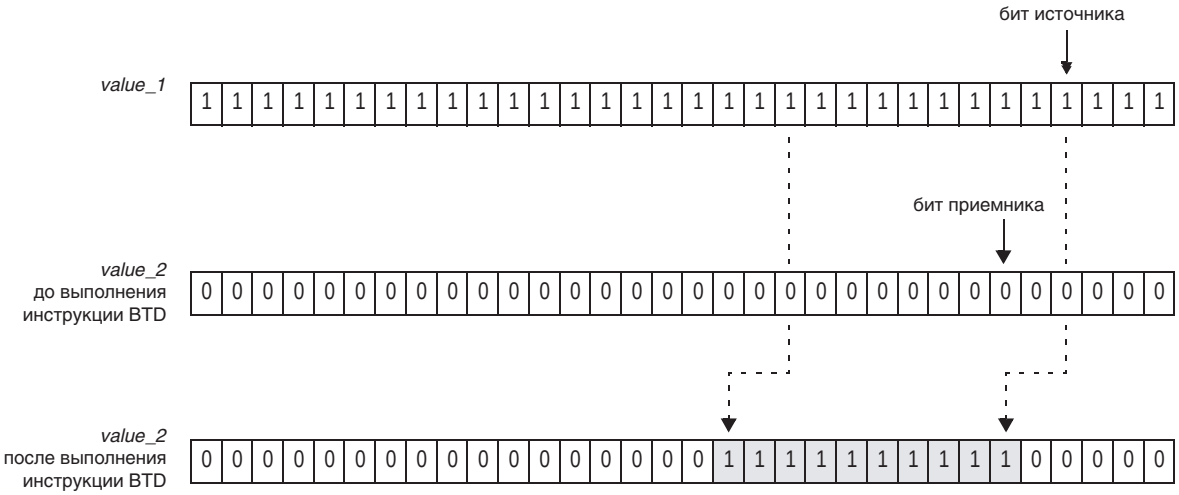
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Инструкция копирует и сдвигает биты Source в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример 1: Когда инструкция BTD разрешена, она перемещает биты внутри *value_1*.



Пример 2: Когда инструкция BTD разрешена, она перемещает 10 битов из *value_1* в *value_2*.





Закрашенные ячейки показывают биты, которые изменились в *value_2*.

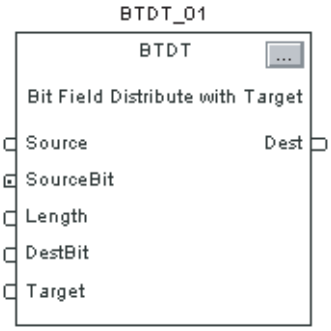
Bit Field Distribute with Target (BTDT) (Распределение битовых полей с целевым значением)

Инструкция BTDT сначала копирует Target (целевое значение) в Destination (приемник). Затем эта инструкция копирует заданные биты из Source (источника), сдвигает эти биты на соответствующую позицию и записывает их в Destination.

В релейной логике этой инструкции соответствует инструкция BTD (см. стр. 6-11).

Операнды:

```
BTDT(BTDT_tag);
```



Структурированный текст

Переменная:	Тип:	Формат:	Описание:
Тег BTDT	FBD_BIT_FIELD_DISTRIBUTE	структура	структура BTDT

Функциональный блок

Операнд:	Тип:	Формат:	Описание:
Тег BTDT	FBD_BIT_FIELD_DISTRIBUTE	структура	структура BTDT

Структура FDB_BIT_FIELD_DISTRIBUTE

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Функциональный блок: Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. Если этот параметр установлен, инструкция выполняется. По умолчанию параметр установлен. Структурированный текст: Ничего не происходит. Инструкция выполняется.
Source	DINT	Входное значение, содержащее биты для перемещения в Destination. Допустимое значение = любое целое число
SourceBit	DINT	Позиция бита в Source (наименьший номер бита, с которого начинается перемещение). Допустимое значение = 0-31
Length	DINT	Число битов для перемещения Допустимое значение = 1-32
DestBit	DINT	Позиция бита в Dest (наименьший номер бита, с которого начинается копирование) Допустимое значение = 0-31
Target	DINT	Входное значение для перемещения в Dest, предшествующего перемещению битов из Source. Допустимое значение = любое целое число
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	DINT	Результат операции по перемещению битов. Арифметические флаги состояния устанавливаются для этого выхода.

Описание: Когда инструкция BTD разрешена, она копирует группу битов из Source (источника) в Destination (приемник). Группа битов задается битом Source (наименьший номер бита в группе) и Length (длиной) (число битов для копирования). Бит Destination определяет бит с наименьшим номером, с которого необходимо начать копирование в Destination. Source остается без изменений.

Если длина битового поля превышает Destination, инструкция не сохраняет лишние биты. Никакие лишние биты не переносятся в следующее слово.

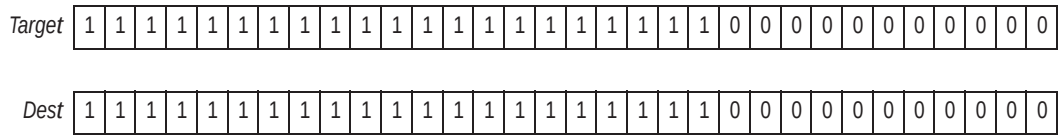
Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

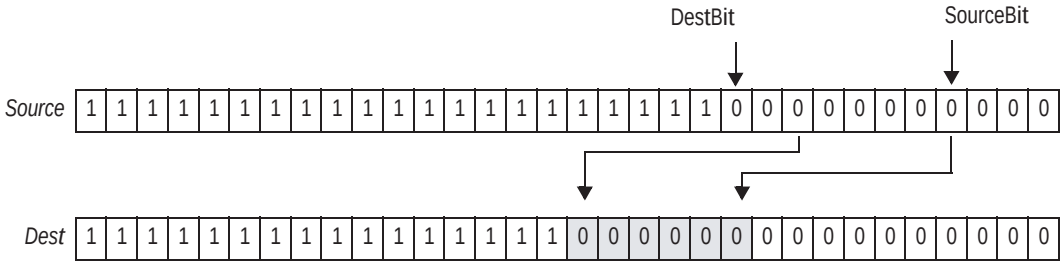
Выполнение:

Условие:	Действие функционального блока:	Действие структурированного текста:
предварительное сканирование	Никакого действия не производится.	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается, инструкция не выполняется, а выходы не обновляются.	не применимо
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.	EnableIn всегда установлен. Инструкция выполняется.
постсканирование	Никакого действия не производится.	Никакого действия не производится.

Пример: **1.** Контроллер копирует Target в Dest.



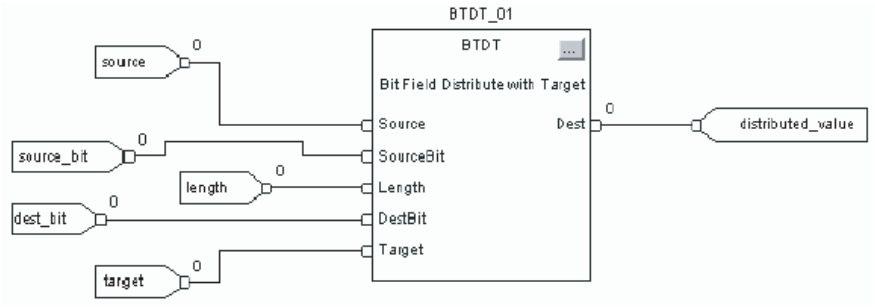
2. SourceBit и Length задают, какие биты в Source необходимо копировать в Dest, начиная с DestBit. Source и Target остаются без изменений.



Структурированный текст

```
BTDT_01.Source := source;
BTDT_01.SourceBit := source_bit;
BTDT_01.Length := length;
BTDT_01.DestBit := dest_bit;
BTDT_01.Target := target;

BTDT(BTDT_01);

distributed_value := BTDT_01.Dest;
```

Функциональный блок

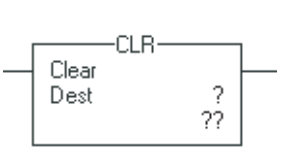


Clear (CLR) (Очистка)

Инструкция CLR сбрасывает все биты Destination (приемника).

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Destination	SINT INT DINT REAL	тег	тег, который будет обнулен

Структурированный текст

```
dest := 0;
```

В структурированном тексте инструкция CLR отсутствует. Вместо этого присвойте 0 тому тегу, который вы хотите обнулить. Этот оператор присвоения обнулит *dest*.

Информацию о синтаксисе операций присваивания и выражений в структурированном тексте можно найти в разделе «Программирование структурированного текста».

Описание:

Инструкция CLR сбрасывает все биты Destination (приемника).

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

Условия ошибки:

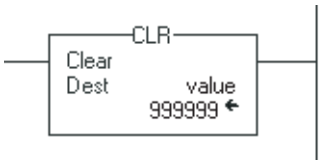
отсутствуют

Выполнение:

Условие:	Действие релейное логики:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Инструкция производит очистку Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример: Обнуление всех битов *value*.

Релейная логика



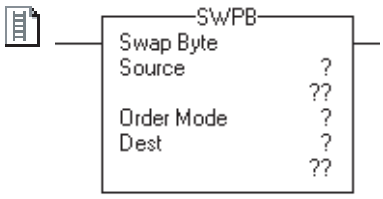
Структурированный текст

```
value := 0;
```

Swap Byte (SWPB) (Переставить байты)

Инструкция SWPB переставляет байты значения.

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Следует ввести:		
Source	INT DINT REAL	тег	тег, содержащий байты, которые вы хотите переставить		
Order Mode			**Если Source:**	**И вы хотите изменить байты следующим образом (каждая буква соответствует отдельному байту):**	**То выберите:**
			INT	не применимо	любую опцию
			DINT REAL	ABCD ⇒ DCBA	REVERSE (или введите 0)
				ABCD ⇒ CDAB	WORD (или введите 1)
				ABCD ⇒ BADC	HIGH/LOW (или введите 2)
Destination	INT DINT REAL	тег	тег для хранения байтов в новом порядке следования		
			Если Source:		**To Destination должен быть:**
			INT		INT
					DINT
			DINT		DINT
			REAL		REAL

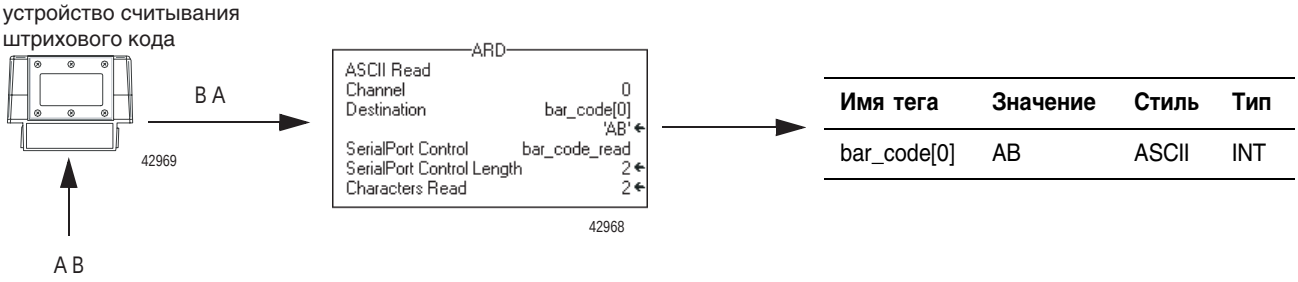
Структурированный текст

```
SWPB(Source,OrderMode,Dest);
```

Операнды такие же, как и операнды для инструкции SWPB в релейной логике. Если вы выбираете режим порядка HIGH/LOW, введите этот режим как HIGHLOW или HIGH_LOW (без косой черты).

Описание: Инструкция SWPB изменяет порядок следования байтов Source (источника) и помещает результат в Destination (приемник).

Если вы считываете или записываете символы ASCII, обычно вам *не* нужно переставлять символы. Инструкции чтения и записи ASCII (ARD, ARL, AWA, AWT) автоматически переставляют символы, как показано ниже.



Имя тега	Значение	Стиль	Тип
bar_code[0]	AB	ASCII	INT

Арифметические флаги состояния: не затрагиваются

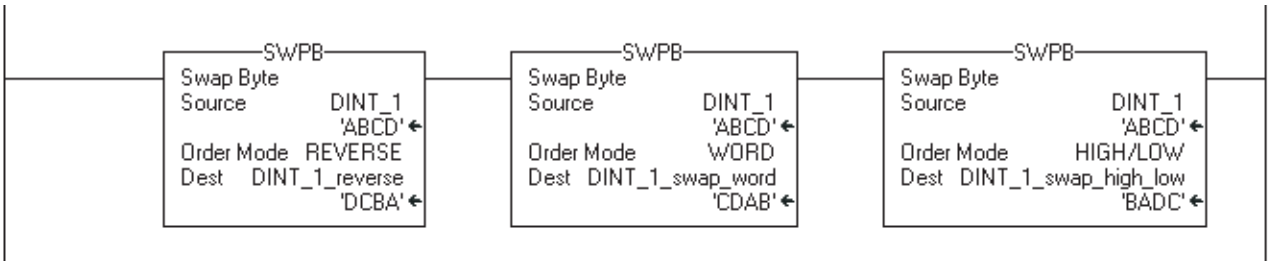
Условия ошибки: отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
инструкция выполняется	Инструкция перегруппировывает заданные байты.	Инструкция переставляет заданные байты.
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится

Пример 1: Каждая из трех инструкций SWPB переставляет байты *DINT_1*, используя различные режимы порядка. Стиль отображения – ASCII, и каждый символ представляет собой один байт. Каждая инструкция помещает байты в новом порядке в разные приемники (Destination).

Релейная логика



Структурированный текст

```
SWPB(DINT_1,REVERSE,DINT_1_reverse);

SWPB(DINT_1,WORD,DINT_1_swap_word);

SWPB(DINT_1,HIGHLOW,DINT_1_swap_high_low);
```

Пример 2: В следующем примере байты в каждом элементе массива меняют свой порядок следования на обратный. Для проекта RSLogix 5000, содержащего этот пример, откройте файл Swap_Bytes_in_Array.ACD в папке RSLogix 5000\Projects\Samples.

1. Инициализируйте теги. Инструкция SIZE находит число элементов в *array* и сохраняет это значение в *array_length*. Последующая инструкция использует это значение, чтобы определить, когда процедура совершила действия над всеми элементами в массиве.

2. Поменяйте порядок следования байтов на обратный в одном элементе *array*.

   - Инструкция SWPB изменяет порядок следования байтов на обратный для элемента с номером, указанным значением *index*. Например, когда *index* равен 0, инструкция SWPB совершает действия над *array[0]*.

   - Инструкция ADD выполняет приращение значания *index*. При следующем выполнении инструкции SWPB она будет совершать действия над следующим элементом в *array*.

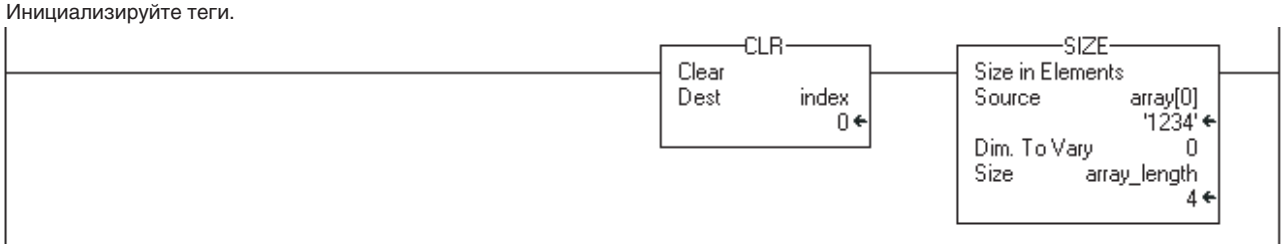
3. Определите, совершила ли инструкция SWPB действия над всеми элементами в массиве.

   - Если *index* меньше, чем число элементов в массиве (*array_length*), продолжите выполнение инструкции для следующего элемента в массиве.

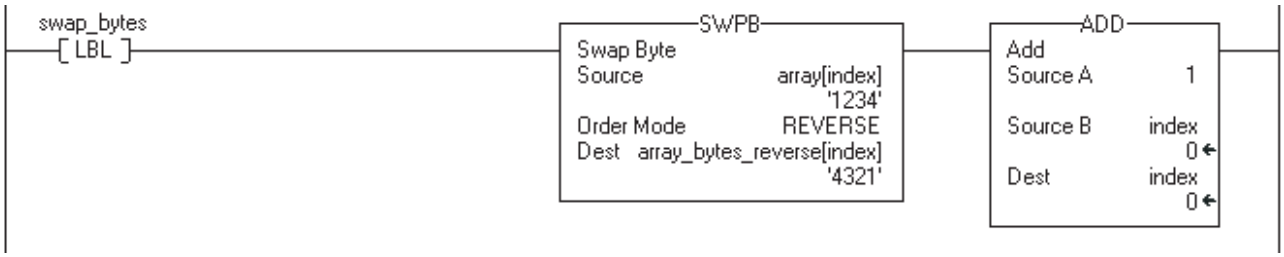
   - Если *index* равен *array_length*, SWPB совершила действия над всеми элементами в массиве.

Релейная логика

Инициализируйте теги.



Поменяйте порядок следования байтов на обратный.



Определите, совершила ли инструкция SWPB действия над всеми элементами в массиве.

Структурированный текст

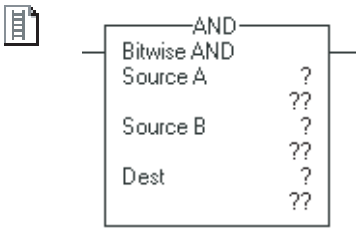
```
index := 0;
SIZE (array[0],0,array_length);
REPEAT
  SWPB(array[index],REVERSE,array_bytes_reverse[index]);
  index := index + 1;
UNTIL(index >= array_length)END_REPEAT;
```

Bitwise AND (AND) (Поразрядное «И»)

Инструкция AND выполняет операцию поразрядное «И», используя биты в Source A (источнике A) и Source B (источнике B), и помещает результат в Destination (приемник).

Описание операции логическое «И» см. на стр. 6-35.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Source A	SINT INT **DINT**	непосредственный тег	значение для выполнения операции побитовое «И» с Source B
	Тег SINT или INT преобразуется в значение DINT посредством заполнения нулями.		
Source B	SINT INT **DINT**	непосредственный тег	значение для выполнения операции побитовое «И» с Source A
	Тег SINT или INT преобразуется в значение DINT посредством заполнения нулями.		
Destination	SINT INT **DINT**	тег	тег для хранения результата

Структурированный текст

```
dest := sourceA AND sourceB
```

Используйте AND или знак «&» в качестве оператора в выражении. Это выражение анализирует операцию *sourceA* AND *sourceB*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в разделе «Программирование структурированного текста».

Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег AND	FBD_LOGICAL	структура	структура AND

Структура FBD_LOGICAL

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
SourceA	DINT	Значение для выполнения операции побитовое «И» с Source B. Допустимое значение = любое целое число
SourceB	DINT	Значение для выполнения операции побитовое «И» с Source A. Допустимое значение = любое целое число
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	DINT	Результат инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание: Когда инструкция разрешена, она анализирует операцию побитовое «И»:

Если бит в Source A:	И бит в Source B:	Бит в Destination:
0	0	0
0	1	0
1	0	0
1	1	1

Если вы смешиваете типы целочисленных данных, инструкция будет заполнять старшие биты меньших значений целочисленных типов данных нулями так, чтобы они имели тот же размер, что и наибольшее значение этого типа данных.

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

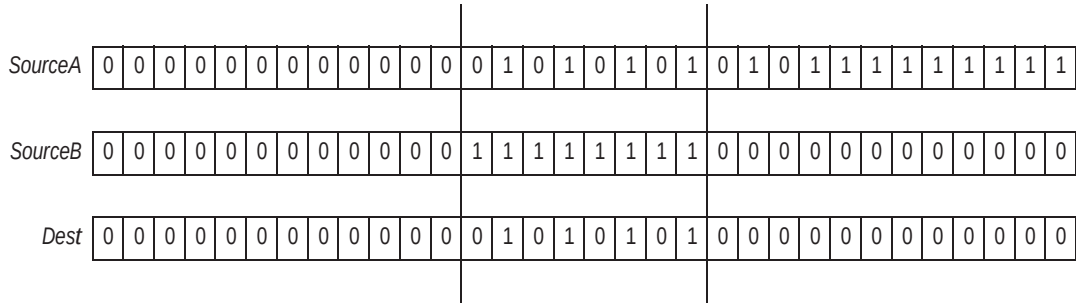
Релейная логика

Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Инструкция выполняет операцию побитовое «И». Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

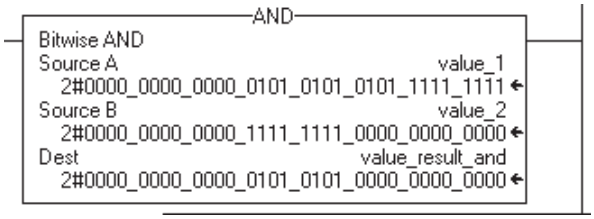
Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Когда инструкция AND разрешена, она выполняет операцию побитовое «И» над SourceA и SourceB и помещает результат в Dest.



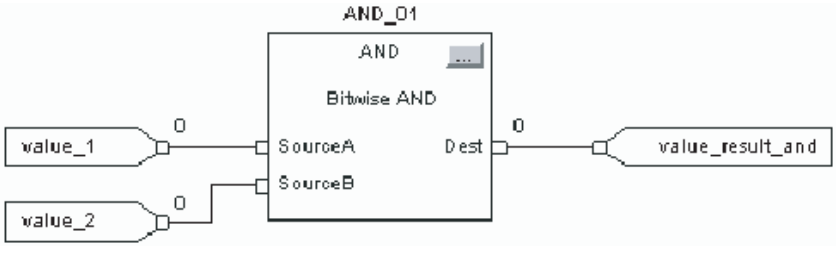
Релейная логика



Структурированный текст

```
value_result_and := value_1 AND value_2;
```

Функциональный блок

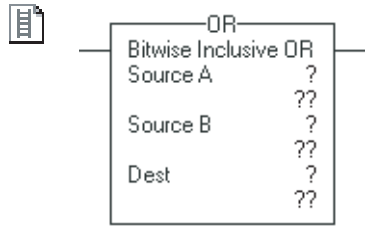


Bitwise OR (OR) (Поразрядное «ИЛИ»)

Инструкция OR выполняет операцию поразрядное «ИЛИ», используя биты в Source A (источнике A) и Source B (источнике B), и помещает результат в Destination (приемник).

Описание операции логическое «ИЛИ» см. на стр. 6-38.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Source A	SINT INT DINT	непосредственный тег	значение для выполнения операции побитовое «ИЛИ» с Source B
	Тег SINT или INT преобразуется в значение DINT посредством заполнения нулями.		
Source B	SINT INT DINT	непосредственный тег	значение для выполнения операции побитовое «ИЛИ» с Source A
	Тег SINT или INT преобразуется в значение DINT посредством заполнения нулями.		
Приемник Destination	SINT INT DINT	тег	тег для хранения результата

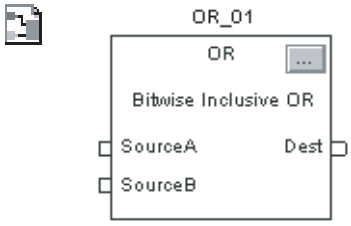
Структурированный текст

```
dest := sourceA OR sourceB
```

Используйте OR в качестве оператора в выражении. Это выражение анализирует операцию *sourceA* OR *sourceB*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в разделе «Программирование структурированного текста».

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег OR	FBD_LOGICAL	структура	структура OR

Структура FBD_LOGICAL

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
SourceA	DINT	Значение для выполнения операции побитовое «ИЛИ» с Source B. Допустимое значение = любое целое число
SourceB	DINT	Значение для выполнения операции побитовое «ИЛИ» с Source A. Допустимое значение = любое целое число
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	DINT	Результат инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание: Когда эта инструкция разрешена, она анализирует операцию побитовое «ИЛИ»:

Если бит в Source A:	И бит в Source B:	Бит в Destination:
0	0	0
0	1	1
1	0	1
1	1	1

Если вы смешиваете типы целочисленных данных, инструкция будет заполнять старшие биты меньших значений целочисленных типов данных нулями так, чтобы они имели тот же размер, что и наибольшее значение этого типа данных.

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

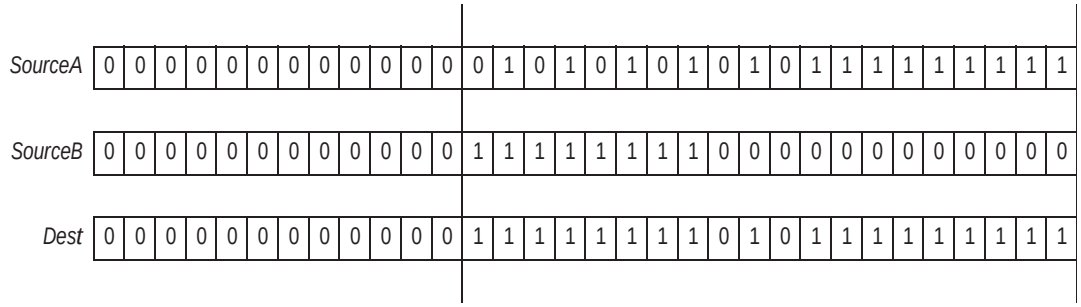
Релейная логика

Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Инструкция выполняет операцию побитовое «ИЛИ». Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

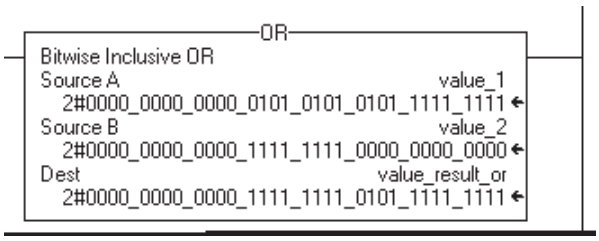
Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Когда инструкция OR разрешена, она выполняет операцию побитовое «ИЛИ» над SourceA и SourceB и помещает результат в Dest.



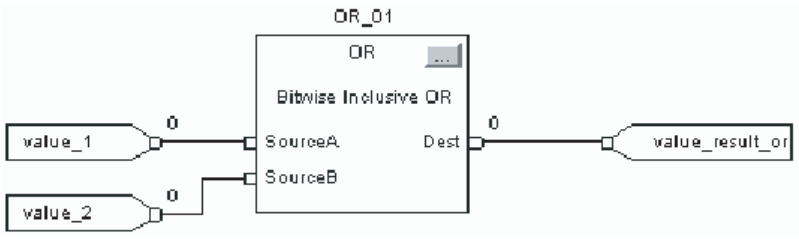
Релейная логика



Структурированный текст

```
value_result_or := value_1 OR value_2;
```

Функциональный блок

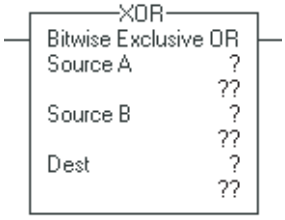


Bitwise Exclusive OR (XOR) (Поразрядное исключающее «ИЛИ»)

Инструкция XOR выполняет операцию поразрядное исключающее «ИЛИ», используя биты в Source A (источнике A) и Source B (источнике B), и помещает результат в Destination (приемник).

Описание операции логическое исключающее «ИЛИ» см. на стр. 6-41.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Source A	SINT INT DINT	непосредственный тег	значение для выполнения операции побитовое исключающее «ИЛИ» с Source B
	Тег SINT или INT преобразуется в значение DINT посредством заполнения нулями.		
Source B	SINT INT DINT	непосредственный тег	значение для выполнения операции побитовое исключающее «ИЛИ» с Source A
	Тег SINT или INT преобразуется в значение DINT посредством заполнения нулями.		
Destination	SINT INT DINT	тег	тег для хранения результата

Структурированный текст

```
dest := sourceA XOR sourceB
```

Используйте XOR в качестве оператора в выражении. Это выражение анализирует операцию *sourceA* XOR *sourceB*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в разделе «Программирование структурированного текста».

Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег XOR	FBD_LOGICAL	структура	структура XOR

Структура FBD_LOGICAL

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
SourceA	DINT	Значение для выполнения операции побитовое исключающее «ИЛИ» с Source B. Допустимое значение = любое целое число
SourceB	DINT	Значение для выполнения операции побитовое исключающее «ИЛИ» с Source A. Допустимое значение = любое целое число
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	DINT	Результат инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание:

Когда инструкция разрешена, она анализирует операцию побитовое исключающее «ИЛИ»:

Если бит в Source A:	И бит в Source B:	Бит в Destination:
0	0	0
0	1	1
1	0	1
1	1	0

Если вы смешиваете типы целочисленных данных, инструкция будет заполнять старшие биты меньших значений целочисленных типов данных нулями так, чтобы они имели тот же размер, что и наибольшее значение этого типа данных.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

отсутствуют

Выполнение:

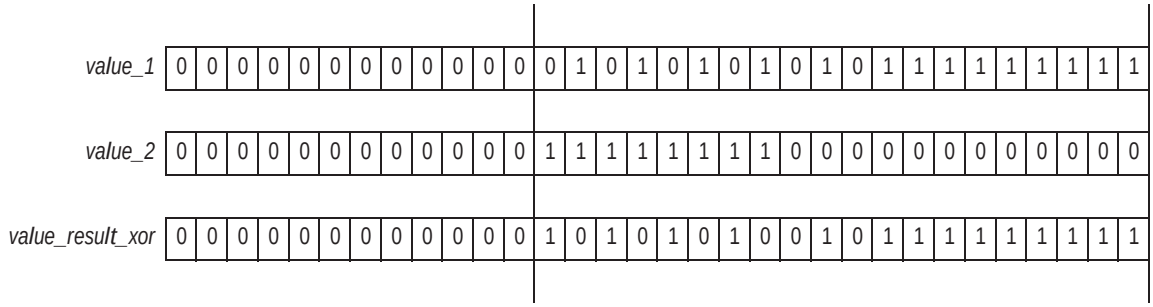
Релейная логика

Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Инструкция выполняет операцию побитовое исключающее «ИЛИ». Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

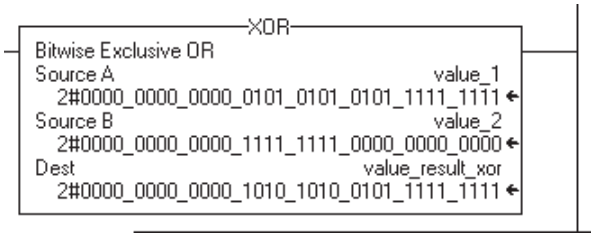
Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Когда инструкция XOR разрешена, она выполняет операцию побитовое исключающее «ИЛИ» над SourceA и SourceB и помещает результат в Dest.



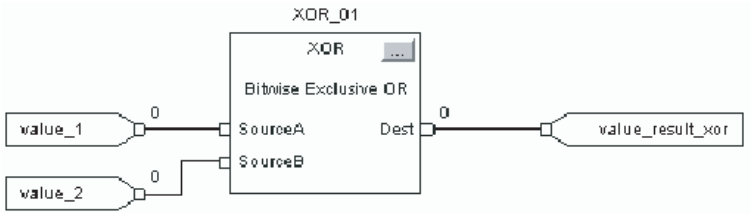
Релейная логика



Структурированный текст

```
value_result_xor := value_1 XOR value_2;
```

Функциональный блок



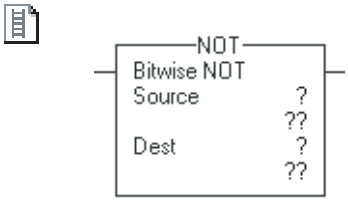
Bitwise NOT (NOT) (Поразрядное «НЕ»)

Инструкция NOT выполняет операцию поразрядное «НЕ», используя биты в Source (источнике), и помещает результат в Destination (приемник).

Описание операции логическое «НЕ» см. на стр. 6-44.

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:
Source	SINT INT DINT	непосредственный тег	значение для выполнения операции побитовое «НЕ»
	Тег SINT или INT преобразуется в значение DINT посредством заполнения нулями.		
Destination	SINT INT DINT	тег	тег для хранения результата

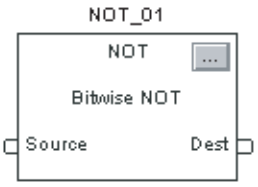
Структурированный текст

```
dest := NOT source
```

Используйте NOT в качестве оператора в выражении. Это выражение анализирует операцию NOT *source*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в разделе «Программирование структурированного текста».

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег NOT	FBD_LOGICAL	структура	структура NOT

Структура FBD_LOGICAL

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	DINT	Значение для выполнения операции побитовое «НЕ». Допустимое значение = любое целое число
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	DINT	Результат инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание: Когда инструкция разрешена, она анализирует операцию побитовое «НЕ»:

Если бит в Source:	Бит в Destination:
0	1
1	0

Если вы смешиваете типы целочисленных данных, инструкция будет заполнять старшие биты меньших значений целочисленных типов данных нулями так, чтобы они имели тот же размер, что и наибольшее значение этого типа данных.

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Релейная логика

Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Инструкция выполняет операцию побитовое «НЕ». Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

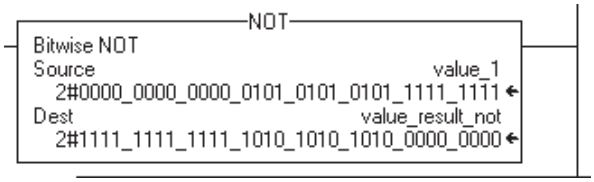
Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Когда инструкция NOT разрешена, она выполняет операцию побитовое «НЕ» над Source (источником) и помещает результат в Dest (приемник).

value_1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1
value_result_not	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0

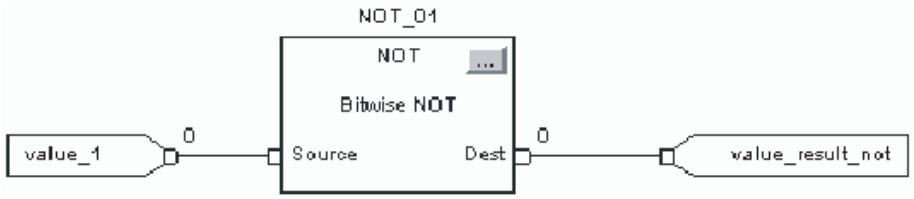
Релейная логика



Структурированный текст

```
value_result_not := NOT value_1;
```

Функциональный блок



Boolean AND (BAND) (Булево «И»)

Инструкция BAND выполняет операцию логическое «И» для макисмум восьми булевых входов.

Описание операции побитовое «И» см. на стр. 6-23.

Операнды:

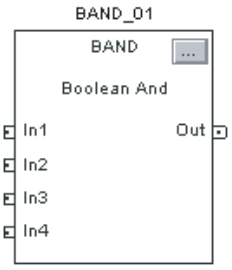
Структурированный текст

```
IF operandA AND operandB THEN
    <statement>;
END_IF;
```

Используйте AND или знак «&» в качестве оператора в выражении. Операнды должны быть значениями BOOL или выражениями, дающими значения BOOL. Это выражение анализирует, установлены ли оба операнда: *operandA* и *operandB* («истина»).

Информацию о синтаксисе выражений в структурированном тексте можно найти в разделе «Программирование структурированного текста».

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег BAND	FBD_BOOLEAN_AND	структура	структура BAND

Структура FBD_BOOLEAN_AND

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
In1	BOOL	Первый булев вход. По умолчанию параметр установлен.
In2	BOOL	Второй булев вход. По умолчанию параметр установлен.
In3	BOOL	Третий булев вход. По умолчанию параметр установлен.
In4	BOOL	Четвертый булев вход. По умолчанию параметр установлен.
In5	BOOL	Пятый булев вход. По умолчанию параметр установлен.
In6	BOOL	Шестой булев вход. По умолчанию параметр установлен.
In7	BOOL	Седьмой булев вход. По умолчанию параметр установлен.
In8	BOOL	Восьмой булев вход. По умолчанию параметр установлен.
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Разрешение выхода.
Out	BOOL	Выход инструкции.

Описание: Инструкция BAND выполняет операцию логическое «И» для максимум восьми булевых входов. Если какой-либо вход не используется, он по умолчанию установлен (1).

Out = In1 AND In2 AND In3 AND In4 AND In5 AND In6 AND In7 AND In8

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Условие:	Действие функционального блока:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

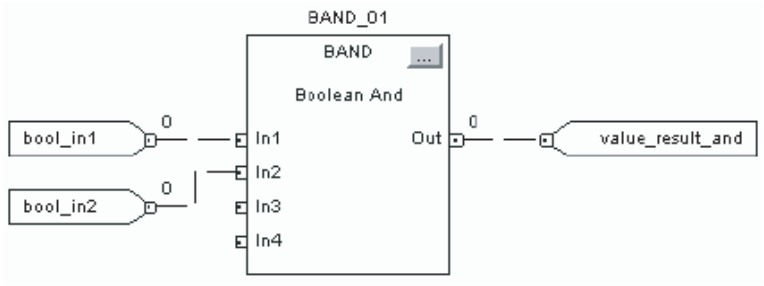
Пример 1: В этом примере операция булево «И» производится с *bool_in1* и *bool_in2*, результат помещается в *value_result_and*.

Если bool_in1:	Если bool_in2:	To value_result_and:
0	0	0
0	1	0
1	0	0
1	1	1

Структурированный текст

```
value_result_and := bool_in1 AND bool_in2;
```

Функциональный блок



Пример 2: Если и *bool_in1*, и *bool_in2* установлены («истина»), то *light1* устанавливается (включается). В противном случае *light1* сбрасывается (выключается).

Структурированный текст

```
IF bool_in1 AND bool_in2 THEN
     light1 := 1;
ELSE
     light1 := 0;
END_IF;
```

Boolean OR (BOR) (Булево «ИЛИ»)

Инструкция BOR выполняет операцию логическое «ИЛИ» для макисмум восьми булевых входов.

Описание операции побитовое «ИЛИ» см. на стр. 6-26.

Операнды:

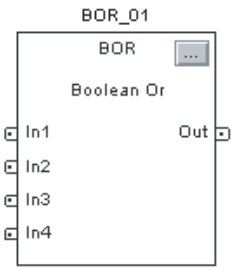
Структурированный текст

```
IF operandA OR operandB THEN
    <statement>;
END_IF;
```

Используйте OR в качестве оператора в выражении. Операнды должны быть значениями BOOL или выражениями, дающими значения BOOL. Это выражение анализирует, установлен ли операнд *operandA*, или операнд *operandB*, или оба операнда («истина»).

Информацию о синтаксисе выражений в структурированном тексте можно найти в разделе «Программирование структурированного текста».

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег BOR	FBD_BOOLEAN_OR	структура	структура BOR

Структура FBD_BOOLEAN_OR

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
In1	BOOL	Первый булев вход. По умолчанию параметр сброшен.
In2	BOOL	Второй булев вход. По умолчанию параметр сброшен.
In3	BOOL	Третий булев вход. По умолчанию параметр сброшен.
In4	BOOL	Четвертый булев вход. По умолчанию параметр сброшен.
In5	BOOL	Пятый булев вход. По умолчанию параметр сброшен.
In6	BOOL	Шестой булев вход. По умолчанию параметр сброшен.
In7	BOOL	Седьмой булев вход. По умолчанию параметр сброшен.
In8	BOOL	Восьмой булев вход. По умолчанию параметр сброшен.
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Разрешение выхода.
Out	BOOL	Выход инструкции.

Описание: Инструкция BOR выполняет операцию логическое «ИЛИ» для макисмум восьми булевых входов. Если какой-либо вход не используется, он по умолчанию сброшен (0).

Out = In1 OR In2 OR In3 OR In4 OR In5 OR In6 OR In7 OR In8

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Условие:	Действие функционального блока:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

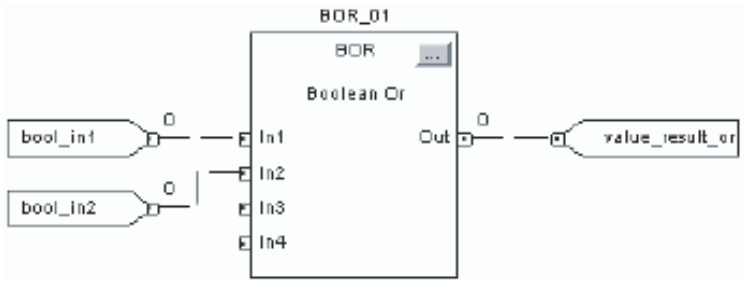
Пример 1: В этом примере операция булево «ИЛИ» производится с *bool_in1* и *bool_in2*, результат помещается в *value_result_or*.

Если bool_in1:	Если bool_in2:	To value_result_or:
0	0	0
0	1	1
1	0	1
1	1	1

Структурированный текст

```
value_result_or := bool_in1 OR bool_in2;
```

Функциональный блок



Пример 2: В этом примере *light_1* устанавливается (включается), если

- только *bool_in1* установлен («истина»);
- только *bool_in2* установлен («истина»);
- и *bool_in1*, и *bool_in2* установлены («истина»).

В противном случае *light1* сбрасывается (выключается).

Структурированный текст

```
IF bool_in1 OR bool_in2 THEN
     light1 := 1;
ELSE
     light1 := 0;
END_IF;
```

Boolean Exclusive Or (BXOR) (Булево исключающее «ИЛИ»)

Инструкция BXOR выполняет операцию исключающее «ИЛИ» на двух булевых входах.

Описание операции побитовое исключающее «ИЛИ» см. на стр. 6-29.

Операнды:

Структурированный текст

```
IF operandA XOR operandB THEN
    <statement>;
END_IF;
```

Используйте XOR в качестве оператора в выражении. Операнды должны быть значениями BOOL или выражениями, дающими значения BOOL. Это выражение анализирует, установлен ли только операнд *operandA*, или только операнд *operandB* («истина»).

Информацию о синтаксисе выражений в структурированном тексте можно найти в разделе «Программирование структурированного текста».

Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег BXOR	FBD_BOOLEAN_XOR	структура	структура BXOR

Структура FBD_BOOLEAN_XOR

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
In1	BOOL	Первый булев вход. По умолчанию параметр сброшен.
In2	BOOL	Второй булев вход. По умолчанию параметр сброшен.
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Разрешение выхода.
Out	BOOL	Выход инструкции.

Описание:

Инструкция BXOR выполняет операцию исключающее «ИЛИ» на двух булевых входах.

Out = In1 XOR In2

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие функционального блока:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

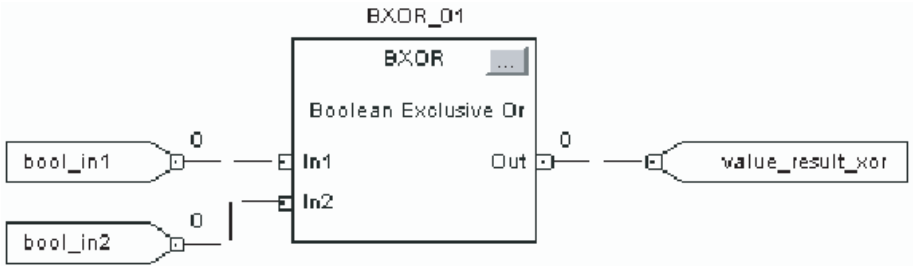
Пример 1: В этом примере операция исключающее «ИЛИ» производится с *bool_in1* и *bool_in2*, результат помещается в *value_result_xor*.

Если bool_in1:	Если bool_in2:	То value_result_xor:
0	0	0
0	1	1
1	0	1
1	1	0

Структурированный текст

```
value_result_xor := bool_in1 XOR bool_in2;
```

Функциональный блок



Пример 2: В этом примере *light_1* устанавливается (включается), если

- только *bool_in1* установлен («истина»);
- только *bool_in2* установлен («истина»).

В противном случае *light1* сбрасывается (выключается).

Структурированный текст

```
IF  bool_in1  XOR  bool_in2  THEN
     light1  :=  1;
ELSE
     light1  :=  0;
END_IF;
```

Boolean NOT (BNOT) (Булево «НЕ»)

Инструкция BNOT дополняет булев вход.

Описание операции побитовое «НЕ» см. на стр. 6-32.

Операнды:

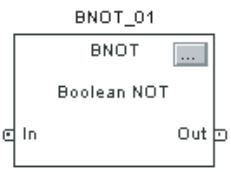
Структурированный текст

```
IF NOT operand THEN
     <statement>;
END_IF;
```

Используйте NOT в качестве оператора в выражении. Операнды должны быть значениями BOOL или выражениями, дающими значения BOOL. Это выражение анализирует, сброшен ли *operand* («ложь»).

Информацию о синтаксисе выражений в структурированном тексте можно найти в разделе «Программирование структурированного текста».

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег BNOT	FBD_BOOLEAN_NOT	структура	структура BNOT

Структура FBD_BOOLEAN_NOT

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
In	BOOL	Вход в инструкцию. По умолчанию параметр установлен.
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Разрешение выхода.
Out	BOOL	Выход инструкции.

Описание:

Инструкция BNOT дополняет булев вход.

Out = NOT In

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие функционального блока:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

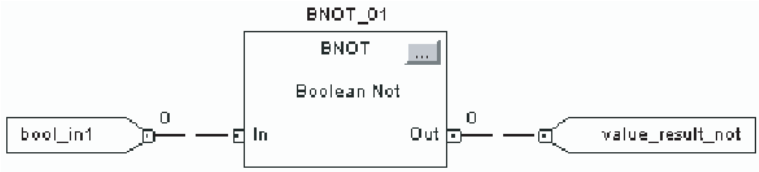
Пример 1: В этом примере операция булево «НЕ» дополняет *bool_in1*, результат помещается в *value_result_not*.

Если bool_in1:	To value_result_not:
0	1
1	0

Структурированный текст

```
value_result_not  :=  NOT  bool_in1;
```

Функциональный блок



Пример 2: Если *bool_in1* сброшен, то *light1* сбрасывается (выключается). В противном случае *light1* устанавливается (включается).

Структурированный текст

```
IF  NOT  bool_in1  THEN
     light1  :=  0;
ELSE
     light1  :=  1;
END_IF;
```

Примечания:

Инструкции Массива (Файла)/Прочие (FAL, FSC, COP, CPS, FLL, AVE, SRT, STD, SIZE)

Введение

Файловые/прочие инструкции оперируют с массивами данных.

Если вы хотите	Используйте эту инструкцию	Доступна в этих языках	См. стр.
производить арифметические, логические, функциональные операции и операции сдвига со значениями в массивах	FAL	релейная логика структурированный текст(1)	7-7
осуществлять поиск и сравнение значений в массивах	FSC	релейная логика	7-19
копировать содержимое одного массива в другой	COP	релейная логика структурированный текст	7-28
копировать содержимое одного массива в другой без прерывания	CPS	релейная логика структурированный текст	7-28
заполнять массив конкретными данными	FLL	релейная логика структурированный текст(1)	7-34
вычислять среднее значение массива	AVE	релейная логика структурированный текст(1)	7-38
сортировать данные массива одинаковой размерности по возрастанию	SRT	релейная логика структурированный текст	7-43
вычислять стандартное отклонение значений массива	STD	релейная логика структурированный текст(1)	7-48
находить размер размерности массива	SIZE	релейная логика структурированный текст	7-53

(1) Не существует эквивалентной инструкции для структурированного текста. Используйте другие процедуры структурированного текста для достижения таких же результатов. Смотрите описание инструкции.

Вы можете смешивать типы данных, но при этом могут появиться отсутствие соответствия и ошибки округления и для выполнения инструкции потребуется больше времени. Проверьте бит S:V, чтобы убедиться, отброшена ли у полученного результата дробная часть.

Для инструкций релейной логики, **жирный** шрифт типов данных означает, что это оптимальные типы данных. Инструкция выполняется быстрее и требует меньший объем памяти, если все операнды инструкции используют один и тот же оптимальный тип данных, обычно DINT или REAL.

Выбор режима работы

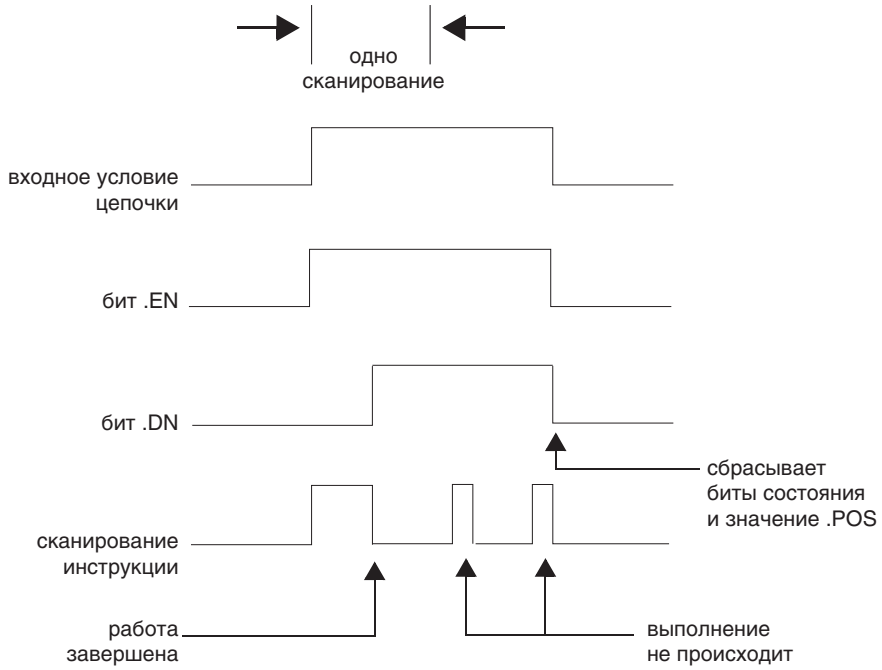
Для инструкций FAL и FSC режим сообщает контроллеру, как распределять операции над массивами.

Если вы хотите:	Выберите этот режим:
оперировать всеми заданными элементами в массиве, прежде чем перейти к следующей инструкции	All (все)
распределять операции над массивами между несколькими сканированиямивводить определенное число элементов для обработки при каждом сканировании(1-2147483647)	Numerical (числовой)
манипулировать одним элементом массива каждый раз, когда входное условие цепочки переходит от значения "ложь" к значению "истина	"Incremental (инкрементный)

Режим All (все)

В режиме All операции со всеми элементами массива совершаются до того, как происходит переход к следующей инструкции. Работа начинается, когда входное условие цепочки инструкции переходит от значения "ложь" к значению "истина". Значение позиции (.POS) в управляющей структуре указывает на элемент в массиве, который в настоящий момент использует инструкция. Работа заканчивается, когда значение .POS уравнивается со значением .LEN.

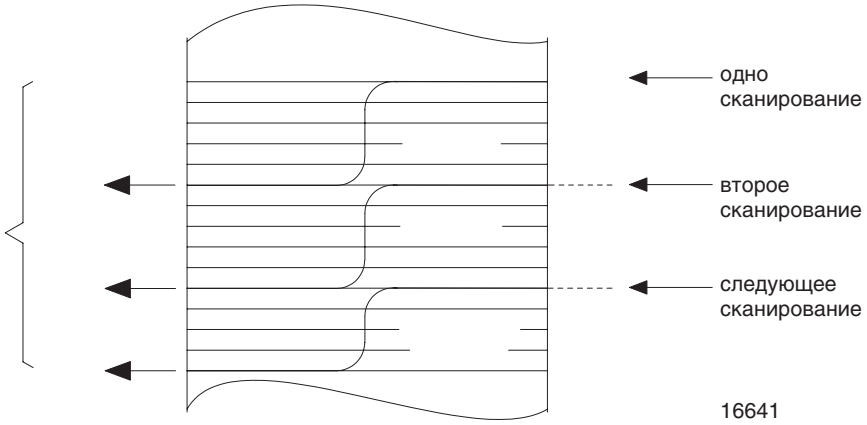
На следующей временной диаграмме показано соотношение между битами состояния и работой инструкции. Когда выполнение инструкции завершено, устанавливается бит .DN. Бит .DN, бит .EN и значение .POS сбрасываются, когда входное условие цепочки - "ложь". Только в этом случае путем изменения входного условия цепочки с "ложь" на "истина" может быть запущено выполнение инструкции..



Режим Numerical (числовой)

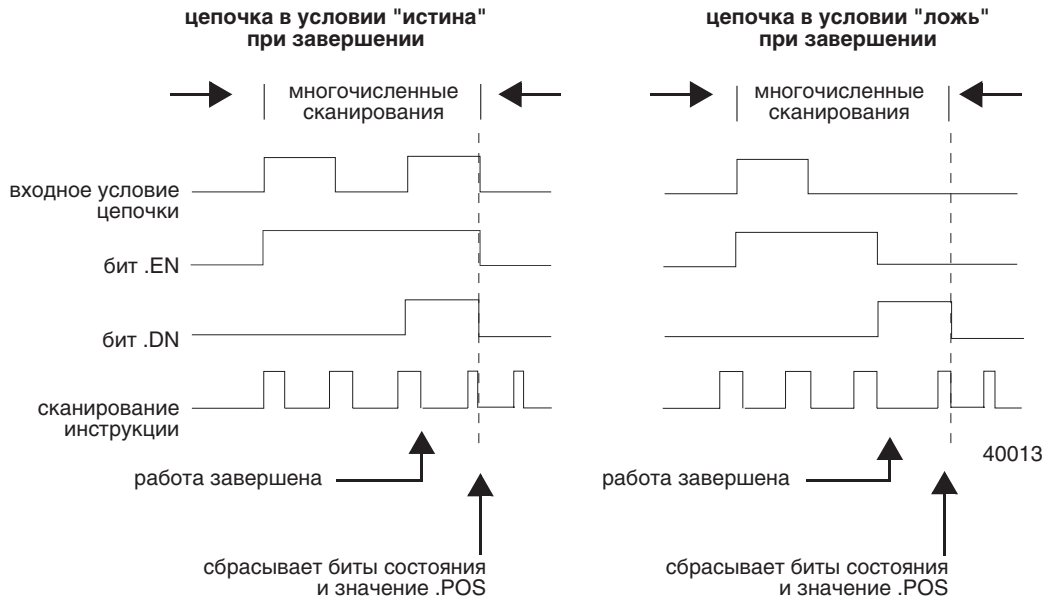
Режим Numerical распределяет операции над массивами между несколькими сканированиями. Этот режим используется при работе с данными, не являющимися критическими во времени, или с большим количеством данных. Вы вводите определенное число элементов для обработки при каждом сканировании, что позволяет уменьшить время сканирования

Выполнение запускается, когда входное условие цепочки переходит от значения "ложь" к значению "истина". После запуска инструкция выполняется при каждом сканировании в течение определенного числа сканирований, необходимых для завершения работы во всем массиве. После запуска входное условие цепочки может неоднократно изменяться, не прерывая выполнение инструкции.



ВАЖНО! Избегайте использовать результаты файловой инструкции, работающей в числовом режиме, пока не будет установлен бит .DN.

На следующей временной диаграмме показано соотношение между битами состояния и работой инструкции. Когда выполнение инструкции завершено, устанавливается бит .DN.

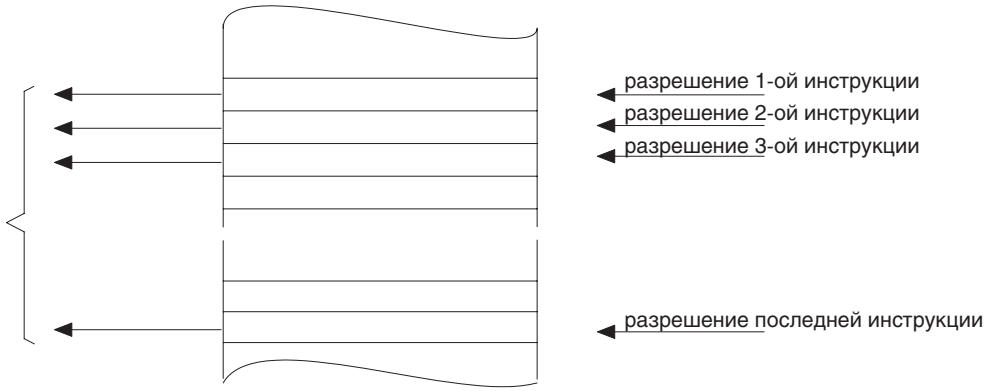


Если при завершении входное условие цепочки - "истина", биты .EN и .DN устанавливаются до тех пор, пока входное условие цепочки не примет значение "ложь". Когда входное условие цепочки примет значение "ложь", эти биты, а также значение .DN сбрасываются.

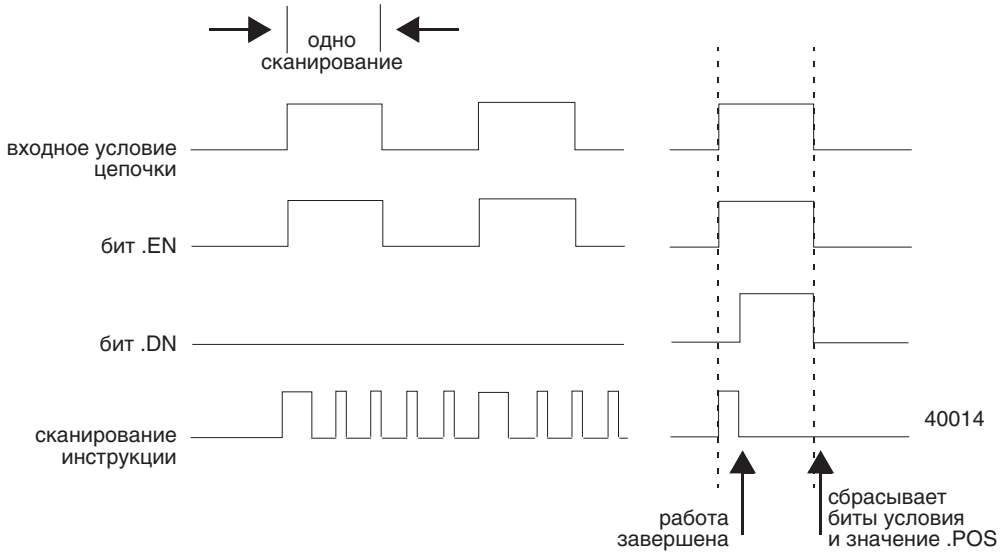
Если при завершении входное условие цепочки - "ложь", бит .EN мгновенно сбрасывается. Через одно сканирование после сброса бита .EN сбрасываются бит .DN и значение .POS.

Режим Incremental (инкрементный)

Режим Incremental манипулирует одним элементом массива каждый раз, когда входное условие цепочки инструкции переходит от значения "ложь" к значению "истина".



На следующей временной диаграмме показано соотношение между битами состояния и работой инструкции. Выполнение происходит только при таком сканировании, когда входное условие цепочки переходит от значения "ложь" к значению "истина". Каждый раз когда это происходит, манипуляции осуществляются только с одним элементом массива. Если входное условие цепочки остается в значении "истина" на протяжении более одного сканирования, инструкция выполняется только при первом сканировании.



Бит .EN устанавливается, только когда входное условие цепочки - "истина". Бит .DN устанавливается, когда проведены манипуляции с последним элементом в массиве. Когда проведены манипуляции с последним элементом и входное условие цепочки - "ложь", бит .EN, бит .DN и значение .POS сбрасываются.

Разница между инкрементным режимом и числовым режимом с частотой один элемент за сканирование:

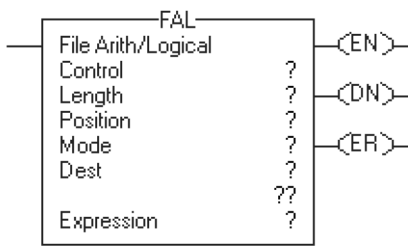
- Числовой режим с любым числом элементов, сканирующихся за раз, требует только одного перехода входного условия цепочки от значения "ложь" к значению "истина", чтобы начать выполнение. Инструкция продолжает выполнять заданное число элементов при каждом сканировании до своего завершения, независимо от значения входного условия цепочки.

- Инкрементный режим требует, чтобы входное условие цепочки изменялось от значения "ложь" к значению "истина", для проведения манипуляций с одним элементом в массиве.

File Arithmetic and Logic (FAL) (Файловая арифметика и логика)

Инструкция FAL выполняет операцию копирования, арифметические, логические и функциональные операции с данными, хранящимися в массиве.

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Описание:
Контроль Control	CONTROL	тег	управляющая структура для операции
Длина Length	DINT	непосредственный	число элементов в массиве, над которыми производятся манипуляции
Позиция Position	DINT	непосредственный	текущий элемент в массивеисходное значение обычно 0
Режим Mode	DINT	непосредственный	как распределить работувыберите INC, ALL или введите число
Приемник Destination	SINT INT DINT REAL	тег	тег для хранения результата
Выражение Expression	SINT INTD INT REAL	непосредственный тег	выражение, состоящее из тегов, и/или прямые значения, разделенные операторами
	Тег SINT или INT конвертируется в значение DINT посредством дополнительного знакового разряда.		

Структурированный текст

В структурированном тексте инструкция FAL отсутствует, но можно получить тот же результат, используя инструкцию SIZE и конструкцию FOR...DO или другую циклическую конструкцию.

```
SIZE(destination,0,length-1);

FOR position = 0 TO length DO

     destination[position] := numeric_expression;

END_FOR;
```

Информацию о синтаксисе конструкций структурированного текста можно найти в Приложении C.

Структура CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения показывает, что инструкция FAL разрешена.
.DN	BOOL	Бит выполнения устанавливается, когда инструкция выполнила операцию с последним элементом (.POS = .LEN).
.ER	BOOL	Бит ошибки устанавливается, если выражение генерирует переполнение (устанавливается S:V). Выполнение инструкции останавливается до тех пор, пока процедура не сбросит бит .ER. Значение .POS содержит позицию элемента, ставшего причиной переполнения.
.LEN	DINT	Длина задает число элементов в массиве, где оперирует инструкция FAL.
.POS	DINT	Позиция содержит позицию текущего элемента, к которому в данный момент обращается инструкция.

Описание: Инструкция FAL производит такие же операции с массивами, какие инструкция CPT производит с элементами.

Примеры, описание которых начинается на стр. 7-14, показывают, как использовать значение .POS для работы с массивом. Если нижний индекс в выражении Destination (приемника) находится за пределами допустимого значения, инструкция FAL генерирует основную ошибку (тип 4, код 20).

Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

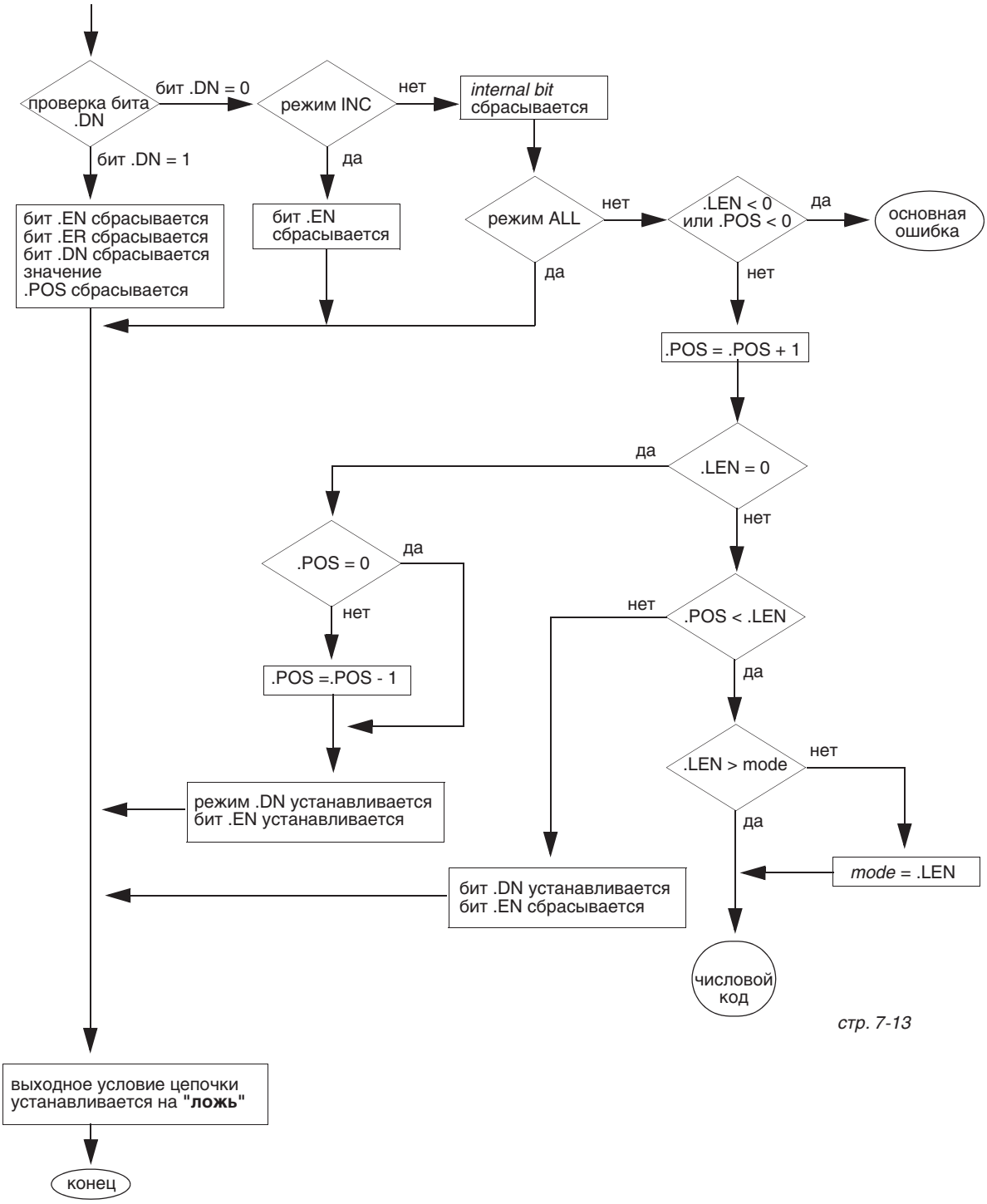
Условия ошибки:

Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
нижний индекс выходит за допустимые пределы	4	20
.POS < 0 или .LEN < 0	4	21

Выполнение:

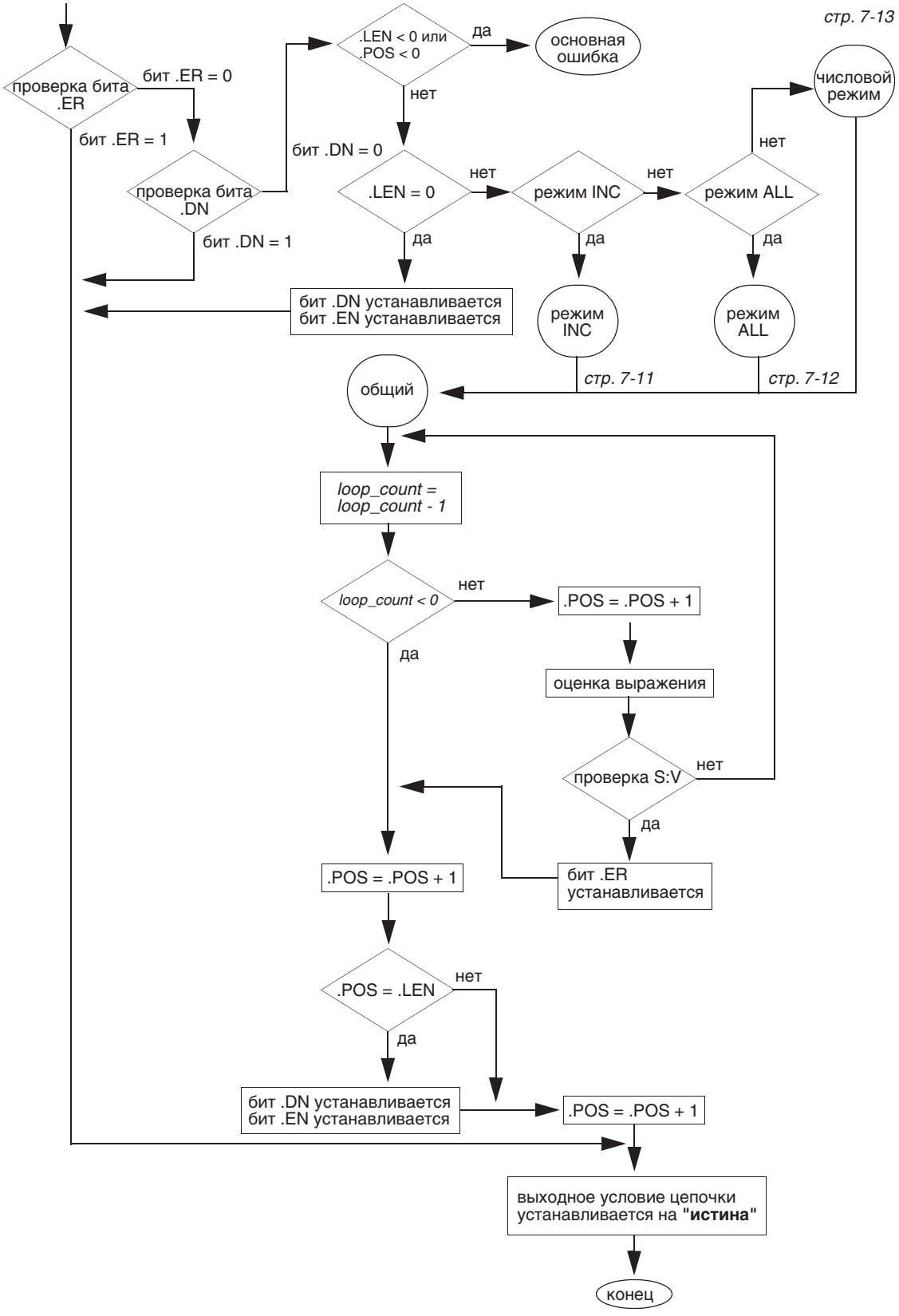
Условие:	**Действие релейной логики:**
предварительное сканирование	Выходное условие цепочки устанавливается на "ложь".

входное условие цепочки - "ложь"

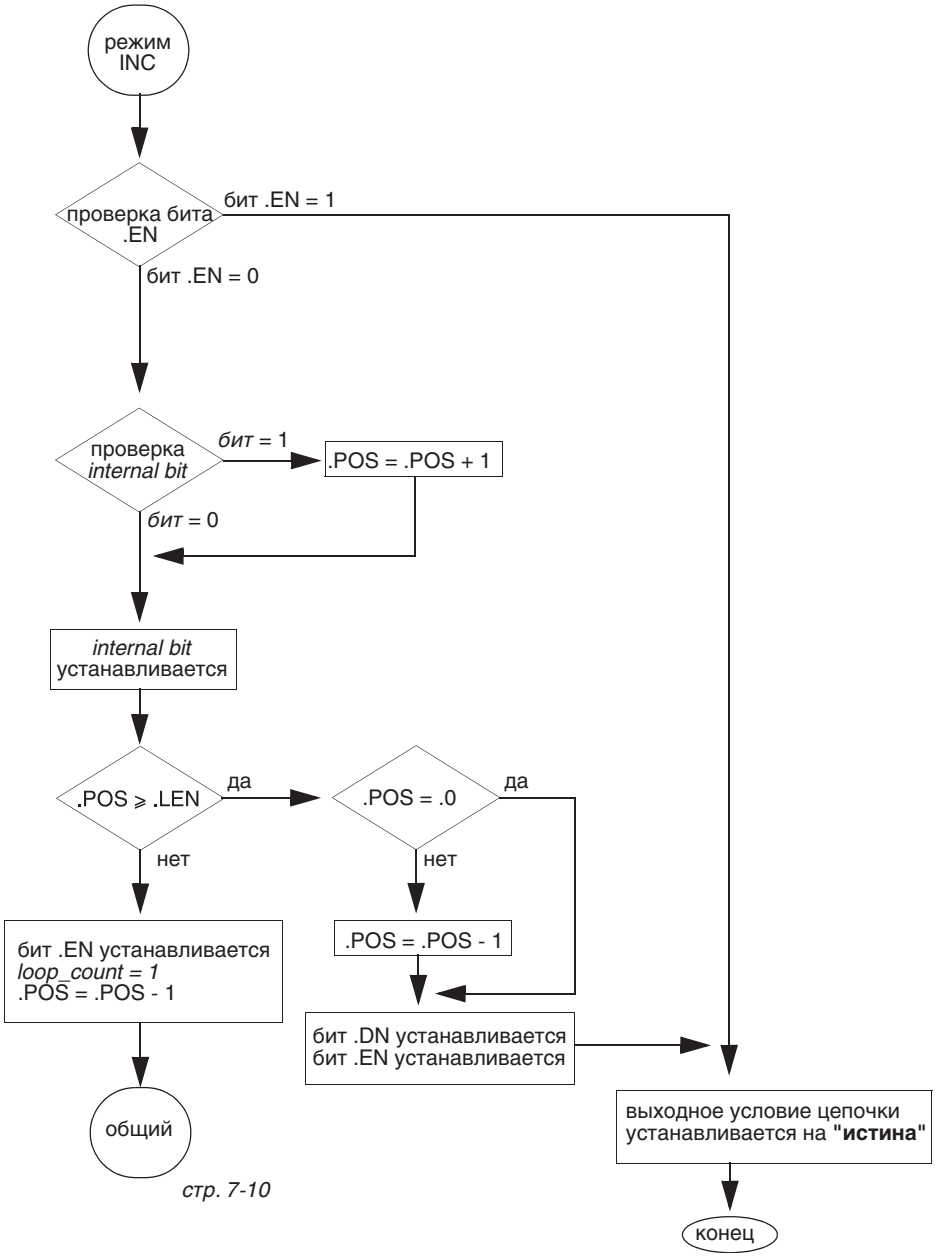


Условие:	**Действие релейной логики:**

входное условие цепочки - "истина"

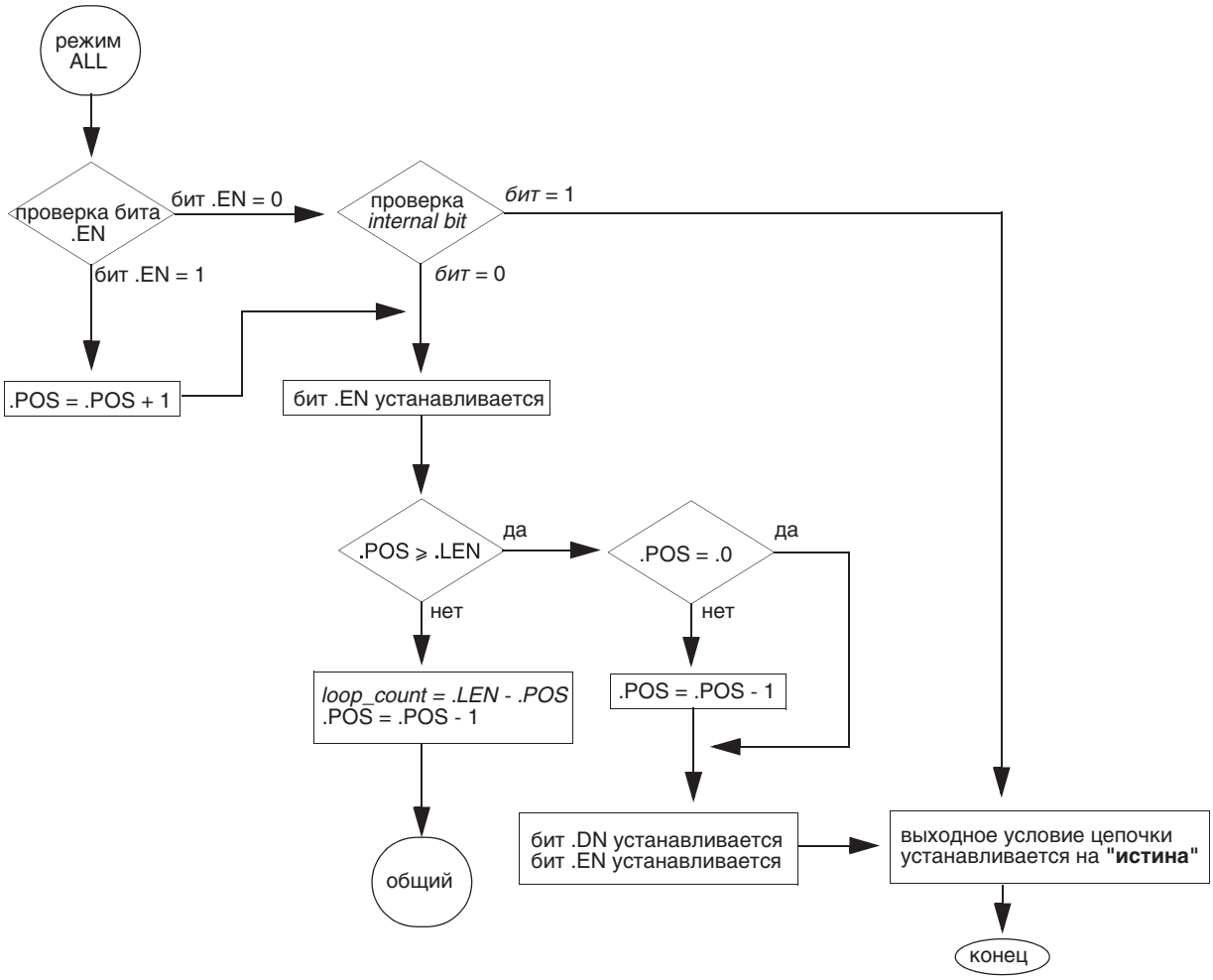


Условие:	**Действие релейной логики:**

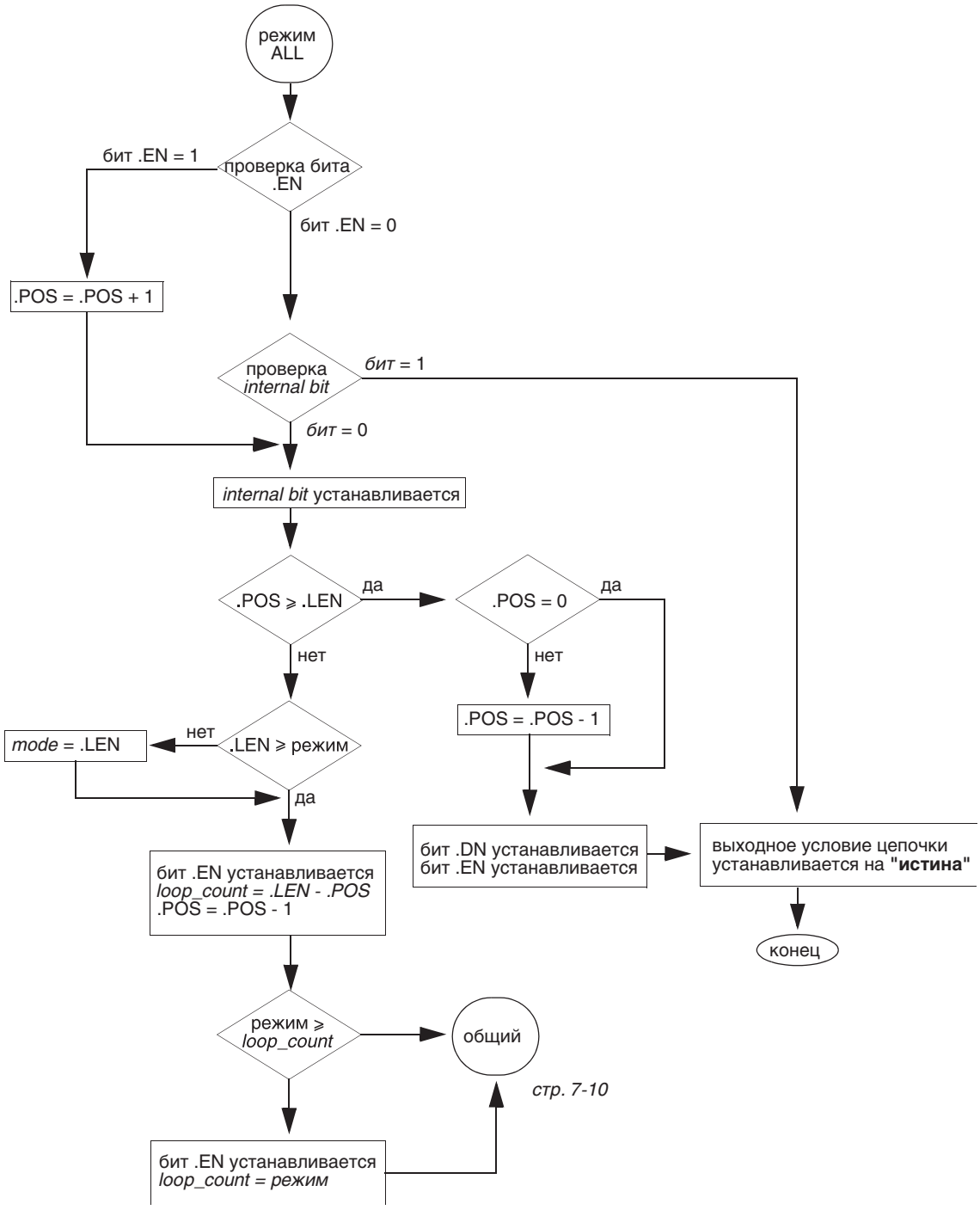


стр. 7-10

Условие:	**Действие релейной логики:**

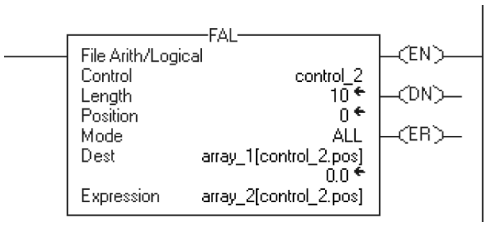


Условие:	**Действие релейной логики:**

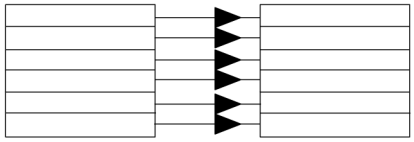


постсканирование	Выходное условие цепочки устанавливается на "ложь"

Пример 1: Когда инструкция FAL разрешена, она копирует каждый элемент *array_2* на такую же позицию в *array_1*



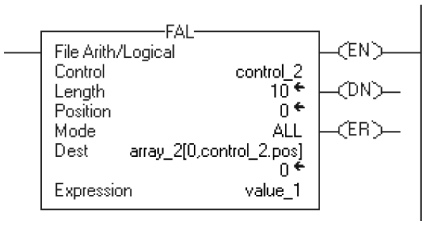
копирование массива в массив



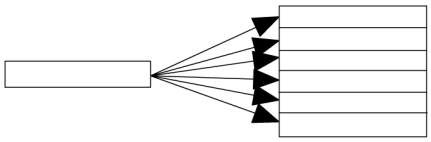
Выражение: *array_2[control_2.pos]*

Приемник: *array_1[control_2.pos]*

Пример 2: Когда инструкция FAL разрешена, она копирует *value_1* в первые 10 позиций второй размерности *array_2*.



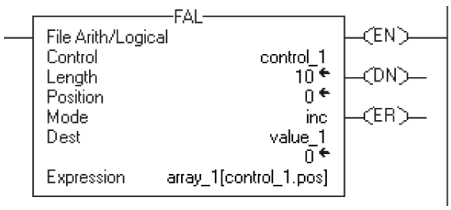
копирование элемента в массив



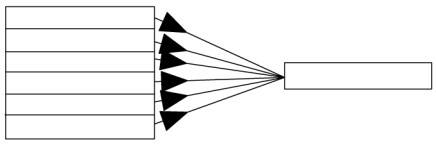
Выражение: *value_1*

Приемник: *array_2[0,control_2.pos]*

Пример: 3 Каждый раз, когда разрешается инструкция FAL, она копирует текущее значение *array_1* в *value_1*. Инструкция FAL использует инкрементный режим, таким образом, каждый раз, когда разрешается эта инструкция, копируется только одно значение массива. В следующий раз при разрешении инструкции, она будет записывать поверх *value_1* следующее значение в *array_1*.



копирование массива в элемент

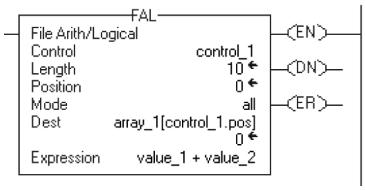


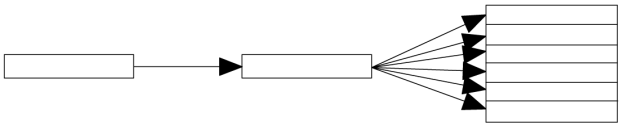
Выражение: *array_1[control_1.pos]*

Приемник: *value_1*

Пример 4: Когда инструкция FAL разрешена, она складывает *value_1* и *value_2* и хранит результат в текущей позиции *array_1*.

арифметическая операция: (элемент + элемент) в массив



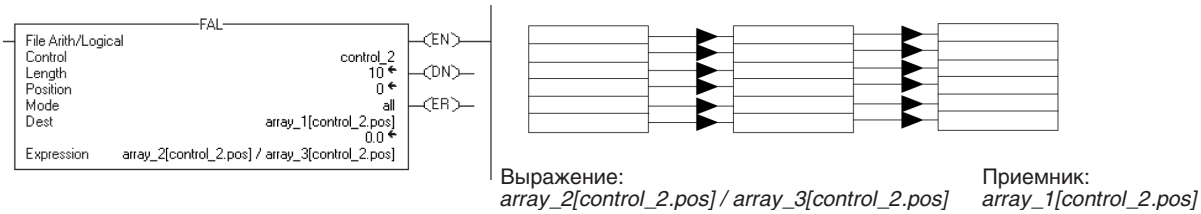


Выражение: *value_1 + value_2*

Приемник: *array_1[control_1.pos]*

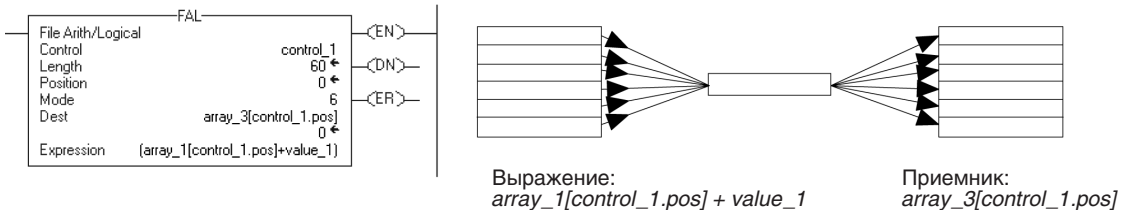
Пример 5: Когда инструкция FAL разрешена, она делит значение в текущей позиции *array_2* на значение в текущей позиции *array_3* и хранит результат в текущей позиции *array_1*.

арифметическая операция: (массив/массив) в массив



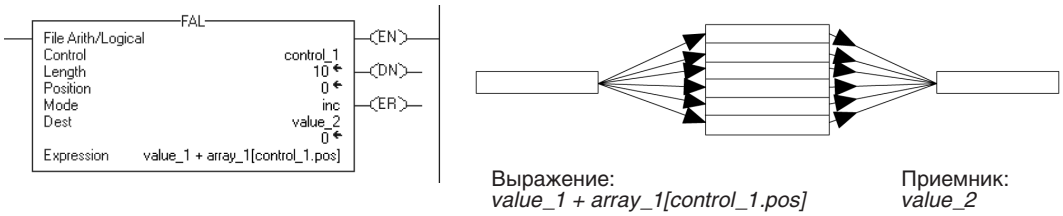
Пример 6: Когда инструкция FAL разрешена, она прибавляет значение в текущей позиции в *array_1* к *value_1* и хранит результат в текущей позиции в *array_3*. Для проведения манипуляции инструкция должна выполнить эту операцию 10 раз для всего *array_1* и *array_3*.

арифметическая операция: (массив + элемент) в массив



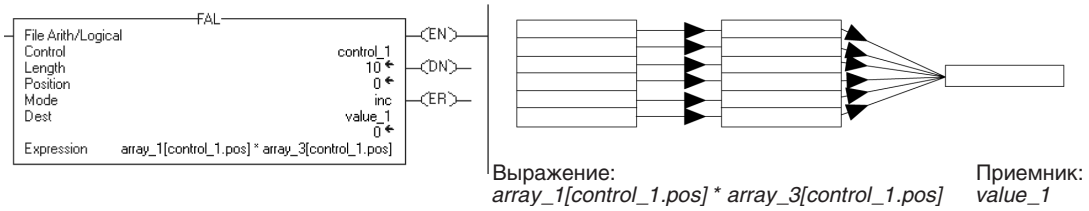
Пример 7: Каждый раз, когда разрешается инструкция FAL, она прибавляет *value_1* к текущему значению *array_1* и хранит результат в *value_2*. Инструкция FAL использует инкрементный режим, таким образом, каждый раз, когда разрешается эта инструкция, только одно значение массива прибавляется к *value_1*. В следующий раз при разрешении инструкции, она будет записывать поверх *value_2*.

арифметическая операция: (элемент + массив) в элемент



Пример 8: Когда инструкция FAL разрешена, она умножает текущее значение *array_1* на текущее значение *array_3* и хранит результат в *value_1*. Инструкция FAL использует инкрементный режим, таким образом, каждый раз, когда разрешается эта инструкция, перемножается только одна пара значений массива. В следующий раз при разрешении инструкции, она будет записывать поверх *value_1*.

арифметическая операция: (массив * массив) в элемент



Выражения FAL

Вы программируете выражения в инструкциях FAL таким же образом, как и выражения в инструкциях CPT. Обратитесь к следующим разделам для получения информации о допустимых операторах, формате и порядке операций, которые являются общими для обеих инструкций.

Допустимые операторы

Оператор:	Описание:	Оптимальный:
+	сложение	DINT, REAL
-	вычитание/отрицание	DINT, REAL
*	умножение	DINT, REAL
/	деление	DINT, REAL
**	степень (x в степени y)	DINT, REAL
ABS	абсолютная величина	DINT, REAL
ACS	арккосинус	REAL
AND	побитовое "И"	DINT
ASN	арксинус	REAL
ATN	арктангенс	REAL
COS	косинус	REAL
DEG	преобразование радиан в градусы	DINT, REAL
FRD	преобразование BCD в целые числа	DINT
LN	натуральный логарифм	REAL
LOG	логарифм с основанием 10	REAL
MOD	деление по модулю	DINT, REAL
NOT	побитовое дополнение	DINT
OR	побитовое "ИЛИ"	DINT
RAD	преобразование градусов в радианы	DINT, REAL
SIN	синус	REAL
SQR	квадратный корень	DINT, REAL
TAN	тангенс	REAL
TOD	преобразование целых чисел в BCD	DINT
TRN	отбросить дробную часть	DINT, REAL
XOR	побитовое "исключающее ИЛИ	"DINT

Форматирование выражений

Для каждого оператора, который вы используете в выражении, вы должны предусмотреть один или два операнда (теги или прямые значения). Используйте следующую таблицу, чтобы отформатировать операторов и операнды внутри выражения:

Для операторов, которые работают:	Используйте этот формат:	Примеры:
с одним операндом	оператор(операнд)	ABS(*tag_a*)
с двумя операндами	операнд_a оператор операнд_b	*tag_b* + 5 *tag_c* AND *tag_d* (*tag_e* ** 2) MOD (*tag_f* / *tag_g*)

Задание порядка операции

Операции, которые вы записываете в выражение, выполняются инструкцией в установленном порядке, не обязательно в том порядке, в котором вы их записываете. Вы можете изменить порядок операции, поместив элементы в скобки и принудив тем самым инструкцию выполнять операцию в скобках раньше других операций.

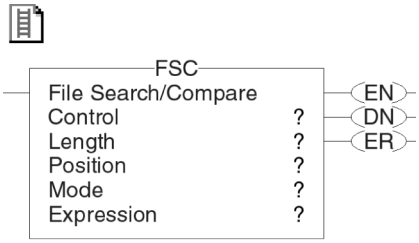
Операции, имеющие один и тот же порядок, выполняются слева направо.

Порядок:	Операция:
1.	()
2.	ABS, ACS, ASN, ATN, COS, DEG, FRD, LN, LOG, RAD, SIN, SQR, TAN, TOD, TRN
3.	**
4.	- (отрицание), NOT
5.	*, /, MOD
6.	- (вычитание), +
7.	AND
8.	XOR
9.	OR

File Search and Compare (FSC) (Файловый поиск и сравнение)

Инструкция FSC сравнивает значения в массиве поэлементно.

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Описание:
Контроль (Control)	CONTROL	тег	управляющая структура для операции
Длина (Length)	DINT	непосредственный	число элементов в массиве для манипулирования
Позиция (Position)	DINT	непосредственный	смещение в массивисходное значение обычно 0

Структура CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения показывает, что инструкция FSC разрешена.
.DN	BOOL	Бит выполнения устанавливается, когда инструкция выполнила операцию с последним элементом (.POS = .LEN).
.ER	BOOL	Бит ошибки не изменен.
.IN	BOOL	Бит запрета показывает, что инструкция FSC обнаружила истинное сравнение. Вы должны сбросить этот бит, чтобы продолжить операцию поиска.
.FD	BOOL	Бит обнаружения показывает, что инструкция FSC обнаружила истинное сравнение.
.LEN	DINT	Длина задает число элементов в массиве, где оперирует инструкция FAL.
.POS	DINT	Позиция содержит позицию текущего элемента, к которому в данный момент обращается инструкция.

Описание:

Если инструкция FSC разрешена и сравнение истинно, инструкция устанавливает бит .FD, а бит .POS отражает позицию массива, где инструкция нашла истинное сравнение. Инструкция устанавливает бит .IN, чтобы предотвратить дальнейший поиск.

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

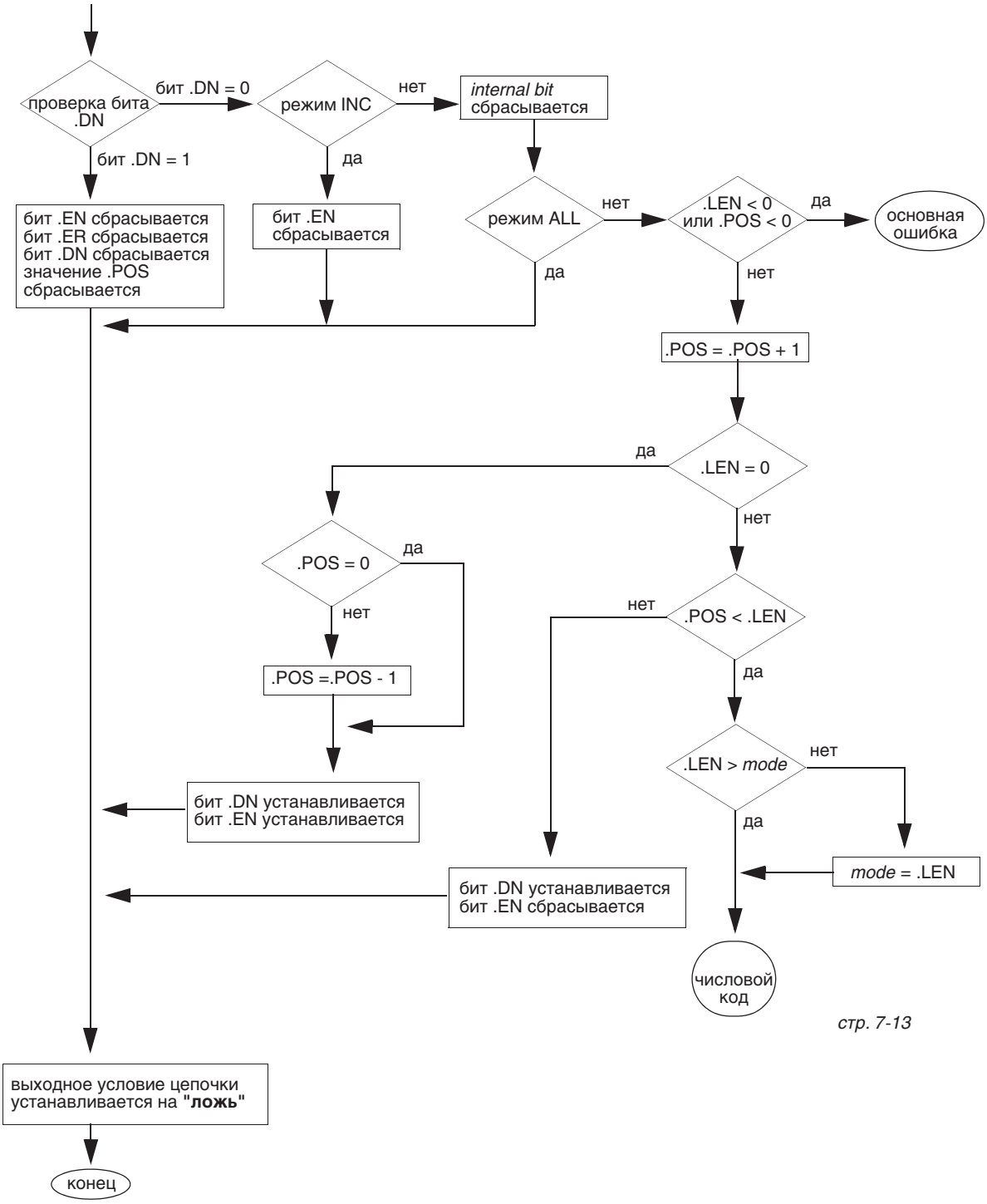
Условия ошибки:

Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
.POS < 0 или .LEN < 0	4	21

Выполнение:

Условие:	**Действие релейной логики:**
предварительное сканирование	Выходное условие цепочки устанавливается на "ложь".

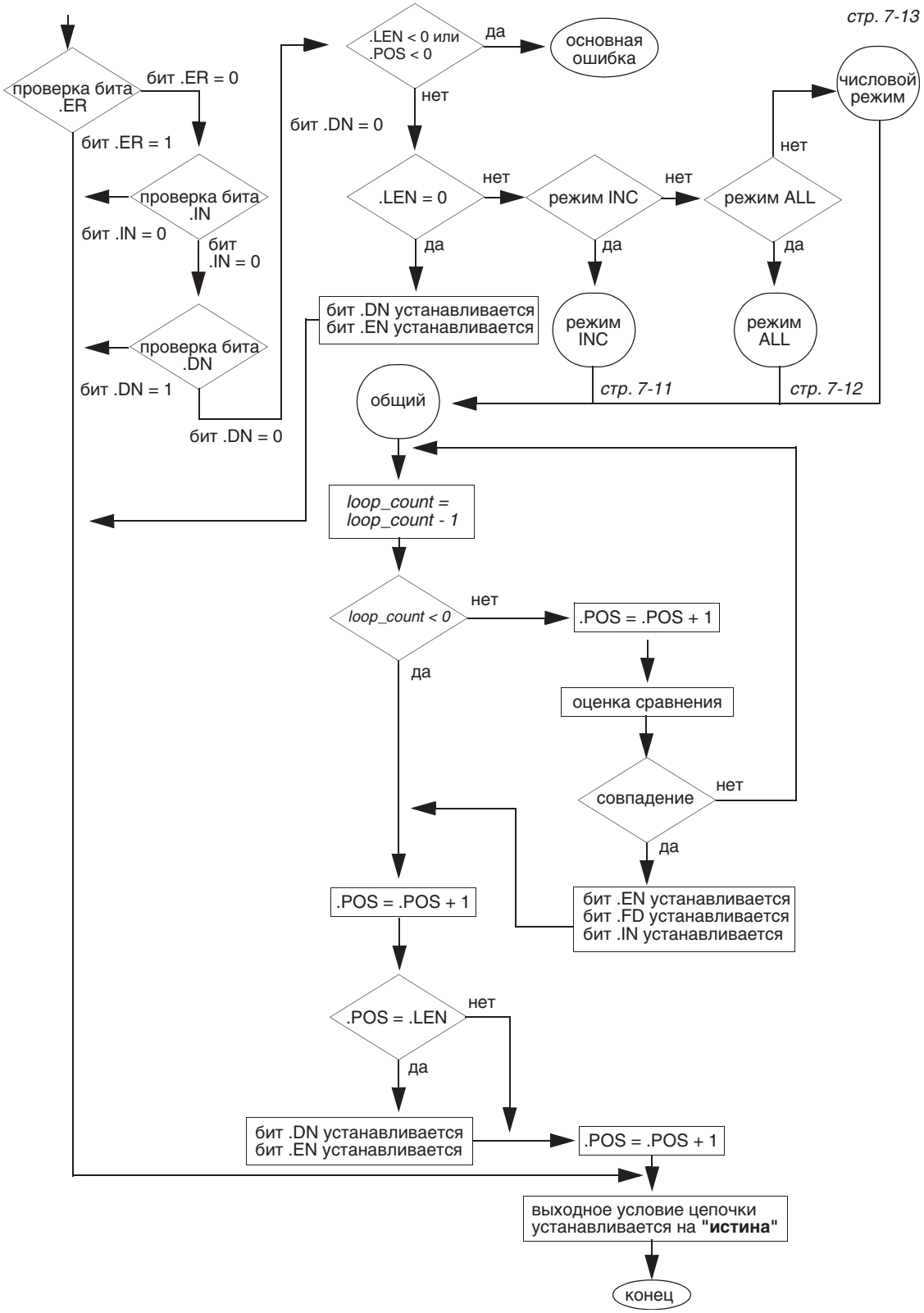
входное условие цепочки - "ложь"



стр. 7-13

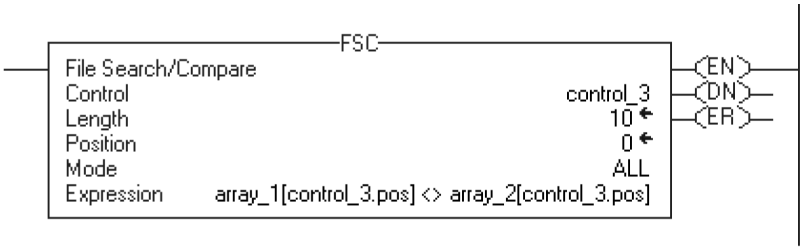
Условие:	**Действие релейной логики:**

входное условие цепочки - "истина"



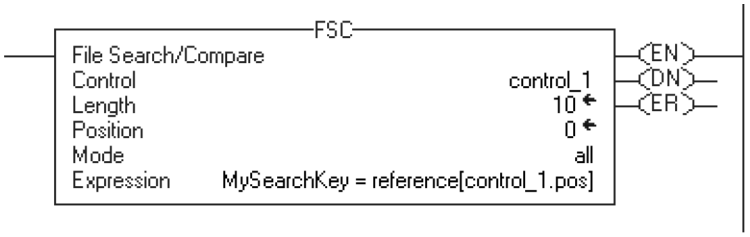
постсканирование	Выходное условие цепочки устанавливается на "ложь"

Пример 1: Поиск совпадения в двух массивах. Когда инструкция FSC разрешена, она сравнивает каждый из первых 10 элементов *array_1* с соответствующими элементами в *array_2*.

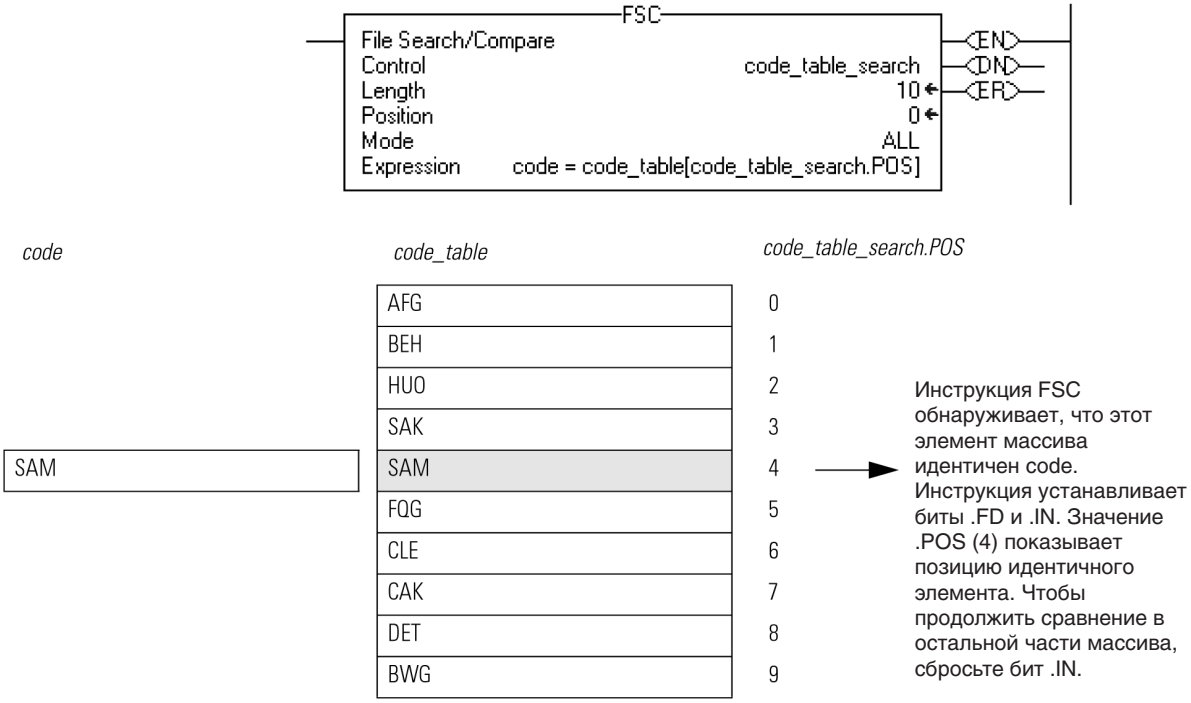


array_1	*array_2*	*control_3.pos*	
00000000000000000000000000000000	00000000000000000000000000000000	0	
00000000000000000000000000000000	00000000000000000000000000000000	1	
00000000000000000000000000000000	00000000000000000000000000000000	2	
00000000000000000000000000000000	00000000000000000000000000000000	3	
00000000000000001111111111111111	11111111111111110000000000000000	4	→ Инструкция FSC обнаруживает, что эти элементы не идентичны. Инструкция устанавливает биты .FD и .IN. Значение .POS (4) показывает позицию элементов, которые не являются идентичными.
11111111111111111111111111111111	11111111111111111111111111111111	5	
11111111111111111111111111111111	11111111111111111111111111111111	6	
11111111111111111111111111111111	11111111111111111111111111111111	7	
11111111111111111111111111111111	11111111111111111111111111111111	8	
11111111111111111111111111111111	11111111111111111111111111111111	9	

Пример 2: Поиск совпадения в массиве. Когда инструкция FSC разрешена, она сравнивает *MySearchKey* с 10 элементами в *array_1*.



Пример 3: Поиск строки в массиве строк. Когда инструкция FSC разрешена, она сравнивает символы в *code* с 10 элементами в *code_table*.



Выражения FSC

Вы программируете выражения в инструкциях FSC таким же образом, как и выражения в инструкциях CMP. Обратитесь к следующим разделам для получения информации о допустимых операторах, формате и порядке операций, которые являются общими для обеих инструкций.

Допустимые операторы

Оператор:	Описание:	Оптимальный:
+	сложение	DINT, REAL
-	вычитание/отрицание	DINT, REAL
*	умножение	DINT, REAL
/	деление	DINT, REAL
=	равно	DINT, REAL
<	меньше	DINT, REAL
<=	меньше или равно	DINT, REAL
>	больше	DINT, REAL
>=	больше или равно	DINT, REAL
<>	неравно	DINT, REAL
**	степень (x в степени y)	DINT, REAL
ABS	абсолютная величина	DINT, REAL
ACS	арккосинус	REAL
AND	побитовое "И"	DINT
ASN	арксинус	REAL
ATN	арктангенс	REAL
COS	косинус	REAL
DEG	преобразование радиан в градусы	DINT, REAL
FRD	преобразование BCD в целые числа	DINT
LN	натуральный логарифм	REAL
LOG	логарифм с основанием 10	REAL
MOD	деление по модулю	DINT, REAL
NOT	побитовое дополнение	DINT
OR	побитовое "ИЛИ"	DINT
RAD	преобразование градусов в радианы	DINT, REAL
SIN	синус	REAL
SQR	квадратный корень	DINT, REAL
TAN	тангенс	REAL
TOD	преобразование целых чисел в BCD	DINT
TRN	отбросить дробную часть	DINT, REAL
XOR	побитовое "исключающее ИЛИ	"DINT

Форматирование выражений

Для каждого оператора, который вы используете в выражении, вы должны предусмотреть один или два операнда (теги или прямые значения). Используйте следующую таблицу, чтобы отформатировать операторов и операнды внутри выражения::

Для операторов, которые работают:	Используйте этот формат:	Примеры:
с одним операндом	оператор(операнд)	ABS(*tag_a*)
с двумя операндами	операнд_a оператор операнд_b	*tag_b* + 5 *tag_c* AND *tag_d* (*tag_e* ** 2) MOD (*tag_f* / *tag_g*)

Задание порядка операции

Операции, которые вы записываете в выражение, выполняются инструкцией в установленном порядке, не обязательно в том порядке, в котором вы их записываете. Вы можете изменить порядок операции, поместив элементы в скобки и принудив тем самым инструкцию выполнять операцию в скобках раньше других операций.

Операции, имеющие один и тот же порядок, выполняются слева направо.

Порядок:	Операция:
1.	()
2.	ABS, ACS, ASN, ATN, COS, DEG, FRD, LN, LOG, RAD, SIN, SQR, TAN, TOD, TRN
3.	**
4.	- (отрицание), NOT
5.	*, /, MOD
6.	<, <=, >, >=, =
7.	- (вычитание), +
8.	AND
9.	XOR
10.	OR

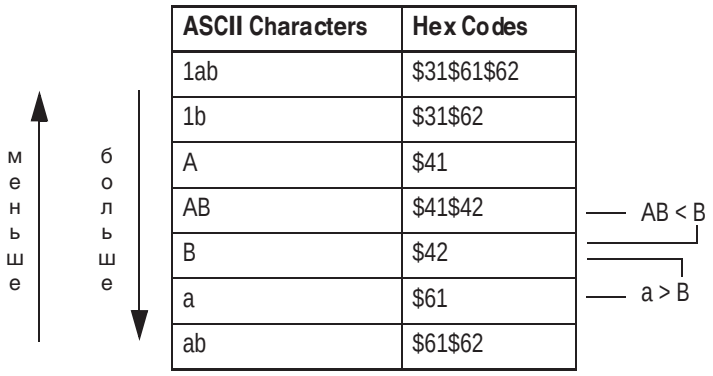
Использование строк в выражении

Чтобы использовать строки символов ASCII в выражении, следуйте ниже приведенным инструкциям:

- Выражение позволяет вам сравнивать два строковых тега.
- Вы *не можете* вводить символы ASCII непосредственно в выражение.
- Разрешается использование только следующих операторов

Оператор:	Описание:
=	равно
<	меньше
<=	меньше или равно
>	больше
>=	больше или равно
<>	неравно

- Строки идентичны, если их символы совпадают.
- Символы ASCII чувствительны к регистру. Заглавная «A» ($41) не идентична строчной «a» ($61).
- Шестнадцатеричные значения символов определяют, больше или меньше одна строка, по сравнению с другой. Для получения информации о шестнадцатеричном коде символов обращайтесь к задней стороне обложки этого руководства.
- Когда две строки рассортированы как в телефонном справочнике, порядок строк показывает, какая строка больше.

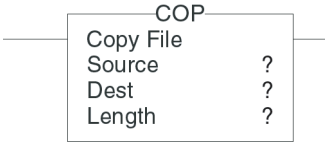


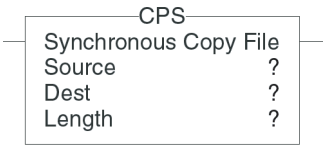
ASCII Characters	Hex Codes
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

Copy File (COP) (Копирование файла) Synchronous Copy File (CPS) (Синхронное копирование файла)

Инструкции COP и CPS копируют значение(я) из Source (источника) в Destination (приемника). Source сохраняется без изменений.

Операнды:





Релейная логика

Операнд:	Тип:	Формат:	Описание:
Источник Source	**SINT** **INT** **DINT** **REAL** **строка** структура	тег	начальный элемент для копирования **Важно**: операнды Source и Destination должны быть одного типа данных, иначе результаты могут быть непредсказуемыми
Приемник Destination	**SINT** **INT** **DINT** **REAL** **строка** структура	тег	начальный элемент для записи поверх Source **Важно**: операнды Source и Destination должны быть одного типа данных, иначе результаты могут быть непредсказуемыми
Длина Length	DINT	непосредственный тег	число элементов Destination для копирования

Структурированный текст

```
COP(Source,Dest,Length);
CPS(Source,Dest,Length);
```

Операнды такие же, как и операнды для инструкций COP и CPS в релейной логике.

Описание:

Во время выполнения инструкций COP и CPS другие действия контроллера могут пытаться прервать операцию копирования и изменить данные источника или приемника:

Если источник или приемник:	И вы хотите:	Тогда выберите:	Примечания:
• произведенный тег • потребляемый тег • данные ввода-вывода • данные, которые может перекрыть другая задача	предотвратить изменение данных во время операции копирования	CPS	• Выполнение задач, которые пытаются прервать инструкцию CPS, откладывается до тех пор, пока инструкция не будет выполнена. • Чтобы оценить время выполнения инструкции CPS, обратитесь к руководству пользователя *ControlLogix System User Manual,* публикация 1756-UM001.
	разрешить изменение данных во время операции копирования	COP	
ничего из выше перечисленного	→	COP	

Количество скопированных байтов равно:

Количество байтов = Length * (количество байтов в типе данных Destination)

ВНИМАНИЕ

Если количество байтов больше, чем длина Source (источника), для оставшихся элементов копирование данных не поддается прогнозированию.

Инструкции COP и CPS производят операции с непрерывной областью памяти данных и выполняют прямое побайтовое копирование сохраненных данных, что требует понимания организации памяти контроллера.

Инструкции COP и CPS не продолжают запись данных, после того как массив заканчивается. Если длина (Length) больше, чем суммарное число элементов в массиве-приемнике (Destination), инструкции COP и CPS перестают выполняться в конце массива. При этом основная ошибка не генерируется.

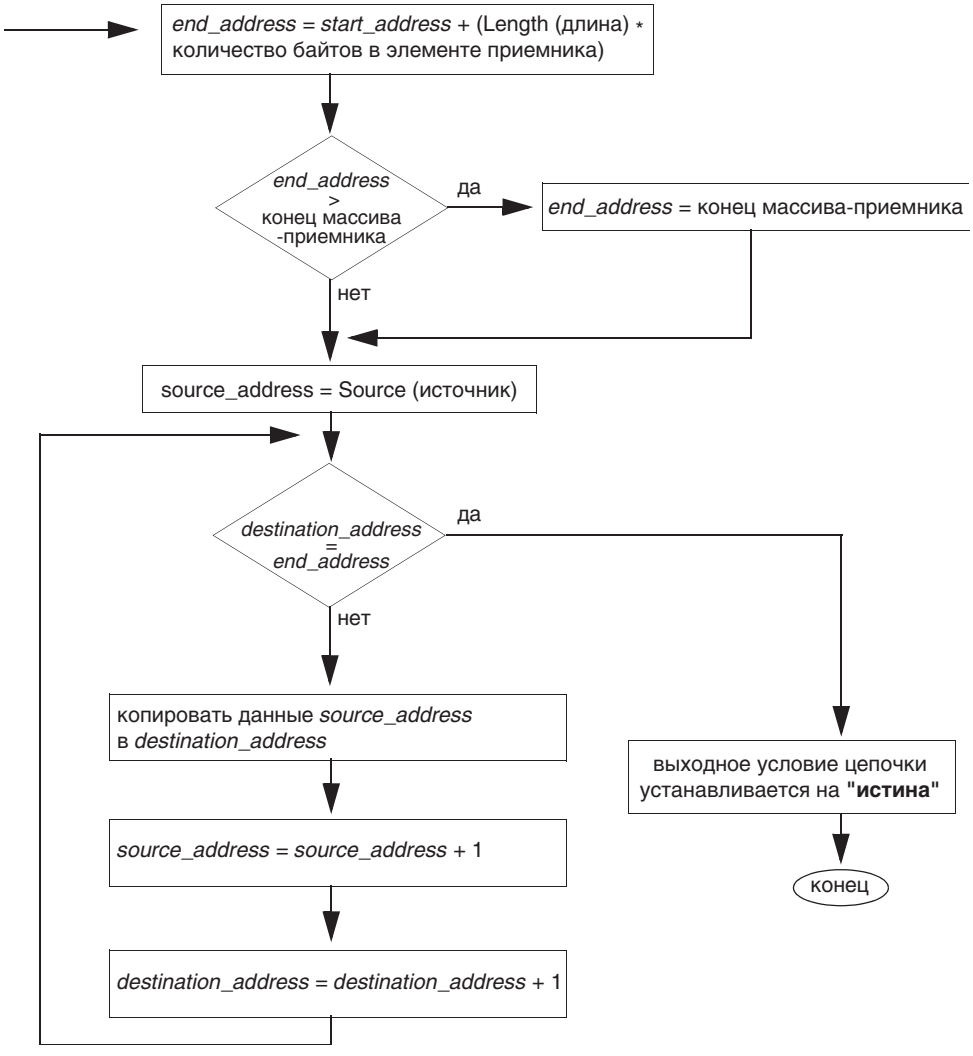
Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.

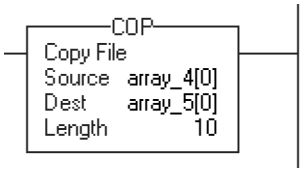
выполнение инструкции



постсканирование	Выходное условие цепочки устанавливается на "ложь".	Никакого действия не производится.

Пример 1: *array_4* и *array_5* имеют один и тот же тип данных. Когда инструкция COP разрешена, она копирует первые 10 элементов *array_4* в первые 10 элементов *array_5*.

Релейная логика

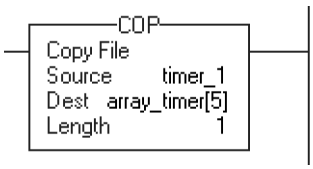


Структурированный текст

```
COP(array_4[0],array_5[0],10);
```

Пример 2: Когда инструкция COP разрешена, она копирует структуру *timer_1* в элемент 5 *array_timer*. Инструкция копирует только одну структуру в один элемент массива.

Релейная логика



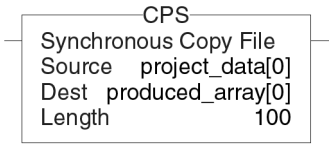
Структурированный текст

```
COP(timer_1,array_timer[5],1);
```

Пример 3: Массив *project_data* (100 элементов) хранит множество значений, которые изменяются в различное время в приложении. Чтобы послать полный образ *project_data* в виде одной копии во времени на другой контроллер, инструкция CRS копирует *project_data* в *produced_array*.

- Пока инструкция CPS копирует данные, ни обновления ввода-вывода, ни выполнение других задач не могут изменять эти данные.
- Тег *produced_array* производит данные в сети ControlNet для дальнейшего использования другими контроллерами.
- Чтобы использовать тот же образ данных (т.е. синхронизированную копию данных), потребляющий контроллер (s) применяет инструкцию CPS для копирования данных из потребляемого тега в другой тег для использования в приложении.

Релейная логика



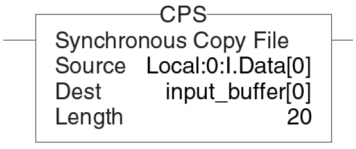
Структурированный текст

```
CPS(project_data[0],produced_array[0],100);
```

Пример 4: *Local:0:IData* хранит входные данные в сети DeviceNet, которая связана с модулем 1756-DNB в слоте 0. Чтобы синхронизировать вводные данные с приложением, инструкция CPS копирует входные данные в *input_buffer*.

- Пока инструкция CPS копирует данные, обновления ввода-вывода не могут изменять эти данные.
- Во время работы приложения оно использует в качестве своих входных данных входные данные *input_buffer*.

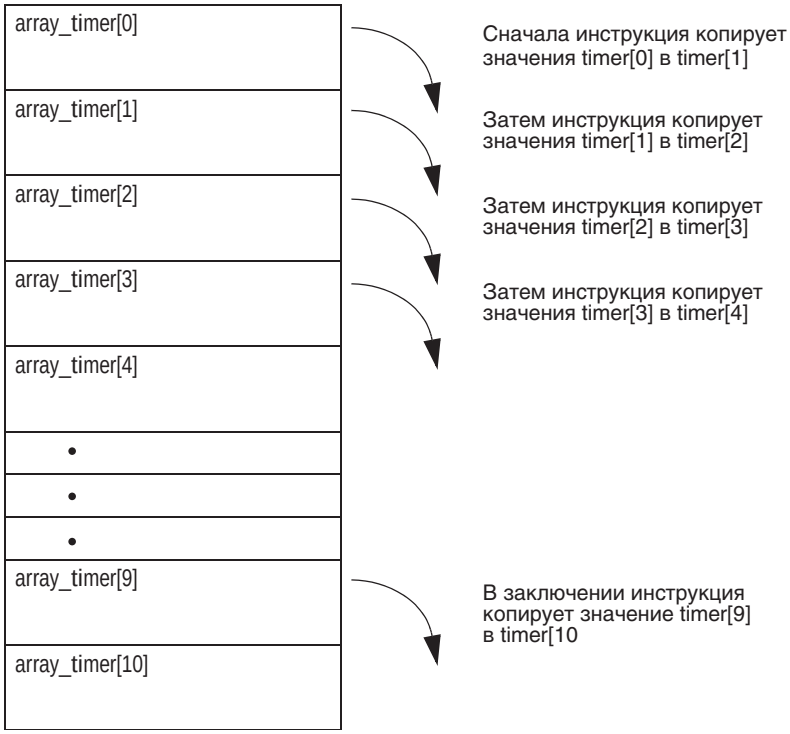
Релейная логика



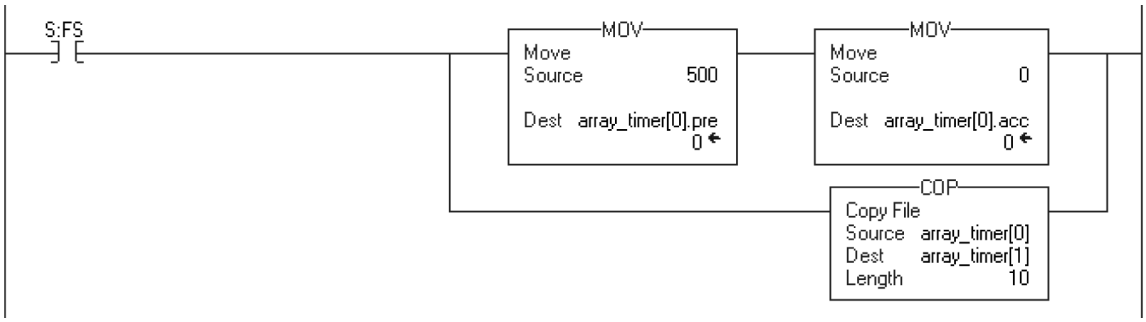
Структурированный текст

```
CPS(Local:0:I.Data[0],input_buffer[0],20);
```

Пример 5: Этот пример описывает, как задаются начальные условия для массива структур таймера. При разрешении инструкции MOV задают начальные значения .PRE и .ACC первого элемента *array_timer*. Когда инструкция COP разрешена, она копирует непрерывный блок байтов, начиная с *array_timer*[0]. Длина составляет девять структур таймера.



Релейная логика



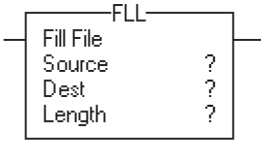
Структурированный текст

```
IF  S:FS  THEN
     array_timer[0].pre  :=  500;
     array_timer[0].acc  :=  0;
     COP(array_timer[0],array_timer[1],10);
END_IF;
```

File Fill (FLL) (Заполнение массива данными)

Инструкция FLL заполняет элементы массива значением Source (источника). Source сохраняется без изменений.

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Описание:
Источник Source	**SINT** **INT** **DINT** **REAL**	непосредственный тег	элемент для копирования **Важно**: операнды Source и Destination должны быть одного типа данных, иначе результаты могут быть непредсказуемыми
Приемник Destination	**SINT** **INT** **DINT** **REAL** структура	тег	начальный элемент для записи поверх Source **Важно**: операнды Source и Destination должны быть одного типа данных, иначе результаты могут быть непредсказуемыми Предпочтительный способ задания начальной структуры – использование инструкции COP.
Длина Length	DINT	непосредственный	число элементов для заполнения

Структурированный текст

В структурированном тексте инструкция FLL отсутствует, но можно получить тот же результат, используя инструкцию SIZE и конструкцию FOR...DO или другую циклическую конструкцию.

```
SIZE(destination,0,length);
FOR position = 0 TO length-1 DO
     destination[position] := source;
END_FOR;
```

Информацию о синтаксисе конструкций структурированного текста можно найти в Приложении C.

Описание:

Количество заполненных байтов равно:

Количество байтов = Length * (количество байтов в типе данных Destination)

Инструкция FLL производит операции с непрерывной областью памяти данных

Инструкция FLL не продолжает запись данных, после того как массив заканчивается. Если длина (Length) больше, чем суммарное число элементов в массиве-приемнике (Destination), инструкция FLL перестает выполняться в конце массива. При этом основная ошибка не генерируется.

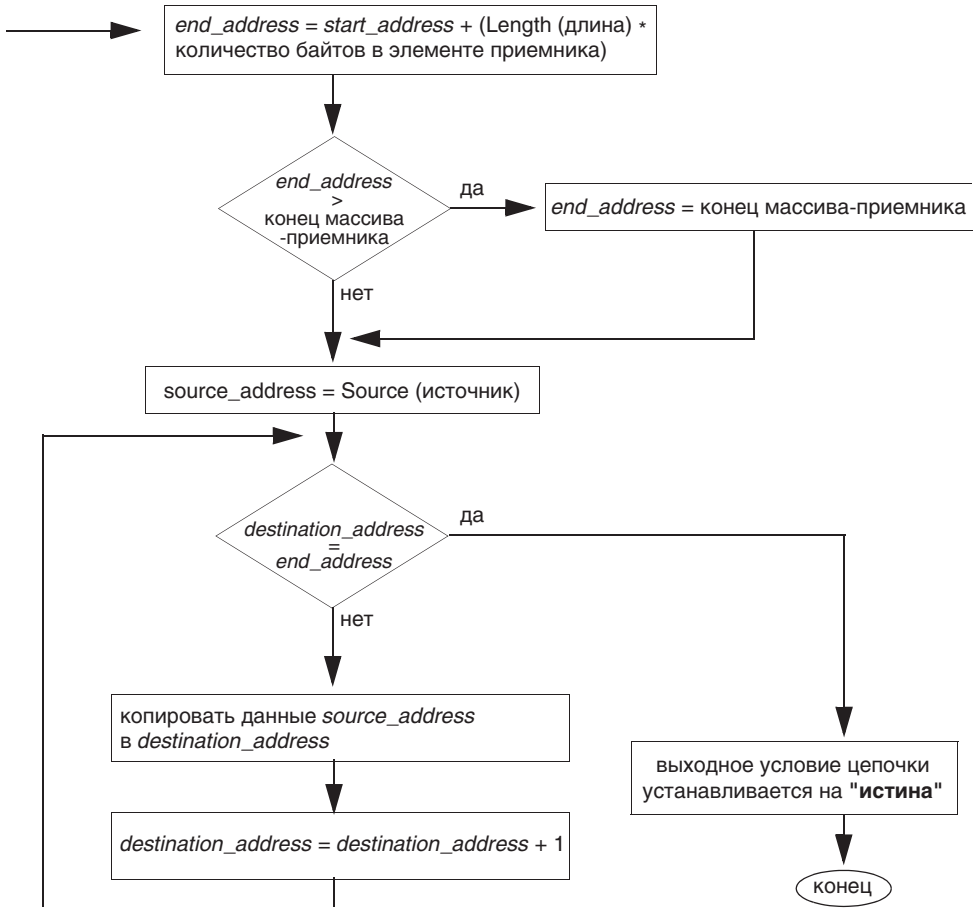
Для достижения наилучших результатов Source и destination должны иметь одинаковый тип данных. Если вы хотите заполнить структуру, используйте инструкцию COP (см. пример 3 на стр. 7-32). Если вы смешиваете типы данных для Source и Destination, элементы Destination будут заполнены преобразованными значениями Source.

Если Source:	Если Destination:	Source будет преобразован в:
SINT, INT, DINT или REAL	SINT	SINT
SINT, INT, DINT или REAL	INT	INT
SINT, INT, DINT или REAL	DINT	DINT
SINT, INT, DINT или REAL	REAL	REAL
SINT	структура	SINT (не преобразованный)
INT	структура	INT (не преобразованный)
DINT	структура	DINT (не преобразованный)
REAL	структура	REAL (не преобразованный)

Арифметические флаги состояния: не затрагиваются

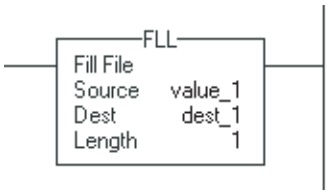
Условия ошибки: отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки - "истина"	
постсканирование	Выходное условие цепочки устанавливается на "ложь".

Пример: Инструкция FLL копирует значение *value_1* в *dest_1*.

Релейная логика



Тип данных Source (*value_1*)	Значение Source (*value_1*)	Тип данных Destination (*dest_1*)	Значение Destination (*dest_1*) после выполнения FLL:
SINT	16#80 (-128)	DINT	16#FFFF FF80 (-128)
DINT	16#1234 5678	SINT	16#78
SINT	16#01	REAL	1.0
REAL	2.0	INT	16#0002
SINT	16#01	TIMER	16#0101 0101 16#0101 0101 16#0101 0101
INT	16#0001	TIMER	16#0001 0001 16#0001 0001 16#0001 0001
DINT	16#0000 001	TIMER	16#0000 0001 16#0000 0001 16#0000 0001

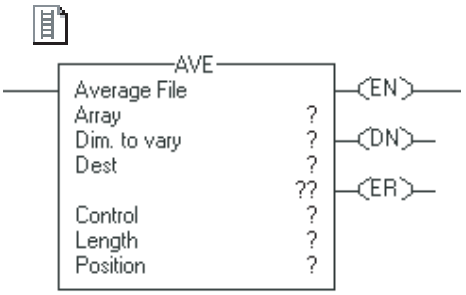
Структурированный текст

```
dest_1 := value_1;
```

File Average (AVE) (Файловое усреднение)

Инструкции AVE вычисляет среднее значение среди набора значений.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Массив Array	SINT INT DINT **REAL**	тег массива	нахождение среднего значения в этом массиве задание первого элемента из группы элементов для нахождения среднего **не используйте** CONTROL.POS в нижнем индексе
Размерность для изменения Dimension to vary	DINT	непосредственный (0, 1, 2)	то, какую размерность использовать, зависит от количества размерностей, существует следующий порядок array[dim_0,dim_1,dim_2] array[dim_0,dim_1] array[dim_0]
Приемник Destination	SINT INT DINT **REAL**	тег	результат операции
Контроль Control	CONTROL	тег	управляющая структура для операции
Длина Length	DINT	непосредственный	число элементов в массиве, в котором вычисляется среднее значение
Позиция Position	DINT	непосредственный	текущий элемент в массиве исходное значение обычно 0

Структурированный текст

В структурированном тексте инструкция AVE отсутствует, но можно получить тот же результат, используя инструкцию SIZE и конструкцию FOR...DO или другую циклическую конструкцию.

```
SIZE(array,0,length);
sum := 0;
FOR position = 0 TO length-1 DO
     sum := sum + array[position];
END_FOR;
destination := sum / length;
```

Информацию о синтаксисе конструкций структурированного текста можно найти в Приложении C.

Структура CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения показывает, что инструкция AVE разрешена.
.DN	BOOL	Бит выполнения устанавливается, когда инструкция выполнила операцию с последним элементом в массиве (Array) (.POS = .LEN).
.ER	BOOL	Бит ошибки устанавливается, если выражение генерирует переполнение. Выполнение инструкции останавливается до тех пор, пока процедура не сбросит бит .ER. Позиция элемента, ставшего причиной переполнения, хранится в значении .POS.
.LEN	DINT	Длина задает число элементов в массиве, где оперирует инструкция.
.POS	DINT	Позиция содержит позицию текущего элемента, к которому в данный момент обращается инструкция.

Описание: Инструкция AVE вычисляет среднее значение среди набора значений.

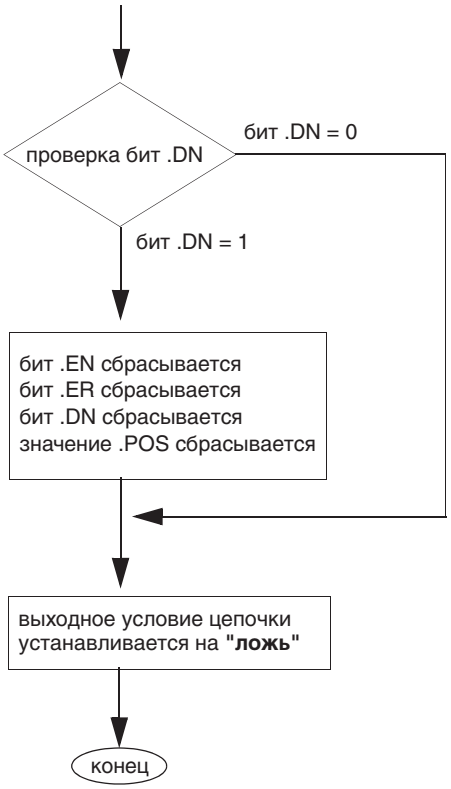
ВАЖНО! Убедитесь, что значение Length (длина) не является причиной того, что инструкция превышает заданное значение Dimension to vary (размерности для изменения). Если же это происходит, значение Destination (приемника) будет некорректным.

Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

Условия ошибки:

Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
.POS < 0 или .LEN < 0	4	21
Для заданного массива не существует Dimension to vary	4	20

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Бит .EN сбрасывается. Бит .DN сбрасывается. Бит .ER сбрасывается. Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	
входное условие цепочки – «истина»	Инструкция AVE вычисляет среднее значение путем сложения всех заданных элементов в массиве и деления суммы на число элементов. Внутри инструкция использует инструкцию FAL для вычисления среднего значения: Expression (выражение) = вычисление среднего Mode (режим) = ALL Для получения дополнительной информации о том, как работает инструкция FAL, обращайтесь к стр. 7-9.
постсканирование	Выходное условие цепочки устанавливается на «ложь».

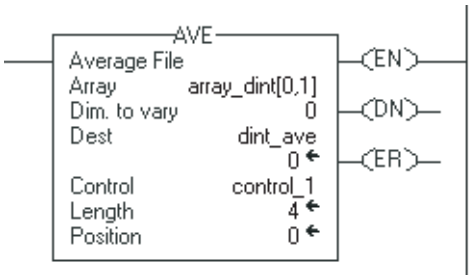
Пример 1: Среднее *array_dint*, представляющее собой DINT[4,5].

нижние индексы	размерность 1				
	0	1	2	3	4
размерность 0 0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

$$AVE = \frac{19 + 14 + 9 + 4}{4} = \frac{46}{4} = 11.5$$

$dint\_ave = 12$

Релейная логика



Структурированный текст

```
SIZE(array_dint,0,length);
sum := 0;
FOR position = 0 TO (length-1) DO
     sum := sum + array_dint[position];
END_FOR;
dint_ave := sum / length;
```

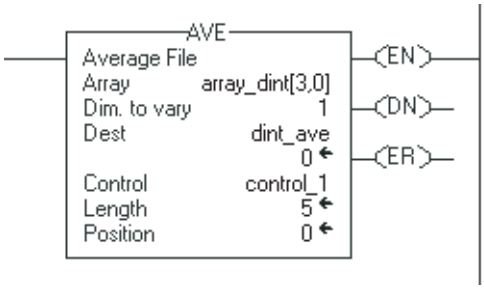
Пример 2: Среднее *array_dint*, представляющее собой DINT[4,5].

нижние индексы	размерность 1				
размерность 0	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

$$AVE = \frac{5+4+3+2+1}{5} = \frac{15}{5} = 3$$

$$dint\_ave = 3$$

Релейная логика



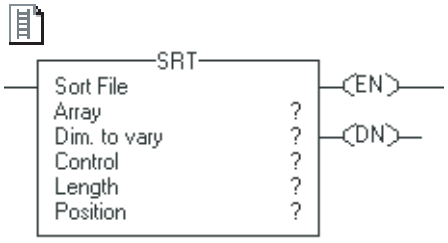
Структурированный текст

```
SIZE(array_dint,1,length);
sum := 0;
FOR position = 0 TO (length-1) DO
     sum := sum + array_dint[position];
END_FOR;
dint_ave := sum / length;
```

File Sort (SRT) (Сортировка файла)

Инструкция SRT сортирует набор значений в одной размерности (Dim to vary) массива в порядке возрастания.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Массив Array	**SINT** **INT** **DINT** **REAL**	тег массива	массив для сортировки задание первого элемента из группы элементов для сортировки **не используйте** CONTROL.POS в нижнем индексе
Размерность для изменения Dimension to vary	DINT	непосредственный (0, 1, 2)	то, какую размерность использовать, зависит от количества размерностей, существует следующий порядок array[dim_0,dim_1,dim_2] array[dim_0,dim_1] array[dim_0]
Контроль Control	CONTROL	тег	управляющая структура для операции
Длина Length	DINT	непосредственный	число элементов в массиве, в котором проводится сортировка
Позиция Position	DINT	непосредственный	текущий элемент в массиве исходное значение обычно 0

Структурированный текст

```
SRT(Array,Dimtovary,
    Control);
```

Операнды такие же, как и операнды для инструкции SRT в релейной логике. Однако вы задаете значения Length и Dimension, обращаясь к членам .LEN и .POS структуры CONTROL, а не включая значения в перечень операндов.

Структура CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения показывает, что инструкция SRT разрешена.
.DN	BOOL	Бит выполнения устанавливается, когда сортировка заданных элементов закончена.
.ER	BOOL	Бит ошибки устанавливается, когда .LEN < 0 или .POS < 0. Любое из этих условий генерирует основную ошибку.
.LEN	DINT	Длина задает число элементов в массиве, где оперирует инструкция.
.POS	DINT	Позиция содержит позицию текущего элемента, к которому в данный момент обращается инструкция.

Описание:

Инструкция SRT сортирует набор значений в одной размерности (Dim to vary) массива в порядке возрастания.

ВАЖНО!

Убедитесь, что значение Length (длина) не является причиной того, что инструкция превышает заданное значение Dimension to vary (размерности для изменения). Если же это происходит, значение Destination (приемника) будет некорректным.

Это промежуточная инструкция:

- В релейной логике переключайте входное условие цепочки с положения «сброшен» в положение «установлен» каждый раз, когда необходимо выполнение инструкции.
- В структурированном тексте задайте условия выполнения инструкции таким образом, чтобы она выполнялась только во время перехода состояний. Обратитесь в Приложению C.

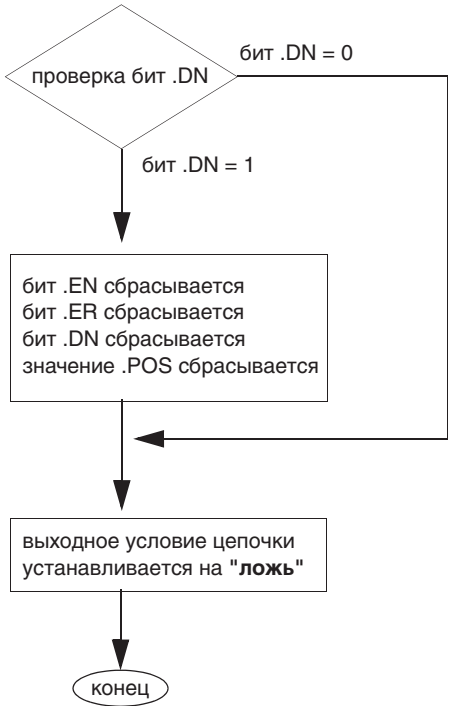
Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

Условия ошибки:

Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
.POS < 0 или .LEN < 0	4	21
Для заданного массива не существует Dimension to vary	4	20
Инструкция пытается получить доступ к данным, находящимся за пределами массива	4	20

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Бит .EN сбрасывается. Бит .DN сбрасывается. Бит .ER сбрасывается. Выходное условие цепочки устанавливается на «ложь».	Бит .EN сбрасывается. Бит .DN сбрасывается. Бит .ER сбрасывается.
входное условие цепочки - "ложь"		не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция сортирует заданные элементы массива в порядке возрастания	инструкция сортирует заданные элементы массива в порядке возрастания
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример 1: Сортировка *int_array*, представляющая собой DINT[4,5].

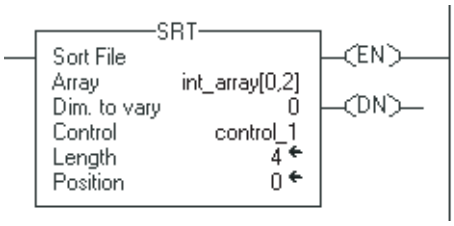
До

нижние индексы	размерность 1				
размерность 0	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

После

нижние индексы	размерность 1				
размерность 0	0	1	2	3	4
0	20	19	3	17	16
1	15	14	8	12	11
2	10	9	13	7	6
3	5	4	18	2	1

Релейная логика



Структурированный текст

```
control_1.LEN  :=  4;
control_1.POS  :=  0;
SRT(int_array[0,2],0,control_1);
```

Пример 2: Сортировка *int_array*, представляющая собой DINT[4,5].

До

нижние индексы / размерность 0	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

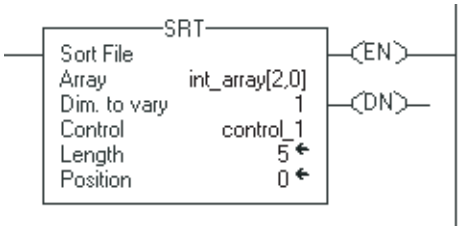
размерность 1

После

нижние индексы / размерность 0	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	6	7	8	9	10
3	5	4	3	2	1

размерность 1

Релейная логика



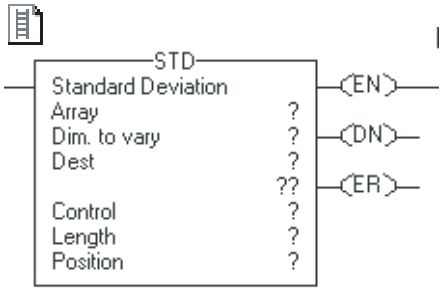
Структурированный текст

```
control_1.LEN  :=  5;
control_1.POS  :=  0;
SRT(int_array[2,0],1,control_1);
```

File Standard Deviation (STD) (Стандартное отклонение для массива)

Инструкция STD вычисляет стандартное отклонение для набора значений в одной размерности массива и хранит результат в Destination (приемнике).

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Массив Array	SINT INT **DINT** **REAL**	тег массива	нахождение стандартного отклонения значений в этом массиве задание первого элемента из группы элементов, которая будет использоваться при вычислении стандартного отклонения **не используйте** CONTROL.POS в нижнем индексе
	Тег SINT или INT конвертируется в значение DINT посредством дополнительного знакового разряда.		
Размерность для изменения Dimension to vary	DINT	непосред-ственный (0, 1, 2)	то, какую размерность использовать, зависит от количества размерностей, существует следующий порядок array[dim_0,dim_1,dim_2] array[dim_0,dim_1] array[dim_0]
Приемник Destination	REAL	тег	результат операции
Контроль Control	CONTROL	тег	управляющая структура для операции
Длина Length	DINT	непосред-ственный	число элементов в массиве, в котором вычисляется стандартное отклонение
Позиция Position	DINT	непосред-ственный	текущий элемент в массиве исходное значение обычно 0

Структура CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения показывает, что инструкция STD разрешена.
.DN	BOOL	Бит выполнения устанавливается, когда вычисление завершено.
.ER	BOOL	Бит ошибки устанавливается, если выражение генерирует переполнение. Выполнение инструкции останавливается до тех пор, пока процедура не сбросит бит .ER. Позиция элемента, ставшего причиной переполнения, хранится в значении .POS.
.LEN	DINT	Длина задает число элементов в массиве, где оперирует инструкция.
.POS	DINT	Позиция содержит позицию текущего элемента, к которому в данный момент обращается инструкция.

Структурированный текст

В структурированном тексте инструкция STD отсутствует, но можно получить тот же результат, используя инструкцию SIZE и конструкцию FOR...DO или другую циклическую конструкцию.

```
SIZE(array,0,length);
sum := 0;
FOR position = 0 TO length-1 DO
     sum := sum + array[position];
END_FOR;
average := sum / length;
sum := 0;
FOR position = 0 TO length-1 DO
     sum := sum + ((array[position] - average)**2);
END_FOR;
destination := SQRT(sum /(length-1));
```

Информацию о синтаксисе конструкций структурированного текста можно найти в Приложении C.

Описание: Стандартное отклонение вычисляется по формуле:

$$\text{Стандартное отклонение} = \sqrt{\frac{\left(\sum_{i=1}^{N}[\langle X_{(start+1)} - AVE\rangle^2]\right)}{(N-1)}}$$

Где:

- start = нижний индекс dimension-to-vary (размерности для изменения) операнда массива
- x_i = переменный элемент в массиве
- N = число заданных элементов в массиве
- AVE =

$$\frac{\left(\sum_{i=1}^{N} X_{(start+1)}\right)}{N}$$

ВАЖНО! Убедитесь, что значение Length (длина) не является причиной того, что инструкция превышает заданное значение Dimension to vary (размерности для изменения). Если же это происходит, значение Destination (приемника) будет некорректным.

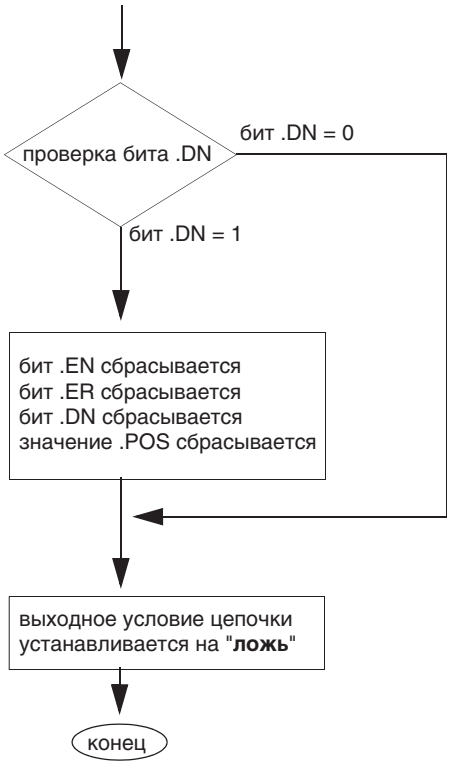
Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

Условия ошибки:

Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
.POS < 0 или .LEN < 0	4	21
Для заданного массива не существует Dimension to vary	4	20

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Бит .EN сбрасывается. Бит .DN сбрасывается. Бит .ER сбрасывается. Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки - "ложь"	
входное условие цепочки - "истина"	Инструкция STD вычисляет стандартное отклонение для заданных элементов.Внутри инструкция использует инструкцию FAL для вычисления среднего значения:Expression (выражение) = вычисление стандартного отклоненияMode (режим) = ALLДля получения дополнительной информации о том, как работает инструкция FAL, обращайтесь к стр. 7-9.
постсканирование	Выходное условие цепочки устанавливается на "ложь".

Пример 1: Вычисление стандартного отклонения *dint_array*, представляющее собой DINT[4,5].

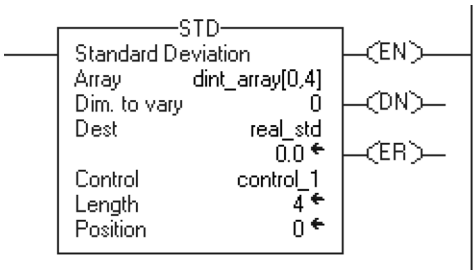
$$AVE = \frac{16 + 11 + 6 + 1}{4} = \frac{34}{4} = 8.5$$

нижние индексы / размерность 0	размерность 1: 0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

$$STD = \sqrt{\frac{\langle 16 - 8.5\rangle^2 + \langle 11 - 8.5\rangle^2 + \langle 6 - 8.5\rangle^2 + \langle 1 - 8.5\rangle^2}{\langle 4 - 1\rangle}} = 6.454972$$

real_std = 6.454972

Релейная логика



Структурированный текст

```
SIZE(dint_array,0,length);
sum := 0;
FOR position = 0 TO (length-1) DO
     sum := sum + dint_array[position];
END_FOR;
average := sum / length;
sum := 0;
FOR position = 0 TO (length-1) DO
     sum := sum + ((dint_array[position] - average)**2);
END_FOR;
real_std := SQRT(sum / (length-1));
```

Пример 2: Вычисление стандартного отклонения *dint_array*, представляющее собой DINT[4,5].

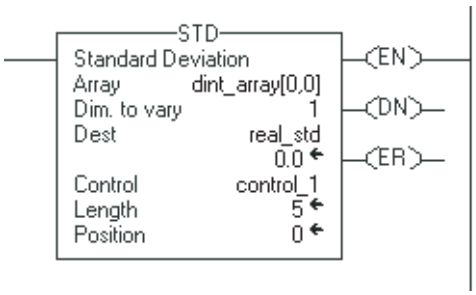
$$AVE = \frac{20 + 19 + 18 + 17 + 16}{5} = \frac{90}{5} = 18$$

нижние индексы	размерность 1				
размерность 0	0	1	2	3	4
0	20	19	18	17	16
1	15	14	13	12	11
2	10	9	8	7	6
3	5	4	3	2	1

$$STD = \sqrt{\frac{\langle 20 - 18\rangle^2 + \langle 19 - 18\rangle^2 + \langle 18 - 18\rangle^2 + \langle 17 - 18\rangle^2 + \langle 16 - 18\rangle^2}{\langle 5 - 1\rangle}} = 1.581139$$

real_std = 1.581139

Релейная логика



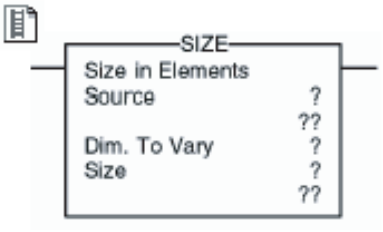
Структурированный текст

```
SIZE(dint_array,1,length);
sum := 0;
FOR position = 0 TO (length-1) DO
     sum := sum + dint_array[position];
END_FOR;
average := sum / length;
sum := 0;
FOR position = 0 TO (length-1) DO
     sum := sum + ((dint_array[position] - average)**2);
END_FOR;
real_std := SQRT(sum / (length-1));
```

Size in Elements (SIZE) (Размер в элементах)

Инструкция SIZE находит размер размерности массива.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:	
Источник Source	SINT INT DINT REAL структура строка	тег массива	массив, где работает инструкция	
Размерность для изменения Dimension to vary	DINT	непосредственный (0, 1, 2)	используемая размерность:	
			Для размера:	**Введите:**
			первая размерность	0
			втораяразмерность	1
			третья размерность	2
Размер Size	SINT INT DINT REAL	тег	тег для хранения количества элементов в заданной размерности массива	

Структурированный текст

```
SIZE(Source,Dimtovary,Size);
```

Операнды такие же, как и операнды для инструкции SIZE в релейной логике.

Описание:

Инструкция SIZE находит число элементов (размер) в обозначенной размерности массива Source и помещает результат в операнд Size.

- Инструкция находит размер одной размерности массива.
- Инструкция оперирует с:
- - массивом
- - массивом в структуре
- - массивом, являющимся частью большего массива

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

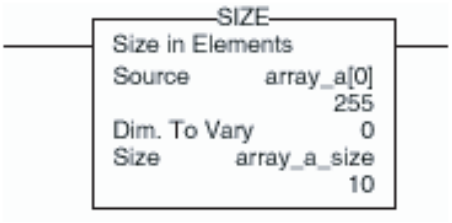
отсутствуют.

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция находит размер размерности.	Инструкция находит размер размерности.
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится

Пример 1: Нахождение числа элементов в размерности 0 (первая размерность) *array_a*. Хранение размера в *array_a_size*. В этом примере размерность 0 *array_a* имеет 10 элементов.

Релейная логика

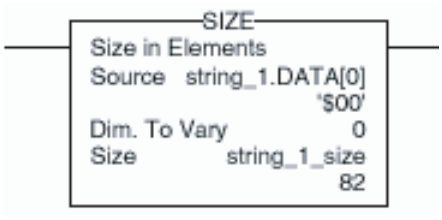


Структурированный текст

```
SIZE(array_a,0,array_a_size);
```

Пример 2: Нахождение числа элементов в члене DATA *string_1*, являющемся строкой. Хранение размера в *string_1_size*. В этом примере член DATA *string_1* имеет 82 элемента. (Строка использует тип данных STRING, установленный по умолчанию). Поскольку каждый элемент содержит один символ, *string_1* может содержать до 82 символов.

Релейная логика

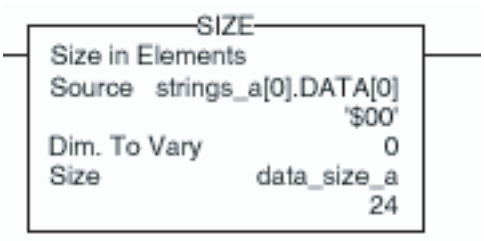


Структурированный текст

```
SIZE(string_1.DATA[0],0,string_1_size);
```

Пример 3: *String_a* является массивом строковых структур. Инструкция SIZE находит число элементов в члене DATA строковой структуры и хранит размер в *data_size_a*. В этом примере член DATA имеет 24 элемента. (Строковая структура имеет заданную пользователем длину – 24.)

Релейная логика



Структурированный текст

```
SIZE(strings_a[0].DATA[0],0,data_size_a);
```

Примечания:

Инструкции Массива(Файла)/Сдвига (BSL, BSR, FFL, FFU, LFL, LFU)

Введение

Используйте инструкции массива (файла)/сдвига, чтобы модифицировать положение данных в массивах.

Если вы хотите:	Используйте эту инструкцию:	Имеющуюся в этих языках:	См. стр.
Загрузить биты в массив битов, сдвинуть биты в массиве и выгрузить биты из массива битов с частотой один бит за раз.	BSL	релейная логика	8-2
	BSR	релейная логика	8-5
Загрузить и выгрузить значения в том же порядке.	FFL	релейная логика	8-8
	FFU	релейная логика	8-14
Загрузить и выгрузить значения в обратном порядке.	LFL	релейная логика	8-20
	LFU	релейная логика	8-26

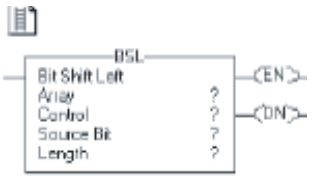
Вы можете смешивать типы данных, но это может привести к потере точности и ошибке округления.

Для инструкций релейной логики, **жирный** шрифт типов данных означает, что это оптимальные типы данных. Инструкция выполняется быстрее и требует меньший объем памяти, если все операнды инструкции используют один и тот же оптимальный тип данных, обычно DINT или REAL.

Bit Shift Left (BSL) (Сдвиг бита влево)

Инструкция BSL сдвигает заданные биты внутри массива на одну позицию влево.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Array	DINT	тег массива	массив для модификации задание первого элемента из группы элементов **не** используйте CONTROL.POS в нижнем индексе
Control	CONTROL	тег	управляющая структура для операции
Source bit	BOOL	тег	бит для сдвига
Length	DINT	непосредственный	число битов в массиве для сдвига

Структура CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция BSL разрешена.
.DN	BOOL	Бит выполнения устанавливается, чтобы показать, что биты сдвинуты на одну позицию влево.
.UL	BOOL	Бит выгрузки является выходом инструкции. Бит .UL хранит статус бита, который был сдвинут из диапазона битов.
.ER	BOOL	Бит ошибки устанавливается, когда .LEN < 0.
.LEN	DINT	Длина задает число битов массива, которые будут сдвинуты.

Описание:

Когда инструкция разрешена, она выгружает самый старший бит из числа заданных битов в бит .UL, сдвигает оставшиеся биты на одну позицию влево и загружает бит Source (источника) в бит 0 массива.

Инструкция BSL производит операции с непрерывной областью памяти данных.

Арифметические флаги состояния:

не затрагиваются

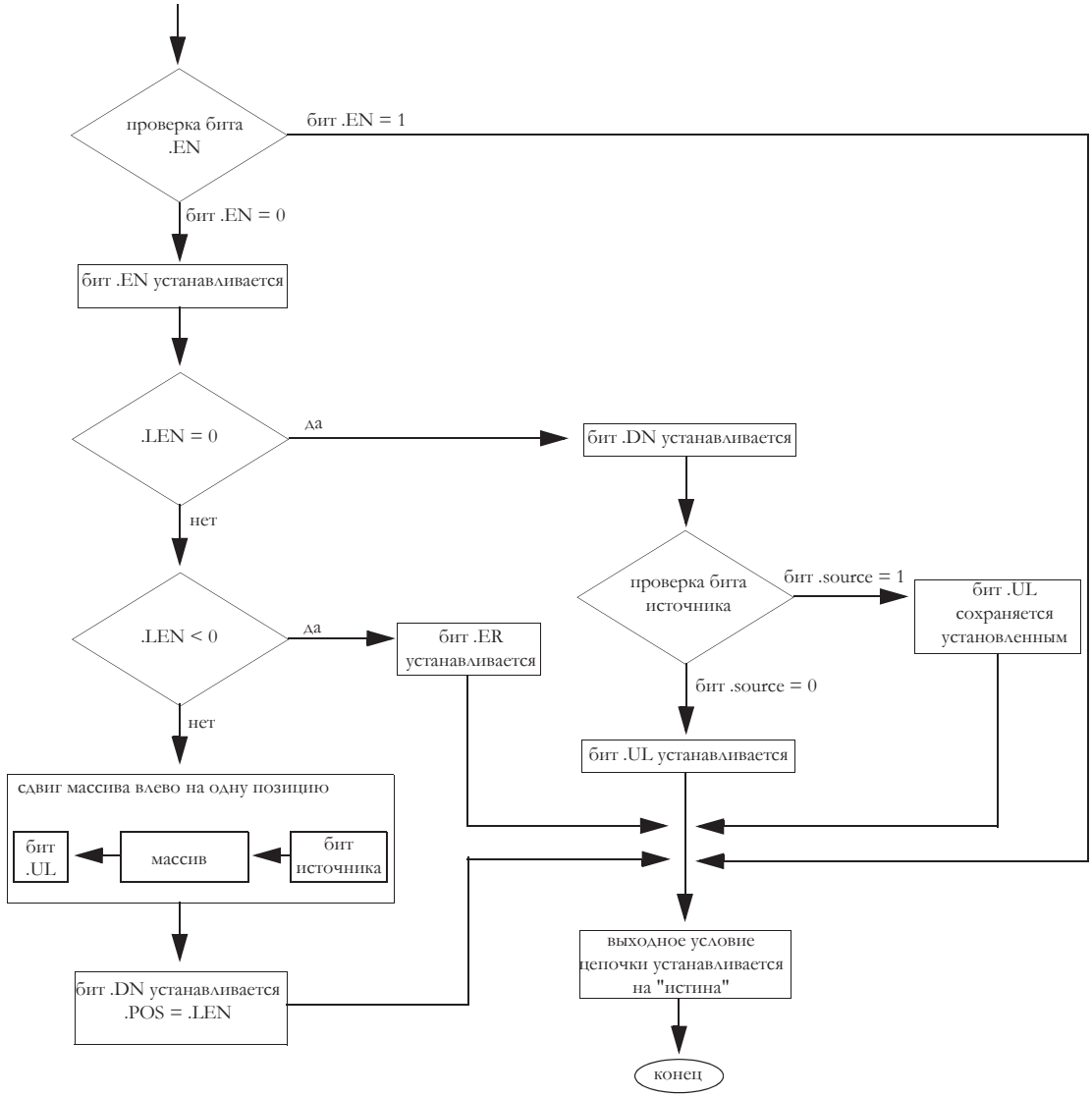
Условия ошибки:

отсутствуют

Выполнение:

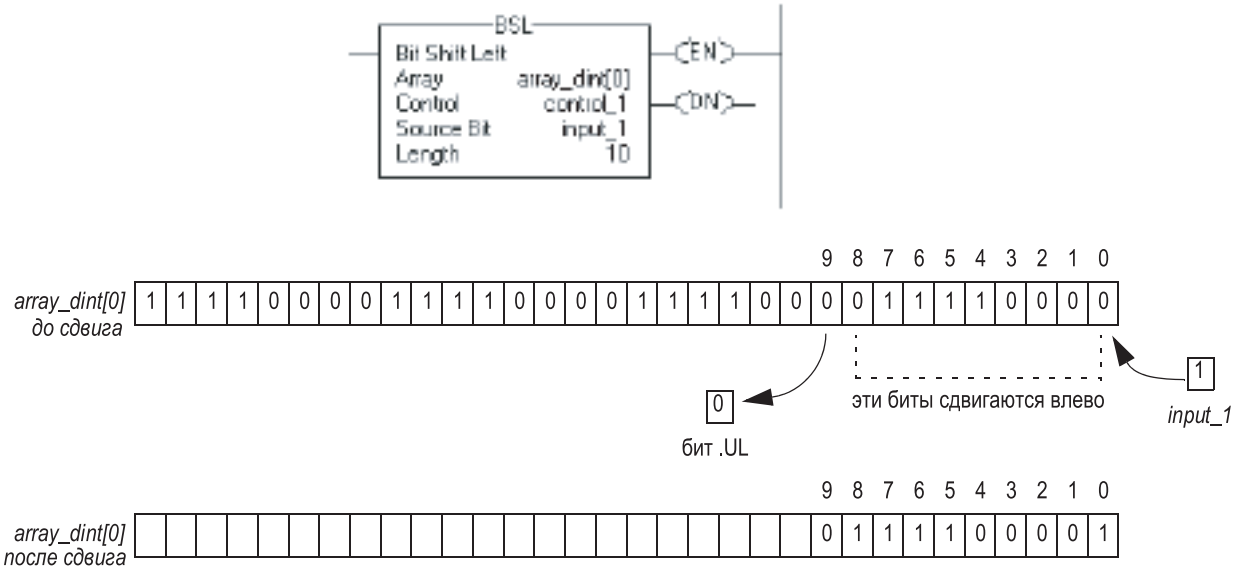
Условие:	**Действие релейной логики:**
предварительное сканирование	Бит .EN сбрасывается. Бит .DN сбрасывается. Бит .ER сбрасывается. Значение .POS сбрасывается. Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Бит .EN сбрасывается. Бит .DN сбрасывается. Бит .ER сбрасывается. Значение .POS сбрасывается. Выходное условие цепочки устанавливается на «ложь».

входное условие цепочки - "истина"

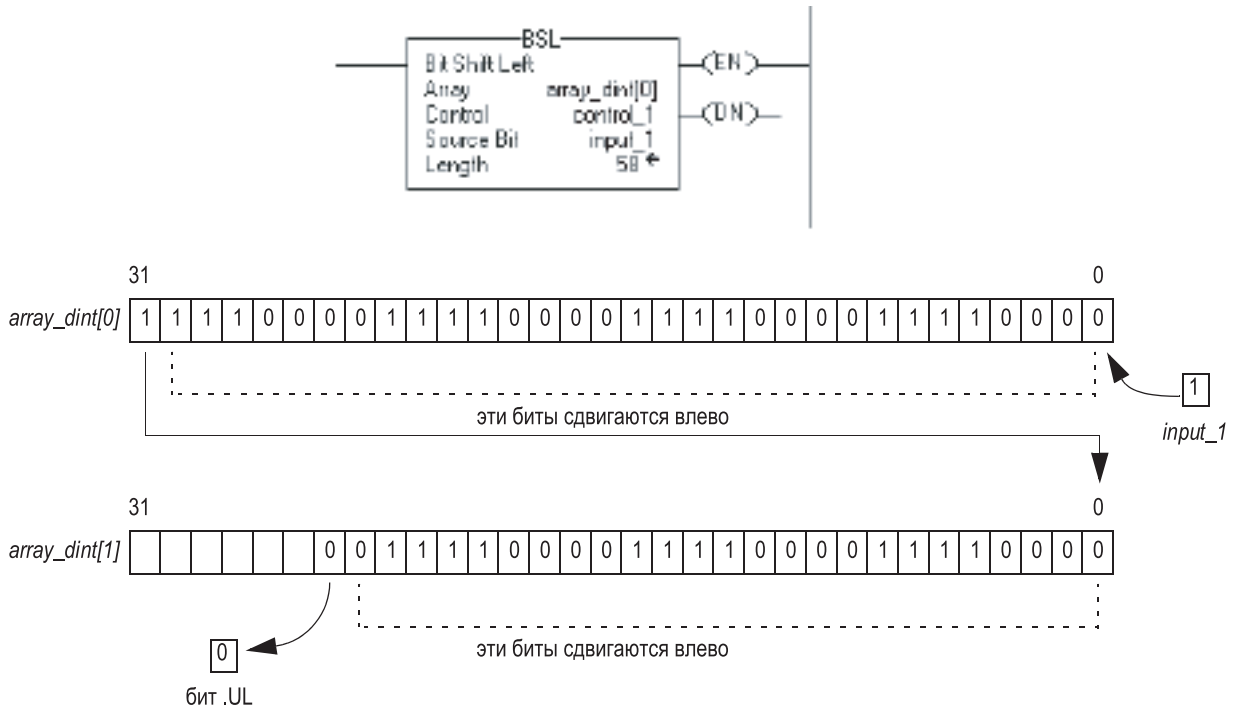


постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример 1: Когда инструкция BSL разрешена, она начинает выполняться с бита 0 в *array_dint[0]*. Инструкция выгружает *array_dint[0].9* в бит .UL, сдвигает оставшиеся биты и загружает *input_1* в *array_dint[0].0*. Значения в оставшихся битах (10-31) не допустимы.



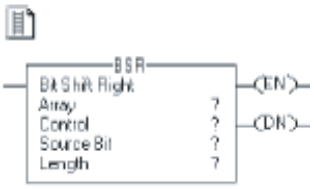
Пример 2: Когда инструкция BSL разрешена, она начинает выполняться с бита 0 в *array_dint[0]*. Инструкция выгружает *array_dint[1].25* в бит .UL, сдвигает оставшиеся биты и загружает *input_1* в *array_dint[0].0*. Значения в оставшихся битах (31-26 в *array_dint[1]*) не допустимы. Обратите внимание на то, как *array_dint[0]31* сдвигается через слова к *array_dint[1].0*.



Bit Shift Right (BSR) (Сдвиг бита вправо)

Инструкция BSL сдвигает заданные биты внутри массива на одну позицию вправо.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Array	DINT	тег массива	массив для модификации задание первого элемента из группы элементов **не** используйте CONTROL.POS в нижнем индексе
Control	CONTROL	тег	управляющая структура для операции
Source bit	BOOL	тег	бит для сдвига
Length	DINT	непосредственный	число битов в массиве для сдвига

Структура CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция BSR разрешена.
.DN	BOOL	Бит выполнения устанавливается, чтобы показать, что биты сдвинуты на одну позицию вправо.
.UL	BOOL	Бит выгрузки является выходом инструкции. Бит .UL хранит статус бита, который был сдвинут из диапазона битов.
.ER	BOOL	Бит ошибки устанавливается, когда .LEN < 0.
.LEN	DINT	Длина задает число битов массива, которые будут сдвинуты.

Описание:

Когда инструкция разрешена, она выгружает значение бита 0 массива в бит .UL, сдвигает оставшиеся биты на одну позицию вправо и загружает бит Source (источника) в самый старший бит из числа заданных битов.

Инструкция BSR производит операции с непрерывной областью памяти данных.

Арифметические флаги состояния:

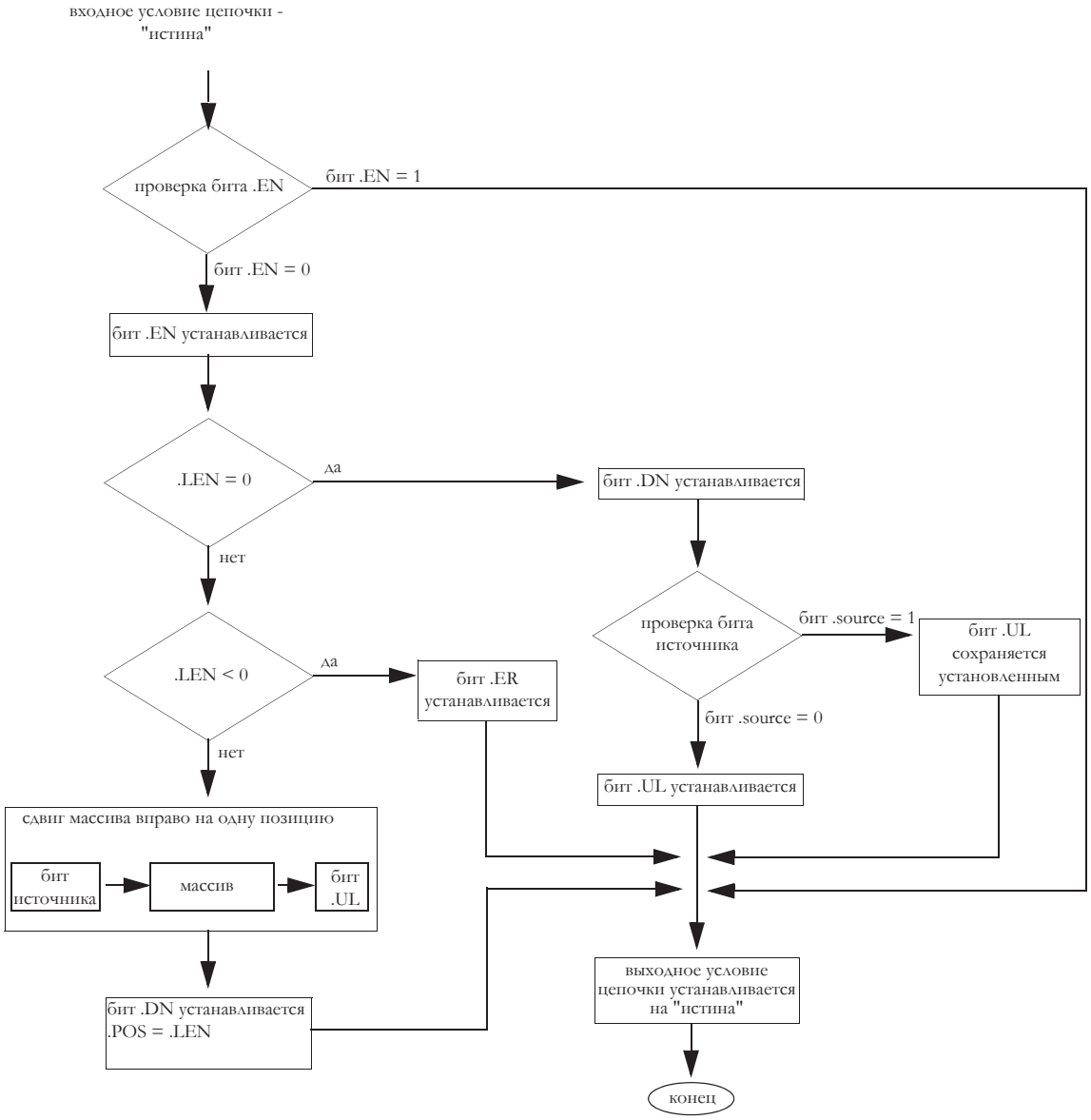
не затрагиваются

Условия ошибки:

отсутствуют

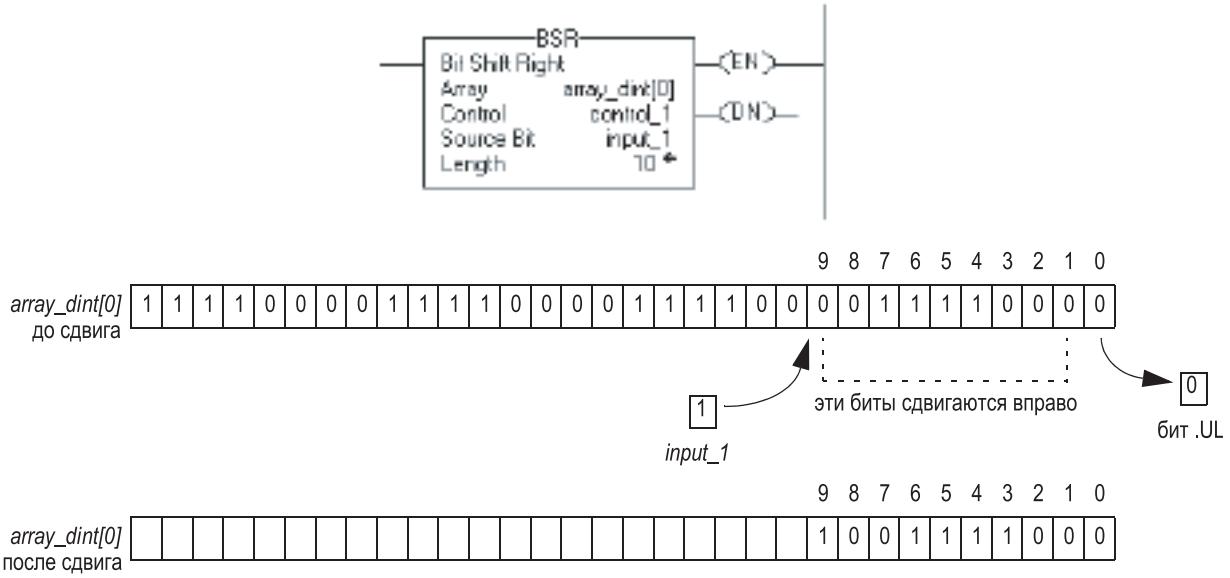
Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Бит .EN сбрасывается. Бит .DN сбрасывается. Бит .ER сбрасывается. Значение .POS сбрасывается. Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Бит .EN сбрасывается. Бит .DN сбрасывается. Бит .ER сбрасывается. Значение .POS сбрасывается. Выходное условие цепочки устанавливается на «ложь».

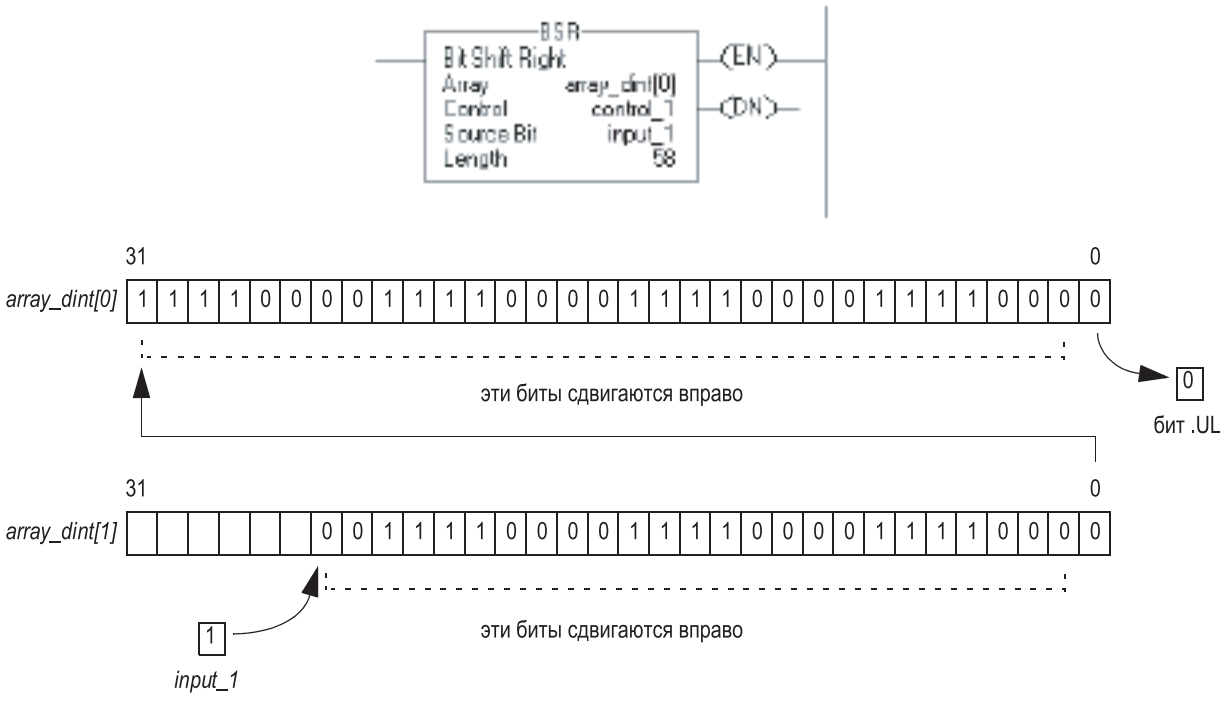


постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример 1: Когда инструкция BSR разрешена, она начинает выполняться с бита 9 в *array_dint[0]*. Инструкция выгружает *array_dint[0].0* в бит .UL, сдвигает оставшиеся биты вправо и загружает *input_1* в *array_dint[0].9*. Значения в оставшихся битах (10-31) не действительны.



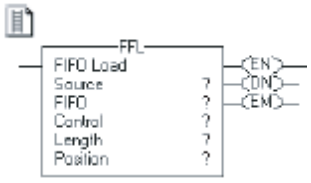
Пример 2: Когда инструкция BSR разрешена, она начинает выполняться с бита 25 в *array_dint[1]*. Инструкция выгружает *array_dint[1].0* в бит .UL, сдвигает оставшиеся биты вправо и загружает *input_1* в *array_dint[0].25*. Значения в оставшихся битах (31-26 в *array_dint[1]*) не действительны. Обратите внимание на то, как *array_dint[1].0* сдвигается через слова к *array_dint[0].31*.



FIFO Load (FFL) (Загрузка в порядке поступления)

Инструкция FFL копирует значение Source (источника) в FIFO.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Source	**SINT** **INT** **DINT** **REAL** **строка** **структура**	непосредственный тег	данные, которые будут храниться в FIFO
	Source преобразует тип данных тега массива. Меньшее целое число преобразуется в большее целое число посредством дополнительного знакового разряда.		
FIFO	**SINT** **INT** **DINT** **REAL** **строка** **структура**	тег массива	FIFO для модификации задание первого элемента FIFO **не** используйте CONTROL.POS в нижнем индексе
Control	CONTROL	тег	управляющая структура для операции обычно используется тот же тип CONTROL, что и для инструкции FFU
Length	DINT	непосредственный	максимальное число элементов, которые FIFO может удерживать за один раз
Position	DINT	непосредственный	следующее положение в FIFO, куда инструкция загружает данные исходное значение обычно 0

Если вы используете определенную пользователем структуру в качестве типа данных для операнда Source (источник) или FIFO (в порядке поступления), применяйте одну и ту же структуру для обоих операндов.

Структура CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция FFL разрешена.
.DN	BOOL	Бит выполнения устанавливается, чтобы показать, что FIFO заполнен (.POS = .LEN). Бит .DN запрещает загрузку FIFO до тех пор, пока не будет выполнено следующее условие .POS < .LEN.
.EM	BOOL	Пустой бит указывает на то, что FIFO пуст. Если .LEN <= 0 или .POS < 0, и бит .EM, и бит .DN устанавливаются.
.LEN	DINT	Длина задает максимальное число элементов, которые FIFO может удерживать за один раз.
.POS	DINT	Позиция показывает положение в FIFO, куда инструкция будет загружать следующее значение.

Описание: Используйте инструкцию FFL вместе с инструкцией FFU для хранения и извлечения данных в порядке поступления. При использовании в паре, инструкции FFL и FFU устанавливают асинхронный сдвиговый регистр.

Обычно операнды Source и FIFO используют один и тот же тип данных.

Когда инструкция FFL разрешена, она загружает значение Source в позицию в FIFO, идентифицированную значением .POS. Инструкция загружает одно значение каждый раз, когда разрешается инструкция, до тех пор, пока FIFO не будет заполнен.

Инструкция FFL производит операции с непрерывной областью памяти.

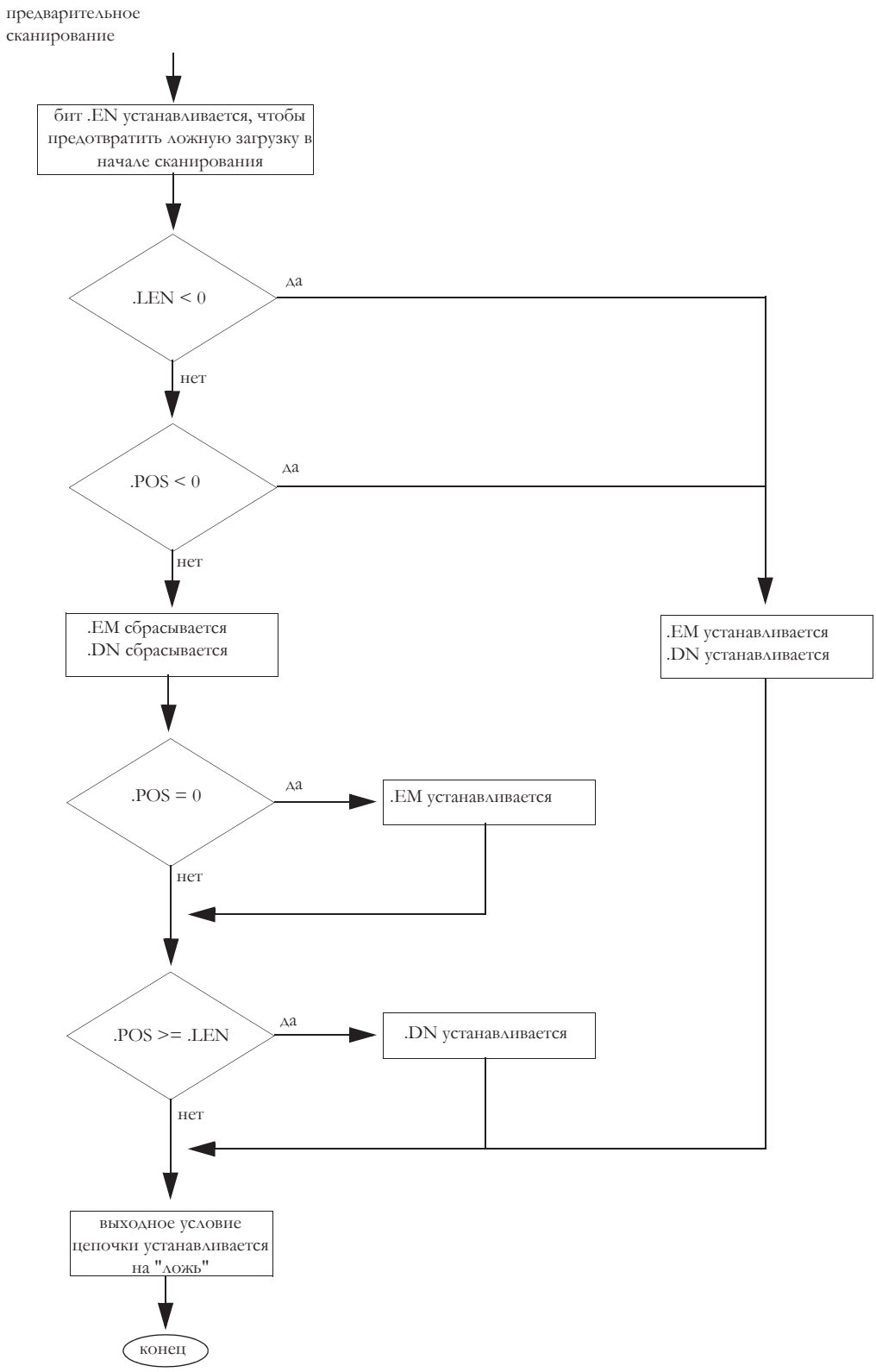
Арифметические флаги состояния: не затрагиваются

Условия ошибки:

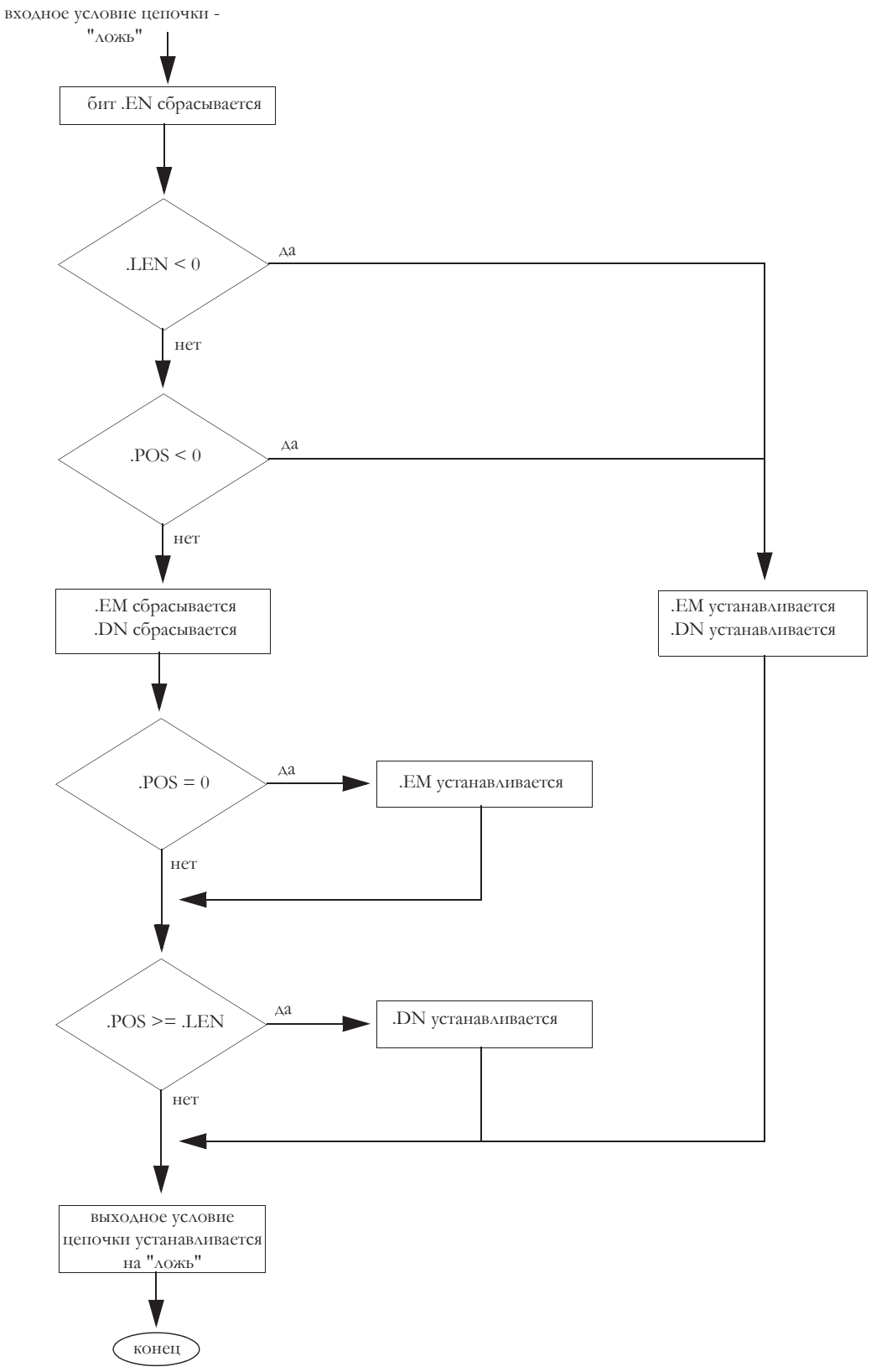
Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
(стартовый элемент + .POS) > размера массива FIFO	4	20

Выполнение:

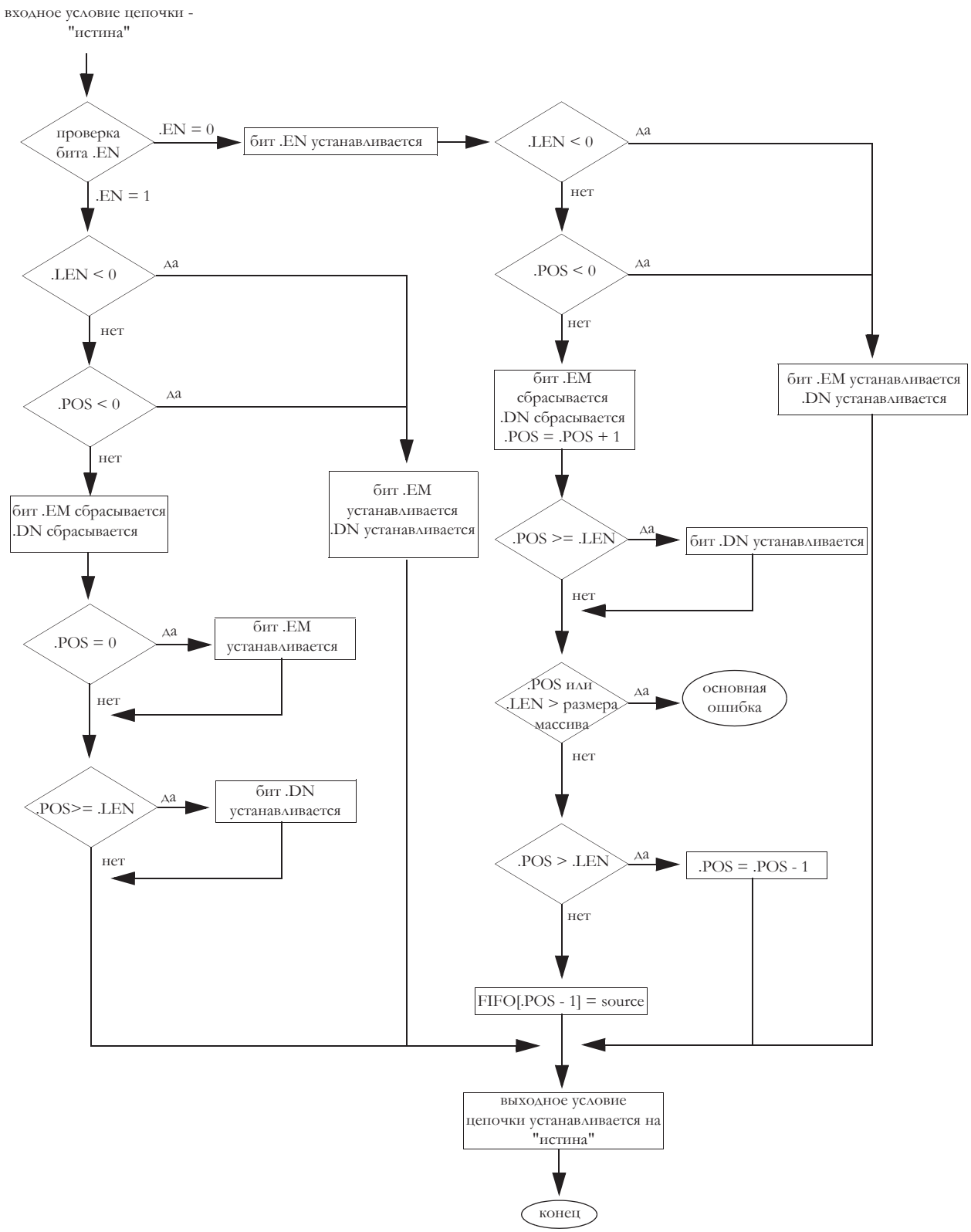
Условие:	Действие релейной логики:



Условие:	**Действие релейной логики:**

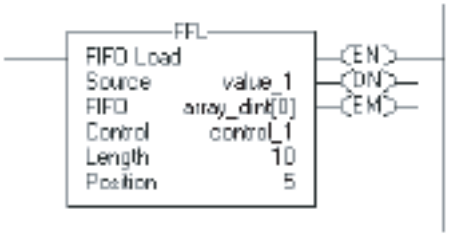


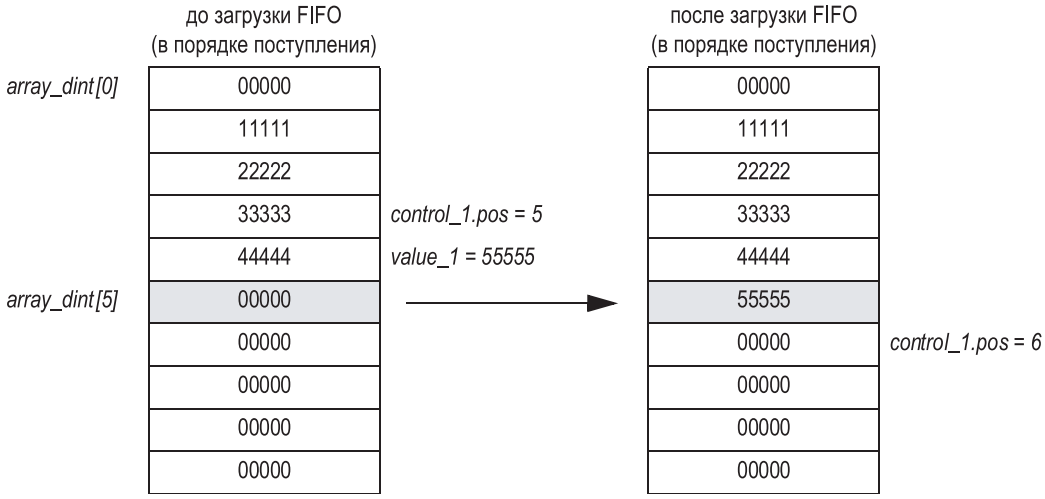
Условие:	**Действие релейной логики:**



постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример: Когда инструкция FFL разрешена, она загружает *value_1* в следующую позицию в FIFO, в данном примере это *array_dint[5].*

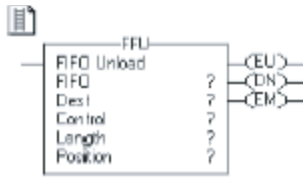




FIFO Unload (FFU) (Выгрузка в порядке поступления)

Инструкция FFU выгружает значение из позиции 0 (первая позиция) FIFO и хранит это значение в Destination (приемнике). Остальные данные в FIFO сдвигаются на одну позицию вниз.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
FIFO	**SINT** **INT** **DINT** **REAL** **строка** **структура**	тег массива	FIFO для модификации задание первого элемента FIFO **не** используйте CONTROL.POS в нижнем индексе
Destination	**SINT** **INT** **DINT** **REAL** **строка** **структура**	тег	значение, которое уходит из FIFO
	Destination преобразует тип данных тега Destination. Меньшее целое число преобразуется в большее целое число посредством дополнительного знакового разряда.		
Control	CONTROL	тег	управляющая структура для операции обычно используется тот же тип CONTROL, что и для инструкции FFL
Length	DINT	непосредственный	максимальное число элементов, которые FIFO может удерживать за один раз
Position	DINT	непосредственный	следующее положение в FIFO, куда инструкция выгружает данные исходное значение обычно 0

Если вы используете определенную пользователем структуру в качестве типа данных для операнда FIFO (в порядке поступления) или Destination (приемник), применяйте одну и ту же структуру для обоих операндов.

Структура CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция FFU разрешена. Бит .EU устанавливается, чтобы заранее установить ложную выгрузку в начале сканирования.
.DN	BOOL	Бит выполнения устанавливается, чтобы показать, что FIFO заполнен (.POS = .LEN).
.EM	BOOL	Пустой бит указывает на то, что FIFO пуст. Если .LEN <= 0 или .POS < 0, и бит .EM, и бит .DN устанавливаются.
.LEN	DINT	Длина задает максимальное число элементов в FIFO.
.POS	DINT	Позиция показывает конец данных, которые были загружены в FIFO.

Описание: Используйте инструкцию FFU вместе с инструкцией FFL для хранения и извлечения данных в порядке поступления.

Когда инструкция FFU разрешена, она выгружает данные из первого элемента FIFO и помещает эти значения в Destination (приемник). Инструкция выгружает одно значение каждый раз, когда разрешается инструкция, до тех пор, пока FIFO не опустеет. Если FIFO пуст, FFU возвращает 0 в Destination.

Инструкция FFU производит операции с непрерывной областью памяти.

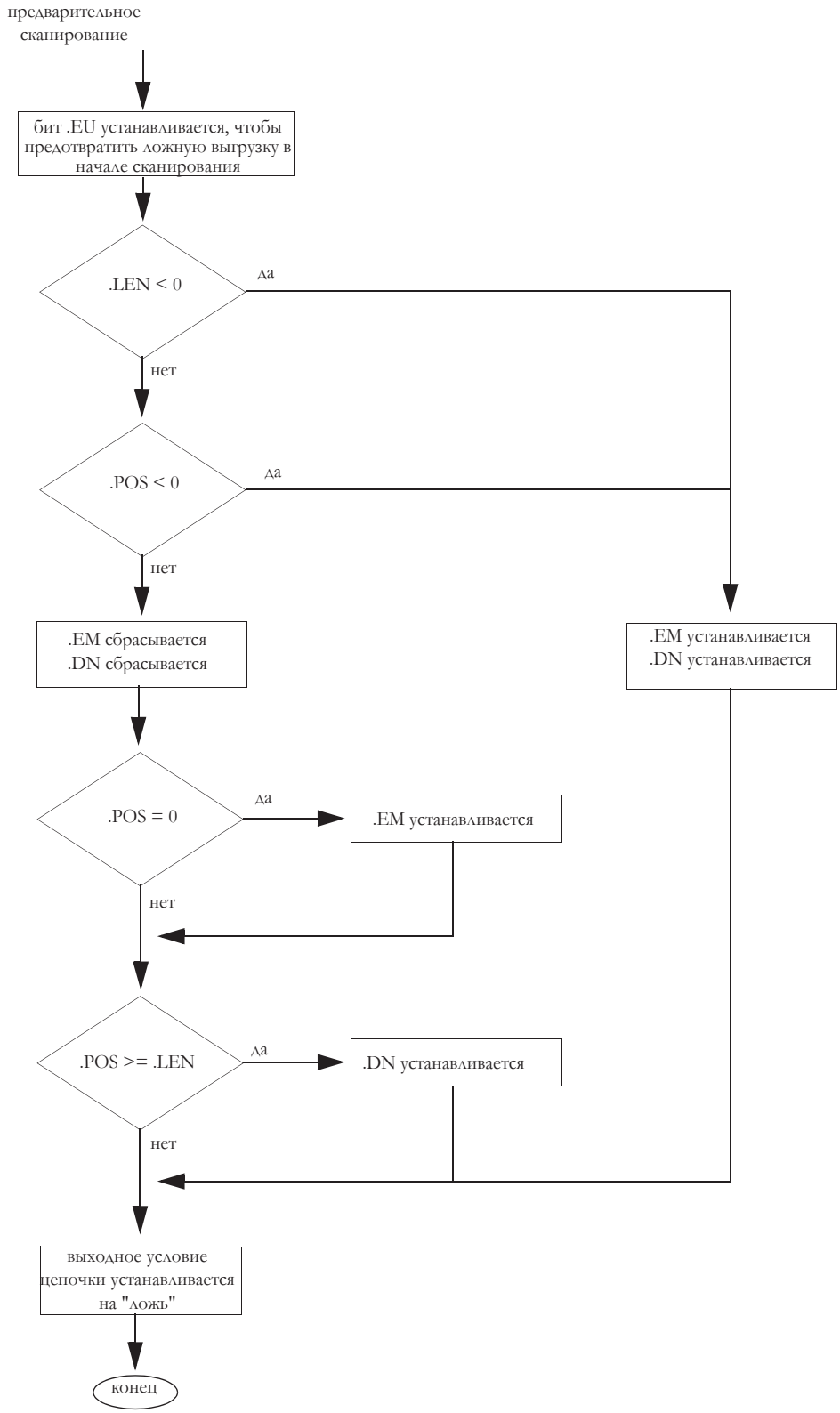
Арифметические флаги состояния: не затрагиваются

Условия ошибки:

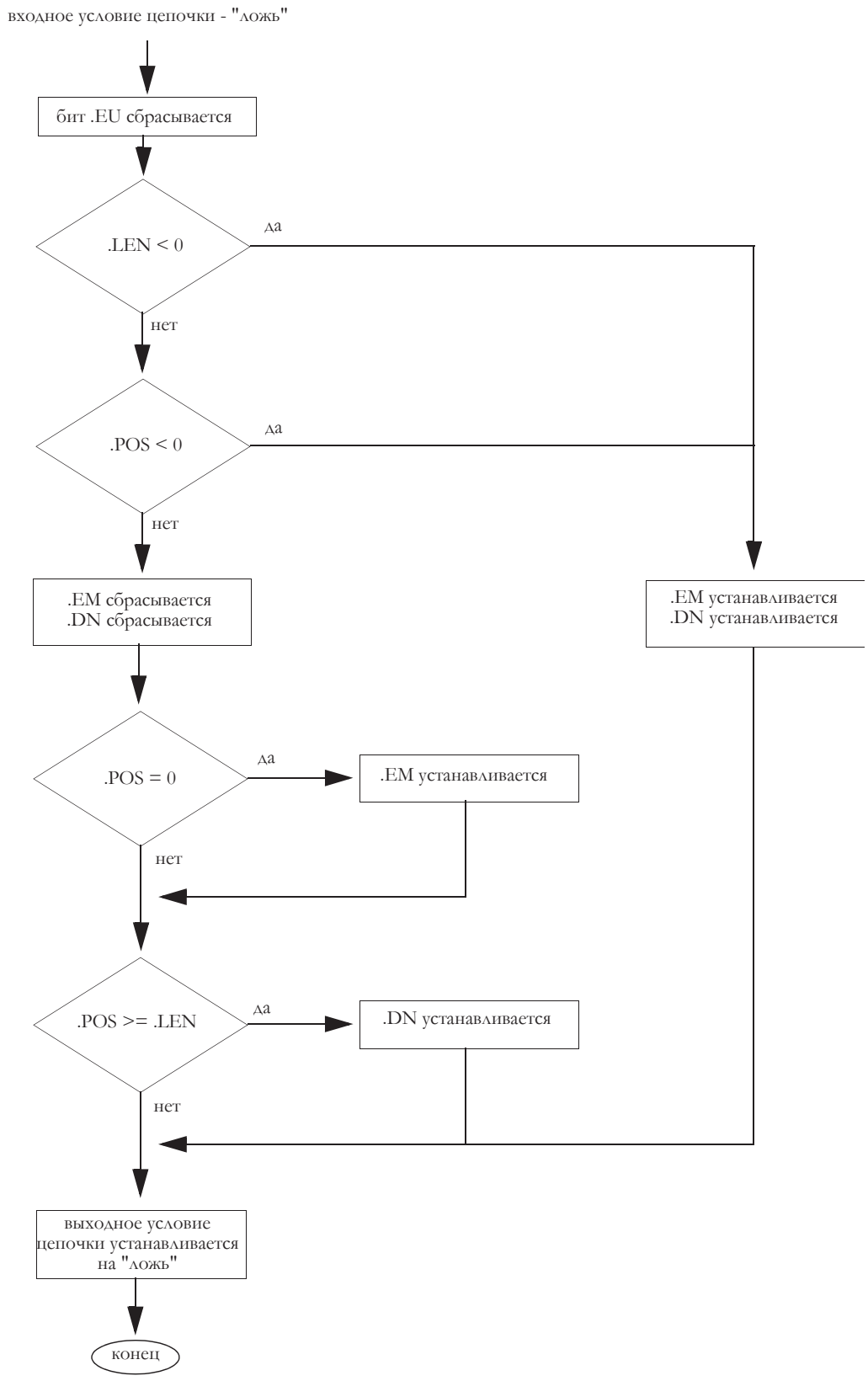
Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
Length (длина) > размера массива FIFO	4	20

Выполнение:

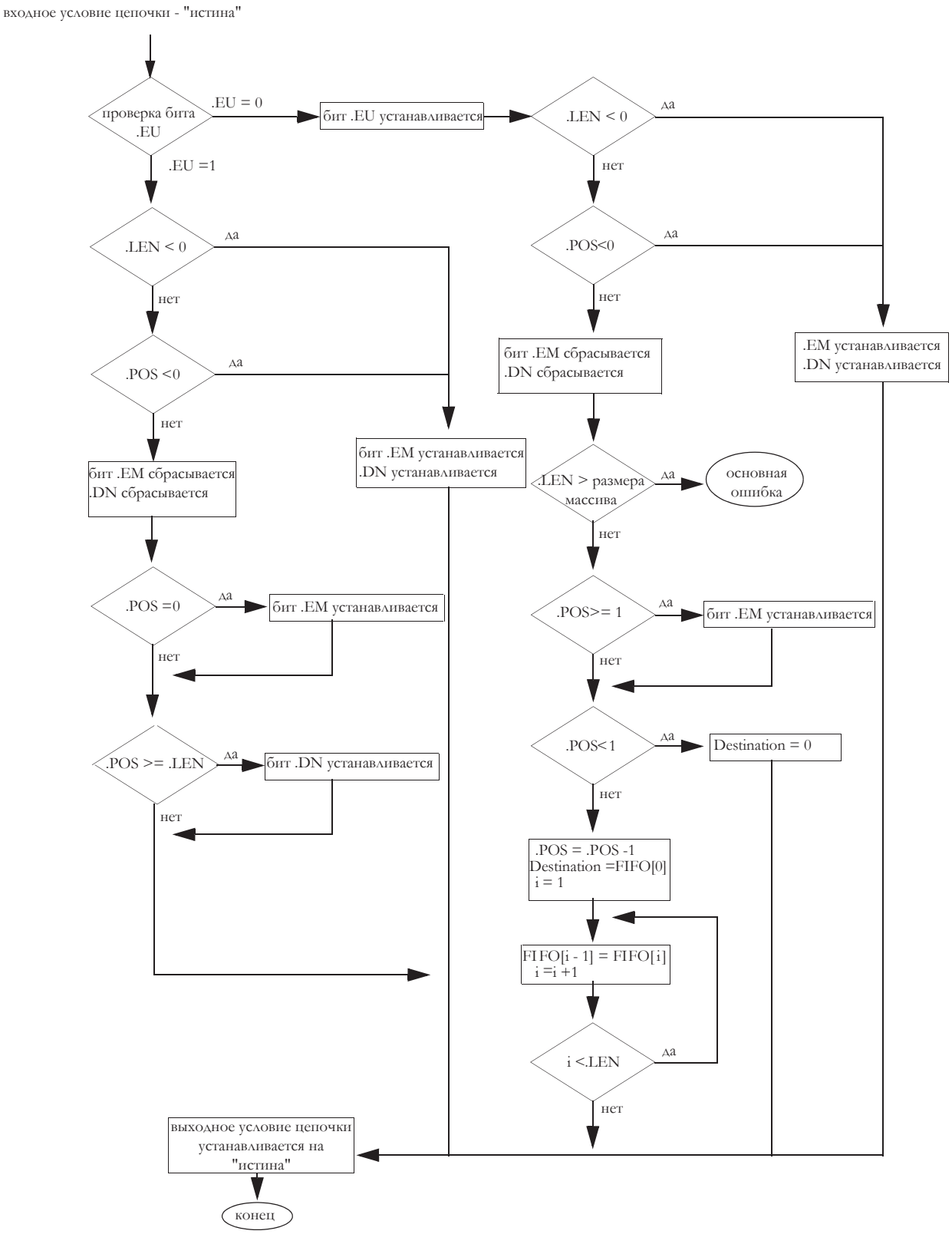
Условие:	Действие релейной логики:



Условие:	**Действие релейной логики:**

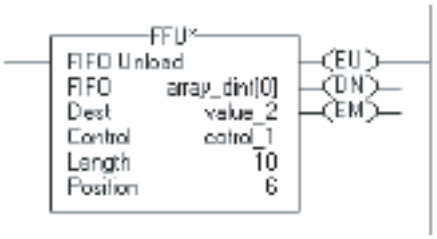


Условие:	**Действие релейной логики:**



постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример: Когда инструкция FFU разрешена, она выгружает *array_dint[0]* в *value_2* и сдвигает остальные элементы в *array_dint*.

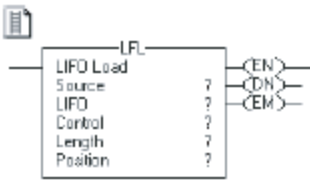


	до выгрузки FIFO (в порядке поступления)		после выгрузки FIFO (в порядке поступления)	
array_dint[0]	00000	→	11111	
	11111		22222	
	22222		33333	
	33333		44444	
	44444		55555	
array_dint[5]	55555		00000	*control_1.pos = 5*
	00000	*control_1.pos = 6*	00000	*value_2 = 00000*
	00000		00000	
	00000		00000	
	00000		00000	

LIFO LOAD (LFL) (Загрузка LIFO)

Инструкция LFL копирует значение Source (источника) в LIFO.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Source	**SINT** **INT** **DINT** **REAL** **строка** **структура**	непосредственный тег	данные, которые будут храниться в LIFO
	Source преобразует тип данных тега массива. Меньшее целое число преобразуется в большее целое число посредством дополнительного знакового разряда.		
FIFO	**SINT** **INT** **DINT** **REAL** **строка** **структура**	тег массива	LIFO для модификации задание первого элемента LIFO **не** используйте CONTROL.POS в нижнем индексе
Control	CONTROL	тег	управляющая структура для операции обычно используется тот же тип CONTROL, что и для инструкции LFU
Length	DINT	непосредственный	максимальное число элементов, которые LIFO может удерживать за один раз
Position	DINT	непосредственный	следующее положение в LIFO, куда инструкция загружает данные исходное значение обычно 0

Если вы используете определенную пользователем структуру в качестве типа данных для операнда Source (источник) или LIFO (в магазинном порядке), применяйте одну и ту же структуру для обоих операндов.

Структура CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция LFL разрешена.
.DN	BOOL	Бит выполнения устанавливается, чтобы показать, что LIFO заполнен (.POS = .LEN). Бит .DN запрещает загрузку LIFO до тех пор, пока не будет выполнено следующее условие .POS < .LEN.
.EM	BOOL	Пустой бит указывает на то, что LIFO пуст. Если .LEN <= 0 или .POS < 0, и бит .EM, и бит .DN устанавливаются.
.LEN	DINT	Длина задает максимальное число элементов, которые LIFO может удерживать за один раз.
.POS	DINT	Позиция показывает положение в LIFO, куда инструкция будет загружать следующее значение.

Описание:

Используйте инструкцию LFL вместе с инструкцией LFU для хранения и извлечения данных в магазинном порядке. При использовании в паре, инструкции LFL и LFU устанавливают асинхронный сдвиговый регистр.

Обычно операнды Source и LIFO используют один и тот же тип данных.

Когда инструкция LFL разрешена, она загружает значение Source в позицию в LIFO, идентифицированную значением .POS. Инструкция загружает одно значение каждый раз, когда разрешается инструкция, до тех пор, пока LIFO не будет заполнен.

Инструкция LFL производит операции с непрерывной областью памяти.

Арифметические флаги состояния:

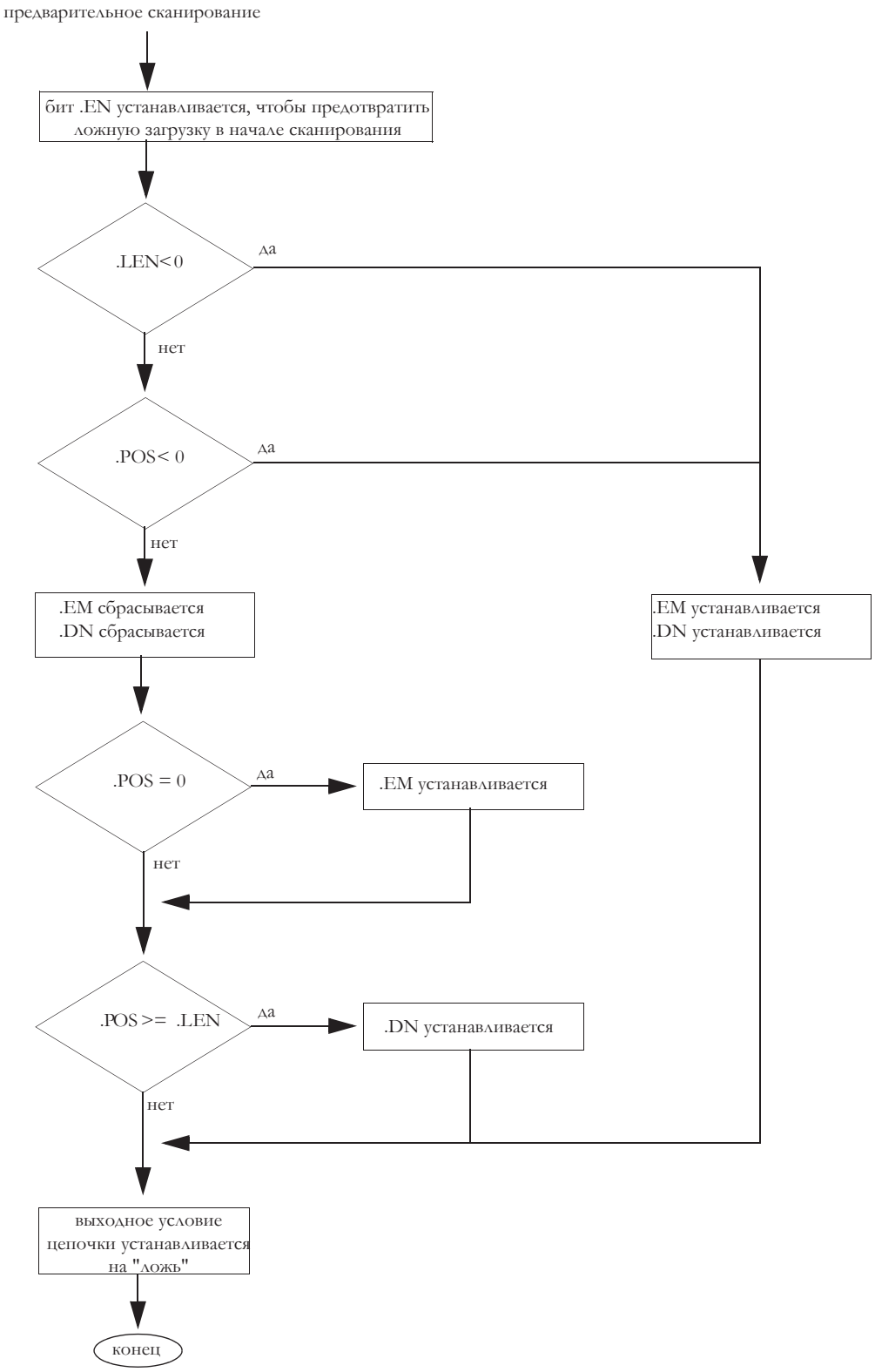
не затрагиваются

Условия ошибки:

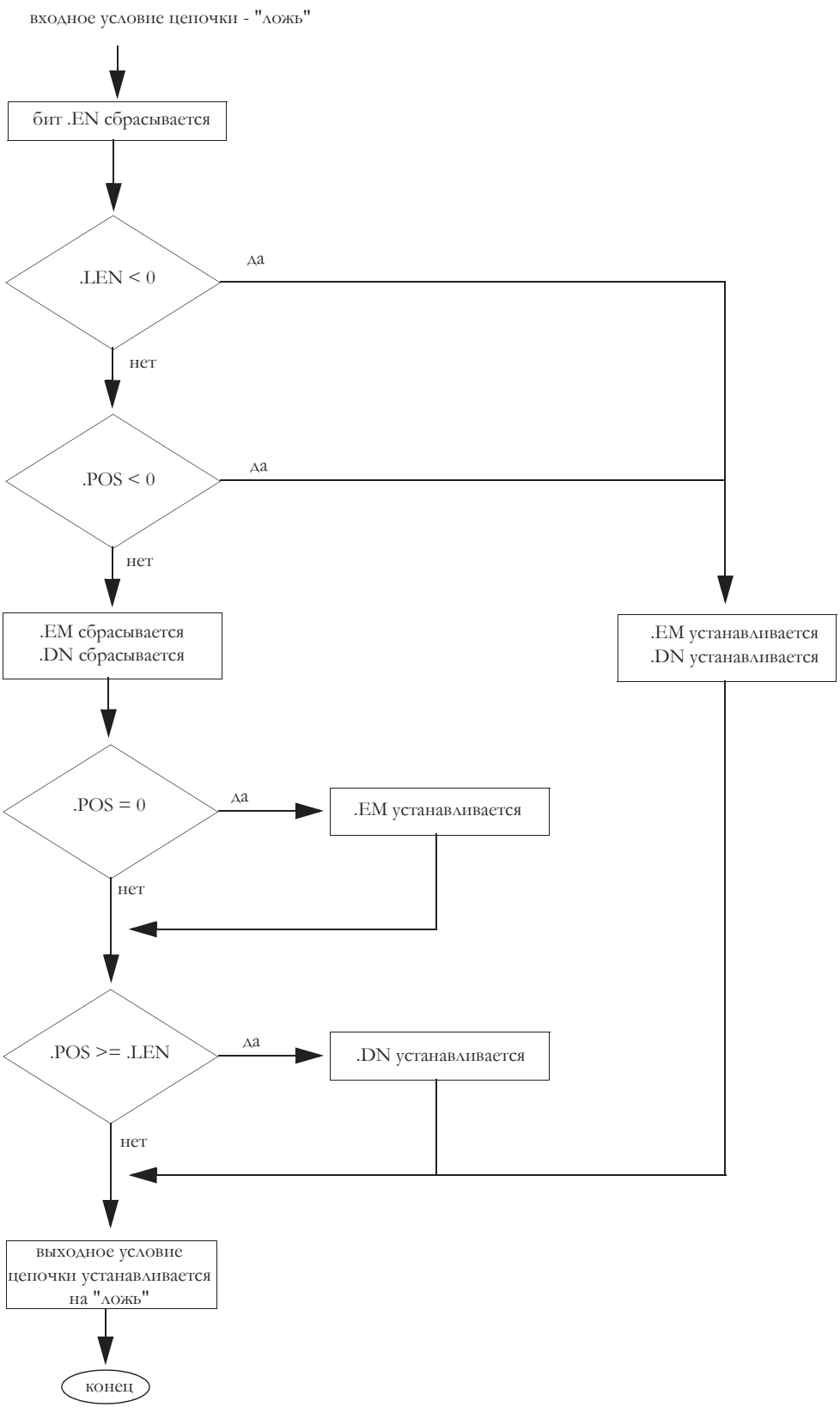
Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
(стартовый элемент + .POS) > размера массива LIFO	4	20

Выполнение:

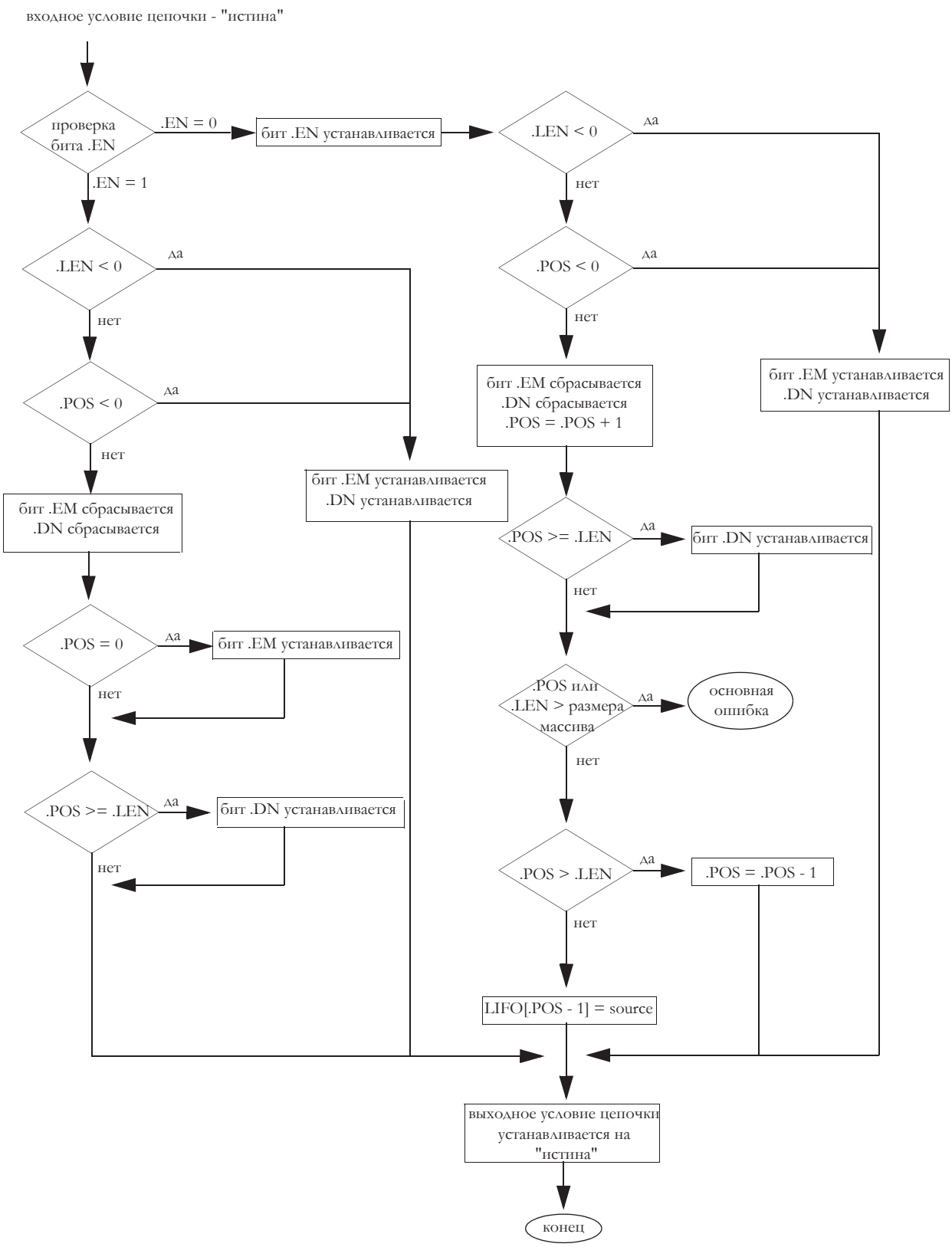
Условие:	Действие релейной логики:



Условие:	**Действие релейной логики:**

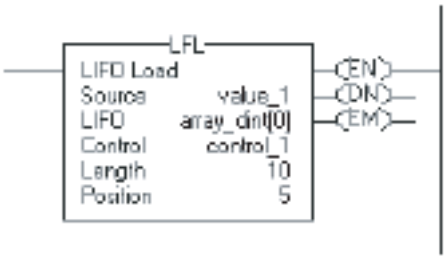


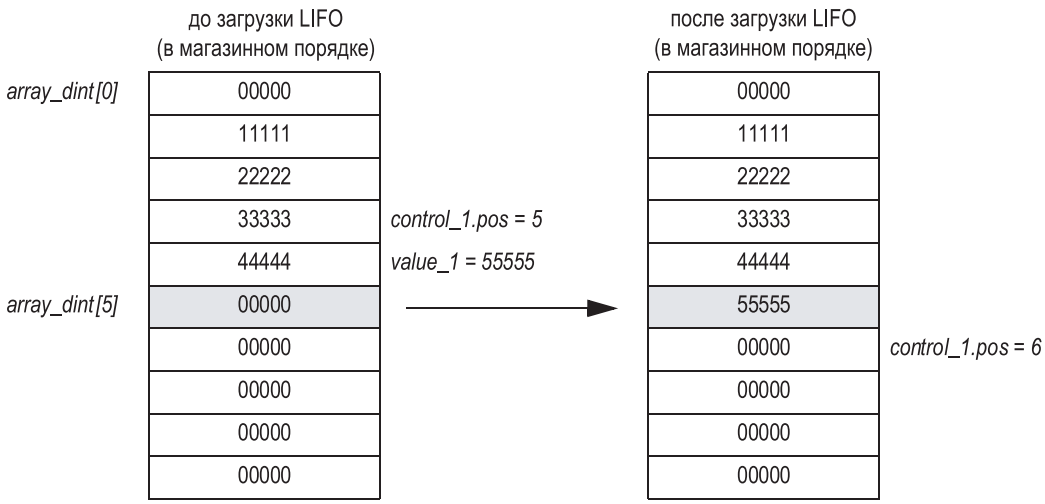
Условие:	Действие релейной логики:



постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример: Когда инструкция LFL разрешена, она загружает *value_1* в следующую позицию в LIFO, в данном примере это *array_dint[5].*

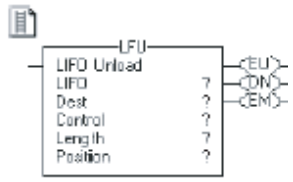




LIFO Unload (LFU) (Выгрузка LIFO)

Инструкция LFU выгружает значение из позиции 0 (первая позиция) LIFO и хранит это значение в Destination (приемнике). Остальные данные в LIFO сдвигаются на одну позицию вниз.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
LIFO	**SINT** **INT** **DINT** **REAL** **строка** **структура**	тег массива	LIFO для модификации задание первого элемента LIFO **не** используйте CONTROL.POS в нижнем индексе
Destination	**SINT** **INT** **DINT** **REAL** **строка** **структура**	тег	значение, которое уходит из LIFO
	Destination преобразует тип данных тега Destination. Меньшее целое число преобразуется в большее целое число посредством дополнительного знакового разряда.		
Control	CONTROL	тег	управляющая структура для операции обычно используется тот же тип CONTROL, что и для инструкции LFL
Length	DINT	непосредственный	максимальное число элементов, которые LIFO может удерживать за один раз
Position	DINT	непосредственный	следующее положение в LIFO, куда инструкция выгружает данные исходное значение обычно 0

Если вы используете определенную пользователем структуру в качестве типа данных для операнда LIFO (в магазинном порядке) или Destination (приемник), применяйте одну и ту же структуру для обоих операндов.

Структура CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция LFU разрешена. Бит .EU устанавливается, чтобы заранее установить ложную выгрузку в начале сканирования.
.DN	BOOL	Бит выполнения устанавливается, чтобы показать, что LIFO заполнен (.POS = .LEN).
.EM	BOOL	Пустой бит указывает на то, что LIFO пуст. Если .LEN <= 0 или .POS < 0, и бит .EM, и бит .DN устанавливаются.
.LEN	DINT	Длина задает максимальное число элементов в LIFO.
.POS	DINT	Позиция показывает конец данных, которые были загружены в LIFO.

Описание: Используйте инструкцию LFU вместе с инструкцией LFL для хранения и извлечения данных в магазинном порядке.

Когда инструкция LFU разрешена, она выгружает значение элемента .POS LIFO и помещает это значение в Destination (приемник). Инструкция выгружает одно значение и заменяет его 0 каждый раз, когда разрешается инструкция, до тех пор, пока LIFO не опустеет. Если LIFO пуст, LFU возвращает 0 в Destination.

Инструкция LFU производит операции с непрерывной областью памяти.

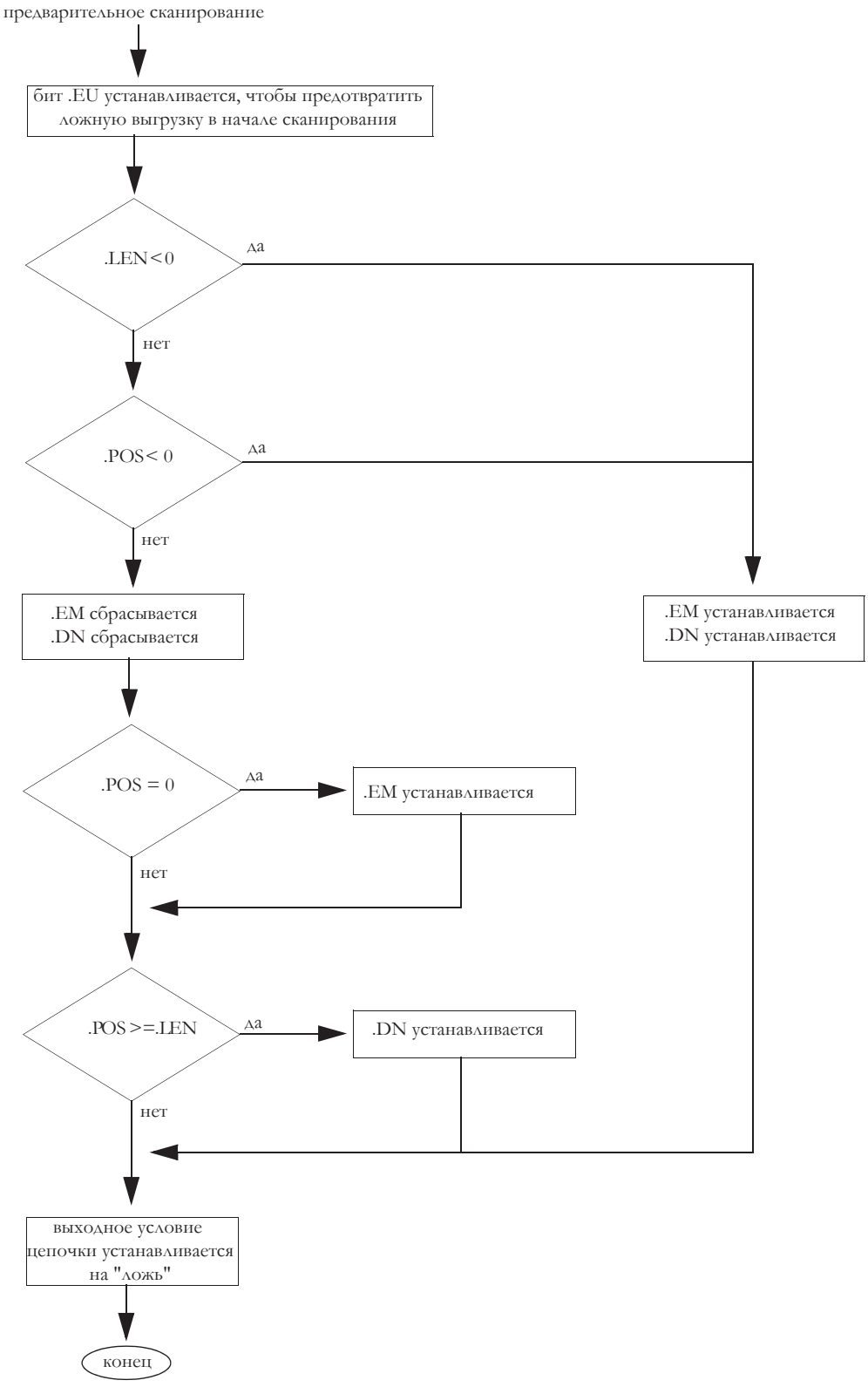
Арифметические флаги состояния: не затрагиваются

Условия ошибки:

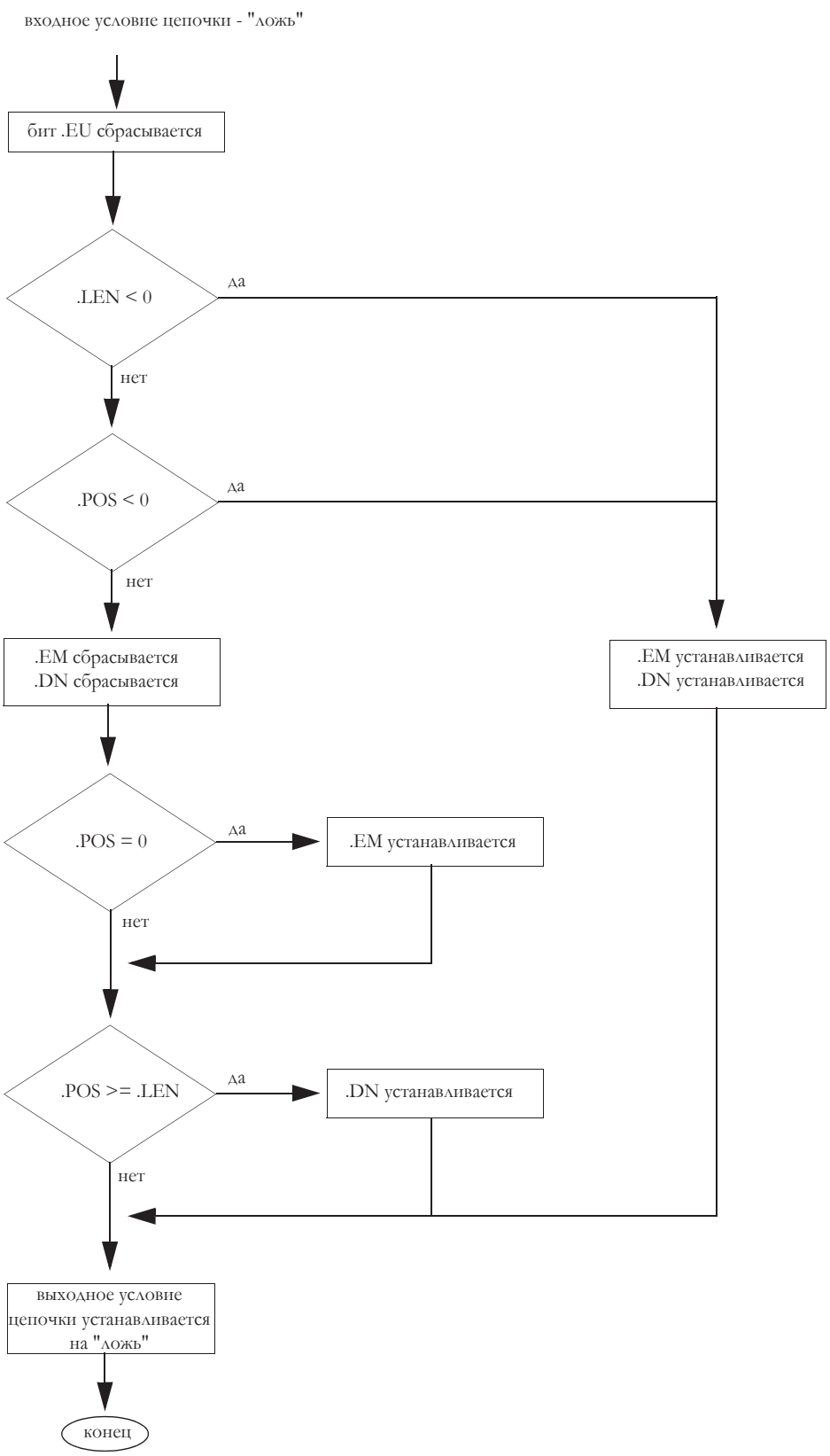
Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
Length (длина) > размера массива LIFO	4	20

Выполнение:

Условие:	Действие релейной логики:

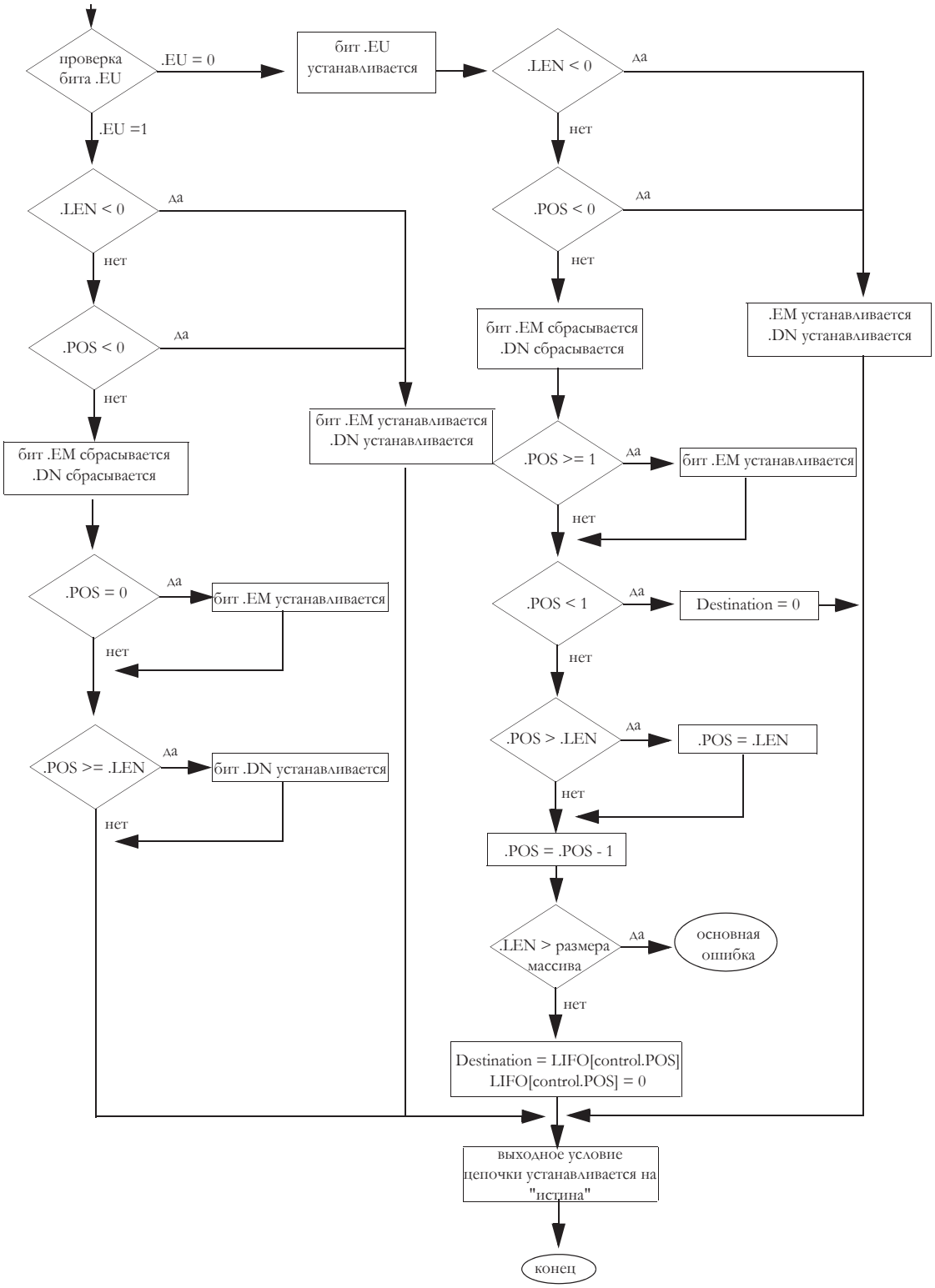


Условие:	**Действие релейной логики:**



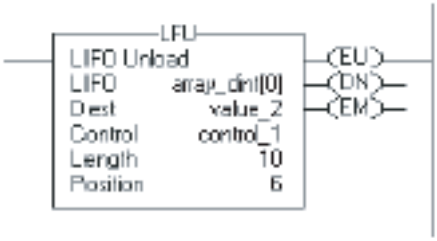
Условие:	**Действие релейной логики:**

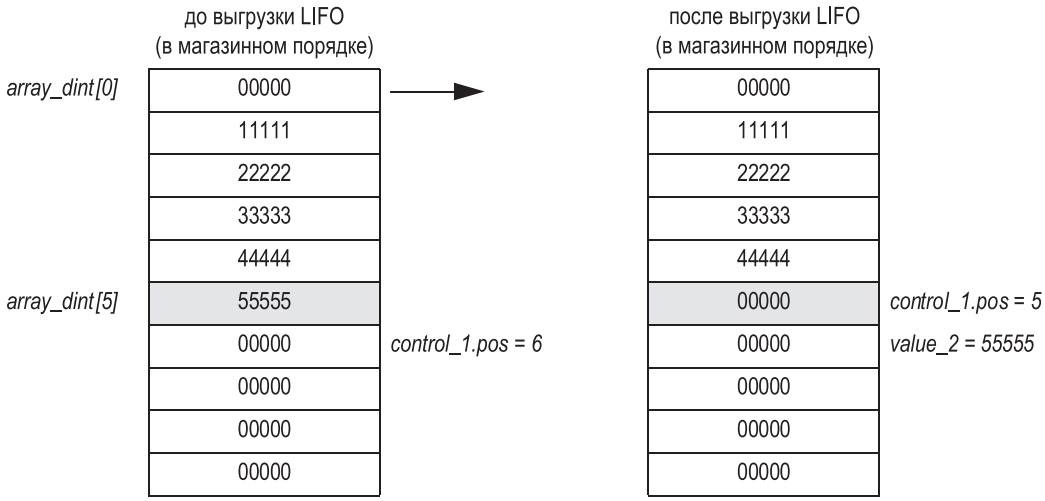
входное условие цепочки - "истина"



постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример: Когда инструкция LFU разрешена, она выгружает *array_dint[5]* в *value_2*.





Примечания:

Инструкции секвенсеров (SQI, SQO, SQL)

Введение

Никакого действия не производится. Инструкции секвенсеров контролируют выполнение последовательных и повторяющихся операций.

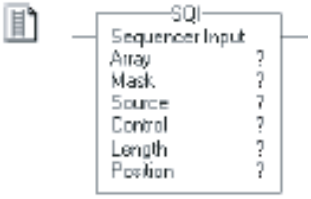
Если вы хотите:	Используйте эту инструкцию:	Имеющуюся в этих языках:	См. стр.
выявить, когда действие завершено.	SQI	релейная логика	9-2
задать выходные условия для следующего действия.	SQO	релейная логика	9-6
загрузить исходные условия в массивы секвенсера	SQL	релейная логика	9-10

Для инструкций релейной логики, **жирный** шрифт типов данных означает, что это оптимальные типы данных. Инструкция выполняется быстрее и требует меньший объем памяти, если все операнды инструкции используют один и тот же оптимальный тип данных, обычно DINT или REAL.

Sequencer Input (SQI) (Секвенсер входа)

Инструкция SQI обнаруживает, когда действие завершено, инструкции SQO/SQI выполняются последовательно в паре.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Array	DINT	тег массива	массив секвенсера задание первого элемента из массива секвенсера **не** используйте CONTROL.POS в нижнем индексе
Mask	SINT INT **DINT**	тег непосредственный	показывает, какие биты блокировать, а какие пропускать
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Source	SINT INT DINT	тег	исходные данные для массива секвенсера
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Control	CONTROL	тег	управляющая структура для операции обычно используется тот же операнд CONTROL, что и для инструкций SQO и SQL
Length	DINT	непосредственный	число элементов в массиве (таблица секвенсера) для сравнения
Position	DINT	непосредственный	текущая позиция в массиве исходное значение обычно 0

Структура CONTROL

Мнемоника:	Тип данных:	Описание
.ER	BOOL	Бит ошибки устанавливается, когда .LEN ? 0, .POS < 0 или .POS > .LEN.
.LEN	DINT	Длина задает число действий в массиве секвенсера.
.POS	DINT	Позиция показывает элемент, который инструкция сравнивает в данный момент.

Описание:

Когда инструкция SQI разрешена, она сравнивает элемент Source (источника) посредством Mask (маски) с элементом Array (массива) на предмет идентичности.

Обычно для этой инструкции используется та же самая структура CONTROL, что и для инструкций SQO и SQL.

Инструкция SQI производит операции с непрерывной областью памяти.

Ввод непосредственного значения маски

Когда вы вводите значение маски, программное обеспечение установлено по умолчанию на восприятие десятичного значения. Если вы хотите ввести маску, используя другой формат, снабдите значение корректным префиксом.

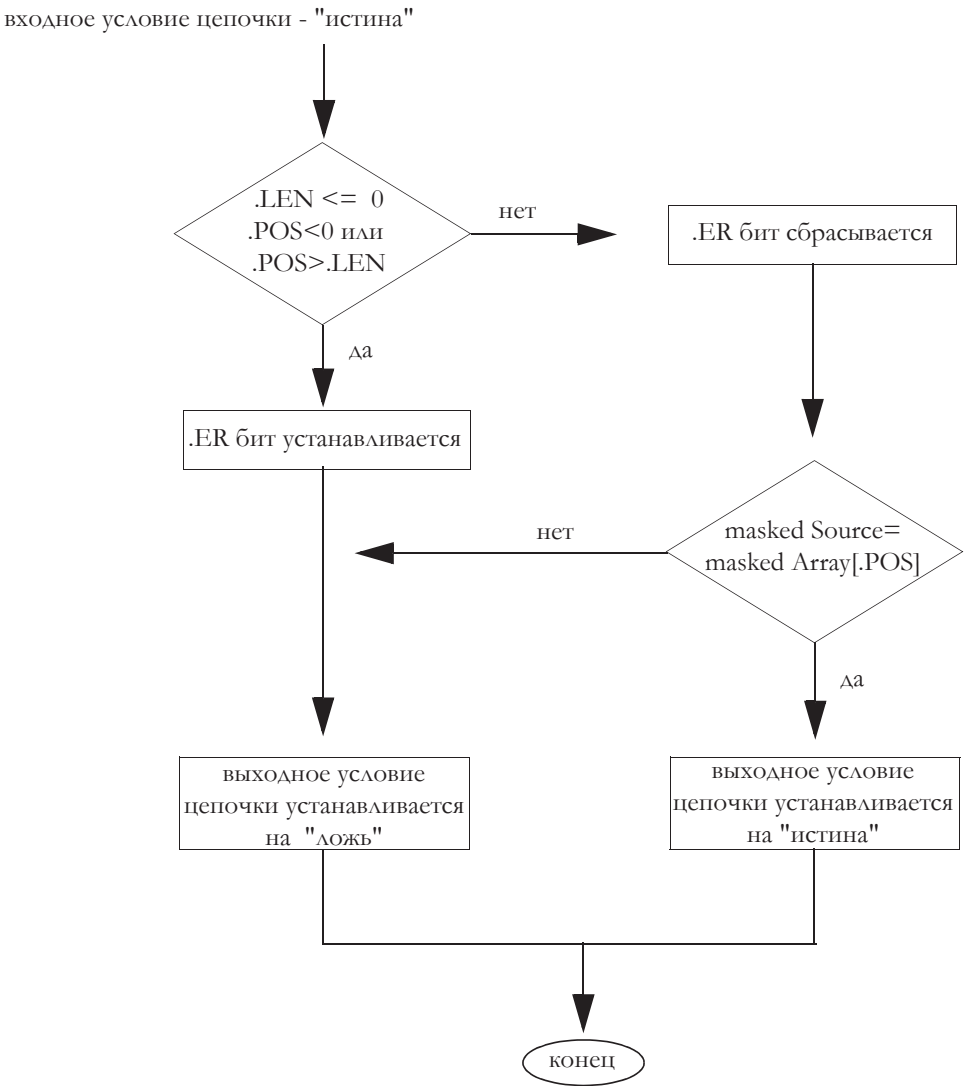
Префикс:	Описание:
16#	шестнадцатеричный например, 16#OFOF
8#	восьмеричный например, 8#16
2#	двоичный например, 2#00110011

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

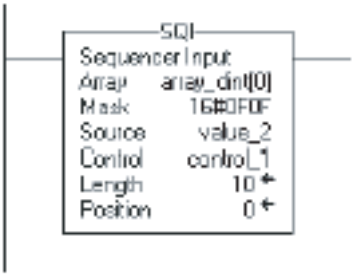
Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».



постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример: Когда инструкция SQI разрешена, она пропускает *value_2* через маску, чтобы определить, является ли результат идентичным текущему элементу в *array_dint*. Маскированное сравнение – «истина», поэтому выходное условие цепочки устанавливается на «истина».



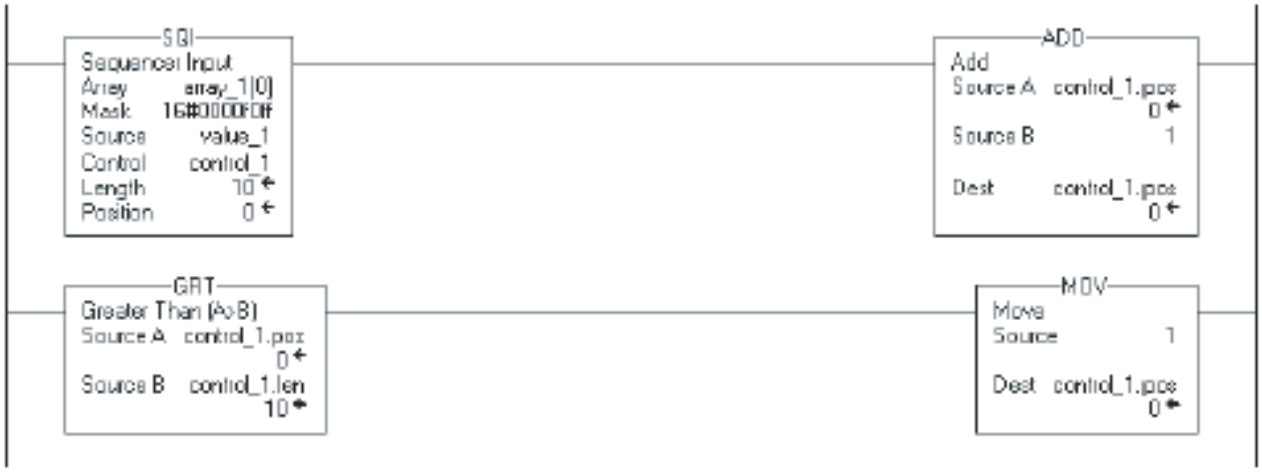
Операнд SQI:	Примеры значений (значения DINT, показанные в двоичном коде):			
Source	xxxxxxxx	xxxxxxxx	xxxx0101	xxxx1010
Mask	00000000	00000000	00001111	00001111
Array	xxxxxxxx	xxxxxxxx	xxxx0101	xxxx1010

0 в значении маски означает, что бит не сравнивается (обозначен xxxx в этом примере).

Использование SQI без SQO

Если вы используете инструкцию SQI без парной ей инструкции SQO, вам необходимо произвести внешнее приращение массива секвенсера.

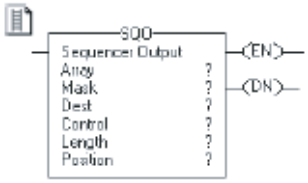
Инструкция SQI сравнивает значение источника. Инструкция ADD производит приращение массива секвенсера. Инструкция GRT определяет, доступно ли другое значение для регистрации в массиве секвенсера. Инструкция MOV переустанавливает значение позиции после полного прохождения один раз через массив секвенсера.



Sequencer Output (SQO) (Секвенсер выхода)

Инструкция SQO устанавливает выходные условия для следующего действия инструкций SQO/SQI, выполняющихся последовательно в паре.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Array	DINT	тег массива	массив секвенсера задание первого элемента из массива секвенсера **не** используйте CONTROL.POS в нижнем индексе
Mask	SINT INT **DINT**	тег непосредственный	показывает, какие биты блокировать, а какие пропускать
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Destination	DINT	тег	выходные данные из массива секвенсера
Control	CONTROL	тег	управляющая структура для операции обычно используется тот же операнд CONTROL, что и для инструкций SQI и SQL
Length	DINT	непосредственный	число элементов в массиве (таблица секвенсера) для выхода
Position	DINT	непосредственный	текущая позиция в массиве исходное значение обычно 0

Структура CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция SQO разрешена.
.DN	BOOL	Бит выполнения устанавливается, когда все заданные элементы перемещены в Destination (приемник).
.ER	BOOL	Бит ошибки устанавливается, когда .LEN <= 0, .POS < 0 или .POS > .LEN.
.LEN	DINT	Длина задает число действий в массиве секвенсера.
.POS	DINT	Позиция показывает элемент, с которым в данный момент контроллер производит манипуляции.

Описание:

Когда инструкция SQO разрешена, она производит приращение позиции, перемещает данные в эту позиции посредством Mask (маски) и хранит результат в Destination (приемнике). Если .POS > .LEN, инструкция автоматически переходит в начало массива секвенсера и продолжает выполняться со значения .POS = 1.

Обычно для этой инструкции используется та же самая структура CONTROL, что и для инструкций SQI и SQL.

Инструкция SQO производит операции с непрерывной областью памяти.

Ввод непосредственного значения маски

Когда вы вводите значение маски, программное обеспечение установлено по умолчанию на восприятие десятичного значения. Если вы хотите ввести маску, используя другой формат, снабдите значение корректным префиксом.

Префикс:	Описание:
16#	шестнадцатеричный например, 16#OFOF
8#	восьмеричный например, 8#16
2#	двоичный например, 2#00110011

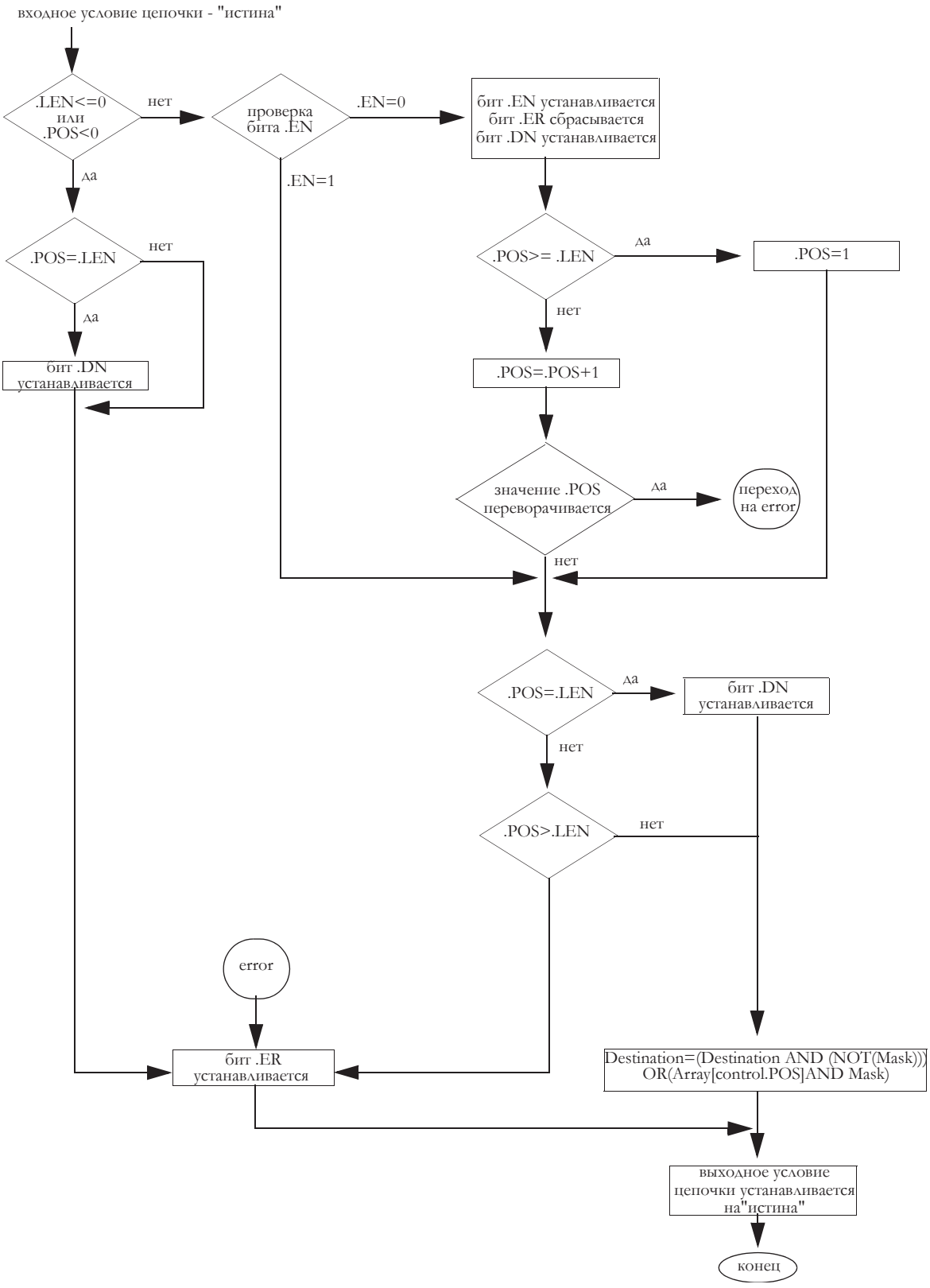
Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

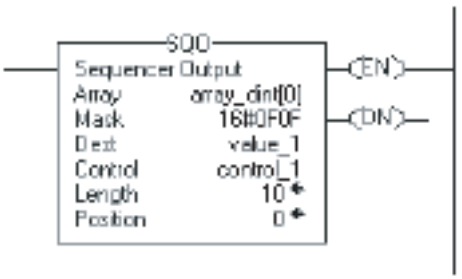
Условие:	Действие релейной логики:
предварительное сканирование	Бит .EN устанавливается, чтобы предотвратить ложную загрузку в начале сканирования программы. Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Бит .EN сбрасывается. Выходное условие цепочки устанавливается на «ложь».

Условие:	Действие релейной логики:



постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример: Когда инструкция SQO разрешена, она производит приращение позиции, перемещает данные в эту позицию в *array_dint* посредством маски и хранит результат в *value_1*.

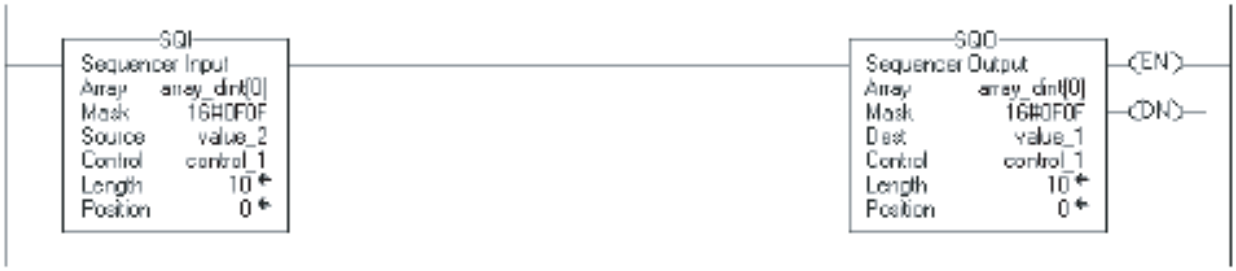


Операнд SQI:	Примеры значений (значения DINT, показанные в двоичном коде):			
Array	xxxxxxxx	xxxxxxxx	xxxx0101	xxxx1010
Mask	00000000	00000000	00001111	00001111
Destination	xxxxxxxx	xxxxxxxx	xxxx0101	xxxx1010

0 в значении маски означает, что бит не сравнивается (обозначен xxxx в этом примере).

Использование SQI вместе с инструкцией SQO

Если вы используете инструкцию SQI вместе с парной ей инструкцией SQO, убедитесь, что обе инструкции используют одинаковые значения Control (контроля), Length (длины) и Position (позиции).



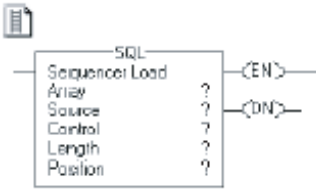
Переустановка позиции SQO

Каждый раз, когда контроллер переходит из режима программ (Program) в режим выполнения (Run), инструкция SQO сбрасывает (устанавливает в исходное состояние) значение .POS. Чтобы переустановить .POS, задав исходное значение (.POS = 0), используйте инструкцию RES для сброса значения позиции. В этом примере применяется статус бита первого сканирования для сброса значения .POS.

Sequencer Load (SQL) (Загрузка секвенсера)

Инструкция SQL загружает исходные условия в массив секвенсера.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Array	DINT	тег массива	массив секвенсера задание первого элемента из массива секвенсера **не** используйте CONTROL.POS в нижнем индексе
Source	SINT INT **DINT**	тег непосредственный	исходные данные для загрузки в массив секвенсера
	Тег SINT или INT преобразуется в значение DINT посредством дополнительного знакового разряда.		
Control	CONTROL	тег	управляющая структура для операции обычно используется тот же операнд CONTROL, что и для инструкций SQI и SQO
Length	DINT	непосредственный	число элементов в массиве (таблица секвенсера) для загрузки
Position	DINT	непосредственный	текущая позиция в массиве исходное значение обычно 0

Структура CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция SQL разрешена.
.DN	BOOL	Бит выполнения устанавливается, когда все заданные элементы загружены в Array (массив).
.ER	BOOL	Бит ошибки устанавливается, когда .LEN <= 0, .POS < 0 или .POS > .LEN.
.LEN	DINT	Длина задает число действий в массиве секвенсера.
.POS	DINT	Позиция показывает элемент, с которым в данный момент контроллер производит манипуляции.

Описание:

Когда инструкция SQL разрешена, она производит приращение следующей позиции в массиве секвенсера и загружает значение Source (источника) в эту позицию. Если устанавливается бит .DN или .POS >= .LEN, инструкция устанавливает значение .POS = 1.

Обычно для этой инструкции используется та же самая структура CONTROL, что и для инструкций SQI и SQO.

Инструкция SQL производит операции с непрерывной областью памяти.

Арифметические флаги состояния: не затрагиваются

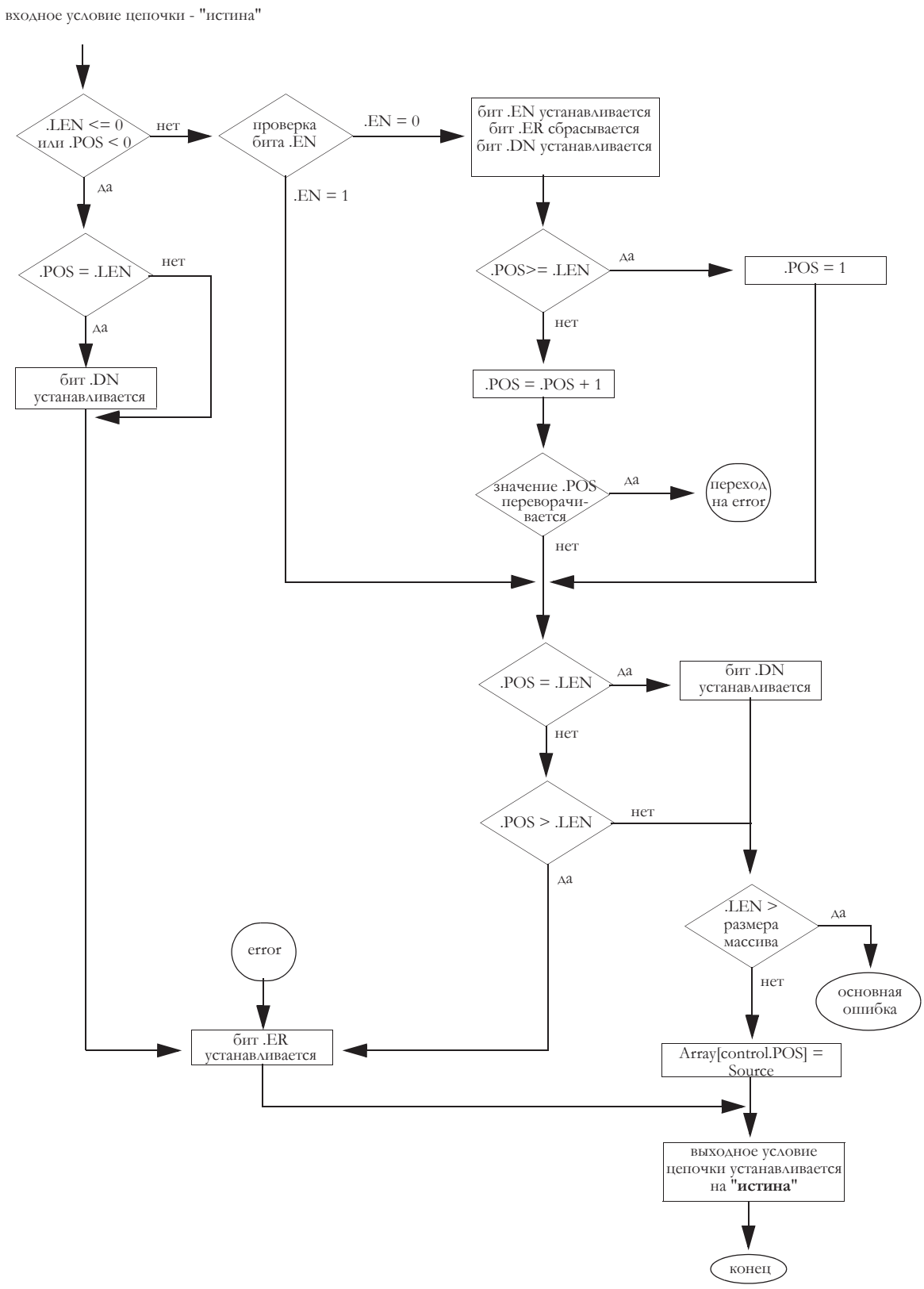
Условия ошибки:

Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
Length (длина) > размера Array (массива)	4	20

Выполнение:

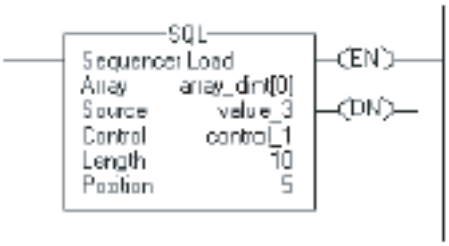
Условие:	Действие релейной логики:
предварительное сканирование	Бит .EN устанавливается, чтобы предотвратить ложную загрузку в начале сканирования программы. Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Бит .EN сбрасывается. Выходное условие цепочки устанавливается на «ложь».

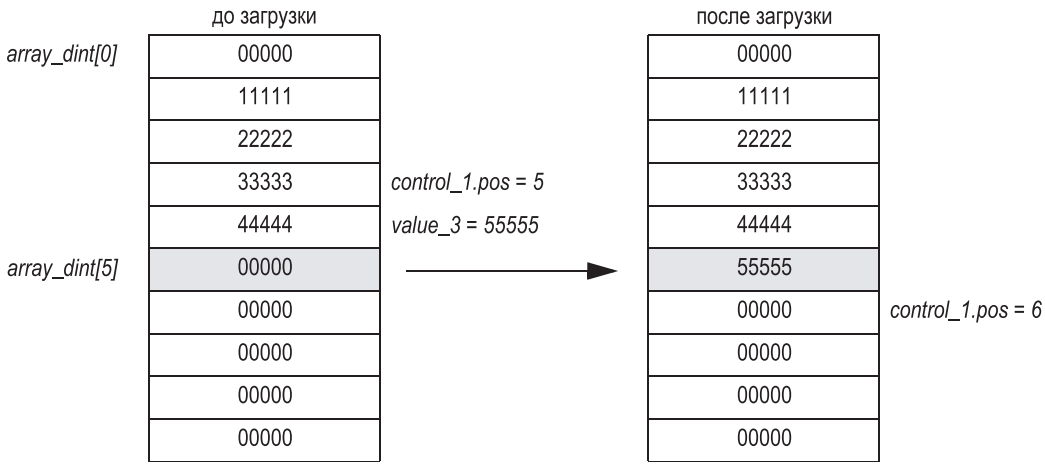
Условие:	Действие релейной логики:



постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример: Когда инструкция SQL разрешена, она загружает *value_3* в следующую позицию в массиве секвенсера, в этом примере в *array_dint[5]*.





Примечания:

Инструкции программного управления (JMP, LBL, JSR, RET, SBR, JXR, TND, MCR, UID, UIE, AFI,NOP, EOT, SFP, SFR, EVENT)

Введение

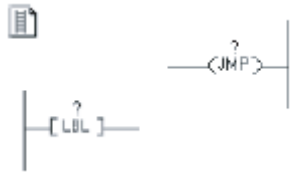
Используйте инструкции управления программой для изменения последовательности выполнения алгоритма.

Если вы хотите:	Используйте эту инструкцию:	Имеющуюся в этих языках:	См. стр.
Обойти участок алгоритма, который не надо выполнять	JMP LBL	релейная логика	10-2
Перейти к отдельной процедуре, передать данные в эту процедуру, выполнить эту процедуру и возвратить результаты.	JSR SBR RET	релейная логика функциональный блок структурированный текст	10-4
Перейти к внешней процедуре (только для контроллера SoftLogix5800)	JXR	релейная логика	10-14
Пометить временный оператор «end», который отменит выполнение процедуры.	TND	релейная логика структурированный текст	10-17
Отменить все цепочки в каком-либо участке алгоритма.	MCR	релейная логика	10-19
Отменить задачи пользователя.	UID	релейная логика структурированный текст	10-21
Разрешить выполнение задачи пользователя.	UIE	релейная логика структурированный текст	10-21
Отменить цепочку.	AFI	релейная логика	10-23
Вставить метку-указатель в алгоритм.	NOP	релейная логика	10-24
Завершить переход для последовательной функциональной схемы.	EOT	релейная логика структурированный текст	10-25
Приостановить последовательную функциональную схему.	SFP	релейная логика структурированный текст	10-27
Перегрузить последовательную функциональную схему.	SFR	релейная логика структурированный текст	10-29
Запустить выполнение задачи обработки событий.	EVENT	релейная логика структурированный текст	10-31

Jump to Label (JMP) (Переход к метке) Label (LBL) (Метка)

Инструкции JMP и LBL позволяют пропустить часть алгоритма.

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Описание:
Инструкция JMP			
Label name		имя метки	введите имя для совместной инструкции LBL
Инструкция LBL			
Label name		имя метки	в результате выполнения осуществляется переход к инструкции LBL с именем метки, на которую была сделана ссылка

Описание:

Когда инструкция JMP разрешена, она осуществляет переход к инструкции LBL, на которую была сделана ссылка, и контроллер продолжает выполнение с этого места. Если инструкция JMP не разрешена, она не влияет на выполнение алгоритма.

Инструкция JMP может перемещать выполнение алгоритма вперед или назад. Переход вперед к метке, расположенной впереди, экономит время сканирования программы, опуская участок алгоритма, пока это не необходимо. Переход назад позволяет контроллеру повторить шаги алгоритма.

Будьте внимательны, и не переходите назад чрезмерное число раз. Сторожевой таймер Watchdog может сработать по превышению времени ожидания, потому что контроллер никогда не достигает конца логической схемы, что в свою очередь вызовет ошибку контроллера.

ВНИМАНИЕ

Пропущенный участок логической схемы не сканируется. Размещайте важные элементы алгоритма вне пропускаемых участков.

Целевым объектом для инструкции JMP служит инструкция LBL с таким же именем метки. **Убедитесь, что инструкция LBL является первой в своей цепочке.**

Имя метки должно быть уникальным в пределах процедуры. Имя может:

- содержать до 40 символов
- содержать буквы, цифры и символы подчеркивания.

Арифметические флаги состояния: не затрагиваются

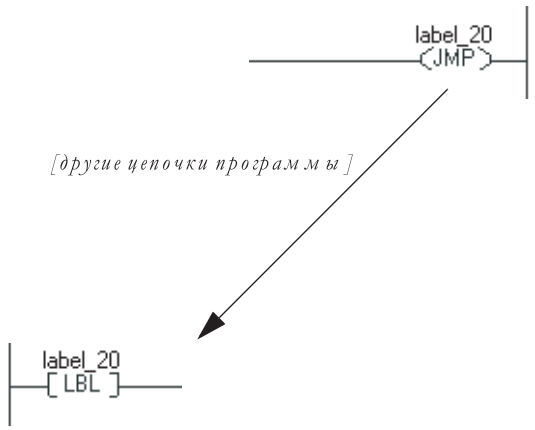
Условия ошибки:

Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
метка не существует	4	42

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Выходное условие цепочки устанавливается на «истина». Выполнение перехода к цепочке, содержащей инструкцию LBL с именем метки, на которую сделана ссылка.
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример: Если инструкция JMP разрешена, то ее выполнение приводит к переходу через последовательные цепочки алгоритма, пока не будет достигнута цепочка, содержащая инструкцию LBL с меткой *label_20*.



Jump to Subroutine (JSR) (Переход к подпрограмме) Subroutine (SBR) (Подпрограмма) Return (RET) (Возврат)

Инструкция JSR передает выполнение различным процедурам. Инструкции SBR и RET являются необязательными инструкциями, которые обмениваются данными с инструкцией JSR.

Операнды JSR:

Релейная логика

Операнд:	Тип:	Формат:	Описание:
Routine name	ROUTINE	имя	процедура для выполнения (напр., подпрограмма)
Input parameter	BOOL SINT INT DINT REAL структура	непосредственный тег тег массива	данные из этой процедуры, которые вы хотите скопировать в какой-либо тег в подпрограмме. • Входные параметры не обязательны. • Если необходимо, введите несколько входных параметров.
Return parameter	BOOL SINT INT DINT REAL структура	тег тег массива	тег в этой процедуре, в который вы хотите скопировать результат подпрограммы. • Параметры Return (возврата) – не обязательны. • Если необходимо, введите несколько параметров возврата.

Структурированный текст

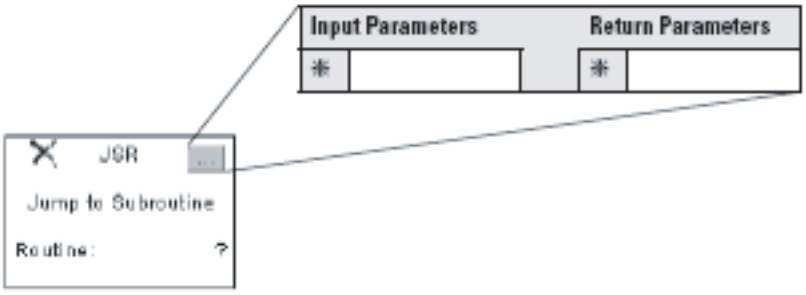
```
JSR(RoutineName,InputCount,
InputPar,ReturnPar);
```

Операнд:	Тип:	Формат:	Описание:
Routine name	ROUTINE	имя	процедура для выполнения (напр., подпрограмма)
Input count	SINT INT DINT REAL	непосредственный	количество входных параметров
Input parameter	BOOL SINT INT DINT REAL структура	непосредственный тег тег массива	данные из этой процедуры, которые вы хотите скопировать в какой-либо тег в подпрограмме. • Входные параметры не обязательны. • Если необходимо, введите несколько входных параметров.
Return parameter	BOOL SINT INT DINT REAL структура	тег тег массива	тег в этой процедуре, в который вы хотите скопировать результат подпрограммы. • Параметры Return (возврата) – не обязательны. • Если необходимо, введите несколько параметров возврата.

Операнды JSR

(продолжение)

Функциональный блок



Операнды такие же, как и операнды для инструкции JSR в релейной логике.

ВНИМАНИЕ

Для каждого параметра в инструкции SBR или RET используйте тот же самый тип данных (включая размерности массивов), что и в соответствующем параметре в инструкции JSR. Использование другого типа данных может привести к непредсказуемому результату.

Операнды SBR:

Инструкция SBR должна быть первой инструкцией в процедуре релейной логики или структурированного текста.

Релейная логика

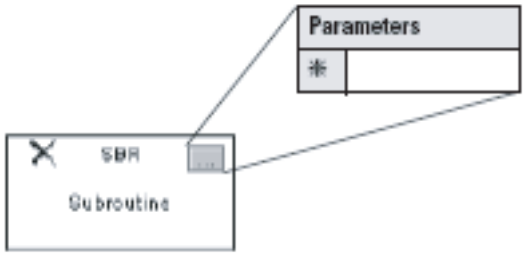
Операнд:	Тип:	Формат:	Описание:
Input parameter	BOOL SINT INT DINT REAL структура	тег тег массива	тег в этой процедуре, в который вы хотите скопировать соответствующий входной параметр из инструкции JSR

Структурированный текст

```
SBR(InputPar);
```

Операнды такие же, как и операнды для инструкции SBR в релейной логике.

Функциональный блок



Операнды такие же, как и операнды для инструкции SBR в релейной логике.

Операнды RET:

Релейная логика

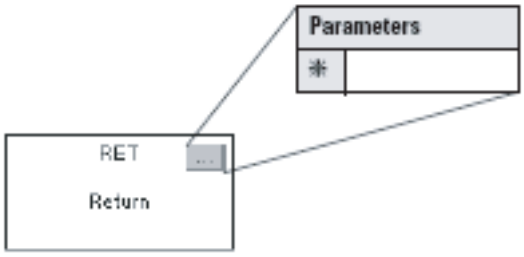
Операнд:	Тип:	Формат:	Описание:
Return parameter	BOOL SINT INT DINT REAL структура	непосредственный тег тег массива	данные из этой процедуры, которые вы хотите скопировать в соответствующий параметр возврата инструкции JSR

Структурированный текст

```
RET(ReturnPar);
```

Операнды такие же, как и операнды для инструкции RET в релейной логике.

Функциональный блок



Операнды такие же, как и операнды для инструкции RET в релейной логике.

Описание:

Инструкция JSR инициирует выполнение заданной процедуры, на которую делается ссылка как на подпрограмму:

- Подпрограмма выполняется один раз.
- После выполнения подпрограммы, выполнение алгоритма возвращается в процедуру, содержащую инструкцию JSR.

При программировании перехода к какой-либо подпрограмме следуйте этим указаниям:

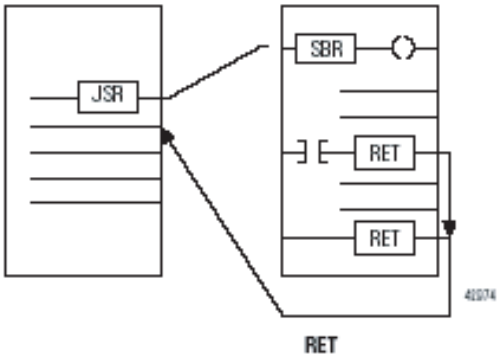
ВНИМАНИЕ

Не используйте инструкцию JSR для вызова (выполнения) основной процедуры.

- Вы можете задать инструкцию JSR в основной процедуре или любой другой процедуре.
- Если вы используете инструкцию JSR для вызова основной процедуры, а затем задаете инструкцию RET в этой основной процедуре, то произойдет основная ошибка (тип 4, код 31).

Работу этой инструкции иллюстрирует следующая схема

Вызываемая процедура Подпрограмма



JSR

1 Если вы хотите скопировать данные в тег в подпрограмме, введите входной параметр.

2 Если вы хотите скопировать результат подпрограммы в какой-либо тег в этой процедуре, введите параметр возврата return.

3 Вводите столько входных параметров и параметров возврата, сколько вам необходимо.

SBR

1 Если инструкция JSR имеет входной параметр, введите инструкцию SBR.

2 Поместите инструкцию SBR первой в этой процедуре.

3 Для каждого входного параметра в инструкции JSR введите тег, в который вы хотите копировать данные.

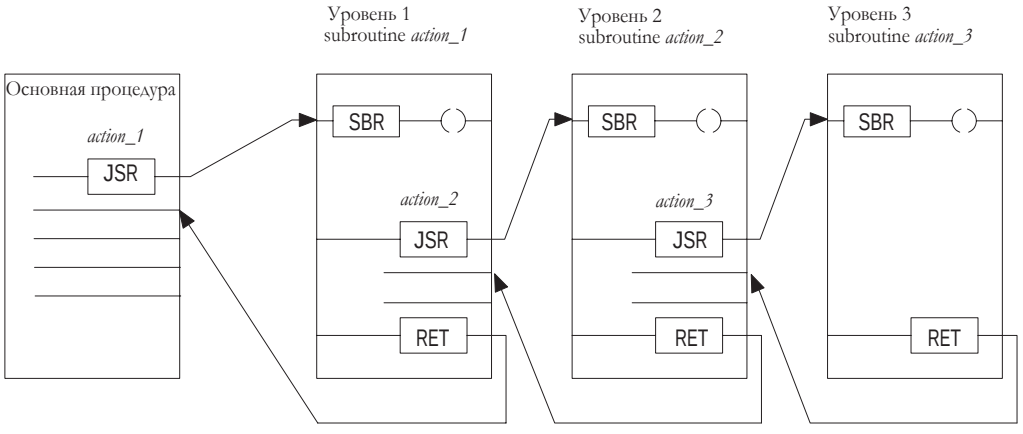
1 Если инструкция JSR имеет параметр возврата, введите инструкцию RET.

2 Поместите инструкцию RET последней в этой процедуре.

3 Для каждого параметра возврата в инструкции JSR введите параметр возврата для отправки в инструкцию JSR.

4 Если потребуется, то в процедуре релейной логики разместите дополнительные инструкции RET для выхода из подпрограммы на основе различных входных условий. (Для процедур функционального блока разрешается только одна инструкция RET.)

На количество процедур, на количество передаваемых и возвращаемых параметров нет других ограничений, кроме памяти контроллера.



Арифметические флаги состояния: Арифметические флаги состояния затрагиваются

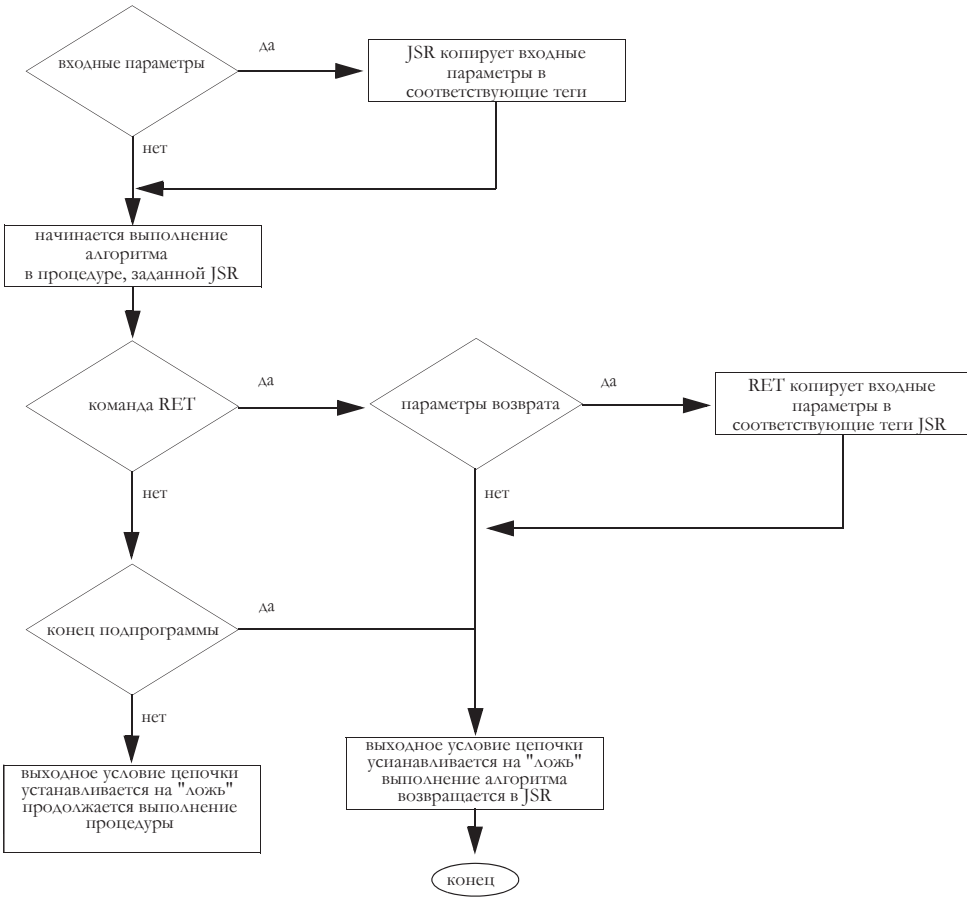
Условия ошибки:

Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
Инструкция JSR имеет меньше входных параметров, чем инструкция SBR	4	31
Инструкция JSR переходит на ошибочную процедуру	4 или задается пользователем	0 или задается пользователем
Инструкция RET имеет меньше параметров возврата, чем инструкция JSR	4	31
Основная процедура содержит инструкцию RET		

Выполнение:

Релейная логика и структурированный текст

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Контроллер выполняет все подпрограммы независимо от условий цепочки. Для того чтобы гарантировать сканирование всех цепочек в подпрограмме, контроллер игнорирует инструкции RET (т.е. инструкции RET не выводят из подпрограммы). • В версии 6.x и более ранних версиях, входные параметры и параметры возврата не рассматриваются. • В версии 7.x и более поздних версиях, входные параметры и параметры возврата рассматриваются. Если существуют рекурсивные обращения к одной и той же подпрограмме, эта подпрограмма предварительно сканируется только первый раз. Если существует несколько обращений (не рекурсивных) к одной и той же подпрограмме, эта подпрограмма предварительно сканируется каждый раз. Выходное условие цепочки устанавливается на «ложь» (только для релейной логики).	
входное условие цепочки – «ложь» для инструкции JSR	Подпрограмма *не* выполняется. Выходные значения подпрограммы сохраняют свои последние значения. Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.

Условие:	Действие релейной логики:	Действие структурированного текста:
		
постсканирова-ние	Те же действия, что описаны выше для предварительного сканирования	Те же действия, что описаны выше для предварительного сканирования

Функциональный блок

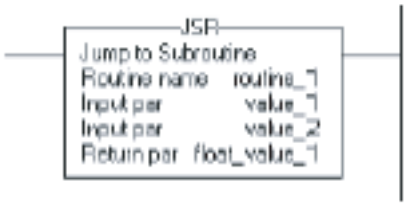
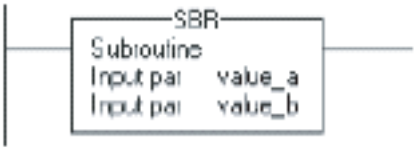
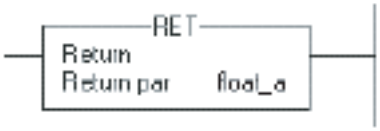
Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
нормальное выполнение	1 Если процедура содержит инструкцию SBR, то контроллер, в первую очередь, выполняет инструкцию SBR. 2 Контроллер фиксирует все значения данных в IREF. 3 Контроллер выполняет другие функциональные блоки в порядке, определенном их схемой. Это включает в себя выполнение других инструкций JSR. 4 Контроллер записывает выходные значения в OREF. 5 Если процедура содержит инструкцию RET, контроллер выполняет инструкцию RET последней.
постсканирование	Вызывается подпрограмма. Если процедура является процедурой SFC, то эта процедура инициализируется таким же образом, как и при предварительном сканировании.

Пример 1: Инструкция JSR пересылает *value_1* и *value_2* в *routine*_1.

Инструкция SBR получает *value_1* и *value_2* от инструкции JSR и копирует эти значения в *value_a* и *value*_b, соответственно. Выполнение алгоритма продолжается в этой процедуре.

Инструкция RET отправляет *float_a* в JSR. JSR получает *float_a* и копирует это значение в *float_value_1*. Выполнение алгоритма продолжается с инструкции, следующей за инструкцией JSR.

Релейная логика

Подпрограмма:	Программирование:
Основная процедура	
Подпрограмма	 *[other rungs of code]* 

Структурированный текст

Процедура:	Программа:
Основная процедура	`JSR(routine_1,2,value_1,value_2,float_value_1);`

Подпрограмма	`SBR(value_a,value_b);`
	`<statements>;`
	`RET(float_a);`

Пример 2:

Релейная логика

MainRoutine (Основная процедура)

Если *abc* включен, выполняется *subroutine_1,* подсчитывая количество пирожков и помещая значение в *cookies_1.*

Прибавляет значение *cookies_1* к *cookies_2* и сохраняет результат в *total_cookies*.

Subroutine_1

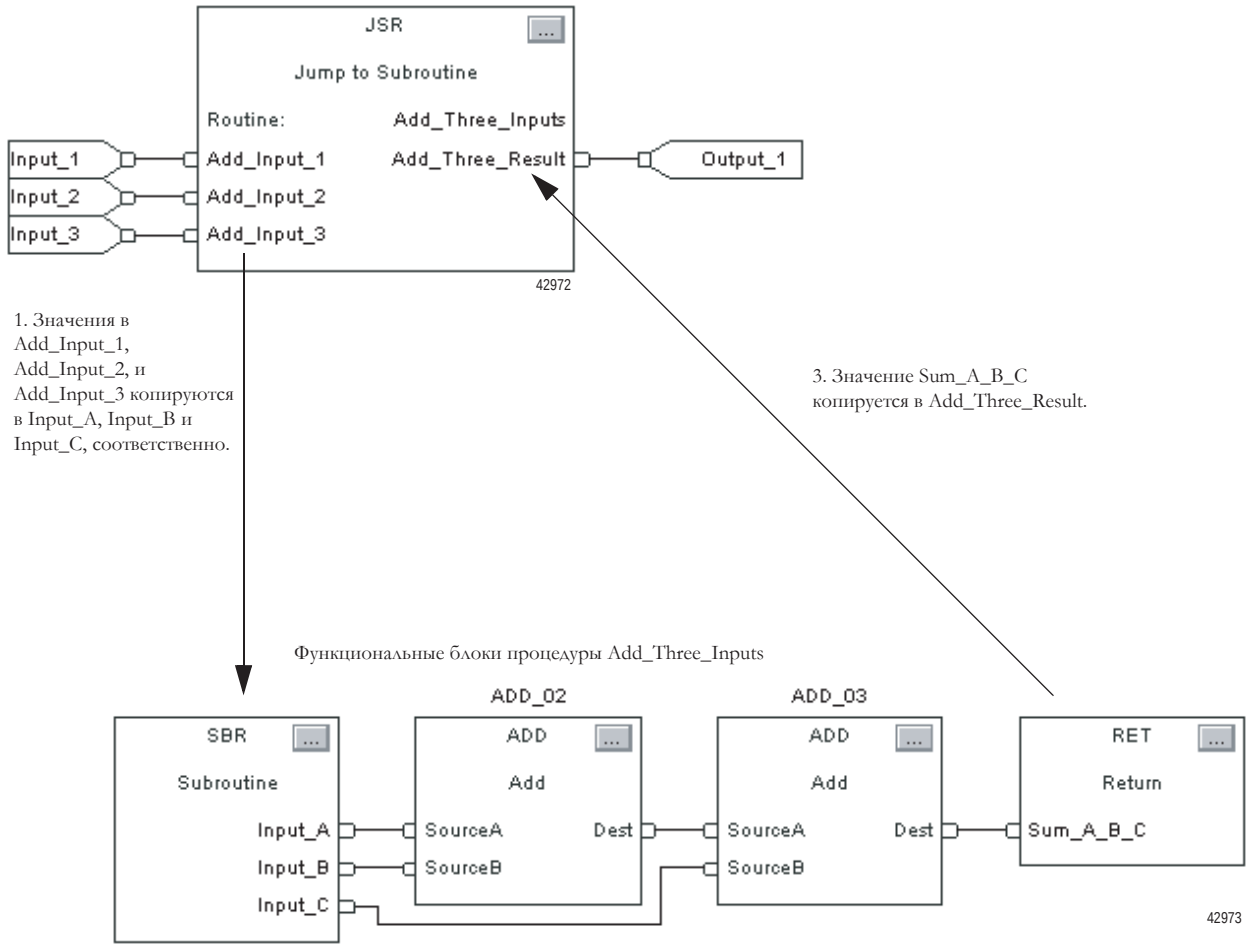
Если *def* включен, то инструкция RET возвращает *value_1* в параметр *cookies_1* инструкции JSR и оставшаяся часть подпрограммы не сканируется.

Если *def* выключен (предыдущая цепочка), а *ghi* включен, то инструкция RET возвращает *value_2* в параметр *cookies_1* инструкции JSR и оставшаяся часть подпрограммы не сканируются.

Если *def* и *ghi* выключены (предыдущая цепочка), то инструкция RET возвращает *value_3* в параметр *cookies_1* инструкции JSR.

Пример 3:

Функциональный блок



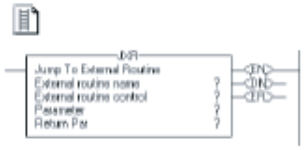
2. Инструкции ADD прибавляют Input_A, Input_B и Input_C и помещают результат в Sum_A_B_C

Jump to External Routine (JXR) (Переход к внешней процедуре)

Инструкция JXR выполняет внешнюю процедуру. Эта инструкция поддерживается только контроллерами SoftLogix5800.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
External routine name	ROUTINE	имя	внешняя процедура для выполнения
External routine control	EXT_ROUTINE_ CONTROL	тэг	управляющая структура (см. следующую стр.)
Parameter	BOOL SINT INT DINT REAL структура	непосредственный тег тег массива	данные из этой процедуры, которые вы хотите скопировать в переменную во внешней процедуру • Параметры не обязательны. • Если необходимо, вводите несколько параметров. • Максимально вы можете использовать 10 параметров.
Return parameter	BOOL SINT INT DINT REAL	тег	тег в этой процедуре, в который вы хотите скопировать результат внешней процедуры • Параметр возврата необязателен. • Вы можете использовать только один параметр возврата.

Структура EXT_ROUTINE_CONTROL

Мнемоника:	Тип данных:	Описание:	Реализация:
ErrorCode	SINT	Если происходит ошибка, это значение идентифицирует ошибку. Допустимы значения 0-255.	Предварительно заданных кодов ошибки не существует. Разработчик внешней процедуры должен предоставить коды ошибок.
NumParams	SINT	Это значение указывает количество параметров, связанных с этой инструкцией.	Только для вывода на экран - эта информация формируется при вводе инструкции.
ParameterDefs	EXT_ROUTINE_ PARAMETERS[10]	Этот массив содержит описания параметров для передачи во внешнюю процедуру. Инструкция может передавать до 10 параметров.	Только для вывода на экран – эта информация формируется при вводе инструкции.
ReturnParamDef	EXT_ROUTIN_ PARAMETERS	Это значение содержит определения параметра возврата из внешней процедуры. Возможен только один параметр возврата.	Только для вывода на экран – эта информация формируется при вводе инструкции.
EN	BOOL	Если бит разрешения установлен, то он указывает, что инструкция JXR разрешена.	Внешняя процедура присваивает этот бит.
ReturnsValue	BOOL	Если этот бит установлен, то он указывает, что для этой инструкции был введен параметр возврата. Если он сброшен (присвоен нуль), то этот бит указывает, что для этой инструкции параметр возврата не был введен.	Только для вывода на экран – эта информация формируется при вводе инструкции.
DN	BOOL	Бит выполнения устанавливается, когда внешняя процедура один раз выполнена до конца.	Внешняя процедура устанавливает этот бит.
ER	BOOL	Бит ошибки устанавливается, если произошла ошибка. Инструкция перестает выполняться до тех пор, пока программа не сбросит бит ошибки.	Внешняя процедура устанавливает этот бит.
FirstScan	BOOL	Этот бит идентифицирует, является ли это сканирование первым после переключения контроллера в режим Run. При необходимости, используйте FirstScan для инициализации внешней процедуры.	Контроллер устанавливает этот бит для отражения условий сканирования.
EnableOut	BOOL	Разрешение выхода.	Внешняя процедура устанавливает этот бит.
EnableIn	BOOL	Разрешение входа.	Этот бит устанавливается контроллером для отображения входного условия цепочки. Эта инструкция выполняется независимо от условия цепочки. Разработчик внешней процедуры должен отслеживать это условие и действовать соответственно.
User1	BOOL	Эти биты доступны пользователю. Контроллер не инициализирует эти биты.	Либо внешняя процедура, либо программа пользователя может установить эти биты.
User0	BOOL		

ScanType0	BOOL	Эти биты идентифицируют тип текущего сканирования: **Значение бита: Тип сканирования**: 00 Нормальное 01 Предварительное 10 Постсканирование (не используется для программ релейной логики)	Контроллер устанавливает эти биты для отражения условий сканирования.
ScanType()	BOOL		

Описание:

Используйте инструкцию перехода к внешней процедуре Jump to External Routine (JXR) для вызова внешней процедуы релейной логики в вашем проекте. Инструкция JXR поддерживает несколько параметров, поэтому вы можете передавать значения между процедурой релейной логики и внешней процедурой.

Инструкция JXR похожа на инструкцию перехода к подпрограмме Jump to Subroutine (JSR). Инструкция JXR инициирует выполнение заданной внешней процедуры:

- Внешняя процедура выполняется один раз.
- После выполнения внешней процедуры алгоритм осуществляет переход в процедуру, содержащую инструкцию JXR.

Арифметические флаги состояния:

Арифметические флаги состояния не затрагиваются.

Условия ошибки:

Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
•Нештатная ситуация случается во внешней процедуре DLL •DLL не может быть загружен •Точка входа не была найдена в DLL	4	88

Выполнение:

JXR может быть синхронизирована или асинхронизирована в зависимости от реализации DLL. Код в DLL может также определять реакцию на статус сканирования, статус входного условия цепочки и статус выходного условия цепочки.

За более подробной информацией по использованию инструкций JXR и созданию внешних процедур обращайтесь к руководству пользователя *SoftLogix5800 System*, публикация 1789-UM002.

Temporary End (TND) (Временный конец)

Инструкция TND работает как граница.

Операнды:

Операнды релейной логики

отсутствуют

Структурированный текст

отсутствуют

Вы должны ввести круглые скобки () после мнемоники этой инструкции, даже если операнды отсутствуют.

Описание:

Когда инструкция TND разрешена, она позволяет контроллеру выполнять алгоритм только до этой инструкции.

Когда инструкция TND разрешена, она работает как оператор «конец процедуры». Когда контроллер сканирует инструкцию TND, он перемещается на конец текущей процедуры. Если инструкция TND находится в какой-либо подпрограмме, контроллер возвращается в вызывающую процедуру. Если инструкция TND находится в основной процедуре, управление передается следующей программе в пределах текущей задачи.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

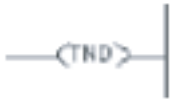
отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Текущая процедура завершается.	Текущая процедура завершается.
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Вы можете использовать инструкцию TND при отладке или поиске неисправностей, выполняя алгоритм до определенной точки. Постепенно перемещайте инструкцию TND по алгоритму по мере отладки новых разделов. Когда инструкция TND разрешена, контроллер останавливает сканирование текущей процедуры.

Релейная логика



Структурированный текст

```
TND();
```

Master Control Reset (MCR) (Сброс основного управления)

Инструкция MCR, используемая попарно, создает зону программы, в которой могут быть отключены все цепочки внутри инструкций MCR.

Операнды:

Релейная логика

отсутствуют

Описание: Если зона MCR разрешена, то цепочки в зоне MCR сканируются на предмет нормальных условий «истина» или «ложь». Если зона MCR отключена, контроллер по-прежнему сканирует цепочки внутри зоны MCR, но время сканирования уменьшается, так как все несохраняемые выходы в данной зоне запрещаются. Выходное условие цепочки устанавливается на «ложь» для всех инструкций внутри запрещенной зоны MCR.

Когда вы программируете зону MCR, обратите внимание, что:

- Вы должны закрыть зону безусловной инструкцией MCR.
- Вы не можете вкладывать одну зону MCR в другую.
- Не программируйте переход в зону MCR. Если зона имеет значение «ложь», то переход в зону разрешает зону с той точки, куда был осуществлен переход, до конца зоны.
- Если зона MCR простирается до конца процедуры, вам нет нужды программировать инструкцию MCR на конце зоны.

Инструкция MCR не является заменой жестко смонтированного управляющего реле, предоставляющего возможность аварийной остановки. Вам следует по-прежнему устанавливать жестко смонтированное управляющее реле для аварийного отключения питания ввода/вывода.

ВНИМАНИЕ

Не перекрывайте и не вкладывайте одну зону MCR в другую. Каждая зона MCR должна стоять отдельно и быть замкнутой. Если зоны перекроются или будут находиться одна внутри другой, то возможны непредсказуемые механические операции, которые могут привести к повреждению оборудования или травмированию персонала.

Размещайте важные операции вне зоны MCR. Если вы запустили такую инструкцию, как таймер, внутри зоны MCR, выполнение инструкции остановится, когда эта зона запрещена, и таймер сбросится.

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь». Инструкции в зоне сканируются, но входное условие цепочки – «ложь», несохраняемые выходы в этой зоне запрещаются.
входное условие цепочки – «истина»	Выходное условие цепочки устанавливается на «истина». Инструкции в зоне сканируются обычным образом.
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример: Когда первая инструкция MCR разрешается (*input*_1, *input*_2 и *input_3* устанавливаются), контроллер выполняет цепочки внутри зоны MCR (между двумя инструкциями MCR) и устанавливает или сбрасывает выходы в зависимости от входных условий.

Когда первая инструкция MCR отключена *(input*_1, *input*_2 и *input_3* не установлены), контроллер выполняет цепочки в зоне MCR (между двумя инструкциями MCR) и входное условие цепочки приобретает значение «ложь» для всех цепочек в зоне MCR независимо от входных условий.

User Interrupt Disable (UID) User Interrupt Enable (UIE) (Запрещение/ Разрешение прерываний)

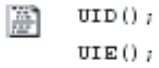
Инструкции UID и UIE работают вместе, предохраняя небольшое количество основных цепочек от прерывания другими задачами.

Операнды:

Релейная логика

отсутствуют

Структурированный текст



отсутствуют

Вы должны ввести круглые скобки () после мнемоники этой инструкции, даже если операнды отсутствуют.

Описание:

Если входное условие цепочки – «истина», то:

- Инструкция UID предотвращает прерывание текущей задачи более приоритетными задачами, но *не* запрещает выполнение процедуры отказов или Controller Fault Handler (обработчика отказов контроллера).
- Инструкция UIE разрешает другим задачам прерывать текущую задачу.

Чтобы предохранить часть цепочек от прерывания:

1. Сократите до минимума количество цепочек, которые вы хотите предохранить от прерывания. Запрещение прерываний на продолжительный период времени может вызвать потерю связи.
2. Над первой цепочкой из тех, которые вы *не* хотите прерывать, введите одну цепочку и инструкцию UID.
3. После последней цепочки из тех, которые вы *не* хотите прерывать, введите одну цепочку и инструкцию UIE.
4. Если необходимо, вы можете вкладывать пары инструкций UID/ UIE внутрь других пар.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

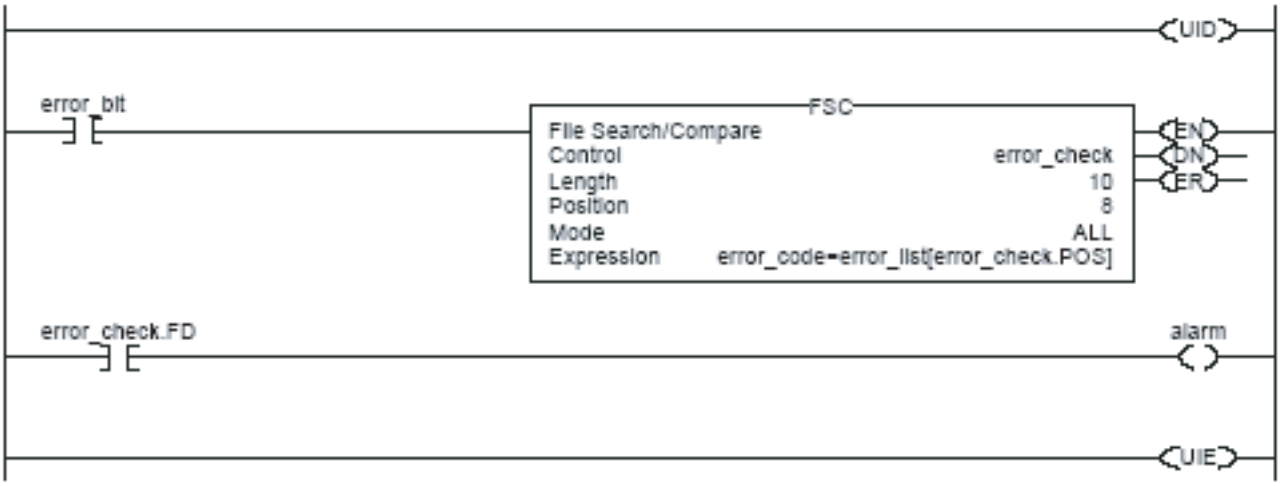
отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция UID предотвращает прерывание более высокоприоритетными задачами. Инструкция UIE разрешает прерывание более высокоприоритетными задачами.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Когда происходит ошибка (устанавливается бит *error_bit*), инструкция FSC проверяет код ошибки по списку основных ошибок. Если инструкция FSC обнаружила, что ошибка является основной, (включен error_check.FD), выдается сигнал тревоги. Инструкции UID и UIE предохраняют проверку ошибки и выдачу аварийного сигнала от прерывания другими задачами.

Релейная логика



Структурированный текст

```
UID();

<statements>

UIE();
```

Always False Instruction (AFI) (Всегда ложная инструкция)

Инструкция AFI устанавливает выходное условие цепочки на «ложь».

Операнды:

Релейная логика

отсутствуют

Описание:

Инструкция AFI устанавливает выходное условие цепочки на «ложь».

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Выходное условие цепочки устанавливается на «ложь».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример:

Использование инструкции AFI для временного запрещения цепочки во время отладки программы.

Когда инструкция AFI разрешена, она запрещает все инструкции данной цепочки.

No Operation (NOP) (Нет операции)

Инструкция NOP работает как метка-заполнитель.

Операнды:

Релейная логика

отсутствуют

Описание:

Вы можете поместить инструкцию NOP в любом месте цепочки. Когда инструкция NOP разрешена, она не выполняет никаких операций. Когда инструкция NOP запрещена, она не выполняет никаких операций.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример:

Эта инструкция полезна для установления безусловных переходов, когда вы размещаете инструкцию NOP на ветви.

Чтобы разрешить выход, инструкция NOP обходит инструкцию XIC.

End of Transition (EOT) (Завершение перехода)

Инструкция EOT возвращает булево условие переходу SFC.

Операнды:

Релейная логика

Операнд:	Тип:	Формат:	Описание:
data bit	BOOL	тег	состояние перехода (0=выполняется, 1=завершен)

Структурированный текст

Операнды такие же, как и операнды для инструкции EOT в релейной логике.

Описание:

Поскольку инструкция EOT возвращает булево условие, то несколько процедур SFC могут использовать одну и ту же процедуру, которая содержит инструкцию EOT. Если вызывающая процедура не является переходом, инструкция EOT действует как инструкция TND (см. стр. 10-17).

Выполнение инструкции EOT контроллером Logix отличается от выполнения этой же инструкции контроллером PLC-5. В контроллере PLC-5 инструкция EOT не имеет параметров. Взамен, инструкция EOT контроллера PLC-5 возвращает условие цепочки в качестве ее состояния. В контроллере Logix параметр возврата возвращает состояние перехода, поскольку условие цепочки отсутствует во всех языках программирования контроллера Logix.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция возвращает значение бита данных в вызывающую процедуру.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Установка state (состояния), когда *limit_switch1* и *interlock_1* установлены. После завершения *timer_1* инструкция EOT возвращает значение *state* вызывающей процедуре.

Релейная логика

Структурированный текст

```
state := limit_switch1 AND interlock_1;

IF timer_1.DN THEN

      EOT(state);

END_IF;
```

SFC Pause (SFP) (Пауза SFC)

Инструкция SFP приостанавливает процедуру SFC.

Операнды:

Релейная логика

Операнд:	Тип:	Формат:	Описание:
SFCRoutine Name	ROUTINE	имя	процедура SFC для приостановки
TargetState	DINT	непосредственный тег	выберите что-нибудь одно: выполнение (или введите 0) приостановлена (или введите 1)

Структурированный текст

```
SFP(SFCRoutineName,
    TargetState);
```

Операнды такие же, как и операнды для инструкции EOT в релейной логике.

Описание:

Инструкция SFP позволяет вам приостановить выполнение процедуры SFC. Если процедура SFC приостановлена, то повторное использование инструкции SFP вновь сменит условие и продолжит выполнение процедуры.

Используйте также инструкцию SFP для возобновления выполнения процедуры SFC после применения инструкции SFR (см. стр. 10-29) для перезагрузки процедуры SFC.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
тип процедуры не SFC	4	85

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция приостанавливает или возобновляет выполнение заданной процедуры SFC.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Если *sfc_en_p* установлен, приостановка процедуры SFC с именем *normal*. Если *sfc_en_e* установлен, перезапуск SFC.

Релейная логика

Приостановка процедуры SFC.

Возобновление выполнения процедуры SFC.

Структурированный текст

Приостановка процедуры SFC

```
IF  (sfp_en_p)  THEN
      SFP(normal,paused);
      sfp_en_p  :=  0;
END_IF;
```

Возобновление выполнения процедуры SFC:

```
IF  (sfp_en_e)  THEN
      SFP(normal,executing);
      sfp_en_e  :=  0;
END_IF;
```

SFC Reset (SFR) (Сброс SFC)

Инструкция SFR сбрасывает выполнение процедуры SFC на заданном шаге.

Операнды:

Операнды релейной логики

Операнд:	Тип:	Формат:	Описание:
SFCRoutine Name	ROUTINE	имя	процедура SFC для сброса
Step Name	SFC_STEP	тег	целевой шаг, с которого продолжается выполнение

Структурированный текст

```
SFR(SFCRoutineName,StepName);
```

Операнды такие же, как и операнды для инструкции SFR в релейной логике.

Описание:

Когда инструкция SFR разрешена:

- В заданной процедуре SFC выполнение всех хранимых операций останавливается (сброс).
- SFC начинает выполняться с заданного шага.

Если целевой шаг 0, схема будет перезагружена на свой исходный шаг.

Контроллер Logix выполняет инструкцию SFR не так, как PLC-5. В контроллере PLC-5 SFR выполнялась при условии цепочки – «истина». После сброса SFC должна была оставаться приостановленной, пока цепочка, содержащая SFR, не принимала значение «ложь». Это позволяло задерживать выполнение, следующее за сбросом. Это качество приостановки/возобновления работы было оторвано от условия цепочки и передано инструкции SFP.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
тип процедуры не SFC	4	85
заданный целевой шаг не существует в процедуре SFC	4	89

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция сбрасывает заданную процедуру SFC.	Инструкция сбрасывает заданную процедуру SFC.
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Если имеет место заданное состояние *(shutdown* установлен), перезагрузка SFC с шага *initialize.*

Релейная логика

Структурированный текст

```
IF  shutdown  THEN
      SFR(mySFC,initialize);
END_IF;
```

Trigger Event Task (EVENT) (Запуск задачи обработки событий)

Инструкция EVENT запускает одноразовое выполнение какой-либо задачи по обработке события.

Операнды:

Релейная логика

Операнд:	Тип:	Формат:	Описание:
Task	TASK	имя	задача по обработке события для выполнения. Эта инструкция позволяет вам выбирать другие типы задач, но она не выполняет их.

Структурированный текст

```
EVENT(task_name);
```

Операнды такие же, как и операнды для инструкции EVENT в релейной логике.

Описание:

Используйте инструкцию EVENT для программного выполнения задачи по обработке события:

- Каждый раз, когда выполняется инструкция, она запускает заданную задачу обработки события.
- Убедитесь, что вы дали задаче обработки события достаточно времени для завершения выполнения, до того как вы запустите ее еще раз. Если времени не достаточно, произойдет перекрытие.
- Если вы выполняете инструкцию EVENT, когда задача обработки события еще выполняется, контроллер увеличивает значение счетчика перекрытий, но задача обработки события не запускается.

Программное определение факта запуска задачи инструкцией EVENT

Чтобы определить, запустила ли инструкция EVENT задачу обработки события, используйте инструкцию Get System Value (GSV) (получить системное значение) для контроля атрибута Status (статус) этой задачи.

Таблица 10.1 Атрибут Status объекта TASK

Атрибут:	Тип данных:	Инструкция:	Описание:
Status	DINT	GSV SSV	Предоставляет информацию о состоянии задачи. Как только контроллер установил какой-либо бит, вы должны вручную сбросить этот бит для определения, произошла ли другая ошибка этого типа.

Чтобы определить, что:	Проверьте этот бит:
Какая-либо инструкция EVENT запустила эту задачу (только задача обработки события).	0
Истечение времени ожидания запустило эту задачу *(только задача обработки события)*.	1
Для этой задачи произошло перекрытие.	2

После того как биты атрибута Status установлены, контроллер их не сбрасывает.

- Для того чтобы использовать бит для новой информации о состоянии, вы должны сбросить его вручную.
- Используйте инструкцию Set System Value (SSV) (задать системное значение) для установки этого атрибута для различных значений.

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция запускает одноразовое выполнение задачи обработки события.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример 1: Контроллер использует несколько программ, но общую процедуру завершения. Каждая программа использует тег слежения за программой, называемый *Shut_Down_Line,* который включается, если программа обнаружила условие, которое требует завершения. В каждой программе этот алгоритм выполняется следующим образом:

Если *Shut_Down_Line* = включен (условия, требующие завершения), то однократно выполняется задача *Shut_Down*

Релейная логика

Программа A

Программа B

Структурированный текст

Программа A

```
IF Shut_Down_Line AND NOT Shut_Down_Line_One_Shot THEN
      EVENT  (Shut_Down);
END_IF;
Shut_Down_Line_One_Shot  := Shut_Down_Line;
```

Программа B

```
IF Shut_Down_Line AND NOT Shut_Down_Line_One_Shot THEN
      EVENT  (Shut_Down);
END_IF;
Shut_Down_Line_One_Shot  := Shut_Down_Line;
```

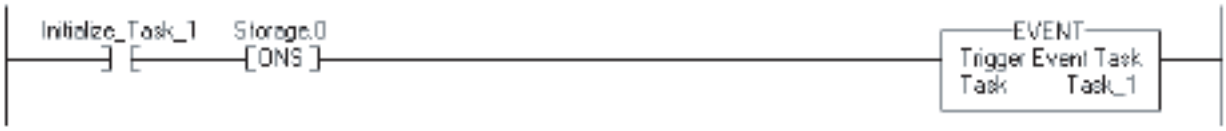
Пример 2: В следующем примере инструкция EVENT используется для запуска задачи обработки события. (Обычно другой тип события запускает задачу обработки событий).

Непрерывная задача

Если *Initialize_Task_1* = 1, то

Инструкция ONS ограничивает выполнение инструкции EVENT одним сканированием.

Инструкция EVENT запускает выполнение *Task_1* (задача обработки события).

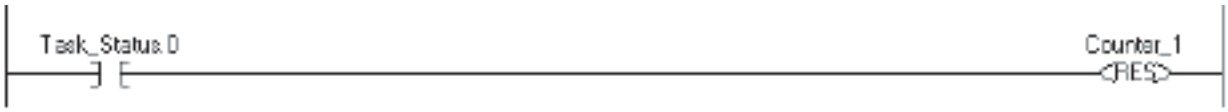


Task_1 (задача обработки события)

Инструкция GSV устанавливает *Task_Status* (тег DINT) = атрибут Status для задачи обработки события. В атрибуте Instance Name, THIS означает объект TASK для задачи, в которой расположена инструкция (т.е. *Task*_1).

Если *Task_Status*.0 = 1, то инструкция EVENT запустила эту задачу обработки события (т.е. ситуация, когда непрерывная задача выполняет свою инструкцию EVENT, чтобы инициировать задачу обработки события).

Инструкция RES сбрасывает счетчик, который используется этой задачей обработки события.

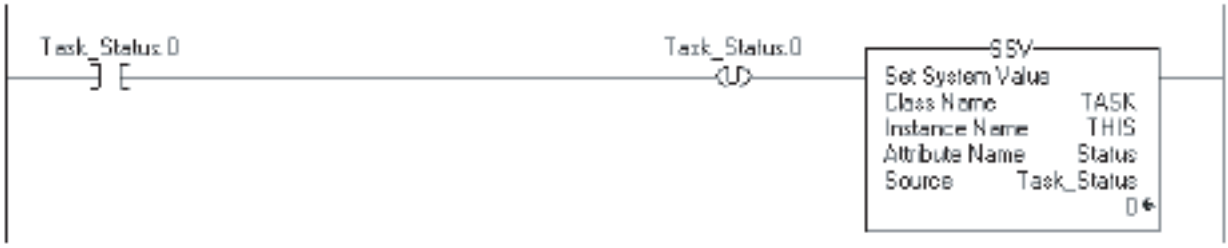


Если биты атрибута Status установлены, контроллер их не сбрасывает. Чтобы использовать бит для новой информации состояния, вам необходимо вручную сбросить этот бит.

Если *Task_St at us*.0 = 1, то сбросьте этот бит.

Инструкция OTU устанавливает *Task_Status*.0 = 0.

Инструкция SSV устанавливает атрибут Status задачи THIS *(Task*_1) = *Task_Status*. Эта операция включает сброшенный бит.



Инструкции FOR/BREAK (FOR, FOR...DO, BRK, EXIT, RET)

Введение

Используйте инструкцию FOR для неоднократного вызова какой-либо подпрограммы. Используйте инструкцию BRK для прерывания выполнения какой-либо подпрограммы.

Если вы хотите:	Используйте эту инструкцию:	Имеющуюся в этих языках:	См. стр.
Неоднократно выполнить процедуру.	FOR FOR...DO(1)	релейная логика структурированный текст	11-2
Отключить многократное выполнение процедуры.	BRK EXIT (1)	релейная логика структурированный текст	11-5
Вернуться в инструкцию FOR.	RET	релейная логика	11-6

(1) Только структурированный текст.

For (FOR) (Цикл For)

Инструкция FOR многократно выполняет какую-либо процедуру.

Операнды:

Релейная логика

Операнд:	Тип:	Формат:	Описание:
Routine name	ROUTINE	имя процедуры	процедура для выполнения
Index	DINT	тег	подсчитывает, сколько раз была выполнена процедура
Initial value	SINT INT DINT	непосредственный тег	значение, с которого запускается индекс (index)
Terminal value	SINT INT DINT	непосредственный тег	значение, при котором останавливается выполнение процедуры
Step size	SINT INT DINT	непосредственный тег	значение, прибавляемое к индексу каждый раз, когда инструкция FOR выполняет процедуру

Структурированный текст

```
FOR count := initial_value TO
final_value BY increment DO
    <statement>;
END_FOR;
```

Используйте конструкцию FOR...DO. Информацию о конструкциях структурированного текста можно найти в Приложении C.

Описание:

ВАЖНО!

Не используйте инструкцию FOR для вызова (выполнения) основной процедуры.

- Вы можете помещать инструкцию FOR в основную процедуру или любую другую процедуру.
- Если вы используете инструкцию FOR для вызова основной процедуры, а затем поместите инструкцию RET в эту основную процедуру, то произойдет основная ошибка (тип 4, код 31).

Когда инструкция FOR разрешена, она многократно выполняет процедуру Routine, пока значение Index (индекс) не превысит значение Terminal (конечное). Эта инструкция не передает параметры в процедуру.

Каждый раз, когда инструкция FOR выполняет процедуру, она прибавляет размер Step (шага) к Index (индексу).

Не делайте слишком много циклов в одном сканировании. Излишнее количество повторений может привести к останову по времени, вызванному контрольным таймером watchdog, что послужит причиной возникновения основной ошибки.

Арифметические флаги состояния:

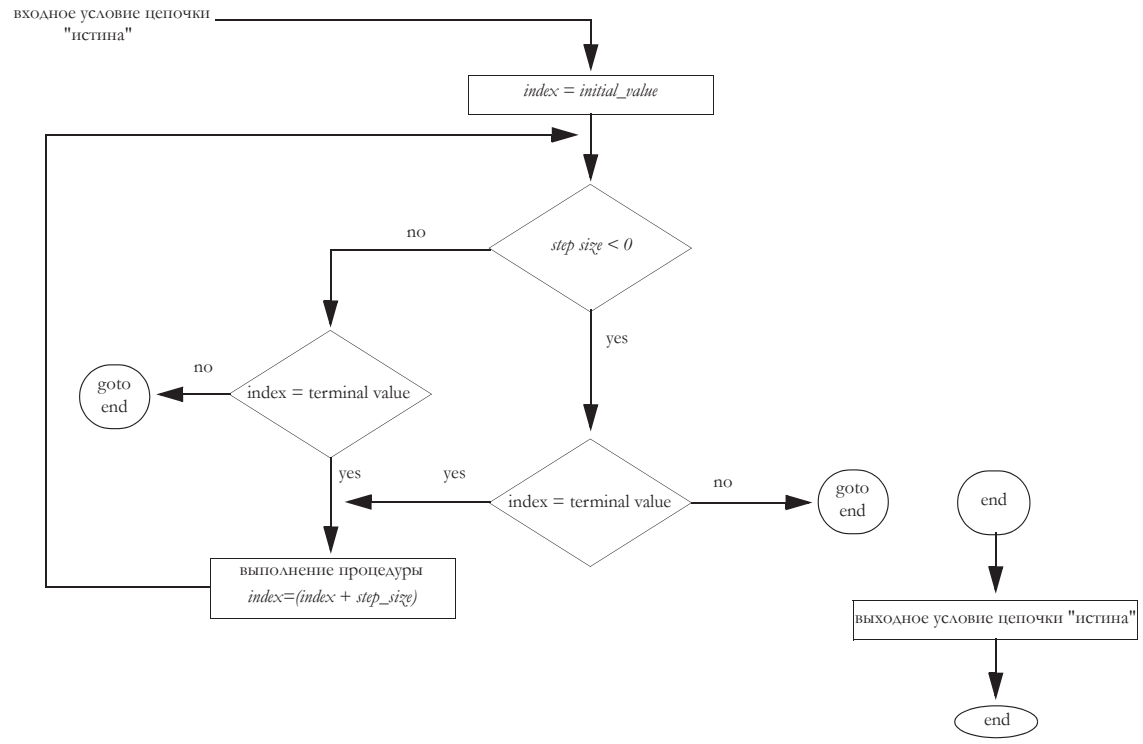
не затрагиваются

Условия ошибки:

Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
основная процедура содержит инструкцию RET	4	31

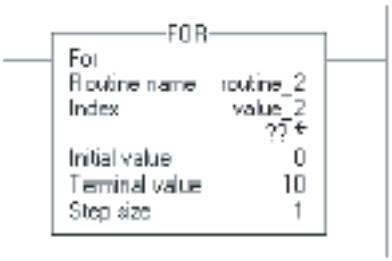
Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Выходное состояние цепочки устанавливается на «ложь». Контроллер выполняет подпрограмму один раз. Если для этой подпрограммы существуют рекурсивные инструкции FOR, подпрограмма предварительно сканируется только первый раз. Если для этой подпрограммы существует несколько инструкций FOR (не рекурсивных), подпрограмма предварительно сканируется каждый раз.
входное состояние цепочки – «ложь»	Выходное состояние цепочки устанавливается на «ложь».



постсканирование	Выходное состояние цепочки устанавливается на «ложь».

Пример: Когда инструкция FOR разрешена, она многократно выполняет *routine_2* и каждый раз увеличивает *value_2* на 1. Если *value_2* больше 10 или если разрешена инструкция BRK, инструкция FOR прекращает выполнение *routine*_2.



Break (BRK) (Прерывание)

Инструкция BRK прерывает выполнение процедуры, которая была вызвана инструкцией FOR.

Операнды:

Релейная логика

отсутствуют

Структурированный текст

Используйте в конструкции цикла оператор EXIT. Информацию о конструкциях структурированного текста можно найти в руководстве «Программирование структурированного текста».

Описание:

Когда инструкция BRK разрешена, она осуществляет выход из процедуры и возвращает контроллер к инструкции, следующей за FOR.

Если есть вложенные инструкции FOR, инструкция BRK возвращает управление инструкции FOR наиболее глубокого уровня.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Выходное состояние цепочки устанавливается на «ложь».
входное состояние цепочки – «ложь»	Выходное состояние цепочки устанавливается на «ложь».
входное состояние цепочки – «истина»	Выходное состояние цепочки устанавливается на «истина». Выполнение возвращается к инструкции, следующей за инструкцией FOR.
постсканирование	Выходное состояние цепочки устанавливается на «ложь».

Пример:

Когда инструкция BRK разрешена, она останавливает выполнение текущей процедуры и возвращается к инструкции, следующей за вызывающей инструкцией FOR.

Return (RET) (Возврат)

Инструкция RET осуществляет возврат к вызывающей инструкции FOR.

Операнды:

Релейная логика

отсутствуют

Описание:

ВАЖНО! Не размещайте инструкцию RET в основной процедуре. Если вы ввели инструкцию RET в основную процедуру, то произойдет основная ошибка (типе 4, код 31).

Когда инструкция RET разрешена, она возвращается к инструкции FOR. Инструкция увеличивает значение Index (индекс) на значение Step size (размер шага) и вновь выполняет подпрограмму. Если значение Index превышает значение Terminal value (конечное значение), инструкция FOR завершается, и выполнение передается инструкции, следующей за этой инструкцией FOR.

В инструкции FOR параметры не используются. Инструкция FOR игнорирует любые инструкции, которые вы вводите в инструкции RET.

Для завершения выполнения подпрограммы вы также можете использовать инструкцию TND.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

Основная ошибка произойдет, если	Тип ошибки:	Код ошибки:
основная процедура содержит инструкцию RET	4	31

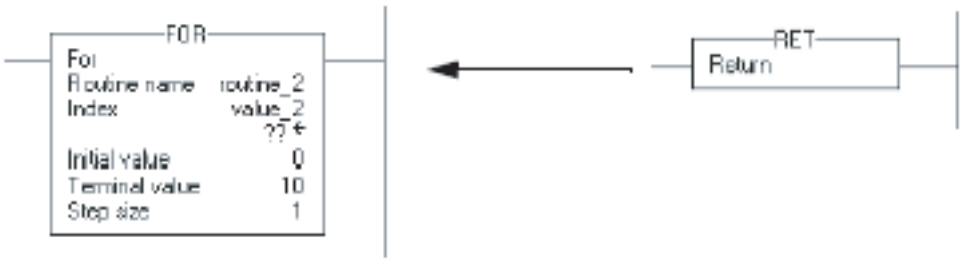
Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Выходное состояние цепочки устанавливается на «ложь».
входное состояние цепочки – «ложь»	Выходное состояние цепочки устанавливается на «ложь».
входное состояние цепочки – «истина»	Возвращает заданные параметры в вызывающую процедуру. Выходное состояние цепочки устанавливается на «истина».
постсканирование	Выходное состояние цепочки устанавливается на «ложь».

Пример: Инструкция FOR многократно выполняет *routine_2* и каждый раз увеличивает значение *value_2* на 1. Если *value_2* больше 10 или если разрешена инструкция BRK, инструкция FOR прекращает выполнение *routine*_2.

Инструкция RET осуществляет возврат к вызывающей инструкции FOR. Инструкция FOR либо опять выполняет подпрограмму и увеличивает значение Index на Step size, либо, если значение Index превышает значение Terminal value, завершается и выполнение передается инструкции, следующей за этой инструкцией FOR.

вызывающая процедура **подпрограмма**



Примечания:

Специальные инструкции (FBC, DDT, DTR, PID)

Введение

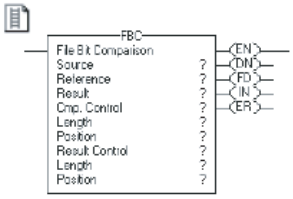
Специальные инструкции выполняют проблемно-ориентированные операции.

Если вы хотите:	Используйте эту инструкцию:	Имеющуюся в этих языках:	См. стр. :
Сравнить данные с известными контрольными данными и записать любые несоответствия.	FBC	релейная логика	12-2
Сравнить данные с известными контрольными данными, записать любые несоответствия и обновить контрольные данные, чтобы они соответствовали данным источника.	DOT	релейная логика	12-10
Пропустить исходные данные через маску и сравнить результат с контрольными данными. Затем записать исходные данные в контрольные для следующего сравнения.	DTR	релейная логика	12-18
Управлять циклом PID.	PID	релейная логика структурированный текст	12-21

File Bit Comparison (FBC) (Файловое сравнение битов)

Инструкция FBC сравнивает биты массива Source (источнике) с битами контрольного массива Reference.

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Описание:
Source	DINT	тег массива	массив для сравнения с массивом reference **не используйте** CONTROL.POS в нижнем регистре
Reference	DINT	тег массива	массив для сравнения с массивом source **не используйте** CONTROL.POS в нижнем регистре
Result	DINT	тег массива	массив для хранения результата **не используйте** CONTROL.POS в нижнем регистре
Cmp control	CONTROL	структура	управляющая структура для сравнения
Length	DINT	непосредственный	количество битов для сравнения
Position	DINT	непосредственный	текущая позиция в массиве source исходное значение обычно 0
Result control	CONTROL	структура	управляющая структура для результатов
Length	DINT	непосредственный	количество позиций для хранения в результате
Position	DINT	непосредственный	текущая позиция в результирующем массиве исходное значение обычно 0

ВНИМАНИЕ

Используйте разные теги для управляющей структуры для сравнения и управляющей структуры для результатов. Использование одних и тех же тегов может привести к непредсказуемым операциям, способным повредить оборудование и/или нанести травму персоналу.

Конструкция COMPARE

Мнемоника:	Тип данных:	Описание:
.EN	BOOL	Бит разрешения указывает на то, что инструкция FBC разрешена.
.DN	BOOL	Бит выполнения устанавливается, когда инструкция FBC сравнивает последний бит в массивах Source и Reference.
.FD	BOOL	Бит обнаружения устанавливается каждый раз, когда инструкция FBC записывает несоответствие (по одному), или после записи всех несоответствий (все на сканирование).
.IN	BOOL	Бит запрещения указывает режим поиска FBC. 0 = сразу всех 1 = по одному
.ER	BOOL	Бит ошибки устанавливается, если .POS < 0, .LEN < 0, .POS<0 или .LEN < 0. Выполнение инструкции останавливается до тех пор, пока инструкция не сбросит бит .ER.
.LEN	DINT	Значение длины указывает количество битов для сравнения.
.POS	DINT	Значение позиции указывает текущий бит.

Структура RESULT

Мнемоника:	Тип данных:	Описание:
.DN	BOOL	Бит выполнения устанавливается, когда массив Result заполнен.
.LEN	DINT	Значение длины указывает количество позиций для хранения в массиве Result.
.POS	DINT	Значение позиции указывает текущую позицию в массиве Result.

Описание: Когда инструкция FBC разрешена, она сравнивает биты в массиве Source с битами в массиве Reference и при каждом несоответствии записывает номер бита в массив Result.

Инструкция FBC производит операции с непрерывной областью памяти.

Различие между инструкциями DDT и FBC заключается в том, что каждый раз, когда DDT находит несоответствие, она изменяет бит массива reference так, чтобы он совпадал с битом массива source. Инструкция FBC не изменяет бит массива reference.

Выбор режима поиска

Если вы хотите обнаружить:	Выберите этот режим:
Одно несоответствие за один раз.	Установите бит .IN в структуре CONTROL. Каждый раз, когда входное условие цепочки меняет значение «ложь» на «истина», инструкция FBC ищет следующее несоответствие между массивами Source и Reference. Как только несоответствие найдено, инструкция устанавливает бит .FD, записывает позицию этого несоответствия и останавливает выполнение.
Все несоответствия.	Сбросьте бит.IN в структуре CONTROL. Каждый раз, когда входное условие цепочки меняет значение «ложь» на «истина», инструкция FBC ищет все несоответствия между массивами Source и Reference.

Арифметические флаги состояния:

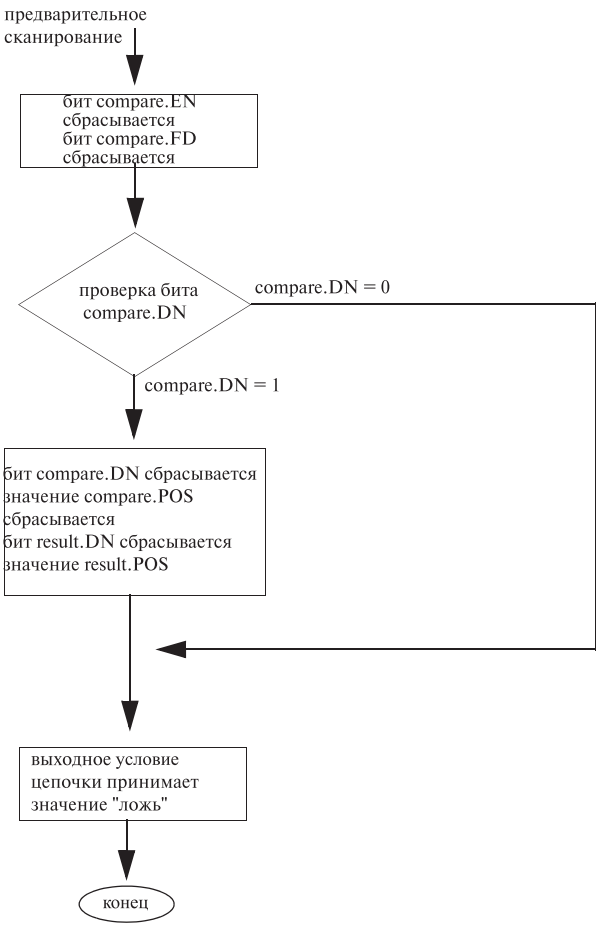
не затрагиваются

Условия ошибки:

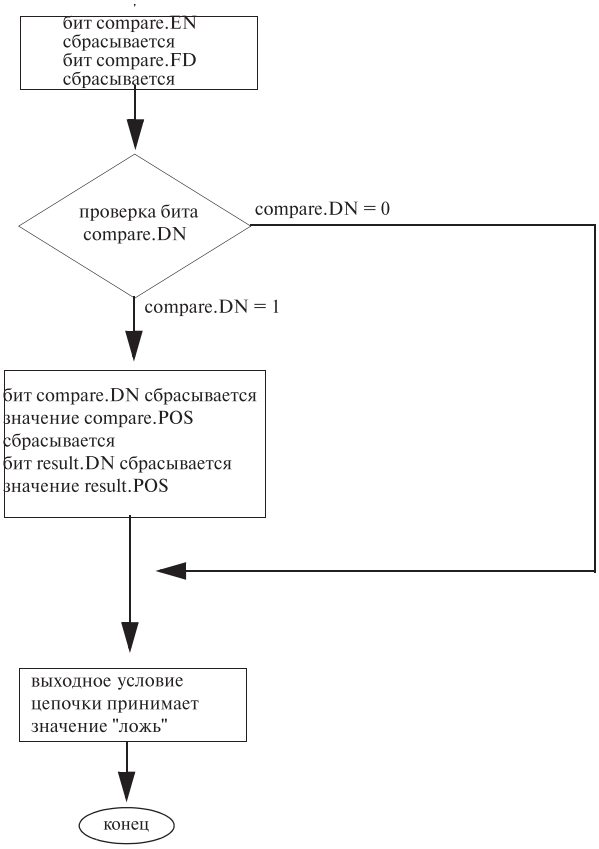
Основная ошибка произойдет, если:	Тип ошибки:	Код ошибки:
Result.POS > размера массива Result	4	20

Выполнение:

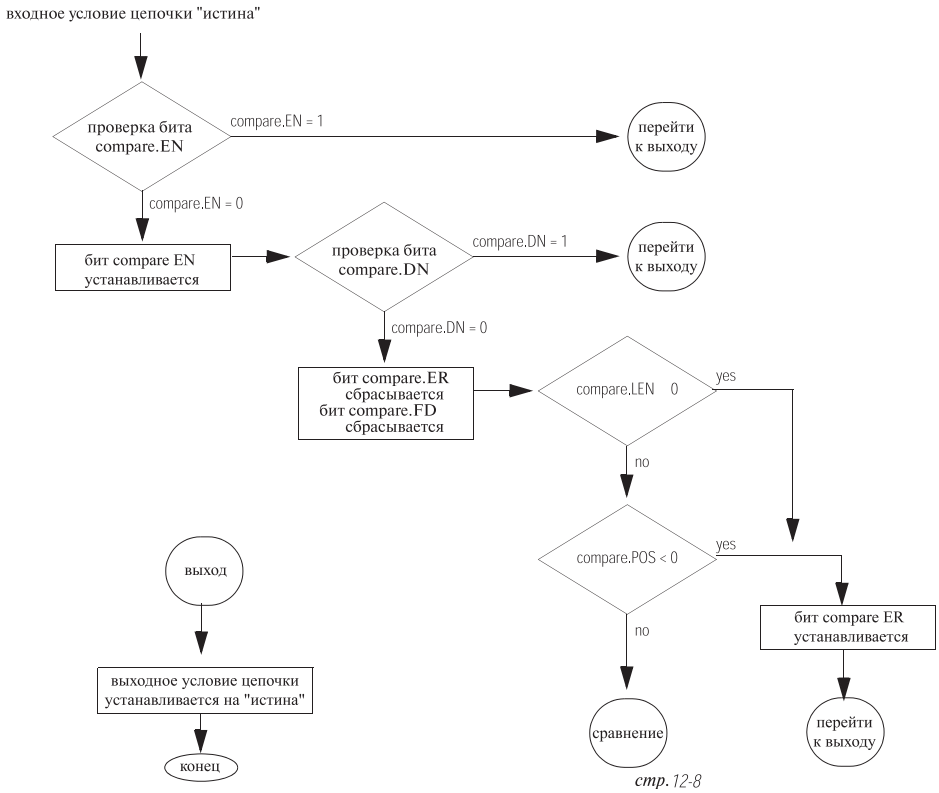
Условие:	**Действие релейной логики:**



Условие:	**Действие релейной логики:**

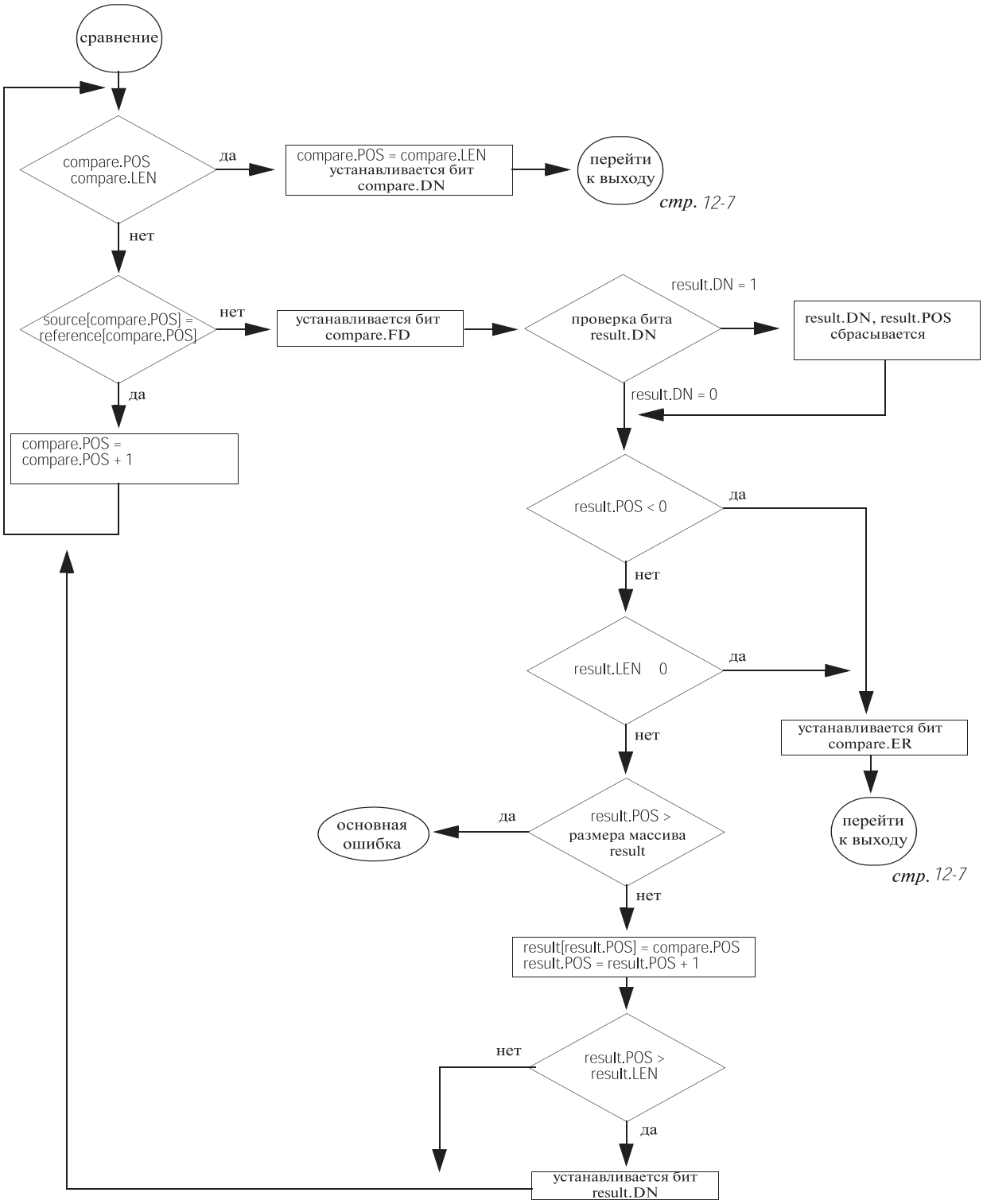


Условие:	**Действие релейной логики:**



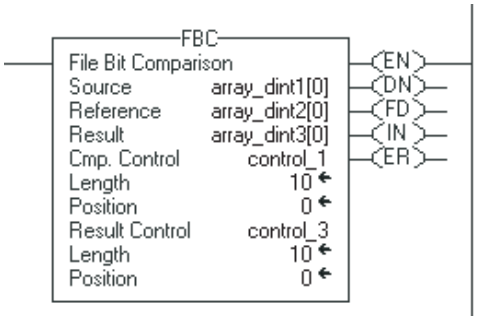
стр. 12-8

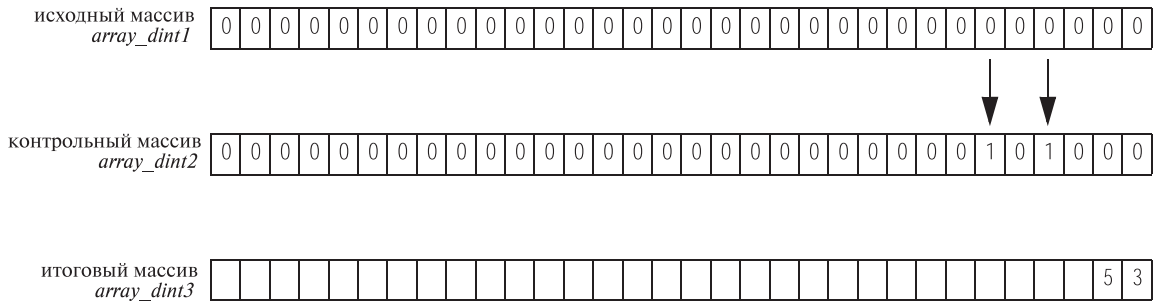
Условие:	**Действие релейной логики:**



постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример: Когда инструкция FBC разрешена, она сравнивает исходный массив array_dint1 с контрольным массивом array_dint2 и сохраняет позиции любых несоответствий в итоговом массиве array_dint3.

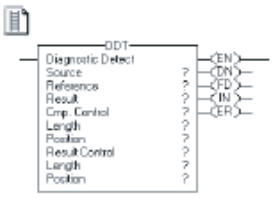




Diagnostic Detect (DDT) (Диагностика)

Инструкция DDT сравнивает биты в массиве Source (источнике) с битами в контрольном массиве Reference для обнаружения изменения состояния.

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Описание:
Source	DINT	тег массива	Массив для сравнения с массивом reference не используйте CONTROL.POS в нижнем регистре
Reference	DINT	тег массива	Массив для сравнения с массивом source не используйте CONTROL.POS в нижнем регистре
Result	DINT	тег массива	массив для сохранения результатов не используйте CONTROL.POS в нижнем регистре
Cmp control	CONTROL	структура	управляющая структура для сравнения
Length	DINT	непосредственный	количество битов для сравнения
Position	DINT	непосредственный	текущая позиция в массиве source исходное значение обычно 0
Result control	CONTROL	структура	управляющая структура для результатов
Length	DINT	непосредственный	количество позиций для хранения в результате
Position	DINT	непосредственный	текущая позиция в результирующем массиве исходное значение обычно 0

ВНИМАНИЕ

Используйте разные теги для управляющей структуры для сравнения и управляющей структуры для результатов. Использование одних и тех же тегов может привести к непредсказуемым операциям, способным повредить оборудование и/или нанести травму персоналу.

Структура COMPARE

Мнемоника:	Тип данных:	Описание:
.EN	BOOL	Бит разрешения указывает на то, что инструкция DDT разрешена.
.DN	BOOL	Бит выполнения устанавливается, когда инструкция DDT сравнивает последний бит в массивах Source и Reference.
.FD	BOOL	Бит обнаружения устанавливается каждый раз, когда инструкция FBC записывает несоответствие (по одному), или после записи всех несоответствий (все на сканирование).
.IN	BOOL	Бит запрещения указывает режим поиска DDT. 0 = сразу всех 1 = по одному
.ER	BOOL	Бит ошибки устанавливается, если .POS < 0, .LEN < 0, .POS<0 или .LEN < 0. Выполнение инструкции останавливается до тех пор, пока инструкция не сбросит бит .ER.
.LEN	DINT	Значение длины указывает количество битов для сравнения.
.POS	DINT	Значение позиции указывает текущий бит.

Структура RESULT

Мнемоника:	Тип данных:	Описание:
.DN	BOOL	Бит выполнения устанавливается, когда массив Result заполнен.
.LEN	DINT	Значение длины указывает количество позиций для хранения в массиве Result.
.POS	DINT	Значение позиции указывает текущую позицию в массиве Result.

Описание: Когда инструкция DDT разрешена, она сравнивает биты в массиве Source с битами в массиве Reference и при каждом несоответствии записывает номер бита в массив Result и изменяет значение бита в массиве Reference, чтобы он совпадал со значением бита в массиве Source.

Инструкция DDT производит операции с непрерывной областью памяти.

Различие между инструкциями DDT и FBC заключается в том, что каждый раз, как DDT находит несоответствие, она изменяет бит массива reference так, чтобы он совпадал с битом массива source. FBC не изменяет бит массива reference.

Выбор режима поиска

Если вы хотите обнаружить:	Выберите этот режим:
Одно несоответствие за один раз.	Установите бит .IN в структуре CONTROL. Каждый раз, когда входное условие цепочки меняет значение «ложь» на «истина», инструкция DDT ищет следующее несоответствие между массивами Source и Reference. Как только несоответствие найдено, инструкция устанавливает бит .FD, записывает позицию этого несоответствия и останавливает выполнение.
Все несоответствия.	Сбросьте бит.IN в структуре CONTROL. Каждый раз, когда входное условие цепочки меняет значение «ложь» на «истина», инструкция DDT ищет все несоответствия между массивами Source и Reference.

Арифметические флаги состояния:

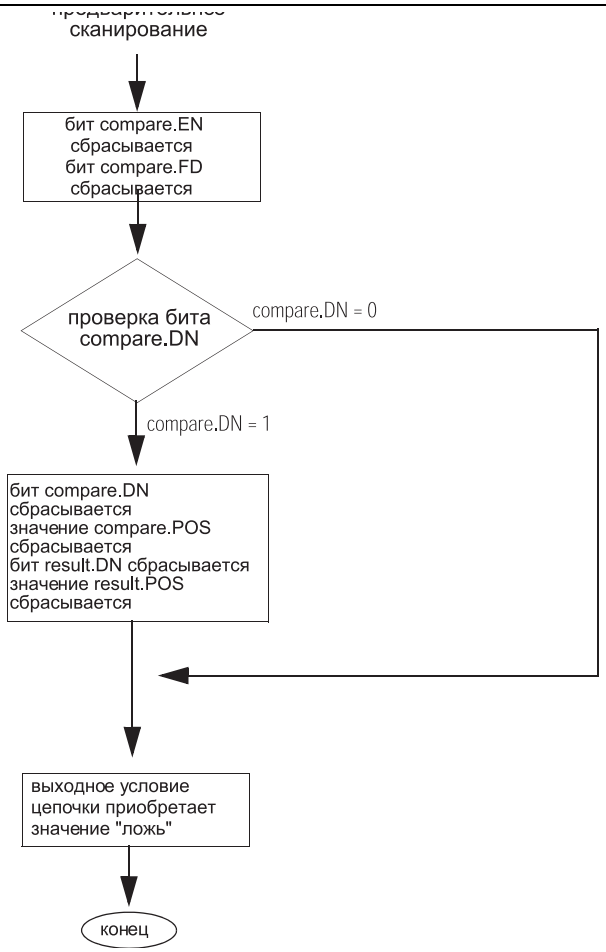
не затрагиваются

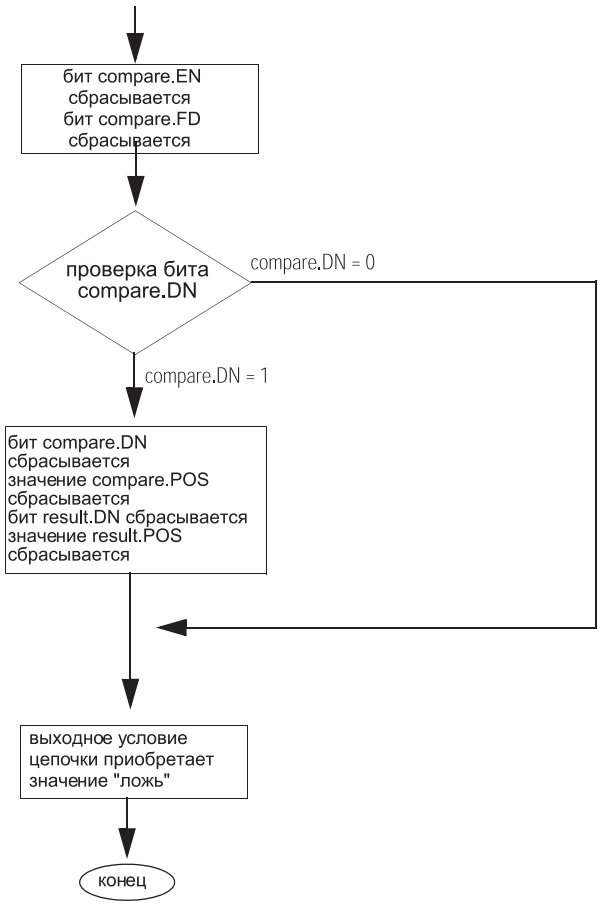
Условия ошибки:

Основная ошибка произойдет, если:	Тип ошибки:	Код ошибки:
Result.POS > размера массива Result	4	20

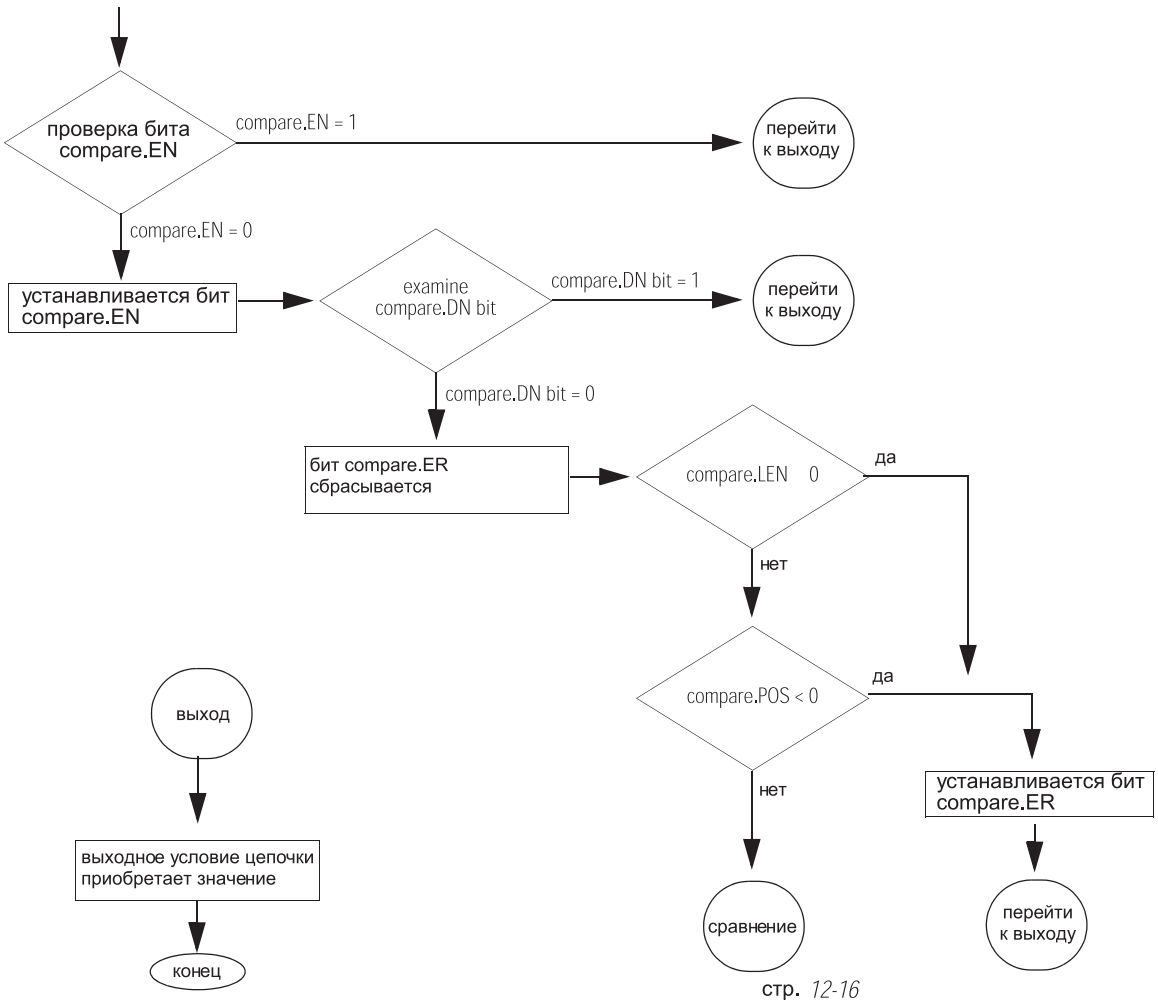
Выполнение:

Условие:	Действие релейной логики:



Условие:	**Действие релейной логики:**
входное условие цепочки "ложь"	

Условие:	**Действие релейной логики:**



стр. *12-16*

Условие:	**Действие релейной логики:**

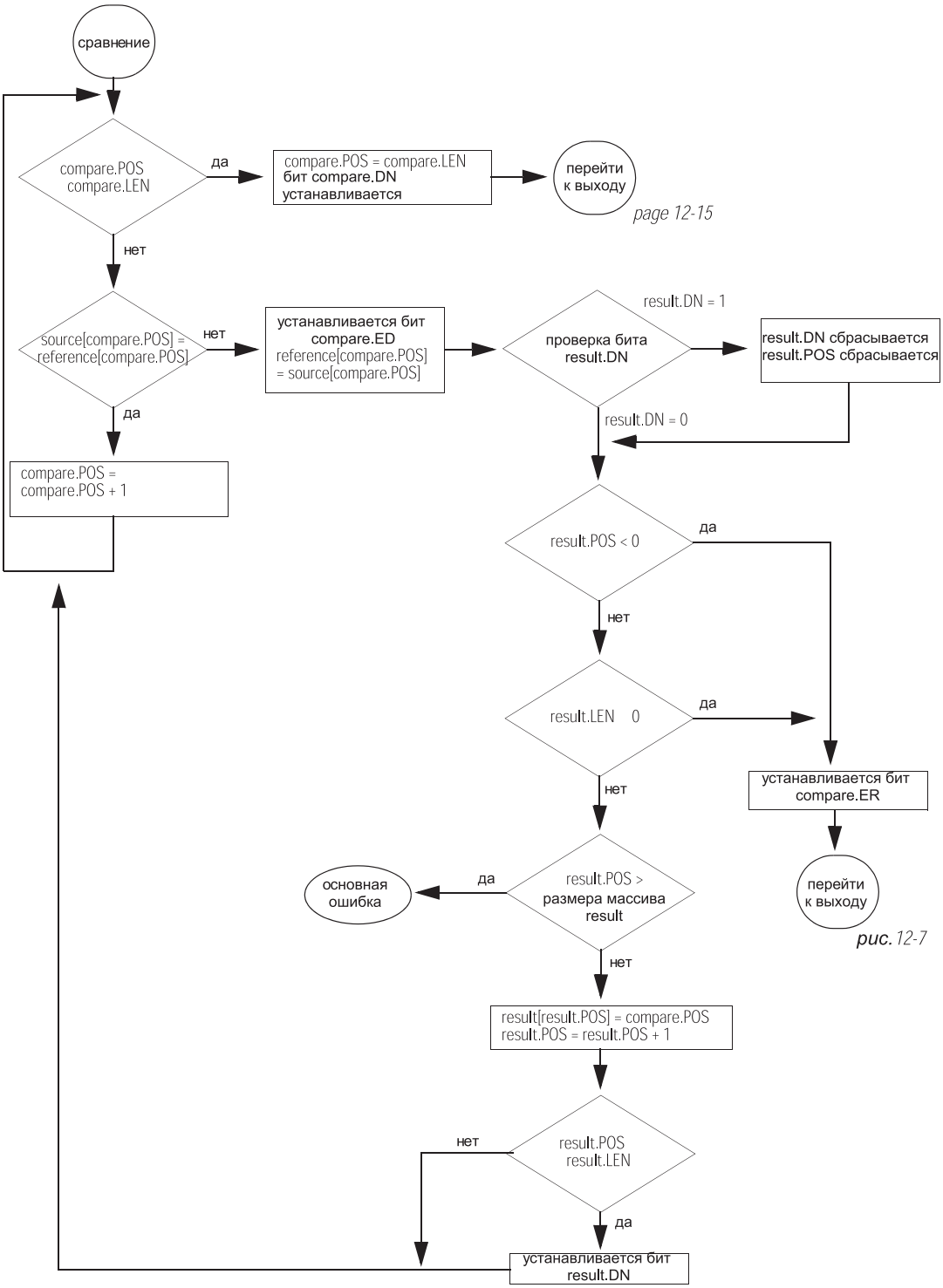
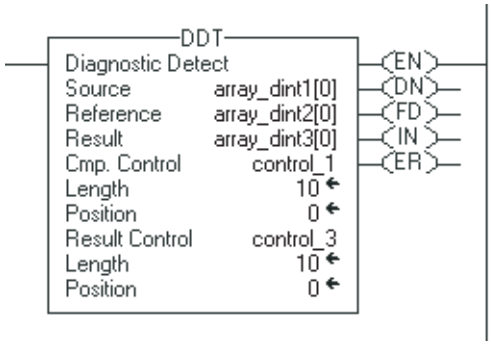
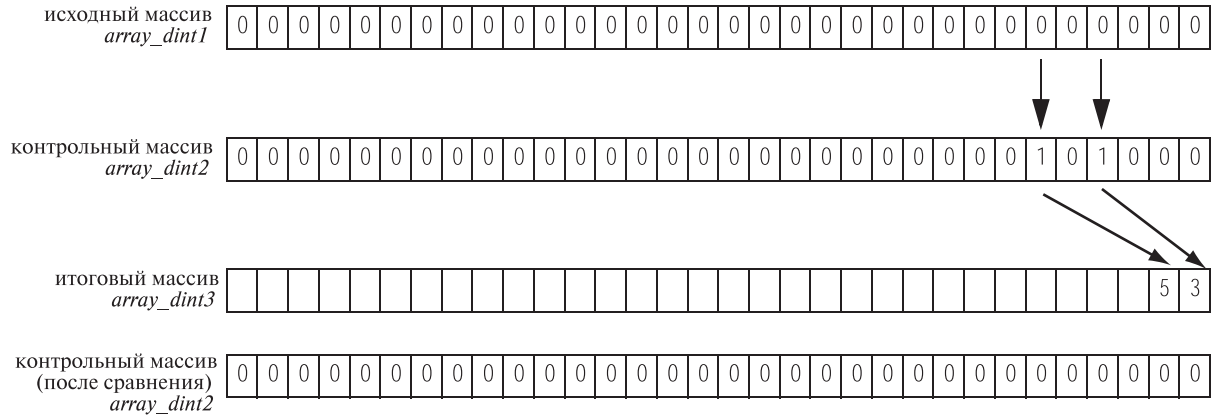


рис. 12-7

постсканирование	Выходное условие цепочки устанавливается на «ложь».

Пример: Когда инструкция DDT разрешена, она сравнивает исходный массив *array_dint1* с контрольным массивом *array_dint2* и сохраняет позиции любых несоответствий в итоговом массиве array_dint3. Контролер также изменяет несовпадающие биты в контрольном массиве *array_dint2* так, чтобы они совпадали со значениями в исходном массиве *array_dint1*.

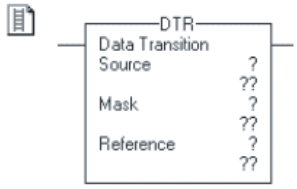




Data Transitional (DTR) (Изменение данных)

Инструкция DTR пропускает значение Source (источника) через Mask (маску) и сравнивает результат со значением Reference (контрольное значение).

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Описание:
Source	DINT	непосредственный тег	массив для сравнения с массивом reference
Mask	DINT	непосредственный тег	показывает, какие биты блокировать, а какие пропускать
Reference	DINT	тег	массив для сравнения с массивом source

Описание:

Инструкция DTR пропускает значение массива Source через маску Mask и сравнивает результат со значениями массива Reference. Также, инструкция DTR записывает значение Source, прошедшее маску, в Reference для последующего сравнения. Массив Source остается неизменным.

«1» в маске означает, что бит данных проходит. «0» в маске означает, что бит данных блокируется.

Если значение Source, прошедшее маску, отличается от Reference, выходное условие цепочки принимает значение «истина» для одного сканирования. Если значение Source, прошедшее маску, такое же как в Reference, выходное условие цепочки принимает значение «ложь».

ВНИМАНИЕ

Программирование этой инструкции в режиме он-лайн может быть опасным. Если значение Reference отличается от значения Source, выходное условие цепочки принимает значение «истина». Проявляйте осмотрительность, если вы вводите эту инструкцию, когда процессор находится в режиме Run или Remote Run.

Ввод непосредственных значений маски

Когда вы вводите значение маски, программное обеспечение установлено по умолчанию на восприятие десятичного значения. Если вы хотите ввести маску, используя другой формат, снабдите значение корректным префиксом

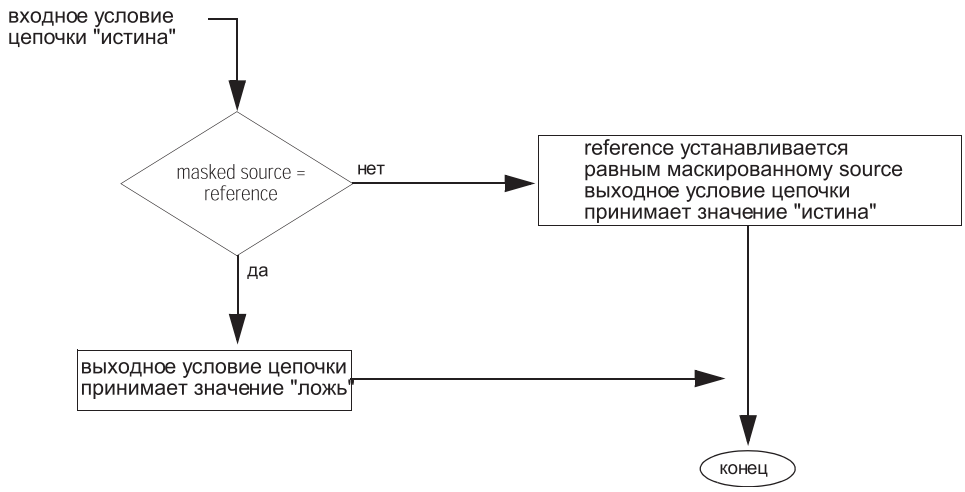
Префикс:	Описание:
16#	шестнадцатеричный, например, 16#0F0F
8#	восьмеричный, например, 8#16
2#	двоичный, например, 2#00110011

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

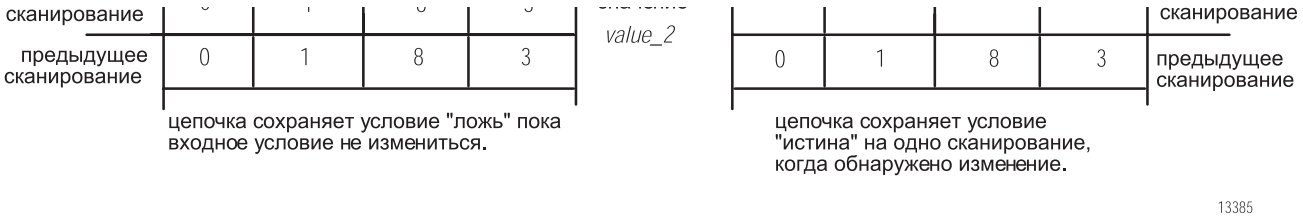
Выполнение:

Условие:	Действие релейной логики:
предварительное сканирование	Reference = Source AND Mask. Выходное условие цепочки устанавливается на «ложь»
входное условие цепочки «ложь»	Reference = Source AND Mask. Выходное условие цепочки устанавливается на «ложь»



постсканирование	Выходное условие цепочки устанавливается на “ложь”.

Пример: Когда инструкция DTR разрешена, она маскирует *value_1*. Если есть отличие между двумя значениями, выходное условие цепочки устанавливается на «истина».



Publication 1756-RM003G-EN-P - June 2003

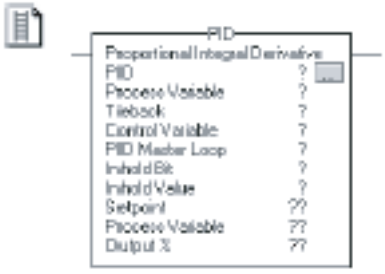
«0» в маске оставляет бит неизменным.

Proportional Integral Derivative (PID) (ПИД регулятор)

Инструкция PID управляет такими параметрами процесса как расход, давление, температура или уровень.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
PID	PID	структура	структура PID
Process variable	SINT INT DINT REAL	тег	значение, которым вы хотите управлять
Tieback	SINT INT DINT REAL	непосредственный тег	(необязательный) выход аппаратной станции (с ручным или автоматическим управлением), который обходит выход контроллера. Введите 0, если вы не хотите использовать этот параметр.
Control variable	SINT INT DINT REAL	тег	значение, которое передается в конечное управляющее устройство (клапан, демпфер и т.д.) Если вы используете зону нечувствительности, то параметр Control должен быть REAL, иначе он будет вынужден принимать значени 0, если ошибка будет в зоне нечувствительности.
PID master loop	PID	структура	(необязательный) тег PID для основного PID Если вы реализуете многоуровневое управление и этот PID является подчиненным циклом, введите имя основного цикла. Введите 0, если вы не хотите использовать этот параметр.
Inhold bit	BOOL	тег	(необязательный) текущее состояние бита inhold из аналогового канала выхода 1756 для обеспечения плавного перезапуска Введите 0, если вы не хотите использовать этот параметр.
Inhold value	SINT INT DINT REAL	тег	(необязательный) значение эха данных из аналогового канала выхода 1756 для обеспечения плавного перезапуска Введите 0, если вы не хотите использовать этот параметр.
Setpoint			выводит на экран текущее значение уставки
Process variable			выводит на экран текущее значение масштабированного значения Process Variable
Output %			выводит на экран текущее значение выхода в процентах

Структурированный текст

```
PID(PID,ProcessVariable,
Tieback,ControlVariable,
PIDMasterLoop,InholdBit,
InHoldValue);
```

Операнды такие же, как и операнды для инструкции PID в релейной логике. Однако вы задаете Setpoint, Process Variable и Output % путем организации доступа к членам .SP, .PV. и .OUT структуры PID, а не включения этих значений в список операндов.

Структура PID

Мнемоника:	Тип данных:	Описание:	
.CTL	DINT	Член .CTL предоставляет доступ к членам состояния (битам) в одном 32х битовом слове. Инструкция PID устанавливает биты 07 -15.	
		Этот бит:	**Является этим членом:**
		31	.EN
		30	.CT
		29	.CL
		28	.PVT
		27	.DOE
		26	.SWM
		25	.CA
		24	.MO
		23	.PE
		22	.NDF
		21	.NOBC
		20	.NOZC
		Этот бит:	**Является этим членом, который устанавливается инструкцией PID:**
		15	.INI
		14	.SPOR
		13	.OLL
		12	.OLH
		11	.EWD
		10	.DVNA
		09	.DVPA
		08	.PVLA
		07	.PVHA
.SP	REAL	уставка	
.KP	REAL	независимый	коэффициент передачи пропорционального регулятора (безразмерный)
		зависимый	коэффициент передачи регулятора (безразмерный)
.KI	REAL	независимый	коэффициент передачи интегрального регулятора (1/сек.)
		зависимый	время возврата (мин. на цикл)
KD	REAL	независимый	коэффициент передачи дифференциального регулятора (сек.)
		зависимый	масштаб времени (мин.)
.BIAS	REAL	предварение или смещение %	
.MAXS	REAL	максимальное значение масштабирования технических единиц	
.MINS	REAL	минимальное значение масштабирования технических единиц	

Мнемоника:	Тип данных:	Описание:
.DB	REAL	зона нечувствительности в технических единицах
.SO	REAL	нормировка %
.MAXO	REAL	максимальное ограничение выхода (% выхода)
.MINO	REAL	минимальное ограничение выхода (% выхода)
.UPD	REAL	время корректировки цикла (сек.)
.PV	REAL	масштабированное значение PV
.ERR	REAL	масштабированное значение ошибки
.OUT	REAL	выход %
.PVH	REAL p	верхний аварийный предел параметра процесса
.PVL	REAL	нижний аварийный предел параметра процесса
.DVP	REAL	положительное отклонение аварийного предела
.DVN	REAL	отрицательное отклонение аварийного предела
.PVDB	REAL	зона нечувствительности для параметра процесса
.DVDB	REAL	зона нечувствительности для аварийной сигнализации
.MAXI	REAL	максимальное значение PV (немасштабированный вход)
.MINI	REAL	минимальное значение PV (немасштабированный вход)
.TIE	REAL	значения для ручного контроля
.MAXCV	REAL	максимальное значение CV (соответствующее 100%)
.MINCV	REAL	минимальное значение CV (соответствующее 0%)
.MINTIE	REAL	минимальное значение tieback (соответствующее 100%)
.MAXTIE	REAL	максимальное значение tieback (соответствующее 0%)

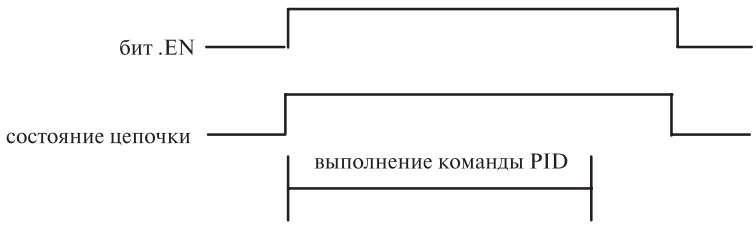
Мнемоника:	Тип данных:	Описание:	
.DATA	REAL[17]	Элемент .DATA сохраняет:	
		Элемент	**Описание:**
		.DATA[0]	интегральное накопление
		.DATA[1]	временное значение сглаживания производной
		.DATA[2]	предыдущее значение .PV
		.DATA[3]	предыдущее значение .ERR
		.DATA[4]	предыдущее точное значение .SP
		.DATA[5]	коэффициент масштабирования в процентах
		.DATA[6]	коэффициент масштабирования .PV
		.DATA[7]	коэффициент масштабирования производной
		.DATA[8]	предыдущее значение .KP
		.DATA[9]	предыдущее значение .KI
		.DATA[10]	предыдущее значение .KD
		.DATA[11]	зависимый коэффициент передачи .KP
		.DATA[12]	зависимый коэффициент передачи .KI
		.DATA[13]	зависимый коэффициент передачи .KD
		.DATA[14]	предыдущее значение .CV
		.DATA[15]	постоянная демасштабирования .CV
		.DATA[16]	постоянная демасштабирования tieback
.EN	BOOL	разрешенный	
.CT	BOOL	тип цикла (0=подчиненный; 1=основной)	
.CL	BOOL	многоуровневый (каскадный) цикл (0=нет; 1=да)	
.PVT	BOOL	отслеживание параметра процесса (0=нет; 1=да)	
.DOE	BOOL	производная (0=PV; 1=error)	
.SWM	BOOL	программный ручной режим (0=нет-авто; 1=да-ручной)	
.CA	BOOL	управляющее воздействие (0 означает E=SP-PV; 1 означает E=PV-SP)	
.MO	BOOL	режим станции (0=автоматический; 1=ручной)	
.PE	BOOL	уравнение PID (0=независимое; 1=зависимое)	
.NDF	BOOL	нет сглаживания производной (0=фильтр сглаживания производной разрешен; 1=фильтр сглаживания производной запрещен)	
.NOBC	BOOL	нет расчета обратного смещения (0=расчет обратного смещения разрешен; 1=расчет обратного смещения запрещен)	
.NOZC	BOOL	нет перехода через ноль полосы нечувствительности (0=есть переходит через ноль полосы нечувствительности; 1=нет перехода через ноль полосы нечувствительности)	
.INI	BOOL	PID инициализирована (0=нет; 1=да)	
.SPOR	BOOL	уставка вне диапазона (0=нет; 1=да)	
.OLL	BOOL	CV ниже минимального предела выхода (0=нет; 1=да)	
.OLH	BOOL	CV выше максимального предела выхода (0=нет; 1=да)	

Мнемоника:	Тип данных:	Описание:
EWD	BOOL	ошибка внутри полосы нечувствительности (0=нет; 1=да)
.DVNA	BOOL	предупреждение сигналом тревоги, что отклонение низкое (0=нет; 1=да)
.DVPA	BOOL	предупреждение сигналом тревоги, что отклонение высокое (0=нет; 1=да)
.PVLA	BOOL	предупреждение сигналом тревоги, что значение PV низкое (0=нет; 1=да)
.PVHA	BOOL	предупреждение сигналом тревоги, что значение PV высокое (0=нет; 1=да)

Описание:

Инструкция PID обычно получает параметр процесса (PV) от аналогового входного модуля и модулирует выход управляющей переменной (CV) в модуле аналогового выхода для того, чтобы сохранить параметр процесса на требуемой уставке.

Бит .EN указывает состояние выполнения. Бит .EN устанавливается, когда входное условие цепочки переходит от значения «ложь» к значению «истина». Бит .EN сбрасывается, когда входное условие цепочки приобретает значение «ложь». Инструкция PID не использует бит .DN. Инструкция PID исполняется на каждом сканировании, пока значение входного условия цепочки «истина».



Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

ВАЖНО! Эти ошибки были основными ошибками для контроллера PLC-5.

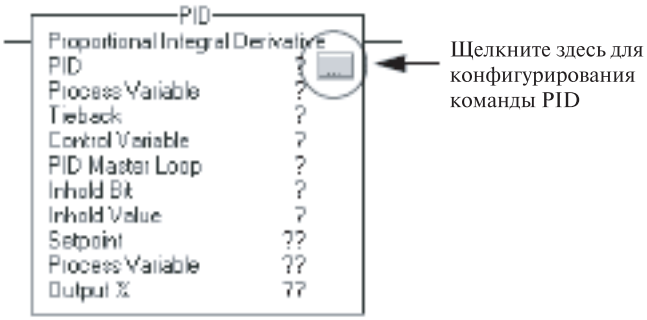
Неосновная ошибка произойдет, если:	Тип ошибки:	Код ошибки:
UPD . ≤ 0	4	35
уставка вне диапазона	4	35

Выполнение:

Условие:	Действие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается «ложь».	не применимо
входное условие цепочки «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция выполняет цикл PID.	Инструкция выполняет цикл PID.
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Конфигурирование инструкции PID

После того, как вы введете инструкцию PID и зададите структуру PID, используйте закладку конфигурирования для того, чтобы задать, как должна работать инструкция PID.



Задание настройки

Выберите закладку Tuning (настройка). Изменения вступят в силу, как только вы щелкните на другом поле, щелкните на кнопке OK, щелкните на кнопке Apply (применить) или нажмете Enter.

В этом поле:	**Задайте:**
Setpoint (SP)	Введите значение уставки (.SP).
Set output %	Введите установленный выход в процентах (.SO). В программном ручном режиме это значение используется для выхода. В автоматическом режиме этот значение выводится на экран в %.
Output bias	Введите смещения выхода в процентах (.BIAS).
Proportional gain (Kp)	Введите коэффициент передачи пропорционального регулятора (.KP). Для независимых коэффициентов это коэффициент передачи пропорционального регулятора (безразмерный). Для зависимых коэффициентов это коэффициент передачи контроллера. (безразмерный).
Integral gain (Ki)	Введите коэффициент передачи интегрального регулятора (.KI). Для независимых коэффициентов передачи это коэффициент передачи интегрального регулятора (1/сек.). Для зависимых коэффициентов передачи это время возврата (мин. на повторение).
Derivative time (Kd)	Введите коэффициент передачи дифференциального регулятора (.KD). Для независимых коэффициентов передачи это коэффициент передачи дифференциального регулятора (сек.). Для зависимых коэффициентов передачи это масштаб времени (мин.).
Manual mode	Выберите либо ручной режим (.MO), либо программный ручной режим (.SWM). Ручной режим подменяет программный ручной режим, если выбраны оба режима.

Задание конфигурации

Выберите закладку Configuration (конфигурация). Для того чтобы изменения вступили в силу, вы должны щелкнуть на OK или Apply.

В этом поле:	**Задайте:**
уравнение PID	Выберите независимые или зависимые коэффициенты передачи (.PE). Используйте независимые коэффициенты передачи, если вы хотите чтобы три коэффициента передачи (P, I и D) работали независимо. Используйте зависимые коэффициенты передачи, если вы хотите иметь полный коэффициент передачи регулятора, влияющий на все три члена (P, I и D).
управляющее воздействие	Выберите E=PV-SP или E=SP-PV для управляющего воздействия (.CA).
Производная	Выберите PV или error (ошибка) (.DOE). Используйте производную PV, чтобы устранить всплески выхода, появляющиеся в результате изменений уставки. Используйте производную ошибки для быстрых откликов на изменения уставки, когда алгоритм может допускать выбросы.
Вемя обновления цикла	Введите время обновления цикла (.UPD) для инструкции.
Верхний предел CV	Введите верхний предел для управляющей переменной (.MAXO).
Нижний предел CV	Введите нижний предел для управляющей переменной (.MINO).
Значение зоны нечувствительности	Введите значение зоны нечувствительности (.DB).
Нет сглаживания производной	Разрешите или запретите этот элемент (.NDF).

В этом поле:	Задайте:
No bias calculation	Разрешите или запретите этот элемент (.NOBC).
No zero crossing in deadband	Разрешите или запретите этот элемент (.NOZC).
PV tracking	Разрешите или запретите этот элемент (.PVT).
Cascade loop	Разрешите или запретите этот элемент (.CL).
Cascade type	Если каскадный цикл разрешен, выберите подчиненный или основной цикл (.CT).

Задание аварийных сигналов

Выберите закладку Select the Alarms (выбрать сигналы тревоги). Для того чтобы изменения вступили в силу, вы должны щелкнуть на OK или Apply.

В этом поле:	Задайте:
PV high	Введите верхнее значение сигнала тревоги PV (.PVH).
PV low	Введите нижнее значение сигнала тревоги PV (.PVL).
PV deadband	Введите значение зоны нечувствительности PV (.PVDB).
positive deviation	Введите значение положительного отклонения (.DVP).
negative deviation	Введите значение отрицательного отклонения (.DVN).
deviation deadband	Введите значение зоны нечувствительности отклонения для сигнала тревоги (.DVDB).

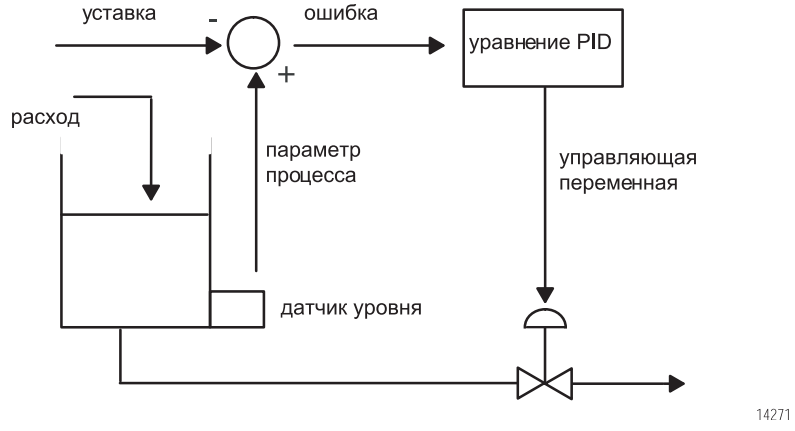
Задание масштабирования

Выберите закладку Scaling (масштабирование). Для того чтобы изменения вступили в силу, вы должны щелкнуть на OK или Apply.

В этом поле:	Задайте:
PV unscaled maximum	Введите максимум значения PV (.MAXI), который равен максимуму немасштабированного значения для PV, полученному от аналогового входного канала.
PV unscaled minimum	Введите минимум значения PV (.MINI), который равен минимуму немасштабированного значения для PV, полученному от аналогового входного канала.
PV engineering units maximum	Введите максимальное значение в технических единицах, соответствующие .MAXI (.MAXS)
PV engineering units minimum	Введите минимальное значение в технических единицах, соответствующие .MINI (.MINS)
CV maximum	Введите максимальное значение CV, соответствующее 100% (.MAXCV).
CV minimum	Введите минимальное значение CV, соответствующее 0% (.MINCV).
Tieback maximum	Введите максимальное значение tieback (.MAXTIE), которое равно максимуму немасштабированного значения для tieback, получаемому от аналогового входного канала.
Tieback minimum	Введите минимальное значение tieback (.MAXTIE), которое равно минимуму немасштабированного значения для tieback, получаемому от аналогового входного канала.
PID Initialized	Если вы изменяете коэффициент масштабирования в режиме Run, выключите этот элемент, чтобы реинициализировать внутренние значения коэффициентов масштабирования (.INI).

Использование инструкций PID

Закрытый цикл управления PID удерживает параметр процесса в нужной точке. Следующий рисунок иллюстрирует пример управления расходом/уровнем жидкости.



В приведенном выше примере значение уровня в баке сравнивается с уставкой. Если уровень выше, чем уставка, уравнение PID увеличивает управляющую переменную и заставляет открыться выходной клапан бака, что приводит к снижению уровня.

Уравнение, используемое в инструкции PID, является уравнением в позиционной форма с возможностью использования независимых или зависимых коэффициентов передачи. При использовании независимых коэффициентов передачи, коэффциценты передачи пропорционального, интегрального и дифференциального регуляторов влияют только на пропорциональные, интегральные и дифференциальные члены соответственно. При использовании зависимых коэффициентов, пропорциональный коэффициент заменяется коэффициентом передачи контроллера, который влияет на все три члена. Вы можете использовать оба типа уравнения для решения одной задачи управления. Два типа уравнения применяются лишь для того, чтобы вы могли использовать тот тип, с которым больше знакомы.

Коэффициент	Производная от	Уравнение
Зависимые коэффициенты (стандарт ISA)	ошибка (E)	$$CV = K_C\left[E + \frac{1}{T_i}\int_0^t E\,dt + T_d\frac{dE}{dt}\right] + BIAS$$
	параметр процесса (PV)	E = SP - PV $$CV = K_C\left[E + \frac{1}{T_i}\int_0^t E\,dt - T_d\frac{dPV}{dt}\right] + BIAS$$ E = PV - SP $$CV = K_C\left[E + \frac{1}{T_i}\int_0^t E\,dt + T_d\frac{dPV}{dt}\right] + BIAS$$
Независимые коэффициенты	ошибка (E)	$$CV = K_P E + K_i\int_0^t E\,dt + K_d\frac{dE}{dt} + BIAS$$
	параметр процесса (PV)	E = SP - PV $$CV = K_P E + K_i\int_0^t E\,dt - K_d\frac{dPV}{dt} + BIAS$$ E = PV - SP $$CV = K_P E + K_i\int_0^t E\,dt + K_d\frac{dPV}{dt} + BIAS$$

Где:

Перемен-ная:	**Описание:**
K_P	коэффициент передачи пропорционального регулятора (безразмерный) $K_p = K_c$ безразмерный
Ki	коэффициент передачи интегрального регулятора (сек.$^{-1}$) Для преобразования Ki (интегральный коэффициент) и Ti (время повторения), используйте: $K_i = \frac{K_c}{60 \cdot T_i}$
K_d	коэффициент передачи дифференциального регулятора (сек.) Для преобразования Kd (дифференциальный коэффициент) и Td (масштаб времени), используйте: $K_d = K_c(T_i)60$
K_C	коэффициент передачи контроллера (безразмерный)
T_i	время повторения (мин.повторение)
T_d	масштаб времени (мин.
SP	уставка
PV	параметр процесса
E	ошибка [(SP-PV) или (PV-SP)]
BIAS	предварение или смещение
CV	управляющая переменная
dt	время обновления цикла

Если вы не хотите использовать определенный член уравнения PID, просто установите его коэффициент равным нулю. Например, если вы не хотите иметь воздействие дифференциального регулятора, установите Kd или Td равным нулю.

Исключение повторений и плавный переход от ручного режима к автоматическому

Инструкция PID позволяет автоматически избегать ненужных повторений, предохраняя интегральный член от суммирования, когда выход достигает своего максимального или минимального значения, заданных с помощью .MAXO и .MINO. Накапливающийся член остается неизменным, пока выход CV не упадет ниже максимального предела или не поднимется выше минимального. Затем, автоматически восстановится нормальное накопление интеграла.

Инструкция PID поддерживает два ручных режима управления:

Ручной режим управления:	Описание:
Программный ручной режим (.SWM)	Также известен как настройка режима выхода. Позволяет пользователю устанавливать % выхода из программного обеспечения. Установленное значение выхода (.SO) используется как выход цикла. Обычно это значение вводится оператором при помощи интерфейса оператора.
Ручной режим (.MO)	Берет значение tieback в качестве входа и настраивает его внутренние переменные для того, чтобы сгенерировать такое же значение на выходе. Ввод значения tieback в инструкцию PID масштабируется в диапазоне 0-100% в соответствии со значениями .MINTIE и .MAXTIE и используется как выход цикла. Входное значение tieback обычно приходит с выхода аппаратной станции в ручном/автоматическом режиме, которое обходит выход контроллера. **Примечание:** Ручной режим имеет приоритет по отношению к программному ручному режиму, если установлены биты обоих режимов.

Инструкция PID также обеспечивает плавный переход от ручного режима к автоматическому и наоборот. Инструкция PID производит обратный расчет интегрального члена, который требуется для установки CV вместо значения .SO для программного ручного режима и входа tieback в ручном режиме. Таким образом, когда цикл включает автоматический режим, выход CV начинается со значения установленного выхода или со значения tieback, избегая «броска» выходного значения.

Инструкция PID также обеспечивает плавный переход от ручного к автоматическому режиму, даже если интегральный регулятор не используется (т.е. Ki= 0). В этом случае инструкция изменяет член .BIAS, чтобы установить CV вместо значений установленного выхода или tieback. Когда восстановится автоматическое управление, член .BIAS восстановит свое последнее значение. Вы можете отключить обратный расчет члена .BIAS, установив бит .NOBC в структуре данных PID. Помните, что если вы установили значение «истина» для .NOBC, инструкция PID больше не будет обеспечивать плавный переход с ручного на автоматический режим, если не используется интегральный регулятор.

Синхронизация инструкции PID

Инструкция PID и выборка параметра процесса требуют периодического обновления. Время обновления определяется физическим процессом, которым вы управляете. Для очень медленных циклов, таких, как температурный цикл, для получения хороших результатов при управлении достаточно иметь время обновления раз в секунду или даже реже. Для более быстрых циклов, таких, как давление или расход, может понадобиться скорость обновления - один раз за 250 миллисекунд. И только в редких случаях, таких, как управление напряжениями на перемоточной шпуле, может понадобиться обновление со скоростью один раз за 10 миллисекунд или быстрее.

Поскольку инструкция PID выполняет расчеты с учетом времени, вам необходимо синхронизировать выполнение этой инструкции с выборкой значений параметра процесса (PV).

Самый простой способ выполнения инструкции PID - размещение этой инструкции в какой-либо периодической задаче. Установите время обновления цикла (.UPD) равным периодичности задачи и убедитесь, что инструкция PID выполняется при каждом сканировании этой периодической задачи.

Релейная логика

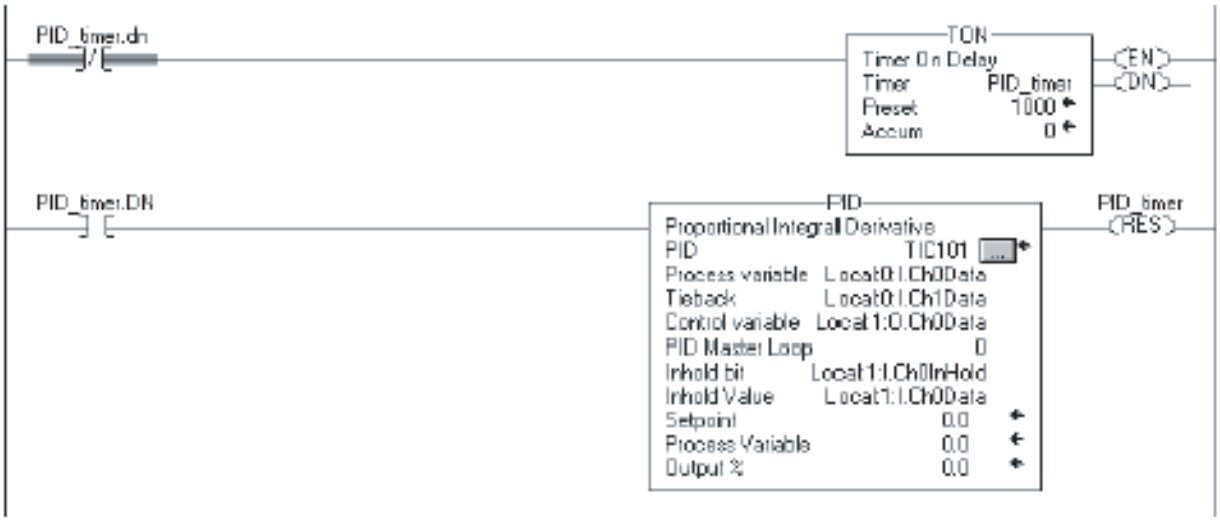
Структурированный текст

```
PID(TIC101,Local:0:I.Ch0Data,Local:0:I.Ch1Data,
    Local:1:O.Ch4Data,0,Local:1:I.Ch4InHold,
    Local:1:I.Ch4Data);
```

При использовании периодической задачи убедитесь, что аналоговый вход, используемый для параметра процесса, направляет данные в процессор со скоростью, значительно более высокой, чем периодичность задачи. В идеальном случае параметр процесса должен отсылаться в процессор в пять или десять раз быстрее скорости периодической задачи. Это минимизирует временную разницу между выборкой параметра процесса и выполнением цикла PID. Например, если цикл PID находится в задаче с периодичностью 250 миллисекунд, используйте время обновления цикла 250 мсек. (.UPD = .25) и сконфигурируйте аналоговый модуль ввода так, чтобы он выдавал данные каждые 25...50 мсек.

Другим в некоторых случаях менее точным способом выполнения инструкции PID является ее размещение в непрерывной задаче и использование бита таймера для включения выполнения инструкции PID.

Релейная логика



Структурированный текст

```
PID_timer.pre  :=  1000

TONR(PID_timer);

IF  PID_timer.DN  THEN

     PID(TIC101,Local:0:I.Ch0Data,Local:0:I.Ch1Dat,
     Local:1:O.Ch0Data,0,Local:1:I.Ch0InHold,
     Local:1:I.Ch0Data);

END_IF;
```

При использовании этого способа, время обновления цикла инструкции PID должно быть равно заранее заданному значению таймера. Как и в случае использования периодической задачи, вы должны настроить модуль аналогового ввода таким образом, чтобы он выдавал параметр процесса со значительно большей скоростью, чем время обновления цикла. Для циклов, у которых время обновления в несколько раз больше, чем время выполнения для вашей непрерывной задачи в наихудшем случае, вы должны использовать метод таймера для выполнения инструкции PID.

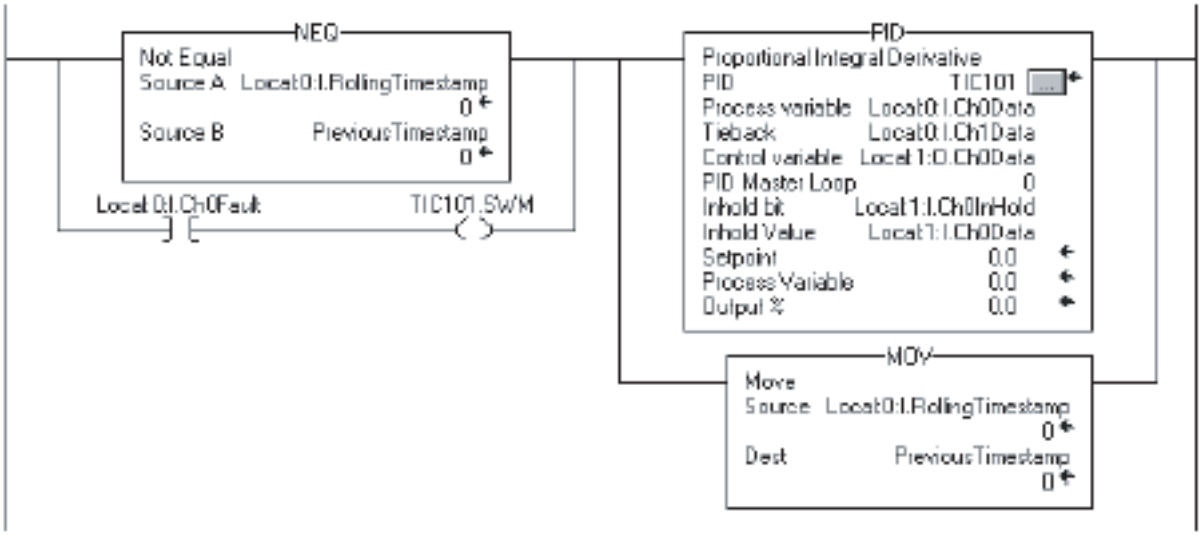
Наиболее точным способом выполнения инструкции PID является использование возможности выборки в режиме реального времени, предоставляемой модулями аналогового ввода 1756. Модуль аналогового ввода производит выборку в режиме реального времени, который вы сконфигурировали при его настройке. Когда истекает период выборки в режиме реального времени, модуль обновляет свои входные значения и временные метки прокручивания, (представленные членом .RollingTimestamp структуры ввода аналоговых данных).

Временные метки лежат в диапазоне 0-32767 миллисекунд. Следите за временной меткой. Ее смена означает, что уже получен новый отсчет параметра процесса. Каждый раз, когда меняется временная выборка, выполняйте один раз инструкцию PID. Поскольку выборка параметра процесса управляется модулем аналогового ввода, время выборки очень точно, и время обновления цикла, используемое инструкцией PID, должно быть установлено равным времени RTS модуля аналогового ввода.

Для того чтобы убедиться, что вы не пропустили отсчеты параметра процесса, выполните ваш алгоритм со скоростью, большей, чем время RTS. Например, если время RTS=250 мсек., вы должны поместить алгоритм PID в периодическую задачу, которая выполняется каждые 100 мсек., для того чтобы быть уверенным, что вы никогда не пропустите отсчет. Вы даже можете разместить алгоритм PID в непрерывной задаче, пока вы уверены, что этот алгоритм обновляется более часто, чем один раз в каждые 250 миллисекунд.

Пример использования метода RTS представлен ниже. Выполнение инструкции PID зависит от получения новых входных аналоговых данных. Если модуль аналогового ввода вышел из строя или удален, контроллер остановит получение меток времени, и выполнение цикла PID остановится. Вы должны следить за состоянием бита PV аналогового ввода, и если его состояние не правильно, принудительно переведите цикл в программный ручной режим и выполняйте цикл на каждом сканировании. Это позволит оператору вручную изменять выход цикла PID.

Релейная логика



Структурированный текст

```
IF  (Local:0:I.Ch0Fault)  THEN
    TIC101.SWM  [:=]  1;

ELSE
    TIC101.SWM  :=  0;

END_IF;

IF  (Local:0:I.RollingTimestamp<>PreviousTimestamp)  OR
    (Local:0:I.Ch0Fault)  THEN


    PreviousTimestamp  :=  Local:0:I.RollingTimestamp;


    PID(TIC101,Local:0:I.Ch0Data,
    Local:0:I.Ch1Data,Local:1:O.Ch0Data,0,
    Local:1:I.Ch0InHold,Local:1:I.Ch0Data);

END_IF;
```

Плавный повторный пуск

Инструкция PID может взаимодействовать с модулями аналогового выхода типа 1756 для обеспечения «безударного» перезапуска, когда контроллер переходит из режима программирования (Program) в режим выполнения (Run), или при включении питания контроллера.

Когда модуль аналогового выхода 1756 теряет связи с контроллером или датчиками, и контроллер находится в режиме программирования, модуль аналогового выхода присваивает выходу значение ошибки, которое вы задали при конфигурировании модуля. Затем, когда контроллер возвращается в режим выполнения (Run) , или восстанавливает связи с модулем аналогового выхода, инструкция PID может автоматически восстановить выходное значение управляющей переменной равное аналоговому выходу, используя бит Inhold и параметры Inhold Value в инструкции PID.

Чтобы задать плавный перезапуск:

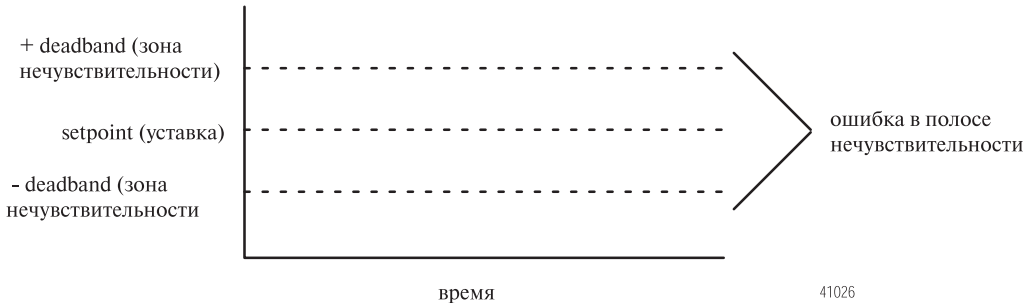
Сделайте это:	**Подробности:**
Сконфигурируйте тот канал модуля аналогового выхода 1756, который получает управляющую переменную от инструкции PID.	Выберите окошко метки “hold for initialization” для заданного модуля на странице свойств. Это сообщит аналоговому модулю, что когда контроллер вернется в режим выполнения Run или восстановит связи с модулем, этот модуль должен удерживать аналоговый выход на его текущем значении, пока значение, посланное от контроллера, не совпадет (в пределах 0.1% диапазона) с текущим значением, используемым выходным каналом. Выход контроллера будет стремиться к удерживаемому значению за счет члена .BIAS. Такое стремление аналогично автоматическому плавному переходу.
Введите битовый тег Inhold и тег Inhold Value в инструкции PID	Модуль аналогового выхода 1756 возвратит два значения для каждого канала в его структуру входных данных. Если бит состояния InHold (например, .Ch2InHold) имеет значение «истина», это указывает, что канал аналогового выхода сохраняет свое значение. Значение Data readback (например, .Ch2Data) показывает текущее значение выхода в технических единицах. Введите тег бита состояния InHold как битовый параметр InHold инструкции PID. Введите тег значения Data readback как параметр Inhold Value. Когда бит Inhold принимает значение «истина», инструкция PID перемещает Inhold Value в выход управляющей переменной Control и вновь инициализирует обеспечение плавного перезапуска при этом значении. Когда модуль аналогового выхода получает это значение от контроллера, он запрещает бит состояния InHold, что позволяет инструкции PID начать управление обычным образом.

Сглаживание производной

Расчет производной обрабатывается фильтром сглаживания производной. Этот цифровой фильтр сглаживания первого порядка помогает минимизировать большие «всплески» дифференциального члена, вызванные шумом в PV. Это сглаживание становится более «агрессивным» для больших значений коэффициента передачи дифференциального регулятора. Вы можете отключить сглаживание производной, если ваш процесс требует очень больших значений дифференциального коэффициента передачи (например, Kd > 10). Для того чтобы отключить сглаживание производной, выберите опцию “No derivative smoothing” (ет сглаживания производной) в закладке Configuration (конфигурация) или установите бит .NDF в структуре PID.

Настройка полосы нечувствительности

Настраиваемая полоса нечувствительности позволяет вам задать диапазон ошибки выше и ниже уставки, для которого выходное значение не изменяется, пока ошибка остается в этом диапазоне. Такая полоса нечувствительности позволяет вам контролировать, как точно параметр процесса совпадает с уставкой без изменения значения выхода. Полоса нечувствительности позволяет также минимизировать износ и повреждения конечного управляющего устройства.



Переход через ноль (Zero-crossing) является элементом управления, который позволяет инструкции использовать ошибку для расчетных нужд в то время, когда параметр процесса попал в полосу нечувствительности, и до того момента, пока он не пересечет значение уставки. Как только параметр процесса пересекает значение уставки (ошибка пересекает ноль и меняет знак) и до тех пор, пока параметр процесса остается внутри зоны нечувствительности, выходное значение не изменится.

Полоса нечувствительности простирается выше и ниже уставки на значение, которую вы зададите. Для отмены полосы нечувствительности введите ноль. Значение зоны нечувствительности имеет те же самые масштабированные единицы измерения, что и уставка. Вы можете использовать полосу нечувствительности без опции «переход через ноль», выбрав опцию “no zero crossing for deadband” в закладке Configuration (конфигурация) или установив бит .NOZC в структуре PID.

Если вы используете полосу нечувствительности, то управляющая переменная Control должна иметь тип REAL, иначе она примет значение 0, если ошибка будет находится внутри полосы нечувствительности.

Использование ограничений выходного значения

Вы можете настроить предельные значения для управляющего выхода (в процентах от значения выхода). Когда инструкция обнаружит, что значение выхода достигло предела, она установит бит сигнала тревоги и воспрепятствует тому, чтобы значение выхода пересекло верхний или нижний пределы.

Предварение или смещение значения выхода

Вы можете предварить возмущение от системы, передавая значение .BIAS в элемент feedforward/bias инструкции PID.

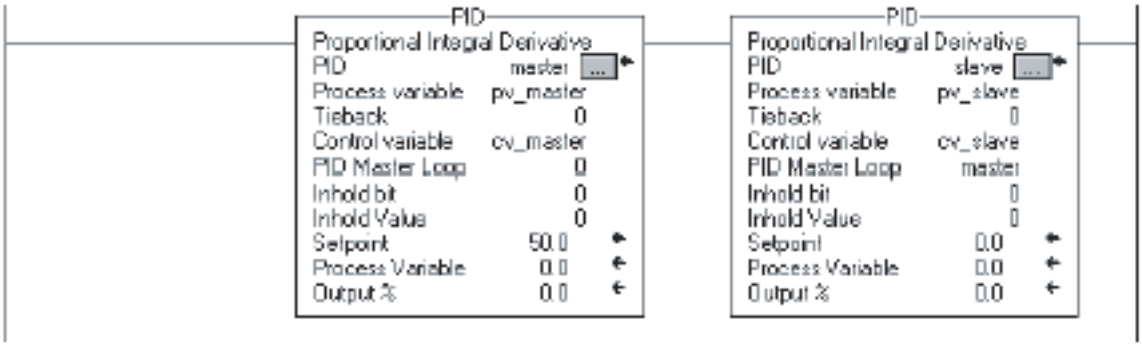
Значение предварения (feedforward) представляет собой возмущение, введенное в инструкцию PID до того, как возмущение будет иметь возможность изменить параметр процесса. Предварение часто используется для управления процессами с транспортным запаздыванием. Например, значение предварения для «холодной воды, наливаемой в теплую смесь», должно форсировать значение выхода быстрее, чем ожидание, пока параметр процесса изменит свое значение в результате смешивания.

Значение смещения обычно используется тогда, когда отсутствует интегральный регулятор. В этом случае значение смещения может быть настроено для удержания значения выхода в диапазоне, необходимом для удержания PV вблизи уставки.

Организация многоуровневых циклов

PID организует работу двух каскадных циклов, приписывая значение выхода (в процентах) основного цикла уставке подчиненного цикла. Подчиненный цикл автоматически преобразует значение выхода основного цикла в соответствующие технические единицы для уставки на основе значений .MAXS и .MINS для этого подчиненного цикла.

Релейная логика



Структурированный текст

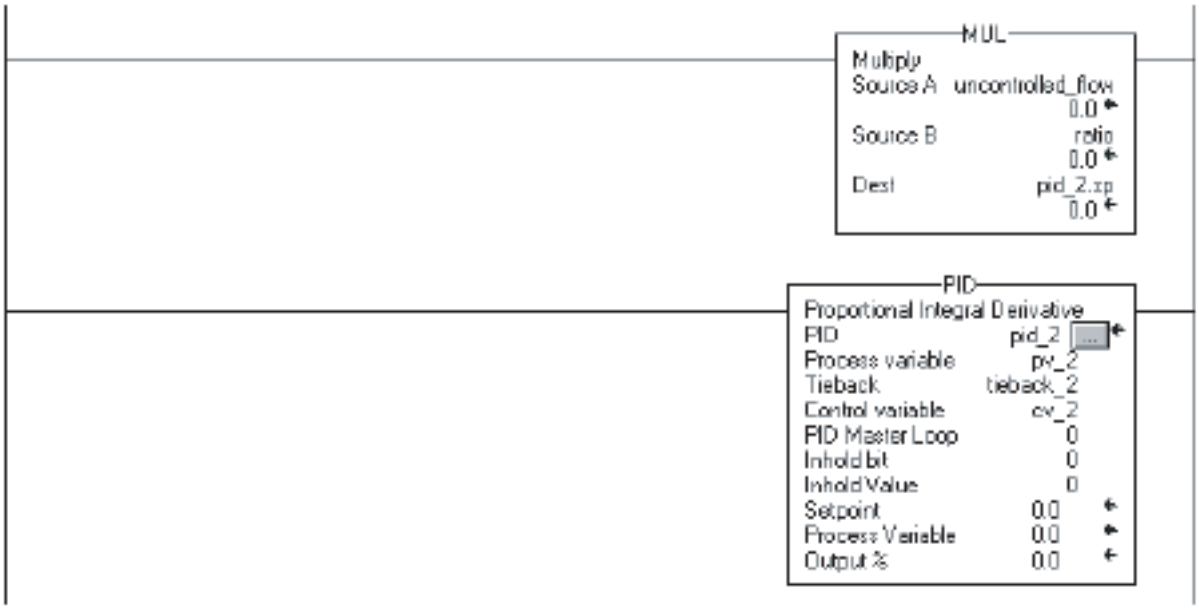
```
PID(master,pv_master,0,cv_master,0,0,0);

PID  (slave,pv_slave,0,cv_slave,master,0,0);
```

Контроль отношения

Вы можете сохранять отношение двух значений, используя следующие параметры:

- нерегламентированное значение (uncontrolled value)
- регламентированное значение (controlled value) (результирующая уставка для использования инструкцией PID)
- отношение между двумя этими значениями

Релейная логика



Структурированный текст

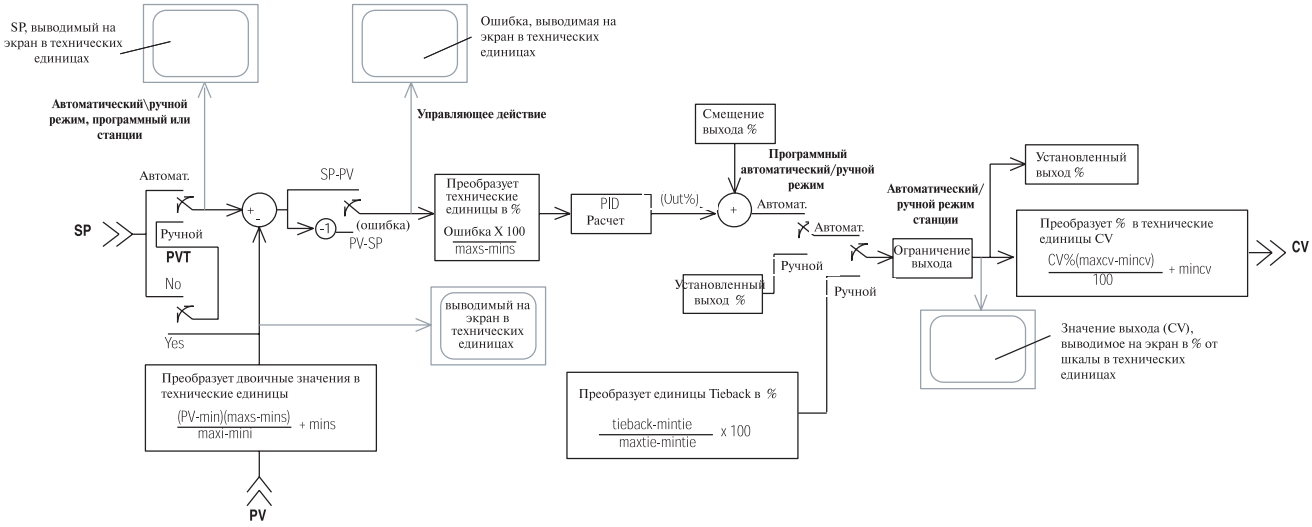
```
pid_2.sp := uncontrolled_flow * ratio
PID(pid_2,pv_2,tieback_2,cv_2,0,0,0);
```

Для этого параметра умножения	Введите это значение:
destination (приемник)	регламентированное значение
source A (источник A)	нерегламентированное значение
source B (источник B)	отношение

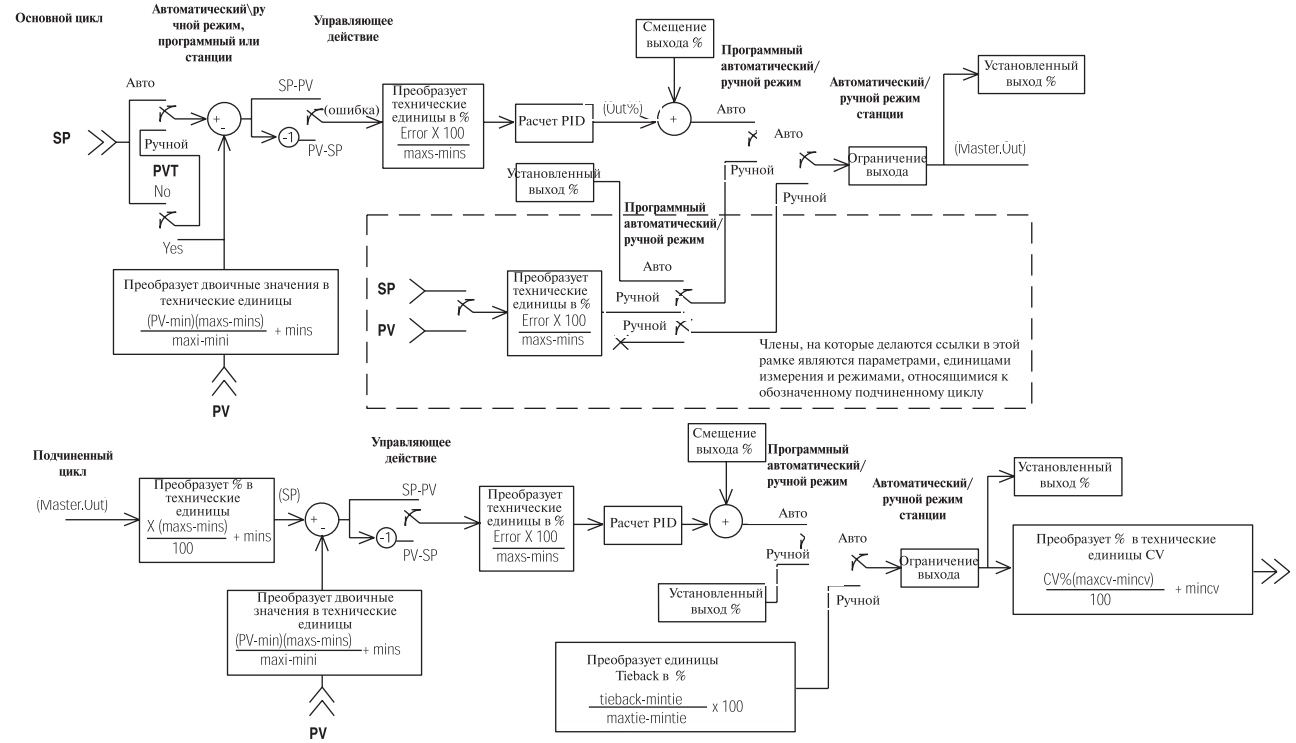
Теория PID

Следующий рисунок демонстрирует работу инструкции PID.

Процесс PID



Работа инструкции PID с основным и подчиненным циклами



Примечания:

Тригонометрические инструкции (SIN, COS, TAN, ASN, ASIN, ACS, ACOS, ATN, ATAN)

Введение

Тригонометрические инструкции производят математические операции с использованием тригонометрических функций.

Если вы хотите:	Используйте эту инструкцию:	Имеющуюся в этих языках:	См. стр.
Взять синус какого-либо значения.	SIN	релейная логика структурированный текст функциональный блок	13-2
Взять косинус какого-либо значения.	COS	релейная логика структурированный текст функциональный блок	13-5
Взять тангенс какого-либо значения.	TAN	релейная логика структурированный текст функциональный блок	13-8
Взять арксинус какого-либо значения.	ASN ASIN(1)	релейная логика структурированный текст функциональный блок	13-11
Взять арккосинус какого-либо значения.	ACS ACOS(1)	релейная логика структурированный текст функциональный блок	13-14
Взять арктангенс какого-либо значения.	ATN ATAN(1)	релейная логика структурированный текст функциональный блок	13-17

(1) Только структурированный текст

Вы можете смешивать типы данных, но это может привести к потере точности и ошибке округления, и для выполнения инструкции потребуется больше времени. Проверьте бит S:V, чтобы убедиться, отброшена ли у полученного результата дробная часть.

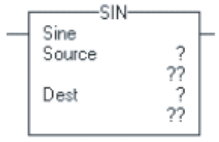
Для инструкций релейной логики, **жирный** шрифт типов данных означает, что это оптимальные типы данных. Инструкция выполняется быстрее и требует меньший объем памяти, если все операнды инструкции используют один и тот же оптимальный тип данных, обычно DINT или REAL.

Sine (SIN) (Синус)

Инструкция SIN берет синус Source (источника) (в радианах) и сохраняет результат в Destination (приемнике).

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Source	SINT	непосредственный	ищется синус этого значения
	INT	тег	
	DINT		
	REAL		
Destination	SINT	тег	тег для хранения результата
	INT		
	DINT		
	REAL		

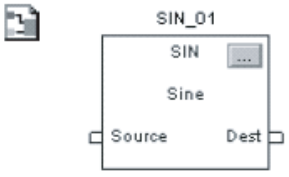
Структурированный текст

```
dest := SIN(source);
```

Используйте SIN в качестве функции. Эта функция рассчитывает синус значения *source* и сохраняет результат в *dest*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег SIN	FBD_MATH_ADVANCED	структура	структура SIN

Структура FBD_MATH_ADVANCED

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Входное значение для математической инструкции. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание:

Значение Source должно быть больше или равно -205887.4 ($-2\pi x2^{15}$) и меньше или равно 205887.4 ($2\pi x2^{15}$). Значение результата в Destination всегда больше или равно -1 и меньше или равно 1.

Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют отсутствуют

Выполнение:

Релейная логика:

Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Контроллер рассчитывает синус от Source и помещает результат в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Вычисление синуса *value* и помещение результата в *result.*

Релейная логика

Структурированный текст

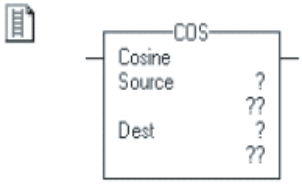
```
result := SIN(value);
```

Функциональный блок

Cosine (COS) (Косинус)

Инструкция COS берет косинус Source (источника) (в радианах) и сохраняет результат в Destination (приемнике).

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Описание:
Source	SINT INT DINT REAL	непосредственный тег	ищется косинус этого значения
Destination	SINT INT DINT REAL	тег	тег для хранения результата

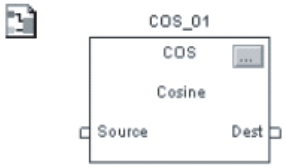
Структурированный текст

```
dest := COS(source);
```

Используйте COS в качестве функции. Эта функция рассчитывает косинус *source* и сохраняет результат в *dest*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег COS	FBD_MATH_ADVANCED	структура	структура COS

Структура FBD_MATH_ADVANCED

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Входное значение для математической инструкции. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание:

Значение Source должно быть больше или равно -205887.4 ($-2\pi x2^{15}$) и меньше или равно 205887.4 ($2\pi x2^{15}$). Значение результата в Destination всегда больше или равно -1 и меньше или равно 1.

Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика:

Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Контроллер рассчитывает косинус от Source и помещает результат в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

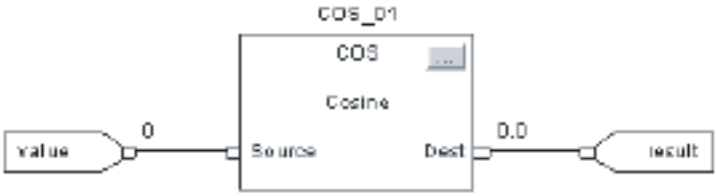
Пример: Вычисление косинуса *value* и помещение результата в *result*.

Релейная логика

Структурированный текст

```
result := SIN(value);
```

Функциональный блок

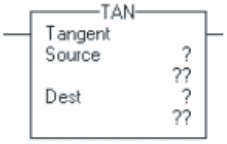


Tangent (TAN) (Тангенс)

Инструкция TAN берет тангенс Source (источника) (в радианах) и сохраняет результат в Destination (приемнике).

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Source	SINT INT DINT REAL	непосредственный тег	ищется тангенс этого значения
Destination	SINT INT DINT REAL	тег	тег для хранения результата

Структурированный текст

```
dest := TAN(source);
```

Используйте TAN в качестве функции. Эта функция рассчитывает тангенс *source* и сохраняет результат в *dest*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег TAN	FBD_MATH_ADVANCED	структура	структура TAN

Структура FBD_MATH_ADVANCED

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Входное значение для математической инструкции. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание:

Значение Source должно быть больше или равно -102943.7 ($-2\pi x2^{14}$) и меньше или равно 102943.7 ($2\pi x2^{14}$).

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика:

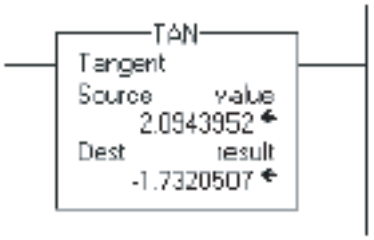
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Контроллер рассчитывает тангенс Source и помещает результат в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Вычисление тангенса *value* и помещение результата в *result*.

Релейная логика



Структурированный текст

```
result := TAN(value);
```

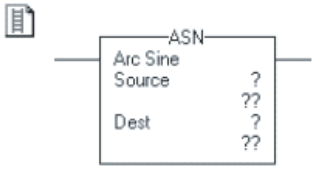
Функциональный блок

Arc Sine (ASN) (Арксинус)

Инструкция ASN берет арксинус Source (источника) (в радианах) и сохраняет результат в Destination (приемнике) (в радианах).

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Source	SINT INT DINT REAL	непосредственный тег	ищется арксинус этого значения
Destination	SINT INT DINT REAL	тег	тег для хранения результата

Структурированный текст

```
dest := ASIN(source);
```

Используйте ASIN в качестве функции. Эта функция рассчитывает арксинус *source* и сохраняет результат в *dest*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег ASN	FBD_MATH_ADVANCED	структура	структура ASN

Структура FBD_MATH_ADVANCED

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Входное значение для математической инструкции. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание:

Значение Source должно быть больше или равно -1 и меньше или равно 1. Значение результата в Destination всегда больше или равно $-\pi/2$ и меньше или равно $\pi/2$ (где π=3.141593).

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика:

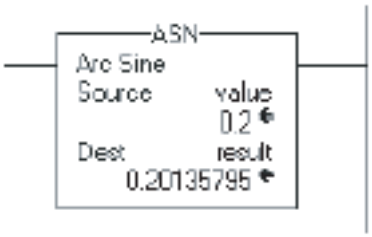
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Контроллер рассчитывает арксинус от Source и помещает результат в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Вычисление арксинуса *value* и помещение результата в *result*.

Релейная логика



Структурированный текст

```
result := ASIN(value);
```

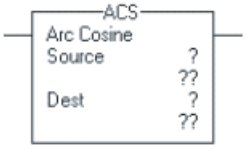
Функциональный блок

Arc Cosine (ACS) (Арккосинус)

Инструкция ACS берет арккосинус Source (источника) (в радианах) и сохраняет результат в Destination (приемнике) (в радианах).

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Source	SINT INT DINT **REAL**	непосредственный тег	ищется арккосинус этого значения
Destination	SINT INT DINT **REAL**	тег	тег для хранения результата

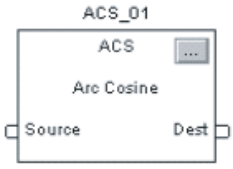
Структурированный текст

```
dest := ACOS(source);
```

Используйте ACOS в качестве функции. Эта функция рассчитывает арккосинус *source* и сохраняет результат в *dest*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег ACS	FBD_MATH_ADVANCED	структура	структура ACS

Структура FBD_MATH_ADVANCED

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Входное значение для математической инструкции. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание:

Значение Source должно быть больше или равно -1 и меньше или равно 1. Значение результата в Destination всегда больше или равно 0 и меньше или равно π (где π=3.141593).

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика:

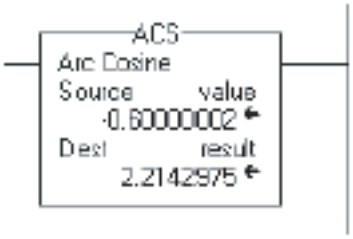
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Контроллер рассчитывает арккосинус от Source и помещает результат в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Вычисление арккосинуса *value* и помещение результата в *result*.

Релейная логика



Структурированный текст

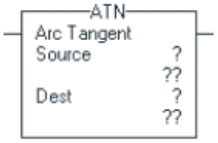
```
result := ACOS(value);
```

Функциональный блок

Arc Tangent (ATN) (Арктангенс)

Инструкция ATN берет арктангенс Source (источника) и сохраняет результат в Destination (приемнике) (в радианах).

Операнды:



Релейная логика

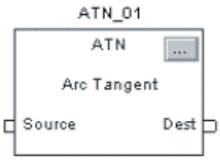
Операнд:	Тип:	Формат:	Описание:
Source	SINT INT DINT **REAL**	непосредственный тег	ищется арктангенс этого значения
Destination	SINT INT DINT **REAL**	тег	тег для хранения результата

```
dest := ATAN(source);
```

Структурированный текст

Используйте ATAN в качестве функции. Эта функция рассчитывает арктангенс *source* и сохраняет результат в *dest.*

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.



Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег ATN	FBD_MATH_ADVANCED	структура	структура ATN

Структура FBD_MATH_ADVANCED

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Входное значение для математической инструкции. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	Тип данных:	Описание:
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание:

Значение результата в Destination всегда больше или равно $-\pi/2$ и меньше или равно $\pi/2$ (где π=3.141593).

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика:

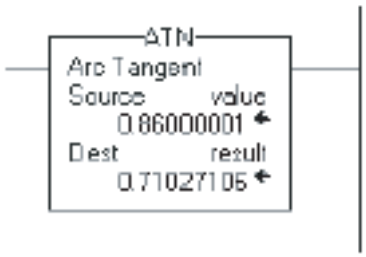
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Контроллер рассчитывает арктангенс от Source и помещает результат в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Вычисление арктангенса *value* и помещение результата в *result*.

Релейная логика



Структурированный текст

```
result := ATAN(value);
```

Функциональный блок

Примечания:

Научные математические инструкции (LN, LOG, XPY)

Введение

К сложным математическим инструкциям относятся:

Если вы хотите:	Используйте эту инструкцию:	Имеющуюся в этих языках:	См. стр.
Взять натуральный логарифм значения.	LN	релейная логика структурированный текст функциональный блок	14-2
Взять десятичный логарифм значения.	LOG	релейная логика структурированный текст функциональный блок	14-4
Возвести значение в степень.	XPY	релейная логика структурированный текст(1) функциональный блок	14-6

(1) Не существует эквивалентной инструкции для структурированного текста. Используйте оператор в выражении.

Вы можете смешивать типы данных, но это может привести к потере точности и ошибке округления, и для выполнения инструкции потребуется больше времени. Проверьте бит S:V, чтобы убедиться, отброшена ли у полученного результата дробная часть.

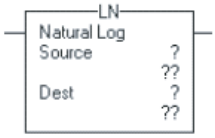
Для инструкций релейной логики, **жирный** шрифт типов данных означает, что это оптимальные типы данных. Инструкция выполняется быстрее и требует меньший объем памяти, если все операнды инструкции используют один и тот же оптимальный тип данных, обычно DINT или REAL.

Natural Log (LN) (Натуральный логарифм)

Инструкция LN берет натуральный логарифм Source (источника) и сохраняет результат в Destination (приемнике).

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Source	SINT INT DINT **REAL**	непосредственный тег	ищется натуральный логарифм этого значения
Destination	SINT INT DINT **REAL**	тег	тег для хранения результата

Структурированный текст

```
dest := LN(source);
```

Используйте LN в качестве функции. Эта функция рассчитывает натуральный логарифм значения *source* и сохраняет результат в *dest*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег LN	FBD_MATH_ADVANCED	структура	структура LN

Структура FBD_MATH_ADVANCED

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Входное значение для математической инструкции. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание:

Значение Source должно быть больше нуля, в противном случае будет установлен бит переполнения (S:V). Значение результата Destination больше или равно -87.33655 или меньше или равно 88.72284.

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Релейная логика:

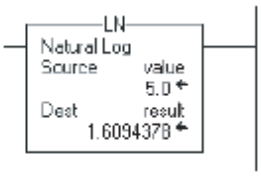
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Контроллер рассчитывает натуральный логарифм Source и помещает результат в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Вычисление натурального логарифма *value* и помещение результата в *result.*

Релейная логика



Структурированный текст

```
result := LN(value);
```

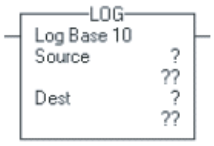
Функциональный блок

Log Base 10 (LOG) (Десятичный логарифм)

Инструкция LOG берет десятичный логарифм Source (источника) и сохраняет результат в Destination (приемнике).

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Source	SINT INT DINT **REAL**	непосредственный тег	ищется десятичный логарифм этого значения
Destination	SINT INT DINT **REAL**	тег	тег для хранения результата

Структурированный текст

```
dest := LOG(source);
```

Используйте LOG в качестве функции. Эта функция рассчитывает десятичный логарифм значения *source* и сохраняет результат в *dest*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег LOG	FBD_MATH_ADVANCED	структура	структура LOG

Структура FBD_MATH_ADVANCED

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Входное значение для математической инструкции. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание:

Значение Source должно быть больше нуля, в противном случае будет установлен бит переполнения (S:V). Значение результата Destination больше или равно -37.92978 или меньше или равно 38.53184.

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика:

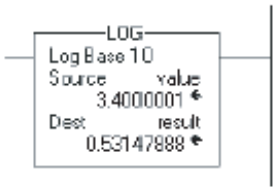
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Контроллер рассчитывает десятичный логарифм Source и помещает результат в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Вычисление десятичного логарифма *value* и помещение результата в *result*.

Релейная логика



Структурированный текст

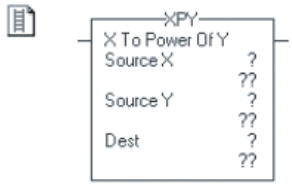
```
result := LOG(value);
```

Функциональный блок

X to the Power of Y (XPY) (Возведение X в степень Y)

Инструкция XPY возводит Source A (источник A) (X) в степень Source B (источник B) (Y) и сохраняет результат в Destination (приемнике).

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Описание:
Source X	SINT	непосредственный	основание
	INT	тег	
	DINT		
	REAL		
Source Y	SINT	непосредственный	показатель
	INT	тег	
	DINT		
	REAL		
Destination	SINT	тег	тег для хранения результата
	INT		
	DINT		
	REAL		

Структурированный текст

```
dest := sourceX ** sourceY;
```

Используйте два следующих друг за другом знака умножения “**” в качестве оператора в выражении. Это выражение возводит *sourceX* в степень *sourceY* и сохраняет результат в *dest*.

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег XPY	FBD_MATH	структура	структура XPY

Структура FBD_MATH

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source X	REAL	Основание. Допустимое значение = любое значение с плавающей точкой
Source Y	REAL	Показатель. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат математической инструкции. Арифметические флаги состояния устанавливаются для этого выхода.

Описание: Если Source X имеет отрицательное значение, SourceY должен быть целым значением, в противном случае будет иметь место неосновная ошибка.

Инструкция XPY использует алгоритм: Destination = X**Y

Контроллер определяет $x^0=1$ и $0^x=0$.

Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

Условия ошибки:

Неосновная ошибка произойдет, если	Тип ошибки:	Код ошибки:
Source X имеет отрицательное значение, а Source Y не является целым числом	4	4

Выполнение:

Релейная логика:

Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Контроллер возводит Source X в степень Source Y и помещает результат в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

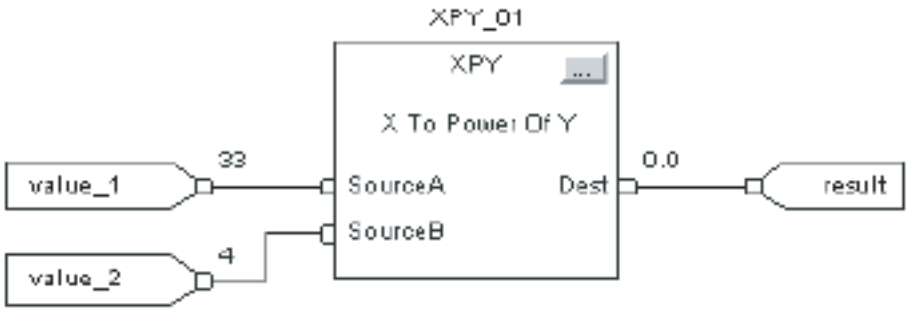
Пример: Инструкция XPY возводит *value_1* в степень *value_2* и помещает результат в *result*.

Релейная логика

Структурированный текст

```
result := (value_1 ** value_2);
```

Функциональный блок



Математические инструкции преобразования (DEG, RAD, TOD, FRD, TRN, TRUNC)

Введение

Математические инструкции преобразования преобразуют значения.

Если вы хотите:	Используйте эту инструкцию:	Имеющуюся в этих языках:	См. стр.
Преобразовать радианы в градусы.	DEG	релейная логика структурированный текст функциональный блок	15-2
Преобразовать градусы в радианы.	RAD	релейная логика структурированный текст функциональный блок	15-4
Преобразовать целые числа в двоично-десятичные (BCD).	TOD	релейная логика функциональный блок	15-6
Преобразовать двоично-десятичные числа в целые.	FRD	релейная логика функциональный блок	15-9
Отбросить дробную часть числа	TRN TRUNC(1)	релейная логика структурированный текст функциональный блок	15-11

(1) Только структурированный текст.

Вы можете смешивать типы данных, но это может привести к потере точности и ошибке округления, и для выполнения инструкции потребуется больше времени. Проверьте бит S:V, чтобы убедиться, отброшена ли у полученного результата дробная часть.

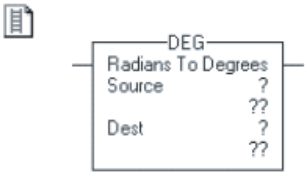
Для инструкций релейной логики, **жирный** шрифт типов данных означает, что это оптимальные типы данных. Инструкция выполняется быстрее и требует меньший объем памяти, если все операнды инструкции используют один и тот же оптимальный тип данных, обычно DINT или REAL.

Degrees (DEG) (Градусы)

Инструкция DEG преобразует Source (источник) (в радианах) в градусы и сохраняет результат в Destination (приемнике).

Операнды:

Релейная логика



Операнд	Тип:	Формат:	Описание:
Source	SINT INT DINT **REAL**	непосредственный тег	значение для преобразования в значение в радианах
Destination	SINT INT DINT **REAL**	тег	тег для хранения результата

Структурированный текст

```
dest := DEG(source);
```

Используйте DEG в качестве функции. Эта функция преобразует *source* в градусы и сохраняет результат в *dest.*

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег DEG	FBD_MATH_ADVANCED	структура	структура DEG

Структура FBD_MATH_ADVANCED

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Входное значение для инструкции преобразования. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат инструкции преобразования. Арифметические флаги состояния устанавливаются для этого выхода.

Описание:

Инструкция DEG использует следующий алгоритм:

Source*180/π (где (π = 3.141593)

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика:

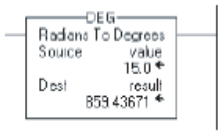
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Контроллер преобразует Source в градусы и помещает результат в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Преобразование *value* в градусы и помещение результата в *result.*

Релейная логика



Структурированный текст

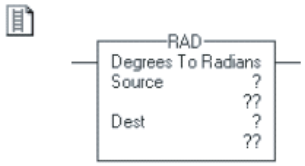
```
result := DEG(value);
```

Функциональный блок

Radians (RAD) (Радианы)

Инструкция RAD преобразует Source (источник) (в градусах) в радианы и сохраняет результат в Destination (приемнике).

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Описание:
Source	SINT INT DINT **REAL**	непосредственный тег	значение для преобразования в значение в градусах
Destination	SINT INT DINT REAL	тег	тег для хранения результата

Структурированный текст

```
dest := RAD(source);
```

Используйте RAD в качестве функции. Эта функция преобразует *source* в радианы и сохраняет результат в *dest.*

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.

Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег RAD	FBD_MATH_ADVANCED	структура	структура RAD

Структура FBD_MATH_ADVANCED

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Входное значение для инструкции преобразования. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат инструкции преобразования. Арифметические флаги состояния устанавливаются для этого выхода.

Описание:

Инструкция RAD использует следующий алгоритм:
Source*180/π (где π = 3.141593)

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика:

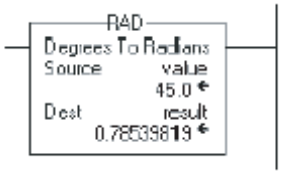
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Контроллер преобразует Source в радианы и помещает результат в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Преобразование *value* в радианы и помещение результата в *result.*

Релейная логика



Структурированный текст

```
result := RAD(value);
```

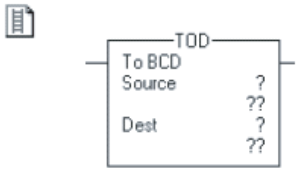
Функциональный блок

Convert to BCD (TOD) (Преобразование в код BCD)

Инструкция TOD преобразует десятичное значение (0≤ Source ≤ 99999999) в двоично-десятичное (BCD) и сохраняет результат в Destination (приемнике).

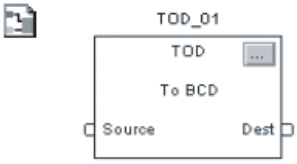
Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Source	SINT INT DINT	непосредственный тег	значение для преобразования в двоично-десятичное значение
Тег SINT или INT преобразуется в значение DINT посредством заполнения нулями.			
Destination	SINT INT DINT REAL	тег	сохраняет результат

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег TOD	FBD_CONVERT	структура	структура TOD

Структура FBD_MATH_ADVANCED

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Входное значение для инструкции преобразования. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат инструкции преобразования. Арифметические флаги состояния устанавливаются для этого выхода.

Описание:

BCD - это система десятичных, двоично-кодированных чисел, использующая четыре бита для кодирования каждой десятичной цифры (0-9).

Если вы введете отрицательное значение Source, инструкция сгенерирует неосновную ошибку и сбросит значение Destination.

Арифметические флаги состояния:

Арифметические флаги состояния затрагиваются.

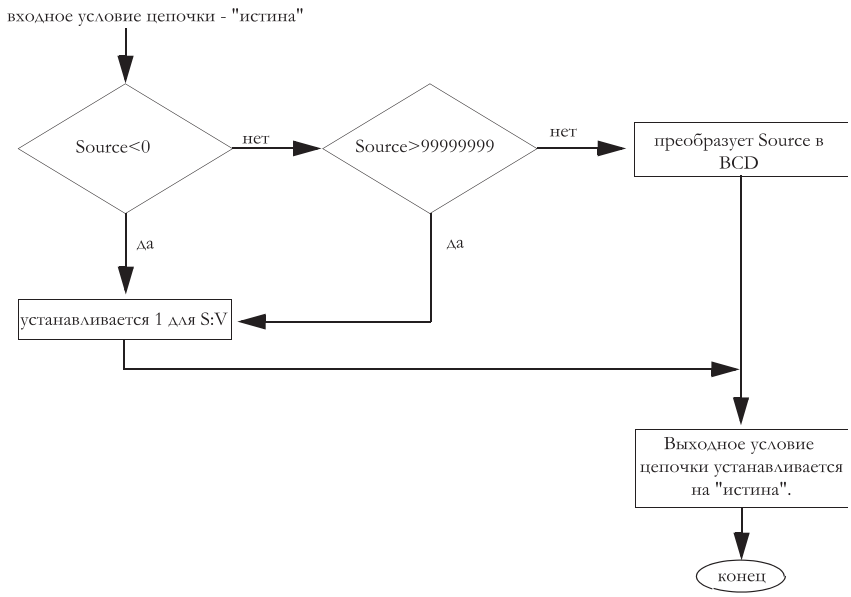
Условия ошибки:

Неосновная ошибка произойдет, если	Тип ошибки:	Код ошибки:
Source < 0	4	4

Выполнение:

Релейная логика:

Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».



S

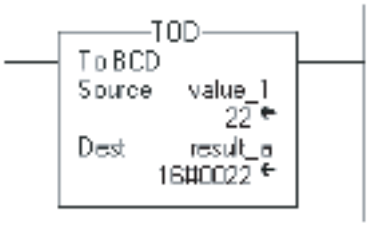
входное условие цепочки – «истина»	Контроллер преобразует Source в BCD и помещает результат в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Инструкция TOD преобразует *value_1* в двоично-десятичное значение (BCD) и помещает результат в *result_a*.

Релейная логика



Функциональный блок

Convert to Integer (FRD) (Преобразование в целое число)

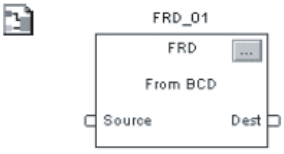
Инструкция FRD преобразует двоично-десятичное значение (BCD) Source (источника) в десятичное значение и сохраняет результат в Destination (приемнике).

Операнды:

Релейная логика

Операнд:	Тип:	Формат:	Описание:
Source	SINT INT DINT	непосредственный тег	значение для преобразования в десятичное значение
	Тег SINT или INT преобразуется в значение DINT посредством заполнения нулями.		
Destination	SINT INT DINT	тег	сохраняет результат

Функциональный блок



Операнд:	Тип:	Формат:	Описание:
тег FRD	FBD_CONVERT	структура	структура FRD

Структура FBD_MATH_ADVANCED

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Входное значение для инструкции преобразования. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	**Тип данных:**	**Описание:**
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат инструкции преобразования. Арифметические флаги состояния устанавливаются для этого выхода.

Описание: Инструкция FRD преобразует значение BCD (Source) в десятичное значение и сохраняет результат в Destination.

Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика:

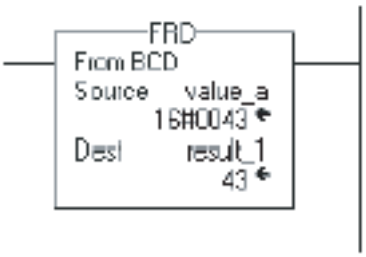
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Контроллер преобразует Source в десятичное значение и помещает результат в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Инструкция FRD преобразует *value_a* в десятичное значение и помещает результат в *result_1*.

Релейная логика

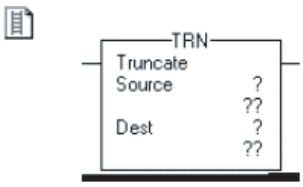


Функциональный блок

Truncate (TRN) (Усечение)

Инструкция TRN усекает дробную часть Source (источника) и сохраняет результат в Destination (приемнике).

Операнды:



Релейная логика

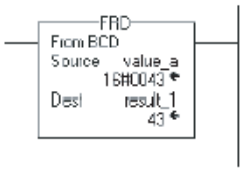
Операнд:	Тип:	Формат:	Описание:
Source	**REAL**	непосредственный тег	значение, дробная часть которого будет усечена
Destination	SINT INT **DINT** **REAL**	тег	тег для хранения результата

Структурированный текст

```
dest := TRUNC(source);
```

Используйте TRUNC в качестве функции. Эта функция усекает дробную часть *source* и сохраняет результат в *dest.*

Информацию о синтаксисе выражений в структурированном тексте можно найти в Приложении C.



Функциональный блок

Операнд:	Тип:	Формат:	Описание:
тег TRU	FBD_TRUNCATE	структура	структура TRN

Структура FBD_TRUNCATE

Входной параметр:	Тип данных:	Описание:
EnableIn	BOOL	Разрешение входа. Если этот параметр сброшен, то инструкция не выполняется, а выходы не обновляются. По умолчанию параметр установлен.
Source	REAL	Входное значение для инструкции преобразования. Допустимое значение = любое значение с плавающей точкой
Выходной параметр:	Тип данных:	Описание:
EnableOut	BOOL	Выполнение инструкции дало допустимый результат.
Dest	REAL	Результат инструкции преобразования. Арифметические флаги состояния устанавливаются для этого выхода.

Описание: Усечение не округляет значение, целая часть остается неизменной, независимо от значения дробной части

Арифметические флаги состояния: Арифметические флаги состояния затрагиваются.

Условия ошибки: отсутствуют

Выполнение:

Релейная логика:

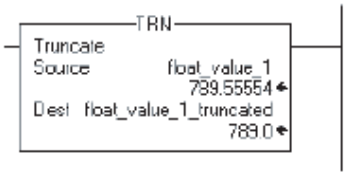
Условие:	Действие:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».
входное условие цепочки – «истина»	Контроллер усекает дробную часть Source и помещает результат в Destination. Выходное условие цепочки устанавливается на «истина».
постсканирование	Выходное условие цепочки устанавливается на «ложь».

Функциональный блок

Условие:	Действие:
предварительное сканирование	Никакого действия не производится.
первое сканирование инструкции	Никакого действия не производится.
первое выполнение инструкции	Никакого действия не производится.
EnableIn сбрасывается	EnableOut сбрасывается.
EnableIn устанавливается	Инструкция выполняется. EnableOut устанавливается.
постсканирование	Никакого действия не производится.

Пример: Усечение дробной части *float_value_1*, при этом целая часть остается неизменной, результат помещается в *float_value_1_truncated*.

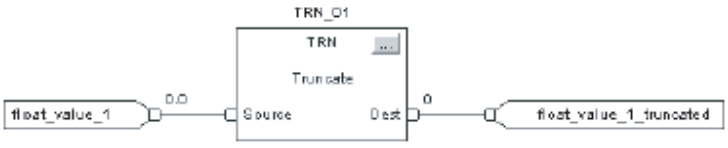
Релейная логика



Структурированный текст

```
float_value_1_truncated := TRUNC(float_value_1);
```

Функциональный блок



Инструкции ASCII последовательного порта (ABL, ACB, ACL, AHL, ARD, ARL, AWA, AWT)

Введение

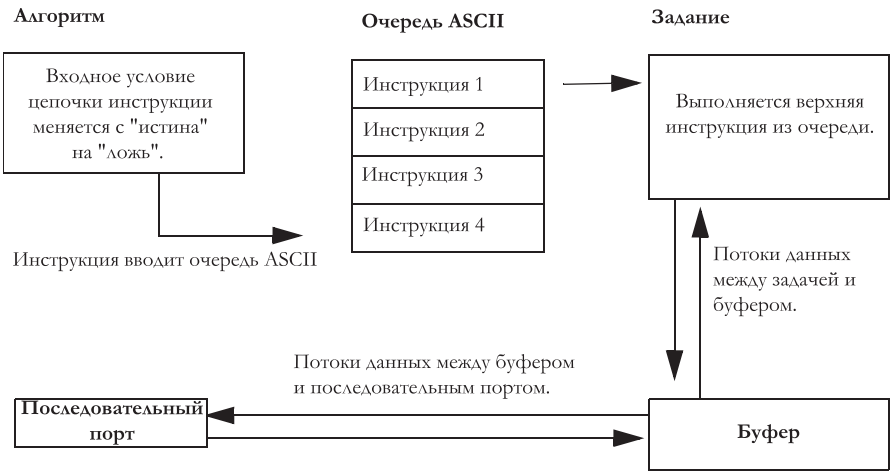
Используйте инструкции последовательного порта ASCII для считывания и записи символов ASCII.

ВАЖНО! Чтобы использовать инструкции ASCII последовательного порта, вы должны сконфигурировать последовательный порт контроллера. См. «*Общие процедуры контроллера Logix5000*», публикация 1756-PM001.

Если вы хотите:	Например:	Используйте эту инструкцию:	Имеющуюся в этих языках:	См. стр.
определить, когда буфер содержит символы завершения	проверить наличие данных, содержащих символы завершения	ABL	релейная логика структурированный текст	16-5
сосчитать символы в буфере	проверить наличие требуемого числа символов в буфере перед считыванием буфера	ACB	релейная логика структурированный текст	16-8
очистить буфер удалить инструкции последовательного порта ASCII, которые в данный момент выполняются или находятся в очереди	• удалить старые данные из буфера при запуске • синхронизировать буфер с каким-либо устройством	ACL	релейная логика структурированный текст	16-10
получить состояние линий управления последовательного порта включить или выключить сигнал DTR (сигнал управления последовательным устройством) включить или выключить сигнал RTS (сигнал запроса на пересылку данных)	вызвать отключение модема	AHL	релейная логика структурированный текст функциональный блок	16-12
считать фиксированное количество символов	считать данные из устройства, которое отправляет одинаковое количество символов при каждой передаче	ARD	релейная логика структурированный текст	16-16
считать меняющееся количество символов до и включая первый набор символов завершения	считать данные из устройства, которое отправляет меняющееся количество символов при каждой передаче	ARL	релейная логика структурированный текст	16-19
отправить символы и автоматически добавить один или два дополнительных символа, чтобы пометить конец данных	отправить сообщения, которые всегда используют одинаковый(е) символ(ы) завершения	AWA	релейная логика структурированный текст	16-23
отправить символы	отправить сообщения, которые используют меняющиеся символы завершения	AWT	релейная логика структурированный текст	16-28

Выполнение инструкции

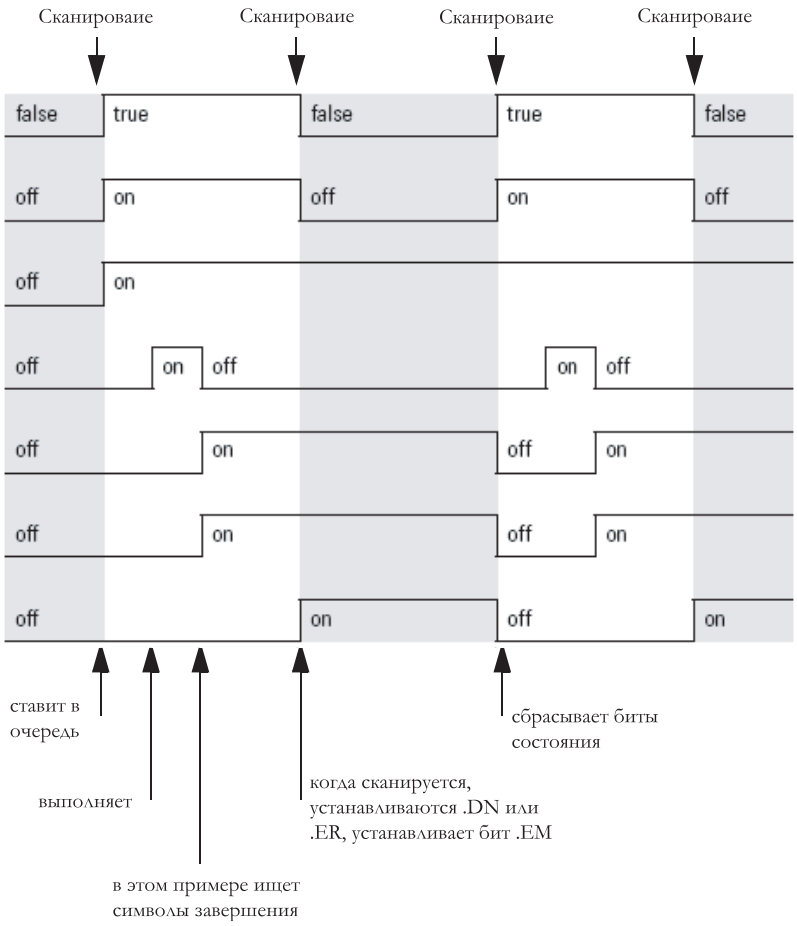
Инструкция ASCII последовательного порта выполняются асинхронно сканированию алгоритма:



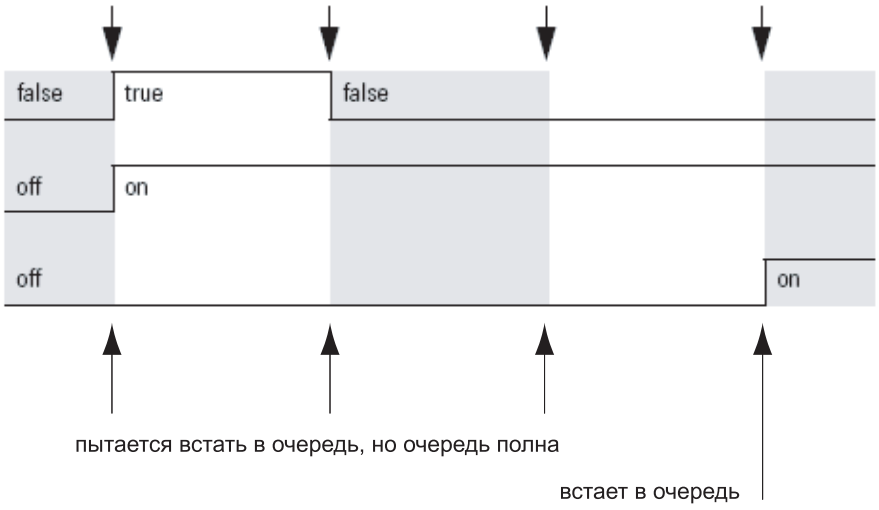
Каждая инструкция последовательного порта ASCII (за исключением ACL) использует структуру SERIAL_PORT_CONTROL для реализации следующих функций:

- управление выполнением инструкции
- предоставление информации о состоянии инструкции

Следующая ниже схема синхронизации изображает изменение битов состояния, когда инструкция ABL проверяет буфер на наличие символов завершения.



Очередь инструкций ASCII удерживает до 16 инструкций. Когда очередь полна, инструкция пытается встать в очередь при каждом последующем сканировании инструкции, как это показано ниже:

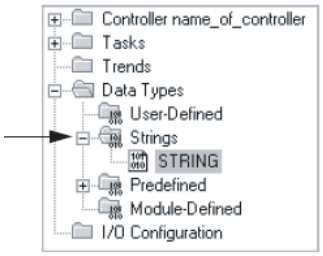


Коды ошибок ASCII

Если какая-либо инструкция ASCII последовательного порта не выполняется, член ERROR ее структуры SERIAL_PORT_CONTROL будет содержать один из следующих ниже шестнадцатеричных кодов ошибки.

Шестнадцате-ричный код:	Указывает что:
16#2	Модем перестал поддерживать связь.
16#3	При связи был потерян сигнал CTS (готовности).
16#4	Последовательный порт находился в системном режиме.
16#A	Перед выполнением инструкции был установлен бит .UL. Это препятствует выполнению инструкции.
16#C	Контроллер перешел из режима выполнения (Run) в режим программирования (Program). Это останавливает выполнение инструкций ASCII последовательного порта и очищает очередь.
16#D	В диалоговом окне Controller Properties (свойства контроллера), закладка User Protocol (протокол пользователя), был изменен размер буфера или параметры эхо-режима. Это останавливает выполнение инструкций ASCII последовательного порта и очищает очередь.
16#E	Выполнена инструкция ACL.
16#F	Пользовательский режим (User) последовательного порта был изменен на системный режим (System). Это останавливает выполнение инструкций ASCII последовательного порта и очищает очередь.
16#51	Значение LEN строкового тега либо отрицательно, либо больше размера DATA строкового тега.
16#54	Serial Port Control Length (контрольная длина последовательного порта) больше, чем размер буфера.
16#55	Serial Port Control Length (контрольная длина последовательного порта) либо отрицательна, либо больше размера Source (источника) или Destination (приемника).

Строковые типы данных



Вы сохраняете символы ASCII в тегах, которые используют строковый тип данных.

- Вы можете использовать строковый тип данных по умолчанию (STRING). Он позволяет хранить до 82 символов.
- Вы можете создать новый строковый тип данных, который позволит хранить меньше или больше символов.

Чтобы создать новый строковый тип данных обращайтесь к документу *«Общие процедуры контроллера Logix5000»*, публикация 1756-PM001.

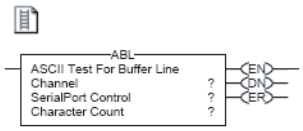
Каждый строковый тип данных содержит следующие члены:

Имя:	Тип данных:	Описание:	Примечания:
LEN	DINT	количество символов в строке	LEN автоматически обновляется всякий раз, когда вы: используете для ввода символов диалоговое окно String Browser используете инструкции, которые считывают, преобразуют или обрабатывают строку LEN указывает длину текущей строки. Член DATA может содержать дополнительные, старые символы, которые не включаются в подсчет LEN.
DATA	массив SINT	символы ASCII строки	Чтобы иметь доступ к символам строки, адресуйтесь к имени тега. Например, чтобы получить доступ к символам тега *string_1*, введите *string_1*. Каждый элемент массива DATA содержит один символ. Вы можете создавать новые строковые типы данных, которые сохраняют меньше или больше символов.

ASCII Test For Buffer Line (ABL) (Проверка буфера на наличие символа завершения)

Инструкция ABL подсчитывает символы в буфере до первого символа завершения и включая первый символ завершения.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Channel	DINT	непосредственный тег	0
Serial Port Control	SERIAL_PORT_ CONTROL	тег	тег, управляющий работой
Character Count	DINT	непосредственный	0 При выполнении выводит на экран количество символов в буфере, включая первый набор символов завершения.

Структурированный текст

```
ABL(Channel
    SerialPortControl);
```

Операнды такие же, как и операнды для инструкции ABL в релейной логике. Вы имеете доступ к значению Character Count посредством члена .POS структуры SERIAL_PORT_CONTROL.

Структура SERIAL_PORT_CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция разрешена.
.EU	BOOL	Бит очереди указывает на то, что инструкция встала в очередь ASCII.
.DN	BOOL	Бит выполнения указывает на то, что инструкция выполнена, но асинхронна сканированию алгоритма.
.RN	BOOL	Бит выполнения указывает на то, что инструкции выполняется.
.EM	BOOL	Пустой бит указывает на то, что инструкция выполнена и синхронна сканированию алгоритма.
.ER	BOOL	Бит ошибки указывает на наличие ошибки.
.FD	BOOL	Бит поиска указывает на то, что инструкция нашла символ или символы завершения.
.POS	DINT	Позиция определяет количество символов в буфере до первого набора символов завершения и включая первый набор символов завершения. Инструкция возвращает этот элемент только после того, как находит символ или символы завершения.
.ERROR	DINT	Ошибка содержит шестнадцатеричное значение, определяющее причину ошибки.

Описание: Инструкция ABL ищет первый набор символов завершения в буфере. Если инструкция находит символы завершения, она:

- устанавливает бит .FD
- подсчитывает количество символов в буфере до первого набора символов завершения и включая первый набор символов завершения

Закладка User Protocol (протокол пользователя) диалогового окна Controller Properties (свойства контроллера) задает символы ASCII, которые эта инструкция рассматривает как символы завершения.

Чтобы запрограммировать инструкцию ABL, следуйте этим указаниям:

1 Сконфигурируйте пользовательский режим последовательного порта контроллера и задайте символы, которые будут служить символами завершения.

2 Это переходная инструкция:

- Для релейных логики переключайте входное условиеусловие цепочки из положения «сброшено» в положение «установлено» каждый раз, когда должна выполнятся эта инструкция.
- Для структурированного текста определите инструкцию так, чтобы она выполнялась только при переходе. См. Приложение C.

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется, когда входное условие цепочки переключается из положения «сброшено» в положение «установлено». Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция подсчитывает количество символов в буфере. Бит .EN устанавливается. Оставшиеся биты состояния, кроме .UL, сбрасываются. Инструкция пытается встать в очередь ASCII.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Непрерывная проверка буфера на предмет символов завершения.

Релейная логика

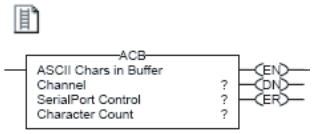
Структурированный текст

```
ABL(0,MV_line);
```

ASCII Chars in Buffer (ACB) (Подсчет числа символов ASCII в буфере)

Инструкция ACB подсчитывает символы в буфере.

Операнды:



Релейная логика:

Операнд:	Тип:	Формат:	Описание:
Channel	DINT	непосредственный тег	0
Serial Port Control	SERIAL_PORT_CONTROL	тег	тег, управляющий работой
Character Count	DINT	непосредственный	0 При выполнении выводит на экран количество символов в буфере

```
ACB(Channel
    SerialPortControl);
```

Структурированный текст

Операнды такие же, как и операнды для инструкции ACD в релейной логике. Однако вы задаете значение Character Count путем организации доступа к члену .POS структуры SERIAL_PORT_CONTROL, а не включения этого значения в список операндов.

Структура SERIAL_PORT_CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция разрешена.
.EU	BOOL	Бит очереди указывает на то, что инструкция встала в очередь ASCII.
.DN	BOOL	Бит выполнения указывает на то, что инструкция выполнена, но асинхронна сканированию алгоритма.
.RN	BOOL	Бит выполнения указывает на то, что инструкции выполняется.
.EM	BOOL	Пустой бит указывает на то, что инструкция выполнена и синхронна сканированию алгоритма.
.ER	BOOL	Бит ошибки указывает на наличие ошибки.
.FD	BOOL	Бит поиска указывает на то, что инструкция нашла символ.
.POS	DINT	Позиция определяет количество символов в буфере до первого набора символов завершения и включая первый набор символов завершения.
.ERROR	DINT	Ошибка содержит шестнадцатеричное значение, определяющее причину ошибки.

Описание: Инструкция ACB подсчитывает количество символов в буфере.

Чтобы запрограммировать инструкцию ACB, следуйте этим указаниям:

1 Сконфигурируйте пользовательский режим последовательного порта контроллера.

2 Это переходная инструкция:

- Для релейной логики переключайте входное условие цепочки из положения «сброшено» в положение «установлено» каждый раз, когда должна выполнятся эта инструкция.
- Для структурированного текста определите инструкцию так, чтобы она выполнялась только при переходе. См. Приложение C.

Арифметические флаги состояния: не затрагиваются

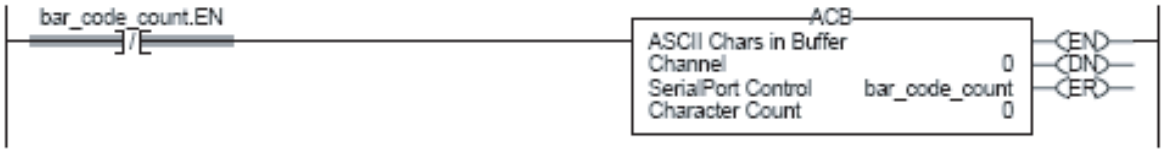
Условия ошибки: отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется, когда входное условие цепочки переключается из положения «сброшено» в положение «установлено». Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция подсчитывает количество символов в буфере. Бит .EN устанавливается. Оставшиеся биты состояния, кроме .UL, сбрасываются. Инструкция пытается встать в очередь ASCII.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Непрерывный подсчет символов в буфере.

Релейная логика



Структурированный текст

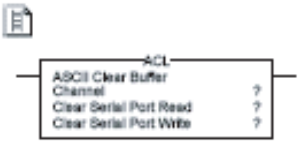
```
ACB(0,bar_code_count);
```

ASCII Clear Buffer (ACL) (Очистка буфера/удаление инструкций ASCII)

Инструкция ACL немедленно очищает буфер и очередь ASCII.

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:
Channel	DINT	непосредственный тег	0
Clear Serial Port Read	BOOL	непосредственный тег	Чтобы очистить буфер и удалить инструкции ARD и ARL из очереди, введите Yes.
Clear Serial Port Write	BOOL	непосредственный тег	Чтобы удалить инструкции AWA и AWT из очереди, введите Yes.

Структурированный текст

```
ACL(Channel,
    ClearSerialPortRead,
    ClearSerialPortWrite);
```

Операнды такие же, как и операнды для инструкции ACL в релейной логике.

Описание:

Инструкция ACL немедленно выполняет одно из следующих действий или оба действия:

- очищает буфер символов и очередь ASCII инструкций считывания
- очищает очередь ASCII инструкций записи.

Чтобы запрограммировать инструкцию ACL следуйте этим указаниям:

1 Сконфигурируйте последовательный порт контроллера:

Если ваше приложение:	То:
использует инструкции ARD или ARL	Выберите режим User
не использует инструкции ARD или ARL	Выберите режим System или User

2 Чтобы определить, была ли инструкция удалена из очереди или прервана, проверьте следующие элементы соответствующей инструкции:

- бит .ER установлен
- член .ERROR имеет значение 16#E

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция очищает заданные инструкции и буфер(ы).	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Очистка буфера и очереди ASCII, когда контроллер входит в режим выполнения (Run).

Релейная логика

Структурированный текст

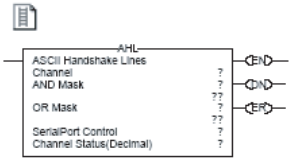
```
osri_1.InputBit  :=  S:FS;
OSRI(osri_1);
IF  (osri_1.OutputBit)  THEN
      ACL(0,0,1);
END_IF;
```

ASCII Handshake Lines (AHL) (Квитирование линий ASCII)

Инструкция AHL получает состояние линий управления и включает или выключает сигналы DTR (сигнал управления последовательным устройством) и RTS (сигнал запроса на пересылку данных).

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:
Channel	DINT	непосредственный тег	0
ANDMask	DINT	непосредственный тег	Обратитесь к описанию.
ORMask	DINT	непосредственный тег	
Serial Port Control	SERIAL_PORT_CONTROL	тег	тег, который управляет работой
Channel Status (десятичный)	DINT	непосредственный	0 При выполнении выводит на экран состояние линий управления.

Для определения состояния этого канала управления:	Проверьте этот бит:
CTS (сигнал готовности)	0
RTS (сигнал запроса на пересылку данных)	1
DSR (отчет о состоянии устройства)	2
DCD (обнаружение информационного сигнала)	3
DTR (сигнал управления последовательным устройством)	4
Получение символа XOFF	5

Структурированный текст

```
AHL(Channel,ANDMask,ORMask,
    SerialPortControl);
```

Операнды такие же, как и операнды для инструкции AHL в релейной логике. Однако вы задаете значение Channel Status путем организации доступа к члену .POS структуры SERIAL_PORT_CONTROL, а не включения этого значения в список операндов.

Структура SERIAL_PORT_CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция разрешена.
.EU	BOOL	Бит очереди указывает на то, что инструкция встала в очередь ASCII.
.DN	BOOL	Бит выполнения указывает на то, что инструкция выполнена, но асинхронна сканированию алгоритма.
.RN	BOOL	Бит выполнения указывает на то, что инструкции выполняется.
.EM	BOOL	Пустой бит указывает на то, что инструкция выполнена и синхронна сканированию алгоритма.
.ER	BOOL	Бит ошибки указывает на наличие ошибки.
.FD	BOOL	Бит поиска не применяется к этой инструкции.
.POS	DINT	Позиция сохраняет статус линий управления.
.ERROR	DINT	Ошибка содержит шестнадцатеричное значение, определяющее причину ошибки.

Описание: Инструкция AHL может:

- получить состояние линий управления последовательного порта
- включить или выключить сигнал управления последовательным устройством (DTR)
- включить или выключить сигнал запроса на пересылку данных (RTS).

Чтобы запрограммировать инструкцию AHL, следуйте этим указаниям:

1 Сконфигурируйте последовательный порт контроллера:

Если ваше приложение:	То:
использует инструкции ARD или ARL	Выберите режим User
не использует инструкции ARD или ARL	Выберите режим System или User

2 Используйте следующую таблицу для выбора правильных значений операндов ANDMask и ORMask:

Чтобы DTR:	А RTS:	Введите это значение ANDMask:	И это значение ORMask:
выключить	выключить	3	0
	включить	1	2
	оставить неизменным	1	0
включить	выключить	2	1
	включить	0	3
	оставить неизменным	0	1
оставить неизменным	выключить	2	0
	включить	0	2
	оставить неизменным	0	0

3 Это переходная инструкция:

- Для релейной логики переключайте входное условие цепочки из положения «сброшено» в положение «установлено» каждый раз, когда должна выполнятся эта инструкция.
- Для структурированного текста определите инструкцию так, чтобы она выполнялась только при переходе. См. Приложение C.

Арифметические флаги состояния: не затрагиваются

Условия ошибки:

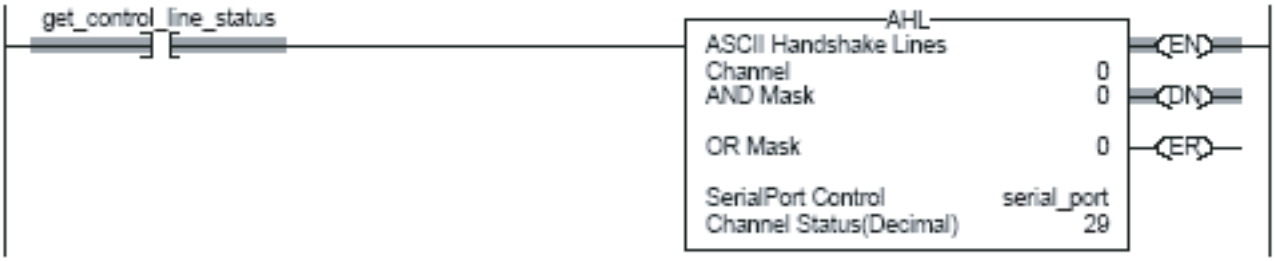
Тип:	Код:	Причина:	Способ исправления:
4	57	Инструкция AHL не выполнена, поскольку последовательный порт не настроен на квитирование установленной связи.	Либо: • Измените настройку Control Line последовательного порта. • Уничтожьте инструкцию AHL.

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется, когда входное условие цепочки переключается из положения «сброшено» в положение «установлено». Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция получает состояние линии управления и включает или выключает сигналы DTR и RTS. Бит .EN устанавливается. Оставшиеся биты состояния, кроме .UL, сбрасываются. Инструкция пытается встать в очередь ASCII.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Когда *get_control_line_status* устанавливается, определите состояние линий управления последовательного порта и сохраните это состояние в операнде Channel Status. Чтобы просмотреть состояние заданной линии управления, отследите тег SerialPortControl и разверните член POS.

Релейная логика



Структурированный текст

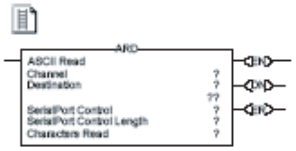
```
osri_1.InputBit  :=  get_control_line_status;
OSRI(osri_1);

IF  (osri_1.OutputBit)  THEN
      AHL(0,0,0,serial_port);
END_IF;
```

ASCII Read (ARD) (Чтение фиксированного количества символов ASCII)

Инструкция ARD удаляет символы из буфера и сохраняет их в Destination (приемнике).

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:	Примечания:
Channel	DINT	непосред-ственный тег	0	
Destination	**строка** SINT INT DINT	тег	тег, в который перемещаются символы (считывание): • Для строкового типа данных введите имя тега. • Для массива SINT, INT или DINT введите первый элемент массива.	• Ели вы хотите сравнить, преобразовать или обработать символы, используйте строковый тип данных. • Строковыми типами данных являются: • строковый тип данных по умолчанию STRING • любой новый строковый тип данных, который вы создадите.
Serial Port Control	SERIAL_PORT_ CONTROL	тег	тег, который управляет работой	
Serial Port Control Length	DINT	непосред-ственный	количество символов для перемещения в приемник (считывание)	• Serial Port Control Length должен быть меньше или равен размеру Destination. • Если вы хотите установить Serial Port Control Length равным размеру Destination, введите 0.
Characters Read	DINT	непосред-ственный	0	При выполнении выводит на экран количество символов, которые были считаны.

Структурированный текст

```
ARD(Channel,Destination,
    SerialPortControl);
```

Операнды такие же, как и операнды для инструкции ARD в релейной логике. Однако вы задаете значения Serial Port Control Length и Characters Read путем организации доступа к членам .LEN и .POS структуры SERIAL_PORT_CONTROL, а не включения этих значений в список операндов.

Структура SERIAL_PORT_CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция разрешена.
.EU	BOOL	Бит очереди указывает на то, что инструкция встала в очередь ASCII.
.DN	BOOL	Бит выполнения указывает на то, что инструкция выполнена, но асинхронна сканированию алгоритма.
.RN	BOOL	Бит выполнения указывает на то, что инструкции выполняется.
.EM	BOOL	Пустой бит указывает на то, что инструкция выполнена и синхронна сканированию алгоритма.
.ER	BOOL	Бит ошибки указывает на наличие ошибки.
.FD	BOOL	Бит поиска не применяется к этой инструкции.
.LEN	DINT	Длина указывает количество символов для перемещения в приемник (считывание).
.POS	DINT	Позиция выводит на экран количество считанных символов.
.ERROR	DINT	Ошибка содержит шестнадцатеричное значение, определяющее причину ошибки.

Описание:

Инструкция ARD удаляет заданное количество символов из буфера и сохраняет их в Destination (приемнике).

- Инструкция ARD продолжает выполняться, пока не удалит заданное количество символов (Serial Port Control Length).
- Пока выполняется инструкция ARD, другие инструкции ASCII последовательного порта не выполняются.

Чтобы запрограммировать инструкцию ARD, следуйте этим указаниям:

1 Сконфигурируйте пользовательский режим последовательного порта контроллера.

2 Используйте результаты инструкции ACB для запуска инструкции ARD. Это предотвратит задержку очереди ASCII, пока инструкция ожидает требуемое количество символов.

3 Это переходная инструкция:

- Для релейной логики переключайте входное условие цепочки из положения «сброшено» в положение «установлено» каждый раз, когда должна выполнятся эта инструкция.
- Для структурированного текста определите инструкцию так, чтобы она выполнялась только при переходе. См. Приложение C.

4 Чтобы запустить следующую операцию, когда инструкция выполнена, проверьте бит EM.

Арифметические флаги состояния: не затрагиваются

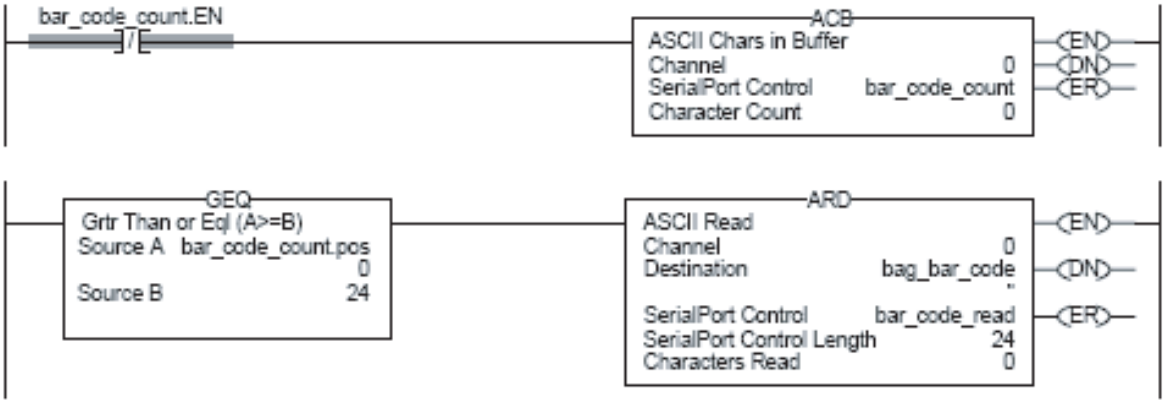
Условия ошибки: отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется, когда входное условие цепочки переключается из положения «сброшено» в положение «установлено». Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция удаляет символы из буфера и сохраняет их в приемнике. Бит .EN устанавливается. Оставшиеся биты состояния, кроме .UL, сбрасываются. Инструкция пытается встать в очередь ASCII.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Устройство считывания штрихового кода отправляет штриховые коды в последовательный порт (канал 0) контроллера. Каждый штриховой код содержит 24 символа. Чтобы определить, когда контроллер получает штриховой код, инструкция ACB непрерывно подсчитывает символы в буфере. Если буфер содержит, по крайней мере, 24 символа, контроллер получает штриховой код. Инструкция ARD перемещает штриховой код в член DATA тега *bag_bar_code*, который представляет собой строку.

Релейная логика



Структурированный текст

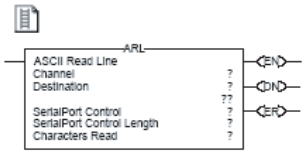
```
ACB(0,bar_code_count);
IF  bar_code_count.POS  >=  24  THEN
      bar_code_read.LEN  :=  24;
      ARD(0,bag_bar_code,bar_code_read);
END_IF;
```

ASCII Read Line (ARL) (Чтение переменного количества символов ASCII)

Инструкция ARL удаляет заданные символы из буфера и сохраняет их в Destination (приемнике).

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:	Примечания:
Channel	DINT	непосред-ственный тег	0	
Destination	строка SINT INT DINT	тег	тег, в который перемещаются символы (считывание): • Для строкового типа данных введите имя тега. • Для массива SINT, INT или DINT введите первый элемент массива.	• Ели вы хотите сравнить, преобразовать или обработать символы, используйте строковый тип данных. • Строковыми типами данных являются: • строковый тип данных по умолчанию STRING • любой новый строковый тип данных, который вы создадите.
Serial Port Control	SERIAL_POR T_ CONTROL	тег	тег, который управляет работой	
Serial Port Control Length	DINT	непосред-ственный	максимальное количество символов для считывания, если не обнаружены символы завершения	•Введите максимальное количество символов, которое может содержать сообщение (т.е. когда остановить считывание, если не обнаружены символы завершения). Например, если сообщения содержат от 3 до 6 символов, введите 6. • Значение Serial Port Control Length должно быть меньше или равно размеру Destination. • Если вы хотите установить значение Serial Port Control Length равным размеру Destination, введите 0.
Characters Read	DINT	непосред-ственный	0	При выполнении выводит на экран количество символов, которые были считаны.

Структурированный текст

```
ARL(Channel,Destination,
    SerialPortControl);
```

Операнды такие же, как и операнды для инструкции ARL в релейной логике. Однако вы задаете значения Serial Port Control Length и Characters Read путем организации доступа к членам .LEN и .POS структуры SERIAL_PORT_CONTROL, а не включения этих значений в список операндов.

Структура SERIAL_PORT_CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция разрешена.
.EU	BOOL	Бит очереди указывает на то, что инструкция встала в очередь ASCII.
.DN	BOOL	Бит выполнения указывает на то, что инструкция выполнена, но асинхронна сканированию алгоритма.
.RN	BOOL	Бит выполнения указывает на то, что инструкции выполняется.
.EM	BOOL	Пустой бит указывает на то, что инструкция выполнена и синхронна сканированию алгоритма.
.ER	BOOL	Бит ошибки указывает на наличие ошибки.
.FD	BOOL	Бит поиска не применяется к этой инструкции.
.LEN	DINT	Длина указывает максимальное количество символов для перемещения в приемник (т.е. когда остановить считывание, если символы завершения не обнаружены).
.POS	DINT	Позиция выводит на экран количество считанных символов.
.ERROR	DINT	Ошибка содержит шестнадцатеричное значение, определяющее причину ошибки.

Описание: Инструкция ARK удаляет символы из буфера и сохраняет их в Destination (приемнике) следующим образом.

- Инструкция ARL продолжает выполняться, пока не удалит:
  - первый набор символов завершения
  - или заданное количество символов (Serial Port Control Length).
- Пока выполняется инструкция ARL, другие инструкции ASCII последовательного порта не выполняются.

Чтобы запрограммировать инструкцию ARL, следуйте этим указаниям:

1 Сконфигурируйте пользовательский режим последовательного порта контроллера.

a. Выберите режим User.

b. Определите символы, которые будут служить символами завершения.

2 Используйте результаты инструкции ABL для запуска инструкции ARL. Это предотвратит задержку очереди ASCII, пока инструкция ожидает требуемое количество символов.

3 Это переходная инструкция:

- Для релейной логики переключайте входное условие цепочки из положения «сброшено» в положение «установлено» каждый раз, когда должна выполнятся эта инструкция.
- Для структурированного текста определите инструкцию так, чтобы она выполнялась только при переходе. См. Приложение C.

4 Чтобы запустить следующую операцию, когда инструкция выполнена, проверьте бит EM.

Арифметические флаги состояния: не затрагиваются

Условия ошибки: отсутствуют

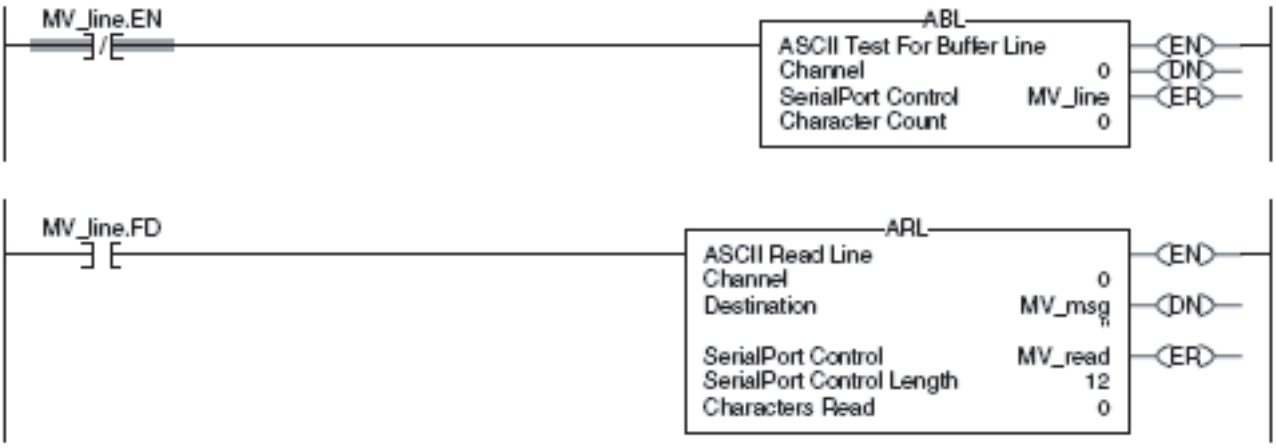
Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется, когда входное условие цепочки переключается из положения «сброшено» в положение «установлено». Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция удаляет заданные символы из буфера и сохраняет их в приемнике. Бит .EN устанавливается. Оставшиеся биты состояния, кроме .UL, сбрасываются. Инструкция пытается встать в очередь ASCII.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Непрерывно проверяйте буфер на наличие сообщений с терминала MessageView. Поскольку каждое сообщение заканчивается возвратом каретки ($r), этот возврат каретки определяется как символ завершения в закладке User Protocol (протокол пользователя) диалогового окна Controller Properties (свойства контроллера). Когда ABL находит возврат каретки, она устанавливает бит FD.

Когда инструкция ABL находит возврат каретки (*MV_lineFD* установлен), это означает, что контроллер уже получил полное сообщение. Инструкция ARL удаляет символы из буфера, до возврата каретки и включая возврат каретки, и размещает их в члене DATA тега *MV_msg*, который является строкой.

Релейная логика



Структурированный текст

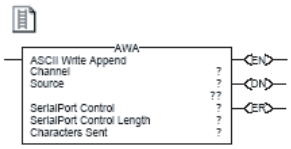
```
ABL(0,MV_line);

osri_1.InputBit  :=  MVLine.FD;
OSRI(osri_1);

IF  (osri_1.OutputBit)  THEN
      mv_read.LEN  :=  12;
      ARL(0,MV_msg,MV_read);
END_IF;
```

ASCII Write Append (AWA) (Присоединение к ASCII при записи)

Инструкция AWA отправляет заданное количество символов из тега Source (источник) в последовательное устройство и добавляет один или два предварительно заданных символа.

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:	Примечания:
Channel	DINT	непосред-ственный тег	0	
Destination	строка SINT INT DINT	тег	тег, который содержит символы для отправки: • Для строкового типа данных введите имя тега. • Для массива SINT, INT или DINT введите первый элемент массива.	• Ели вы хотите сравнить, преобразовать или обработать символы, используйте строковый тип данных. • Строковыми типами данных являются: • строковый тип данных по умолчанию STRING • любой новый строковый тип данных, который вы создадите.
Serial Port Control	SERIAL_PORT_ CONTROL	тег	тег, который управляет работой	
Serial Port Control Length	DINT	непосред-ственный	количество символов для отправки	• Serial Port Control Length должен быть меньше или равен размеру Source. • Если вы хотите установить Serial Port Control Length равным размеру Source, введите 0.
Characters Sent	DINT	непосред-ственный	0	При выполнении выводит на экран количество символов, которые были отправлены.

Структурированный текст

```
AWA(Channel,Source,
    SerialPortControl);
```

Операнды такие же, как и операнды для инструкции AWA в релейной логике. Однако вы задаете значения Serial Port Control Length и Characters Sent путем организации доступа к членам .LEN и .POS структуры SERIAL_PORT_CONTROL, а не включения этих значений в список операндов.

Структура SERIAL_PORT_CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция разрешена.
.EU	BOOL	Бит очереди указывает на то, что инструкция встала в очередь ASCII.
.DN	BOOL	Бит выполнения указывает на то, что инструкция выполнена, но асинхронна сканированию алгоритма.
.RN	BOOL	Бит выполнения указывает на то, что инструкции выполняется.
.EM	BOOL	Пустой бит указывает на то, что инструкция выполнена и синхронна сканированию алгоритма.
.ER	BOOL	Бит ошибки указывает на наличие ошибки.
.FD	BOOL	Бит поиска не применяется к этой инструкции.
.LEN	DINT	Длина указывает количество символов для отправки.
.POS	DINT	Позиция выводит на экран количество отправленных символов.
.ERROR	DINT	Ошибка содержит шестнадцатеричное значение, определяющее причину ошибки.

Описание:

Инструкция AWA:

- отправляет заданное количество символов (Serial Port Control Length) тега Source (источника) в устройство, подсоединенное к последовательному порту контроллера
- добавляет в конец один или два символа, которые определены в диалоговом окне Controller Properties (свойства контроллера), закладка User Protocol (протокол пользователя).

Чтобы запрограммировать инструкцию ARL, следуйте этим указаниям:

1 Сконфигурируйте последовательный порт контроллера:

a. Включает ли ваше приложение также инструкции ARD и ARL?

Если:	То:
Да	Выберите пользовательский режим (User)
Нет	Выберите системный режим (System) или пользовательский режим (User).

b. Задайте символы для присоединения к данным.

2 Это переходная инструкция:

- Для релейной логики переключайте входное условие цепочки из положения «сброшено» в положение «установлено» каждый раз, когда должна выполнятся эта инструкция.
- Для структурированного текста определите инструкцию так, чтобы она выполнялась только при переходе. См. Приложение C.

3 Всегда ли вы отправляете одинаковое количество символов каждый раз, когда выполняется инструкция?

Если:	То:
Да	В Serial Port Control Length введите количество символов для отправки.
Нет	Перед выполнением инструкции установите член LEN тега Source в член LEN тега Serial Port Control.

Арифметические флаги состояния: не затрагиваются

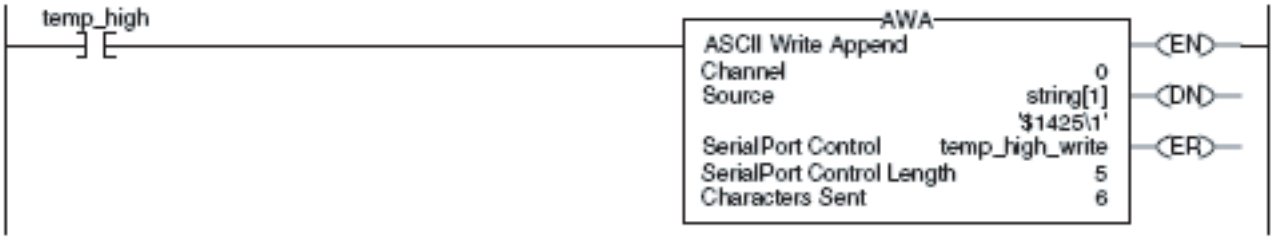
Условия ошибки: отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется, когда входное условие цепочки переключается из положения «сброшено» в положение «установлено». Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция отправляет заданное количество символов и добавляет один или два предварительно заданных символа. Бит .EN устанавливается. Оставшиеся биты состояния, кроме .UL, сбрасываются. Инструкция пытается встать в очередь ASCII.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример 1: Если температура превышает верхний предел (бит *temp_high* устанавливается), инструкция AWA отправляет сообщение на терминал MessageView, который подсоединен к последовательному порту контроллера. Это сообщение содержит пять символов из члена DATA тега (*string[1]*), который является строкой. (*$14* считается за один символ. Это шестнадцатеричный код для символа Ctrl-T). Инструкция также отправляет (добавляет) символы, заданные в окне свойств контроллера. В этом примере инструкция AWA отправляет возврат каретки ($0D), который помечает конец сообщения.

Релейная логика



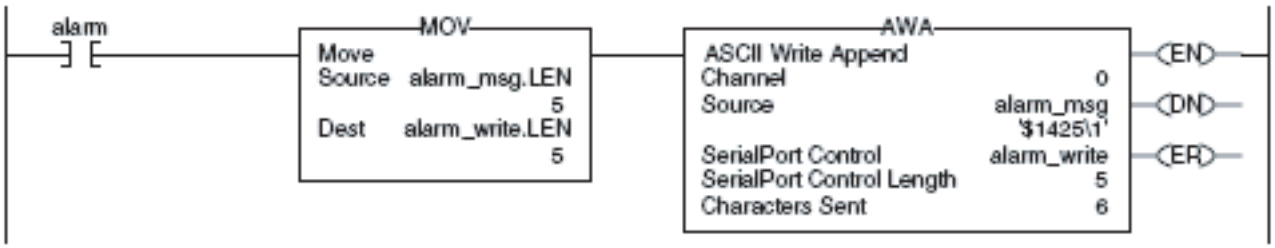
Структурированный текст

```
IF temp_high THEN

      temp_high_write.LEN := 5;

      AWA(0,string[1],temp_high_write);

      temp_high := 0;

END_IF;
```

Пример 2: Когда устанавливается *alarm*, инструкция AWA отправляет заданное количество символов в *alarm_msg* и добавляет символ завершения (s).

Поскольку количество символов в *alarm_msg* меняется, цепочка, в первую очередь, пересылает длину строки (*alarm_msg*.LEN) в Serial Port Control Length инструкции AWA (*alarm_write*.LEN). В *alarm_msg* *$14* считается за один символ. Это шестнадцатеричный код для символа Ctrl-T.

Релейная логика



Структурированный текст

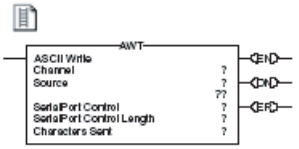
```
osri_1.InputBit := alarm;

OSRI(osri_1);

IF (osri_1.OutputBit) THEN

       alarm_write.LEN := alarm_msg.LEN;

       AWA(0,alarm_msg,alarm_write);

END_IF;
```

ASCII Write (AWT) (Запись ASCII)

Инструкция AWT отправляет заданное количество символов тега Source (источника) в последовательное устройство.

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Описание:	Примечания:
Channel	DINT	непосред-ственный тег	0	
Destination	строка SINT INT DINT	тег	тег, который содержит символы для отправки: • Для строкового типа данных введите имя тега. • Для массива SINT, INT или DINT введите первый элемент массива.	• Ели вы хотите сравнить, преобразовать или обработать символы, используйте строковый тип данных. • Строковыми типами данных являются: • строковый тип данных по умолчанию STRING • любой новый строковый тип данных, который вы создадите.
Serial Port Control	SERIAL_PORT_ CONTROL	тег	тег, который управляет работой	
Serial Port Control Length	DINT	непосред-ственный	количество символов для отправки	Serial Port Control Length должен быть меньше или равен размеру Source. Если вы хотите установить Serial Port Control Length равным размеру Source, введите 0.
Characters Sent	DINT	непосред-ственный	0	При выполнении выводит на экран количество символов, которые были отправлены.

Структурированный текст

```
AWT(Channel,Source,
    SerialPortControl);
```

Операнды такие же, как и операнды для инструкции AWT в релейной логике. Однако вы задаете значения Serial Port Control Length и Characters Sent путем организации доступа к членам .LEN и .POS структуры SERIAL_PORT_CONTROL, а не включения этих значений в список операндов.

Структура SERIAL_PORT_CONTROL

Мнемоника:	Тип данных:	Описание
.EN	BOOL	Бит разрешения указывает на то, что инструкция разрешена.
.EU	BOOL	Бит очереди указывает на то, что инструкция встала в очередь ASCII.
.DN	BOOL	Бит выполнения указывает на то, что инструкция выполнена, но асинхронна сканированию алгоритма.
.RN	BOOL	Бит выполнения указывает на то, что инструкции выполняется.
.EM	BOOL	Пустой бит указывает на то, что инструкция выполнена и синхронна сканированию алгоритма.
.ER	BOOL	Бит ошибки указывает на наличие ошибки.
.FD	BOOL	Бит поиска не применяется к этой инструкции.
.LEN	DINT	Длина указывает количество символов для отправки.
.POS	DINT	Позиция выводит на экран количество отправленных символов.
.ERROR	DINT	Ошибка содержит шестнадцатеричное значение, определяющее причину ошибки.

Описание

Инструкция AWT отправляет заданное количество символов (Serial Port Control Length) тега Source (источника) в устройство, подсоединенное к последовательному порту контроллера.

Чтобы запрограммировать инструкцию AWT, следуйте этим указаниям:

1 Сконфигурируйте последовательный порт контроллера:

Если ваше приложение:	То:
использует инструкции ARD или ARL	Выберите режим User
не использует инструкции ARD или ARL	Выберите режим System или User

2 Это переходная инструкция:

- Для релейной логики переключайте входное условие цепочки из положения «сброшено» в положение «установлено» каждый раз, когда должна выполнятся эта инструкция.
- Для структурированного текста определите инструкцию так, чтобы она выполнялась только при переходе. См. Приложение C.

3 Всегда ли вы отправляете одинаковое количество символов каждый раз, когда выполняется инструкция?

Если:	То:
Да	В Serial Port Control Length введите количество символов для отправки.
Нет	Перед выполнением инструкции переместите член LEN тега Source в член LEN тега Serial Port Control.

Арифметические флаги состояния: не затрагиваются

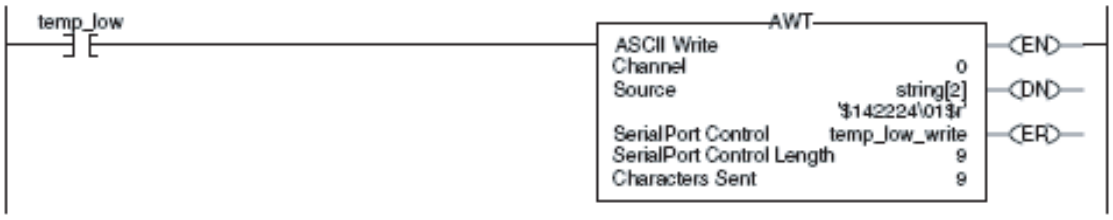
Условия ошибки: отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется, когда входное условие цепочки переключается из положения «сброшено» в положение «установлено». Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция отправляет заданное количество символов. Бит .EN устанавливается. Оставшиеся биты состояния, кроме .UL, сбрасываются. Инструкция пытается встать в очередь ASCII.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример 1: Если температура достигает нижнего предела (устанавливается бит *temp_low*), инструкция AWT отправляет сообщение на терминал MessageView, который подсоединен к последовательному порту контроллера. Это сообщение содержит девять символов из члена DATA тега *string[2]*, который является строкой. (*$14* считается за один символ. Это шестнадцатеричный код для символа Ctrl-T). Последний символ является возвратом каретки ($r), который помечает конец сообщения.

Релейная логика

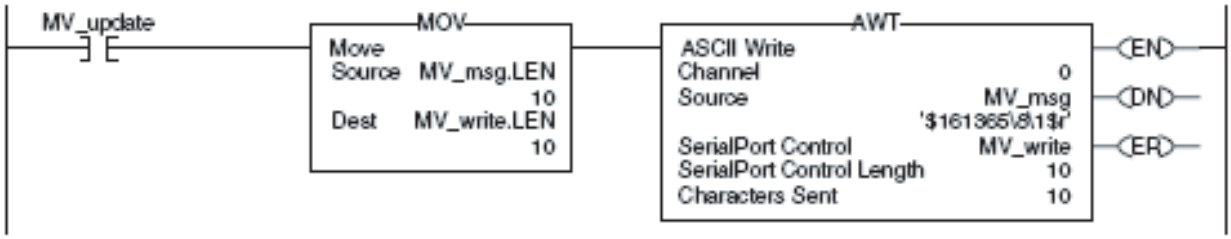


Структурированный текст

```
osri_1.InputBit  :=  temp_low;
OSRI(osri_1);
      IF  (osri_1.OutputBit)  THEN
      tring[2],temp_low_write);
END_IF;
```

Пример 2: Когда устанавливается *MV_update*, инструкция AWT отправляет символы в *MV_msg*. Поскольку количество символов в *MV_msg* меняется, цепочка, в первую очередь, пересылает длину строки (*MV_msg*.LEN) в Serial Port Control Length инструкции AWT (*MV_write*.LEN). В *MV_msg*, *$16* считается за один символ. Это шестнадцатеричный код для символа Ctrl-V.

Релейная логика



Структурированный текст

```
osri_1.InputBit  :=  MV_update;
OSRI(osri_1);
IF  (osri_1.OutputBit)  THEN
      MV_write.LEN  :=  Mv_msg.LEN;
      AWT(0,MV_msg,MV_write);
END_IF;
```

Примечания:

Строковые инструкции ASCII (CONCAT, DELETE, FIND, INSERT, MID)

Введение

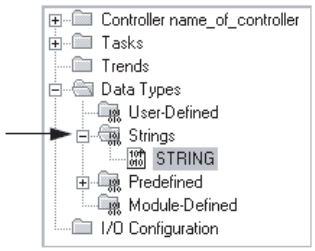
Используйте строковые инструкции ASCII для изменения и создания строк символов ASCII.

Если вы хотите:	Например:	Используйте эту инструкцию:	Имеющуюся в этих языках:	См. стр.
добавить символы в конец строки	добавить символы завершения или разделения к строку	CONCAT	релейная логика структурированный текст	17-3
уничтожить символы в строке	удалить заголовок или управляющие символы из строки	DELETE	релейная логика структурированный текст	17-5
определить начальный символ подстроки	разместить группу символов в пределах строки	FIND	релейная логика структурированный текст	17-7
вставить символы в строку	создать строку, которая использует переменные	INSERT	релейная логика структурированный текст	17-9
извлечь символы из строки	извлечь информацию из штрихового кода	MID	релейная логика структурированный текст	17-11

Для сравнения и преобразования символов ASCII вы можете использовать также следующие инструкции:

Если вы хотите:	Используйте эту инструкцию:	См. стр.
сравнить строку с другой строкой	CMP	4-2
узнать, что символы равны заданным символам	EQU	4-7
узнать, что символы не равны заданным символам	NEQ	4-38
узнать, что символы равны или больше, чем заданные символы	GEQ	4-11
узнать, что символы больше, чем заданные символы	GRT	4-15
узнать, что символы равны или меньше, чем заданные символы	LEQ	4-19
узнать, что символы меньше, чем заданные символы	LES	4-23
перегруппировать байты тега INT, DINT или REAL	SWPB	6-19
найти строку в массиве строк	FSC	7-19
преобразовать символы в значение SINT, INT, DINT или REAL	STOD	18-4
преобразовать символы в значение REAL	STOR	18-6
преобразовать значение SINT, INT, DINT или REAL в строку символов ASCII	DTOS	18-8
преобразовать значение REAL в строку символов ASCII	RTOS	18-10

Строковые типы данных



Вы сохраняете символы ASCII в тегах, которые используют строковый тип данных.

- Вы можете использовать строковый тип данных по умолчанию STRING. Он позволяет хранить до 82 символов.
- Вы можете создать новый строковый тип данных, который будет хранить меньше или больше символов.

Чтобы создать новый строковый тип данных, обратитесь к документу *«Общие процедуры контроллера Logix5000»*, публикация 1756-PM001.

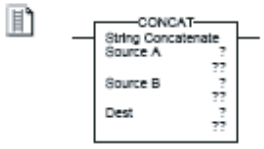
Каждый строковый тип данных содержит следующие элементы:

Имя:	Тип данных:	Описание:	Примечания:
LEN	DINT	количество символов в строке	LEN автоматически обновляется всякий раз, когда вы: • используете для ввода символов диалоговое окно String Browser, • используете инструкции, которые считывают, преобразуют или обрабатывают строку. LEN указывает длину текущей строки. Член DATA может содержать дополнительные, старые символы, которые не включаются в подсчет LEN.
DATA	массив SINT	символы ASCII строки	• Чтобы иметь доступ к символам строки, обратитесь к имени тега. Например, чтобы получить доступ к символам тега *string_1*, введите *string_1*. • Каждый элемент массива DATA содержит один символ. • Вы можете создавать новые строковые типы данных, которые сохраняют меньше или больше символов.

String Concatenate (CONCAT) (Присоединение к строке)

Инструкция CONCAT добавляет символы в конец строки.

Операнды:



Релейная логика

:

Операнд:	Тип:	Формат:	Введите:	Примечания:
Source A	строка	тег	тег, который содержит исходные символы	Строковые типы данных: • строковый тип данных по умолчанию STRING. • любой новый строковый тип данных, который вы создадите.
Source B	строка	тег	тег, который содержит конечные символы	
Destination	строка	тег	тег для хранения результата	

Структурированный текст

```
CONCAT(SourceA,SourceB,
    Dest);
```

Операнды такие же, как и операнды для инструкции CONCAT в релейной логике.

Описание:

Инструкция CONCAT объединяет символы в Source A (источнике A) с символами в Source B (источнике B) и помещает результат в Destination (приемник).

- Символы из Source A идут первыми, за ними следуют символы из Source B.
- Если Source A и Destination не являются одним и тем же тегом, Source A сохраняется без изменений.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

Тип:	Код:	Причина:	Способ устранения:
4	51	Значение LEN строкового тега больше, чем размер DATA строкового тега.	1. Проверьте, чтобы никакая инструкция не осуществляла запись в член LEN строкового тега. 2. В значение LEN введите количество символов, содержащихся в строке.

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция сцепляет строки.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Для того чтобы инициировать сообщение на терминале MessageView, контроллер должен отправить строку ASCII, которая содержит номер сообщения и номер узла. *String_1* содержит номер сообщения. Когда *add_node* устанавливается, инструкция CONCAT добавляет символы из *node_num_ascii* (номер узла) в конец символов строки *string_1* и затем сохраняет результат в *msg*.

Релейная логика

Структурированный текст

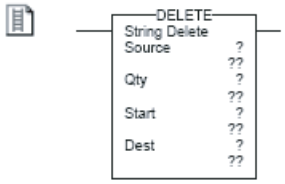
```
IF  add_node  THEN
      CONCAT(string_1,node_num_ascii,msg);
      add_node  :=  0;
END_IF;
```

String Delete (DELETE) (Удаление строки)

Инструкция DELETE удаляет символы ASCII из строки.

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Введите:	Примечания:
Source	строка	тег	тег, содержащий строку, из которой вы хотите удалить символы	Строковые типы данных: • строковый тип данных по умолчанию STRING • любой новый строковый тип данных, который вы создадите.
Quantity	SINT INT DINT	непосред-ственный тег	количество символов для удаления	Start плюс Quantity должно быть меньше или равно размеру DATA Source.
Start	SINT INT DINT	непосред-ственный тег	позицию первого символа для удаления	Введите число в диапазоне между 1 и размером DATA Source.
Destination	строка	тег	тег для хранения результата	

Структурированный текст

```
DELETE(Source,Qty,Start,
    Dest);
```

Операнды такие же, как и операнды для инструкции DELETE в релейной логике.

Описание:

Инструкция DELETE удаляет (уничтожает) группу символов из Source (источника) и размещает оставшиеся символы в Destination (приемнике).

- Позиция Start и Quantity определяют символы для удаления.
- Если Source и Destination не являются одним и тем же тегом, Source сохраняется без изменений.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

Тип:	Код:	Причина:	Способ устранения:
4	51	Значение LEN строкового тега больше, чем размер DATA строкового тега.	1. Проверьте, чтобы никакая инструкция не осуществляла запись в член LEN строкового тега. 2. В значение LEN введите количество символов, содержащихся в строке.
4	56	Недопустимые значения Start или Quantity.	1. Проверьте, чтобы значение Start находилось в диапазоне между 1 и размером DATA Source. 2. Проверьте, чтобы значение Start плюс Quantity было меньше или равно размеру DATA Source.

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция удаляет заданные символы.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Информация ASCII из терминала содержит символ заголовка. После того, как контроллер считывает данные (бит *term_read*.EM устанавливается), инструкция DELETE удаляет символ заголовка.

Релейная логика

Структурированный текст

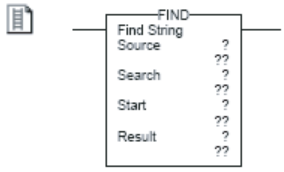
```
IF  term_read.EM  THEN
      DELETE(term_input,1,1,term_text);
      term_read.EM  :=  0;
END_IF;
```

Find String (FIND) (Поиск строки)

Инструкция FIND определяет начальную позицию заданной строки в пределах другой строки.

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Введите:	Примечания:
Source	строка	тег	строку, в которой осуществляется поиск	Строковые типы данных: • строковый тип данных по умолчанию STRING • любой новый строковый тип данных, который вы создадите.
Search	строка	тег	искомую строку	
Start	SINT INT DINT	непосред-ственный тег	позицию в Source для начала поиска	Введите число в диапазоне между 1 и размером DATA Source.
Result	SINT INT DINT	тег	тег для хранения начальной позиции искомой строки	

Структурированный текст

```
FIND(Source,Search,Start,
     Result);
```

Операнды такие же, как и операнды для инструкции FIND в релейной логике, описанной выше.

Описание:

Инструкция FIND производит поиск в строке Source (источник) на предмет наличия строки Search (искомая). Если инструкция находит строку Search, Result (результат) показывает начальную позицию строки Search в строке Source.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

Тип:	Код:	Причина:	Способ устранения:
4	51	Значение LEN строкового тега больше, чем размер DATA строкового тега.	1. Проверьте, чтобы никакая инструкция не осуществляла запись в член LEN строкового тега. 2. В значение LEN введите количество символов, содержащихся в строке.
4	56	Недопустимое значение Start.	Проверьте, чтобы значение Start находилось в диапазоне между 1 и размером DATA Source.

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция ищет заданные символы.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Сообщение от терминала MessageView содержит несколько порций информации. Символ «обратный слеш» [\ ] разделяет порции информации. Чтобы определить порцию информации, инструкция FIND ищет символ «обратный слеш» и записывает его позицию в *find_pos*.

Релейная логика

Структурированный текст

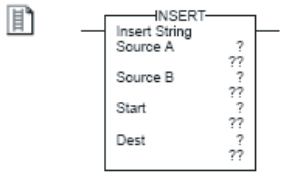
```
IF  MV_read.EM  THEN
      FIND(MV_msg,find,1,find_pos);
      MV_read.EM  :=  0;
END_IF;
```

Insert String (INSERT) (Вставить строку)

Инструкция INSERT добавляет символы ASCII в заданную позицию внутри какой-либо строки.

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Введите:	Примечания:
Source A	строка	тег	строку, в которую будут добавляться символы	Строковые типы данных: • строковый тип данных по умолчанию STRING • любой новый строковый тип данных, который вы создадите.
Source B	строка	тег	строку, содержащую символы для добавления	
Start	SINT INT DINT	непосред-ственный тег	позицию в Source A для добавления символов	Введите число в диапазоне между 1 и размером DATA Source (источника).
Result	строка	тег	строку для хранения результата	

Структурированный текст

```
INSERT(SourceA,SourceB,
    Start,Dest);
```

Операнды такие же, как и операнды для инструкции INSERT в релейной логике.

Описание:

Инструкция INSERT добавляет символы из Source B (источника B) в заданную позицию внутри Source A (источника A) и помещает результат в Destination (приемник):

- Start (старт) определяет позицию в Source A куда будет добавлен Source B.
- Если Source A и Destination не являются одним и тем же тегом, Source A сохраняется без изменений.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

Тип:	Код:	Причина:	Способ устранения:
4	51	Значение LEN строкового тега больше, чем размер DATA строкового тега.	1. Проверьте, чтобы никакая инструкция не осуществляла запись в член LEN строкового тега. 2. В значение LEN введите количество символов, содержащихся в строке.
4	56	Недопустимое значение Start.	Проверьте, чтобы значение Start находилось в диапазоне между 1 и размером DATA Source.

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция вставляет заданные символы.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Когда устанавливается *temp_high*, инструкция INSERT добавляет символы из *string_2* в позицию 2 в *string_1* и помещает результат в *string_3*:

Релейная логика

Структурированный текст

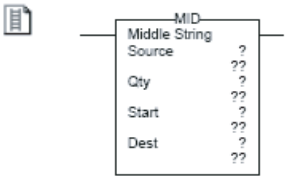
```
IF  temp_high  THEN
      INSERT(string_1,string_2,2,string_3);
      temp_high  :=  0;
END_IF;
```

Middle String (MID) (Средняя строка)

Инструкция MID копирует заданное количество символов ASCII из строки и сохраняет их в другой строке.

Операнды:

Релейная логика:



Операнд:	Тип:	Формат:	Введите:	Примечания:
Source	строка	тег	строку, из которой будут копироваться символы	Строковые типы данных: • строковый тип данных по умолчанию STRING • любой новый строковый тип данных, который вы создадите.
Quantity	SINT INT DINT	непосред-ственный тег	количество символов для копирования	Start плюс Quantity должно быть меньше или равно размеру DATA Source.
Start	SINT INT DINT	непосред-ственный тег	позицию первого символа для копирования	Введите число в диапазоне между 1 и размером DATA Source.
Destination	строка	тег	строку, в которую будут копироваться символы	

Структурированный текст

```
MID(Source,Qty,Start,
    Dest);
```

Операнды такие же, как и операнды для инструкции MID в релейной логике.

Описание:

Инструкция MID копирует группу символов из Source (источника) и помещает результат в Destination (приемнике).

- Позиции Start (старт) и Quantity (количество) определяют символы для копирования.
- Если Source и Destination не являются одним и тем же тегом, Source сохраняется без изменений.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

Тип:	Код:	Причина:	Способ устранения:
4	51	Значение LEN строкового тега больше, чем размер DATA строкового тега.	1. Проверьте, чтобы никакая инструкция не осуществляла запись в член LEN строкового тега. 2. В значение LEN введите количество символов, содержащихся в строке.
4	56	Недопустимые значения Start или Quantity.	1. Проверьте, чтобы значение Start находилось в диапазоне между размером DATA Source. 2. Проверьте, чтобы значение Start плюс Quantity было меньше или равно размеру DATA Source.

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки – «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки – «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция копирует заданные символы из строки и сохраняет их в другой строке.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: На конвейере обработки багажа в аэропорту каждый чемодан получает штриховой код. Символы 9 - 17 штрихового кода - это номер рейса и аэропорт назначения. После считывания штрихового кода (*bag_read*.EM устанавливается) инструкция MID копирует номер рейса и аэропорт назначения в строку *bag_flt_and_dest*.

Релейная логика

Структурированный текст

```
IF bag_read.EM THE
     NMID(bar_barcode,9,9,bag_flt_and_dest);
     bag_read.EM := 0;
END_IF;
```

Инструкции преобразования ASCII (STOD, STOR, DTOS, RTOS, UPPER, LOWER)

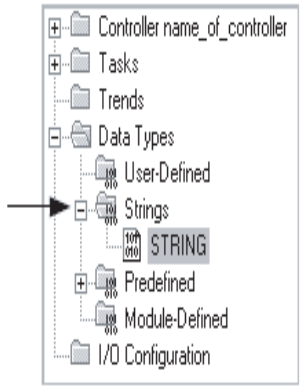
Введение

Используйте инструкции преобразования ASCII для изменения формата данных.

Если вы хотите:	Например:	Используйте эту инструкцию:	Имеющуюся в этих языках:	См. стр.:
преобразовать представление ASCII целого значения в формат SINT, INT, DINT или REAL	преобразовать значение, полученное от весов или другого устройства ASCII, в целое значение, так, чтобы его можно было использовать в программе	STOD	релейная логика структурированный текст	18-4
преобразовать представление ASCII значения с плавающей точкой в значение REAL	преобразовать значение, полученное от весов или другого устройства ASCII, в значение типа REAL, так, чтобы его можно было использовать в программе	STOR	релейная логика структурированный текст	18-6
преобразовать значение SINT, INT, DINT или REAL в строку символов ASCII	преобразовать переменную в строку ASCII, чтобы отправить ее на терминал MessageView	DTOS	релейная логика структурированный текст	18-8
преобразовать значение типа REAL в строку символов ASCII	преобразовать переменную в строку ASCII, чтобы отправить ее на терминал MessageView	RTOS	релейная логика структурированный текст	18-10
преобразовать буквы в строке символов ASCII в прописные	преобразовать символы, введенные оператором в прописные, чтобы можно было организовать их поиск в массиве	UPPER	релейная логика структурированный текст	18-12
преобразовать буквы в строке символов ASCII в строчные	преобразовать символы, введенные оператором в строчные, чтобы можно было организовать их поиск в массиве	LOWER	релейная логика структурированный текст	18-14

Для обработки и сравнения символов ASCII вы можете использовать также следующие инструкции:

Если вы хотите:	**Используйте эту инструкцию:**	**См. стр.:**
добавить символы в конец строки	CONCAT	17-3
удалить символы из строки	DELETE	17-5
определить начальный символ подстроки	FIND	17-7
вставить символы в строку	INSERT	17-9
извлечь символы из строки	MID	17-11
перегруппировать байты тега INT, DINT или REAL	SWPB	6-19
сравнить строку с другой строкой	CMP	4-2
узнать, что символы равны заданным символам	EQU	4-7
узнать, что символы не равны заданным символам	NEQ	4-38
узнать, что символы равны или больше, чем заданные символы	GEQ	4-11
узнать, что символы больше, чем заданные символы	GRT	4-15
узнать, что символы равны или меньше, чем заданные символы	LEQ	4-19
узнать, что символы меньше, чем заданные символы	LES	4-23
найти строку в массиве строк	FSC	7-19



Строковые типы данных

Вы сохраняете символы ASCII в тегах , которые используют строковый тип данных.

- Вы можете использовать строковый тип данных по умолчанию STRING. Он позволяет хранить до 82 символов.
- Вы можете создать новый строковый тип данных, который будет хранить меньше или больше символов.

Чтобы создать новый строковый тип данных обращайтесь к документу «Общие процедуры контроллера Logix5000», публикация 1756-PM001.

Каждый строковый тип данных содержит следующие элементы:

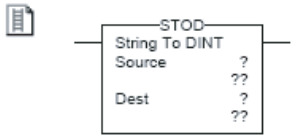
Имя:	Тип данных:	Описание:	Примечание:
LEN	DINT	количество символов в строке	LEN автоматически обновляется всякий раз, когда вы: • используете для ввода символов диалоговое окно String Browser, • используете инструкции, которые считывают, преобразуют или обрабатывают строку. LEN указывает длину текущей строки. Элемент DATA может содержать дополнительные, старые символы, которые не включаются в подсчет LEN.
DATA	массив SINT	символы ASCII строки	• Чтобы иметь доступ к символам строки, обратитесь к имени тега. Например, чтобы получить доступ к символам тега *string_1*, введите *string_1*. • Каждый элемент массива DATA содержит один символ. • Вы можете создавать новые строковые типы данных, которые сохраняют меньше или больше символов.

String to DINT (STOD) (Преобразование строки ASCII в DINT)

Инструкция STOD преобразует представление ASCII целого числа в целое число или значение REAL.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Ввод:	Примечания:
Source	строка	тег	строка, содержащая значение ASCII	Строковые типы данных: • строковый тип данных по умолчанию STRING • любой новый строковый тип данных, который вы создадите
Destination	SINT INT DINT REAL	тег	тег для хранения целочисленного значения	Если значение Source - число с плавающей точкой, то инструкция преобразует только целую часть этого числа (независимо от типа данных приемника).

Структурированный текст

```
STOD(Source,Dest);
```

Операнды такие же, как и операнды для инструкции STOD в релейной логике.

Описание:

Инструкция STOD преобразует Source (источник) в целое число и помещает результат в Destination (приемник).

- Инструкция преобразует положительные и отрицательные числа.
- Если строка Source содержит нецифровые символы, инструкция STOD преобразует первый непрерывный набор цифр:

– Инструкция пропускает начальные управляющие символы или нецифровые символы (за исключением знака минус перед цифрой).

– Если строка содержит несколько групп цифр, разделенных разделителем (напр., /), инструкция преобразует только первую группу цифр.

Арифметические флаги состояния:

Арифметические флаги состояния не затрагиваются

Условия ошибки:

Тип:	Код:	Причина:	Способ устранения:
4	51	Значение LEN строкового тега больше, чем размер DATA строкового тега.	1. Проверьте, чтобы никакая инструкция не осуществляла запись в член LEN строкового тега. 2. В LEN введите количество символов, содержащихся строке.
4	53	Выходное число превосходит пределы типа данных приемника.	Либо: • Уменьшите размер значения ASCII. • Увеличьте размер типа данных для приемника.

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	S:C устанавливается. Destination сбрасывается. Инструкция преобразует Source. Если результат - ноль, устанавливается S:Z	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Когда устанавливается *MV_readEM*, инструкция STOD преобразует первый набор цифровых символов из *MV_msg* в целое число. Инструкция пропускает начальный символ управления ($06) и останавливается у разделителя (\).

Релейная логика

Структурированный текст

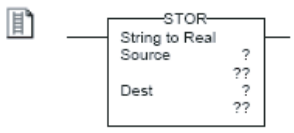
```
IF  MV_read.EM  THEN
      STOD(MV_msg,MV_msg_nmbr);
      MV_read.EM  :=  0;
END_IF;
```

String To REAL (STOR) Преобразование строки ASCII в REAL

Инструкция STOR преобразует представление ASCII значения с плавающей точкой в значение REAL.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Ввод:	Примечания:
Source	строка	тег	тег, содержащий значение ASCII	Строковые типы данных: • строковый тип данных по умолчанию STRING. • любой новый строковый тип данных, который вы создадите.
Destination	REAL	тег	тег для хранения значения REAL	

Структурированный текст

```
STOR(Source,Dest);
```

Операнды такие же, как и операнды для инструкции STOR в релейной логике.

Описание:

Инструкция STOR преобразует Source (источник) в значение REAL и помещает результат в Destination (приемник).

- Инструкция преобразует положительные и отрицательные числа.
- Если строка Source содержит нецифровые символы, инструкция STOR преобразует первый непрерывный набор цифр, включая десятичную точку [.]:

– Инструкция пропускает начальные управляющие символы или нецифровые символы (за исключением знака минус перед цифрой).

– Если строка содержит несколько групп цифр, разделенных разделителем (напр., /), инструкция преобразует только первую группу цифр.

Арифметические флаги состояния:

Арифметические флаги состояния не затрагиваются

Условия ошибки:

Тип:	Код:	Причина:	Способ устранения:
4	51	Значение LEN строкового тега больше, чем размер DATA строкового тега.	1. Проверьте, чтобы никакая инструкция не осуществляла запись в член LEN строкового тега. 2. В LEN введите количество символов, содержащихся строке.
4	53	Выходное число превосходит пределы типа данных приемника	Либо: • Уменьшите размер значения ASCII. • Увеличьте размер типа данных для приемника.

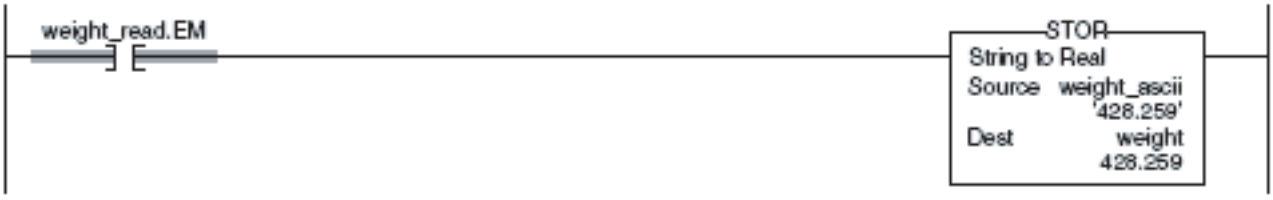
Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn установлен всегда. Инструкция выполняется.
выполнение инструкции	S:C устанавливается. Destination сбрасывается. Инструкция преобразует Source. Если результат - ноль, устанавливается S:Z	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: После считывания значения веса с весов (устанавливается *weight_read.EM*), инструкция STOR преобразует цифровые символы из *weight_ascii* в значение REAL.

Возможно, вы увидите небольшое различие между дробными частями Source и Destination.

Релейная логика



Структурированный текст

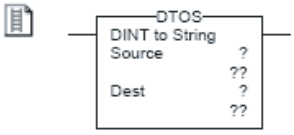
```
IF  weight_read.EM  THEN
      STOR(weight_ascii,weight);
      weight_read.EM  :=  0;
END_IF;
```

DINT to String (DTOS) (Преобразование значения DINT в строку ASCII)

Инструкция DTOS выдает представление ASCII какого либо числа.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Ввод:	Примечания:
Source	SINT INT DINT REAL	тег	тег, содержащий значение	Если Source имеет тип REAL, инструкция преобразует его в число типа DINT. См. преобразование REAL в целое число на стр. A-6.
Destination	строка	тег	тег для хранения значения ASCII	Строковые типы данных: • строковый тип данных по умолчанию STRING. • любой новый строковый тип данных, который вы создадите

Структурированный текст

```
DTOS(Source,Dest);
```

Операнды такие же, как и операнды для инструкции DTOS в релейной логике.

Описание:

Инструкция DTOS преобразует Source (источник) в строку символов ASCII и помещает результат в Destination (приемник).

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

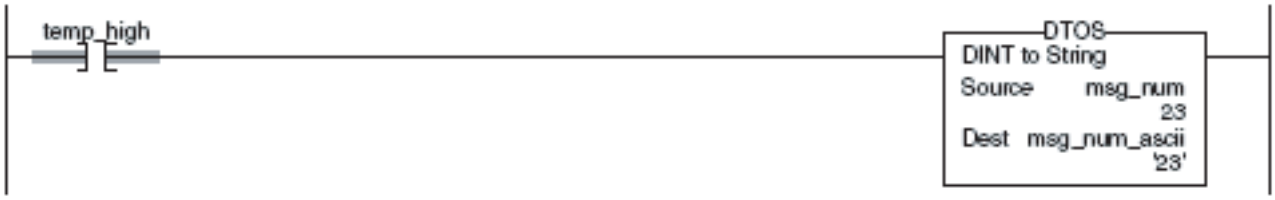
Тип:	Код:	Причина:	Способ устранения:
4	51	Значение LEN строкового тега больше, чем размер DATA строкового тега.	1. Проверьте, чтобы никакая инструкция не осуществляла запись в член LEN строкового тега. 2. В LEN введите количество символов, содержащихся в строке.
4	52	Выходная строка превосходит размер приемника	Создайте новый строковый тип данных, достаточно большой, чтобы уместилась строка выхода. Используйте этот новый строковый тип данных для приемника.

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция преобразует Source.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Когда устанавливается *temp_high*, инструкция DTOS преобразует значение из *msg_num* в строку символов ASCII и помещает результат в *msg_num_ascii*. Последующие цепочки вставляют *msg_num_ascii* в другие строки или присоединяют *msg_num_ascii* к другим строкам для получения полного сообщения на экране терминала.

Релейная логика



Структурированный текст

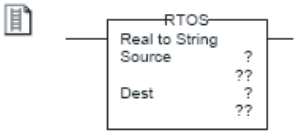
```
IF  temp_high  THEN
      DTOS(msg_num,msg_num_ascii);
      temp_high  :=  0;
END_IF;
```

REAL to String (RTOS) (Преобразование данных типа REAL в строку ASCII)

Инструкция RTOS выдает представление ASCII какого-либо значения REAL.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Ввод:	Примечания:
Source	REAL	тег	тег, содержащий значение REAL	
Destination	строка	тег	тег для хранения представления ASCII	Строковые типы данных: • строковый тип данных по умолчанию STRING. • любой новый строковый тип данных, который вы создадите

```
RTOS(Source,Dest);
```

Структурированный текст

Операнды такие же, как и операнды для инструкции RTOS в реленой логике.

Описание:

Инструкция RTOS преобразует Source (источник) в строку символов ASCII и помещает результат в Destination (приемник).

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

Тип:	Код:	Причина:	Способ устранения:
4	51	Значение LEN строкового тега больше, чем размер DATA строкового тега.	1. Проверьте, чтобы никакая инструкция не осуществляла запись в член LEN строкового тега. 2. В LEN введите количество символов, содержащихся строке.
4	52	Выходная строка превосходит размер приемника.	Создайте новый строковый тип данных, достаточно большой, чтобы уместилась строка выхода. Используйте этот новый строковый тип данных для приемника.

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применомо
входное условие цепочки «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применомо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция преобразует Source.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Когда устанавливается *send_data*, инструкция RTOS преобразует значение из *data_1* в строку символов ASCII и помещает результат в *data_1_ascii*. Последующие цепочки вставляют *data_1_ascii* в другие строки или присоединяют *data_1_ascii* к другим строкам для получения полного сообщения на экране терминала.

Возможно, вы увидите небольшое различие между дробными частями Source и Destination.

Релейная логика

Структурированный текст

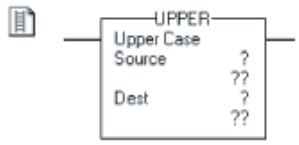
```
IF  send_data  THEN
      RTOS(data_1,data_1_ascii);
      send_data  :=  0;
END_IF;
```

Upper Case (UPPER) (Верхний регистр)

Инструкция UPPER преобразует символы алфавита в строке в прописные символы.

Операнды:

Релейная логика



Операнд:	Тип:	Формат:	Описание:
Source	строка	тег	тег, содержащий символы, которые вы хотите преобразовать в прописные
Destination	строка	тег	тег для хранения прописных символов

```
UPPER(Source,Dest);
```

Структурированный текст

Операнды такие же, как и операнды для инструкции UPPER в релейной логике.

Описание:

Инструкция UPPER преобразует в прописные все буквы из Source (источника) и помещает результат в Destination (приемник).

- Символы ASCII чувствительны к регистру. Код верхнего регистра “A” ($41) *не* совпадает с кодом нижнего “a” ($61).
- Если оператор напрямую вводит символы ASCII, следует преобразовать эти символы в прописные или строчные перед их сравнением.

Любые символы в строке Source, которые не являются буквами, сохраняются без изменений.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на «ложь».	не применимо
входное условие цепочки «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция преобразует символы Source в символы верхнего регистра.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Для поиска информации о заданном элементе, оператор вводит его номер по каталогу в терминал ASCII. После этого контроллер считывает этот ввод с терминала (устанавливается *terminal_readEM*), а инструкция UPPER преобразует символы из *catalog_number* в прописные и сохраняет результат в *catalog_number_upper_case.* Последующая цепочка ищет в массиве символы, совпадающие с теми, которые находятся в *catalog_number_upper_case.*

Релейная логика

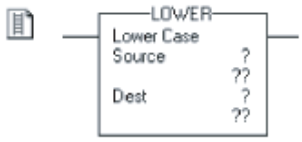
Структурированный текст

```
IF  terminal_read.EM  THEN
      UPPER(catalog_number,catalog_number_upper_case);
      terminal_read.EM  :=  0;
END_IF;
```

Lower Case (LOWER) (Нижний регистр)

Инструкция LOWER преобразует символы алфавита в строчные.

Операнды:



Релейная логика

Операнд:	Тип:	Формат:	Описание:
Source	строка	тег	тег, содержащий символы, которые вы хотите конвертировать в строчные
Destination	строка	тег	тег для хранения строчных символов

```
LOWER(Source,Dest);
```

Структурированный текст

Операнды такие же, как и операнды для инструкции LOWER в релейной логике.

Описание:

Инструкция LOWER преобразует в строчные все буквы из Source (источника) и помещает результат в Destination (приемник).

- Символы ASCII чувствительны к регистру. Код верхнего регистра “A” ($41) *не* совпадает с кодом нижнего “a” ($61).
- Если оператор напрямую вводит символы ASCII, следует преобразовать эти символы в прописные или строчные перед их сравнением.

Любые символы в строке Source, которые не являются буквами, сохраняются без изменений.

Арифметические флаги состояния:

не затрагиваются

Условия ошибки:

отсутствуют

Выполнение:

Условие:	Действие релейной логики:	Действие структурированного текста:
предварительное сканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.
входное условие цепочки «ложь»	Выходное условие цепочки устанавливается на«ложь».	не применимо
входное условие цепочки «истина»	Инструкция выполняется. Выходное условие цепочки устанавливается на «истина».	не применимо
EnableIn устанавливается	не применимо	EnableIn всегда установлен. Инструкция выполняется.
выполнение инструкции	Инструкция преобразует символы Source в символы нижнего регистра.	
постсканирование	Выходное условие цепочки устанавливается на «ложь».	Никакого действия не производится.

Пример: Для поиска информации о заданном элементе, оператор вводит его номер по каталогу в терминал ASCII. После этого контроллер считывает этот ввод с терминала (устанавливается *terminal_readEM*), а инструкция LOWER преобразует символы из *item_number* в строчные и сохраняет результат в *item_number_lower_case*. Последующая цепочка ищет в массиве символы, совпадающих с теми, которые находятся в *item_number_lower_case*.

Релейная логика

Структурированный текст

```
IF  terminal_read.EM  THEN
      LOWER(item_number,item_number_lower_case);
      terminal_read.EM  :=  0;
END_IF;
```

Примечания:

Общие атрибуты

Введение

В этом приложении описываются атрибуты, общие для инструкций контроллеров Logix.

За информацией о:	Обращайтесь на стр. :
Непосредственные значения	A-1
Преобразованиях данных	A-1

Непосредственные значения

Когда вы вводите непосредственное значение (постоянную) в десятичном формате, (например, -2, 3), контроллер сохраняет это значение, используя 32 разряда. Если вы вводите в другой системе счисления, отличной от десятичной, такой, как двоичная или шестнадцатиричная, и не задаете все 32 разряда, контроллер помещает ноль в разряды, которые вы не задали (заполнение нулями).

ПРИМЕР Заполнение нулями непосредственных значений

Если в вводите:	Контроллер сохраняет:
-1	16#ffff ffff (-1)
16#ffff (-1)	16#0000 ffff (65535)
8#1234 (668)	16#0000 029c (668)
2#1010 (10)	16#0000 000a (10)

Преобразования данных

Преобразование данных происходит тогда, когда вы сочетаете разные типы данных при программировании:

При программировании в:	Преобразования могут иметь место когда:
релейной логике	смешиваются типы данных для параметров в одной инструкции
функциональном блоке	вы связываете два параметра разных типов

Инструкции выполняются быстрее и требуют меньший объем памяти, если все операнды:

- имеют одинаковый тип данных
- имеют оптимальный тип данных:
  - В разделе “Операнды” описания каждой инструкции в данном руководстве, **жирный** шрифт типов данных означает, что это оптимальный тип данных.
  - Обычно оптимальными типами данных являются DINT и REAL.
  - Большинство инструкций функционального блока поддерживают один тип данных (оптимальный) для своих операндов.

Если вы смешиваете типы данных и используете теги с неоптимальными типами данных, контроллер преобразует данные в соответствии со следующими правилами.

- Существуют ли операнды, имеющие тип данных REAL?

Если:	**Тогда входные данные (т.е. источник, тег в выражении, предел) преобразуются в:**
да	REAL
нет	DINT

- После выполнения инструкции, если это необходимо, результат (значение DINT или REAL) преобразуется в тип, определяемый операндом destination.

Вы не можете задать тег BOOL в инструкции, которая оперирует с целочисленными типами данных или типом данных REAL.

Поскольку преобразование данных занимает дополнительное время и память, вы можете повысить эффективность программы:

- используя один тип данных внутри инструкции
- минимизировав использование типов данных SINT или INT

Другими словами, используйте в ваших инструкциях теги DINT, REAL совместно с непосредственными значениями.

Следующие разделы объясняют, как преобразуются данные, когда вы используете теги SINT или INT или когда вы смешиваете типы данных.

SINT или INT в DINT

Для тех инструкций, которые преобразуют значения SINT или INT в DINT, разделы “Операнды” данного руководства определяют способ преобразования.

Этот метод преобразования:	**Преобразует данные размещая:**
Дополнительный знаковый разряд	значение самого левого разряда (знак значения) в каждую позицию разряда слева от значащих разрядов, пока не будут заполнены все 32 разряда.
Заполнение нулями	нули в каждый разряд слева от значащих разрядов, пока не будут заполнены все 32 разряда.

Следующий пример демонстрирует результат преобразования с использованием дополнительного знакового разряда и заполнения нулями.

Это значение	**2#1111_1111_1111_1111**	**(-1)**
Преобразуется в это значение посредством дополнительного знакового разряда з	2#1111_1111_1111_1111_1111_1111_1111_1111	(-1)
Преобразуется в эту значение посредством заполнения нулем	2#0000_0000_0000_0000_1111_1111_1111_1111	(65535)

Поскольку непосредственные значения всегда заполняются нулями, преобразование значения SINT или INT может вызвать непредсказуемые результаты. В следующем примере результат сравнения - «ложь», потому что Source A (источник A), имеющий тип INT, преобразуется посредством дополнительного знакового разряда, а Source B (источник B), являясь непосредственным значением, заполняется нулем.

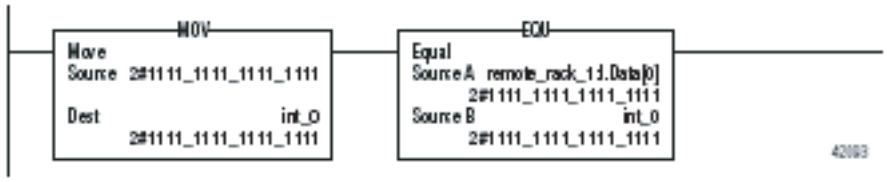
Если вы используете тег SINT или INT и непосредственное значение в инструкции, которая преобразует данные посредством дополнительного знакового разряда, используйте один из следующих методов обработки для непосредственных значений:

- Задавайте любое непосредственное значение в десятичной системе.
- Если вы вводите значение в системе счисления , отличной от десятичной, задавайте все 32 разряда этого непосредственного значения. Чтобы сделать это, введите значение самого левого разряда в каждую позицию, пока не будут заполнены все 32 разряда.
- Задавайте тег для каждого операнда и используйте один и тот же тип данных внутри инструкции. Чтобы присвоить постоянное значение либо:
  - введите его в один из тегов
  - добавьте инструкцию MOV, которая переместит это значение в один из тегов.
- Используйте инструкцию MEQ для проверки только требующихся разрядов.

Следующие примеры показывают две возможности смешивания непосредственных значений и тегов INT. В обоих примерах проверяются разряды модуля ввода/вывода 1771 для определения, все ли разряды заполнены. Поскольку слово входных данных модуля ввода/вывода 1771 является тегом INT, самое простое – использовать 16-ти разрядную константу.

ПРИМЕР Смешивание тега INT с непосредственным значением. Поскольку *remote_rack_1:I.Data[0]* является тегом INT, значение для сверки также вводится как тег INT.

ПРИМЕР Смешивание тега INT с непосредственным значением. Поскольку *remote_rack_1:I.Data[0]* является тегом INT, значение для сверки с ним вначале помещается в *int_0* (также тег INT). Затем инструкция EQU производит сравнение обоих тегов.



Целое число в REAL

Контроллер сохраняет значенияа REAL в формате числа с плавающей точкой одинарной точности стандарта IEEE. Он использует один разряд для знака числа, 23 разряда для основания и восемь разрядов для показателя степени (всего 32). Если вы смешиваете целочисленный тег (SINT, INT, или DINT) и тег REAL как входные данные в одной инструкции, контроллер преобразует целочисленное значение в значение REAL перед выполнением инструкции.

- Значение SINT или INT всегда преобразуется в то же самое значение REAL.
- Значение DINT может и не преобразовываться в REAL:
  - Для значения REAL используются 24 разряда для основания (23 разряда для сохранения, плюс «спрятанный» разряд).
  - Для значения DINT используются 32 разряда (один для знака и 31 для значения).
  - Если для значения DINT требуется более 24 значащих разрядов, *возможно*, оно не будет преобразовано в то же самое значение REAL. Если значение DINT не преобразовано, контроллер округляет его до ближайшего значения REAL, используя 24 значащих разряда.

DINT в SINT или INT

Чтобы преобразовать значение DINT в значение SINT или INT, контроллер усекает верхнюю часть DINT и, если это необходимо, устанавливает флаг состояния переполнения. Следующий пример демонстрирует результат преобразования DINT в SINT или INT.

ПРИМЕР Преобразование DINT в INT и SINT

Это значение DINT:	Преобразуется в это меньшее значение	
16#0001_0081 (65665)	INT:	16#0081 (129)
	SINT:	16#81 (-127)

REAL в целое число

Чтобы преобразовать значение REAL в целочисленное значение, контроллер округляет дробную часть и усекает верхнюю часть целой части. Если данные утеряны, контроллер устанавливает флаг переполнения. Числа округляются следующим образом:

- Числа, отличные от x.5, округляются до ближайшего целого числа.
- Числа X.5 округляются до ближайшего четного числа.

ПРИМЕР Следующий пример демонстрирует результат преобразования значений REAL в значения DINT.

Это значение REAL:	Преобразуется в это значение DINT
-2.5	-2
-1.6	-2
-1.5	-2
-1.4	-1
1.4	1
1.5	2
1.6	2
2.5	2

ВАЖНО! Арифметические флаги состояния устанавливаются на основе хранящихся значений. Инструкции, которые обычно не влияют на арифметические ключи состояния, могут оказать на них влияние, если произойдет преобразование типа из-за смешивания типов данных для параметров инструкции. Процесс преобразования типа устанавливает арифметические ключи состояния.

Атрибуты функционального блока

Введение

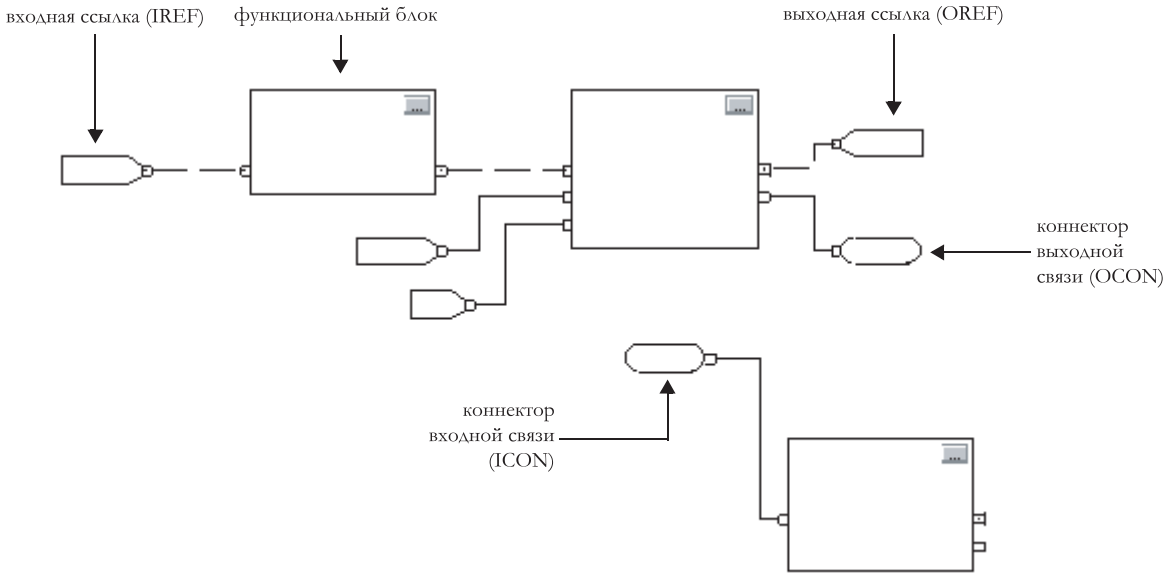
В этом приложении описываются вопросы, свойственные только инструкциям функционального блока. Чтобы убедиться, что вы понимаете, как работают программы на языке функциональных блоков, прочитайте это приложение.

ВАЖНО!

При программировании на языке функциональных блоков, ограничьте диапазон технических единиц до $+/-10^{+/-15}$, поскольку внутренние расчеты с плавающей точкой используют числа с плавающей точкой одинарной точности. Наличие технических единиц вне этого диапазона может приводить к потере точности, если результаты приближаются к границам для величин с плавающей точкой одинарной точности ($+/-10^{+/-38}$).

Выбор элементов функционального блока

Для управления устройством используйте следующие элементы:



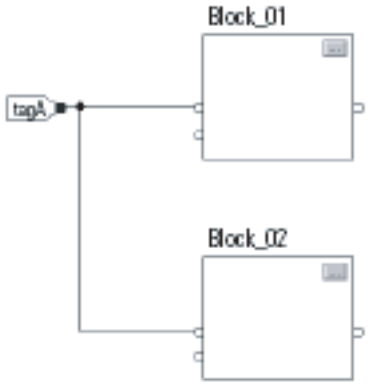
При выборе элементов функционального блока используйте следующую таблицу:

Если вы хотите:	Используется:
получить значение из входного устройства или тега	входную ссылку (input reference (IREF))
отправить значение в выходное устройство или тег	выходную ссылку (output reference (OREF))
выполнить операцию над входным значением (значениями) и выдать выходное значение (значения)	функциональный блок (function block)
передавать данные между функциональными блоками когда они: • на большом расстоянии друг от друга на одном листе • на разных листах в пределах одной процедуры	выходной коннектор связи (output wire connector (OCON)) и входной коннектор связи (input wire connector (ICON))
рассредоточить данные по нескольким точкам в одной процедуре	один выходной коннектор связи (OCON) и несколько входных коннекторов связи (ICON)

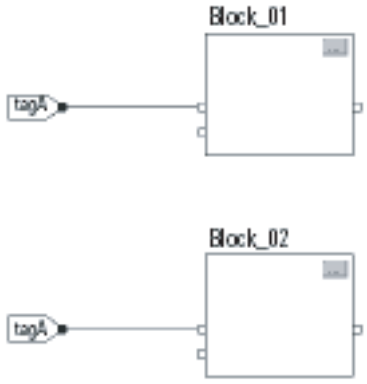
Фиксация данных

Если вы используете элемент IREF для того, чтобы задать исходные данные для какого-либо функционального блока, данные в этом элементе «защелкиваются» для сканирования процедуры этого функционального блока. Элемент IREF «защелкивает» данные для тегов программы и контроллера. Контроллер обновляет все данные в IREF при начале каждого сканирования.

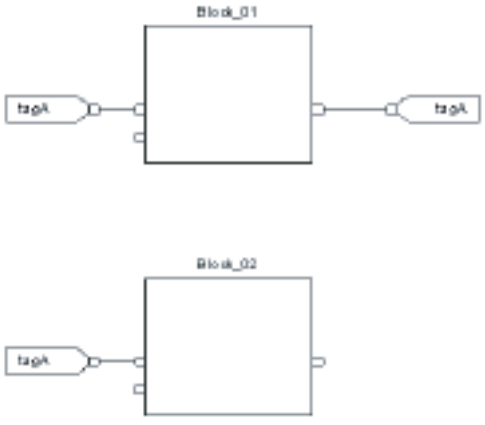
В этом примере значение в tagA сохраняется в начале выполнения процедуры. Это сохраненное значение используется при выполнении блока Block_01. Это же самое значение используется при выполнении блока Block_02. Если значение tagA изменяется во время выполнения процедуры, сохраненное значение tagA в элементе IREF не изменяется до следующего выполнения процедуры.



Этот пример аналогичен представленному выше. Значение tagA сохраняется один раз в начале выполнения процедуры. Процедура использует это значение внутри себя.



Начиная с версии 1 программного обеспечения RSLogix 5000, вы можете использовать один и тот же тег в нескольких элементах IREF и одном элементе OREF одной процедуры. Поскольку значения тегов в элементах IREF защелкиваются при каждом сканировании процедуры, все IREF будут использовать одно и то же значение, даже если OREF будет получать различные значения тегов при выполнении этой процедуры. В этом примере, если tagA имеет значение 25.4, то когда процедура начинает выполнять данное сканирование, это значение сканируется, а Block_01 изменяет значение tagA на 50.9, второй элемент IREF подключенный к Block_02 будет по-прежнему использовать значение 25.4 при выполнении Block_02 на этом сканировании. Новое значение tagA, равное 50.9, не будет использоваться никаким элементом IREF в данной процедуре, пока не начнется следующее сканирование.



Порядок выполнения

Программное обеспечение RSLogix 5000 автоматически определяет порядок выполнения функциональных блоков в процедуре, когда вы:

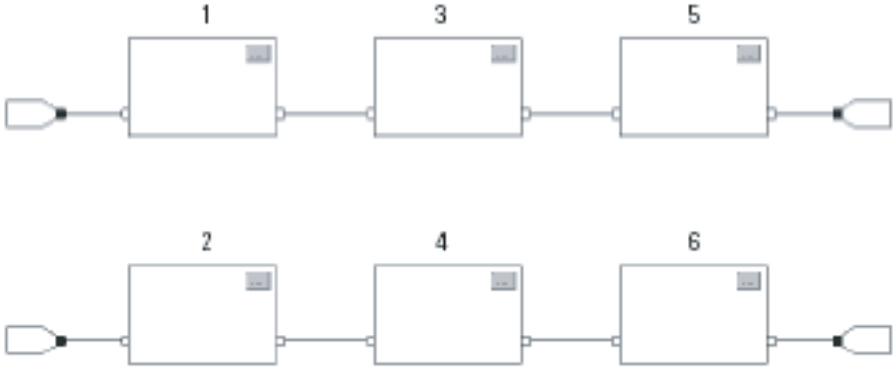
- проверяете процедуру с функциональными блоками,
- проверяете проект, содержащий процедуру на языке функциональных блоков,
- загружаете проект, содержащий процедуру на языке функциональных блоков.

Вы определяете порядок выполнения, связывая функциональные блоки между собой и указывая, если это необходимо, поток данных обратных связей.

Если функциональные блоки не связаны друг с другом, это неважно, какой блок выполняется первым. Между этими блоками нет потока данных.

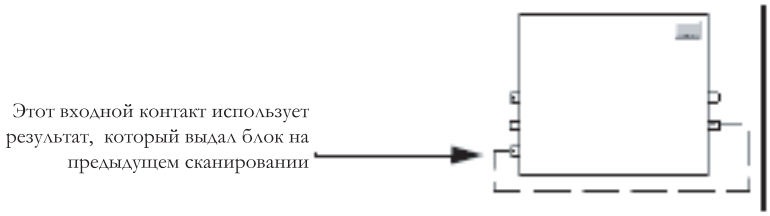
Если вы связали блоки последовательно, порядок выполнения идет от входа к выходу. Входы блока требуют наличия данных до того, как контроллер начинает выполнять этот блок. Например, блок 2 должен выполняться раньше блока 3, потому что блок 2 предоставляет входные данные для блока 3.

Порядок выполнения блоков имеет значение только для блоков, связанных друг с другом. Следующий пример справедлив, поскольку две группы блоков не связаны друг с другом. Блоки в пределах заданной группы выполняются в последовательности, определяемой их положением внутри этой группы.

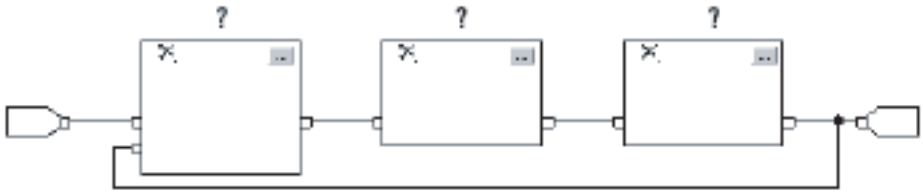


Организация циклов

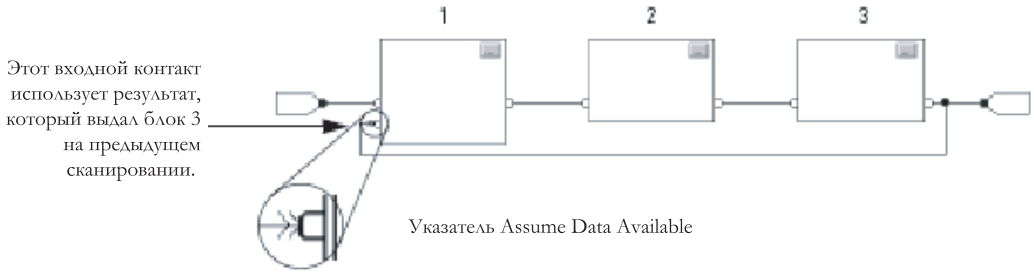
Для создания цикла обратной связи свяжите выводной контакт блока с входным контактом того же блока. Следующий пример справедлив. Цикл содержит один единственный блок и порядок выполнения роли не играет.



Если группа блоков объединена в цикл, то контроллер не может определить какой блок выполнять первым. Другими словами, он не может разрешить цикл.

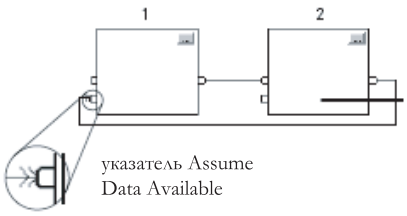
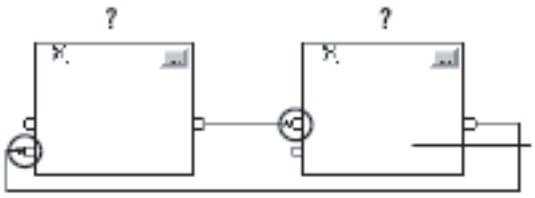


Чтобы определить блок, который будет выполняться первым, пометьте входной контакт, который создает цикл (обратная связь), при помощи указателя Assume Data Available (допустим, что данные имеются). В следующем ниже примере блок 1 использует результат блока 3, который был получен на предыдущем сканировании процедуры.



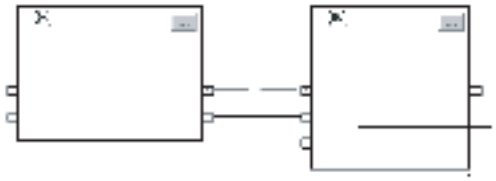
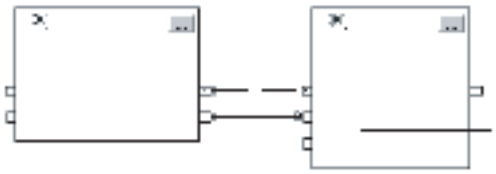
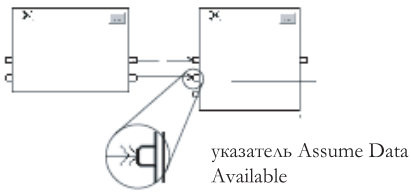
Указатель *Assume Data Available* определяет поток данных внутри цикла. Стрелка указывает, что эти данные служат входом для первого блока цикла.

Не помечайте все связи цикла указателем *Assume Data Available.*

Это правильно	Это неправильно
	
Указатель *Assume Data Available* определяет поток данных внутри цикла.	Контроллер не может разрешить цикл, поскольку все связи используют указатель *Assume Data Available*.

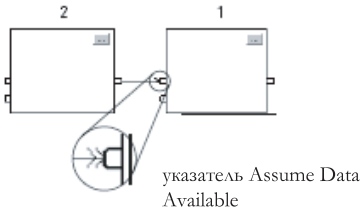
Разрешение потока данных между двумя блоками

Если вы используете две или более связи между двумя блоками, используйте одинаковые указатели потока данных для всех связей между двумя блоками.

Это правильно	Это неправильно
	
Ни та, ни другая связь не использует указатель *Assume Data Available*.	Одна связь использует указатель *Assume Data Available,* в то время как другая нет.
	
Обе связи используют указатель *Assume Data Available.*	

Создание задержки на одно сканирование

Чтобы организовать задержку на одно сканирование между блоками, используйте указатель Assume Data Available. В следующем примере блок 1 выполняется первым. Он использует результат блока, который был получен на предыдущем сканировании процедуры.



Резюме

В итоге, программа на языке функциональных блоков выполняется следующим образом:

1 Контроллер «защелкивает» все значения данных в элементах IREF.

2 Контроллер выполняет функциональные блоки в той последовательности, в которой они связаны.

3 Контроллер записывает результаты в элементы OREF.

Реакция функционального блока на состояние переполнения

В целом, инструкции функционального блока, которые поддерживают предысторию, не обновляют предысторию при наличии значений ±NAN или ±INF при переполнении.

Каждая инструкция реагирует одним из следующих образов на состояние переполнения:

Вариант 1:	**Вариант 2:**	**Вариант 3:**
Блоки выполняют свои алгоритмы и проверяют результат на ±NAN или ±INF. Если имеет место ±NAN или ±INF, блок выдает ±NAN или ±INF.	Блоки с ограничениями на выходные значения выполняют свои алгоритмы и проверяют результат на наличие ±NAN или ±INF. Ограничения выходных значений определяются входными параметрами HighLimit или LowLimit. Если имеет место ±INF, блок выдает ограниченный результат. Если имеет место ±NAN, ограничения выходных значений не используются и блок выдает ±NAN.	Состояние переполнения не применяется. Эти инструкции обычно имеют булев выход.
ALM NTCH	HLL	BAND OSRI
DEDT PMUL	INTG	BNOT RESD
DERV POSP	PI	BOR RTOR
ESEL RLIM	PIDE	BXOR SETD
FGEN RMPS	SCL	CUTD TOFR
HPF SCRV	SOC	D2SD TONR
LDL2 SEL		D3SD
LDLG SNEG		DFF
LPF SRTP		JKFF
MAVE SSUM		OSFI
MAXC TOT		
MINC UPDN		
MSTD		
MUX		

Режимы синхронизации

Следующие инструкции управления процессом поддерживают различные режимы синхронизации.

DEDT	LDLG	RLIM
DERV	LPF	SCRV
HPF	NTCH	SOC
INTG	PI	TOT
LDL2	PIDE	

Существует три режима синхронизации:

<table>
<tr><th>Режим синхронизации:</th><th colspan="2">Описание</th></tr>
<tr><td>периодический (periodic)</td><td colspan="2">Периодический режим является режимом по умолчанию и приемлем для большинства управляющих приложений. Мы рекомендуем вам размещать инструкции, использующие этот режим, в процедурах, которые выполняются в периодической задаче. Приращение времени (DeltaT) для инструкции определяется следующим образом:</td></tr>
<tr><td></td><td>Если эта инструкция выполняется в:</td><td>DeltaT равно</td></tr>
<tr><td></td><td>периодической задаче</td><td>периоду задачи
Период задается в целых значениях миллисекунд (мс). У DeltaT контроллер усекает любую дробную часть периода. Например, если период этой задачи = 10.5 мс, контроллер устанавливает DeltaT = 10 мс.
Если вы хотите использовать дробное значение периода задачи, используйте режим супердискретизации (oversample). При помощи режима супердискретизации вы можете задать параметр OversampleDT равным периоду задачи, включая дробную часть.</td></tr>
<tr><td></td><td>событии или непрерывной задаче</td><td>полному времени после предыдущего выполнения
Контроллер усекает полное время до целого значения в миллисекундах (мс). Например, если полное время = 10.5 мс, контроллер установит DeltaT = 10 мс.</td></tr>
<tr><td></td><td colspan="2">Процесс обновления входных данных должен быть синхронизирован с выполнением задачи или выполняться в 5-10 раз быстрее, чтобы минимизировать ошибку выборки данных между вводом и выполнением инструкции..</td></tr>
<tr><td>супердискретизация (oversample)</td><td colspan="2">В режиме супердискретизации приращением времени (DeltaT), используемым инструкцией, является значение, записанное в параметре OversampleDT этой инструкции. Используйте этот режим, когда инструкция выполняется в событии или непрерывной задаче и процесс ввода не имеет временной отметки, связанной с его обновлениями. Если процесс ввода имеет значение временной отметки, то используйте режим выборки в реальном времени.
Для управления во время выполнения инструкции добавьте соответствующий алгоритм в вашу программу. Например, вы можете использовать таймер, настроенный на значение OversampleDeltaT, для управления выполнением посредством использования входа EnableIn этой инструкции.
Обработка входных данных должна быть быть в 5-10 раз быстрее, чем выполнение инструкции, чтобы минимизировать ошибку выборки данных между вводом и выполнением инструкции.</td></tr>
</table>

Режим синхронизации:	**Описание**
выборка в режиме реального времени (real time sampling)	В режиме выборки в реальнov времени приращение времени (DeltaT), используемое инструкцией, является разностью между двумя значениями отметок времени, соответствующих обновлениям параметров процесса. Используйте этот режим когда инструкция выполняется в событии или непрерывной задаче, и ввод характеристик процесса имеет отметку времени, связанную с его обновлением. Отметка времени считывается из тега, вводимого для параметра TSTimeStamp этой инструкции. Обычно именем этого тега является параметр модуля ввода, связанный с этой характеристикой процесса. Инструкция сравнивает сконфигурированное значение RTSTime (предполагаемая периодичность обновления) с рассчитанным значением DeltaT, для того чтобы определить, каждое ли обновление характеристики процесса считывается инструкцией. Если DeltaT не находится в пределах 1 миллисекунды от сконфигурированного времени, инструкция устанавливает бит состояния RTSMissed, чтобы показать, что есть проблема при считывании обновлений для ввода этого модуля.

Для инструкций, связанных с синхронизацией, требуется, чтобы значение DeltaT было постоянным для того, чтобы контролировать правильность расчета выходного значения. Если DeltaT изменяется, происходит разрывность выходного значения. Степень разрывности зависит от инструкции и диапазона изменения DeltaT. Разрывность имеет место если:

- инструкция не выполняется при каком-либо сканировании.
- инструкция выполняется несколько раз в задаче.
- задача выполняется и скорость сканирования задачи или время опроса характеристик процесса изменяется.
- пользователь меняет режим синхронизации при выполнении задачи.
- параметр Order меняется на фильтре блока при выполнении задачи. Изменение параметра Order приводит к выбору другого алгоритма управления в инструкции.

Общие параметры инструкций, связанные с режимами синхронизации

Инструкции, которые поддерживают режим синхронизации, имеют следующие входные и выходные параметры:

Входные параметры

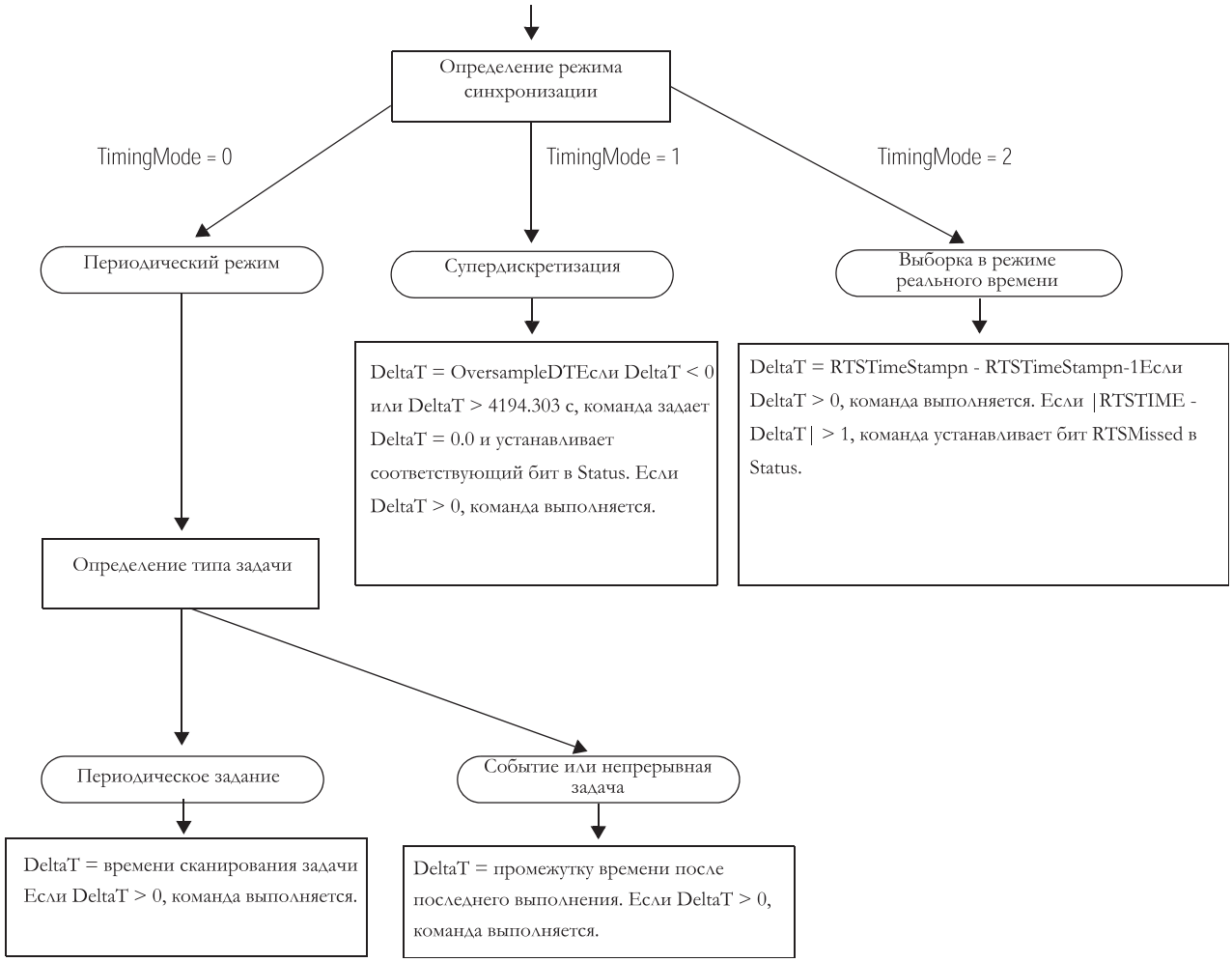
Входной параметр:	**Тип данных:**	**Описание:**
TimingMode	DINT	Выбирает режим синхронизации. **Значение:** — **Описание:** 0 — периодический режим 1 — режим супердискретизации 2 — выборка в режиме реального времени допустимые значения = от 0 до 2 значение по умолчанию = 0 Если TimingMode = 0 и задача является периодической, устанавливается синхронизация по периоду и DeltaT задается равным скорости сканирования задачи. Если TimingMode = 0, а задача является событием или непрерывной задачей, разрешается синхронизация по периоду и DeltaT задается равным полному интервалу времени после последнего выполнения этой инструкции. Если TimingMode = 1, устанавливается режим супердискретизации и DeltaT задается равным значению параметра OversampleDT. Если TimingMode = 2, устанавливается выборка в режиме реального времени и DeltaT задается равным разности между значениями текущей и предыдущей отметок времени, считываемой из модуля, связанного с этим вводом. Если TimingMode имеет недопустимое значение, инструкция устанавливает соответствующий бит в Status.
OversampleDT	REAL	Время выполнения для режима супердискретизации. Значение DeltaT задается в секундах. Если TimingMode = 1, то OversampleDT = 0.0, что запрещает выполнение управляющего алгоритма. Если значение недопустимо, инструкция задает DeltaT = 0.0 и устанавливает соответствующий бит в Status. допустимые значения = от 0 до 4194.303 секунд значение по умолчанию = 0.0
RTSTime	DINT	Период обновления модуля для выборки в режиме реального времени. Предполагаемый период обновления DeltaT задается в миллисекундах. Период обновления обычно является значением, которое было использовано при задании времени обновления для модуля. Если значение недопустимо, инструкция устанавливает соответствующий бит в Status и запрещает проверку RTSMissed. допустимые значения = от 1 до 32767 мс значение по умолчанию = 1 мс
RTSTimeStamp	DINT	Значение отметки времени модуля для выборки в режиме реального времени.. Значение отметки времени, которое соответствует последнему обновлению входного сигнала. Это значение используется для расчета DeltaT. Если значение недопустимо, инструкция устанавливает соответствующий бит в Status, запрещая выполнение управляющего алгоритма и проверку RTSMissed. допустимые значения = от 1 до 32767 мс (от 32767 до 0) 1 отсчет = 1 миллисекунда значение по умолчанию = 0

Выходные параметры

Выходной параметр:	Тип данных:	Описание:
DeltaT	REAL	Интервал времени между обновлениями. Это интервал времени в секундах, используемый управляющим алгоритмом для расчета выходного значения. Периодический режим: DeltaT = скорости сканирования задачи, если задача является периодической (Periodic), DeltaT = промежутку времени после предыдущего выполнения задачи, если задача является событием (Event) или непрерывной задачей (Continuous). Режим супердискретизации: DeltaT = OversampleDT Выборка в режиме реального времени: DeltaT = (RTSTimeStampn - RTSTimeStampn-1)
Status	DINT	Состояние функционального блока.
TimingModeInv (Status.27)	BOOL	Недопустимое значение TimingMode.
RTSMissed (Status.28)	BOOL	Используется только для выборки в режиме реального времени. Задается когда ABS | DeltaT - RTSTime | > 1 (.001 с).
RTSTimeInv (Status.29)	BOOL	Недопустимое значение RTSTime.
RTSTimeStampInv (Status.30)	BOOL	Недопустимое значение RTSTimeStamp.
DeltaTInv (Status.31)	BOOL	Недопустимое значение DeltaT.

Общее представление о режимах синхронизации

Представленная ниже диаграмма показывает, как инструкция определяет режим синхронизации.



Управление программа/оператор (Program/Operator)

Несколько инструкций поддерживают концепцию управления программа/оператор.

Эти инструкции включают в себя:

- Расширенный выбор (ESEL) Enhanced Select
- Сумматор (TOT) Totalizer
- Усовершенствованный регулятор PID (PIDE) Enhanced PID
- Нарастание/выдержка (RMPS) Ramp/Soak
- Дискретное 2-х режимное устройство (D2SD) Discrete 2-State Device
- Дискретное 3-х режимное устройство (D3SD) Discrete 3-State Device

Способ управления программа/оператор позволяет вам управлять этими инструкциями одновременно как из пользовательской программы, так и при помощи интерфейса оператора. При программном управлении (Program) инструкция управляется при помощи программных входов (Program inputs) для этой инструкции, а при управлении оператором (Operator) инструкция управляется входами оператора (Operator inputs).

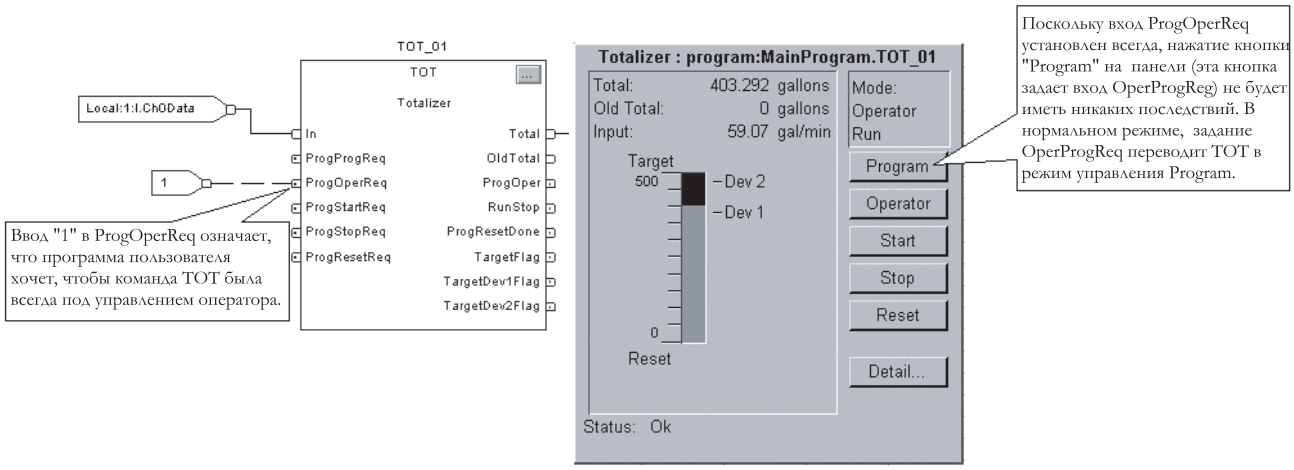
Режимы управления Program или Operator задаются при помощи следующих входов:

Вход:	Описание:
.ProgProgReq	Программный запрос на переход к программному управлению (Program).
.ProgOperReq	Программный запрос на переход к управлению оператором (Operator).
.OperProgReq	Запрос оператора на переход к программному управлению (Program).
.OperOperReq	Запрос оператора на переход к управлению оператором (Operator).

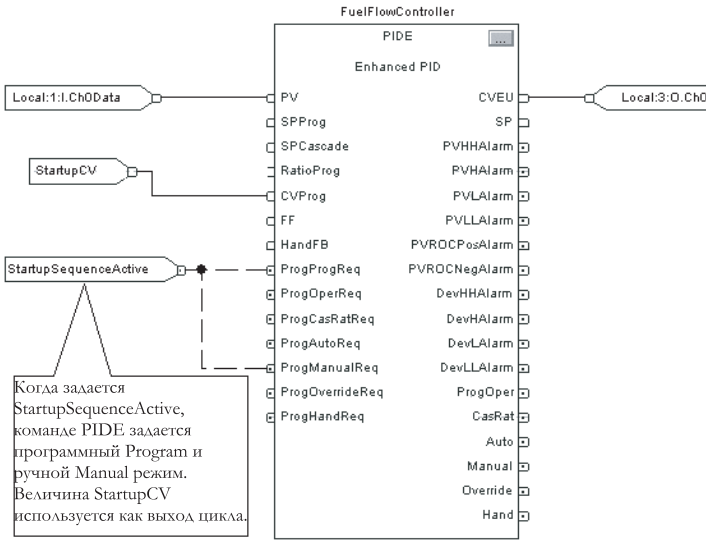
Для того чтобы определить, в каком режиме управления находится инструкция (Program или Control), проверьте выходное значение ProgOper. Если ProgOper установлен, инструкция находиться в режиме управления Program; если ProgOper сброшено, инструкция находится в режиме управления Operator.

Если установлены оба бита запроса режима управления, то режим Operator имеет приоритет над режимом Program. Например, если оба бита ProgProgReq и ProgOperReq установлены, инструкция переходит в режим управления Operator.

Входы программных запросов (Program request) имеют приоритет над входами запросов оператора (Operator request). Это дает возможность использовать входные значения ProgProgReq и ProgOperReq для «фиксации» инструкции в желаемом режиме управления. Например, допустим, что инструкция Totalizer будет всегда работать под управлением оператора, и ваша программа никогда не сможет управлять запуском или остановом инструкции Totalizer. В этом случае вы должны ввести (подсоединить) литерную величину 1 в ProgOperReq. Это не даст возможности оператору перевести инструкцию Totalizer в программный режим управления, задав OperProgReq с интерфейса оператора.



Аналогично, постоянная установка ProgProgReq может «зафиксировать» инструкцию в программном режиме управления. Это полезно при последовательном включении, когда вы хотите, чтобы программа управляла работой инструкции не опасаясь, что оператор, по невнимательности, возьмет управление над инструкцией. В этом примере программа задает вход ProgProgReq при запуске, а затем очищает ProgProgReq, когда запуск завершен. После того как ProgProgReq сброшен, инструкция останется в программном режиме управления, пока не получит запрос на его изменение. Например, оператор может задать OperOperReq с панели для того, чтобы взять управление инструкцией в свои руки. В следующем примере показано как «зафиксировать» инструкцию в программном режиме управления.



Входные значения запроса оператора в инструкцию всегда сбрасываются инструкцией при выполнении. Это позволяет интерфейсам оператора работать с этими инструкциями, просто задавая бит запроса желаемого режима. Вам не нужно программировать интерфейс оператора на сброс битов запроса. Например, если интерфейс оператора задает вход OperAutoReq для инструкции PIDE, то когда выполняется инструкция PIDE, она определяет, каким должен быть соответствующий отклик, и сбрасывает OperAutoReq.

Входные значения программных запросов обычно не сбрасываются инструкцией, поскольку они задаются как входные связи. Если инструкция сбросит эти входные значения, они будут восстановлены элементом входной связи. Может возникнуть ситуация, когда вы захотите задать программные запросы (Program requests) таким образом, чтобы они сбрасывались инструкцией. В этом случае вы можете задать вход ProgValueReset и программа будет всегда сбрасывать значения программных запросов при выполнении.

В этом примере цепочка релейной логики в другой процедуре используется для разового защелкивания ProgAutoReq для инструкции PIDE при нажатии на пусковую кнопку. Поскольку инструкция PIDE автоматически сбрасывает программные запросы, у вас нет необходимости записывать какой либо алгоритм релейных схем для того, чтобы сбрасывать ProgAutoReq после выполнения процедуры, и инструкция будет получать только один запрос на переход в режим Auto при каждом нажатии кнопки.

Когда TIC101AutoReq Pushbutton нажата, происходит разовое защелкивание ProgAutoReq для инструкции PIDE TIC101. TIC101 уже была сконфигурирована с заданным входным значением ProgValueReset так, что когда инструкция PIDE выполняется, она автоматически сбрасывает ProgAutoReq.

Примечания:

Программирование структурированного текста

Введение

В этом приложении описываются особенности программирования структурированного текста. Чтобы убедиться, что вы понимаете, как работают программы в структурированном тексте, прочитайте это приложение.

За информацией о:	Обращайтесь на стр.:
Синтаксисе структурированного текста	C-1
Присваивании	C-2
Выражениях	C-4
Инструкциях	C-11
Конструкциях	C-12
Комментариях	C-28

Синтаксис структурированного текста

Структурированный текст является текстовым языком программирования, использующим операторы для задания действий для выполнения. Операторы структурированного текста не чувствительны к регистру. Структурированный текст может содержать следующие элементы:

Термин:	Определение:		Примеры:
присваивание (см. стр. C-2)	Используйте оператор присваивания для присвоения значений тегам. Символом присваивания является «:=». Присваивание заканчивается точкой с запятой «;».		*tag := expression;*
выражение (см. стр. C-4)	Выражение является частью таких элементов, как присваивание и конструкция. Выражение работает с числами (числовое выражение) или логическими величинами (логическое выражение (BOOL)). Выражение содержит:		
	теги	Поименованную область памяти для хранения данных (типа BOOL, SINT,INT,DINT, REAL, строка).	*value1*
	непосредственные значения	Постоянные значения.	4
	операторы	Символ или мнемоника, задающие операцию в пределах выражения.	*tag1 + tag2* *tag1 >= value1*
	функции	Прп выполнении функция выдает одно значение. Операнд функции помещается в круглые скобки. Даже при одинаковом синтаксисе функции отличаются от инструкций, в выражениях используются только функции. Инструкции не могут использоваться в выражениях.	*function(tag1)*

Термин:	Определение:	Примеры:
инструкция (см. стр. C-11)	Инструкция является отдельно расположенным оператором. Операнды инструкции помещаются в круглые скобки. В зависимости от типа инструкции, она может не содержать операндов, содержать один, два или несколько операндов. При выполнении инструкция имеет на выходе одно или более значений, которые являются частью структуры данных. Инструкция завершается точкой с запятой «;». Даже при одинаковом синтаксисе инструкции отличаются от функций, инструкции не могут использоваться в выражениях. В выражениях могут использоваться только функции.	*instruction ()* *instruction (operand);* *instruction (operand1, operand2,operand3);*

конструкция
(см. стр. C-12)
	Условный оператор, используемый для включения программы структурированного текста (т.е. других операторов).
	Конструкция заканчивается точкой с запятой «;».
		`IF...THEN`
		`CASE`
		`FOR...DO`
		`WHILE...DO`
		`REPEAT...UNTIL`
		`EXIT`

комментарий (см. стр. C-28)	Текст пояснения к части программы на языке структурированного текста. • используйте комментарий для того, чтобы облегчить понимание программы. • Комментарии не влияют на выполнение программы. • Комментарии могут быть в любом месте структурированного текста.	*//комментарий* *(*начало комментария . . . конец комментария*)* */*начало комментария . . . конец комментария*/*

Присваивание

Используйте присваивание для изменения значения, хранящегося в теге. Оператор присваивания имеет следующий синтаксис:

tag := expression ;

где:

Элемент:	Описание:	
tag (тег)	представляет собой тег, который должен получить новое значение. Тег должен иметь тип BOOL, SINT, INT, DINT или REAL	
:=	символ присваивания	
expression (выражение)	представляет собой новое значение, присваиваемое тегу	
	Если *tag (тег)* имеет тип:	**Используйте выражение типа:**
	BOOL	BOOL
	SINT INT DINT REAL	числовое выражение
;	конец оператора присваивания	

Тег сохраняет присвоенное значение до тех пор, пока новое присваивание его не изменит.

Выражение может быть простым, например, непосредственным значением, или именем тега, а может быть сложным и включать несколько операторов и/или функций. Более подробно этот случай изложен в разделе «Выражения» на стр. C-4.

Задание присваивания без сохранения

Присваивание без сохранения отличается от обычного присваивания, описанного выше, тем, что тег при присваивании без сохранения сбрасывается на ноль всякий раз, когда контроллер:

- входит в режим выполнения RUN
- выходит из шага SFC, если вы сконфигурировали SFC на автоматический сброс (Automatic reset) (Это применимо только если вы вставили оператор присваивания в action (действие) для данного шага или используете action для вызова процедуры структурированного текста через инструкцию JSR).

Оператор присваивания без сохранения имеет следующий синтаксис:

tag [:=] expression ;

где:

Элемент:	**Описание:**	
tag (тег)	представляет собой тег, который должен получить новое значение. Тег должен иметь тип BOOL, SINT, INT, DINT или REAL	
[:=]	символ присваивания без сохранения	
expression (выражение)	представляет собой новое значение, присваиваемое тегу	
	Если *tag (тег)* имеет тип:	**Используйте выражение типа:**
	BOOL	BOOL
	SINT INT DINT REAL	числовое выражение
;	конец оператора присваивания	

Присваивание символов ASCII строке

Используйте оператор присваивания для того, чтобы присвоить символ ASCII элементу члена DATA строкового тега. Чтобы присвоить символ, задайте значение символа или задайте имя тега, член DATA или элемент символа. Например:

Это правильно:	Это неправильно:
`string1.DATA[0]:= 65;`	`string1.DATA[0] := A;`
`string1.DATA[0]:= string2.DATA[0];`	`string1 := string2;`

Для того чтобы добавить или вставить строку символом в строковый тег, используйте следующие строковые инструкции ASCII:

Для того чтобы:	Используйте эту инструкцию:
добавить символы в конец строки	CONCAT
вставить символы в строку	INSERT

Выражения

Выражение является именем тега, уравнением или сравнением. Для того, чтобы записать выражение, используйте:

- имя тега, который хранит значение (переменную)
- число, которое вы хотите ввести (непосредственное значение)
- функции, такие как: ABS, TRUNC
- операторы, такие как: +, -, <, >, And, Or

Когда вы записываете выражение, следуйте этим общим правилам:

- Используйте любую комбинацию букв верхнего и нижнего регистров. Например, допустимы три варианта оператора "AND": AND, And и and.
- Для более сложных выражений используйте круглые скобки для создания групп внутри выражения. Это облегчит чтение и гарантирует, что выражение будет выполняться в нужной последовательности. См. раздел «Определение порядка выполнения» на стр. C-10.

В структурированном тексте вы можете использовать выражения следующих типов:

Логическое выражение (BOOL): это выражение, имеющее на выходе булево значение 1 (истина) или 0 (ложь).

- В логическом выражении используются теги типа bool, операторы отношения и логические операторы для сравнения значений и проверки условий «истина» или «ложь». Например, tag1>65.
- Простые логические выражения могут содержать один тег типа BOOL.
- Обычно вы используете логические выражения для проверки условия выполнения другого алгоритма.

Числовое выражение: это выражение, использующее в расчете целые числа или числа с плавающей точкой.

- Числовое выражение использует арифметические операторы, арифметические функции и побитовые операторы. Например, tag1+5.
- Часто вы вкладываете арифметическое выражение в логическое. Например, (tag1+5)>65.

Используйте следующую таблицу для выбора операторов выражения:

Если вы хотите:	**То:**
Вычислить арифметическое значение	Обратитесь к разделу «Использование арифметических операторов и функций» на стр. C-6.
Сравнить два значения или строки	Обратитесь к разделу «Использование операторов отношения» на стр. C-7.
Проверить, какое условие - «истина» или «ложь» - имеет значение	Обратитесь к разделу «Использование логических операторов» на стр. C-9.
Сравнить биты внутри значения	Обратитесь к разделу «Использование поразрядных операторов» на стр. C-10.

Использование арифметических операторов и функций

В арифметических выражениям вы можете комбинировать несколько операторов и функций.

Арифметические операторы рассчитывают новые значения.

Чтобы:	Используйте этот оператор:	Оптимальный тип данных:
сложить	+	DINT, REAL
вычесть/отрицать	-	DINT, REAL
умножить	*	DINT, REAL
возвести в степень (x в степень y)	**	DINT, REAL
разделить	/	DINT, REAL
разделить по модулю	MOD	DINT, REAL

Арифметические функции выполняют арифметические операции. Для этих функций необходимо задавать константу, тег не логического типа или выражение.

Для вычисления:	Используйте эту функцию:	Оптимальный тип данных:
абсолютной величины	ABS *(numeric_expression)*	DINT, REAL
арккосинуса	ACOS *(numeric_expression)*	REAL
арксинуса	ASIN *(numeric_expression)*	REAL
арктангенса	ATAN *(numeric_expression)*	REAL
косинуса	COS *(numeric_expression)*	REAL
перевода радиан в градусы	DEG *(numeric_expression)*	DINT, REAL
натурального логарифма	LN *(numeric_expression)*	REAL
десятичного логарифма	LOG *(numeric_expression)*	REAL
перевода градусов в радианы	RAD *(numeric_expression)*	DINT, REAL
синуса	SIN *(numeric_expression)*	REAL
квадратного корня	SQRT *(numeric_expression)*	DINT, REAL
тангенса	TAN *(numeric_expression)*	REAL
округления	TRUNC *(numeric_expression)*	DINT, REAL

Например:

Используйте этот формат:	**Пример:** **Для этой ситуации:**	**Вам следует использовать запись**
value1 operator value2	Если *gain_4* и *gain_4_adj* являются тегами типа DINT и ваше задание гласит: "Прибавить 15 к *gain_4* и сохранить результат в *gain_4_adj*».	gain_4_adj := gain_4+15;
operator value1	Если *alarm* и *high_alarm* (сигналы тревоги) являются тегами типа DINT и ваше задание гласит: «Поменять знак у *high_alarm* и сохранить результат в *alarm*».	alarm:= -high_alarm;
function(numeric_expression)	Если *overtravel* и *overtravel_POS* (перебег) являются тегами типа DINT и ваше задание гласит: «Рассчитать абсолютную величину перебега и сохранить результат в *overtravel_POS*».	overtravel_POS := ABS(overtravel);
value1 operator (function((value2+value3)/2)	Если *adjustment* и *position* являются тегами типа DINT, а *sensor1* и *sensor2* теги типа REAL и ваше задание гласит: "Найти абсолютную величину среднего значения *sensor1* и *sensor2*, прибавить *adjustment* и сохранить результат в *position*».	position := adjustment + ABS((sensor1 + sensor2)/2);

Использование операторов отношения

Операторы отношения сравнивают два значения или две строки и выдают логический результат «истина» или «ложь». Результатом операции сравнения является булево значение:

Если сравнение:	**Результат:**
истина	1
ложь	0

Используйте следующие операторы отношения:

Для следующего сопоставления:	**Используйте этот оператор:**	**Оптимальный тип данных:**
равно	=	DINT, REAL, строка
меньше, чем	<	DINT, REAL, строка
меньше или равно	<=	DINT, REAL, строка
боьше, чем	>	DINT, REAL, строка
больше или равно	>=	DINT, REAL, строка
не равно	<>	DINT, REAL, строка

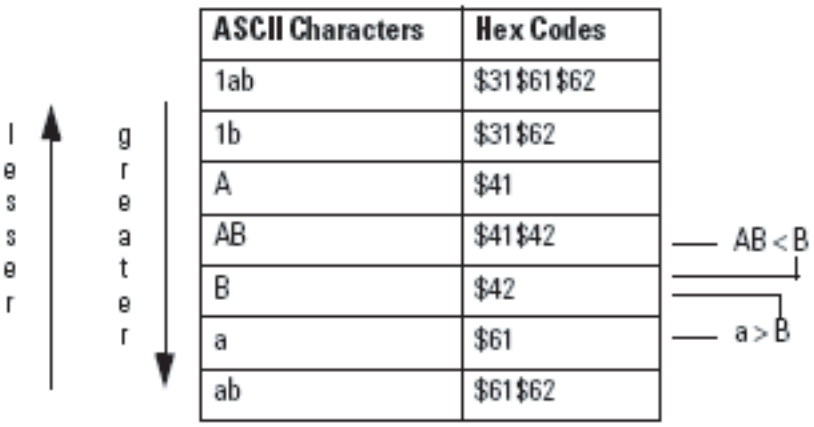
Например:

Используйте этот формат:	Пример:	
	Для этой ситуации:	**Вам следует использовать запись**
value1 operator value2	Если *temp* является тегом типа DINT и ваше задание гласит: "Если *temp* меньше 100°, то…».	`IF temp<100 THEN...`
cstringtag1 operator stringtag2	Если *bar_code* и *dest* являются строковыми тегами и ваше задание гласит: «Если *bar_code* равен *dest*, то…».	`IF bar_code=dest THEN..`
char1 operator char2 Чтобы ввести символ ASCII прямо в выражение, введите десятичное значение этого символа.	Если *bar_code* строковый тег и ваше задание гласит: «Если *bar_code.DATA[0]* равен 'A', то …».	`IFbar_code.DATA[0]=65 THEN...`
bool_tag := bool_expressions	Если *count* и *length* теги типа DINT , а *done* – тег типа BOOL и ваше задание гласит: «Если *count* больше или равен *length*, то необходимо выполнить подсчет».	`done := (count >= length);`

Как производятся операции со строками

Шестнадцатиричные значения символов ASCIE определяют, больше одна строка, чем другая, или меньше.

- когда две строки сортируются в телефонном справочнике, порядок следования строк определяется тем, какая строка больше.



- Строки равны, если их символы совпадают.
- Символы чувствительны к регистру. Прописная "A" ($41) *не* равна строчной "a" ($61).

Десятичные и шестнадцатеричные коды символов приведены в конце данного руководства.

Использование логических операторов

Логические операторы позволяют вам проверять, являются ли многочисленные условия истиной или ложью. Результатом логической операции является булево значение:

Если сравнение:	**Результат:**
истина	1
ложь	0

Используйте следующие логические операторы:

Для:	**Используйте этот оператор:**	**Тип данных:**
логического “И”	&, AND	BOOL
логического “ИЛИ”	OR	BOOL
логического исключающего “ИЛИ”	XOR	BOOL
логического “НЕ”	NOT	BOOL

Например:

Используйте этот формат:	**Пример:**	
	Для этой ситуации:	**Вам следует использовать запись**
BOOLtag	Если *photoeye* является тегом BOOL и ваше задание гласит: «Если *photoeye_1* установлен, то …».	`IF photoeye THEN...`
NOT BOOLtag	Если *photoeye* является тегом тBOOL и ваше задание гласит: «Если *photoeye_1* сброшен, то …».	`IF NOT photoeye THEN...`
expression1 & expression2	Если *photoeye* является тегом BOOL , *temp* является тегом DINT и ваше задание гласит: «Если *photoeye* установлен и *temp* меньше 100° то…».	`IF photoeye & (temp<100) THEN...`
expression1 OR expression2	Если *photoeye* является тегом BOOL, *temp* является тегом DINT и ваше задание гласит: «Если *photoeye* установлен или *temp* меньше 100° то…».	`IF photoeye OR (temp<100) THEN...`
expression1 XOR expression2	Если *photoeye1* и *photoeye2* являются тегами BOOL и ваше задание гласит: «Если: • *photoeye1* установлен, в то время как *photoeye2* сброшен или • *photoeye1* сброшен, в то время как *photoeye2* установлен то…».	`IF photoeye1 XOR photoeye2 THEN...`
BOOLtag := expression1 & expression2	Если *photoeye1, photoeye2* и *open* являются тегами BOOL и ваше задание гласит: «Если *photoeye1* и *photoeye2* оба установлены, присвоить *open* значение «истина».	`open := photoeye1 & photoeye2;`

Использование поразрядных операторов

Поразрядные операторы обрабатывают биты для двух значений.

Для:	Используйте этот оператор:	Оптимальный тип данных:
поразрядного “И”	&, AND	DINT
поразрядного “ИЛИ”	OR	DINT
поразрядного исключающего “ИЛИ2	XOR	DINT
поразрядного “НЕ”	NOT	DINT

Например:

Используйте этот формат:	Пример:	
	Для этой ситуации:	Вам следует использовать запись
value1 operator value2	Если *input1*, *input2* и *result1* являются тегами DINT, и ваше задание гласит: «Рассчитать побитовый результат от *input1* и *input2*. Сохранить результат в *result1*».	result1 := input1 AND input2;

Определение порядка выполнения

Операции, которые вы записываете в выражение, выполняются в установленном порядке, и этот порядок не обязательно слева направо.

- Операции одного порядка выполняются слева направо.
- Если выражение содержит несколько операторов или функций, группируйте их в круглых скобках "( )". Это гарантирует правильный порядок выполнения и облегчит понимание выражения.

Порядок:	Операция:
1.	()
2.	функция(...)
3.	**
4.	-(смена знака)
5.	NOT
6.	*,/, MOD
7.	+,-(вычитание)
8.	<,<=,>,>=
9.	=,<>
10.	&, AND
11.	XOR
12.	OR

Инструкции

Операторами структурированного текста могут быть и инструкции. Список инструкций, имеющихся в структурированном тексте, приведен в начале данного руководства. Инструкции структурированного текста выполняются всякий раз, когда они сканируются. Инструкции структурированного текста внутри конструкции выполняются всякий раз, когда условия конструкции принимают значения «истина». Если условия конструкции имеют значение «ложь», операторы внутри конструкции не сканируются. Не существуе условия цепочки или перехода, которое запускало бы выполнение.

Это отличает инструкции структурированного текста от инструкций функционального блока, в котором для включения выполнения инструкции используется EnableIn. В структурированном тексте инструкции выполняются так, как будто бы EnableIn всегда установлен.

Это также отличает инструкции структурированного текста от инструкций релейной логики, в которой входное условие цепочки запускает выполнение. Некоторые инструкции релейной логики выполняются только в том случае, когда входное условие цепочки переключается со значения «ложь» на значение «истина». В релейной логике это так называемые переходные инструкции. В структурированном тексте инструкции будут выполняться при каждом своем сканировании, если вы не введете какое-либо предварительное условие на выполнение инструкции.

Например, инструкция ABL является переходной инструкцией в релейной логике. В этом примере инструкция ABL выполняется только при сканировании, когда *tag_xic* переходит из положения сброшен в положение установлен. Инструкция ABL не выполняется, если *tag_xic* находится в положении установлен или сброшен.

Для структурированного текста, если вы запишите этот пример следующим образом:

```
IF tag_xic THEN ABL(0,serial_control);
END_IF;
```

то инструкция ABL будет выполняться при каждом сканировании, когда *tag_xic* установлен, а не только тогда, когда *tag_xic* переходит из положения сброшен в положение установлен.

Если вы хотите, чтобы инструкция ABL выполнялась только тогда, когда *tag_xic* переходит из положения сброшен в положение установлен, вы должны ввести специальное условие. Используйте единичное включение инструкции.

```
osri_1.InputBit := tag_xic;
OSRI(osri_1);


IF (osri_1.OutputBit) THEN
     ABL(0,serial_control);
END_IF;
```

Конструкции

Конструкции могут программироваться как одиночные или как вложенные в другие конструкции.

Если вы хотите:	Используйте эту конструкцию:	Имеющуюся в этих языках:	См. стр.
что-то сделать, если или когда имеет место заданное условие	IF...THEN	структурированный текст	C-13
выбрать, что делать на основе числового значения	CASE...OF	структурированный текст	C-16
сделать что-либо заданное число раз, до того, как сделать что-либо еще	FOR...DO	структурированный текст	C-19
продолжать делать что-либо, пока определенное условие имеет значение «истина»	WHILE...DO	структурированный текст	C-22
продолжать делать что-либо, пока определенное условие не примет значение «истина»	REPEAT...UNTIL	структурированный текст	C-25

IF...THEN

Используйте IF...THEN (если ... , то ....) если или когда имеет место определенное условие.

Операнды:

```
IF bool_expression THEN
    <statement>;
END_IF;
```

Структурированный текст

Операнд:	Тип:	Формат:	Ввод:
bool_ expression	BOOL	тег выражение	тег BOOL или выражение, которое рассчитывает значение BOOL (логическое выражение)

Описание:

Синтаксис:

```
IF bool_expression1 THEN
      <statement >;        <-- statements to execute when bool_expression1 is true
                .
                .
                .
optional { ELSIF bool_expression2 THEN
      <statement>;         <-- statements to execute when bool_expression2 is true
                .
                .
                .
optional { ELSE
      <statement>;         <-- statements to execute when both expressions are false
                .
                .
                .
END_IF;
```

При использовании ELSIF или ELSE следуйте этим указаниям:

1 Чтобы произвести выбор из нескольких групп операторов, добавьте один или более операторов ELSIF.

- Каждый оператор ELSIF представляет альтернативный путь.
- Задавайте столько путей ELSIF, сколько вам нужно.
- Контроллер выполняет первое значение «истина» для IF или ELSIF и игнорирует оставшиеся ELSIF и ELSE.

2 Для того чтобы что-либо сделать, когда все условия для IF или ELSIF имеют значения «ложь», добавьте оператор ELSE.

В следующей ниже таблице представлены различные комбинации IF, THEN, ELSIF и ELSE.

Если вы хотите:	И:	Используйте эту конструкцию:
что-либо сделать, если или когда заданные условия имеют значение «истина»	ничего не делать, если эти условия имеют значение «ложь»	IF…THEN
	сделать что-либо другое, если эти условия имеют значение «ложь»	IF…THEN…ESLE
сделать выбор из альтернативных операторов (или групп операторов) на основе входных условий	ничего не делать, если эти условия имеют значение «ложь»	IF…THEN…ELSIF
	присвоить операторы по умолчанию, если все эти условия имеют значение «ложь»	IF…THEN…ELSIF…ELSE

Пример 1: IF…THEN

Если вы хотите, чтобы:	Введите структурированный текст:

IF rejects (если бракованные детали) > 3 then (тогда)
conveyor = off (0) (конвейер остановить)
alarm = on (1) (включить сигнал тревоги)
	`IF rejects > 3 THEN`
	`  conveyor := 0;`
	`  alarm := 1;`
	`END_IF;`

Пример 2: IF…THEN…ELSE

Если вы хотите, чтобы:	Введите структурированный текст:

If conveyor direction contact (если контакт направления движения конвейера) = forward (направлен вперед) (1) then (тогда)
light = off (лампочку выключить)
Otherwise (в противном случае) light = on (лампочку включить)
	`IF conveyor_direction THEN`
	`  light := 0;`
	`ELSE`
	`  light [:=] 1;`
	`END_IF;`

Символ [:=] приказывает контроллеру сбрасывать *light* всякий раз, когда контроллер

- входит в режим выполнения RUN,
- выходит из шага SFC, если вы сконфигурировали SFC на Automatic reset (автоматический сброс). (Это применимо, только если вы вставляете оператор в action (действие) или используете action для вызова процедуры структурированного текста посредством инструкции JSR).

Пример 3: IF…THEN…ELSIF

Если вы хотите, чтобы:	Введите структурированный текст:

If sugar low limit switch (если нижний концевой выключатель уровня сахара) = low (on) (низкий) and sugar high limit switch (и верхний концевой выключатель уровня сахара) = not high (on) (невысокий) then (то)
inlet valve (впускной клапан) = open (on) (открыт)
Until sugar high limit switch (пока верхний концевой выключатель уровня сахара) = high (off) (высокий)
	`IF Sugar.Low & Sugar.High THEN`
	`Sugar.Inlet [:=] 1;`
	`ELSIF NOT(Sugar.High) THEN`
	`Sugar.Inlet := 0;`
	`END_IF;`

Символ [:=] приказывает контроллеру сбрасывать *SugarInlet* всякий раз, когда контроллер

- входит в режим выполнения RUN,
- выходит из шага SFC, если вы сконфигурировали SFC на Automatic reset (автоматический сброс). (Это применимо, только если вы вставляете оператор в action (действие) или используете action для вызова процедуры структурированного текста посредством инструкции JSR).

Пример 4: IF…THEN…ELSIF…ELSE

Если вы хотите чтобы:	Введите структурированный текст:

If tank temperature (если температура бака) > 100
then pump (то насос работает) = slow (медленно)
If tank temperature (если температура бака) > 200
then pump (то насос работает) = fast (быстро)
otherwise pump (в противном случае насос) = off (отключить)
	`IF tank.temp > 200 THEN`
	`pump.fast :=1; pump.slow :=0;`
	`pump.off :=0;`
	`ELSIF tank.temp > 100 THEN`
	`pump.fast :=0; pump.slow :=1;`
	`pump.off :=0;`
	`ELSE`
	`pump.fast :=0; pump.slow :=0;`
	`pump.off :=1;`
	`END_IF;`

CASE...OF

Используйте конструкцию CASE для выбора последующих действий на основе числового значения.

Операнды:

```
CASE numeric_expression OF
    selector1: statement;
    selectorN: statement;
ELSE
    statement;
END_CASE;
```

Структурированный текст

Операнд:	Тип:	Формат:	Ввод:
numeric_expression	SINT INT DINT REAL	тег выражение	тег или выражение, рассчитывающее числовое значение (числовое выражение)
selector	SINT INT DINT REAL	непосредственный	тот же самый тип, что и *numeric_expression*

ВАЖНО! Если вы используете значение REAL, используйте для selector некий диапазон значений, поскольку для значений типа REAL более верно говорить о попадании в диапазон, а не о строгом совпадении одного значения с другим.

Описание:

Синтаксис:

```
CASE numeric_expression OF
    selector1 : <statement>;      ← операторы для выполнения, если numeric_expression = selector1
                .
                .
                .
    selector2 : <statement>;      операторы для выполнения, если numeric_expression = selector2
                .
                .
                .
    selector3 : <statement>;      операторы для выполнения, если numeric_expression = selector3.
                .
                .
                .
ELSE                              необязательно
                <statement>;      ← операторы для выполнения, если numeric_expression неравен любому из selector
                .
                .
                .
END_CASE;
```

задавайте столько альтернативных значений selector (путей), сколько вам нужно

Допустимые значения selector представлены в таблице на следующей странице.

Синтаксис для ввода значений selector:

Когда selector:	Вводите:
одно значение	*value: statement* (значение: оператор)
несколько различных значений	*value1, value2, valueN:<statement>* Используйте запятую (,) для разделения значений.
диапазон значений	*value1..valueN:<statement>* Используйте две точки (..) для определения диапазона.
различные значения + диапазон значений	*valuea, valueb, value1..valueN:<statement>*

Конструкция CASE подобна оператору switch в языках программирования C или C++. Однако при использовании CASE контроллер выполняет только те операторы, которые соответствуют *первому совпадению* со значением selector. После выполнения этих операторов управление передается на оператор END_CASE.

Пример:

Если вы хотите, чтобы:	Введите структурированный текст:

If recipe number (Если номер рецепта) = 1 then (то)
Ingredient A outlet 1 (выход 1 ингредиента A) = open (открыт) (1)
Ingredient B outlet 4 = open (1)
If recipe number = 2 or 3 then
Ingredient A outlet 4 = open (1)
Ingredient B outlet 2 = open (1)
If recipe number = 4, 5, 6, or 7 then
Ingredient A outlet 4 = open (1)
Ingredient B outlet 2 = open (1)
If recipe number = 8, 11, 12, or 13 then
Ingredient A outlet 1 = open (1)
Ingredient B outlet 4 = open (1)
Otherwise all outlets (в противном случае все выходы) = closed (закрыты) (0)
	CASE recipe_number OF
	1: Ingredient_A.Outlet_1 :=1;
	Ingredient_B.Outlet_4 :=1;
	2,3: Ingredient_A.Outlet_4 :=1;
	I ngredient_B.Outlet_2 :=1;
	4..7: Ingredient_A.Outlet_4 :=1;
	Ingredient_B.Outlet_2 :=1;
	8,11..13 Ingredient_A.Outlet_1 :=1;
	I ngredient_B.Outlet_4 :=1;
	ELSE
	Ingredient_A.Outlet_1 [:=]0;
	Ingredient_A.Outlet_4 [:=]0;
	Ingredient_B.Outlet_2 [:=]0;
	Ingredient_B.Outlet_4 [:=]0;
	END_CASE;

Символ [:=] приказывает контроллеру сбрасывать *outlet* всякий раз, когда контроллер

- входит в режим выполнения RUN,
- выходит из шага SFC, если вы сконфигурировали SFC на Automatic reset (автоматический сброс). (Это применимо, только если вы вставляете оператор в action (действие) или используете action для вызова процедуры структурированного текста посредством инструкции JSR).

FOR...DO

Используйте цикл FOR...DO для выполнения каких-либо операций заданное число раз перед тем, как перейти к другим операциям.

Операнды:

```
FOR count:= initial_value TO
final_value BY increment DO
    <statement>;
END_FOR;
```

Структурированный текст

Операнд:	Тип:	Формат:	Ввод:
count	SINT INT DINT	тег	тег для хранения позиции count (счетчика) при выполнении FOR...DO
initial_ value	SINT INT DINT	тег выражение непосредственный	рассчитывает первое значение для count
final_ value	SINT INT DINT	тег выражение непосредственный	рассчитывает последнее значение для count, определяющее, когда выходить из цикла
increment	SINT INT DINT	тег выражение непосредственный	(необязательно) приращение count Если вы не задаете increment, то приращение равно 1.

ВАЖНО!

Убедитесь, что вы не делаете слишком много итераций в пределах одного сканирования.

- Контроллер *не* будет выполнять никаких других операторов в процедуре, пока не завершит цикл.
- Если время, требующееся для завершения цикла, больше, чем время контролирующего таймера для этой задачи, произойдет основная ошибка.
- Тщательно обдумывайте использование таких конструкций, как IF...THEN.

Описание:

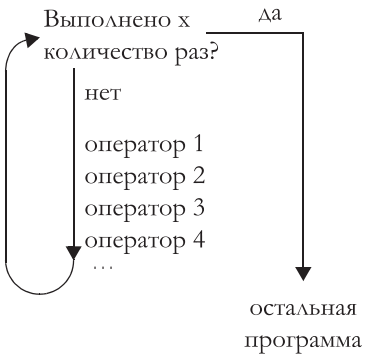
Синтаксис:

```
FOR count := initial_value
   TO final_value
   BY increment
   DO
   <statement>;
   IF bool_expression THEN
       EXIT;
   END_IF;
END_FOR;
```

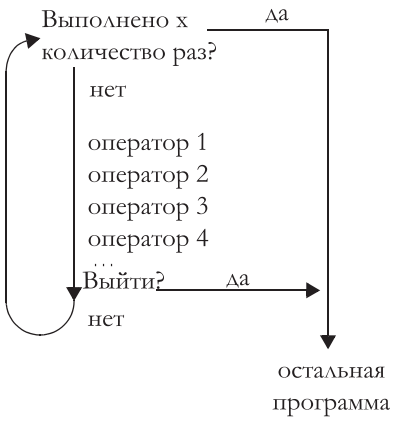
необязательно { BY increment — Если вы не задаете приращение, значение увеличивается на 1.

необязательно { IF bool_expression THEN … END_IF; — Если существуют условия, при который вы хотите выйти из цикла раньше, используйте такую

Следующая схема демонстрирует выполнение цикла FOR...DO и то, как правильно использовать оператор EXIT для более раннего выхода из цикла.



Цикл FOR...DO выполняется заданное число раз.



Для остановки цикла до того, как счетчик count достигнет последнего значения, используйте оператор EXIT.

Пример 1:

Если вы хотите:	**Введите:**
Очистить разряды 0 - 31 в массиве BOOL:	For subscript:=0 to 31 by 1 do
1. Присвоить 0 тегу *subscript* (индекса).	array[subscipt] := 0;
2 Очистить *array[subscript]* (элемент массива [индекс]) . Например, когда *subscript* = 5, очистить элемент *array[5]*.	End_for;
3.Прибавить 1 к *subscript.*	
4. Если *subscript* <= 31, повторить пункты 2 и 3. В противном случае, остановиться.	

Пример 2:

Если вы хотите, чтобы:	Введите структурированный текст:

Структура хранения данных пользователя содержала следующую информацию:
•Идентификационный штриховой код продукта (строковый тип)
•Количество данного продукта на складе (тип данных DINT)
Массив (Inventory), описанной выше структуры содержал один элемент для каждого продукта. Вы хотите найти заданный продукт (используя штриховой код) и определить его количество на складе.
1. Определите размер массива Inventory (ассортимент продуктов) и сохраните результат в *Inventory_Items* (тег типа DINT).
2. Присвойте 0 тегу *position*.
3. Если штриховой код *Barcode* совпадает с идентификатором продукта ID в массиве, то:
a. Присвоqnt тегу *Quantity* (количество) значение *Inventory[position].Qty*. Это количество продукта на складе.
b. Остановитесь.
Barcode является строковым тегом, в котором хранится штриховой код продукта, который вы ищете. Например, если *position* = 5, сравнивается *Barcode* с *Inventory[5].ID*.
4. Прибавьте 1 к *position.*
5. Если *position* <= (*Inventory_Items -1*), повторите пункты 3 и 4.
Поскольку нумерация начинается с 0, последний элемент на 1 меньше, чем номер в массиве.
В противном случае, остановитесь.
	`SIZE(Inventory,0,Inventory_Items);`
	`For position:=0 to Inventory_Items - 1 do`
	`  If Barcode=Inventory[position].ID then`
	`  Quantity:=Inventory[position].Qty;`
	`  Exit;`
	`  End_if;`
	`End_for;`

WHILE...DO

Используйте цикл WHILE...DO для того, чтобы продолжать выполнять какие-либо операции, пока заданные условия сохраняют значение «истина».

Операнды:

```
WHILE bool_expression DO
    <statement>;
END_WHILE;
```

Структурированный текст

Операнд:	Тип:	Формат:	Ввод:
bool_expression	BOOL	тег выражение	тег BOOL или выражение, которое рассчитывает значение BOOL

ВАЖНО!

Убедитесь, что вы не делаете слишком много итераций в пределах одного сканирования.

- Контроллер *не* будет выполнять никаких других операторов в процедуре, пока не завершит цикл.
- Если время, требующееся для завершения цикла, больше, чем время контролирующего таймера для этой задачи, произойдет основная ошибка.
- Тщательно обдумывайте использование таких конструкций, как IF...THEN.

Описание:

Синтаксис:

```
WHILE bool_expression1 DO
   <statement>;
   IF bool_expression2 THEN
     EXIT;
   END_IF;
END_WHILE;
```

Операторы для выполнения, пока bool_expression1 имеет значение "истина".

Необязательно

Если существуют условия, при который вы хотите выйти из цикла раньше, используйте такую структуру как IF...THEN для задания условий перехода к оператору EXIT.

Следующая схема демонстрирует выполнение цикла WHILE...DO и то, как использовать оператор EXIT для более раннего выхода из цикла.

Пока логическое выражение bool_expression сохраняет значение «истина», контроллер выполняет только операторы внутри цикла WHILE...DO.

Для остановки цикла, когда сохраняется значение «истина», используйте оператор EXIT.

Пример 1:

Если вы хотите, чтобы:	Введите структурированный текст:

Цикл WHILE...DO, в первую очередь, рассчитал условия выполнения. Если эти условия имеют значение «истина», контроллер выполнит операторы внутри цикла.
Это отличается от цикла REPEAT...UNTIL, поскольку цикл REPEAT...UNTIL выполняет операторы в конструкции, а потом определяет, имеют ли условия значение «истина» перед следующим выполнением операторов. Операторы в цикле REPEAT...UNTIL всегда выполняются хотя бы один раз. Операторы в цикле WHILE...DO могут не выполняться ни разу.
	`pos := 0;`
	`While ((pos <= 100)structarray[pos].value <> targetvalue) do`
	`pos := pos + 2;`
	`String_tag.DATA[pos] := SINT_array[pos];`
	`end_while;`

Пример 2:

Если вы хотите:	Введите структурированный текст:

Переместить символы ASCII из массива SINT в строковый тег. (В массиве SINT каждый элемент содержит один символ). Остановиться при достижении символа возврата каретки.
1. Присвойте 0 значению *Element_number* (номер элемента).
2. Подсчитайте количество элементов в массиве *SINT_array* (массив, содержащий символы ASCII) и *сохраните* результат в *SINT_array_size* (тег DINT).
3. Если символ в *SINT_array[element_number]* = 13 (десятичный код возврата каретки), то остановитесь.
4. Присвойте *String_tag[element_number]* значение, равное символу в *SINT_array[element_number]*.
5. Прибавьте 1 к *element_number*. Это позволит контроллеру проверить следующий символ в *SINT_array*.
6. Присвойте члену Length тега *String_tag* значение *element_number*. (Это запишет количество символов в *String_tag*).
7. Если *element_number* = *SINT_array_size*, остановитесь. (Вы в конце массива и он не содержит символа возврата каретки.)
8. Перейдите к пункту 3.
	`element_number := 0;`
	`SIZE(SINT_array, 0, SINT_array_size);`
	`While SINT_array[element_number] <> 13 do`
	`  String_tag.DATA[element_number] := SINT_array[element_number];`
	`  element_number := element_number+1;`
	`  String_tag.LEN := element_number;`
	`  If element_number = SINT_array_size then exit;`
	`  end_if;`
	`end_while;`

REPEAT...UNTIL

Используйте цикл REPEAT...UNTIL для того, чтобы продолжать выполнять какие-либо операции, пока заданные условия сохраняют значение «истина».

Операнды:

```
REPEAT
    <statement>;
UNTIL bool_expression
END_REPEAT;
```

Структурированный текст

Операнд:	Тип:	Формат:	Ввод:
bool_expression	BOOL	тег выражение	тег BOOL или выражение, которое рассчитывает значение BOOL (логическое выражение)

ВАЖНО!

Убедитесь, что вы *не* делаете слишком много итераций в пределах одного сканирования.

- Контроллер не будет выполнять никаких других операторов в процедуре, пока не завершит цикл.
- Если время, требующееся для завершения цикла, больше, чем время контролирующего таймера для этой задачи, произойдет основная ошибка.
- Тщательно обдумывайте использование таких конструкций, как IF...THEN.

Описание:

Синтаксис:

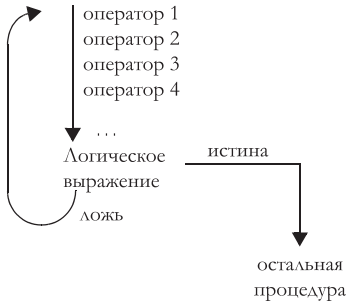
```
REPEAT
   <statement>;
   IF bool_expression2 THEN
     EXIT;
   END_IF;
UNTIL bool_expression1
END_REPEAT;
```

Операторы для выполнения, пока bool_expression1 имеет значение "ложь".

Необязательно

Если существуют условия, при который вы хотите выйти из цикла раньше, используйте такую конструкцию как IF...THEN, для задания условий перехода к оператору EXIT.

Следующая схема демонстрирует выполнение цикла REPEAT...UNTIL и как использовать оператор EXIT для более раннего выхода из цикла.



Пока логическое выражение bool_expression сохраняет значение «ложь», контроллер выполняет только операторы внутри цикла REPEAT...UNTIL .

Для остановки цикла, когда сохраняется значение «истина», используйте оператор EXIT.

Пример 1:

Если вы хотите, чтобы:	Введите структурированный текс:

Цикл REPEAT...UNTIL выполнял операторы конструкции, а затем, перед выполнением этих операторов еще раз, проверял, имеют ли условия значение «истина».
Цикл REPEAT...UNTIL отличается от цикла WHILE...DO, потому что WHILE...DO сначала оценивает условия. Если эти условия имеют значение «истина», контроллер выполняет операторы внутри цикла. Операторы в цикле REPEAT...UNTIL всегда выполняются хотя бы один раз. Операторы в цикле WHILE...DO могут не выполняться ни разу.
	`pos := -1;`
	`REPEAT`
	`  pos := pos + 2;`
	`UNTIL ((pos = 101) OR (structarray[pos].value = targetvalue))`
	`end_repeat;`

Пример 2:

Если вы хотите:	Введите структурированный текст:

Переместить символы ASCII из массива SINT в строковый тег. (В массиве SINT каждый элемент содержит один символ). Остановиться при достижении символа возврата каретки.
1. Присвойте 0 значению *Element_number* (номер элемента).
2. Подсчитайте количество элементов в массиве *SINT_array* (массив, содержащий символы ASCII) и сохрагиете результат в *SINT_array_size* (тег DINT).
3. Присвойте *String_tag[element_number]* значение, равное символу в *SINT_array[element_number]*.
4. Прибавьте 1 к *element_number*. Это позволит контроллеру проверить следующий символ в *SINT_array*.
5. Присвойте члену Length тега *String_tag* значение *element_number*. (Это запишет количество символов в *String_tag*).
6. Если *element_number* = *SINT_array_size*, остановитесь. (Вы в конце массива и он не содержит символа возврата каретки.)
7. Если символ в SINT_array[element_number] = 13 (десятичный код возврата каретки), то остановитесь. В противном случае, перейдите к пункту 3.
	`element_number := 0;`
	`SIZE(SINT_array, 0, SINT_array_size);`
	`Repeat`
	`  String_tag.DATA[element_number] := SINT_array[element_number];`
	`  element_number := element_number + 1;`
	`  String_tag.LEN := element_number;`
	`  If element_number = SINT_array_size then`
	`  exit;`
	`  end_if;`
	`  Until SINT_array[element_number] = 13`
	`end_repeat;`

Комментарии

Используйте комментарии для того, чтобы сделать структурированный текст более понятным при чтении.

- Комментарии позволяют вам использовать простой язык для описания работы структурированного текста.
- Комментарии не влияют на выполнение программы.

Чтобы добавить комментарии в структурированный текст:

Чтобы добавить комментарий:	**Используйте следующие форматы:**
на одной строке	//comment
в конце строки структурированного текста	(*comment*) /*comment*/
внутри одной строки структурированного текста	(*comment*) /*comment*/
который занимает более одной строки	(*начало комментария . . .конец комментария*) /* начало комментария . . . конец комментария*/

Например:

Формат:	**Пример:**

//comment	В начале строки
	//Check conveyor belt direction
	IF conveyor_direction THEN...
	
	В конце строки
	ELSE //If conveyor isn't moving, set alarm light
	light := 1;
	END_IF;

(*comment*)	Sugar.Inlet[:=]1;(*open the inlet*) IF Sugar.Low (*low level LS*)& Sugar.High (*high level LS*)THEN... (*Controls the speed of the recirculation pump. The speed depends on the temperature in the tank.*) IF tank.temp > 200 THEN...
/*comment*/	Sugar.Inlet:=0;/*close the inlet*/ IF bar_code=65 /*A*/ THEN... /*Gets the number of elements in the Inventory array and stores the value in the Inventory_Items tag*/ SIZE(Inventory,0,Inventory_Items);

A

D

E

F

P

R

S

T

U

W

X

How Are We Doing?

Your comments on our technical publications will help us serve you better in the future. Thank you for taking the time to provide us feedback.

You can complete this form and mail it back to us, visit us online at www.ab.com/manuals, or email us at RADocumentComments@ra.rockwell.com

Pub. Title/Type Logix5000™ Controllers General Instructions

Cat. No. 1756 ControlLogix®, 1769 CompactLogix™, 1789 SoftLogix™, 1794 FlexLogix™, PowerFlex 700S with DriveLogix

Pub. No. 1756-RM003G-EN-P Pub. Date June 2003 Part No. 957726-86

Please complete the sections below. Where applicable, rank the feature (1=needs improvement, 2=satisfactory, and 3=outstanding).

Overall Usefulness	1	2	3	How can we make this publication more useful for you?
Completeness (all necessary information is provided)	1	2	3	Can we add more information to help you? procedure/step illustration feature example guideline other explanation definition
Technical Accuracy (all provided information is correct)	1	2	3	Can we be more accurate? text illustration
Clarity (all provided information is easy to understand)	1	2	3	How can we make things clearer?
Other Comments				You can add additional comments on the back of this form.

Your Name _____ Location/Phone _____

Your Title/Function _____ Would you like us to contact you regarding your comments?

___No, there is no need to contact me

___Yes, please call me

___Yes, please email me at _____

___Yes, please contact me via _____

Return this form to: Allen-Bradley Marketing Communications, 1 Allen-Bradley Dr., Mayfield Hts., OH 44124-9705

Phone: 440-646-3176 Fax: 440-646-3525 Email: RADocumentComments@ra.rockwell.com

PLEASE FASTEN HERE (DO NOT STAPLE)

Other Comments

PLEASE FOLD HERE

PLEASE REMOVE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 18235 CLEVELAND OH

POSTAGE WILL BE PAID BY THE ADDRESSEE

Allen-Bradley
RELIANCE ELECTRIC
DODGE
ROCKWELL SOFTWARE
Rockwell Automation

1 ALLEN-BRADLEY DR
MAYFIELD HEIGHTS OH 44124-9705

ASCII Character Codes (ASCII коды)

Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex
[ctrl-@] NUL	0	$00	SPACE	32	$20	@	64	$40	‘	96	$60
[ctrl-A] SOH	1	$01	!	33	$21	A	65	$41	a	97	$61
[ctrl-B] STX	2	$02	“	34	$22	B	66	$42	b	98	$62
[ctrl-C] ETX	3	$03	#	35	$23	C	67	$43	c	99	$63
[ctrl-D] EOT	4	$04	$	36	$24	D	68	$44	d	100	$64
[ctrl-E] ENQ	5	$05	%	37	$25	E	69	$45	e	101	$65
[ctrl-F] ACK	6	$06	&	38	$26	F	70	$46	f	102	$66
[ctrl-G] BEL	7	$07	‘	39	$27	G	71	$47	g	103	$67
[ctrl-H] BS	8	$08	(	40	$28	H	72	$48	h	104	$68
[ctrl-I] HT	9	$09	)	41	$29	I	73	$49	i	105	$69
[ctrl-J] LF	10	$I ($0A)	*	42	$2A	J	74	$4A	j	106	$6A
[ctrl-K] VT	11	$0B	+	43	$2B	K	75	$4B	k	107	$6B
[ctrl-L] FF	12	$0C	,	44	$2C	L	76	$4C	l	108	$6C
[ctrl-M] CR	13	$r ($0D)	-	45	$2D	M	77	$4D	m	109	$6D
[ctrl-N] SO	14	$0E	.	46	$2E	N	78	$4E	n	110	$6E
[ctrl-O] SI	15	$0F	/	47	$2F	O	79	$4F	o	111	$6F
[ctrl-P] DLE	16	$10	0	48	$30	P	80	$50	p	112	$70
[ctrl-Q] DC1	17	$11	1	49	$31	Q	81	$51	q	113	$71
[ctrl-R] DC2	18	$12	2	50	$32	R	82	$52	r	114	$72
[ctrl-S] DC3	19	$13	3	51	$33	S	83	$53	s	115	$73
[ctrl-T] DC4	20	$14	4	52	$34	T	84	$54	t	116	$74
[ctrl-U] NAK	21	$15	5	53	$35	U	85	$55	u	117	$75
[ctrl-V] SYN	22	$16	6	54	$36	V	86	$56	v	118	$76
[ctrl-W] ETB	23	$17	7	55	$37	W	87	$57	w	119	$77
[ctrl-X] CAN	24	$18	8	56	$38	X	88	$58	x	120	$78
[ctrl-Y] EM	25	$19	9	57	$39	Y	89	$59	y	121	$79
[ctrl-Z] SUB	26	$1A	:	58	$3A	Z	90	$5A	z	122	$7A
ctrl-[ESC	27	$1B	;	59	$3B	[	91	$5B	{	123	$7B
[ctrl-\] FS	28	$1C	<	60	$3C	\	92	$5C	|	124	$7C
ctrl-] GS	29	$1D	=	61	$3D	]	93	$5D	}	125	$7D
[ctrl-^] RS	30	$1E	>	62	$3E	^	94	$5E	~	126	$7E
[ctrl-_] US	31	$1F	?	63	$3F	_	95	$5F	DEL	127	$7F

Поддержка Rockwell Automation

Rockwell Automation предоставляет информацию в интернете для того, чтобы помочь вам в использовании наших продуктов. По адресу http://support.rockwellautomation.com вы найдете технические руководства, базу FAQ, заметки по поводу использования, простые программы для архивирования и систему MySupport, которую можете настроить под себя для лучшего использования предоставляемых возможностей.

Дополнительно к телефонной поддержке при установке, конфигурировании и поиск неисправностей, мы предлагаем программы TechConnect Support. За боле подробной информацией вы можете обратиться к местным дистрибьюторам, представителям Rockwell Automation или на сайт http://support.rockwellautomation.com.

Помощь при установке

Если у вас есть проблемы с аппаратными средствами в первые 24 часа установки, пожалуйста, еще раз обратитесь к данному руководству. Вы так же можете обратиться в службу поддержки пользователей по адресу:

США	1.440.646.3223 Понедельник – пятница, 8am – 5pm EST
Для других стран	Пожалуйста, обращайтесь к местным представителям Rockwell Automation за любой технической поддержкой.

Возврат продуктов

Компания Rockwell проверяет все свои продукты на предмет работоспособности после доставки потребителю. Однако, если ваш продукт не работает в полном объеме и должен быть возвращен:

США	Обратитесь к дистрибьютору. Для организации возврата вы должны сообщить дистрибьютору номер Customer Support (см. телефонный номер выше, чтобы получить его).
Для других стран	Обратитесь к местному представителю Rockwell Automation.

www.rockwellautomation.com

Corporate Headquarters
Rockwell Automation, 777 East Wisconsin Avenue, Suite 1400, Milwaukee, WI, 53202-5302 USA, Tel: (1) 414.212.5200, Fax: (1) 414.212.5201

Headquarters for Allen-Bradley Products, Rockwell Software Products and Global Manufacturing Solutions
Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe: Rockwell Automation SA/NV, Vorstlaan/Boulevard du Souverain 36-BP 3A/B, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Headquarters for Dodge and Reliance Electric Products
Americas: Rockwell Automation, 6040 Ponders Court, Greenville, SC 29615-4617 USA, Tel: (1) 864.297.4800, Fax: (1) 864.281.2433
Europe: Rockwell Automation, Brühlstraße 22, D-74834 Elztal-Dallau, Germany, Tel: (49) 6261 9410, Fax: (49) 6261 17741
Asia Pacific: Rockwell Automation, 55 Newton Road, #11-01/02 Revenue House, Singapore 307987, Tel: (65) 351 6723, Fax: (65) 355 1733

Publication 1756-RM003G-EN-P - June 2003
Supersedes Publication 1756-RM003F-EN-P - May 2002

PN 957726-86

Связывайтесь с нами теперь по адресу www.rockwellautomation.com

Всякий раз, когда вы нуждаетесь в нас, Rockwell Automation осуществляет комплексное использование ведущих марок в промышленной автоматике, включающих элементы управления Allen-Bradley, продукты подачи питания Reliance Electric, механические компоненты подачи питания Dodge и Rockwell Software. Уникальный гибкий подход Rockwell Automation в помощи клиентам достигнуть конкурентного преимущества поддерживается тысячами авторизованными партнерами, дистрибьюторами и системными интеграторами по всему миру.

Представительство Rockwell Automation в Москве: 113054, Москва, Большой Строченовский пер., 22/25, Офис 402
Телефон: (095)956-0464, (095)956-0465; Факс: (095)956-0469; E-mail: software@rockwell.ru, info@rockwell.ru

Americas Headquarters, 1201 South Second Street, Milwaukee, WI 53204, USA, Tel: (1) 414 382-2000, Fax: (1) 414 382-4444
European Headquarters SA/NV, avenue Herrmann Debroux, 46, 1160 Brussels, Belgium, Tel: (32) 2 663 06 00, Fax: (32) 2 663 06 40
Asia Pacific Headquarters, 27/F Citicorp Centre, 18 Whitfield Road Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Публикация /534ÅPK. .1EÅPSÅN - Июнь 0. .1