

Команды SIMATIC

9

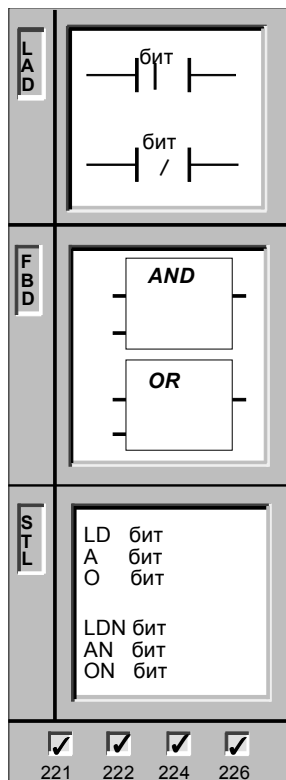
Эта глава описывает набор команд SIMATIC для S7-200.

Обзор главы

Раздел	Описание	Стр.
9.1	Битовые логические команды SIMATIC	9–2
9.2	Команды SIMATIC для операций сравнения	9–10
9.3	Таймерные команды SIMATIC	9–15
9.4	Команды SIMATIC для операций счета	9–23
9.5	Команды SIMATIC, выполняемые над часами	9–71
9.6	Арифметические команды SIMATIC над целыми числами	9–73
9.7	Арифметические команды SIMATIC над вещественными числами	9–82
9.8	Команды SIMATIC с числовыми функциями	9–85
9.9	Команды SIMATIC для пересылки	9–105
9.10	Табличные команды SIMATIC	9–110
9.11	Логические команды SIMATIC	9–117
9.12	Команды SIMATIC для сдвига и циклического сдвига	9–123
9.13	Команды SIMATIC для выполнения преобразований	9–133
9.14	Команды SIMATIC для управления программой	9–148
9.15	Команды SIMATIC для организации прерываний и связи	9–171
9.16	Команды SIMATIC, выполняемые над логическим стеком	9–199

9.1 Битовые логические команды SIMATIC

Стандартные контакты



Эти команды получают значение из памяти или из регистра образа процесса, если типом данных является I или Q. В блоках AND [И] и OR [ИЛИ] можно использовать не более семи входов.

Нормально открытый контакт замкнут (включен), когда бит равен 1.

Нормально замкнутый контакт замкнут (включен), когда бит равен 0.

В LAD нормально открытый и нормально замкнутый контакты представлены контактами.

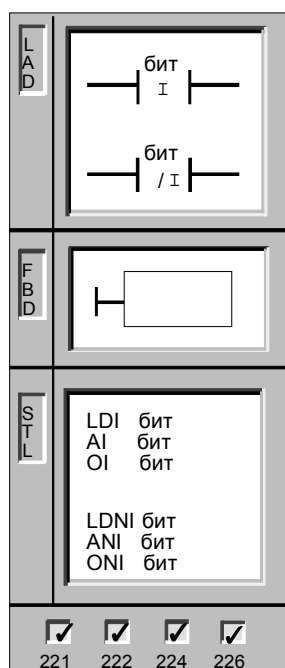
В FBD команды, соответствующие нормально открытым контактам, представлены блоками AND/OR [И/ИЛИ]. Эти команды могут быть использованы для манипулирования булевыми сигналами таким же образом, как контакты LAD. Команды, соответствующие нормально замкнутым контактам, тоже представлены блоками. Команда, соответствующая нормально замкнутому контакту, строится путем помещения символа отрицания на отметке входного сигнала. Количество входов блоков AND [И] и OR [ИЛИ] может быть увеличено максимум до семи.

В STL нормально открытый контакт представляется командами **Загрузить (LD)**, **И (A)** и **ИЛИ (O)**. Эти команды загружают значение адресного бита в вершину стека или выполняют логическое сопряжение значения адресного бита со значением в вершине стека в соответствии с таблицей истинности логического И или ИЛИ.

В STL нормально замкнутый контакт представляется командами **Загрузить инверсное значение (LDN)**, **И-НЕ (AN)** и **ИЛИ-НЕ (ON)**. Эти команды загружают логическое отрицание значения адресного бита в вершину стека или выполняют логическое сопряжение логического отрицания значения адресного бита со значением в вершине стека в соответствии с таблицей истинности логического И или ИЛИ.

Входы/выходы	Операнды	Типы данных
бит (LAD, STL)	I, Q, M, SM, T, C, V, S, L	BOOL
Входы (FBD)	I, Q, M, SM, T, C, V, S, L, поток сигнала	BOOL
Выходы (FBD)	I, Q, M, SM, T, C, V, S, L, поток сигнала	BOOL

Контакты непосредственного опроса



Команда непосредственного опроса при ее исполнении получает значение физического входа, однако, регистр образа процесса не обновляется.

Нормально открытый контакт непосредственного опроса замкнут (включен), когда физический вход (бит) находится в состоянии 1.

Нормально замкнутый контакт непосредственного опроса замкнут (включен), когда физический вход (бит) находится в состоянии 0.

В LAD нормально открытые и нормально замкнутые контакты непосредственного опроса представляются контактами.

В FBD нормально открытому контакту непосредственного опроса соответствует индикатор непосредственного опроса, стоящий перед отметкой операнда. Индикатор непосредственного опроса не может присутствовать, когда используется поток сигнала. Команда может использоваться для манипулирования физическими сигналами таким же образом, как и контакты LAD.

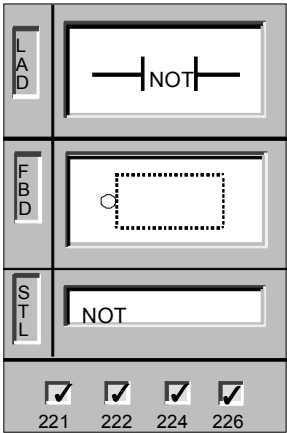
В FBD нормально замкнутому контакту непосредственного опроса также соответствует индикатор непосредственного опроса, стоящий перед отметкой операнда, и символ отрицания. Индикатор непосредственного опроса не может присутствовать, когда используется поток сигнала. Команда, соответствующая нормально замкнутому контакту, строится путем помещения символа отрицания на отметке входного сигнала.

В STL нормально открытый контакт непосредственного опроса представляется командами **Загрузить непосредственно (LDI)**, **Непосредственное И (AI)** и **Непосредственное ИЛИ (OI)**. Эти команды непосредственно загружают значение физического входа в вершину стека или выполняют логическое сопряжение значения физического входа со значением в вершине стека в соответствии с таблицей истинности логического И или ИЛИ.

В STL нормально замкнутый контакт непосредственного опроса представляется командами **Загрузить непосредственно инверсное значение (LDNI)**, **Непосредственное И-НЕ (ANI)** и **Непосредственное ИЛИ-НЕ (ONI)**. Эти команды непосредственно загружают логическое отрицание значения физического входа в вершину стека или выполняют логическое сопряжение отрицания значения физического входа со значением в вершине стека в соответствии с таблицей истинности логического И или ИЛИ.

Входы/выходы	Операнды	Типы данных
бит (LAD, STL)	I	BOOL
вход (FBD)	I	BOOL

NOT (НЕ)



Контакт **NOT** изменяет состояние входа потока сигнала. Если поток сигнала достигает контакта NOT, то он останавливается у контакта. Если поток сигнала не достигает контакта NOT, то на контакте генерируется поток сигнала.

В LAD команда **NOT** изображается как контакт.

В FBD команда **NOT** использует графический символ отрицания с входами булева блока.

В STL команда **NOT** изменяет значение в вершине стека с 0 на 1 или с 1 на 0.

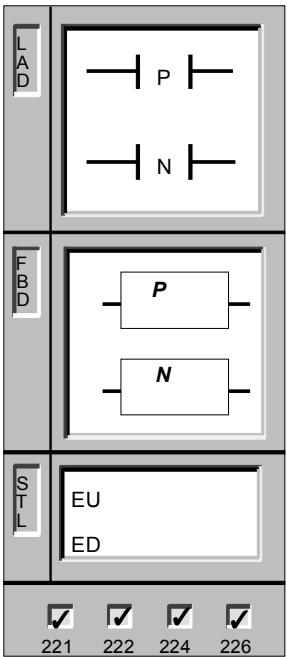
Операнды:

Нет

Типы данных:

Нет

Положительный, отрицательный фронт



Контакт **Положительный фронт** пропускает поток сигнала в течение одного цикла при каждом появлении положительного фронта.

Контакт **Отрицательный фронт** пропускает поток сигнала в течение одного цикла при каждом появлении отрицательного фронта.

В LAD команды Положительный и отрицательный фронт представляются контактами.

В FBD эти команды представляются блоками P и N.

В STL контакту Положительный фронт соответствует команда **Нарастающий фронт (EU = Edge Up)**. При обнаружении перехода значения в вершине стека с 0 на 1 значение в вершине стека устанавливается в 1; в противном случае оно устанавливается в 0.

В STL контакту Отрицательный фронт соответствует команда **Падающий фронт (ED = Edge Down)**.

При обнаружении перехода значения в вершине стека с 1

на 0 значение в вершине стека устанавливается в 1; в противном случае оно устанавливается в 0.

Входы/выходы	Операнды	Типы данных
IN (FBD)	I, Q, M, SM, T, C, V, S, L, поток сигнала	BOOL
OUT (FBD)	I, Q, M, SM, T, C, V, S, L, поток сигнала	BOOL

Примеры контактов

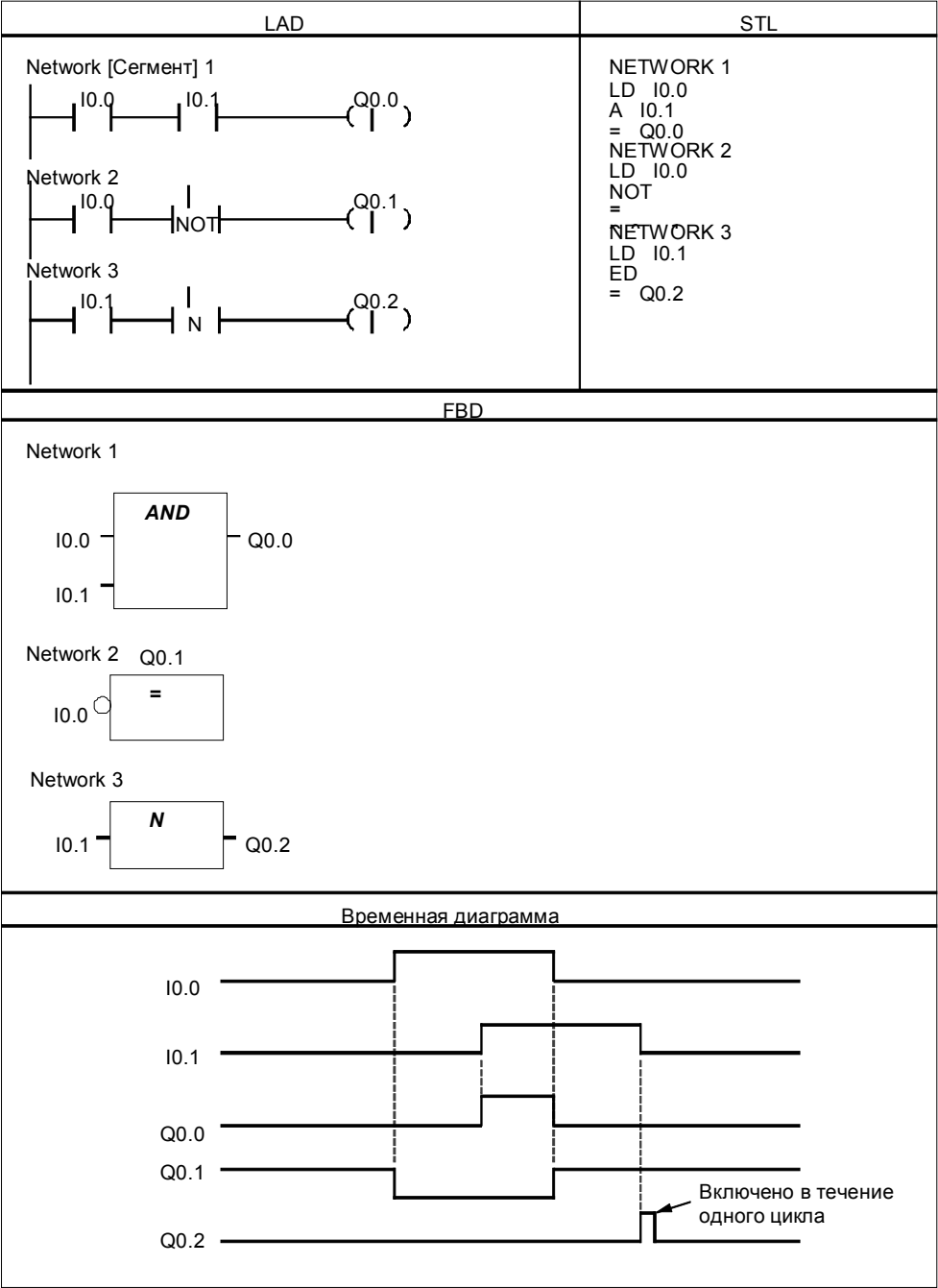
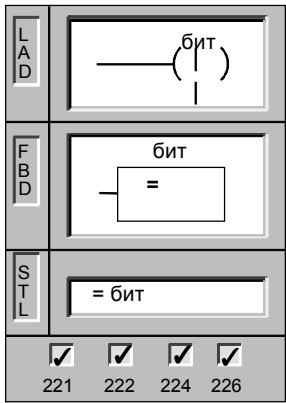


Рис. 9-1. Примеры булевых команд SIMATIC для LAD, STL и FBD

Выход



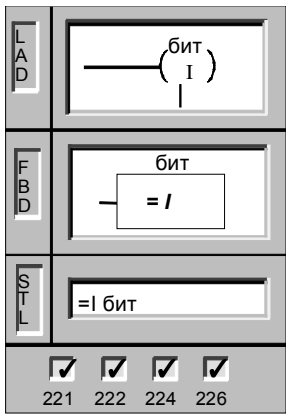
Когда выполняется команда **Выход**, в регистре образа процесса устанавливается выходной бит

В LAD и FBD при выполнении команды Выход указанный бит устанавливается равным потоку сигнала.

В STL команда Выход копирует вершину стека в указанный бит.

Входы/выходы	Операнды	Типы данных
Бит	I, Q, M, SM, T, C, V, S, L	BOOL
Вход (LAD)	Поток сигнала	BOOL
Вход (FBD)	I, Q, M, SM, T, C, V, S, L, поток сигнала	BOOL

Непосредственный выход



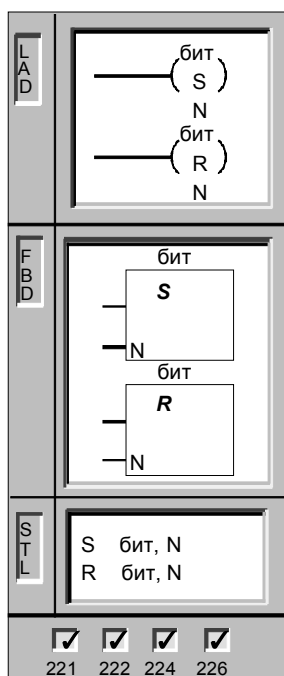
Когда выполняется команда **Непосредственный выход**, физический выход (бит или OUT) устанавливается равным потоку сигнала.

Символ “I” означает непосредственный доступ; при исполнении команды новое значение записывается в физический выход и в соответствующую ячейку регистра образа процесса. Здесь имеется отличие от других видов доступа, которые записывают новое значение только в регистр образа процесса.

В STL Непосредственный выход копирует вершину стека непосредственно в указанный физический выход (бит).

Входы/выходы	Операнды	Типы данных
Бит	Q	BOOL
Вход (LAD)	Поток сигнала	BOOL
Вход (FBD)	I, Q, M, SM, T, C, V, S, L, поток сигнала	BOOL

Установка, сброс (N битов)



Когда выполняются команды **Установка** и **Сброс**, устанавливается (включается) или сбрасывается (выключается) указанное количество разрядов (N), начиная со значения, определенного битом или параметром OUT.

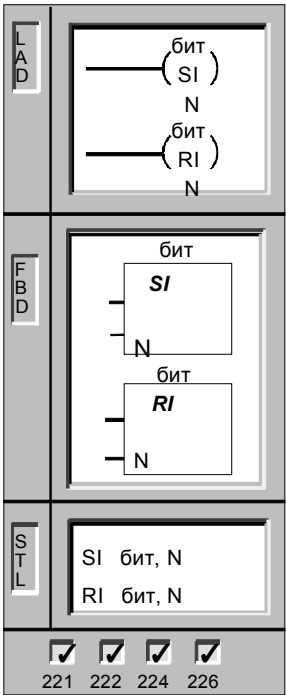
Диапазон разрядов, которые могут быть установлены

или сброшены, составляет от 1 до 255. При использовании команды Сброс, если указанный бит является битом таймера или счетчика, то сбрасывается как таймер или счетчик, так и текущее значение таймера или счетчика.

Ошибки, устанавливающие ENO в 0: SM4.3 (этап исполнения), 0006 (косвенная адресация), 0091 (выход операнда за пределы допустимого диапазона).

Входы/выходы	Операнды	Типы данных
бит	I, Q, M, SM, T, C, V, S, L	BOOL
N	VB, IB, QB, MB, SMB, SB, LB, AC, константа, *VD, *AC, *LD	BYTE

Непосредственная установка, непосредственный сброс (N битов)



Когда выполняются команды **Непосредственная установка** и **Непосредственный сброс**, непосредственно устанавливается (включается) или сбрасывается (выключается) указанное количество (N) физических выходов, начиная с адреса «бит» или OUT.

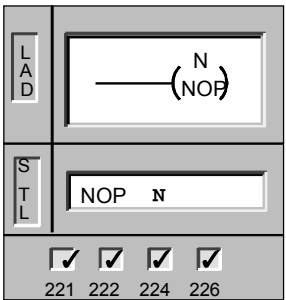
Диапазон выходов, которые могут быть установлены или сброшены, составляет от 1 до 128.

Символ “I” указывает на непосредственный доступ; при выполнении команды новое значение записывается в физический выход и в соответствующую ячейку регистра образа процесса. Здесь имеется отличие от других видов доступа, которые записывают новое значение только в регистр образа процесса.

Ошибки, устанавливающие ENO в 0: SM4.3 (этап выполнения), 0006 (косвенная адресация), 0091 (выход операнда за пределы допустимого диапазона)

Входы/выходы	Операнды	Типы данных
бит	Q	BOOL
N	VB, IB, QB, MB, SMB, SB, LB, AC, константа, *VD, *AC, *LD	BYTE

Пустая операция



Команда **Пустая операция** не оказывает влияния на выполнение программы пользователя. Эта команда отсутствует в FBD. Операнд N – это число от 0 до 255.

Операнды: N: константа (от 0 до 255)

Типы данных: BYTE

Примеры выходов, установки и сброса

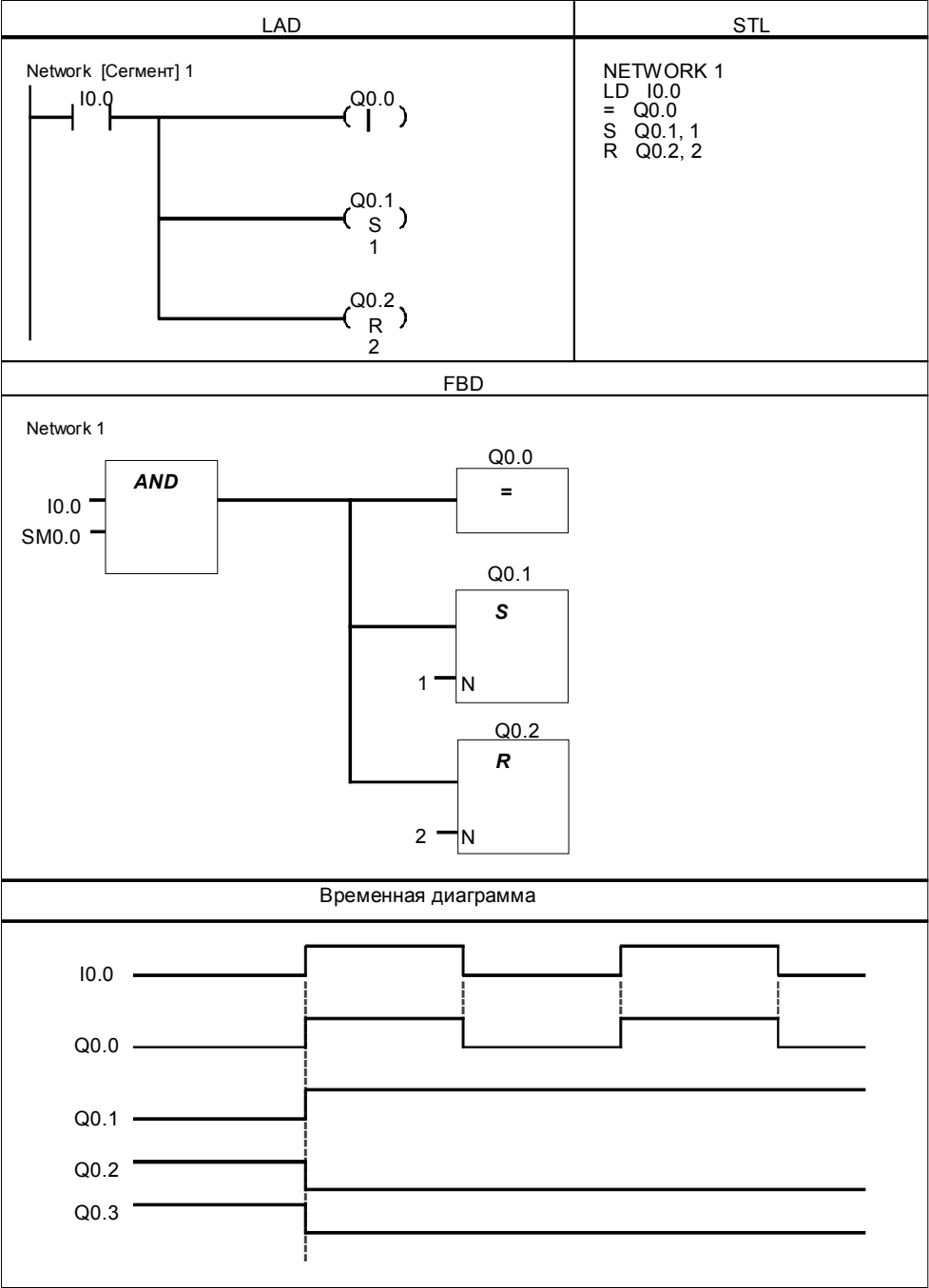
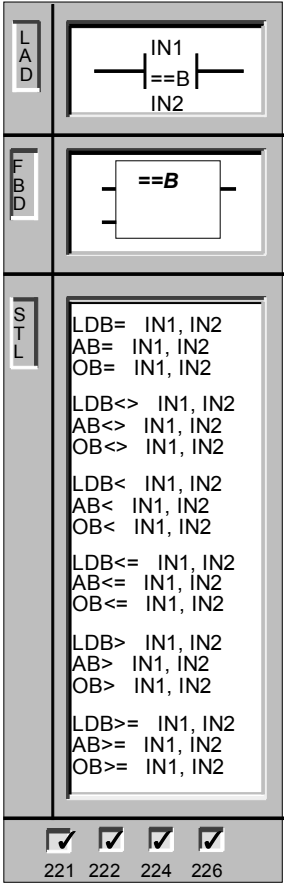


Рис. 9-2. Примеры команд SIMATIC Выход, Установка и Сброс для LAD, STL и FBD

9.2 Команды SIMATIC для операций сравнения

Сравнение байтов



Команда **Сравнить байты** используется для сравнения двух величин: IN1 и IN2. Возможны следующие сравнения: IN1 = IN2, IN1 >= IN2, IN1 <= IN2, IN1 > IN2, IN1 < IN2 и IN1 <> IN2.

Байты сравниваются без знака.

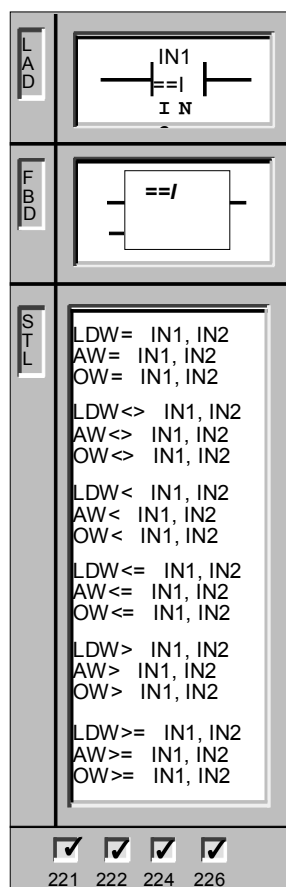
В LAD контакт включен, когда сравнение истинно.

В FBD выход включен, когда сравнение истинно.

В STL, если сравнение истинно, то эти команды загружают «1» в вершину стека или выполняют логическое сопряжение значения «1» со значением в вершине стека в соответствии с таблицей истинности для И или ИЛИ.

Входы/выходы	Операнды	Типы данных
Входы	IB, QB, MB, SMB, VB, SB, LB, AC, константа, *VD, *AC,*LD	BYTE
Выходы (FBD)	I, Q, M, SM, T, C, V, S, L, поток сигнала	BOOL

Сравнение целых чисел



Команда **Сравнить целые числа** используется для сравнения двух величин: $IN1$ и $IN2$.

Возможны следующие сравнения: $IN1 = IN2$, $IN1 >= IN2$,

$IN1 <= IN2$, $IN1 > IN2$, $IN1 < IN2$ и $IN1 <> IN2$.

Целые числа сравниваются с учетом знака ($16\#7FFF > 16\#8000$).

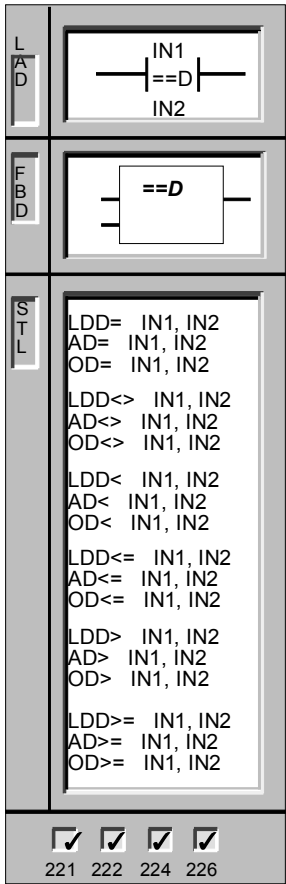
В LAD контакт включен, когда сравнение истинно.

В FBD выход включен, когда сравнение истинно.

В STL, если сравнение истинно, то эти команды загружают «1» в вершину стека или выполняют логическое сопряжение значения «1» со значением в вершине стека в соответствии с таблицей истинности для И или ИЛИ.

Входы/выходы	Операнды	Типы данных
Входы	IW, QW, MW, SW, SMW, T, C, VW, LW, AIW, AC, константа, *VD, *AC, *LD	INT
Выходы (FBD)	I, Q, M, SM, T, C, V, S, L, поток сигнала	BOOL

Сравнение двойных слов



Команда **Сравнить двойные слова** используется для сравнения двух величин: IN1 и IN2. Возможны следующие сравнения: IN1 = IN2, IN1 >= IN2, IN1 <= IN2, IN1 > IN2, IN1 < IN2 и IN1 <> IN2.

Двойные слова сравниваются с учетом знака (16#7FFFFFFF > 16#80000000).

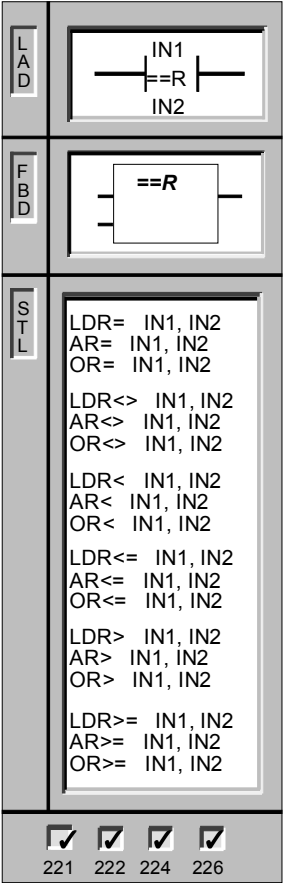
В LAD контакт включен, когда сравнение истинно.

В FBD выход включен, когда сравнение истинно.

В STL, если сравнение истинно, то эти команды загружают «1» в вершину стека или выполняют логическое сопряжение значения «1» со значением в вершине стека в соответствии с таблицей истинности для И или ИЛИ.

Входы/выходы	Операнды	Типы данных
Входы	ID, QD, MD, SD, SMD, VD, LD, HC, AC, константа, *VD, *AC, *LD	DINT
Выходы (FBD)	I, Q, M, SM, T, C, V, S, L, поток сигнала	BOOL

Сравнение вещественных чисел



Команда **Сравнить вещественные числа** используется для сравнения двух величин: N1 и IN2. Возможны следующие сравнения: IN1 = IN2, IN1 >= IN2, IN1 <= IN2, IN1 > IN2, IN1 < IN2 и IN1 <> IN2.

Вещественные числа сравниваются с учетом знака.

В LAD контакт включен, когда сравнение истинно.

В FBD выход включен, когда сравнение истинно.

В STL, если сравнение истинно, то эти команды загружают «1» в вершину стека или выполняют логическое сопряжение значения «1» со значением в вершине стека в соответствии с таблицей истинности для И или ИЛИ.

Входы/выходы	Операнды	Типы данных
Входы	ID, QD, MD,SD, SMD, VD, LD, AC, константа, *VD, *AC, *LD	REAL
Выходы (FBD)	I, Q, M, SM, T, C, V, S, L, поток сигнала	BOOL

Примеры команд сравнения

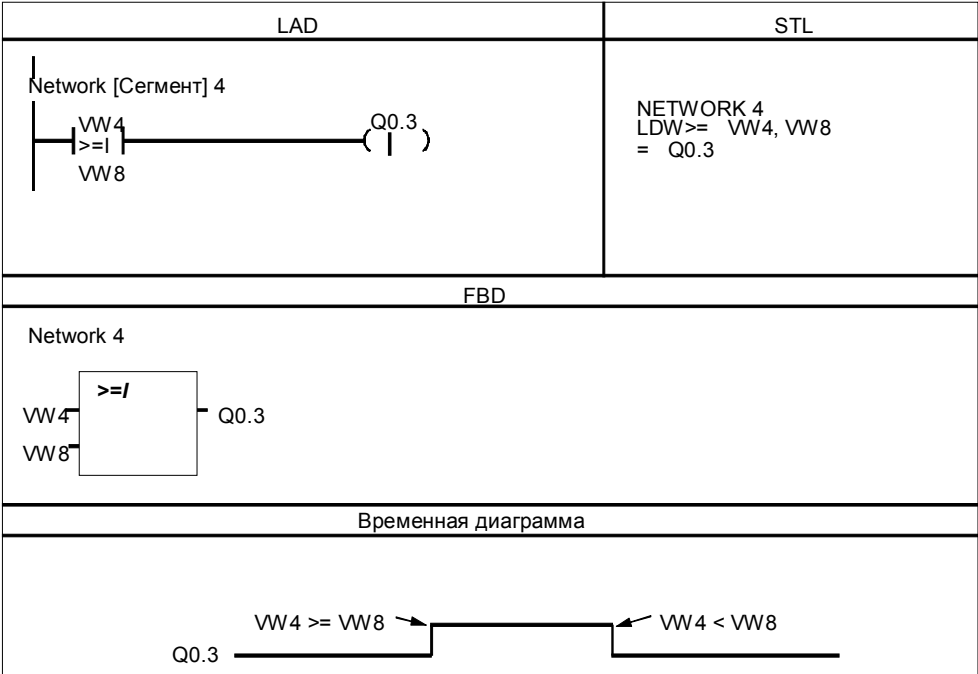
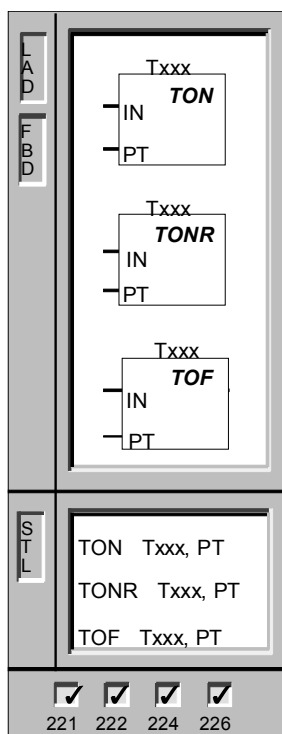


Рис. 9-3. Примеры команд сравнения SIMATIC для LAD, STL и FBD

9.3 Таймерные команды SIMATIC

Таймер с задержкой включения, таймер с задержкой включения с запоминанием, таймер с задержкой выключения



Команды Таймер с задержкой включения (TON) и Таймер с задержкой включения с запоминанием (TONR) отсчитывают время, когда включен разрешающий вход. Когда текущее значение (Txxx) становится больше или равно предустановленному времени (PT), бит таймера устанавливается.

Текущее значение Таймера с задержкой включения сбрасывается, когда выключается разрешающий вход, тогда как текущее значение Таймера с задержкой включения с запоминанием сохраняется, когда этот вход выключается. Вы можете использовать Таймер с задержкой включения с запоминанием для накопления времени за несколько периодов, когда включен разрешающий вход. Для стирания текущего значения Таймера с задержкой включения с запоминанием используется команда Сброс (R).

Таймер с задержкой включения и Таймер с задержкой включения с запоминанием продолжают счет после достижения предустановленного значения, они останавливают счет при достижении максимального значения, равного 32767.

Таймер с задержкой выключения (TOF) используется для задержки выключения выхода на фиксированный интервал времени после выключения входа. Когда включается разрешающий вход, немедленно включается бит таймера, а текущее значение устанавливается в 0. Когда вход выключается, таймер ведет отсчет времени, пока истекшее время не достигнет предустановленного времени. Когда предустановленное время достигнуто, бит таймера сбрасывается, а отсчет текущего значения прекращается. Если выключен в течение более короткого интервала времени, чем предустановленное значение, то бит таймера остается включенным. Команда TOF должна обнаружить переход от включенного состояния к выключенному, чтобы начать отсчет времени.

Если таймер TOF находится внутри области SCR, и область SCR не активна, то текущее значение устанавливается в 0, бит таймера выключается, и отсчет текущего значения не ведется.

Входы/выходы	Операнды	Типы данных
Txxx	Константа	WORD
IN (LAD)	Поток сигнала	BOOL
IN (FBD)	I, Q, M, SM, T, C, V, S, L, поток сигнала	BOOL
PT	VW, IW, QW, MW, SW, SMW, LW, AIW, T, C, AC, константа, *VD, *AC, *LD	INT

Имеется три вида таймеров TON, TONR и TOF, отличающихся разрешением. Разрешение определяется номером таймера, как это показано в таблице 9–1. Любое текущее значение таймера является кратным базы времени. Например, текущее значение 50 на 10–миллисекундном таймере представляет 500 мс.

Таблица 9–1. Номера таймеров и разрешения

Тип таймера	Разрешение в миллисекундах (мс)	Максимальное значение в секундах (с)	Номер таймера
TONR (с запоминанием)	1 мс	32,767 с (0,546 мин.)	T0, T64
	10 мс	327,67 с (0,546 мин.)	T1 – T4, T65 – T68
	100 мс	3276,7 с (0,546 мин.)	T5 – T31, T69 – T95
TON, TOF (без запоминания)	1 мс	32,767 с (0,546 мин.)	T32, T96
	10 мс	327,67 с (0,546 мин.)	T33 – T36, T97 – T100
	100 мс	3276,7 с (0,546 мин.)	T37 – T63, T101 – T255

Примечание

Таймеры TOF и TON не могут иметь один и тот же номер. Например, у вас не могут одновременно использоваться TON T32 и TOF T32.

Описание таймерных команд S7–200

Вы можете использовать таймеры для реализации функций отсчета времени. Набор команд S7–200 предоставляет в ваше распоряжение три типа таймеров, показанных ниже. В таблице 9–2 описаны действия, выполняемые различными таймерами.

- Таймер с задержкой включения (TON) для отсчета одиночного интервала.
- Таймер с задержкой включения с запоминанием (TONR) для накопления отсчитанных интервалов времени.
- Таймер с задержкой выключения (TOF) для увеличения интервала времени после сбойных ситуаций, например, для охлаждения двигателя после его отключения.

Таблица 9–2. Действия таймеров

Тип таймера	Текущее \geq предустановленного	Разрешающий вход включен	Разрешающий вход выключен	Выключение-включение питания / первый цикл
TON	Бит таймера установлен, отсчет текущего значения продолжается до 32 767	Текущее значение отсчитывает время	Бит таймера сброшен, текущее значение = 0	Бит таймера сброшен, текущее значение = 0
TONR	Бит таймера установлен, отсчет текущего значения продолжается до 32 767	Текущее значение отсчитывает время	Бит таймера и текущее значение сохраняют последнее значение	Бит таймера сброшен, текущее значение может ¹ быть сохранено
TOF	Бит таймера сброшен, текущее значение = предустановленному, отсчет времени прекращен	Бит таймера установлен, текущее значение = 0	Отсчет времени ведется после перехода от включенного состояния к выключенному	Бит таймера сброшен, текущее значение = 0

¹ Текущее значение таймера с запоминанием может быть выбрано для сохранения на время выключения-включения питания. Информацию о сохранении памяти для CPU S7–200 вы найдете в разделе 5.3.

Примечание

Для сброса любого таймера может быть использована команда Сброс (R).

Команда Сброс выполняет следующие операции:

Бит таймера сброшен.

Текущее значение = 0.

Таймер TONR может быть сброшен только командой Сброс.

После сброса таймеры TOF, чтобы перезапуститься, требуют наличия разрешающего входа для перехода из включенного состояния в выключенное.

Действия таймеров с различными разрешениями объяснены ниже.

Разрешение 1 миллисекунда

Таймеры с разрешением 1 мс отсчитывают количество интервалов времени длиной 1 мс, прошедших с момента, когда активный 1-миллисекундный таймер был разблокирован. Выполнение таймерной команды запускает отсчет времени; однако 1-миллисекундные таймеры обновляют бит таймера и текущее значение каждую миллисекунду асинхронно с циклом обработки программы. Иначе говоря, бит таймера и текущее значение обновляются несколько раз в течение цикла сканирования, имеющего длительность более 1 мс.

Таймерная команда используется для включения таймера, сброса таймера и, в случае таймера TONR, для выключения таймера.

Так как таймер может быть запущен в любой момент интервала в 1 мс, предустановленное время должно быть установлено на один отсчет времени больше, чем минимальное желаемое время работы таймера. Например, чтобы гарантировать интервал работы таймера не менее 56 мс с помощью 1-миллисекундного таймера, предустановленное значение должно быть равно 57.

Разрешение 10 миллисекунд

Таймеры с разрешением 10 мс отсчитывают количество интервалов времени длиной 10 мс, прошедших с момента, когда активный 10-миллисекундный таймер был разблокирован. Выполнение таймерной команды запускает отсчет времени; однако, 10-миллисекундные таймеры актуализируются в начале каждого цикла сканирования (иными словами, текущее значение таймера и бит таймера остаются неизменными на протяжении цикла) путем добавления накопленного количества 10-миллисекундных интервалов (с начала предыдущего цикла) к текущему значению для активного таймера.

Так как таймер может быть запущен в любой момент интервала в 10 мс, предустановленное время должно быть установлено на один отсчет времени больше, чем минимальное желаемое время работы таймера. Например, чтобы гарантировать интервал работы таймера не менее 140 мс с помощью 10-миллисекундного таймера, предустановленное значение должно быть равно 15.

Разрешение 100 миллисекунд

Таймеры с разрешением 100 мс отсчитывают количество интервалов времени длиной 100 мс, прошедших с момента, когда активный 100-миллисекундный таймер был последний раз актуализирован. Эти таймеры актуализируются путем добавления накопленного количества 100-миллисекундных интервалов (с начала предыдущего цикла сканирования) к текущему значению таймера, когда выполняется таймерная команда.

Текущее значение 100-миллисекундного таймера обновляется только в том случае, если исполняется таймерная команда. Следовательно, если 100-миллисекундный таймер разблокирован, но таймерная команда выполняется не в каждом цикле, то текущее значение для этого таймера не актуализируется, и он теряет время. Аналогично, если один и тот же 100-миллисекундный таймер выполняется несколько раз в одном цикле сканирования, то число 100-миллисекундных интервалов добавляется к текущему значению таймера несколько раз, и он увеличивает свое время. 100-миллисекундные таймеры должны использоваться там, где команда таймера исполняется ровно один раз за цикл сканирования.

Так как таймер может быть запущен в любой момент интервала в 100 мс, предустановленное время должно быть установлено на один отсчет времени больше, чем минимальное желаемое время работы таймера. Например, чтобы гарантировать интервал работы таймера не менее 2100 мс с помощью 100-миллисекундного таймера, предустановленное значение должно быть равно 22.

Обновление текущего значения таймера

Воздействие различных способов обновления текущих значений времени зависит от того, как таймеры используются. Например, рассмотрим работу таймера, показанного на рис. 9–4.

- При использовании 1-миллисекундного таймера (1) Q0.0 включается на один цикл всякий раз, когда обновляется текущее значение таймера, а именно, после срабатывания нормально замкнутого контакта T32 и перед срабатыванием нормально открытого контакта T32.

- При использовании 10-миллисекундного таймера (2) Q0.0 никогда не включается, так как бит таймера T33 включен с начала цикла до момента времени, когда выполняется таймерный блок. После выполнения таймерного блока текущее значение и бит таймера сбрасываются в ноль. Если срабатывает нормально открытый контакт T33, то T33 выключается и Q0.0 тоже выключается.
- При использовании 100-миллисекундного таймера (3) Q0.0 включается на один цикл всякий раз, когда текущее значение таймера достигает предустановленного значения.

При использовании в качестве разрешающего входа в блок таймера нормально замкнутого контакта Q0.0 вместо бита таймера гарантируется, что выход Q0.0 будет включен на один цикл каждый раз, как таймер достигнет предустановленного значения.

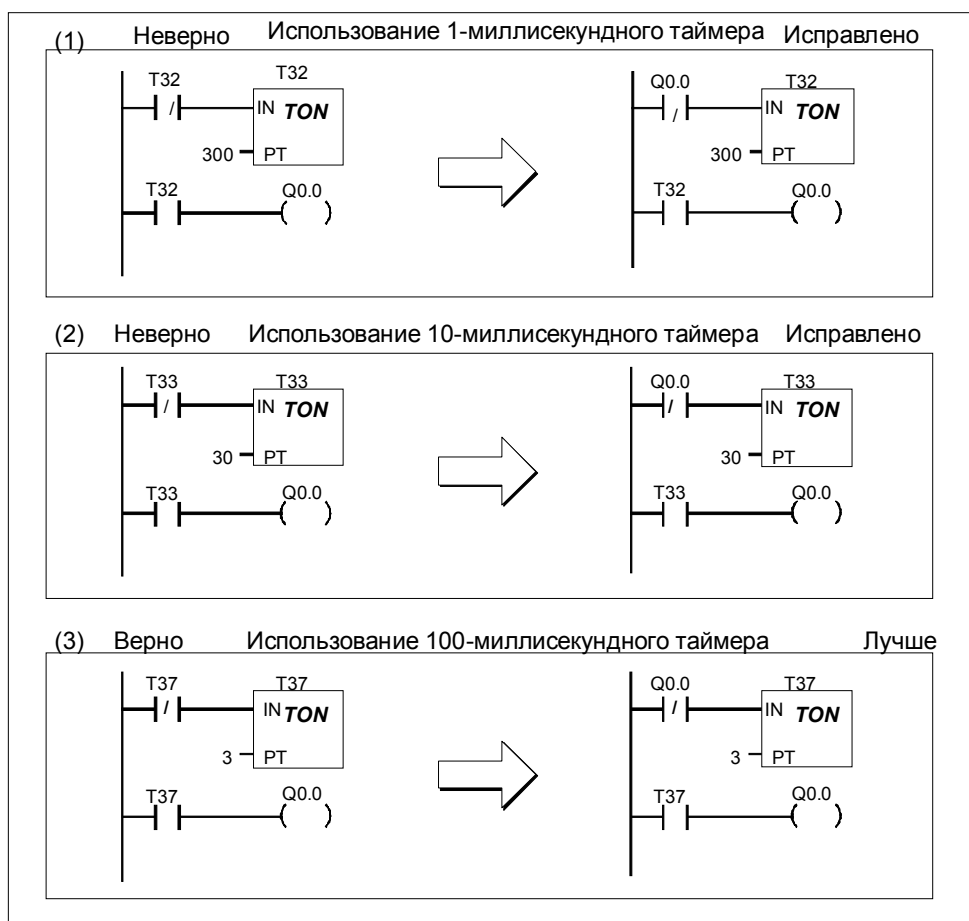


Рис. 9-4. Пример автоматического перезапуска таймера

Пример таймера с задержкой включения

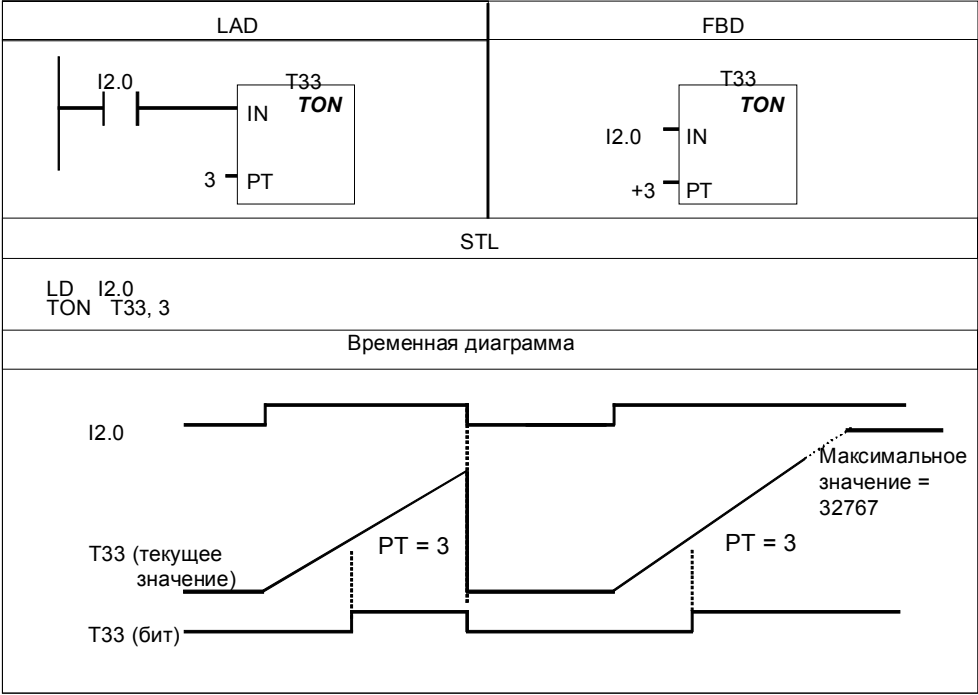


Рис. 9-5. Пример команды SIMATIC «Таймер с задержкой включения» для LAD, FBD и STL

Пример таймера с задержкой включения с запоминанием

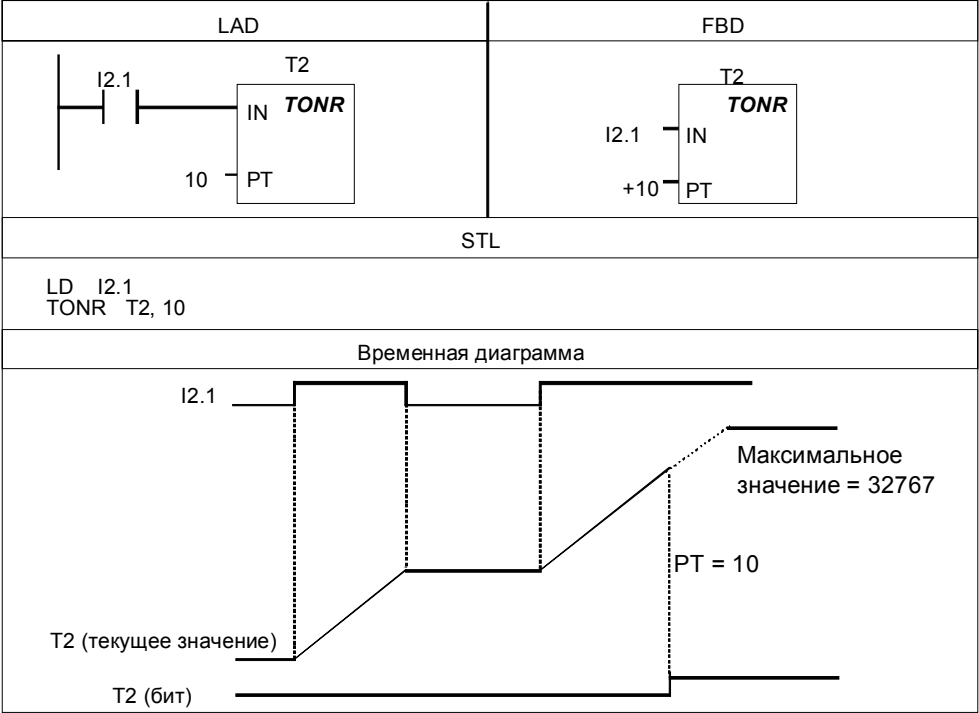


Рис. 9-6. Пример команды SIMATIC «Таймер с задержкой включения с запоминанием» для LAD, FBD и STL

Пример таймера с задержкой выключения

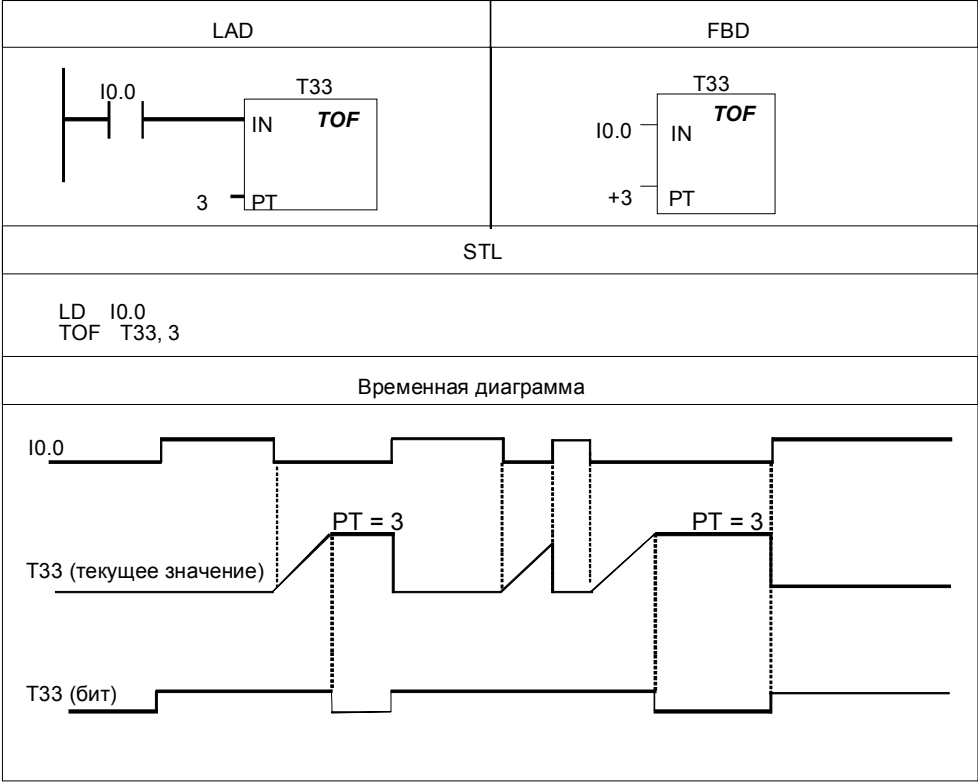
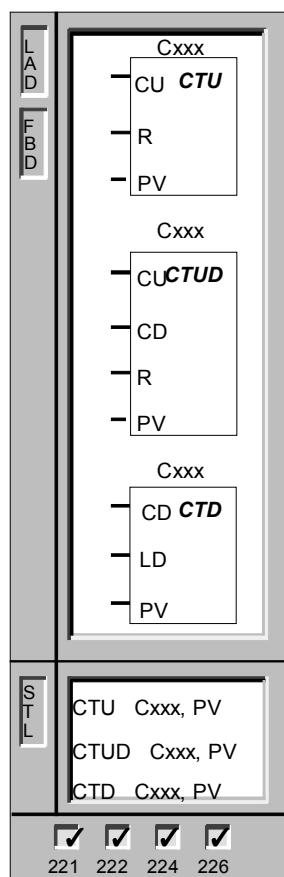


Рис. 9-7. Пример команды SIMATIC «Таймер с задержкой выключения» для LAD, FBD и STL

9.4 Команды SIMATIC для операций счета

Прямой, реверсивный и обратный счет



Команда **Прямой счет** увеличивает значение счетчика вплоть до максимального значения при появлении нарастающих фронтов сигнала на входе CU (**C**ount **U**p = Прямой счет). Когда текущее значение (Cxxx) больше или равно предустановленному значению (PV), бит счетчика (Cxxx) устанавливается. Счетчик сбрасывается, когда включается вход сброса ®. Он прекращает счет при достижении PV.

Команда **Реверсивный счет** увеличивает значение счетчика при появлении нарастающих фронтов сигнала на входе CU (**C**ount **U**p = Прямой счет). Она уменьшает значение счетчика при появлении нарастающих фронтов сигнала на входе CD (**C**ount **D**own = Обратный счет). Когда текущее значение (Cxxx) больше или равно предустановленному значению (PV), бит счетчика (Cxxx) устанавливается. Счетчик сбрасывается, когда включается вход сброса ®.

Команда **Обратный счет** уменьшает значение счетчика от предустановленного значения при появлении нарастающих фронтов сигнала на входе CD (**C**ount **D**own = Обратный счет). Когда текущее значение равно нулю, бит счетчика (Cxxx) включается. Счетчик сбрасывает свой бит (Cxxx) и загружает текущее значение предустановленным значением (PV), когда включается вход загрузки (LD). Обратный счет прекращается при достижении нуля.

Область счетчиков: $Cxxx = C0 \div C255$

В STL входу сброса CTU соответствует значение, находящееся в вершине стека, а входу Прямой счет – значение, загруженное во второй уровень стека.

В STL входу сброса CTUD соответствует значение, находящееся в вершине стека, входу Обратный счет – значение, загруженное во второй уровень стека, а входу Прямой счет – значение, загруженное в третий уровень стека.

В STL входу загрузки CTD соответствует вершина стека, а входу Обратный счет – значение, загруженное во второй уровень стека.

Входы/выходы	Операнды	Типы данных
Cxxx	Константа	WORD
CU, CD, LD, R (LAD)	Поток сигнала	BOOL
CU, CD, R, LD (FBD)	I, Q, M, SM, T, C, V, S, L, поток сигнала	BOOL
PV	VW, IW, QW, MW, SMW, LW, SW, AIW, AC, T, C, константа, *VD, *AC, *LD	INT

Описание команд счета S7–200

Прямой счетчик (CTU) увеличивает свое текущее значение каждый раз, когда на входе прямого счета происходит переход от выключенного состояния к включенному. Счетчик сбрасывается, когда включается вход сброса или когда выполняется команда Сброс. Счетчик останавливается при достижении максимального значения (32 767).

Реверсивный счетчик (CTUD) увеличивает свое значение каждый раз, когда на входе прямого счета происходит переход от выключенного состояния к включенному, и уменьшает свое значение каждый раз, когда переход от выключенного состояния к включенному происходит на входе обратного счета. Счетчик сбрасывается, когда включается вход сброса или когда выполняется команда Сброс. При достижении максимального значения

(32 767) следующий нарастающий фронт на входе прямого счета вызывает переход к минимальному значению (-32 768). Аналогично, при достижении минимального значения (-32 768) следующий нарастающий фронт на входе обратного счета вызывает переход к максимальному значению (32 767).

Прямой и реверсивный счетчики имеют текущее значение, в котором хранится текущее значение счета. У них есть также предустановленное значение (PV), которое сравнивается с текущим значением всякий раз, когда выполняется команда счета. Если текущее значение больше или равно предустановленному значению, то бит счетчика (C–бит) устанавливается. В противном случае C–бит выключается.

Обратный счетчик уменьшает свое текущее значение каждый раз, когда на входе обратного счета происходит переход от выключенного состояния к включенному. Счетчик сбрасывает бит счетчика и загружает текущее значение предустановленным значением, когда включается вход загрузки. Счетчик останавливается при достижении нуля, его бит (C–бит) при этом устанавливается.

Если вы сбрасываете счетчик с помощью команды Сброс, то бит счетчика сбрасывается, а текущее значение устанавливается в ноль. Для обращения как к текущему значению, так и к C-биту используется номер счетчика.

Примечание

Так как для каждого счетчика имеется только одно текущее значение, не назначайте один и тот же номер более, чем одному счетчику (прямые, реверсивные и обратные счетчики с одним и тем же номером обращаются к одному и тому же текущему значению).

Примеры счетчиков

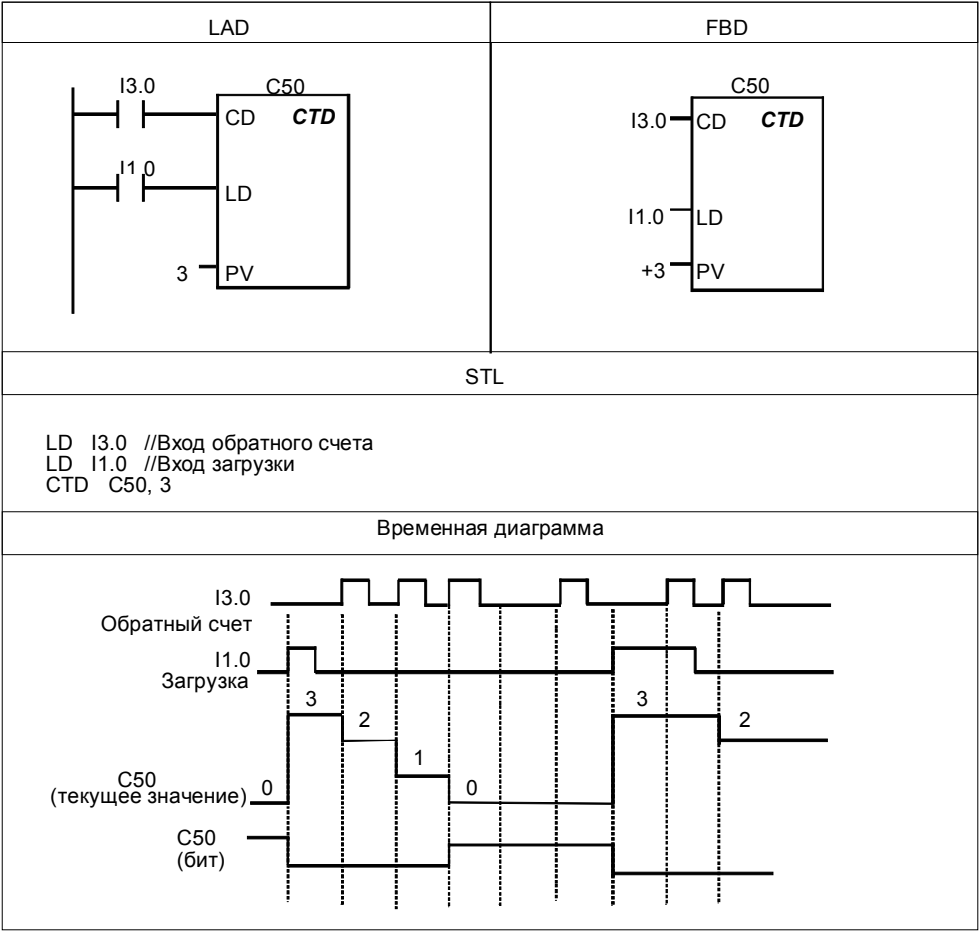


Рис. 9–8. Пример команды SIMATIC «Обратный счетчик» (CTD) для LAD, FBD и STL

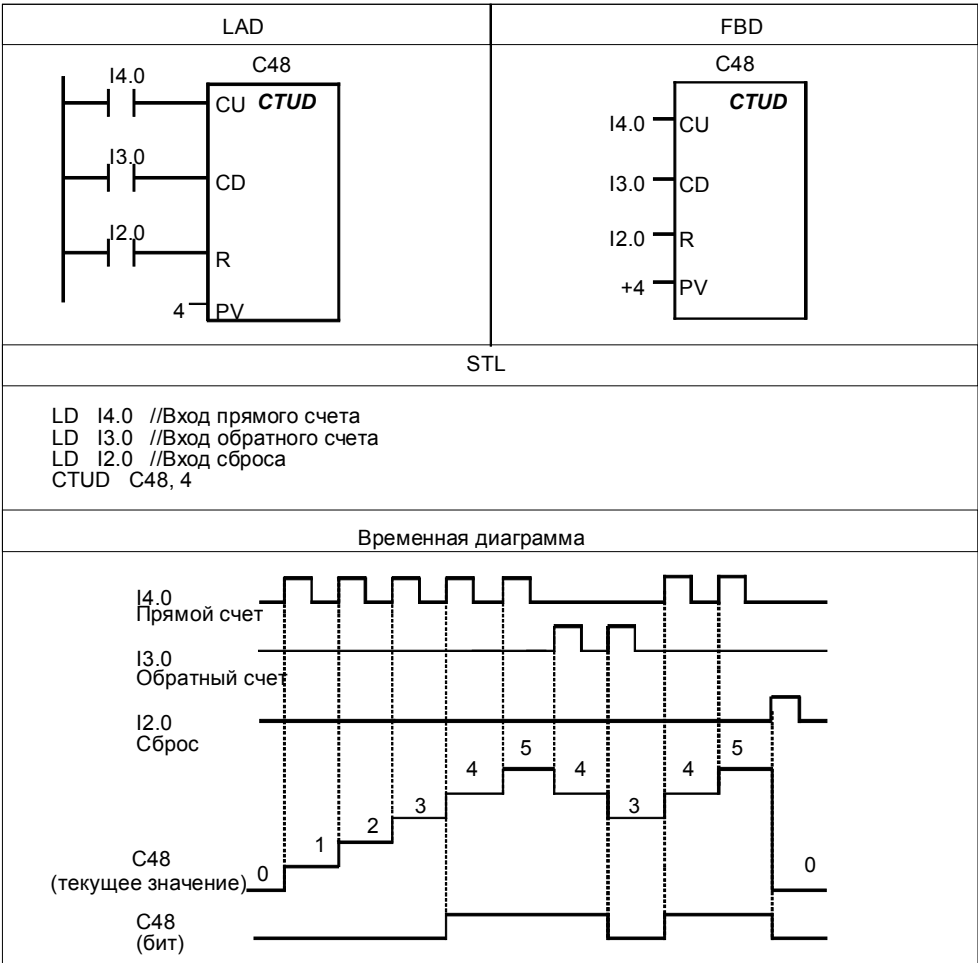
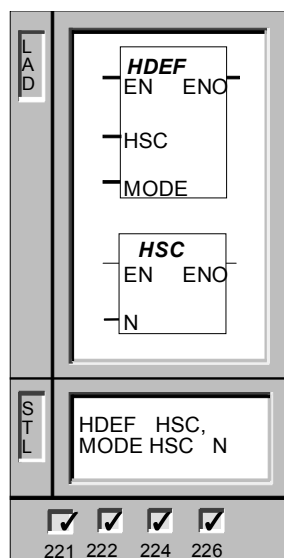


Рис. 9-9. Пример команды SIMATIC «Реверсивный счетчик» (CTUD) для LAD, FBD и STL

9.5 Команды SIMATIC для скоростного счета

Определение скоростного счетчика, Скоростной счетчик



Команда **Определение скоростного счетчика (HDEF)** назначает режим(MODE) скоростному счетчику (HSC), к которому производится обращение. См. табл. 9–5 на стр. 9–33.

Команда **Скоростной счетчик (HSC)** при своем исполнении конфигурирует и управляет режимом работы скоростного счетчика, основанном на состоянии специальных битов памяти HSC. Параметр N определяет номер скоростного счетчика.

CPU 221 и CPU 222 не поддерживают HSC1 и HSC2.

На один счетчик может быть использован только один блок HDEF.

HDEF: Ошибки, устанавливающие ENO в 0: SM4.3 (этап исполнения), 0003 (конфликт входов), 0004 (недопустимая команда в прерывании), 000A (повторное определение HSC)

HSC: Ошибки, устанавливающие ENO в 0:

SM4.3 (этап исполнения), 0001 (HSC перед HDEF), 0005 (одновременно HSC/PLS)

Входы/выходы	Операнды	Типы данных
HSC	Константа	BYTE
MODE	Константа	BYTE
N	Константа	WORD

Описание команд Скоростной счетчик

Скоростные счетчики считают происходящие с высокой скоростью события, которые не могут контролироваться при скорости сканирования CPU. Они могут быть сконфигурированы на двенадцать различных режимов работы. Режимы счетчиков перечислены в таблице 9–5. Максимальная частота счета скоростного счетчика зависит от типа вашего CPU. Информацию о вашем CPU вы найдете в Приложении G.

Каждый счетчик имеет специализированные входы, которые поддерживают такие функции, как датчик тактовых импульсов, управление направлением, сброс и запуск. Для двухфазных счетчиков оба датчика тактовых импульсов могут работать со своей максимальной скоростью. В квадратурных режимах предоставляется возможность выбора однократной (1x) или четырехкратной (4x) скорости счета. Все счетчики работают с максимальной скоростью, не создавая помех друг другу.

Использование скоростных счетчиков

Обычно скоростные счетчики используются в качестве привода для счетных механизмов, в которых вал, вращающийся с постоянной скоростью, снабжен угловым шаговым датчиком. Угловой шаговый датчик дает определенное количество отсчетов на оборот, а также импульс сброса один раз за оборот. Датчик (датчики) тактовых импульсов и импульс сброса от углового шагового датчика обеспечивают входы для скоростного счетчика. Скоростной счетчик загружается первым из нескольких предустановленных значений, и желаемые выходы активизируются на интервал времени, в течение которого текущее значение счетчика меньше текущего предустановленного значения. Счетчик настроен таким образом, что, когда текущее значение счетчика становится равным предустановленному значению, или при появлении сброса происходит прерывание.

Когда при равенстве текущего значения счетчика и предустановленного значения происходит прерывающее событие, загружается новое предустановленное значение, и устанавливается следующее состояние для выходов. Когда происходит событие, вызывающее прерывание по сбросу, то устанавливаются первое предустановленное значение и первые состояния выходов, и цикл повторяется.

Так как прерывания происходят со значительно меньшей частотой, чем считает скоростной счетчик, то может быть реализовано точное управление быстрыми операциями при относительно малом воздействии на общий цикл обработки программы программируемого логического контроллера. Метод подключения прерываний позволяет выполнять каждую загрузку нового предустановленного значения в отдельной программе обработки прерывания для облегчения управления состоянием, делая программу простой и легкой для понимания. Конечно, все события, вызывающие прерывания, могут быть обработаны и в одной единственной программе обработки прерываний. За дополнительной информацией о командах прерывания обратитесь к разделу 9.15.

Описание временных диаграмм для скоростных счетчиков

Следующие временные диаграммы (рисунки 9–10 ÷ 9–16) показывают, как работает каждый счетчик в соответствии с режимом. Работа входов сброса и запуска показана на отдельной временной диаграмме, которая применима ко всем режимам, использующим входы сброса и запуска. На диаграммах для входов сброса и запуска активность обоих входов запрограммирована для высокого уровня сигнала.

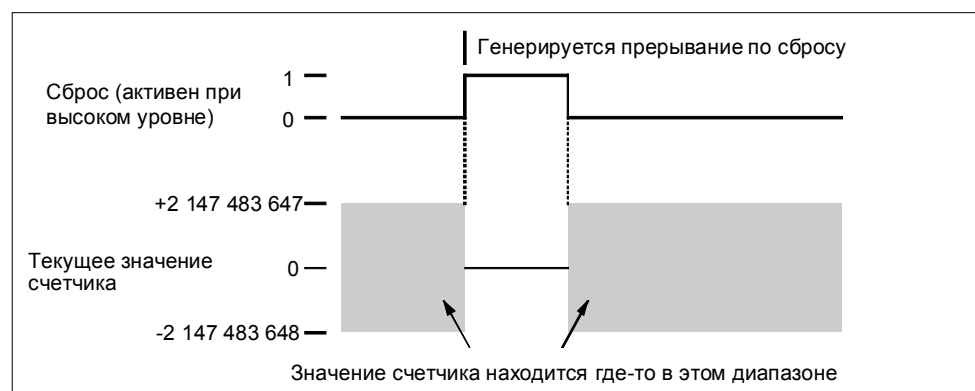


Рис. 9-10. Пример работы счетчика с входом сброса, но без входа запуска

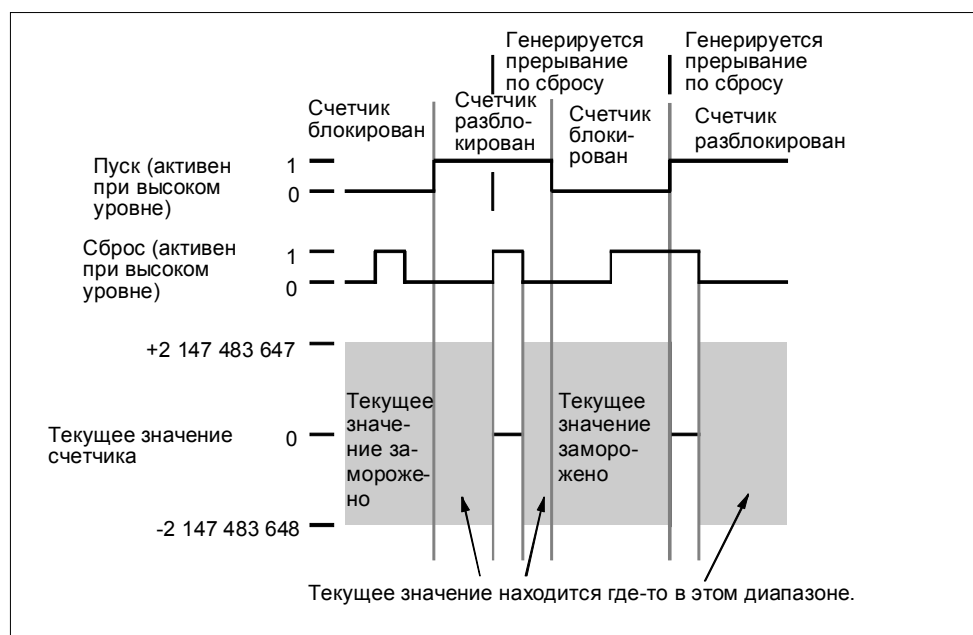


Рис. 9-11. Пример работы счетчика с входами сброса и запуска

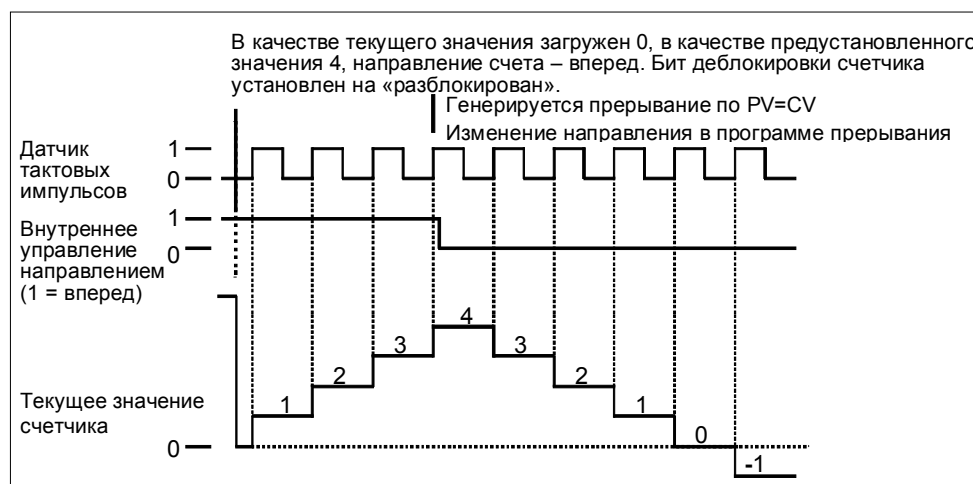


Рис. 9-12. Пример работы в режимах 0, 1 или 2

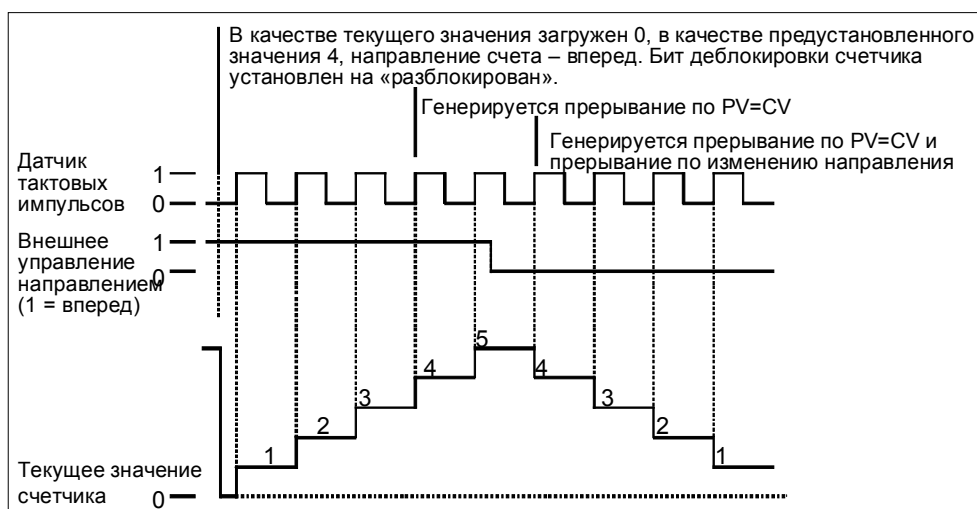


Рис. 9-13. Пример работы в режимах 3, 4 или 5

Когда используются режимы счета 6, 7 или 8, и в течение 0,3 микросекунды друг за другом появляется нарастающий фронт на тактовых входах счета вперед и счета назад, скоростной счетчик может рассматривать эти события как происходящие одновременно. Если это происходит, то текущее значение не меняется и не отображается изменение в направлении счета. Если между поступлениями нарастающих фронтов на тактовые входы счета вперед и счета назад проходит больше 0,3 микросекунды, то скоростной счетчик воспринимает эти события отдельно. В этом случае ошибки не происходит, и счетчик сохраняет правильное счетное значение. См. рисунки 9–14, 9–15 и 9–16.



Рис. 9-14. Пример работы в режимах 6, 7 или 8

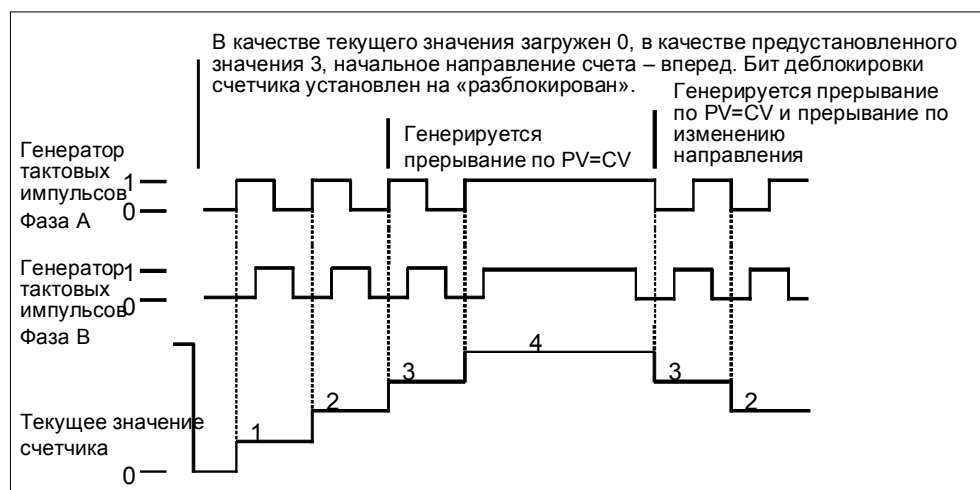


Рис. 9-15. Пример работы в режимах 9, 10 или 11 (квадратурный режим, 1-кратная скорость)

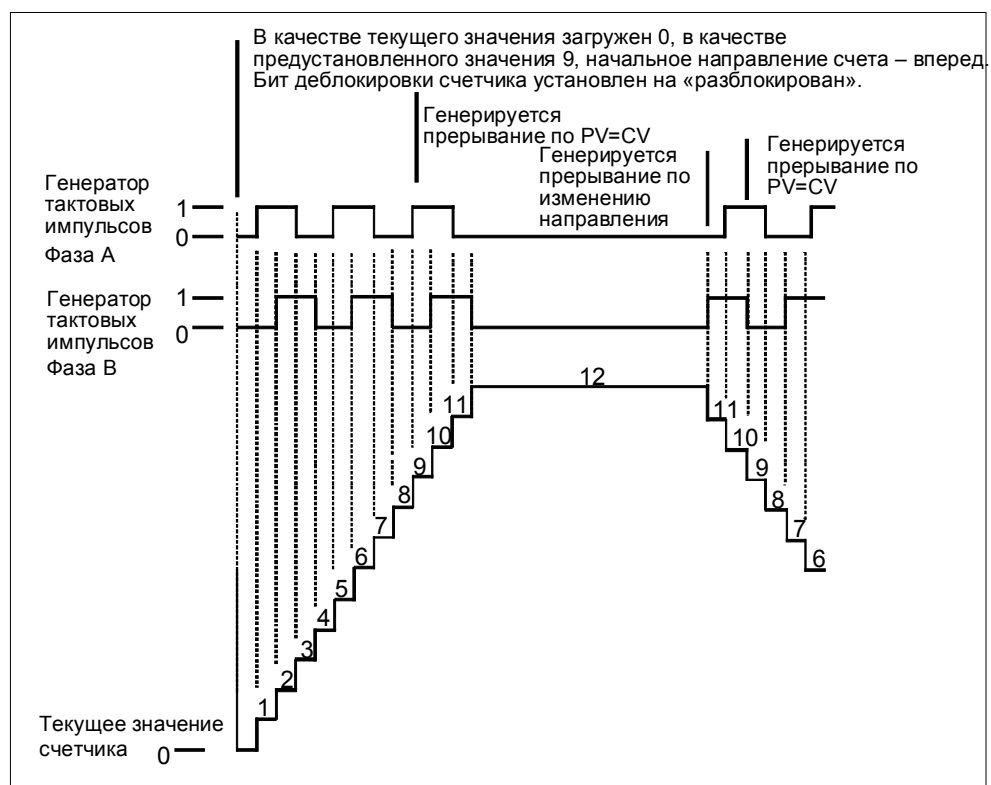


Рис. 9-16. Пример работы в режимах 9, 10 или 11 (квадратурный режим, 4-кратная скорость)

Подключение входов скоростных счетчиков

В таблице 9–3 показаны входы для таких функций скоростных счетчиков, как генератор тактовых импульсов, управление направлением, сброс и запуск. Эти функции входов и режимы работы скоростных счетчиков описаны в таблицах 9–5 ÷ 9–10.

Таблица 9–3. Специализированные входы скоростных счетчиков

Скоростной счетчик	Используемые входы
HSC0	I0.0, I0.1, 0.2
HSC1	I0.6, I0.7, I1.0, I1.1
HSC2	I1.2, I1.3, I1.4, I1.5
HSC3	I0.1
HSC4	I0.3, I0.4, I0.5
HSC5	I0.4

Как показано в выделенной серым цветом области таблицы 9–4, имеется некоторое перекрытие в назначении входов для некоторых скоростных счетчиков и прерываний по фронту сигнала. Один и тот же вход не может быть использован для двух разных функций, но любой вход, не используемый текущим режимом скоростного счетчика, может быть использован для другой цели. Например, если HSC0 используется в режиме 2, который использует I0.0 и I0.2, то I0.1 может быть использован для прерываний по фронту сигнала или для HSC3.

Если используется режим HSC0, который не использует вход I0.1, то этот вход доступен для использования или как HSC3, или для прерываний по фронту сигнала. Аналогично, если I0.2 не используется в выбранном режиме HSC0, то этот вход доступен для прерываний по фронту сигнала; и если I0.4 не используется в выбранном режиме HSC4, то этот вход доступен для HSC5. Примите во внимание, что все режимы HSC0 всегда используют I0.0, а все режимы HSC4 всегда используют I0.3, так что эти входы не бывают доступными для других целей, когда используются данные счетчики.

Таблица 9–4. Назначения входов для скоростных счетчиков и прерываний по фронту сигнала

Вход (I)														
Элемент	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	1.0	1.1	1.2	1.3	1.4	1.5
HSC0	x	x	x											
HSC1							x	x	x	x				
HSC2											x	x	x	x
HSC3		x												
HSC4				x	x	x								
HSC5					x									
Прерывания по фронту	x	x	x	x										

Таблица 9–5. Режимы работы HSC0 (CPU 221, CPU 222, CPU 224 и CPU 226)

HSC0					
Режим	Описание	I0.0	I0.1	I0.2	
0	Однофазный реверсивный счетчик с внутренним управлением направлением SM37.3 = 0, счет назад SM37.3 = 1, счет вперед	Тактовый генератор			
1				Сброс	
3	Однофазный реверсивный счетчик с внешним управлением направлением I0.1 = 0, счет назад I0.1 = 1, счет вперед	Тактовый генератор	Направл.		
4				Сброс	
6	Двухфазный счетчик с тактовыми входами для счета вперед и назад	Такт. генер. (вперед)	Такт. генер. (назад)		
7				Сброс	
9	Квадратурный счетчик с фазами А и В, фаза А опережает В на 90 градусов при вращении по часовой стрелке, фаза В опережает А на 90 градусов при вращении против часовой стрелки	Такт. генер. Фаза А	Такт. генер. Фаза В		
10				Сброс	

Таблица 9–6. Режимы работы HSC1 (CPU 224 и CPU 226)

HSC1					
Режим	Описание	I0.6	I0.7	I1.0	I1.1
0	Однофазный реверсивный счетчик с внутренним управлением направлением SM47.3 = 0, счет назад SM47.3 = 1, счет вперед	Тактовый генератор			
1				Сброс	
2					Пуск
3	Однофазный реверсивный счетчик с внешним управлением направлением I0.7 = 0, счет назад I0.7 = 1, счет вперед	Тактовый генератор	Направл.		
4				Сброс	
5					Пуск
6	Двухфазный счетчик с тактовыми входами для счета вперед и назад	Такт. генер. (вперед)	Такт. генер. (назад)		
7				Сброс	
8					Пуск
9	Квадратурный счетчик с фазами А и В, фаза А опережает В на 90 градусов при вращении по часовой стрелке, фаза В опережает А на 90 градусов при вращении против часовой стрелки	Такт. генер. Фаза А	Такт. генер. Фаза В		
10				Сброс	
11					Пуск

Таблица 9–7. Режимы работы HSC2 (CPU 224 и CPU 226)

HSC2					
Режим	Описание	I1.2	I1.3	I1.4	I1.5
0	Однофазный реверсивный счетчик с внутренним управлением направлением SM57.3 = 0, счет назад SM57.3 = 1, счет вперед	Тактовый генератор			
1				Сброс	
2					Пуск
3	Однофазный реверсивный счетчик с внешним управлением направлением I1.3 = 0, счет назад I1.3 = 1, счет вперед	Тактовый генератор	Направл.		
4				Сброс	
5					Пуск
6	Двухфазный счетчик с тактовыми входами для счета вперед и назад	Такт. генер. (вперед)	Такт. генер. (назад)		
7				Сброс	
8					Пуск
9	Квадратурный счетчик с фазами А и В, фаза А опережает В на 90 градусов при вращении по часовой стрелке, фаза В опережает А на 90 градусов при вращении против часовой стрелки	Такт. генер. Фаза А	Такт. генер. Фаза В		
10				Сброс	
11					Пуск

Таблица 9–8. Режимы работы HSC3 (CPU 221, CPU 222, CPU 224 и CPU 226)

HSC3					
Режим	Описание	I0.1			
0	Однофазный реверсивный счетчик с внутренним управлением направлением SM137.3 = 0, счет назад SM137.3 = 1, счет вперед	Тактовый генератор			

Таблица 9–9. Режимы работы HSC4 (CPU 221, CPU 222, CPU 224 и CPU 226)

HSC4					
Режим	Описание	I0.3	I0.4	I0.5	
0	Однофазный реверсивный счетчик с внутренним управлением направлением SM147.3 = 0, счет назад SM147.3 = 1, счет вперед	Тактовый генератор			
1				Сброс	
3	Однофазный реверсивный счетчик с внешним управлением направлением I0.4 = 0, счет назад I0.4 = 1, счет вперед	Тактовый генератор	Направл.		
4				Сброс	
6	Двухфазный счетчик с тактовыми входами для счета вперед и назад	Такт. генер. (вперед)	Такт. генер. (назад)		
7				Сброс	
9	Квадратурный счетчик с фазами А и В, фаза А опережает В на 90 градусов при вращении по часовой стрелке, фаза В опережает А на 90 градусов при вращении против часовой стрелки	Такт. генер. Фаза А	Такт. генер. Фаза В		
10				Сброс	

Таблица 9–10. Режимы работы HSC5 (CPU 221, CPU 222, CPU 224 и CPU 226)

HSC5					
Режим	Описание	I0.4			
0	Однофазный реверсивный счетчик с внутренним управлением направлением SM157.3 = 0, счет назад SM157.3 = 1, счет вперед	Тактовый генератор			

Адресация скоростных счетчиков (НС)

Для доступа к счетному значению скоростного счетчика указывается адрес этого счетчика с помощью типа памяти (НС) и номера счетчика (например, НС0). Текущее значение скоростного счетчика может быть только считано и, как показано на рис. 9–17, может быть адресовано только как двойное слово (32 бита).

Формат: **НС[номер скоростного счетчика]**

НС2

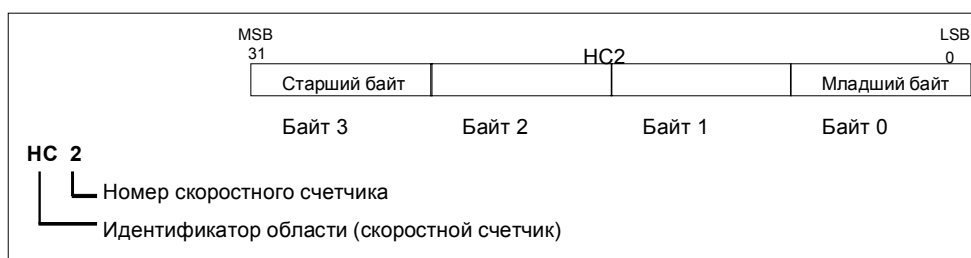


Рис. 9-17. Доступ к текущему значению скоростного счетчика

Описание различных скоростных счетчиков

Все счетчики в одном и том же режиме работают одинаково. Как показано в таблице 9–5, имеется четыре основных режима для счетчиков.

Обратите внимание, что каждый счетчик поддерживает не все режимы.

Каждый счетчик можно использовать: без входов сброса и пуска, со сбросом, но без пуска, или с входами пуска и сброса.

Когда вы активизируете вход сброса, он сбрасывает текущее значение и сохраняет его сброшенным, пока вы не деактивизируете сброс. Когда вы активизируете вход пуска, он разрешает счетчику считать. Если вход пуска деактивизирован, текущее значение счетчика остается постоянным, а тактовые события игнорируются. Если сброс активизируется, когда пуск неактивен, то сброс игнорируется, а текущее значение не изменяется. Если вход пуска становится активным, когда активен вход сброса, текущее значение сбрасывается.

Перед использованием скоростного счетчика вы должны выбрать его режим. Вы можете сделать это с помощью команды HDEF (High-Speed Counter Definition = Определение скоростного счетчика). HDEF устанавливает соответствие между скоростным счетчиком (НСCx) и режимом работы. Для каждого скоростного счетчика можно использовать только одну команду HDEF. Определяйте скоростной счетчик с помощью бита памяти первого цикла SM0.1 (этот бит включен в течение первого цикла обработки программы, а затем выключается), чтобы вызвать подпрограмму, которая содержит команду HDEF.

Выбор активного состояния и однократной или четырехкратной скорости

Четыре счетчика имеют три управляющих бита, которые используются для конфигурирования активного состояния входов сброса и пуска и для выбора односкоростного или четырехскоростного режима счета (только для квадратурных счетчиков). Эти биты находятся в управляющем байте соответствующего счетчика и используются только тогда, когда выполняется команда HDEF. Эти биты определены в таблице 9–11.

Вы должны установить эти управляющие биты в соответствии с желаемым состоянием до исполнения команды HDEF. В противном случае счетчик принимает конфигурацию, определенную по умолчанию для выбранного режима работы счетчика. По умолчанию входы сброса и пуска активны при высоком уровне сигнала, а в квадратурных счетчиках скорость счета установлена четырехкратной (по отношению к частоте входного датчика тактовых импульсов). Если команда HDEF была выполнена, вы не можете изменить настройку счетчика, не переведя сначала CPU в состояние STOP.

Таблица 9–11. Активный уровень для управляющих битов сброса, пуска и выбора 1-кратной или 4-кратной скорости

HSC0	HSC1	HSC2	HSC4	Описание (используются только при исполнении HDEF)
SM37.0	SM47.0	SM57.0	SM147.0	Активный уровень управляющего бита для сброса: 0 = сброс активен при высоком уровне; 1 = сброс активен при низком уровне
--	SM47.1	SM57.1	--	Активный уровень управляющего бита для пуска: 0 = пуск активен при высоком уровне; 1 = пуск активен при низком уровне
SM37.2	SM47.2	SM57.2	SM147.2	Выбор скорости счета для квадратурных счетчиков: 0 = 4-кратная скорость; 1 = 1-кратная скорость

Управляющий байт

Определив счетчик и режим его работы, вы можете программировать динамические параметры счетчика. Каждый скоростной счетчик имеет управляющий байт, который позволяет разблокировать или заблокировать счетчик; управлять направлением (только для режимов 0, 1 и 2) или устанавливать начальное направление счета для всех остальных режимов; загружать текущее значение; загружать предустановленное значение. Проверка управляющего байта и соответствующих текущего и предустановленного значений производится при выполнении команды HSC. В таблице 9–12 описан каждый из этих управляющих битов.

Таблица 9–12. Управляющие биты для HSC0, HSC1 и HSC2

HSC0	HSC1	HSC2	HSC3	HSC4	HSC5	Описание
SM37.3	SM47.3	SM57.3	SM137.3	SM147.3	SM157.3	Бит управления направлением счета: 0 = счет назад; 1 = счет вперед
SM37.4	SM47.4	SM57.4	SM137.4	SM147.4	SM157.4	Записать направление счета в HSC: 0 = не актуализировать; 1 = актуализировать направление
SM37.5	SM47.5	SM57.5	SM137.5	SM147.5	SM157.5	Записать новое предустановленное значение в HSC: 0 = не актуализировать; 1 = актуализировать предустановленное значение
SM37.6	SM47.6	SM57.6	SM137.6	SM147.6	SM157.6	Записать новое текущее значение в HSC: 0 = не актуализировать; 1 = актуализировать текущее значение
SM37.7	SM47.7	SM57.7	SM137.7	SM147.7	SM157.7	Разблокировка HSC: 0 = заблокировать HSC; 1 = разблокировать HSC

Установка текущего и предустановленного значений

Каждый скоростной счетчик имеет 32-битное текущее значение и 32-битное предустановленное значение. Оба значения являются целыми числами со знаком. Чтобы загрузить новое текущее или предустановленное значение, вы должны настроить управляющий байт и байты специальной памяти, содержащие текущее и/или предустановленное значение. Затем вы должны выполнить команду HSC, чтобы новые значения были переданы в скоростной счетчик. Таблица 9–13 описывает байты специальной памяти, используемые для хранения новых текущих и предустановленных значений.

В дополнение к управляющим байтам и байтам, содержащим новые текущие и предустановленные значения, текущее значение каждого скоростного счетчика может быть прочитано путем задания типа данных HC, за которым следует номер (0, 1, 2, 3, 4 или 5) счетчика. Таким образом, текущее значение непосредственно доступно для операций чтения, но оно может быть записано только с помощью описанной выше команды HSC.

Таблица 9–13. Текущее и предустановленное значения HSC0, HSC1, HSC2, HSC3, HSC4 и HSC5

Загружаемое значение	HSC0	HSC1	HSC2	HSC3	HSC4	HSC5
Новое текущее	SMD38	SMD48	SMD58	SMD138	SMD148	SMD158
Новое предустановленное	SMD42	SMD52	SMD62	SMD142	SMD152	SMD162

Байт состояния

Каждому скоростному счетчику поставлен в соответствие байт состояния, предоставляющий в распоряжение биты памяти, указывающие текущее направление счета, а также информацию о том, действительно ли текущее значение больше или равно предустановленному. Таблица 9–14 определяет эти биты состояния для каждого скоростного счетчика.

Таблица 9–14. Биты состояния для HSC0, HSC1, HSC2, HSC3, HSC4 и HSC5

HSC0	HSC1	HSC2	HSC3	HSC4	HSC5	Описание
SM36.0	SM46.0	SM56.0	SM136.0	SM146.0	SM156.0	Не используются
SM36.1	SM46.1	SM56.1	SM136.1	SM146.1	SM156.1	Не используются
SM36.2	SM46.2	SM56.2	SM136.2	SM146.2	SM156.2	Не используются
SM36.3	SM46.3	SM56.3	SM136.3	SM146.3	SM156.3	Не используются
SM36.4	SM46.4	SM56.4	SM136.4	SM146.4	SM156.4	Не используются
SM36.5	SM46.5	SM56.5	SM136.5	SM146.5	SM156.5	Бит состояния текущего направления счета: 0 = счет назад; 1 = счет вперед
SM36.6	SM46.6	SM56.6	SM136.6	SM146.6	SM156.6	Бит состояния, указывающий, равно ли текущее значение предустановленному: 0 = не равно; 1 = равно
SM36.7	SM46.7	SM56.7	SM136.7	SM146.7	SM156.7	Бит состояния, указывающий, больше ли текущее значение, чем предустановленное: 0 = меньше или равно; 1 = больше

Примечание

Биты состояния действительны только во время исполнения программы обработки прерывания скоростного счетчика. Цель контроля состояния скоростного счетчика состоит в том, чтобы разблокировать прерывания для событий, оказывающих воздействие на выполняемую операцию.

Прерывания HSC

Все режимы счетчиков поддерживают прерывание по равенству текущего значения предустановленному. Режимы счетчиков, использующие вход внешнего сброса, поддерживают прерывание по активизации внешнего сброса. Все режимы счетчиков, кроме режимов 0, 1 и 2, поддерживают прерывание по изменению направления счета. Каждое из этих условий возникновения прерываний может быть заблокировано или разблокировано по отдельности. Полностью использование прерываний обсуждается в разделе 9.15.

Примечание

Если вы используете прерывание по внешнему сбросу, не пытайтесь загрузить текущее значение или заблокировать, а затем снова разблокировать скоростной счетчик изнутри программы обработки прерывания, закрепленной за этим событием. Результатом этих действий может быть фатальная ошибка.

Чтобы помочь вам понять работу скоростных счетчиков, в ваше распоряжение предоставляются следующие описания инициализации и последовательности обработки. На всем протяжении этих описаний в качестве примера используется счетчик HSC1. При описании инициализаций предполагается, что S7-200 только что переведен в режим RUN, и поэтому бит памяти первого цикла установлен. Если это не так, помните, что команда HDEF может быть выполнена только один раз для каждого скоростного счетчика после вхождения в режим RUN. Выполнение HDEF для скоростного счетчика во второй раз приводит к ошибке выполнения и не изменяет настройку счетчика по сравнению с тем, как она была выполнена для данного счетчика при первом выполнении HDEF.

Инициализация режимов 0, 1 и 2

Следующие шаги описывают, как инициализировать HSC1 в качестве однофазного реверсивного счетчика с внутренним управлением направлением счета (режим 0, 1 или 2).

1. Используйте бит памяти первого цикла для вызова подпрограммы, в которой будет выполняться операция по инициализации. Когда вы используете вызов подпрограммы, следующие циклы эту подпрограмму не вызывают, что сокращает время цикла и делает программу более структурированной.
2. В подпрограмме инициализации загрузите SMB47 в соответствии с желаемой операцией управления. Например:
 SMB47 = 16#F8 дает следующие результаты:
 Разблокирует счетчик
 Записывает новое текущее значение
 Записывает новое предустановленное значение
 Устанавливает направление счета вперед
 Настраивает входы пуска и сброса на активность при высоком уровне сигнала.
3. Выполните команду HDEF с входом HSC, установленным в 1, и входом MODE [режим], установленным в 0 при отсутствии внешнего сброса и пуска, 1 для внешнего сброса без пуска или 2 для внешнего сброса и пуска.
4. Загрузите SMD48 (двойное слово) желаемым текущим значением (загрузите 0, чтобы его очистить).
5. Загрузите SMD52 (двойное слово) желаемым предустановленным значением.
6. Чтобы распознавать равенство текущего и предустановленного значений, запрограммируйте прерывание, поставив в соответствие программе обработки прерывания прерывающее событие CV = PV (событие 13). Подробную информацию об обработке прерываний вы найдете в разделе 9.15.
7. Чтобы распознавать внешний сброс, запрограммируйте прерывание, поставив в соответствие программе обработки прерывания прерывающее событие «внешний сброс» (external reset) (событие 15).
8. Для разблокировки прерываний выполните команду разрешения глобальных прерываний (ENI).
9. Выполните команду HSC, чтобы S7-200 запрограммировал HSC1.
10. Выйдите из подпрограммы.

Инициализация режимов 3, 4 и 5

Следующие шаги описывают, как инициализировать HSC1 в качестве однофазного реверсивного счетчика с внешним управлением направлением счета (режим 3, 4 или 5):

1. Используйте бит памяти первого цикла для вызова подпрограммы, в которой будет выполняться операция по инициализации. Когда вы используете вызов подпрограммы, следующие циклы эту подпрограмму не вызывают, что сокращает время цикла и делает программу более структурированной.
2. В подпрограмме инициализации загрузите SMB47 в соответствии с желаемой операцией управления. Например:
SMB47 = 16#F8 дает следующие результаты:
Разблокирует счетчик
Записывает новое текущее значение
Записывает новое предустановленное значение
Устанавливает начальное направление счета HSC вперед
Настраивает входы пуска и сброса на активность при высоком уровне сигнала
3. Выполните команду HDEF с входом HSC, установленным в 1, и входом MODE [режим], установленным на 3 при отсутствии внешнего сброса и пуска, 4 для внешнего сброса без пуска или 5 для внешнего сброса и пуска.
4. Загрузите SMD48 (двойное слово) желаемым текущим значением (загрузите 0, чтобы его очистить).
5. Загрузите SMD52 (двойное слово) желаемым предустановленным значением.
6. Чтобы распознать равенство текущего и предустановленного значений, запрограммируйте прерывание, поставив в соответствие программе обработки прерывания прерывающее событие CV = PV (событие 13). Подробную информацию об обработке прерываний вы найдете в разделе 9.15.
7. Чтобы распознавать изменения направления счета, запрограммируйте прерывание, поставив в соответствие программе обработки прерывания прерывающее событие «изменение направления» (direction changed) (событие 14).
8. Чтобы распознавать внешний сброс, запрограммируйте прерывание, поставив в соответствие программе обработки прерывания прерывающее событие «внешний сброс» (external reset) (событие 15).
9. Для разблокировки прерываний выполните команду разрешения глобальных прерываний (ENI).
10. Выполните команду HSC, чтобы S7-200 запрограммировал HSC1.
11. Выйдите из подпрограммы.

Инициализация режимов 6, 7 и 8

Следующие шаги описывают, как инициализировать HSC1 в качестве двухфазного реверсивного счетчика с датчиками тактовых импульсов вперед и назад (режим 6, 7 или 8):

1. Используйте бит памяти первого цикла для вызова подпрограммы, в которой будет выполняться операция по инициализации. Когда вы используете вызов подпрограммы, следующие циклы эту подпрограмму не вызывают, что сокращает время цикла и делает программу более структурированной.
2. В подпрограмме инициализации загрузите SMB47 в соответствии с желаемой операцией управления. Например:
 SMB47 = 16#F8 дает следующие результаты:
 Разблокирует счетчик
 Записывает новое текущее значение
 Записывает новое предустановленное значение
 Устанавливает начальное направление счета HSC вперед
 Настраивает входы пуска и сброса на активность при высоком уровне сигнала
3. Выполните команду HDEF с входом HSC, установленным в 1, и входом MODE [режим], установленным на 6 при отсутствии внешнего сброса и пуска, 7 для внешнего сброса без пуска или 8 для внешнего сброса и пуска.
4. Загрузите SMD48 (двойное слово) желаемым текущим значением (загрузите 0, чтобы его очистить).
5. Загрузите SMD52 (двойное слово) желаемым предустановленным значением.
6. Чтобы распознать равенство текущего и предустановленного значений, запрограммируйте прерывание, поставив в соответствие программе обработки прерывания прерывающее событие CV = PV (событие 13). Подробную информацию об обработке прерываний вы найдете в разделе 9.15.
7. Чтобы распознавать изменения направления счета, запрограммируйте прерывание, поставив в соответствие программе обработки прерывания прерывающее событие «изменение направления» (direction changed) (событие 14).
8. Чтобы распознавать внешний сброс, запрограммируйте прерывание, поставив в соответствие программе обработки прерывания прерывающее событие «внешний сброс» (external reset) (событие 15).
9. Для разблокировки прерываний выполните команду разрешения глобальных прерываний (ENI).
10. Выполните команду HSC, чтобы S7-200 запрограммировал HSC1.
11. Выйдите из подпрограммы.

Инициализация режимов 9, 10 и 11

Следующие шаги описывают, как инициализировать HSC1 в качестве квадратурного счетчика с фазами А и В (режим 9, 10 или 11):

1. Используйте бит памяти первого цикла для вызова подпрограммы, в которой будет выполняться операция по инициализации. Когда вы используете вызов подпрограммы, следующие циклы эту подпрограмму не вызывают, что сокращает время цикла и делает программу более структурированной.
2. В подпрограмме инициализации загрузите SMB47 в соответствии с желаемой операцией управления.
 Например (однократная скорость счета):
 SMB47 = 16#FC дает следующие результаты:
 Разблокирует счетчик
 Записывает новое текущее значение
 Записывает новое предустановленное значение
 Устанавливает начальное направление счета HSC вперед
 Настраивает входы пуска и сброса на активность высокого уровня сигнала
 при
 Например (четырёхкратная скорость счета):
 SMB47 = 16#F8 дает следующие результаты:
 Разблокирует счетчик
 Записывает новое текущее значение
 Записывает новое предустановленное значение
 Устанавливает начальное направление счета HSC вперед
 Настраивает входы пуска и сброса на активность высокого уровня сигнала
 при
3. Выполните команду HDEF с входом HSC, установленным в 1, и входом MODE [режим], установленным на 9 при отсутствии внешнего сброса и пуска, 10 для внешнего сброса без пуска или 11 для внешнего сброса и пуска.
4. Загрузите SMD48 (двойное слово) желаемым текущим значением (загрузите 0, чтобы его очистить).
5. Загрузите SMD52 (двойное слово) желаемым предустановленным значением.
6. Чтобы распознать равенство текущего и предустановленного значений, запрограммируйте прерывание, поставив в соответствии программе обработки прерывания прерывающее событие CV = PV (событие 13). Подробную информацию об обработке прерываний вы найдете в разделе 9.15.
7. Чтобы распознавать изменения направления счета, запрограммируйте прерывание, поставив в соответствии программе обработки прерывания прерывающее событие «изменение направления» (direction changed) (событие 14).
8. Чтобы распознавать внешний сброс, запрограммируйте прерывание, поставив в соответствии программе обработки прерывания прерывающее событие «внешний сброс» (external reset) (событие 15).
9. Для разблокировки прерываний выполните команду разрешения глобальных прерываний (ENI).
10. Выполните команду HSC, чтобы S7-200 запрограммировал HSC1.
11. Выйдите из подпрограммы.

Изменение направления в режиме 0, 1 или 2

Следующие шаги описывают, как сконфигурировать изменение направления для HSC1 как однофазного счетчика с внутренним управлением направлением счета (режим 0, 1 или 2):

1. Загрузите SMB47, чтобы записать желаемое направление:
 - SMB47 = 16#90 Разблокирует счетчик
Устанавливает направление счета HSC назад
 - SMB47 = 16#98 Разблокирует счетчик
Устанавливает направление счета HSC вперед
2. Выполните команду HSC, чтобы S7-200 запрограммировал HSC1.

Загрузка нового текущего значения (любой режим)

Изменение текущего значения приводит к принудительной блокировке счетчика на время выполнения изменения. Пока счетчик заблокирован, он не считает и не генерирует прерываний.

Следующие шаги описывают, как изменить текущее значение счетчика HSC1 (любой режим):

1. Загрузите SMB47, чтобы записать желаемое текущее значение:
 - SMB47 = 16#C0 Разблокирует счетчик
Записывает новое текущее значение
2. Загрузите SMD48 (двойное слово) желаемым текущим значением (загрузите 0, чтобы его очистить).
3. Выполните команду HSC, чтобы S7-200 запрограммировал HSC1.

Загрузка нового предустановленного значения (любой режим)

Следующие шаги описывают, как изменить предустановленное значение HSC1 (любой режим):

1. Загрузите SMB47, чтобы записать желаемое предустановленное значение:
 - SMB47 = 16#A0 Разблокирует счетчик
Записывает новое предустановленное значение
2. Загрузите SMD52 (двойное слово) желаемым предустановленным значением.
3. Выполните команду HSC, чтобы S7-200 запрограммировал HSC1.

Блокировка скоростного счетчика (любой режим)

Следующие шаги описывают, как заблокировать скоростной счетчик HSC1 (любой режим):

1. Загрузите SMB47, чтобы заблокировать счетчик:
 - SMB47 = 16#00 Блокирует счетчик
2. Выполните команду HSC, чтобы заблокировать счетчик.

Хотя вышеприведенные последовательности показывают, как изменить направление, текущее и предустановленное значение по отдельности, вы можете изменить все эти настройки или любую их комбинацию в той же последовательности, устанавливая надлежащим образом SMB47, а затем выполняя команду HSC.

Пример скоростного счетчика

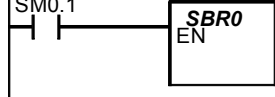
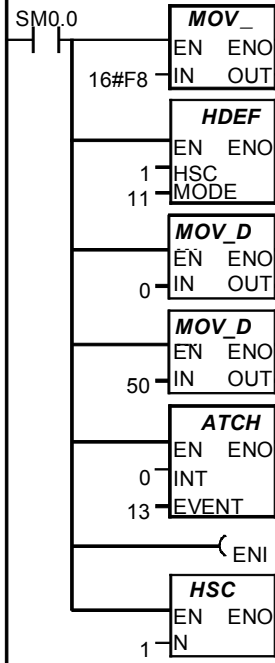
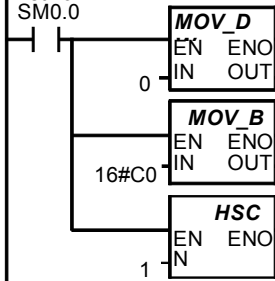
LAD		STL
MAIN OB1		
Network [Сегмент] 1 SM0.1		В первом цикле вызвать подпрограмму 0. Конец главной программы.
SUBROUTINE 0 [Подпрограмма 0]		
Network 1 SM0.0		Разблокировать счетчик. Записать новое текущее значение. Записать новое предустановленное значение. Записать начальное направление счета вперед. Сделать входы пуска и сброса активными при высоком уровне сигнала. Установить 4-кратную скорость. HSC1 сконфигурирован для квадратурного режима с входами сброса и пуска. Сбросить текущее значение HSC1. Ввести предустановленное значение HSC1, равное 50. Программе прерывания 0 поставлено в соответствие событие 13 (текущее значение = предустановленному) Разрешены глобальные прерывания Запрограммировать HSC1.
		Network 1 LD SM0.0 MOVB 16#F8, SMB47 HDEF 1, 11 MOVD 0, SMD48 MOVD 50, SMD52 ATCH 0, 13 ENI HSC 1
INTERRUPT 0 [Прерывание 0]		
Network 1 SM0.0		Сбросить текущее значение HSC1. Записать новое текущее значение и разблокировать счетчик. Запрограммировать HSC1.
		Network 1 LD SM 0.0 MOVD 0, SMD48 MOVB 16#C0, SMB47 HSC 1

Рис. 9-18. Пример инициализации HSC1 (SIMATIC LAD и STL)

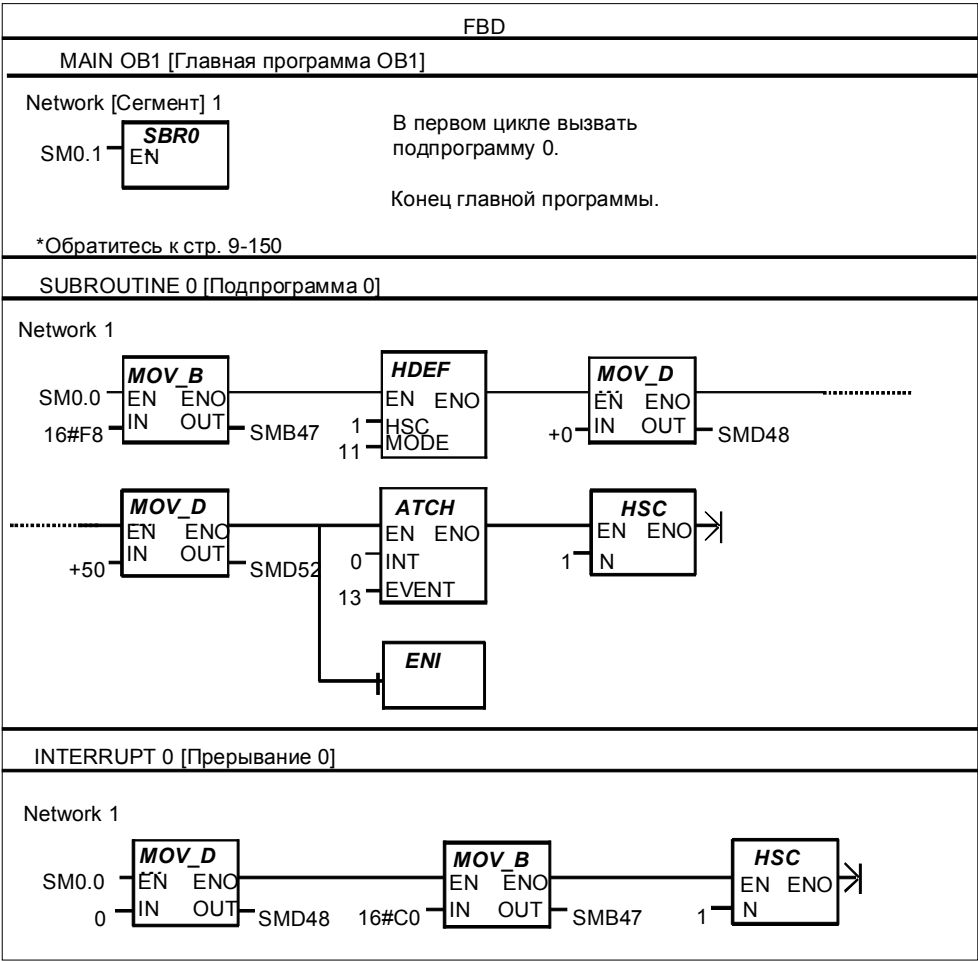
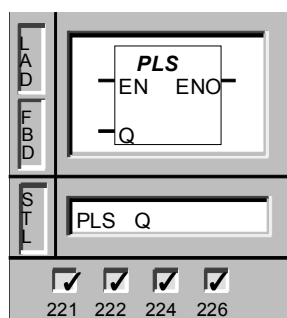


Рис. 9-19. Пример инициализации HSC1 (SIMATIC FBD)

9.6 Команды SIMATIC для импульсного вывода

Импульсный вывод



Команда **Импульсный вывод** проверяет биты специальной памяти для импульсного выхода (Q0.0 или Q0.1). Затем вызывается импульсная операция, определенная битами специальной памяти.

Операнды: Q: константа (0 или 1)
 Типы данных: WORD
 Импульсные выходы Q0.0 и Q0.1

Описание команд со скоростными выходами для S7-200

Каждый CPU имеет два генератора: PTO – для вывода последовательностей импульсов и PWM – для управления с помощью широтно-импульсной модуляции. Один генератор поставлен в соответствие цифровому выходу Q0.0, другой генератор – цифровому выходу Q0.1.

Генераторы PTO и PWM и регистр образа процесса совместно используют Q0.0 и Q0.1. Когда функция PTO или PWM активна на Q0.0 или Q0.1, то выходом управляет генератор PTO или PWM, а нормальное использование выхода заблокировано. На форму выходного сигнала не влияет ни состояние регистра образа процесса, ни принудительное присваивание значений выходам, ни выполнение команд непосредственного вывода. Когда генератор PTO/PWM не активен, управление выходом возвращается регистру образа процесса. Регистр образа процесса определяет начальное и конечное состояние импульсного выходного сигнала, вызывая его начало и завершение на высоком или низком уровне.

Примечание

Рекомендуется устанавливать регистр образа процесса для Q0.0 и Q0.1 на нулевое значение перед разблокировкой операции PTO или PWM.

Функция «Последовательность импульсов» (PTO) предоставляет в распоряжение выходной сигнал в виде прямоугольных импульсов (с относительной длительностью импульсов 50%), период следования которых и количество определяется пользователем. Функция «Широтно-импульсная модуляция» (PWM) предоставляет в распоряжение выходной импульсный сигнал с постоянным периодом следования и переменной относительной длительностью импульсов, причем период следования импульсов и их ширину определяет пользователь.

Каждому генератору PTO/PWM поставлены в соответствие управляющий байт (8 битов), период следования и ширина импульса (16-битовое значение без знака) и количество импульсов (32-битовое значение без знака). Все эти величины хранятся в определенных ячейках области специальной памяти (SM). Когда эти ячейки специальной битовой памяти настроены для выбора желаемой операции, операция вызывается выполнением команды «Импульсный вывод» (PLS). Эта команда заставляет S7-200 прочитать ячейки SM и соответствующим образом запрограммировать генератор PTO или PWM.

Вы можете изменять характеристики сигнала PTO или PWM, изменяя желаемые ячейки области SM (включая управляющий байт), а затем выполняя команду PLS.

Вы можете в любое время заблокировать импульсный сигнал PTO или PWM, записав ноль в бит разблокировки PTO или PWM управляющего байта (SM67.7 или SM77.7), а затем выполнив команду PLS.

Примечание

По умолчанию значения всех управляющих битов, периода следования, ширины и количества импульсов равны нулю.

Примечание

Выходы PTO/PWM должны иметь минимальную нагрузку не менее 10% от номинальной нагрузки, чтобы обеспечить четкий переход от выключенного состояния к включенному и наоборот.

Функционирование PWM

Функция PWM обеспечивает выход с переменной относительной длительностью импульсов. Период следования и ширина импульсов могут быть заданы в микросекундной или миллисекундной базе времени. Период следования импульсов имеет диапазон от 50 микросекунд до 65 535 микросекунд или от 2 миллисекунд до 65 535 миллисекунд. Ширина импульса имеет диапазон от 0 микросекунд до 65 535 микросекунд или от 0 миллисекунд до 65 535 миллисекунд. Когда ширина импульса задается большей или равной периоду следования импульсов, то относительная длительность импульсов равна 100%, и выход включен постоянно. Когда ширина импульсов задана равной 0, то относительная длительность импульсов равна 0%, и выход выключен. Если период следования импульсов задан меньшим, чем две единицы времени, то он устанавливается равным двум единицам времени.

Имеется два способа изменения характеристик импульсов PWM: с синхронным обновлением и с асинхронным обновлением.

- Синхронное обновление: Если не требуется изменение базы времени, то может быть выполнено синхронное обновление. При синхронном обновлении изменение характеристик импульсов происходит на границе периода следования, обеспечивая плавный переход.
- Асинхронное обновление: Обычно при работе PWM меняется ширина импульсов при постоянном периоде следования. Поэтому изменение базы времени не требуется. Однако, если требуется изменение базы времени генератора PTO или PWM. То используется асинхронное обновление. Асинхронное обновление вызывает на мгновение блокирование генератора PTO или PWM асинхронно со следованием импульсов PWM. Это может вызвать нежелательную неустойчивость работы управляемого устройства. Поэтому рекомендуется синхронное обновление PWM. Выбирайте базу времени, которая будет работать со всеми ожидаемыми вами значениями периода следования импульсов.

Для задания типа обновления используется бит метода обновления PWM (SM67.4 или SM77.4) в управляющем байте. Чтобы произвести изменения, выполните команду PLS. Имейте в виду, что если изменяется база времени, то произойдет асинхронное обновление независимо от состояния бита метода обновления PWM.

Функционирование РТО

Функция РТО обеспечивает генерирование последовательности импульсов прямоугольной формы (с относительной длительностью 50%) с заданным количеством импульсов. Период следования импульсов может быть задан в микросекундах или миллисекундах. Период следования импульсов имеет диапазон от 50 микросекунд до 65 535 микросекунд или от 2 миллисекунд до 65 535 миллисекунд. Если заданное время цикла является нечетным числом, то в результате будет несколько искажена относительная длительность импульсов. Количество выводимых импульсов находится в диапазоне от 1 до 4 294 967 295.

Если период следования импульсов задан меньшим двух единиц времени, то он устанавливается равным двум единицам времени. Если количество импульсов задано равным нулю, то оно устанавливается равным единице.

Бит незаятости РТО в байте состояния (SM66.7 или SM76.7) предназначен для индикации завершения запрограммированной последовательности импульсов. Кроме того, после завершения последовательности импульсов может быть вызвана программа обработки прерывания (за информацией о командах прерывания и связи см. раздел 9.15). Если вы используете многосегментную обработку, то программа обработки прерывания будет вызвана после завершения таблицы профиля последовательности импульсов. См. ниже многосегментную конвейерную обработку.

Функция РТО разрешает сцепление или конвейерную обработку последовательностей импульсов. Когда активная последовательность импульсов завершена, немедленно начинается вывод новой последовательности импульсов. Это обеспечивает непрерывность следующих друг за другом последовательностей импульсов.

Конвейерная обработка может быть выполнена одним из двух способов: в виде односегментной конвейерной обработки или в виде многосегментной конвейерной обработки.

Односегментная конвейерная обработка. При односегментной конвейерной обработке вы несете ответственность за обновление ячеек области SM для следующей последовательности импульсов. Как только был запущен первый сегмент PTO, вы должны немедленно изменить ячейки SM в соответствии с требованиями второй последовательности и снова выполнить команду PLS. Атрибуты второй последовательности импульсов будут храниться в конвейере до завершения первой последовательности импульсов. В конвейере в каждый момент времени может храниться только одна запись. Как только завершится первая последовательность импульсов, начнется вывод второй последовательности, и конвейер становится доступным для задания характеристик новой последовательности импульсов. Вы можете затем повторить этот процесс, чтобы установить характеристики следующей последовательности импульсов.

Между последовательностями импульсов происходит плавный переход, кроме следующих ситуаций:

- Если меняется база времени
- Если активная последовательность импульсов завершается раньше, чем настройка новой последовательности импульсов распознается при исполнении команды PLS.

Если вы пытаетесь загрузить конвейер, когда он полон, то в регистре состояния устанавливается бит переполнения PTO (SM66.6 или SM76.6). Этот бит инициализируется нулем при переходе в режим RUN. Если вы хотите обнаруживать следующие переполнения, то после обнаружения переполнения вы должны сбрасывать этот бит вручную.

Многосегментная конвейерная обработка. При многосегментной конвейерной обработке CPU автоматически считывает характеристики каждого сегмента последовательности импульсов из таблицы профиля, расположенной в V-памяти. Единственными ячейками области SM, используемыми в этом режиме, являются управляющий байт и байт состояния. Для выбора многосегментного режима работы должно быть загружено начальное смещение таблицы профиля (SMW168 или SMW178). В качестве базы времени могут быть заданы микросекунды или миллисекунды, но этот выбор применяется ко всем значениям периода следования импульсов в таблице профиля и не может быть изменен, когда таблица используется. Многосегментный режим работы затем может быть запущен исполнением команды PLS.

Запись для каждого сегмента имеет длину 8 байтов 16-битового значения периода следования импульсов, 16-битового значения приращения периода следования импульсов и 32-битового значения количества импульсов.

Формат таблицы профиля показан в табл. 9–15. Дополнительной характеристикой, доступной в многосегментном режиме работы РТО, является возможность автоматически увеличивать или уменьшать период следования импульсов на заданную для каждого импульса величину. Программирование положительного значения в поле приращения периода следования импульсов увеличивает этот период. Программирование отрицательного значения в поле приращения периода следования импульсов уменьшает этот период. Нулевое значение не меняет период следования импульсов.

Если задать приращение периода следования импульсов таким, что через некоторое количество импульсов величина периода следования импульсов становится недопустимой, то возникает математическое переполнение. Функция РТО завершается, а выход возвращается под управление регистра образа процесса. Кроме того, в байте состояния устанавливается в единицу бит ошибки расчета приращения периода следования импульсов (SM66.4 или SM76.4).

Если вы вручную прерываете использование выполняемого в данный момент профиля РТО, то в байте состояния пользователем будет установлен в единицу бит прерывания (SM66.5 или SM76.5).

Когда профиль РТО работает, количество активных в данный момент сегментов доступно в SMB166 (или SMB176).

Таблица 9–15. Формат таблицы профиля для многосегментного режима РТО

Смещение в байтах от начала таблицы	Номер сегмента	Описание записей таблицы
0		Количество сегментов (от 1 до 255); значение 0 генерирует нефатальную ошибку, выход РТО не генерируется.
1	#1	Начальный период следования импульсов (от 2 до 65535 единиц базы времени)
3		Приращение периода следования импульсов (величина со знаком) (от –32768 до 32767 единиц базы времени)
5		Количество импульсов (от 1 до 4294967295)
9	#2	Начальный период следования импульсов (от 2 до 65535 единиц базы времени)
11		Приращение периода следования импульсов (величина со знаком) (от –32768 до 32767 единиц базы времени)
13		Количество импульсов (от 1 до 4294967295)
:	:	:
:	:	:

Расчет значений таблицы профиля

Возможность многосегментного режима конвейерной обработки генераторов РТО/PWM может быть полезной во многих приложениях, в частности, при управлении шаговыми двигателями.

Пример, показанный на рис. 9–20, иллюстрирует, как нужно определять значения таблицы профиля, необходимые для генерирования последовательности выходных импульсов, которая разгоняет шаговый двигатель, обеспечивает его работу с постоянной скоростью, а затем тормозит двигатель.

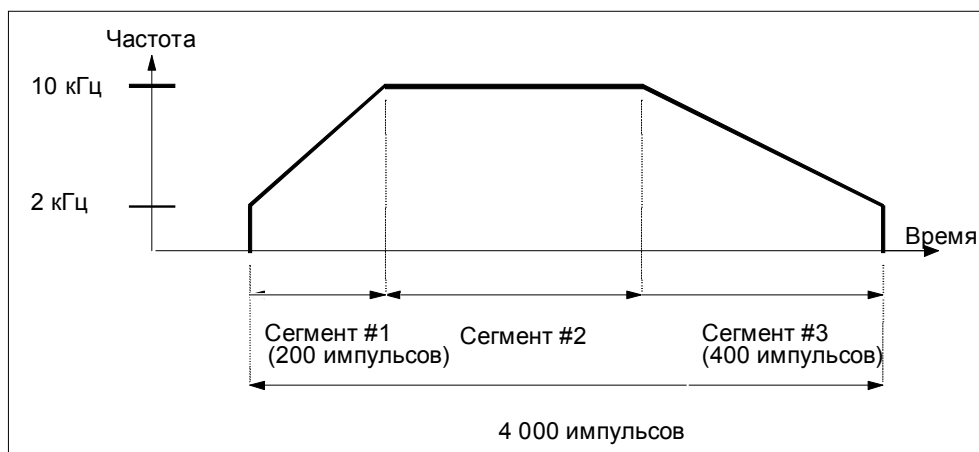


Рис. 9–20. Пример диаграммы частота – время применительно к простому шаговому двигателю

В этом примере предполагается, что для достижения желаемого количества оборотов двигателя требуется 4 000 импульсов. Начальная и конечная частота следования импульсов равна 2 кГц, а максимальная частота следования импульсов равна 10 кГц. Так как величины в таблице профиля выражаются в периодах следования импульсов, а не в частоте, преобразуйте заданные значения частоты в значения периодов следования импульсов. Тогда начальный и конечный период следования импульсов составит 500 мкс, а период следования импульсов, соответствующий максимальной частоте, составит 100 мкс.

На интервале ускорения желательно, чтобы желаемая максимальная частота следования импульсов была достигнута примерно через 200 импульсов. Предполагается также, что интервал замедления должен быть реализован примерно за 400 импульсов.

В примере на рис. 9–20 для определения приращения периода следования импульсов для данного сегмента может быть использована простая формула (показана ниже), которую генератор РТО/PWM использует для настройки периода каждого импульса:

Приращение периода следования импульсов для данного сегмента = $\frac{|\text{ЕСТ} - \text{ИСТ}|}{Q}$,
 где ЕСТ = конечный период следования импульсов для данного сегмента
 ИСТ = начальный период следования импульсов для данного сегмента
 Q = количество импульсов в данном сегменте

По этой формуле приращение периода следования импульсов для этапа ускорения (или сегмента #1) составляет –2. Аналогично, приращение периода следования импульсов для этапа замедления (или сегмента #3) равно 1. Так как для сегмента #2 скорость постоянна, то приращение периода следования импульсов для этого сегмента равно нулю.

Предполагая, что таблица профиля расположена в V-памяти, начиная с V500, значения таблицы, используемые для генерирования желаемой последовательности импульсов, показаны в табл. 9–16.

Таблица 9–16. Значения в таблице профиля

Адрес V-памяти	Значение
VB500	3 (общее количество сегментов)
VW501	500 (начальный период следования импульсов – сегмент #1)
VW503	-2 (приращение периода следования импульсов – сегмент #1)
VD505	200 (количество импульсов – сегмент #1)
VW509	100 (начальный период следования импульсов – сегмент #2)
VW511	0 (приращение периода следования импульсов – сегмент #2)
VD513	3400 (количество импульсов – сегмент #2)
VW517	100 (начальный период следования импульсов – сегмент #3)
VW519	1 (приращение периода следования импульсов – сегмент #3)
VD521	400 (количество импульсов – сегмент #3)

Значения из этой таблицы могут быть помещены в V-память с помощью команд в вашей программе. Альтернативным методом является определение значений профиля в блоке данных. Пример программы с командами для использования многосегментного режима РТО показан на рис. 9–23.

Период для последнего импульса сегмента прямо в профиле не указывается, но вместо этого он должен быть рассчитан (конечно, за исключением случая, когда приращение периода следования импульсов равно нулю). Знание периода для последнего импульса сегмента полезно для определения приемлемости перехода сегментами последовательности импульсов. Ниже приведена формула расчета периода для последнего импульса сегмента:

Период для последнего импульса сегмента = $ICT + (DEL * (Q - 1))$, где ICT = начальный период следования импульсов для данного сегмента DEL = приращение периода следования импульсов для данного сегмента Q = количество импульсов в данном сегменте

В то время как приведенный выше упрощенный пример полезен в качестве введения, реальные приложения могут потребовать более сложных профилей последовательностей импульсов:

- Приращение периода следования импульсов может быть задано только как целое количество микросекунд или миллисекунд
- Изменение периода следования выполняется на каждом импульсе.

Влияние этих двух пунктов состоит в том, что расчет приращения периода следования импульсов для данного сегмента может потребовать итеративного подхода. Может потребоваться некоторая гибкость в значении конечного периода следования импульсов или количества импульсов для данного сегмента.

В процессе определения правильных значений таблицы профиля может быть полезна длительность данного сегмента профиля. Время, необходимое для завершения данного сегмента профиля, может быть рассчитано с помощью следующей формулы:

Длительность сегмента = $Q * (ICT + ((DEL/2) * (Q-1)))$,	
где Q	= количество импульсов в данном сегменте
ICT	= начальный период следования импульсов для данного сегмента
DEL	= приращение периода следования импульсов для данного сегмента

Управляющие регистры PTO/PWM

Таблица 9–17 описывает регистры, используемые для управления режимом PTO/PWM. Таблицу 9–18 вы можете использовать в качестве быстрой справки, чтобы определить значение, которое следует поместить в управляющий регистр PTO/PWM, чтобы вызвать желаемую операцию. Используйте SMB67 для PTO/PWM 0 и SMB77 для PTO/PWM 1. Если вы собираетесь загрузить новое количество импульсов (SMD72 или SMD82), ширину импульсов (SMW70 или SMW80) или период следования импульсов (SMW68 или SMW78), вы должны загрузить эти значения, а также управляющий регистр перед выполнением команды PLS. Если вы используете многосегментную последовательность импульсов, то перед выполнением команды PLS вы также должны загрузить начальное смещение (SMW168 или SMW178) таблицы профиля и значения таблицы профиля.

Таблица 9–17. Управляющие регистры PTO /PWM

Q0.0	Q0.1	Биты состояния
SM66.4	SM76.4	Профиль PTO завершен из-за ошибки расчета приращения 0 = нет ошибки; 1 = завершен
SM66.5	SM76.5	Профиль PTO завершен по команде пользователя 0 = нет завершения; 1 = завершен
SM66.6	SM76.6	Переполнение/потеря значимости в конвейере PTO 0 = нет переполнения; 1 = переполнение/потеря значимости
SM66.7	SM76.7	PTO не действует 0 = действует; 1 = PTO не действует
Q0.0	Q0.1	Управляющие биты
SM67.0	SM77.0	Обновить значение периода следования импульсов PTO/PWM 0 = не обновлять; 1 = обновить период следования импульсов
SM67.1	SM77.1	Обновить значение ширины импульсов PWM 0 = не обновлять; 1 = обновить ширину импульсов
SM67.2	SM77.2	Обновить количество импульсов PTO 0 = не обновлять; 1 = обновить количество импульсов
SM67.3	SM77.3	Выбрать базу времени PTO/PWM 0 = 1 мкс/такт; 1 = 1мс/такт
SM67.4	SM77.4	Метод обновления PWM: 0 = асинхронное обновление, 1 = синхронное обновление
SM67.5	SM77.5	Режим PTO: 0 = односегментный 1 = многосегментный
SM67.6	SM77.6	Выбрать PTO/PWM 0 = выбирает PTO; 1 = выбирает PWM
SM67.7	SM77.7	Разблокировать PTO/PWM 0 = блокирует PTO/PWM; 1 = разблокирует PTO/PWM
Q0.0	Q0.1	Другие регистры PTO/PWM
SMW68	SMW78	Значение периода следования импульсов PTO/PWM (диапазон: от 2 до 65535)
SMW70	SMW80	Значение ширины импульса PWM (диапазон: от 0 до 65535)
SMD72	SMD82	Количество импульсов PTO (диапазон: от 1 до 4294967295)
SMB166	SMB176	Номер действующего сегмента (используется только в многосегментном режиме PTO)
SMW168	SMW178	Начальный адрес таблицы профиля, выраженный как байтовое смещение от V0 (используется только в многосегментном режиме PTO)

Таблица 9–18. Справочные данные об управляющем байте PTO/PWM

Управля- ющий регистр (16-ричное значение)	Результат выполнения операции PLS							
	Раз- реш.	Режим	Режим сегментиро- вания PTO	Метод обновления PWM	База времени	Количество импульсов	Ширина импульса	Период следов. импуль- сов
16#81	Да	PTO	Односегмент.		1 мкс/такт			Загруж.
16#84	Да	PTO	Односегмент.		1 мкс/такт	Загружено		
16#85	Да	PTO	Односегмент.		1 мкс/такт	Загружено		Загруж.
16#89	Да	PTO	Односегмент.		1 мс/такт			Загруж.
16#8C	Да	PTO	Односегмент.		1 мс/такт	Загружено		
16#8D	Да	PTO	Односегмент.		1 мс/такт	Загружено		Загруж.
16#A0	Да	PTO	Многосегм.		1 мкс/такт			
16#A8	Да	PTO	Многосегм.		1 мс/такт			
16#D1	Да	PWM		Синхронный	1 мкс/такт			Загруж.
16#D2	Да	PWM		Синхронный	1 мкс/такт		Загруж.	
16#D3	Да	PWM		Синхронный	1 мкс/такт		Загруж.	Загруж.
16#D9	Да	PWM		Синхронный	1 мс/такт			Загруж.
16#DA	Да	PWM		Синхронный	1 мс/такт		Загруж.	
16#DB	Да	PWM		Синхронный	1 мс/такт		Загруж.	Загруж.

Инициализация и последовательность функционирования PTO/PWM

Далее следуют описания инициализации и последовательности функционирования. Они могут помочь вам лучше понять работу функций PTO и PWM. На всем протяжении этих описаний используется импульсный выход Q0.0. При описании инициализации подразумевается, что S7–200 переведен в режим RUN и поэтому бит памяти первого цикла установлен. Если это не так или если функция PTO/PWM должна быть повторно инициализирована, вы можете вызвать программу инициализации без использования бита памяти первого цикла.

Инициализация PWM

Для инициализации PWM для Q0.0 выполните следующие шаги:

1. С помощью бита памяти первого цикла (SM0.1) установите выход в 0 и вызовите подпрограмму, которая вам необходима для выполнения операций по инициализации. Когда вы используете вызов подпрограммы, следующие циклы эту подпрограмму не вызывают, что сокращает время цикла и делает программу более структурированной.
2. В подпрограмме инициализации загрузите SMB67 значением 16#D3 для PWM, использующего микросекундные приращения (или 16#DB для PWM, использующего миллисекундные приращения). Эти значения устанавливают управляющий байт на разрешение функции РТО/PWM, выбирают режим PWM, выбирают микросекундные или миллисекундные приращения и устанавливают обновление ширины импульса и периода следования импульсов.
3. Загрузите SMW68 (слово) желаемым периодом следования импульсов.
4. Загрузите SMW70 (слово) желаемой шириной импульса.
5. Выполните команду PLS, чтобы S7-200 запрограммировал генератор РТО/PWM.
6. Загрузите SMB67 значением 16#D2 для микросекундных приращений (или 16#DA для миллисекундных приращений). Это предварительно загружает новое значение управляющего байта для последующих изменений ширины импульсов.
7. Выйдите из подпрограммы.

Изменение ширины импульсов для выходов PWM

Для изменения ширины импульсов для выходов PWM в подпрограмме выполните следующие шаги. (Предполагается, что SMB67 был предварительно загружен значением 16#D2 или 16#DA.)

1. Вызовите подпрограмму для загрузки SMW70 (слово) желаемой шириной импульса.
2. Выполните команду PLS, чтобы S7-200 запрограммировал генератор РТО/PWM.
3. Выйдите из подпрограммы.

Инициализация РТО – односегментный режим

Для инициализации РТО выполните следующие шаги:

1. С помощью бита памяти первого цикла (SM0.1) установите выход в 0 и вызовите подпрограмму, которая вам необходима для выполнения операций по инициализации. Это сокращает время цикла и делает программу более структурированной.
2. В подпрограмме инициализации загрузите SMB67 значением 16#85 для РТО, использующего микросекундные приращения (или 16#8D для РТО, использующего миллисекундные приращения). Эти значения устанавливают управляющий байт на разрешение функции РТО/PWM, выбирают режим РТО, выбирают микросекундные или миллисекундные приращения и устанавливают обновление количества импульсов и периода следования импульсов.
3. Загрузите SMW68 (слово) желаемым периодом следования импульсов.
4. Загрузите SMD72 (двойное слово) желаемым количеством импульсов.
5. Этот шаг необязателен. Если вы хотите выполнить соответствующую функцию, когда вывод импульсной последовательности будет завершен, вы можете запрограммировать прерывание, поставив в соответствие программе обработки прерывания событие «Импульсная последовательность завершена» (pulse train complete) (категория прерывания 19) с помощью команды ATCH и выполнив команду ENI, разрешающую глобальные прерывания. Полное описание обработки прерываний вы найдете в разделе 9.15.
6. Выполните команду PLS, чтобы S7-200 запрограммировал генератор РТО/PWM.
7. Выйдите из подпрограммы.

Изменение периода следования импульсов РТО – односегментный режим

Для изменения периода следования импульсов РТО в программе обработки прерывания или подпрограмме при использовании односегментного режима РТО выполните следующие шаги:

1. Загрузите SMB67 значением 16#81 для РТО, использующего микросекундные приращения (или 16#89 для РТО, использующего миллисекундные приращения). Эти значения устанавливают управляющий байт на разрешение функции РТО/PWM, выбирают режим РТО, выбирают микросекундные или миллисекундные приращения и устанавливают обновление периода следования импульсов.
2. Загрузите SMW68 (слово) желаемым периодом следования импульсов.
3. Выполните команду PLS, чтобы S7-200 запрограммировал генератор РТО/PWM. CPU должен завершить любую выводимую в данный момент РТО, прежде чем начнется вывод последовательности импульсов РТО с обновленным периодом следования импульсов.

4. Выйдите из программы обработки прерывания или подпрограммы.

Изменение количества импульсов РТО – односегментный режим

Для изменения количества импульсов РТО в программе обработки прерывания или подпрограмме при использовании односегментного режима РТО выполните следующие шаги:

1. Загрузите SMB67 значением 16#84 для РТО, использующего микросекундные приращения (или 16#8C для РТО, использующего миллисекундные приращения). Эти значения устанавливают управляющий байт на разрешение функции РТО/PWM, выбирают режим РТО, выбирают микросекундные или миллисекундные приращения и устанавливают обновление количества импульсов.
2. Загрузите SMD72 (двойное слово) желаемым количеством импульсов.
3. Выполните команду PLS, чтобы S7-200 запрограммировал генератор РТО/PWM. CPU должен завершить любую выводимую в данный момент РТО, прежде чем начнется вывод последовательности импульсов РТО с обновленным количеством импульсов.
4. Выйдите из программы обработки прерывания или подпрограммы.

Изменение периода следования и количества импульсов РТО – односегментный режим

Для изменения периода следования и количества импульсов РТО в программе обработки прерывания или подпрограмме при использовании односегментного режима РТО выполните следующие шаги:

1. Загрузите SMB67 значением 16#85 для РТО, использующего микросекундные приращения (или 16#8D для РТО, использующего миллисекундные приращения). Эти значения устанавливают управляющий байт на разрешение функции РТО/PWM, выбирают режим РТО, выбирают микросекундные или миллисекундные приращения и устанавливают обновление периода следования и количества импульсов.
2. Загрузите SMW68 (слово) желаемым периодом следования импульсов.
3. Загрузите SMD72 (двойное слово) желаемым количеством импульсов.
4. Выполните команду PLS, чтобы S7-200 запрограммировал генератор РТО/PWM. CPU должен завершить любую выводимую в данный момент РТО, прежде чем начнется вывод последовательности импульсов РТО с обновленным периодом следования и количеством импульсов.
5. Выйдите из программы обработки прерывания или подпрограммы.

Инициализация РТО – многосегментный режим

Для инициализации РТО выполните следующие шаги:

1. С помощью бита памяти первого цикла (SM0.1) установите выход в 0 и вызовите подпрограмму, которая вам необходима для выполнения операций по инициализации. Это сокращает время цикла и делает программу более структурированной.
2. В подпрограмме инициализации загрузите SMB67 значением 16#A0 для РТО, использующего микросекундные приращения (или 16#A8 для РТО, использующего миллисекундные приращения). Эти значения устанавливают управляющий байт на разрешение функции РТО/PWM, выбирают режим РТО и многосегментную обработку и выбирают микросекундные или миллисекундные приращения
3. Загрузите SMW168 (слово) начальным смещением таблицы профиля в V-памяти.
4. Установите необходимые значения для сегмента в таблице профиля. Убедитесь, что поле «Номер сегмента» (Number of Segment) (первый байт таблицы) заполнено правильно.
5. Этот шаг необязателен. Если вы хотите выполнить соответствующую функцию, когда вывод импульсной последовательности будет завершен, вы можете запрограммировать прерывание, поставив в соответствие программе обработки прерывания событие «Импульсная последовательность завершена» (pulse train complete) (категория прерывания 19) с помощью команды ATCH и выполнив команду ENI, разрешающую глобальные прерывания. Полное описание обработки прерываний вы найдете в разделе 9.15.
6. Выполните команду PLS, чтобы S7-200 запрограммировал генератор РТО/PWM.
7. Выйдите из подпрограммы.

Пример широтно-импульсной модуляции

На рис. 9–21 показан пример широтно-импульсной модуляции.

LAD	STL
MAIN OB1 [Главная программа OB1]	
<p>Network 1</p> <p>Network 2</p>	<p>Network 1</p> <pre>LD SM0.1 R Q0.1, 1 CALL 0</pre> <p>Network 2</p> <pre>LD M0.0 EU CALL 1</pre>
SUBROUTINE 0 [Подпрограмма 0]	
<p>Network 1</p>	<p>Network 1</p> <pre>LD SM0.0 MOVB 16#DB, SMB77 MOVW 10000, SMW78 MOVW 1000, SMW80 PLS 1 MOVB 16#DA, SMB77</pre>
SUBROUTINE 1 [Подпрограмма 1]	
<p>Network 1</p>	<p>Network 1</p> <pre>LD SM0.0 MOVW 5000, SMW80 PLS 1</pre>

Рис. 9-21. Пример скоростного вывода, использующего широтно-импульсную модуляцию

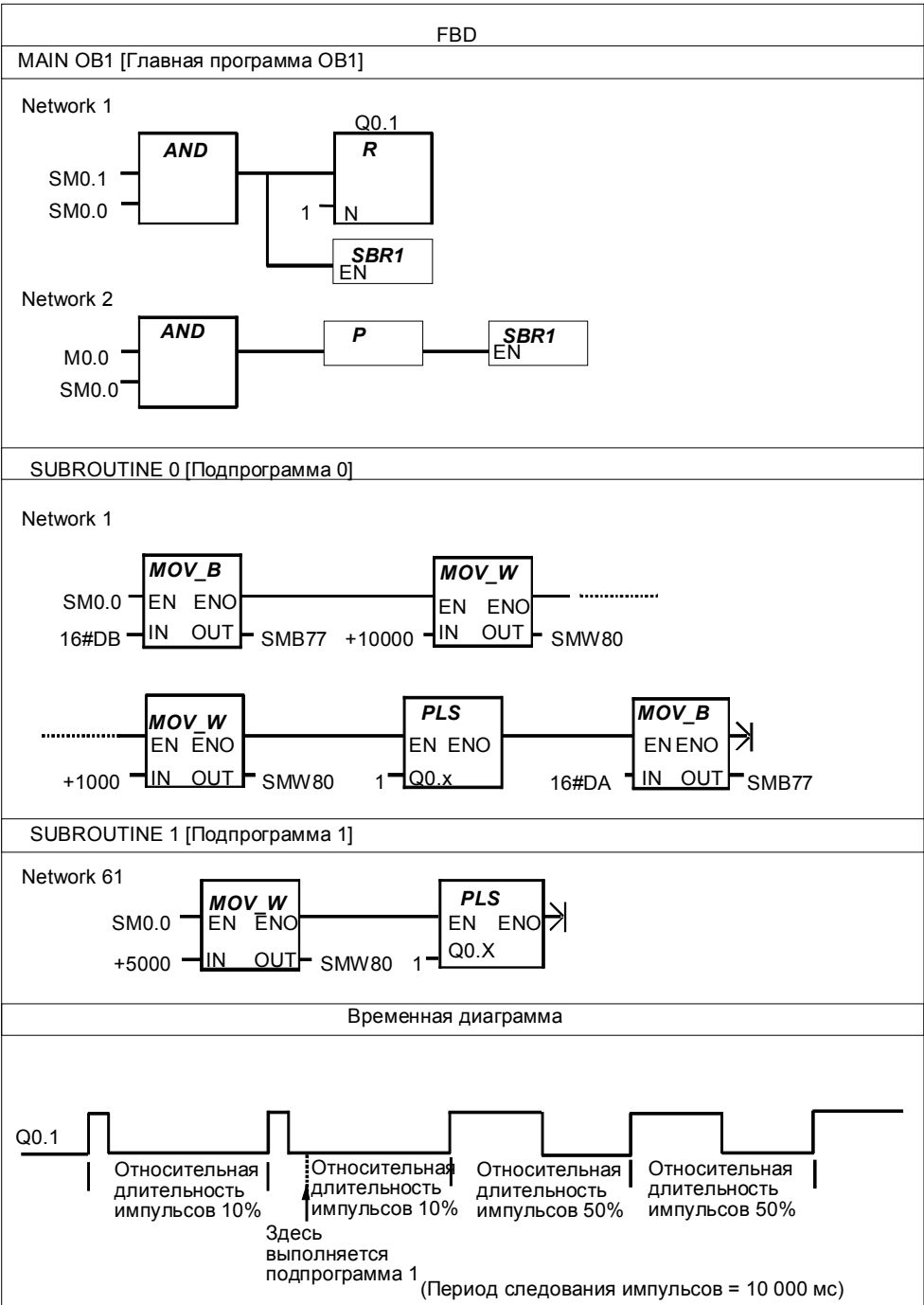


Рис. 9-21. Пример скоростного вывода, использующего широтно-импульсную модуляцию (продолжение)

Пример вывода последовательности импульсов с использованием односегментного режима

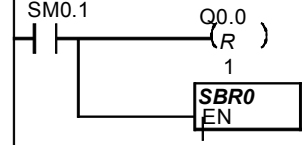
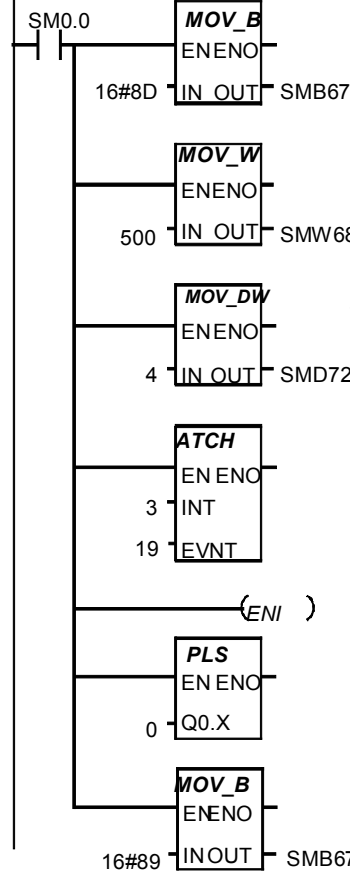
LAD	STL
MAIN OB1 [Главная программа OB1]	
<div>Network 1</div> <div></div> <div>В первом цикле сбросить бит в регистре образа процесса и вызвать подпрограмму 0.</div>	<div>Network 1</div> <div>LD SM0.1</div> <div>R Q0.0, 1</div> <div>CALL 0</div>
SUBROUTINE 0 [Подпрограмма 0]	
<div>Network 1</div> <div></div> <div>Установка управляющего байта: - выбрать режим PTO - выбрать миллисекундные приращения - установить количество и период следования импульсов - разблокировать функцию PTO Установить период следования импульсов 500 мс. Установить число импульсов = 4. Поставить программу прерывания 3 в соответствие событию «Завершение обработки PTO» Разрешить глобальные прерывания. Вызвать режим PTO PLS 0 => Q0.0 Предварительно загрузить управляющий байт для последующих изменений периода следования импульсов.</div>	<div>Network 1</div> <div>LD SM0.0</div> <div>MOVB 16#8D, SMB67</div> <div>MOVW 500, SMW68</div> <div>MOVD 4, SMD72</div> <div>ATCH 3, 19</div> <div>ENI</div> <div>PLS 0</div> <div>MOVB 16#89, SMB67</div>

Рис. 9-22. Пример вывода последовательности импульсов с использованием односегментного режима в SM-памяти

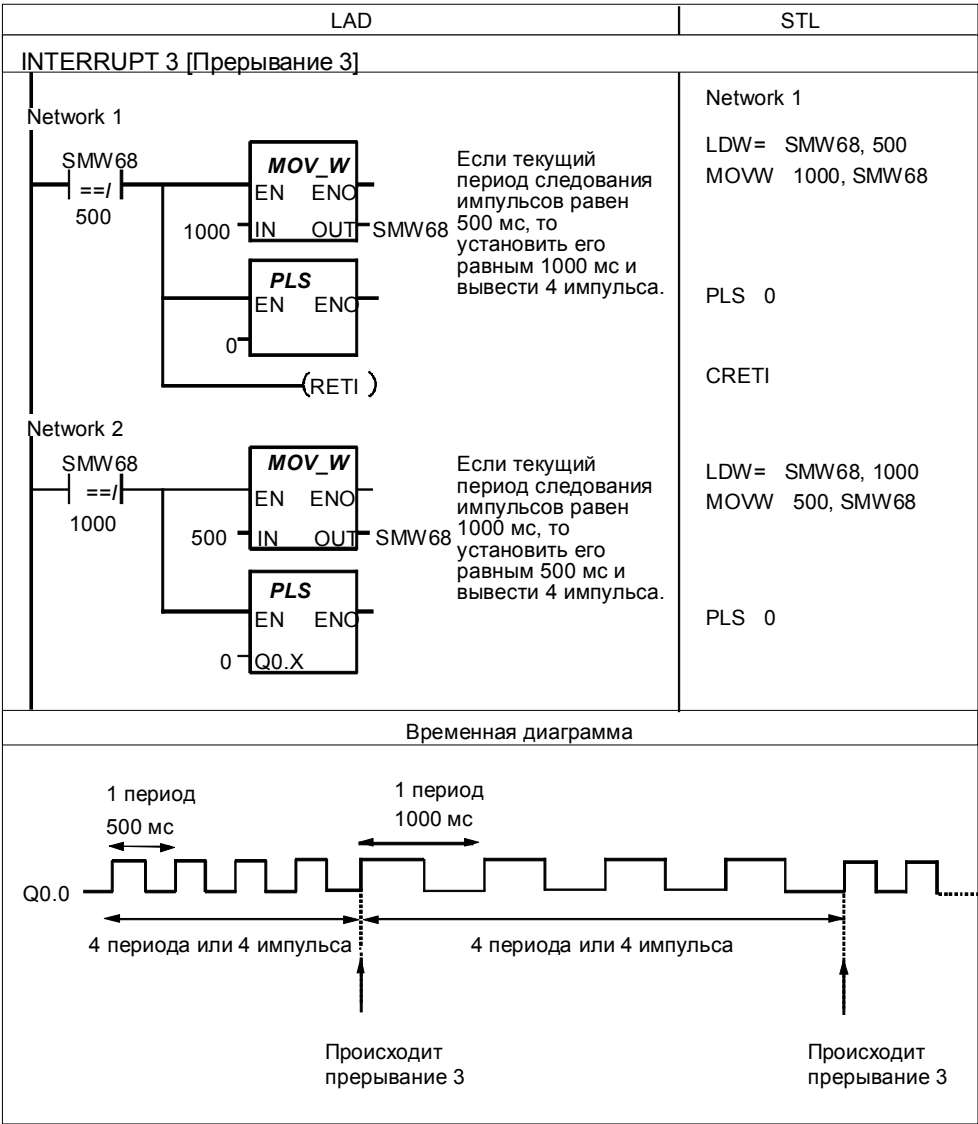


Рис. 9-22. Пример вывода последовательности импульсов с использованием односегментного режима (продолжение)

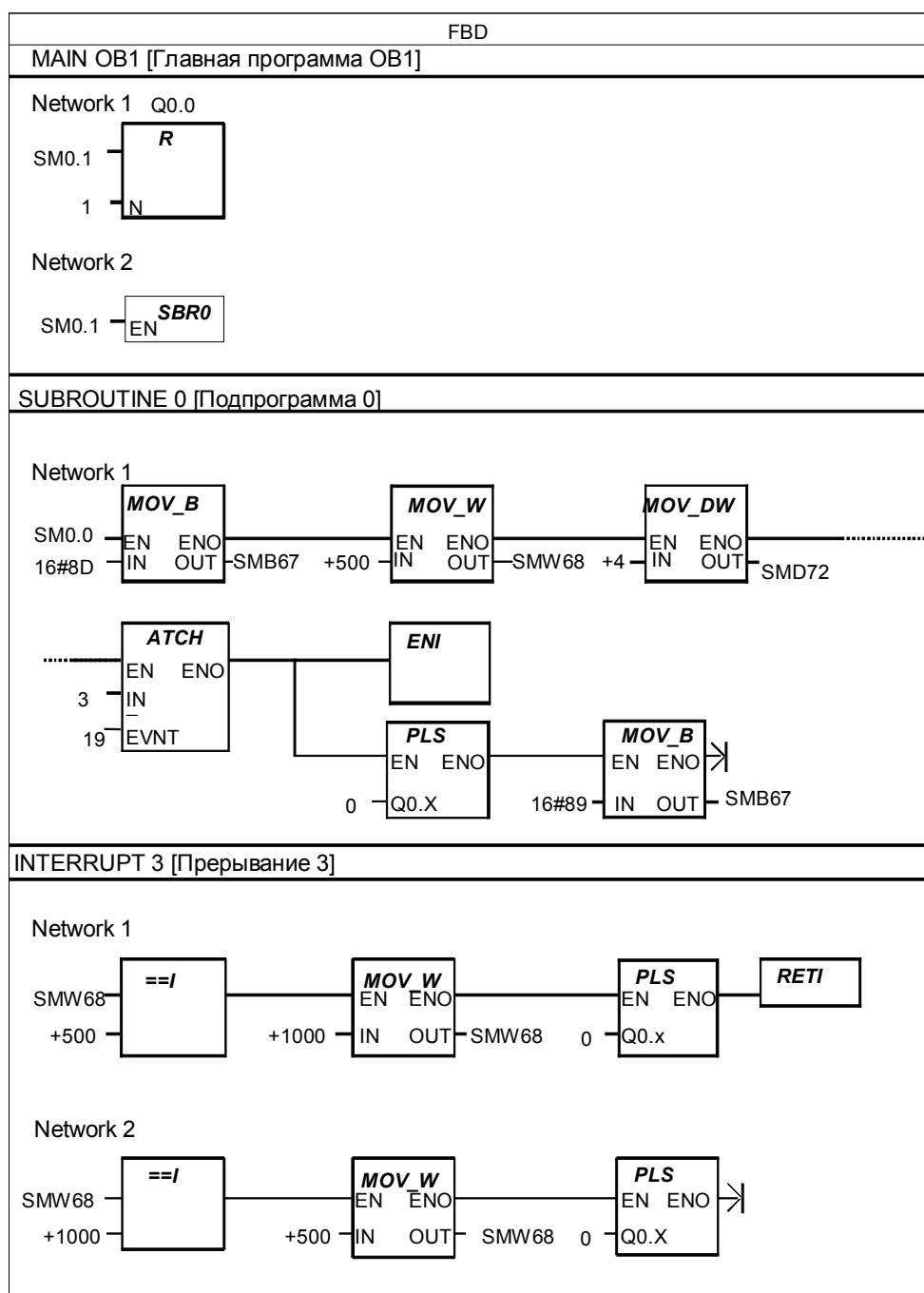


Рис. 9-22. Пример вывода последовательности импульсов с использованием односегментного режима (продолжение)

Пример вывода последовательности импульсов с использованием многосегментного режима

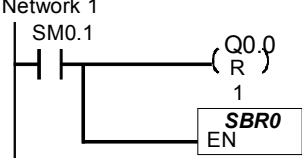
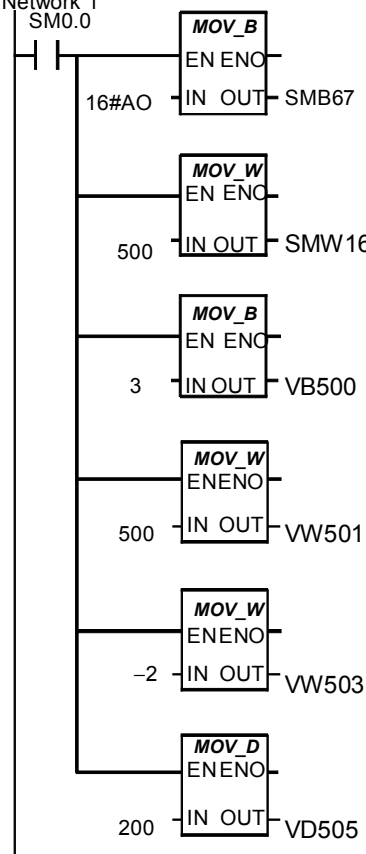
LAD	STL
MAIN OB1 [Главная программа OB1]	
<p>Network 1</p>  <p>В первом цикле сбросить бит в регистре образа процесса и вызвать подпрограмму 0</p>	<p>Network 1</p> <pre>LD SM0.1 R Q0.0, 1 CALL 0</pre>
SUBROUTINE 0 [Подпрограмма 0]	
<p>Network 1</p>  <p>Установка управляющего байта: - выбрать режим PTO - выбрать многосегментную обработку - выбрать миллисекундные приращения - разблокировать функцию PTO</p> <p>Задать начальный адрес таблицы профиля V500</p> <p>Установить количество сегментов таблицы профиля = 3</p> <p>Установить начальный период следования импульсов для сегмента #1 = 500 мс</p> <p>Установить приращение периода следования импульсов для сегмента #1 = -2 мс</p> <p>Установить количество импульсов в сегменте #1 = 200.</p>	<p>Network 1</p> <pre>LD SM0.0 MOVB 16#A0, SMB67 MOVW 500, SMW168 MOVB 3, VB500 MOVW 500, VW501 MOVW -2, VD503 MOVD 200, VD505</pre>

Рис. 9-23. Пример вывода последовательности импульсов с использованием многосегментного режима

LAD		STL
Network 1		
		MOVW 100, VW509
		MOVW 0, VW511
		MOVD 3400, VD513
		MOVW 100, VW517
		MOVW 1, VW519
		MOVD 400, VD521
		ATCH 2, 19
		ENI
		PLS 0
INTERRUPT 0 [Прерывание 0]		
Network 1		Network 1
		LD SM0.0 = Q0.5

Рис. 9-23. Пример вывода последовательности импульсов с использованием многосегментного режима (продолжение)

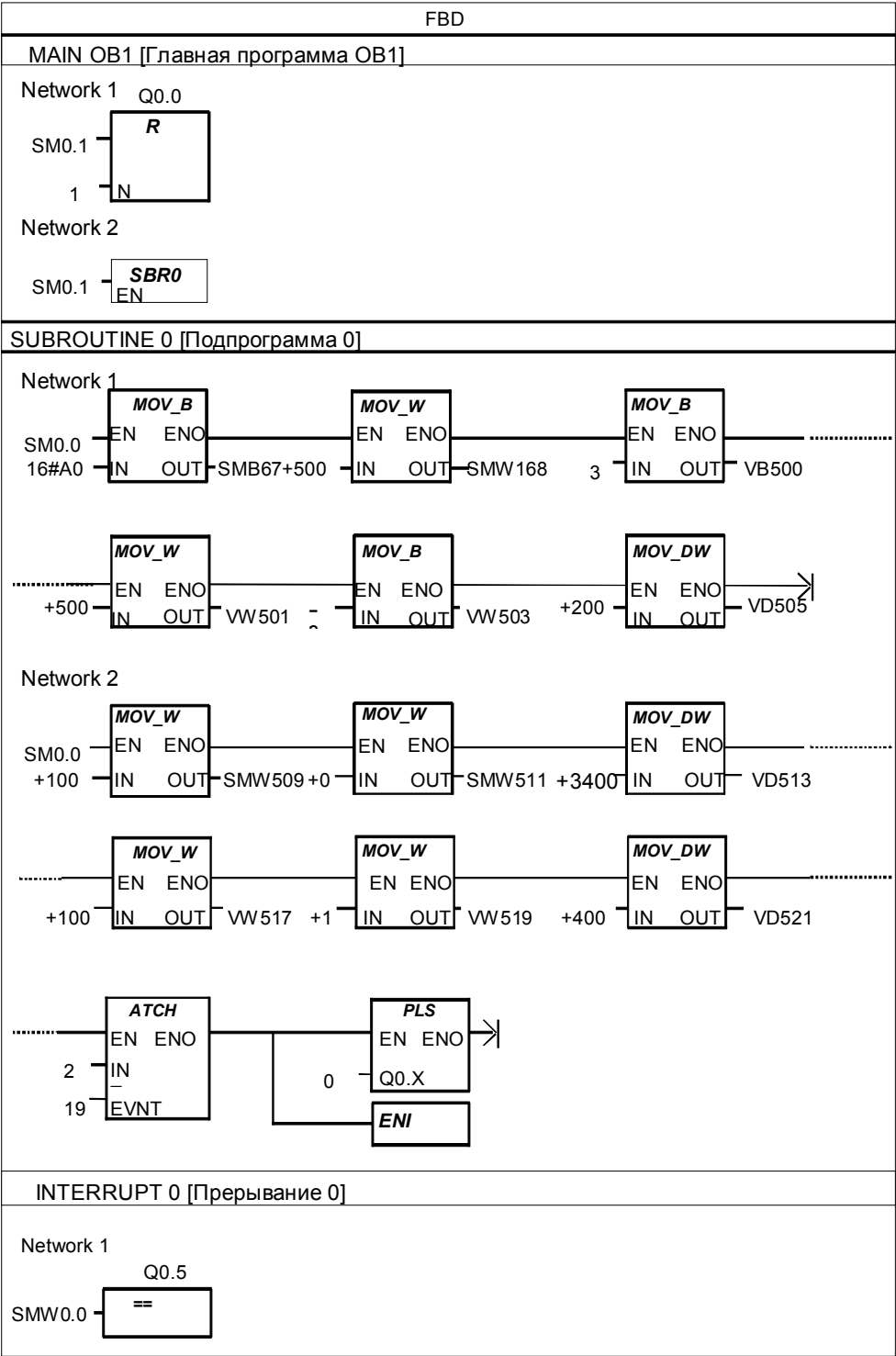
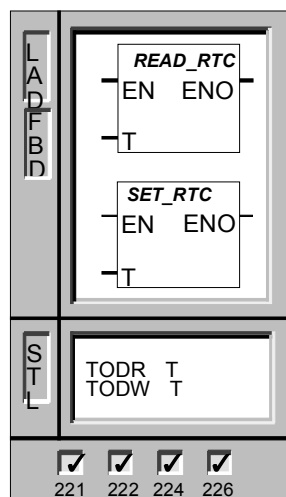


Рис. 9-23. Пример вывода последовательности импульсов с использованием многосегментного режима (продолжение)

9.7 Команды SIMATIC, выполняемые над часами

Чтение и установка часов реального времени



Команда **Прочитать часы реального времени** считывает текущее время и дату из часов и загружает их в 8-байтовый буфер, начиная с адреса T.

Команда **Установить часы реального времени** записывает в часы текущее время и дату, загруженные в 8-байтовый буфер, начинающийся с адреса T.

В STL команды TODR и TODW означают «Прочитать время суток» (Time of Day Read, TODR) и «Записать время суток» (Time of Day Write, TODW).

TODR: Ошибки, устанавливающие ENO в 0: SM4.3 (этап исполнения), 0006 (косвенная адресация), 000C (отсутствует картридж часов)

TODW: Ошибки, устанавливающие ENO в 0: SM 4.3 (исполнение), 0006 (косвенная адресация), 0007 (ошибка данных TOD), 000C (отсутствует картридж часов)

Входы/выходы	Операнды	Типы данных
T	VB, IB, QB, MB, SMB, SB, LB, *VD, *AC, *LD	BYTE

Рис. 9–24 показывает формат буфера времени (T).

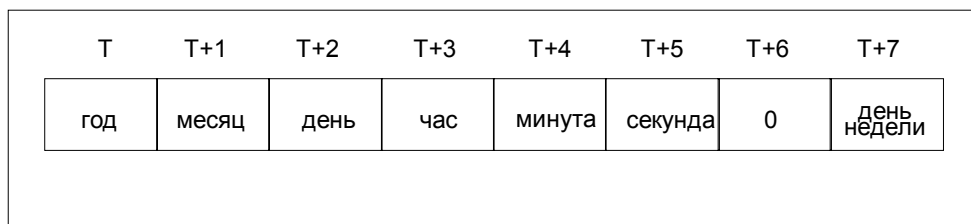


Рис. 9-24. Формат буфера времени

После продолжительного отключения питания или потери памяти часы реального времени инициализируют следующую дату и время:

Дата: 01-Янв-90

Время: 00:00:00

День недели: Воскресенье

Часы реального времени в S7-200 используют только две младшие значащие цифры для года, так что 2000-й год представлен как 00.

Все значения даты и времени должны быть закодированы в двоично-десятичном (BCD) формате (например, 16#97 для 1997 года).

Используйте следующие форматы:

Год/месяц	гмм	гг - 0 – 99	мм - 1 – 12
День/час	ддчч	дд - 1 – 31	чч - 0 – 23
Минута/секунда	ммсс	мм - 0 – 59	сс - 0 – 59
День недели	д	д - 0 – 7	1 = воскресенье 0 = блокирует день недели (сохраняется 0)

Примечание

CPU S7-200 не проверяет соответствие дня недели дате. Могут быть восприняты неверные даты, например, 30 февраля. Правильность даты должны обеспечивать вы сами.

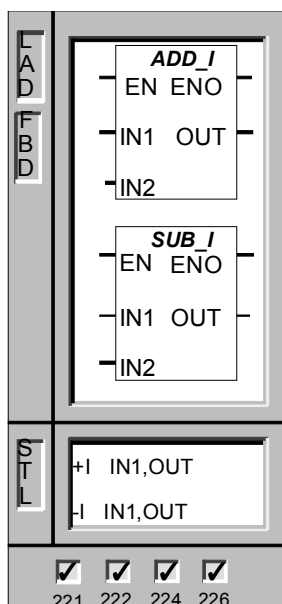
Не используйте команду TODR/TODW одновременно в главной программе и программе обработки прерывания. Эта команда не будет исполнена в программе обработки прерывания, которая пытается ее выполнить, когда действует другая команда TODR/TODW. Если делается попытка одновременно двух обращений к часам, то устанавливается SM4.3 (нефатальная ошибка 0007).

ПЛК S7-200 никак не использует информацию о годе, и на него не влияет переход из одного столетия в другое (проблема 2000). Однако прикладные программы, которые используют арифметические операции или сравнения со значением года, должны учитывать двухзначное представление и изменение столетия.

Високосные годы обрабатываются правильно до 2096 года.

9.8 Арифметические команды SIMATIC над целыми числами

Сложение и вычитание целых чисел



Команды **Сложить целые числа** и **Вычесть целые числа** складывают или вычитают два 16-битовых целых числа и дают 16-битовый результат (OUT).

В LAD и FBD: $IN1 + IN2 = OUT$
 $IN1 - IN2 = OUT$

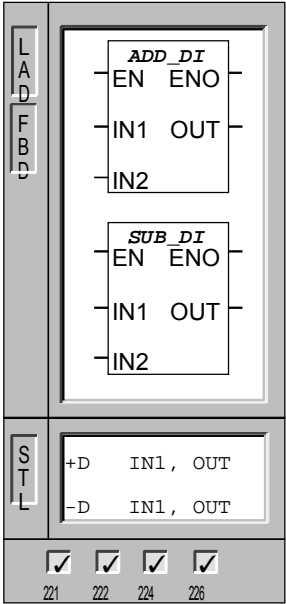
В STL: $IN1 + OUT = OUT$
 $OUT - IN1 = OUT$

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение); SM1.2 (отрицательное число)

Входы/выходы	Операнды	Типы данных
IN1, IN2	VW, IW, QW, MW, SW, SMW, LW, AIW, T, C, AC, константа, *VD, *AC, *LD	INT
OUT	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, *VD, *AC, *LD	INT

Сложение и вычитание двойных целых чисел



Команды **Сложить двойные целые числа** и **Вычесть двойные целые числа** складывают или вычитают два 32-битовых целых числа и дают 32-битовый результат (OUT).

В LAD и FBD: $IN1 + IN2 = OUT$
 $IN1 - IN2 = OUT$

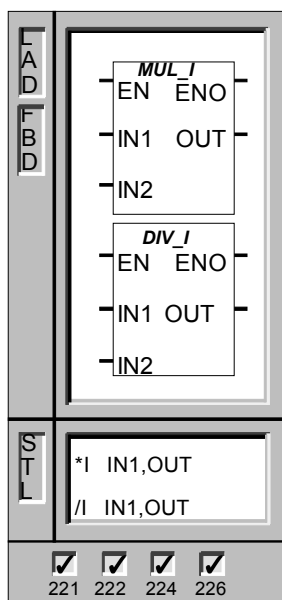
В STL: $IN1 + OUT = OUT$
 $OUT - IN1 = OUT$

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM4.3 (этап исполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение); SM1.2 (отрицательное число)

Входы/выходы	Операнды	Типы данных
IN1, IN2	VD, ID, QD, MD, SMD, SD, LD, AC, HC, константа, *VD, *AC, *LD	DINT
OUT	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *AC, *LD	DINT

Умножение и деление целых чисел



Команда **Умножить целые числа** перемножает два 16-битовых целых числа и дает 16-битовое произведение.

Команда **Разделить целые числа** делит два 16-битовых целых числа и дает 16-битовое частное. Остаток не сохраняется.

Бит переполнения устанавливается, если результат занимает более одного слова.

В LAD и FBD:

$IN1 * IN2 = OUT$
 $IN1 / IN2 = OUT$

В STL:

$IN1 * OUT = OUT$
 $OUT / IN1 =$

OUT

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM1.3 (деление на ноль), SM4.3 (этап исполнения), 0006 (косвенная адресация)

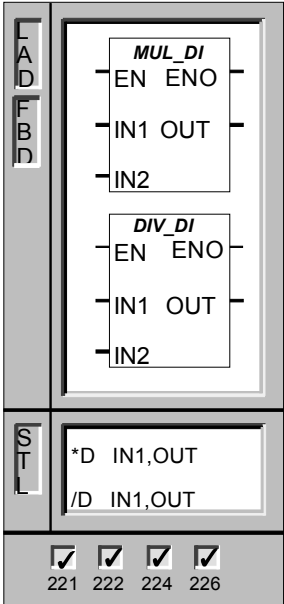
Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение); SM1.2 (отрицательное число); SM1.3 (деление на ноль)

Если во время операции умножения или деления устанавливается SM1.1 (переполнение), то выход не записывается, и все остальные биты состояния арифметической операции устанавливаются в ноль.

Если во время операции деления устанавливается SM1.3 (деление на ноль), то все остальные биты состояния арифметической операции остаются неизменными, первоначальные входные операнды не меняются. В противном случае все поддерживаемые биты состояния арифметической операции содержат после завершения арифметической операции допустимый статус.

Входы/выходы	Операнды	Типы данных
IN1, IN2	VW, IW, QW, MW, SW, SMW, LW, AIW, T, C, AC, константа, *VD, *AC, *LD	INT
OUT	VW, QW, IW, MW, SW, SMW, LW, T, C, AC, *VD, *LD, *AC	INT

Умножение и деление двойных целых чисел



Команда **Умножить двойные целые числа** перемножает два 32-битовых целых числа и дает 32-битовое произведение.

Команда **Разделить двойные целые числа** делит два 32-битовых целых числа и дает 32-битовое частное. Остаток не сохраняется.

В LAD и FBD:

$IN1 * IN2 = OUT$
 $IN1 / IN2 = OUT$

В STL:

$OUT = IN1 * OUT =$
 $OUT / IN1 =$

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM1.3 (деление на ноль), SM4.3 (этап исполнения), 0006 (косвенная адресация)

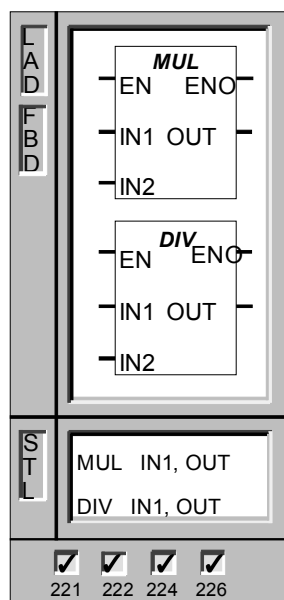
Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение); SM1.2 (отрицательное число); SM1.3 (деление на ноль)

Если во время операции умножения или деления устанавливается SM1.1 (переполнение), то выход не записывается, и все остальные биты состояния арифметической операции устанавливаются в ноль.

Если во время операции деления устанавливается SM1.3 (деление на ноль), то все остальные биты состояния арифметической операции остаются неизменными, первоначальные входные операнды не меняются. В противном случае все поддерживаемые биты состояния арифметической операции содержат после завершения арифметической операции допустимый статус.

Входы/выходы	Операнды	Типы данных
IN1, IN2	VD, ID, QD, MD, SMD, SD, LD, HC, AC, константа, *VD, *AC, *LD	DINT
OUT	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *LD, *AC	DINT

Умножение и деление целых чисел с представлением результата в виде двойного целого числа



Команда **Умножить целые числа с представлением результата в виде двойного целого числа** перемножает два 16-битовых целых числа и дает 32-битовое произведение.

Команда **Разделить целые числа с представлением результата в виде двойного целого числа** делит два 16-битовых целых числа и дает 32-битовый результат, состоящий из 16-битового остатка (старшее слово) и 16-битового частного (младшее слово).

В команде умножения STL младшее слово (16 битов) 32-битового выхода OUT используется как один из сомножителей.

В команде деления STL младшее слово (16 битов) 32-битового выхода OUT используется как делимое.

В LAD и FBD:

$IN1 * IN2 = OUT$
 $IN1 / IN2 = OUT$

В STL:
 $IN1 * OUT = OUT$
 $OUT / IN1 = OUT$

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM1.3 (деление на ноль), SM4.3 (этап исполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение); SM1.2 (отрицательное число); SM1.3 (деление на ноль)

Если во время операции деления устанавливается SM1.3 (деление на ноль), то все остальные биты состояния арифметической операции остаются неизменными, первоначальные входные операнды не меняются. В противном случае все поддерживаемые биты состояния арифметической операции содержат после завершения арифметической операции допустимый статус.

Входы/выходы	Операнды	Типы данных
IN1, IN2	VW, IW, QW, MW, SW, SMW, LW, AC, AIW, T, C, константа, *VD, *AC, *LD	INT
OUT	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *LD, *AC	DINT

Примеры арифметических операций

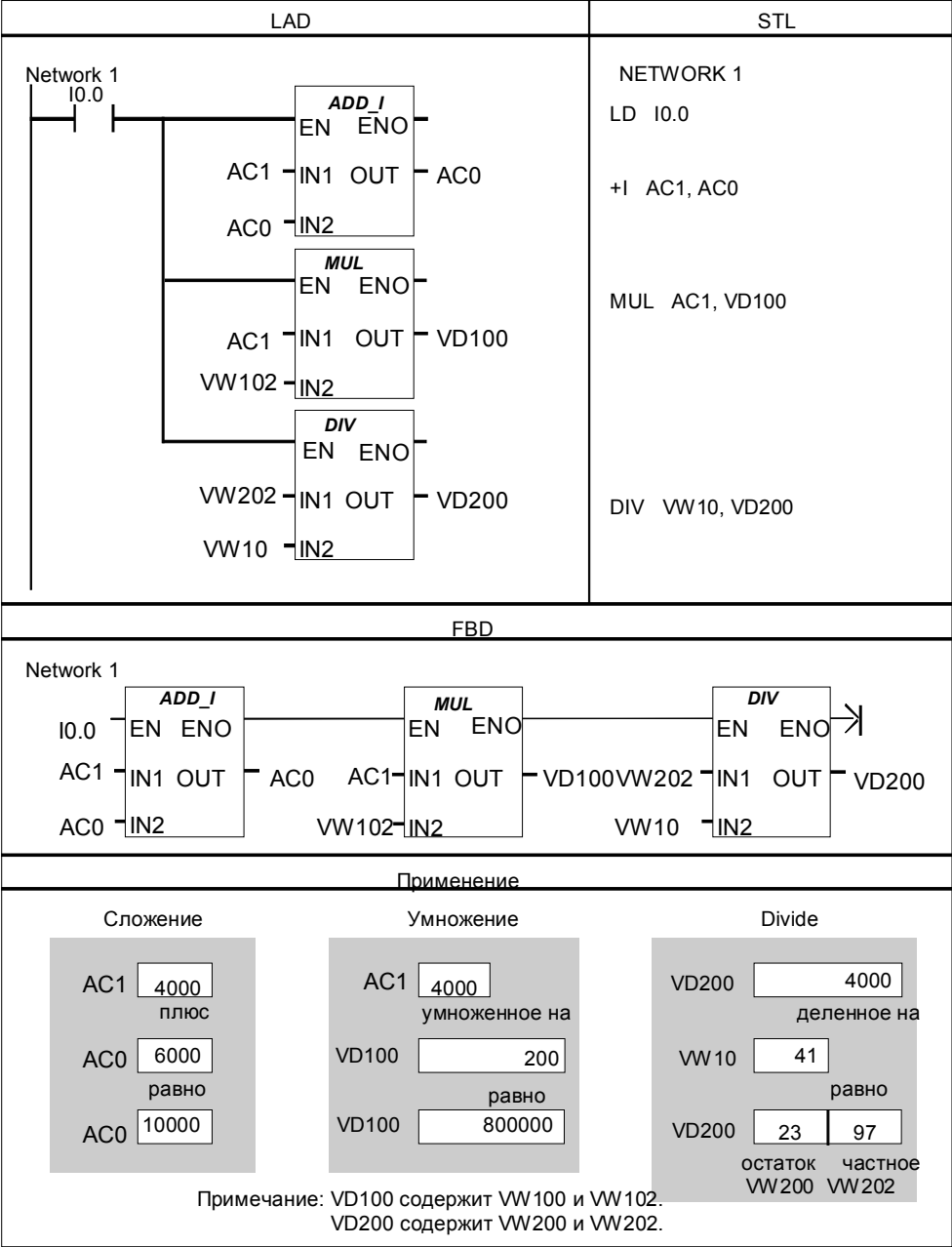
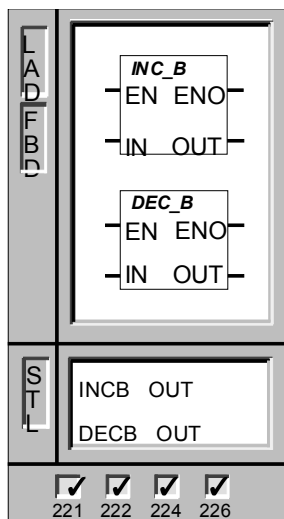


Рис. 9-25. Примеры арифметических операций с целыми числами для SIMATIC LAD, STL и FBD

Увеличение и уменьшение байта на 1



Команды **Увеличить байт на 1** и **Уменьшить байт на 1** прибавляют к входному байту (IN) или вычитают из него 1 и помещают результат в переменную, определенную OUT.

Операции увеличения и уменьшения байта на 1 являются беззнаковыми.

В LAD и FBD:

$IN + 1 = OUT$

$IN - 1 = OUT$

В STL:

$OUT + 1 = OUT$

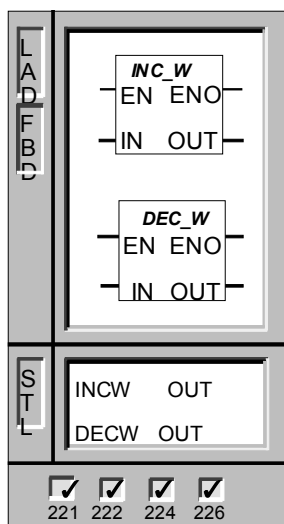
$OUT - 1 = OUT$

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM4.3 (этап исполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение)

Входы/выходы	Операнды	Типы данных
IN	VB, IB, QB, MB, SB, SMB, LB, AC, константа, *VD, *AC, *LD	BYTE
OUT	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD	BYTE

Увеличение и уменьшение слова на 1



Команды **Увеличить слово на 1** и **Уменьшить слово на 1** прибавляют к входному слову (IN) или вычитают из него 1 и помещают результат в OUT. Операции увеличения и уменьшения слова на 1 учитывают знак ($16\#7FFF > 16\#8000$).

В LAD и FBD:

$IN + 1 = OUT$

$IN - 1 = OUT$

В STL:

$OUT + 1 = OUT$

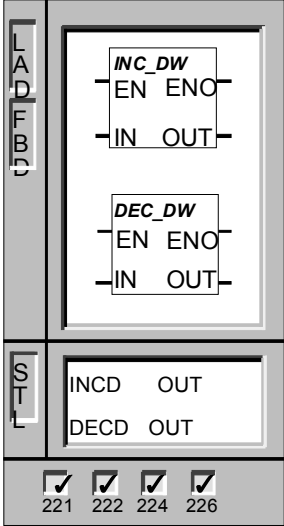
$OUT - 1 = OUT$

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM4.3 (этап исполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение); SM1.2 (отрицательное число)

Входы/выходы	Операнды	Типы данных
IN	VW, IW, QW, MW, SW, SMW, AC, AIW, LW, T, C, константа, *VD, *AC, *LD	INT
OUT	VW, IW, QW, MW, SW, SMW, LW, AC, T, C, *VD, *AC, *LD	INT

Увеличение и уменьшение двойного слова на 1



Команды **Увеличить двойное слово на 1** и **Уменьшить двойное слово на 1** прибавляют к входному двойному слову (IN) или вычитают из него 1 и помещают результат в OUT.

В LAD и FBD: $IN + 1 = OUT$
 $IN - 1 = OUT$

Операции увеличения и уменьшения двойного слова на 1 учитывают знак (16#7FFFFFFF > 16#80000000).

В STL: $OUT + 1 = OUT$
 $OUT - 1 = OUT$

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM4.3 (этап исполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение); SM1.2 (отрицательное число)

Входы/выходы	Операнды	Типы данных
IN	VD, ID, QD, MD, SD, SMD, LD, AC, HC, константа, *VD, *AC, *LD	DINT
OUT	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD	DINT

Пример увеличения и уменьшения на 1

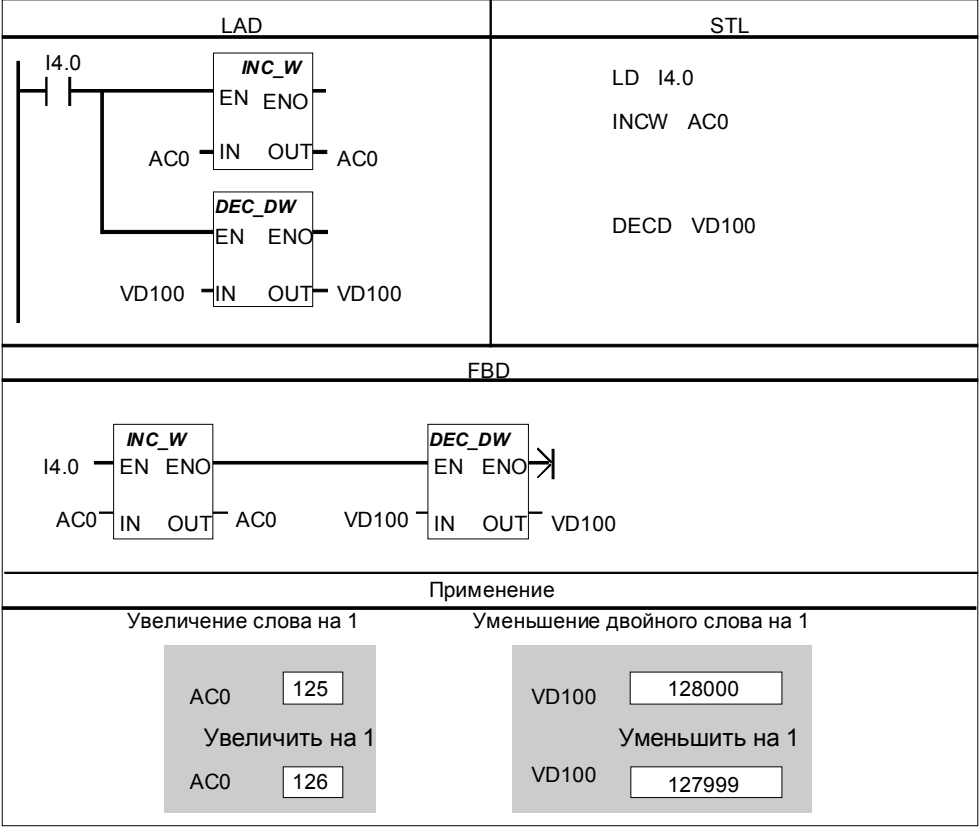
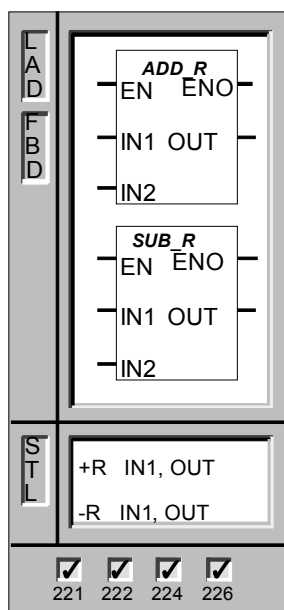


Рис. 9-26. Пример команд увеличения/уменьшения на 1 для SIMATIC LAD, STL и FBD

9.9 Арифметические команды SIMATIC над вещественными числами

Сложение и вычитание вещественных чисел



Команды **Сложить вещественные числа** и **Вычесть вещественные числа** складывают или вычитают два 32-битовых вещественных числа и дают результат в виде 32-битового вещественного числа(OUT).

В LAD и FBD: $IN1 + IN2 = OUT$

$IN1 - IN2 = OUT$

В STL: $IN1 + OUT = OUT$

$OUT - IN1 = OUT$

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM4.3 (этап исполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение); SM1.2 (отрицательное число)

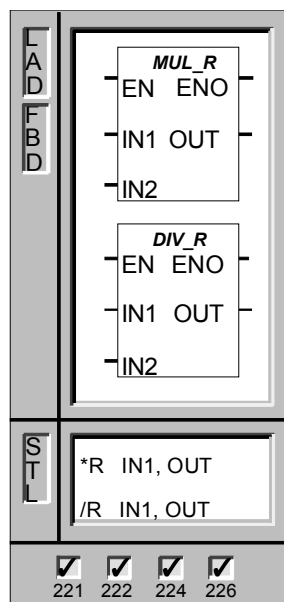
SM1.1 используется для индикации ошибок переполнения и недопустимых значений. Если SM1.1 установлен, то состояние SM1.0 и SM1.2 не имеет значения, первоначальные входные операнды не изменяются. Если SM1.1 не установлен, то арифметическая операция завершилась с допустимым результатом, а SM1.0 и SM1.2 содержат действительное состояние.

Входы/выходы	Операнды	Типы данных
IN1, IN2	VD, ID, QD, MD, SD, SMD, AC, LD, константа, *VD, *AC, *LD	REAL
OUT	VD, ID, QD, MD, SD, SMD, AC, LD, *VD, *AC, *LD	REAL

Примечание

Вещественные числа, или числа с плавающей точкой, представляются в формате, описанном в стандарте ANSI/IEEE 754-1985 (одинарная точность). За дополнительной информацией обратитесь к стандарту.

Умножение и деление вещественных чисел



Команда **Умножить вещественные числа** перемножает два 32-битовых вещественных числа и дает результат в виде 32-битового вещественного числа (OUT).

Команда **Разделить вещественные числа** делит два 32-битовых вещественных числа и дает частное в виде 32-битового вещественного числа.

В LAD и FBD: $IN1 * IN2 = OUT$
 $IN1 / IN2 = OUT$

В STL: $IN1 * OUT = OUT$
 $OUT / IN1 = OUT$

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM1.3 (деление на ноль), SM4.3 (этап исполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение, или во время операции образовалось недопустимое значение, или обнаружен недопустимый входной параметр); SM1.2 (отрицательное число); SM1.3 (деление на ноль). Если во время операции деления устанавливается SM1.3, то все остальные биты состояния арифметической операции остаются неизменными, первоначальные входные операнды не меняются. SM1.1 используется для индикации ошибок переполнения и недопустимых значений. Если SM1.1 установлен, то состояние SM1.0 и SM1.2 не имеет значения, и первоначальные входные операнды не меняются. Если SM1.1 и SM1.3 (во время операции деления) не устанавливаются, то арифметическая операция завершилась с допустимым результатом, и SM1.0 и SM1.2 содержат допустимый статус.

Входы/выходы	Операнды	Типы данных
IN1, IN2	VD, ID, QD, MD, SMD, SD, LD, AC, константа, *VD, *AC, *LD	REAL
OUT	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *AC, *LD	REAL

Примечание

Вещественные числа, или числа с плавающей точкой, представляются в формате, описанном в стандарте ANSI/IEEE 754-1985 (одинарная точность). За дополнительной информацией обратитесь к стандарту.

Примеры арифметических операций

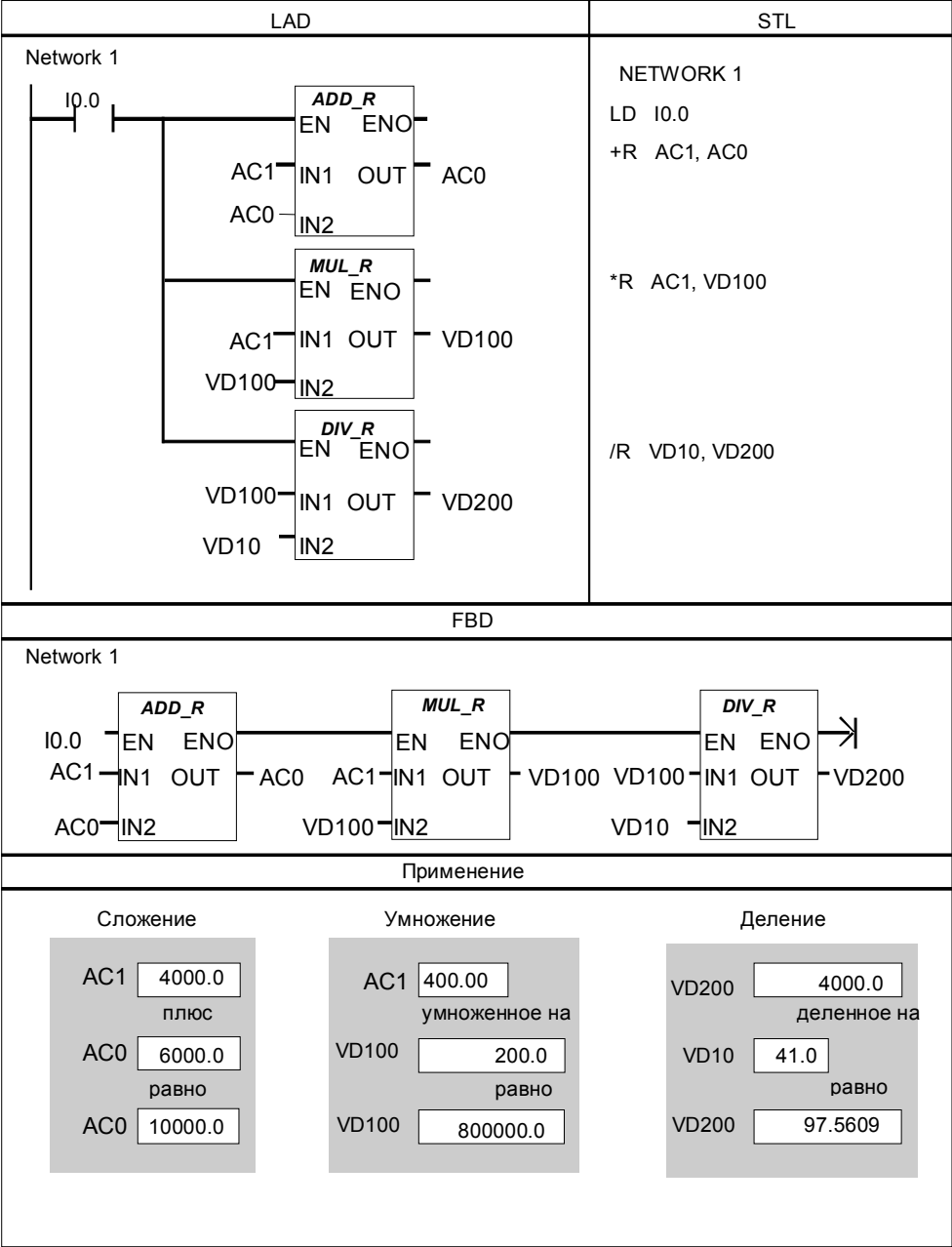
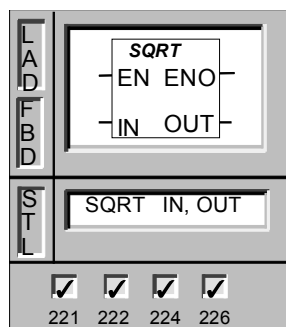


Рис. 9-27. Примеры арифметических операций с вещественными числами для SIMATIC LAD, STL и FBD

9.10 Команды SIMATIC с числовыми функциями

Квадратный корень



Команда **Квадратный корень** извлекает квадратный корень из 32-битового вещественного числа (IN) и дает результат в виде 32-битового вещественного числа (OUT), как показано в уравнении:

$$\sqrt{IN} = OUT$$

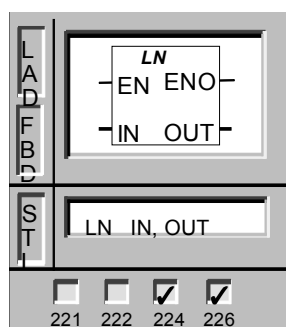
Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM4.3 (этап исполнения), 0006 (косвенная адресация)

Эта команда влияет на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение); SM1.2 (отрицательное число).

SM1.1 используется для индикации ошибок переполнения и недопустимых значений. Если SM1.1 установлен, то состояние SM1.0 и SM1.2 не имеет значения, первоначальные входные операнды не изменяются. Если SM1.1 не установлен, то арифметическая операция завершилась с допустимым результатом, а SM1.0 и SM1.2 содержат действительное состояние. Для получения других корней см. команду **Натуральный экспонента** на стр. 9–86.

Входы/выходы	Операнды	Типы данных
IN	VD, ID, QD, MD, SMD, SD, LD, AC, константа, *VD, *AC, *LD	REAL
OUT	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *AC, *LD	REAL

Натуральный логарифм



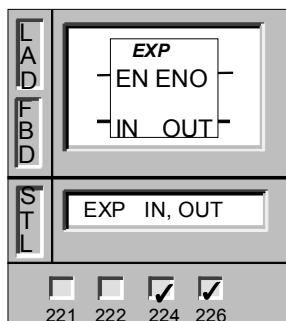
Команда **Натуральный логарифм** находит натуральный логарифм числа в IN и помещает результат в OUT. Для получения десятичного логарифма из натурального используйте DIV_R (/R) для деления натурального логарифма на 2.302585 (примерно натуральный логарифм 10).

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), 0006 (косвенная адресация)

Эта команда влияет на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение); SM1.2 (отрицательное число), SM4.3 (ошибка этапа выполнения).

Входы/выходы	Операнды	Типы данных
IN	VD, ID, QD, MD, SMD, SD, LD, AC, константа, *VD, *AC, *LD	REAL
OUT	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *AC, *LD	REAL

Натуральная экспонента



Команда **Натуральная экспонента** выполняет экспоненциальную операцию по возведению e в степень, заданную значением в IN, и помещает результат в OUT. Натуральная экспонента может быть использована в сочетании с Натуральным логарифмом для возведения любого вещественного числа в степень, заданную другим вещественным числом, включая дробные. А именно, X в степени Y может быть вычислен как $EXP(Y * LN X)$.

Примеры:

5 в кубе = $5^3 = EXP(3 * LN(5)) = 125$

Кубический корень из 125 = $125^{1/3} = EXP(1/3 * LN(125)) = 5$

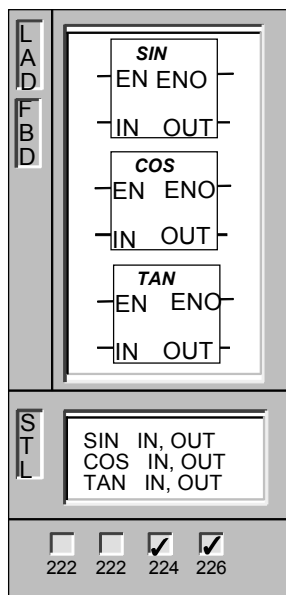
Квадратный корень из 5 в кубе = $5^{3/2} = EXP(3/2 * LN(5)) = 11.18034 \dots$

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), 0006 (косвенная адресация)

Эта команда влияет на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение); SM1.2 (отрицательное число), SM4.3 (ошибка этапа выполнения).

Входы/выходы	Операнды	Типы данных
IN	VD, ID, QD, MD, SMD, SD, LD, AC, константа, *VD, *AC, *LD	REAL
OUT	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *AC, *LD	REAL

Синус, косинус и тангенс



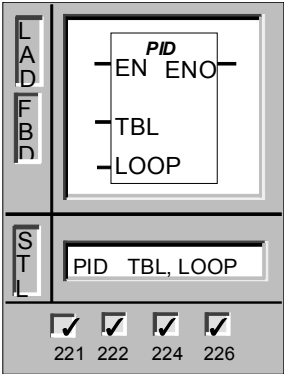
Команды **Синус**, **Косинус** и **Тангенс** вычисляют тригонометрическую функцию угловой величины IN и помещают результат в OUT. Входной угол задается в радианах. Для преобразования угла из градусов в радианы используйте MUL_R (*R) для умножения угла в градусах на 1.745329E-2 (примерно на $\pi / 180$).

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), 0006 (косвенная адресация)

Эта команда влияет на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение); SM1.2 (отрицательное число), SM4.3 (ошибка этапа выполнения).

Входы/выходы	Операнды	Типы данных
IN	VD, ID, QD, MD, SMD, SD, LD, AC, константа, *VD, *AC, *LD	REAL
OUT	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *AC, *LD	REAL

PID-регулятор



Команда **PID-регулятор** выполняет расчет контура PID-регулятора для заданного контура регулирования LOOP с помощью данных о входных величинах и конфигурации в таблице (TBL).
Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)
Эта команда влияет на следующие биты специальной памяти: SM1.1 (переполнение)

Входы/выходы	Операнды	Типы данных
TBL	VB	BYTE
LOOP	Константа (от 0 до 7)	BYTE

Команда PID-регулятор (пропорционально-интегрально-дифференциальный регулятор) предназначена для расчета PID-регуляторов. Чтобы эти расчеты можно было выполнять, вершина логического стека (TOS) должна быть активизирована (поток сигнала). Команда имеет два операнда: TABLE, являющийся начальным адресом таблицы с данными контура регулирования, и LOOP – номер контура регулирования, являющийся константой от 0 до 7. В программе можно использовать до восьми команд PID. Если две или более команд PID используются с одним и тем же номером контура регулирования (даже если у них разные номера таблиц), то расчеты PID-регуляторов будут влиять друг на друга, и выход будет непредсказуемым.

Таблица контура регулирования хранит девять параметров, используемых для управления и контроля за работой контура регулирования. Сюда входят текущее и предыдущее значение регулируемой переменной (фактическое значение), заданное значение, регулирующее воздействие (выход), коэффициент усиления, период квантования, постоянная времени интегрирования (сброс), постоянная времени воздействия по производной (скорость) и интегральная сумма (смещение).

Для выполнения расчета PID-регулятора с желаемым временем квантования команда PID-регулятор должна выполняться или из программы обработки прерывания по времени, или из главной программы со скоростью, управляемой таймером. Время квантования должно подаваться как вход в команду PID-регулятор через таблицу контура регулирования.

Использование мастера PID в STEP 7–Micro/WIN 32

STEP 7–Micro/WIN 32 предоставляет в распоряжение мастер PID для руководства вами при определении PID-алгоритма для процесса с замкнутым контуром управления. Выберите команду меню **Tools** → **Instruction Wizard** [Инструментальные средства → Мастер команд], а затем выберите PID из окна Instruction Wizard [Мастер команд].

PID-алгоритм

В установленном режиме PID-регулятор управляет своим выходом (регулирующим воздействием) таким образом, чтобы свести ошибку регулирования (e) к нулю. Мерой ошибки является разность между заданным значением (setpoint, SP) и значением регулируемой переменной (process variable, PV) (фактическое значение). Принцип PID-регулятора основан на следующем уравнении, которое выражает регулирующее воздействие $M(t)$ как функцию пропорциональной составляющей, интегральной составляющей и дифференциальной составляющей:

$M(t)$	=	$K_C * e$	+	$K_C \int e \, dt + M_{\text{нач}}$	+	$K_C * de/dt$
Регулирующее воздействие	=	Пропорциональная часть	+	Интегральная часть	+	Дифференциальная часть

где:

- $M(t)$ - регулирующее воздействие (выход регулятора) как функция времени
- K_C - коэффициент усиления контура регулирования
- e - ошибка регулирования (разность между заданным значением и регулируемой переменной)
- $M_{\text{нач}}$ - начальное значение регулирующего воздействия

Чтобы реализовать эту функцию управления в цифровой вычислительной машине, должно быть выполнено квантование непрерывной функции в соответствии с периодическими замерами значения ошибки с последующим расчетом регулирующего воздействия. Соответствующее уравнение, являющееся основой для решения на цифровой вычислительной машине, имеет вид:

M_n	=	$K_C * e_n$	+	$K_I * \sum_{i=1}^n e_n + M_{\text{нач}}$	+	$K_D * (e_n - e_{n-1})$
Регулирующее воздействие	=	Пропорциональная часть	+	Интегральная часть	+	Дифференциальная часть

где:

- M_n - расчетное значение регулирующего воздействия в момент квантования n
- K_C - коэффициент усиления контура регулирования
- e_n - значение ошибки регулирования в момент квантования n
- e_{n-1} - предыдущее значение ошибки регулирования (в момент квантования $n - 1$)
- K_I - коэффициент пропорциональности интегральной составляющей
- $M_{\text{нач}}$ - начальное значение регулирующего воздействия
- K_D - коэффициент пропорциональности дифференциальной составляющей

Из этого уравнения следует, что интегральная составляющая является функцией всех составляющих ошибки от первого до текущего отсчета. Дифференциальная составляющая является функцией текущего и предыдущего отсчета, тогда как пропорциональная составляющая является функцией только текущего отсчета. В цифровой вычислительной машине нецелесообразно хранить все отсчеты ошибки регулирования, да в этом и нет необходимости.

Так как компьютер должен вычислять регулирующее воздействие каждый раз, как опрашивается значение ошибки, начиная с первого отсчета, то необходимо сохранять только предыдущее значение ошибки и предыдущее значение интегральной составляющей. Как результат повторяющейся природы компьютерного решения, может быть получено упрощение уравнения, подлежащего решению в каждый момент квантования. Упрощенное уравнение имеет вид:

M_n	=	$K_c * e_n$	+	$K_i * e_n + MX$	+	$K_D * (e_n - e_{n-1})$
Регулирующее воздействие	=	Пропорциональная часть	+	Интегральная часть	+	Дифференциальная часть

где:

- M_n - расчетное значение регулирующего воздействия в момент квантования n
- K_c - коэффициент усиления контура регулирования
- e_n - значение ошибки регулирования в момент квантования n
- e_{n-1} - предыдущее значение ошибки регулирования (в момент квантования $n - 1$)
- K_i коэффициент пропорциональности интегральной составляющей
- MX - предыдущее значение интегральной составляющей (в момент квантования $n - 1$)
- K_D - коэффициент пропорциональности дифференциальной составляющей

CPU использует модифицированную форму упрощенного выше уравнения при расчете регулирующего воздействия. Это модифицированное уравнение имеет вид:

M_n	=	MP_n	+	MI_n	+	MD_n
Регулирующее воздействие	=	Пропорциональная часть	+	Интегральная часть	+	Дифференциальная часть

где:

- M_n - расчетное значение регулирующего воздействия в момент квантования n
- MP_n - значение пропорциональной составляющей регулирующего воздействия в момент квантования n
- MI_n - значение интегральной составляющей регулирующего воздействия в момент квантования n
- MD_n - значение дифференциальной составляющей регулирующего воздействия в момент квантования n

Пропорциональная составляющая

Пропорциональная составляющая MP – это произведение коэффициента усиления (K_C), определяющего точность расчета регулирующего воздействия, и ошибки регулирования (e), представляющей собой разность между заданным значением (SP) и регулируемой переменной (PV) в данный момент квантования. Уравнение для пропорциональной составляющей, решаемое CPU, имеет вид:

$$MP_n = K_C * (SP_n - PV_n)$$

где:

- MP_n - значение пропорциональной составляющей регулирующего воздействия в момент квантования n
- K_C - коэффициент усиления контура регулирования
- SP_n - заданное значение регулируемой величины в момент квантования n
- PV_n - значение регулируемой переменной в момент квантования n

Интегральная составляющая

Интегральная составляющая MI пропорциональна сумме ошибок за все время управления. Уравнение для интегральной составляющей, решаемое CPU, имеет вид:

$$MI_n = K_C * T_S / T_I * (SP_n - PV_n) + MX$$

где:

- MI_n - значение интегральной составляющей регулирующего воздействия в момент квантования n
- K_C - коэффициент усиления контура регулирования
- T_S - период квантования контура регулирования
- T_I - постоянная времени интегрирования контура регулирования (называемая также сбросом)
- SP_n - заданное значение регулируемой величины в момент квантования n
- PV_n - значение регулируемой переменной в момент квантования n
- MX - значение интегральной составляющей в момент квантования n – 1 (называемое также интегральной суммой или смещением)

Интегральная сумма или смещение (MX) – это текущая сумма всех предыдущих значений интегральной составляющей. После каждого расчета MI_n смещение обновляется значением MI_n , которое может быть согласовано или ограничено (подробности см. в разделе «Переменные и диапазоны» на стр. 9–97). Начальное значение смещения обычно устанавливается равным значению регулирующего воздействия ($M_{initial}$) сразу перед его первым расчетом. Частью интегральной составляющей являются также несколько констант: коэффициент усиления (K_C), период квантования (T_S), представляющий собой время цикла, с которым PID-регулятор пересчитывает регулирующее воздействие, и постоянная времени интегрирования (или сброс) (T_I), которая используется для управления влиянием интегральной составляющей на расчет регулирующего воздействия.

Дифференциальная составляющая

Дифференциальная составляющая MD пропорциональна изменению ошибки регулирования. Ниже показано уравнение для расчета дифференциальной составляющей:

$$MD_n = K_C * T_D / T_S * ((SP_n - PV_n) - (SP_{n-1} - PV_{n-1}))$$

Во избежание ступенчатых изменений или скачков регулирующего воздействия при изменениях заданного значения это уравнение модифицировано в предположении, что заданное значение постоянно ($SP_n = SP_{n-1}$). В результате рассчитывается изменение регулируемой переменной, а не изменение ошибки регулирования. Это показывает следующее уравнение:

$$MD_n = K_C * T_D / T_S * (SP_n - PV_n - SP_n + PV_{n-1})$$

или:

$$MD_n = K_C * T_D / T_S * (PV_{n-1} - PV_n)$$

где:

- MD_n - значение дифференциальной составляющей регулирующего воздействия в момент квантования n
- K_C - коэффициент усиления контура регулирования
- T_S - период квантования контура регулирования
- T_D - постоянная времени воздействия по производной контура регулирования (называемая также временем упреждения)
- SP_n - заданное значение регулируемой величины в момент квантования n
- SP_{n-1} - заданное значение регулируемой величины в момент квантования n – 1
- PV_n - значение регулируемой переменной в момент квантования n
- PV_{n-1} - значение регулируемой переменной в момент квантования n – 1

Для использования в следующих расчетах дифференциальной составляющей должна сохраняться регулируемая переменная, а не ошибка регулирования. При первом отсчете значение PV_{n-1} инициализируется равным PV_n .

Выбор регулятора

Во многих системах управления может оказаться необходимым использовать только один или два метода регулирования. Например, может потребоваться только пропорциональное или пропорционально-интегральное управление. Выбор желаемого типа регулятора выполняется установкой значений постоянных параметров.

Если вам не нужно интегральное воздействие (нет составляющей "I" в расчете PID-регулятора), то постоянная времени интегрирования должна быть задана равной бесконечности. Даже при отсутствии интегрального воздействия значение этой составляющей не может быть равно нулю из-за начального значения интегральной суммы MX.

Если вам не нужно дифференцирующее воздействие (нет составляющей "D" в расчете PID-регулятора), то постоянная времени воздействия по производной (упреждение) должна быть задана равной 0.0.

Если вам не нужно пропорциональное воздействие (нет составляющей "P" в расчете PID-регулятора), а вы хотите иметь интегральный или интегрально-дифференциальный регулятор, то значение 0.0 должно быть задано для коэффициента усиления. Так как усиление является коэффициентом в уравнениях для расчета интегральной и дифференциальной составляющей, установка значения 0.0 для коэффициента усиления контура регулирования приведет к тому, что при расчете интегральной и дифференциальной составляющей для коэффициента усиления будет использовано значение 1.0.

Преобразование и нормализация входов контура регулирования

Контур регулирования имеет две входных переменных – заданное значение и регулируемую переменную. Заданное значение – это обычно фиксированная величина, например, уставка скорости для регулятора скорости движения в вашем автомобиле. Регулируемая переменная – это величина, связанная с регулирующим воздействием, поэтому она измеряет влияние, оказываемое регулирующим воздействием на управляемую систему. В примере с регулятором скорости движения автомобиля регулируемой переменной является вход тахометра, измеряющего скорость вращения колес.

Заданное значение и регулируемая переменная – это реальные физические величины, диапазон значений которых и единицы измерения могут быть различными. Прежде чем эти физические величины могут быть использованы командой PID-регулятор, они должны быть преобразованы в нормализованные представления с плавающей точкой.

Первый шаг состоит в преобразовании физической величины, представленной в виде 16-битового целого числа в вещественное число, или число с плавающей точкой. Следующая последовательность команд показывает, как можно преобразовать целое число в вещественное.

XORD	AC0, AC0	//Очистить аккумулятор.
MOVW	AIW0, AC0	//Сохранить в аккумуляторе аналоговое значение.
LDW>=	AC0, 0	//Если аналоговое значение положительно,
JMP	0	//то преобразовать его в вещественное число.
NOT		//В противном случае
ORD	16#FFFF0000, AC0	//снабдить значение в AC0 знаком.
LBL	0	
DTR	AC0, AC0	//Преобразовать 32-битовое целое число в вещественное.

Следующий шаг состоит в преобразовании вещественного представления реальной физической величины в нормализованное значение между 0.0 и 1.0. Для нормализации заданного значения или регулируемой переменной используется следующее уравнение:

$$R_{\text{Norm}} = ((R_{\text{Raw}} / \text{Span}) + \text{Offset})$$

где:

R_{Norm}	- нормализованное вещественное число, представляющее реальную физическую величину
R_{Raw}	- ненормализованное, или необработанное (raw), вещественное представление реальной физической величины
Offset	- смещение, равное 0.0 для униполярных величин, и 0,5 для биполярных величин
Span и	- диапазон, равный разности между максимально возможным и минимально возможным значением = 32 000 для униполярных величин (типичное значение) = 64 000 для биполярных величин (типичное значение)

Следующая последовательность команд, являющаяся продолжением предыдущей последовательности, показывает, как нормализовать биполярную величину в AC0 (с диапазоном 64 000):

```

/R      64000.0, AC0      //Нормализовать значение в
аккумуляторе
+R      0.5, AC0          //Смещение для величины в диапазоне
от
                                //0.0 до 1.0
MOVR    AC0, VD100        //Сохранить нормализованное значение в
                                //таблице контура регулирования TABLE

```

Преобразование регулирующего воздействия в масштабированную целую величину

Регулирующее воздействие – это управляющая переменная, например, установка дроссельной заслонки в примере с регулятором скорости движения автомобиля. Регулирующее воздействие – это нормализованное вещественное значение между 0.0 и 1.0. Прежде чем регулирующее воздействие может быть использовано для управления аналоговым выходом, оно должно быть преобразовано в 16-битовую масштабированную целую величину. Этот процесс противоположен преобразованию PV и SP в нормализованную величину. Первый шаг состоит в преобразовании регулирующего воздействия в масштабированное вещественное число с помощью следующей формулы:

$$R_{\text{Scal}} = (M_n - \text{Offset}) * \text{Span}$$

где:

R_{Scal} - масштабированное (scaled) вещественное значение регулирующего воздействия
 M_n - нормализованное вещественное значение регулирующего воздействия
Offset - смещение, равное 0.0 для униполярных величин, и 0,5 для биполярных величин
Span и - диапазон, равный разности между максимально возможным и минимально возможным значением
= 32 000 для униполярных величин (типичное значение)
= 64 000 для биполярных величин (типичное значение)

Следующая последовательность команд показывает, как масштабировать регулирующее воздействие:

```

MOVR    VD108, AC0      //переместить регулирующее воздействие в
                                //аккумулятор.
-R      0.5, AC0          //Этот оператор включить только для
биполярного                                //значения.
*R      64000.0, AC0      //Масштабировать значение в аккумуляторе.

```

Затем масштабированное вещественное представление регулирующего воздействия должно быть преобразовано в 16-битовое целое. Как выполнить это преобразование, показывает следующая последовательность команд:

```

ROUND   AC0  AC0      //Преобразовать вещественное число в 32-
битовое
                                //целое.
MOVW    AC0, AQW0      //Записать 16-битовое целое число в
аналоговый                //выход.

```


Контуры регулирования с положительной и отрицательной обратной связью

Контур регулирования имеет положительную обратную связь, если его коэффициент усиления положителен, и отрицательную обратную связь, если его коэффициент усиления отрицателен. (Для интегрального и интегрально-дифференциального регулятора, где значение коэффициента усиления равно 0.0, задание положительных значений для постоянных времени интегрирования и воздействия по производной приведет к положительной обратной связи, а задание отрицательных значений – к отрицательной обратной связи.)

Переменные и диапазоны

Регулируемая переменная и заданное значение являются входами для расчета PID-регулятора. Поэтому поля таблицы контура регулирования для этих переменных могут считываться, но не могут быть изменены командой PID-регулятор.

Регулирующее воздействие генерируется как результат расчетов PID-регулятора, так что поле регулирующего воздействия в таблице контура регулирования обновляется после каждого расчета PID-регулятора. Регулирующее воздействие заключено между 0.0 и 1.0. Поле регулирующего воздействия может быть использовано для задания начального регулирующего воздействия при организации перехода от ручного управления к автоматическому с помощью команды PID-регулятор (см. обсуждение этого вопроса в нижеследующем разделе «Режимы»).

Если используется интегральный регулятор, то значение смещения обновляется в результате расчета PID-регулятора, и обновленное значение используется как вход в следующем расчете PID-регулятора. Если рассчитанное регулирующее воздействие выходит за пределы допустимого диапазона (меньше 0.0 или больше 1.0), то смещение корректируется в соответствии со следующими формулами:

$$MX = 1.0 - (MP_n + MD_n), \quad \text{если расчетное значение } M_n > 1.0$$

или

$$MX = - (MP_n + MD_n), \quad \text{если расчетное значение } M_n < 0.0,$$

где:

- MX - значение скорректированного смещения
- MP_n - значение пропорциональной составляющей регулирующего воздействия в момент квантования n
- MD_n - значение дифференциальной составляющей регулирующего воздействия в момент квантования n
- M_n - значение регулирующего воздействия в момент квантования n

При корректировке смещения описанным выше способом достигается улучшение чувствительности системы, когда рассчитанное регулирующее воздействие возвращается в надлежащий диапазон. Рассчитанное смещение также устанавливается в диапазоне между 0.0 и 1.0, а затем записывается в поле смещения таблицы контура регулирования при завершении каждого расчета PID-регулятора. Значение, сохраняемое в таблице контура регулирования, используется в следующем цикле расчетов PID-регулятора.

Значение смещения в таблице контура регулирования может быть изменено пользователем перед выполнением команды PID-регулятор для оказания воздействия на определенные ситуации в некоторых приложениях. При ручной корректировке смещения необходимо учитывать, что любое смещение, записываемое в таблицу контура регулирования, должно быть вещественным числом в диапазоне между 0.0 и 1.0.

Значение регулируемой величины, используемое для сравнения при расчете дифференциальной составляющей PID-регулятора, сохраняется в таблице контура регулирования. Это значение не может быть изменено пользователем.

Режимы

Для PID-регуляторов S7-200 нет встроенного управления режимом работы. Расчет PID-регулятора выполняется только тогда, когда поток сигнала достигает блока PID. Поэтому «автоматический» режим имеет место, когда расчеты PID-регулятора выполняются циклически. «Ручной» режим имеет место, когда расчеты PID-регулятора не выполняются.

Команда PID-регулятор имеет бит истории потока сигнала, аналогичный биту, используемому в команде счета. Команда использует этот бит для обнаружения изменения потока сигнала с 0 на 1. Когда это изменение потока сигнала обнаруживается, это заставляет команду выполнить ряд действий, обеспечивающих плавный переход от ручного управления к автоматическому. Для плавного перехода к автоматическому режиму управления значение регулирующего воздействия, установленное при ручном управлении, должно быть передано в качестве входа команде PID-регулятор (записано в таблицу контура регулирования для M_n) перед переключением в автоматический режим управления. Команда PID-регулятор выполняет следующие действия с величинами в таблице контура регулирования, чтобы обеспечить плавный переход от ручного режима управления к автоматическому при обнаружении перехода потока сигнала из 0 в 1:

- устанавливает заданное значение (SP_n) = регулируемой переменной (PV_n)
- устанавливает старое значение регулируемой переменной (PV_{n-1}) = регулируемой переменной (PV_n)
- регулируемой смещение (MX) = регулирующему воздействию (M_n)

По умолчанию бит истории PID-регулятора установлен, и это состояние формируется при запуске CPU и при каждом переходе контроллера из STOP в RUN. Если сигнал достигает блока PID при первом его исполнении после перехода в режим RUN, то переход потока сигнала из 0 в 1 не обнаруживается, и действия, обеспечивающие плавный переход, не выполняются.

Аварийный контроль и специальные операции

PID-регулятор – это простая, но мощная команда, выполняющая расчеты, необходимые для PID-регулирования. Если требуются другие функции, например, контроль аварийных ситуаций или выполнение специальных расчетов с переменными контура регулирования, то они должны быть реализованы с помощью основных команд, поддерживаемых CPU.

Сбойные ситуации

Если начальный адрес таблицы контура регулирования или операнды контура регулирования, указанные в команде, находятся вне допустимого диапазона, то при компиляции CPU выдаст ошибку компиляции (ошибка диапазона), и компиляция потерпит неудачу.

Некоторые входные значения таблицы контура регулирования не проверяются на соответствие допустимому диапазону командой PID-регулятор. Вы должны позаботиться о том, чтобы регулируемая переменная и заданное значение (а также смещение и предыдущее значение регулируемой переменной, если они используются в качестве входов) были вещественными числами в диапазоне между 0.0 и 1.0.

Если при выполнении арифметических операций в расчетах PID-регулятора встретится любая ошибка, то будет установлен бит SM1.1 (переполнение или недопустимое значение), и выполнение команды PID-регулятор будет завершено. (Обновление выходных значений в таблице контура регулирования может быть неполным, так что вы не должны принимать эти значения во внимание, а исправить входное значение, вызвавшее арифметическую ошибку перед следующим выполнением команды PID-регулятор.)

Таблица контура регулирования

Таблица контура регулирования имеет длину 36 байтов и формат, показанный в таблице 9–19.

Таблица 9–19. Формат таблицы контура регулирования

Смещение	Поле	Формат	Тип	Описание
0	Регулируемая переменная (PV_n)	Двойное слово – вещественная	in	Содержит регулируемую переменную, которая должна быть между 0.0 и 1.0.
4	Заданное значение (SP_n)	Двойное слово – вещественная	in	Содержит заданное значение, которое должна быть между 0.0 и 1.0.
8	Регулирующее воздействие (M_n)	Двойное слово – вещественная	in/out	Содержит расчетное регулирующее воздействие, которое должна быть между 0.0 и 1.0.
12	Коэффициент усиления (K_C)	Двойное слово – вещественная	in	Содержит усиление, являющееся коэффициентом пропорциональности. Может быть положительным или отрицательным числом.
16	Период квантования (T_s)	Двойное слово – вещественная	in	Содержит период квантования в секундах. Должен быть положительным числом.
20	Постоянная времени интегрирования (T_I)	Двойное слово – вещественная	in	Содержит постоянную времени интегрирования в минутах. Должно быть положительным числом.
24	Постоянная времени воздействия по производной (T_D)	Двойное слово – вещественная	in	Содержит постоянную времени воздействия по производной в минутах. Должно быть положительным числом.
28	Смещение (MX)	Двойное слово – вещественная	in/out	Содержит значение смещения или интегральной суммы в пределах от 0.0 до 1.0.
32	Предыдущее значение регулируемой переменной (PV_{n-1})	Двойное слово – вещественная	in/out	Содержит предыдущее значение регулируемой переменной, сохраняемое после последнего выполнения команды PID-регулятор.

Пример программы с PID-регулятором

В этом примере в резервуаре с водой должно поддерживаться постоянное давление воды. Вода постоянно забирается из резервуара с различной скоростью. Для добавления воды в резервуар со скоростью, обеспечивающей надлежащее давление воды, а также не допускающей опустошения резервуара, используется насос, снабженный приводом с переменной скоростью.

Заданным значением для этой системы является установка уровня воды, эквивалентная наполнению резервуара на 75%. Регулируемая переменная поставляется поплавковым измерительным устройством, показывающим, насколько полон резервуар. Выдаваемое им значение может находиться в пределах от 0% при пустом резервуаре до 100% при полном резервуаре. Регулирующим воздействием является величина скорости насоса, которая может меняться от 0% до 100% от максимальной скорости.

Заданное значение определяется заранее и непосредственно вводится в таблицу контура регулирования. Регулируемая переменная поставляется как однополярная аналоговая величина из поплавкового измерительного устройства. Регулирующее воздействие записывается как однополярная аналоговая величина, используемая для управления скоростью насоса. Диапазон аналогового входа и аналогового выхода равен 32 000.

В этом примере используется только пропорциональный и интегральный регулятор. Коэффициент усиления контура регулирования и постоянные времени определены из инженерных расчетов и могут быть настроены для достижения оптимального управления. Рассчитанные значения коэффициента усиления и постоянных времени:

$$K_C = 0,25$$

$$T_S = 0,1 \text{ секунды}$$

$$T_I = 30 \text{ минут}$$

Скорость насоса управляется вручную, пока резервуар не будет заполнен на 75%, затем открывается клапан, позволяющий воде вытекать из резервуара. Одновременно насос переключается из ручного режима в автоматический. Для переключения из ручного режима в автоматический используется цифровой вход. Этот вход описан ниже:

I0.0 – ручной/автоматический режим; 0 – ручной, 1 – автоматический.

В ручном режиме скорость насоса записывается оператором в VD108 как вещественное число в диапазоне от 0.0 до 1.0.

Программа управления для этого приложения показана на рис. 9–28.


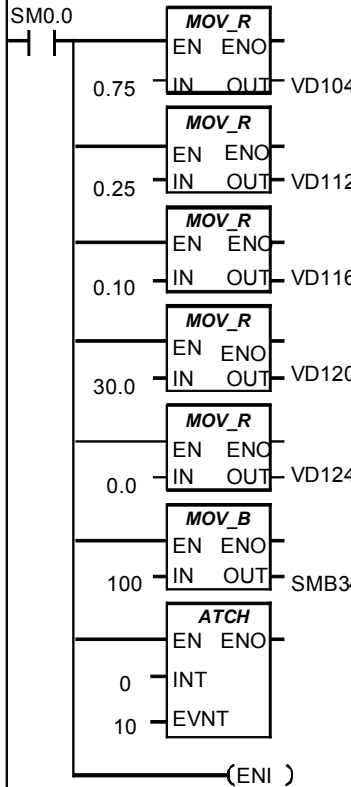
LAD	STL
MAIN [Главная программа]	
OB1 Network 1 	Network 1 LD SM0.1 //В первом цикле вызвать CALL 0 //подпрограмму //инициализации
SUBROUTINE 0 [Подпрограмма 0]	
Network 1 	Network 1 LD SM0.0 MOVR 0.75, VD104 //Загрузить заданное // значение = 75% емкости резервуара. MOVR 0.25, VD112 //Загрузить коэффициент //усиления = 0,25. MOVR 0.10, VD116 //Загрузить период // квантования = 0,1 секунды. MOVR 30.0, VD120 //Загрузить постоянную //времени интегрирования = 30 минут. // MOVR 0.0, VD124 //Исключить действие по // по производной. MOVB 100, SMB34 //Установить интервал //(100 мс) для управляемого временем //прерывания 0. ATCH 0, 10 //Организовать управляемое // временем прерывание для вызова на //исполнение PID-регулятора. ENI //Разрешить прерывания. //Конец подпрограммы 0.

Рис. 9-28. Пример PID-регулятора для SIMATIC LAD, STL и FBD

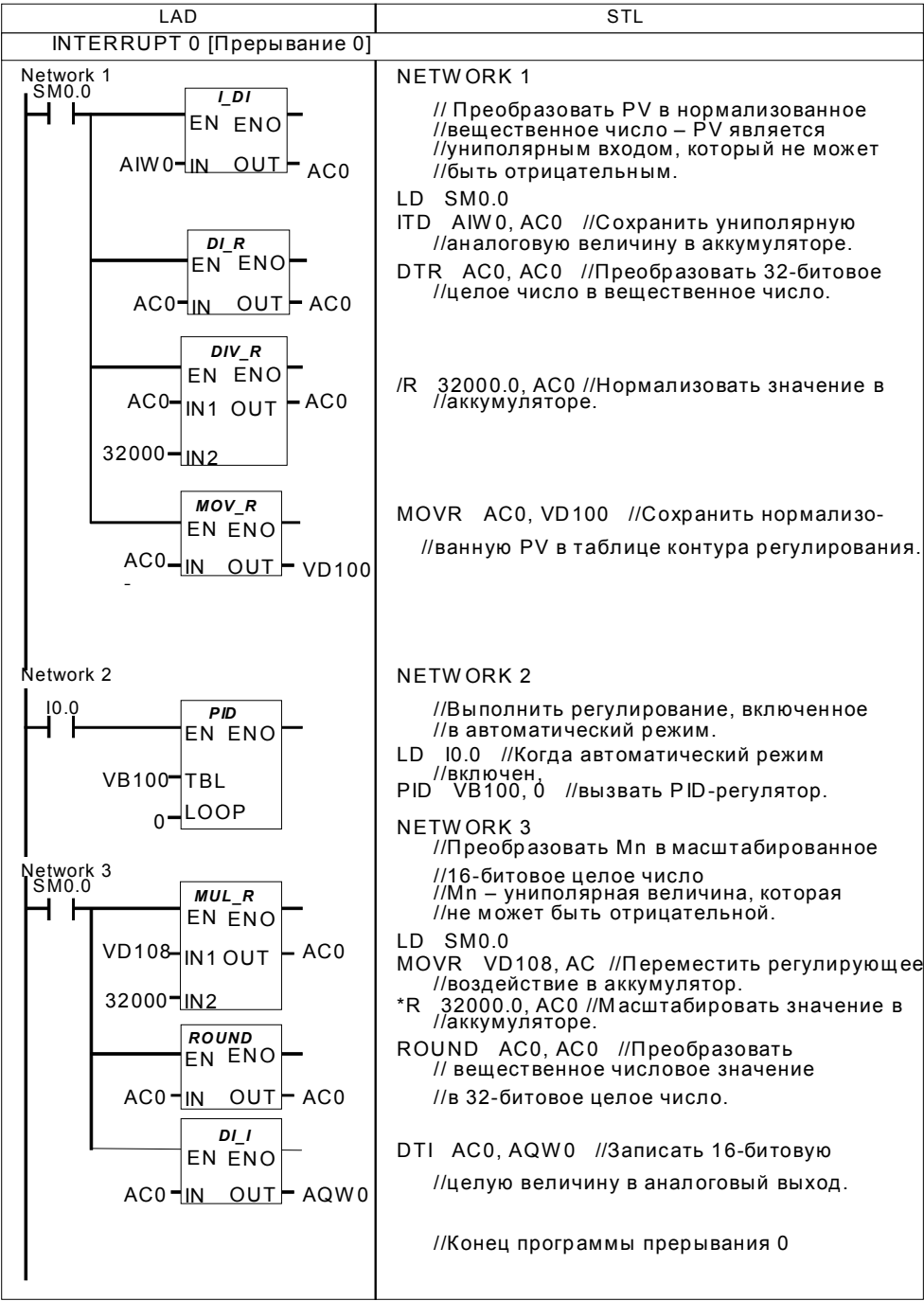


Рис. 9-28. Пример PID-регулятора для SIMATIC LAD, STL и FBD (продолжение)

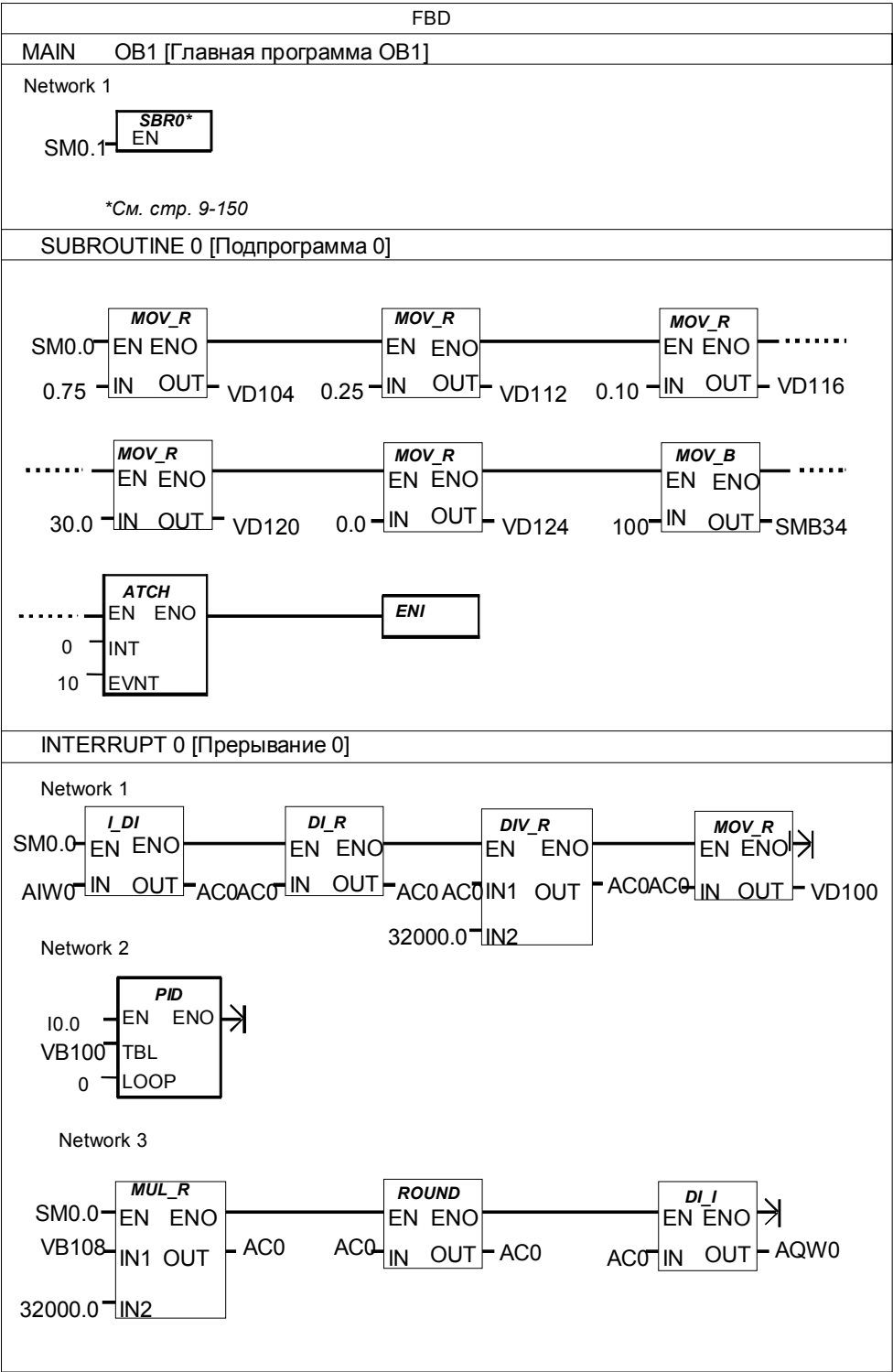
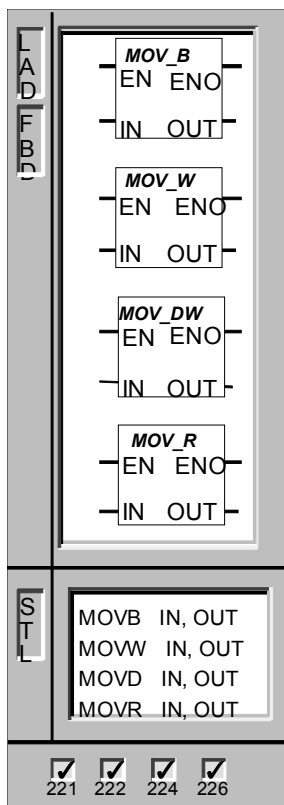


Рис. 9-28. Пример PID-регулятора для SIMATIC LAD, STL и FBD (продолжение)

9.11 Команды SIMATIC для пересылки

Пересылка байта, слова, двойного слова, вещественного числа



Команда **Переслать байт** пересылает входной байт (IN) в выходной байт (OUT). Входной байт командой пересылки не изменяется.

Команда **Переслать слово** пересылает входное слово (IN) в выходное слово (OUT). Входное слово командой пересылки не изменяется.

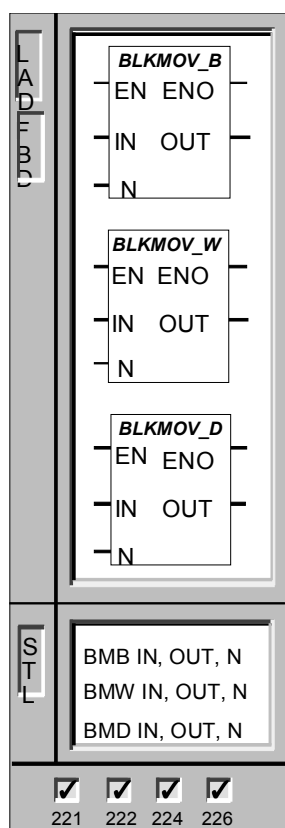
Команда **Переслать двойное слово** пересылает входное двойное слово (IN) в выходное двойное слово (OUT). Входное двойное слово командой пересылки не изменяется.

Команда **Переслать вещественное число** пересылает входное вещественное число (двойное слово, 32 бита) (IN) в выходное (OUT). Входное двойное слово командой пересылки не изменяется.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Переслать ...	Входы/выходы	Операнды	Типы данных
байт	IN	VB, IB, QB, MB, SB, SMB, LB, AC, константа, *VD, *AC, *LD	BYTE
	OUT	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD	BYTE
слово	IN	VW, IW, QW, MW, SW, SMW, LW, T, C, AIW, константа, AC *VD, *AC, *LD	WORD, INT
	OUT	VW, T, C, IW, QW, SW, MW, SMW, LW, AC, AQW, *VD, *AC, *LD	WORD, INT
двойное слово	IN	VD, ID, QD, MD, SD, SMD, LD, HC, &VB, &IB, &QB, &MB, &SB, &T, &C, AC, константа, *VD, *AC, *LD	DWORD, DINT
	OUT	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD	DWORD, DINT
Вещественное число	IN	VD, ID, QD, MD, SD, SMD, LD, AC, константа, *VD, *AC, *LD	REAL
	OUT	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD	REAL

Групповая пересылка байтов, слов, двойных слов



Команда **Переслать группу байтов** пересылает несколько байтов (N) из входного адреса IN в выходной адрес OUT. N может иметь значение от 1 до 255.

Команда **Переслать группу слов** пересылает несколько слов (N) из входного адреса IN в выходной адрес OUT. N может иметь значение от 1 до 255.

Команда **Переслать группу двойных слов** пересылает несколько двойных слов (N) из входного адреса IN в выходной адрес OUT. N может иметь значение от 1 до 255.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация), 0091 (выход операнда за пределы допустимого диапазона)

Переслать несколько...	Входы/выходы	Операнды	Типы данных
Байтов	IN, OUT	VB, IB, QB, MB, SB, SMB, LB, *VD, *AC, *LD	BYTE
	N	VB, IB, QB, MB, SB, SMB, LB, AC, константа, *VD, *AC, *LD	BYTE
слов	IN	VW, IW, QW, MW, SW, SMW, LW, T, C, AIW, *VD, *AC, *LD	WORD
	N	VB, IB, QB, MB, SB, SMB, LB, AC, константа, *VD, *AC, *LD	BYTE
	OUT	VW, IW, QW, MW, SW, SMW, LW, T, C, AQW, *VD, *LD, *AC	WORD
двойных слов	IN, OUT	VD, ID, QD, MD, SD, SMD, LD, *VD, *AC, *LD	DWORD
	N	VB, IB, QB, MB, SB, SMB, LB, AC, константа, *VD, *AC, *LD	BYTE

Пример групповой пересылки

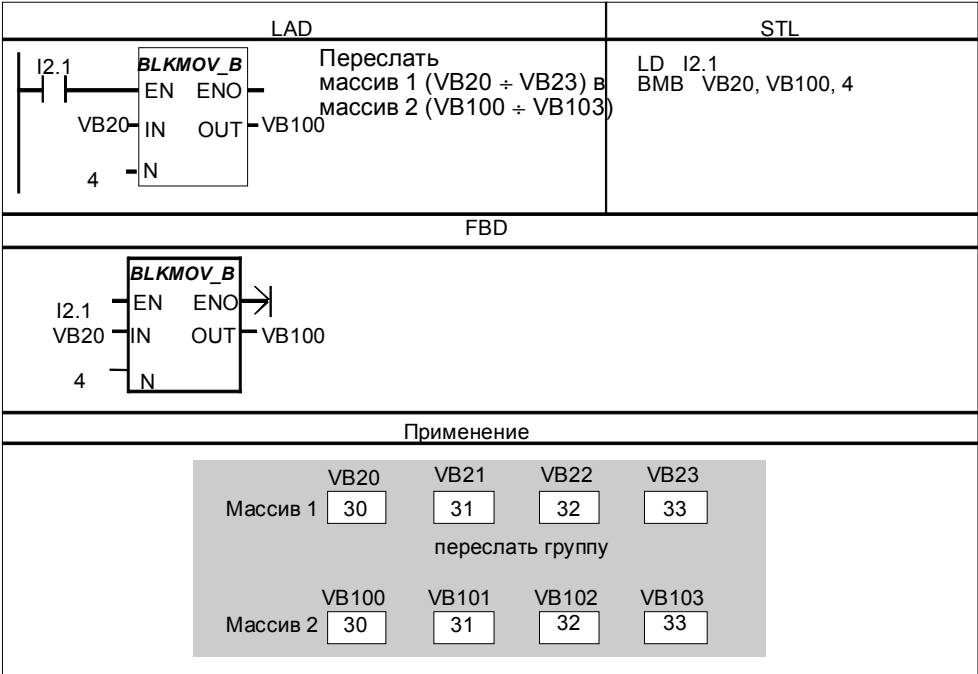
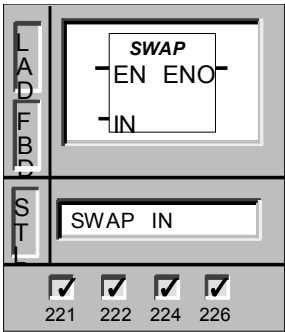


Рис. 9-29. Пример команд групповой пересылки для SIMATIC LAD, STL и FBD

Перестановка байтов



Команда **Переставить байты** меняет местами старший и младший байты слова (IN).
Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Входы/выходы	Операнды	Типы данных
IN	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, *VD, *AC, *LD	WORD

Примеры пересылки и перестановки

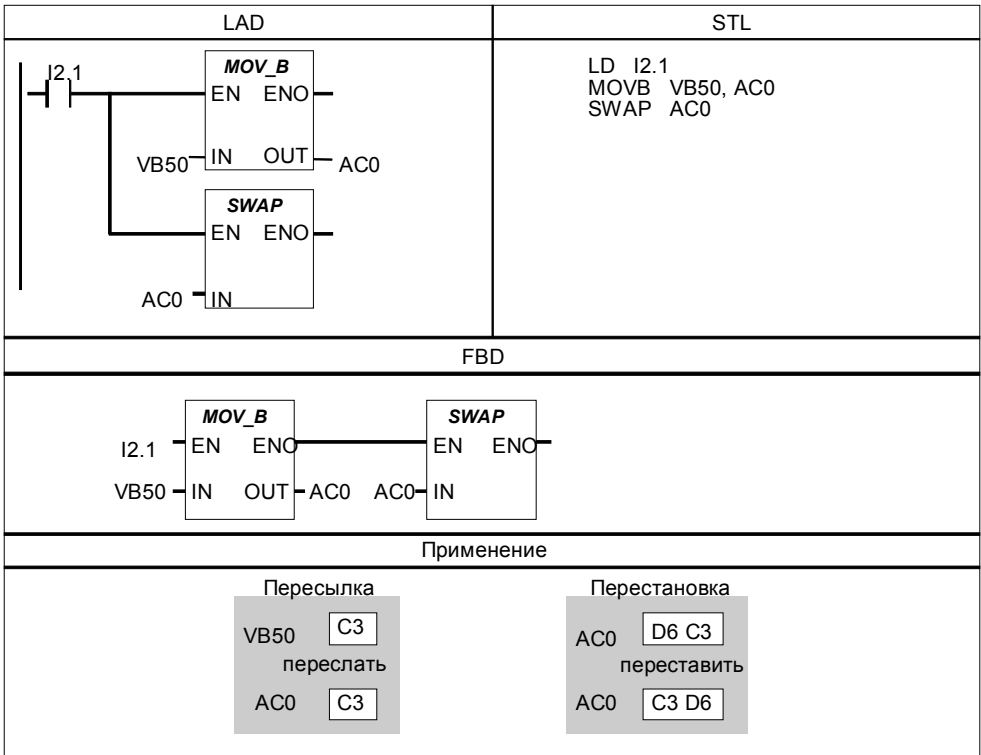
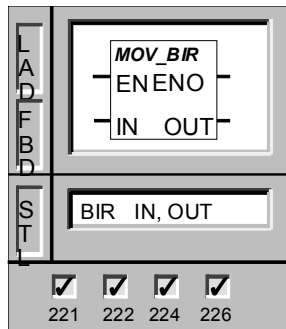


Рис. 9-30. Пример команд пересылки и перестановки для SIMATIC LAD, STL и FBD

Пересылка непосредственно считанного байта

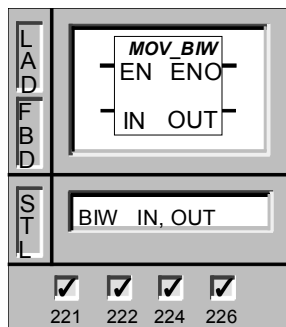


Команда **Переслать непосредственно считанный байт** считывает физический вход IN и записывает результат в OUT.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Входы/выходы	Операнды	Типы данных
IN	IB	BYTE
OUT	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *LD, *AC	BYTE

Пересылка байта для непосредственной записи



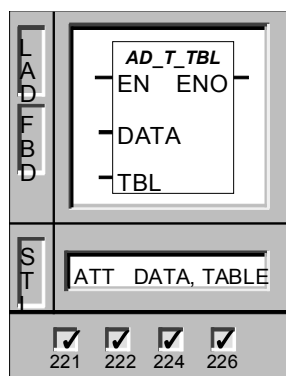
Команда **Переслать байт для непосредственной записи** считывает байт из ячейки IN и записывает в физический выход OUT.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Входы/выходы	Операнды	Типы данных
IN	VB, IB, QB, MB, SB, SMB, LB, AC, константа, *VD, *LD, *AC	BYTE
OUT	QB	BYTE

9.12 Табличные команды SIMATIC

Добавление данных к таблице



Команда **Добавить данные к таблице** добавляет к таблице (TBL) значения, имеющие размер слова (DATA).

Первым значением таблицы является ее максимальная длина (table length, TL). Второе значение – это количество записей (entry count, EC) в таблице. (См. рис. 9–31.) Новые данные добавляются к таблице после последней записи. Каждый раз, когда к таблице добавляются новые данные, количество записей увеличивается на единицу. Таблица может иметь до 100 записей данных (TBL).

Ошибки, устанавливающие ENO в 0: SM1.4 (переполнение таблицы), SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация), 0091 (операнд вне допустимого диапазона)

Эта команда влияет на следующие биты специальной памяти: SM1.4 устанавливается в 1, если вы пытаетесь переполнить таблицу.

Входы/выходы	Операнды	Типы данных
DATA	VW, IW, QW, MW, SW, SMW, LW, T, C, AIW, AC, константа, *VD, *AC, *LD	INT
TBL	VW, IW, QW, MW, SW, SMW, LW, T, C, *VD, *AC, *LD	WORD

Пример добавления данных к таблице

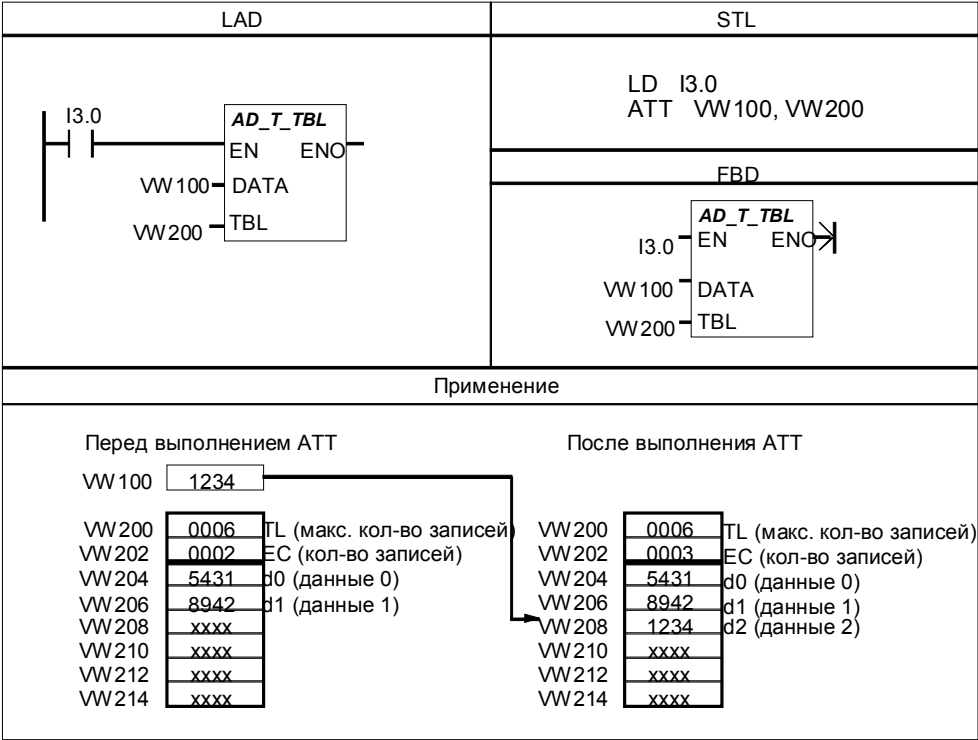
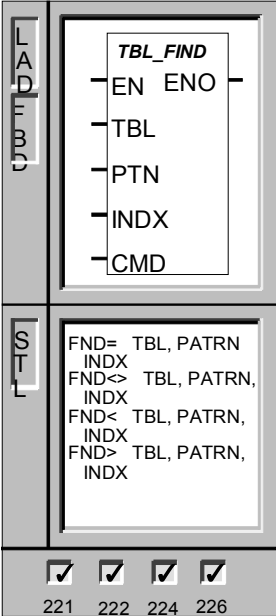


Рис. 9-31. Пример команды «Добавить данные к таблице» для SIMATIC LAD, STL и FBD

Поиск в таблице



Команда **Поиск в таблице** ищет в таблице (TBL), начиная с записи INDX и основываясь на образце (PTN), значение данных, удовлетворяющее критерию поиска, определенному CMD. Командному параметру CMD дается числовое значение от 1 до 4, что соответственно означает =, <>, < и >.

Если соответствующая запись, найдена, то INDX указывает на эту запись в таблице. Для нахождения следующей записи INDX должен быть увеличен, прежде чем команда Поиск в таблице будет вызвана снова. Если запись, удовлетворяющая условию поиска, не найдена, то INDX имеет значение, равное количеству записей.

Таблица может иметь до 100 записей данных. Записи данных (область поиска) нумеруются от 0 до максимум 99.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация), 0091 (операнд вне допустимого диапазона)

Входы/выходы	Операнды	Типы данных
SRC	VW, IW, QW, MW, SMW, LW, T, C, *VD, *AC, *LD	WORD
PTN	VW, IW, QW, MW, SW, SMW, AIW, LW, T, C, AC, константа, *VD, *AC, *LD	INT
INDX	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, *VD, *AC, *LD	WORD
CMD	константа	BYTE

Примечание

Когда вы используете команды поиска в таблицах, созданных с помощью команд ATT, LIFO и FIFO, то количество записей и сами записи данных согласуются непосредственно. Слово, определяющее максимальное количество записей, требуемое для команд ATT, LIFO и FIFO, не требуется для команд поиска. Поэтому операнд SRC команды поиска находится, как показано на рис. 9-32, на один адрес слова (два байта) выше, чем операнд TBL соответствующей команды ATT, LIFO или FIFO.

Формат таблицы для ATT, LIFO и FIFO			Формат таблицы для TBL_FIND		
VW200	0006	TL (макс. кол-во записей)	VW202	0006	EC (число записей)
VW202	0006	EC (число записей)	VW204	xxxx	d0 (данные 0)
VW204	xxxx	d0 (данные 0)	VW206	xxxx	d1 (данные 1)
VW206	xxxx	d1 (данные 1)	VW208	xxxx	d2 (данные 2)
VW208	xxxx	d2 (данные 2)	VW210	xxxx	d3 (данные 3)
VW210	xxxx	d3 (данные 3)	VW212	xxxx	d4 (данные 4)
VW212	xxxx	d4 (данные 4)	VW214	xxxx	d5 (данные 5)
VW214	xxxx	d5 (данные 5)			

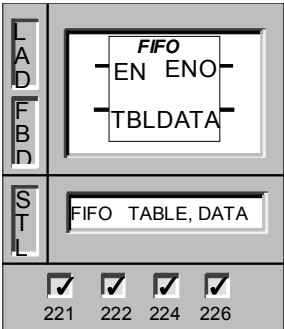
Рис. 9-32. Разница в формате таблицы между командами поиска и ATT, LIFO, FIFO

Пример поиска в таблице

LAD		STL														
		<pre>LD I2.1 FND= VW202, 16#3130, AC1</pre>														
<p>Когда I2.1 включен, искать в таблице значение, равное 3130 HEX.</p>																
<p>Применение</p>																
<p>Эта таблица, в которой производится поиск. Если таблица была создана с помощью команды ATT, LIFO или FIFO, то VW200 содержит максимально разрешенное количество записей и не требуется для команд поиска.</p>																
	<table border="1"><tr><td>VW202</td><td>0006</td></tr><tr><td>VW204</td><td>3133</td></tr><tr><td>VW206</td><td>4142</td></tr><tr><td>VW208</td><td>3130</td></tr><tr><td>VW210</td><td>3030</td></tr><tr><td>VW212</td><td>3130</td></tr><tr><td>VW214</td><td>4541</td></tr></table>	VW202	0006	VW204	3133	VW206	4142	VW208	3130	VW210	3030	VW212	3130	VW214	4541	<p>ЕС (число записей) d0 (данные 0) d1 (данные 1) d2 (данные 2) d3 (данные 3) d4 (данные 4) d5 (данные 5)</p>
VW202	0006															
VW204	3133															
VW206	4142															
VW208	3130															
VW210	3030															
VW212	3130															
VW214	4541															
AC1 <input type="text" value="0"/>	AC1 должен быть установлен в 0 для поиска с самой верхней записи таблицы.															
Выполнить поиск в таблице.																
AC1 <input type="text" value="2"/>	AC1 содержит номер записи данных, соответствующий первому совпадению, найденному в таблице (d2).															
AC1 <input type="text" value="3"/>	Увеличьте INDX на 1 перед поиском среди оставшихся записей в таблице.															
Выполнить поиск в таблице.																
AC1 <input type="text" value="4"/>	AC1 содержит номер записи данных, соответствующий второму совпадению, найденному в таблице (d4).															
AC1 <input type="text" value="5"/>	Увеличьте INDX на 1 перед поиском среди оставшихся записей в таблице.															
Выполнить поиск в таблице.																
AC1 <input type="text" value="6"/>	AC1 содержит значение, равное количеству записей. Вся таблица была просмотрена, но больше совпадений не найдено.															
AC1 <input type="text" value="0"/>	Прежде чем опять можно будет вести поиск в таблице, значение INDX должно быть сброшено в 0.															

Рис. 9-33. Пример команд поиска для SIMATIC LAD, STL и FBD

Удаление первой записи



Команда **Удаление первой записи (FIFO)** удаляет первую запись из таблицы (TBL) и пересылает значение по адресу, указанному в DATA. Все остальные записи таблицы смещаются на одну позицию вверх. Количество записей в таблице после выполнения каждой команды уменьшается на 1.

Ошибки, устанавливающие ENO в 0: SM1.5 (пустая таблица), SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация), 0091 (операнд вне допустимого диапазона)

Эта команда влияет на следующие биты специальной памяти: SM1.5 устанавливается в 1, если вы пытаетесь удалить запись из пустой таблицы.

Входы/выходы	Операнды	Типы данных
TABLE	VW, IW, QW, MW, SW, SMW, LW, T, C, *VD, *AC, *LD	INT
DATA	VW, IW, QW, MW, SW, SMW, LW, AC, AQW, T, C, *VD, *AC, *LD	WORD

Пример удаления первой записи

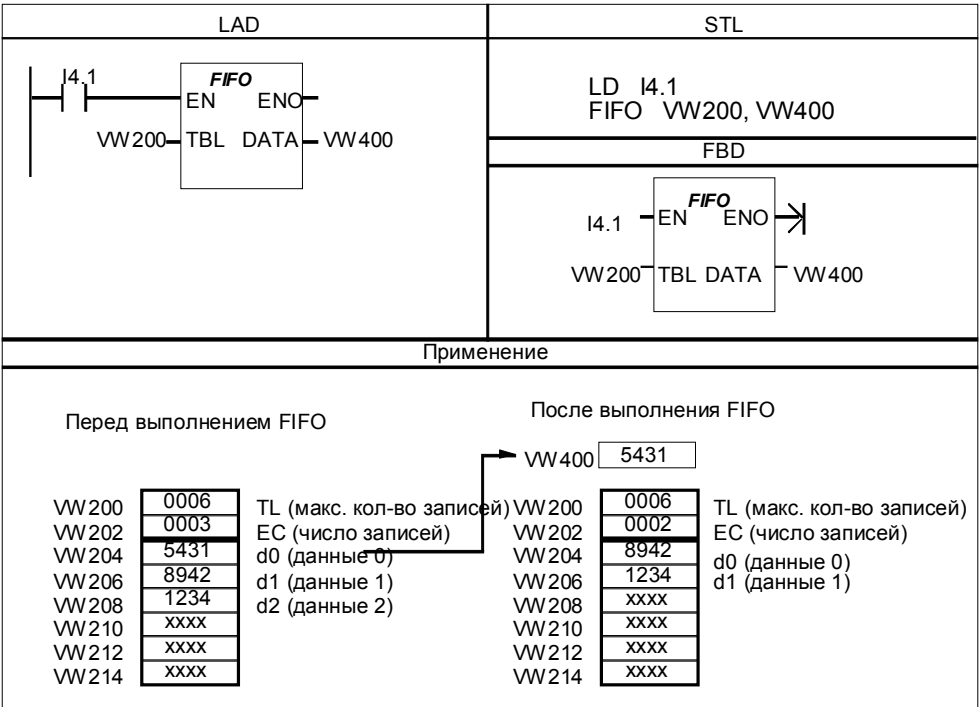
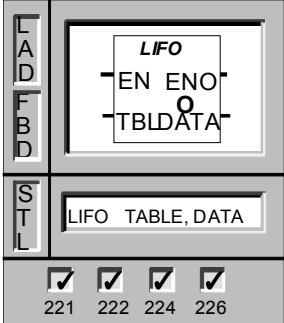


Рис. 9-34. Пример команды «Удаление первой записи» для SIMATIC LAD, STL и FBD

Удаление последней записи



Команда **Удаление последней записи (LIFO)** удаляет последнюю запись из таблицы (TBL) и пересылает значение по адресу, указанному в DATA. Количество записей в таблице после выполнения каждой команды уменьшается на 1.

Ошибки, устанавливающие ENO в 0: SM1.5 (пустая таблица), SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация), 0091 (операнд вне допустимого диапазона)

Эта команда влияет на следующие биты специальной памяти: SM1.5 устанавливается в 1, если вы пытаетесь удалить запись из пустой таблицы.

Входы/выходы	Операнды	Типы данных
TABLE	VW, IW, QW, MW, SW, SMW, LW, T, C, *VD, *AC, *LD	INT
DATA	VW, IW, QW, MW, SW, SMW, LW, AQW, T, C, AC, *VD, *AC, *LD	WORD

Пример удаления последней записи

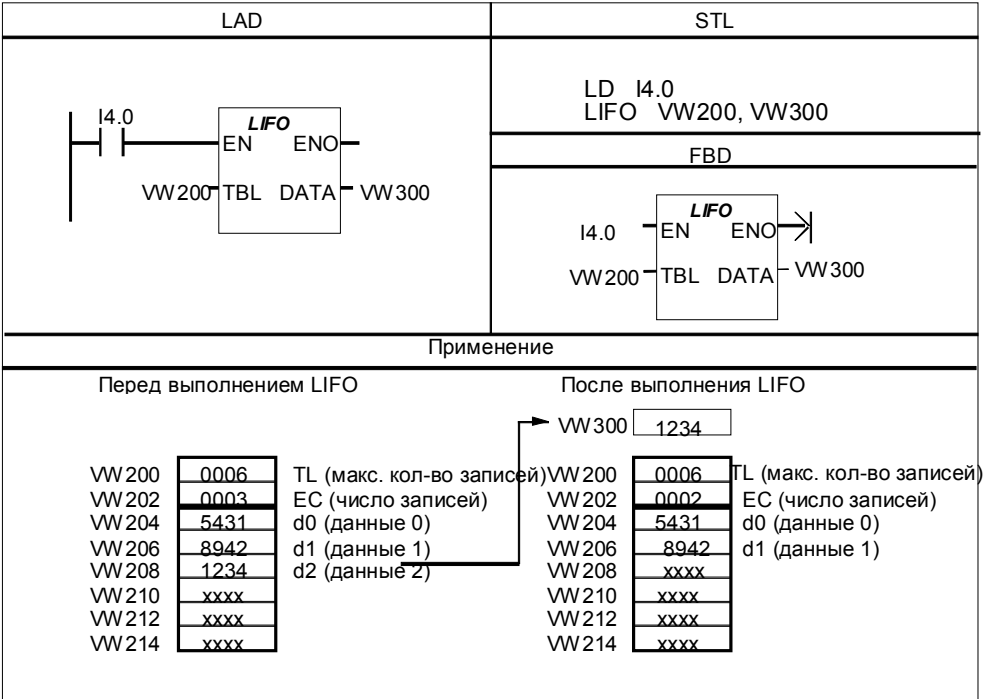
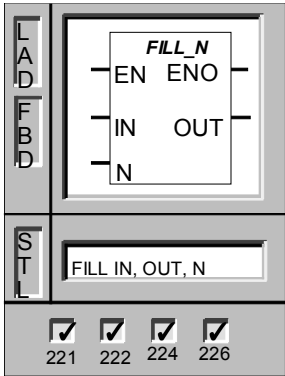


Рис. 9-35. Пример команды «Удалить последнюю запись» для SIMATIC LAD, STL и FBD

Заполнение памяти



Команда **Заполнить память** заполняет область памяти, начинающуюся с адреса OUT, N словами по образцу, расположенному по адресу IN. N имеет диапазон от 1 до 255.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация), 0091 (операнд вне допустимого диапазона)

Входы/выходы	Операнды	Типы данных
IN	VW, IW, QW, MW, SW, SMW, LW, AIW, T, C, AC, константа, *VD, *AC, *LD	WORD
N	VB, IB, QB, MB, SB, SMB, LB, AC, константа, *VD, *AC, *LD	BYTE
OUT	VW, IW, QW, MW, SW, SMW, LW, T, C, AQW, *VD, *AC, *LD	WORD

Пример заполнения

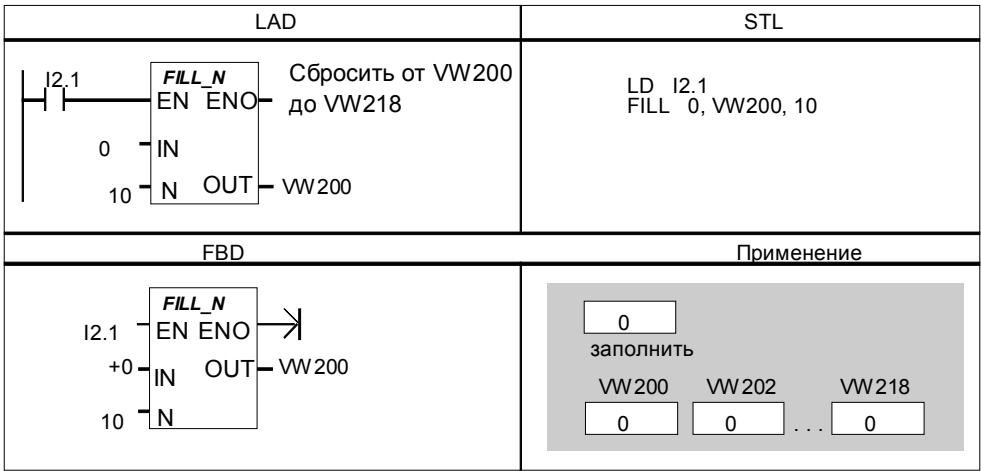
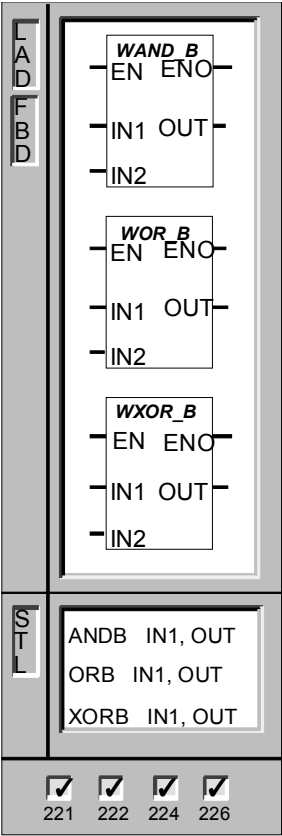


Рис. 9-36. Пример команды «Заполнить память» для SIMATIC LAD, STL и FBD

9.13 Логические команды SIMATIC

Логические операции И, ИЛИ и исключающее ИЛИ с байтами



Команда **Логическое И с байтами** сопрягает соответствующие биты двух входных байтов в соответствии с таблицей истинности логической операции И и загружает результат (OUT) в байт.

Команда **Логическое ИЛИ с байтами** сопрягает соответствующие биты двух входных байтов в соответствии с таблицей истинности логической операции ИЛИ и загружает результат (OUT) в байт.

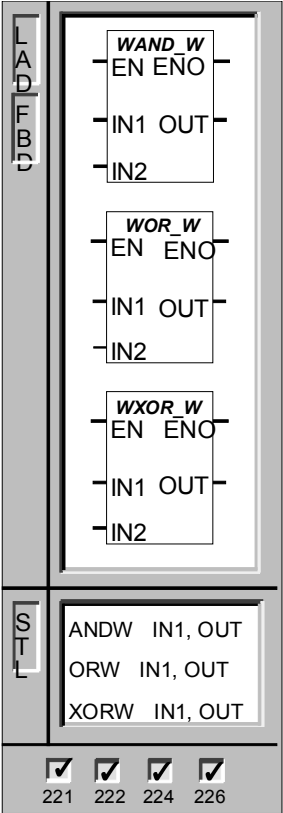
Команда **Исключающее ИЛИ с байтами** сопрягает соответствующие биты двух входных байтов в соответствии с таблицей истинности логической операции Исключающее ИЛИ и загружает результат (OUT) в байт.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль)

Входы/выходы	Операнды	Типы данных
IN1, IN2	VB, IB, QB, MB, SB, SMB, LB, AC, константа, *VD, *AC, *LD	BYTE
OUT	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD	BYTE

Логические операции И, ИЛИ и исключающее ИЛИ со словами



Команда **Логическое И со словами** сопрягает соответствующие биты двух входных слов в соответствии с таблицей истинности логической операции И и загружает результат (OUT) в слово.

Команда **Логическое ИЛИ со словами** сопрягает соответствующие биты двух входных слов в соответствии с таблицей истинности логической операции ИЛИ и загружает результат (OUT) в слово.

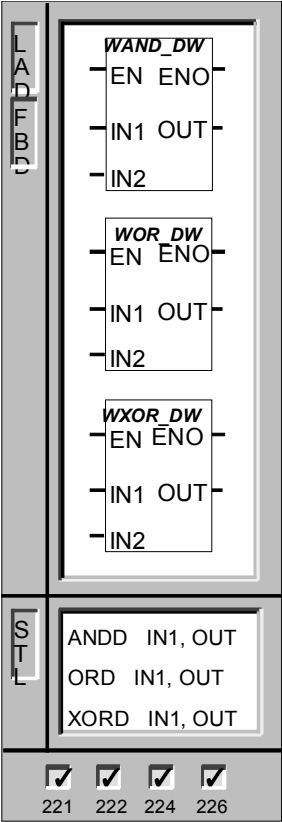
Команда **Исключающее ИЛИ со словами** сопрягает соответствующие биты двух входных слов в соответствии с таблицей истинности логической операции Исключающее ИЛИ и загружает результат (OUT) в слово.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль)

Входы/выходы	Операнды	Типы данных
IN1, IN2	VW, IW, QW, MW, SW, SMW, LW, T, C, AIW, AC, константа, *VD, *AC, *LD	WORD
OUT	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, *VD, *AC, *LD	WORD

Логические операции И, ИЛИ и исключающее ИЛИ с двойными словами



Команда **Логическое И с двойными словами** сопрягает соответствующие биты двух входных двойных слов в соответствии с таблицей истинности логической операции И и загружает результат (OUT) в двойное слово.

Команда **Логическое ИЛИ с двойными словами** сопрягает соответствующие биты двух входных двойных слов в соответствии с таблицей истинности логической операции ИЛИ и загружает результат (OUT) в двойное слово.

Команда **Исключающее ИЛИ с двойными словами** сопрягает соответствующие биты двух входных двойных слов в соответствии с таблицей истинности логической операции Исключающее ИЛИ и загружает результат (OUT) в двойное слово.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль)

Входы/выходы	Операнды	Типы данных
IN1, IN2	VD, ID, QD, MD, SD, SMD, AC, LD, HC, константа, *VD, *AC, SD, *LD	DWORD
OUT	VD, ID, QD, MD, SMD, LD, AC, *VD, *AC, SD, *LD	DWORD

Пример команд И, ИЛИ и Исключающее ИЛИ

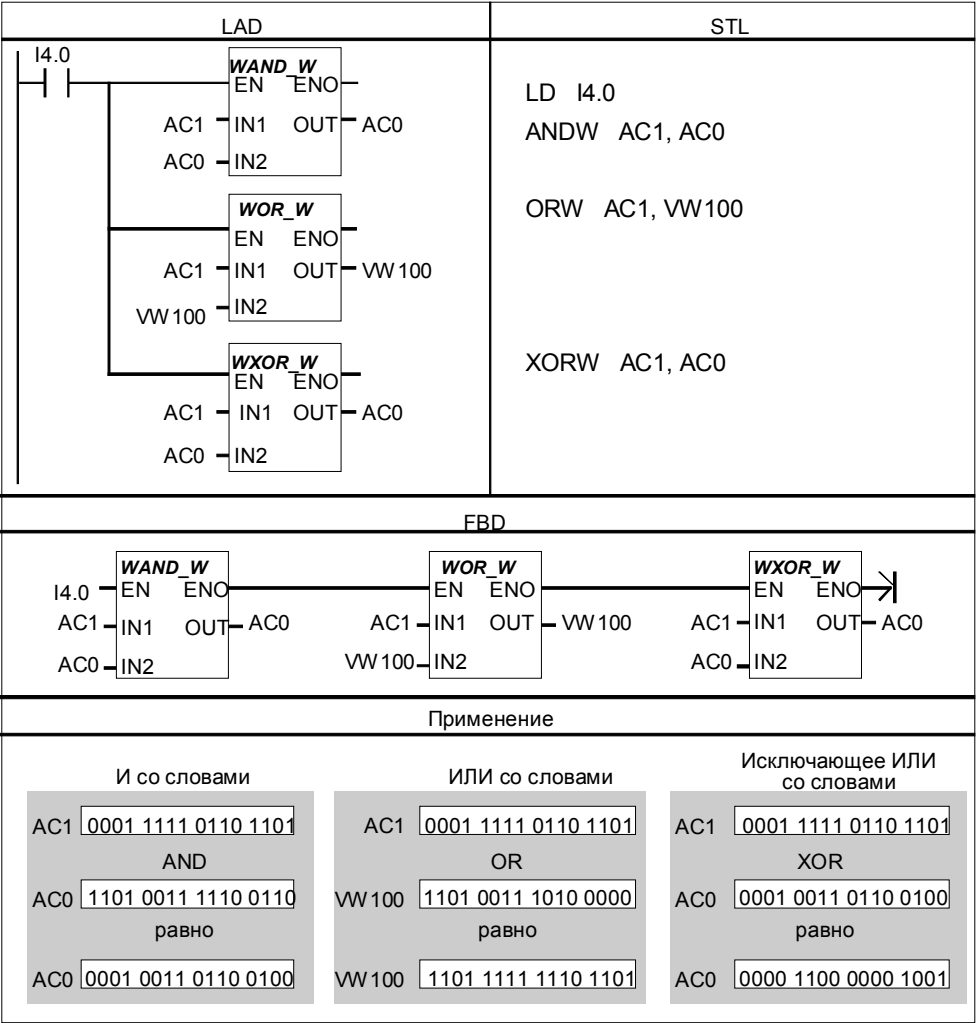
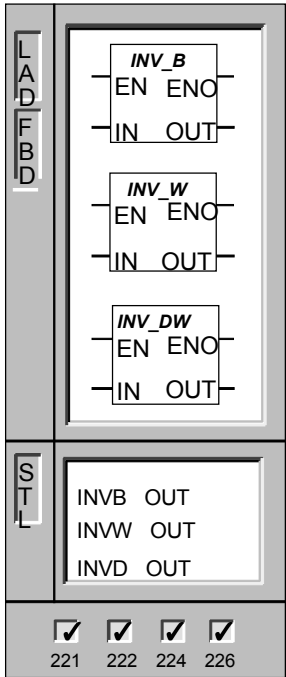


Рис. 9-37. Пример логических операций для SIMATIC LAD, STL и FBD

Команды инвертирования байта, слова, двойного слова



Команда **Инвертировать байт** образует дополнение до единицы входного байта IN и загружает результат в выходной байт OUT.

Команда **Инвертировать слово** образует дополнение до единицы входного слова IN и загружает результат в выходное слово OUT.

Команда **Инвертировать двойное слово** образует дополнение до единицы входного двойного слова IN и загружает результат в выходное двойное слово OUT.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль)

Инверти- ровать...	Входы/выходы	Операнды	Типы данных
байт	IN	VB, IB, QB, MB,SB, SMB, LB, AC, константа, *VD, *AC, *LD	BYTE
	OUT	VB, IB, QB, MB, SB, SMB, LB, AC,*VD, *AC, *LD	BYTE
слово	IN	VW, IW, QW, MW, SW, SMW, T, C, AIW, LW, AC, константа, *VD, *AC, *LD	WORD
	OUT	VW, IW, QW, MW, SW, SMW, T, C, LW, AC, *VD, *AC, *LD	WORD
двойное слово	IN	VD, ID, QD, MD, SD, SMD, LD, HC, AC, константа, *VD, *AC, *LD	DWORD
	OUT	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD	DWORD

Пример инвертирования

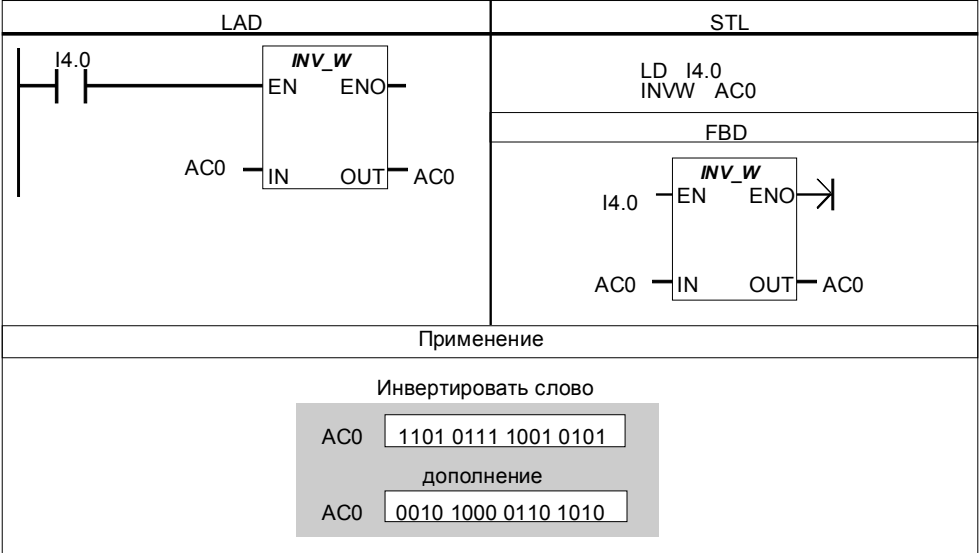
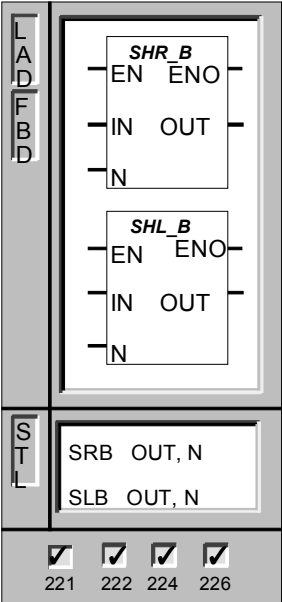


Рис. 9-38. Пример команды инвертирования для SIMATIC LAD, STL и FBD

9.14 Команды SIMATIC для сдвига и циклического сдвига

Сдвиг байта вправо, влево



Команды **Сдвинуть байт вправо** и **Сдвинуть байт влево** сдвигают содержимое входного байта (IN) вправо или влево на число разрядов, указанное в (N), и загружают результат в выходной байт (OUT).

Команды сдвига заполняют позиции выдвигаемых битов нулями. Если величина сдвига (N) больше или равна 8, то сдвиг производится не более 8 раз.

Если величина сдвига больше 0, то бит переполнения (SM1.1) принимает значение последнего выдвинутого бита. Бит нулевого значения (SM1.0) устанавливается, если результат операции сдвига равен нулю.

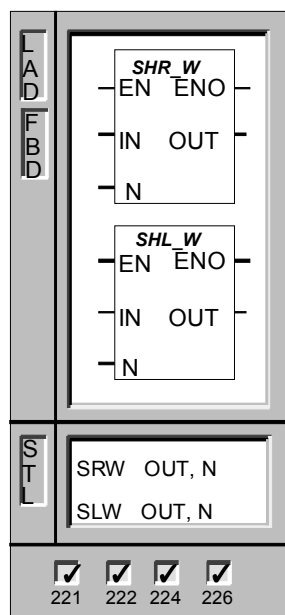
Операции сдвига байта вправо и влево являются беззнаковыми.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение)

Входы/выходы	Операнды	Типы данных
IN	VB, IB, QB, MB, SB, SMB, LB, AC, константа, *VD, *AC, *LD	BYTE
OUT	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD	BYTE
N	VB, IB, QB, MB, SB, SMB, LB, AC, константа, *VD, *AC, *LD	BYTE

Сдвиг слова вправо, влево



Команды **Сдвинуть слово вправо** и **Сдвинуть слово влево** сдвигают содержимое входного слова (IN) вправо или влево на число разрядов, указанное в (N), и загружают результат в выходное слово (OUT).

Команды сдвига заполняют позиции выдвигаемых битов нулями. Если величина сдвига (N) больше или равна 16, то сдвиг производится не более 16 раз. Если величина сдвига больше 0, то бит переполнения (SM1.1) принимает значение последнего выдвинутого бита. Бит нулевого значения (SM1.0) устанавливается, если результат операции сдвига равен нулю.

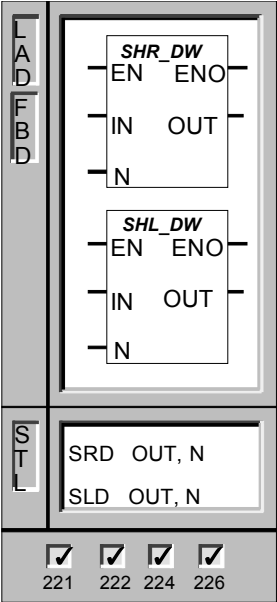
Обратите внимание, что знаковый бит сдвигается, если вы используете типы данных со знаком.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение)

Входы/выходы	Операнды	Типы данных
IN	VW, IW, QW, MW, SW, SMW, LW, T, C, AIW, AC, константа, *VD, *AC, *LD	WORD
N	VB, IB, QB, MB, SB, SMB, LB, AC, константа, *VD, *AC, *LD	BYTE
OUT	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, *VD, *AC, *LD	WORD

Сдвиг двойного слова вправо, влево



Команды **Сдвинуть двойное слово вправо** и **Сдвинуть двойное слово влево** сдвигают содержимое входного двойного слова (IN) вправо или влево на число разрядов, указанное в (N), и загружают результат в выходное двойное слово (OUT). Marker 66

Команды сдвига заполняют позиции выдвигаемых битов нулями. Если величина сдвига (N) больше или равна 32, то сдвиг производится не более 32 раз. Если величина сдвига больше 0, то бит переполнения (SM1.1) принимает значение последнего выдвинутого бита. Бит нулевого значения (SM1.0) устанавливается, если результат операции сдвига равен нулю.

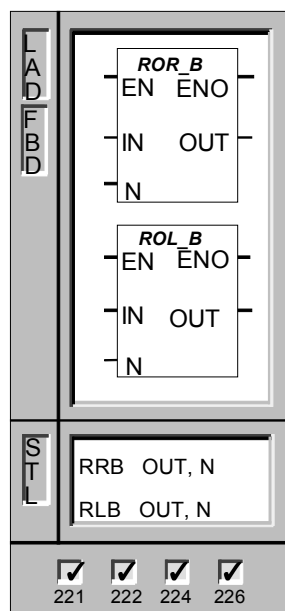
Обратите внимание, что знаковый бит сдвигается, если вы используете типы данных со знаком.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение)

Входы/выходы	Операнды	Типы данных
IN	VD, ID, QD, MD, SD, SMD, LD, AC, HC, константа, *VD, *AC, *LD	DWORD
N	VB, IB, QB, MB, SB, SMB, LB, AC, константа, *VD, *AC, *LD	BYTE
OUT	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD	DWORD

Циклический сдвиг байта вправо, влево



Команды **Сдвинуть циклически байт вправо** и **Сдвинуть циклически байт влево** циклически сдвигают содержимое входного байта (IN) вправо или влево на число разрядов, указанное в (N), и загружают результат в выходной байт (OUT). Циклический сдвиг является кольцевым.

Если величина сдвига (N) больше или равна 8, то перед выполнением циклического сдвига от величины сдвига, указанной в (N), берется остаток от деления на 8. В результате величина сдвига находится в пределах от 0 до 7. Если величина сдвига равна нулю, то циклический сдвиг не производится. Если циклический сдвиг выполняется, то значение последнего циклически сдвинутого бита копируется в бит переполнения (SM1.1).

Если величина сдвига не является целым кратным 8, то последний циклически выдвинутый бит копируется в бит переполнения (SM1.1). Бит нулевого значения (SM1.0) устанавливается, если подлежащая циклическому сдвигу величина равна нулю.

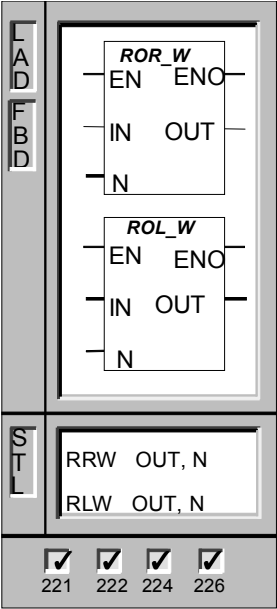
Операции циклического сдвига байта вправо и влево являются беззнаковыми.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение)

Входы/выходы	Операнды	Типы данных
IN	VB, IB, QB, MB, SMB, SB, LB, AC, *VD, *AC, *LD	BYTE
N	VB, IB, QB, MB, SMB, SB, LB, AC, константа, *VD, *AC, *LD	BYTE
OUT	VB, IB, QB, MB, SMB, SB, LB, AC, *VD, *AC, *LD	BYTE

Циклический сдвиг слова вправо, влево



Команды **Сдвинуть циклически слово вправо** и **Сдвинуть циклически слово влево** циклически сдвигают содержимое входного слова (IN) вправо или влево на число разрядов, указанное в (N), и загружают результат в выходное слово (OUT). Циклический сдвиг является кольцевым.

Если величина сдвига (N) больше или равна 16, то перед выполнением циклического сдвига от величины сдвига, указанной в (N), берется остаток от деления на 16. В результате величина сдвига находится в пределах от 0 до 15. Если величина сдвига равна нулю, то циклический сдвиг не производится. Если циклический сдвиг выполняется, то значение последнего циклически сдвинутого бита копируется в бит переполнения (SM1.1).

Если величина сдвига не является целым кратным 16, то последний циклически выдвинутый бит копируется в бит переполнения (SM1.1). Бит нулевого значения (SM1.0) устанавливается, если подлежащая циклическому сдвигу величина равна нулю.

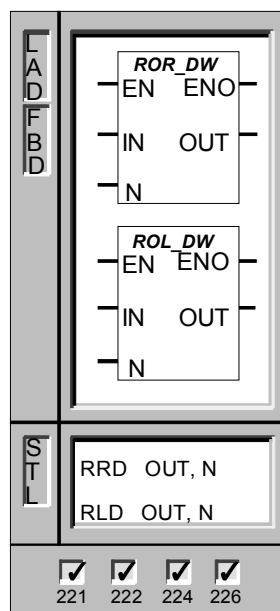
Обратите внимание, что знаковый бит сдвигается, если вы используете типы данных со знаком.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение)

Входы/выходы	Операнды	Типы данных
IN	VW, T, C, IW, MW, SW, SMW, AC, QW, LW, AIW, константа, *VD, *AC, *LD	WORD
N	VB, IB, QB, MB, SMB, LB, AC, константа, *VD, *AC, SB, *LD	BYTE
OUT	VW, T, C, IW, QW, MW, SW, SMW, LW, AC, *VD, *AC, *LD	WORD

Циклический сдвиг двойного слова вправо, влево



Команды **Сдвинуть циклически двойное слово вправо** и **Сдвинуть циклически двойное слово влево** циклически сдвигают содержимое входного двойного слова (IN) вправо или влево на число разрядов, указанное в (N), и загружают результат в выходное двойное слово (OUT). Циклический сдвиг является кольцевым.

Если величина сдвига (N) больше или равна 32, то перед выполнением циклического сдвига от величины сдвига, указанной в (N), берется остаток от деления на 32. В результате величина сдвига находится в пределах от 0 до 31. Если величина сдвига равна нулю, то циклический сдвиг не производится. Если циклический сдвиг выполняется, то значение последнего циклически сдвинутого бита копируется в бит переполнения (SM1.1).

Если величина сдвига не является целым кратным 32, то последний циклически выдвинутый бит копируется в бит переполнения (SM1.1). Бит нулевого значения (SM1.0) устанавливается, если подлежащая циклическому сдвигу величина равна нулю.

Обратите внимание, что знаковый бит сдвигается, если вы используете типы данных со знаком.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.0 (ноль); SM1.1 (переполнение)

Входы/выходы	Операнды	Типы данных
IN	VD, ID, QD, MD, SMD, LD, AC, HC, константа, *VD, *AC, SD, *LD	DWORD
N	VB, IB, QB, MB, SMB, LB, AC, константа, *VD, *AC, SB, *LD	BYTE
OUT	VD, ID, QD, MD, SMD, LD, AC, *VD, *AC, SD, *LD	DWORD

Примеры сдвига и циклического сдвига

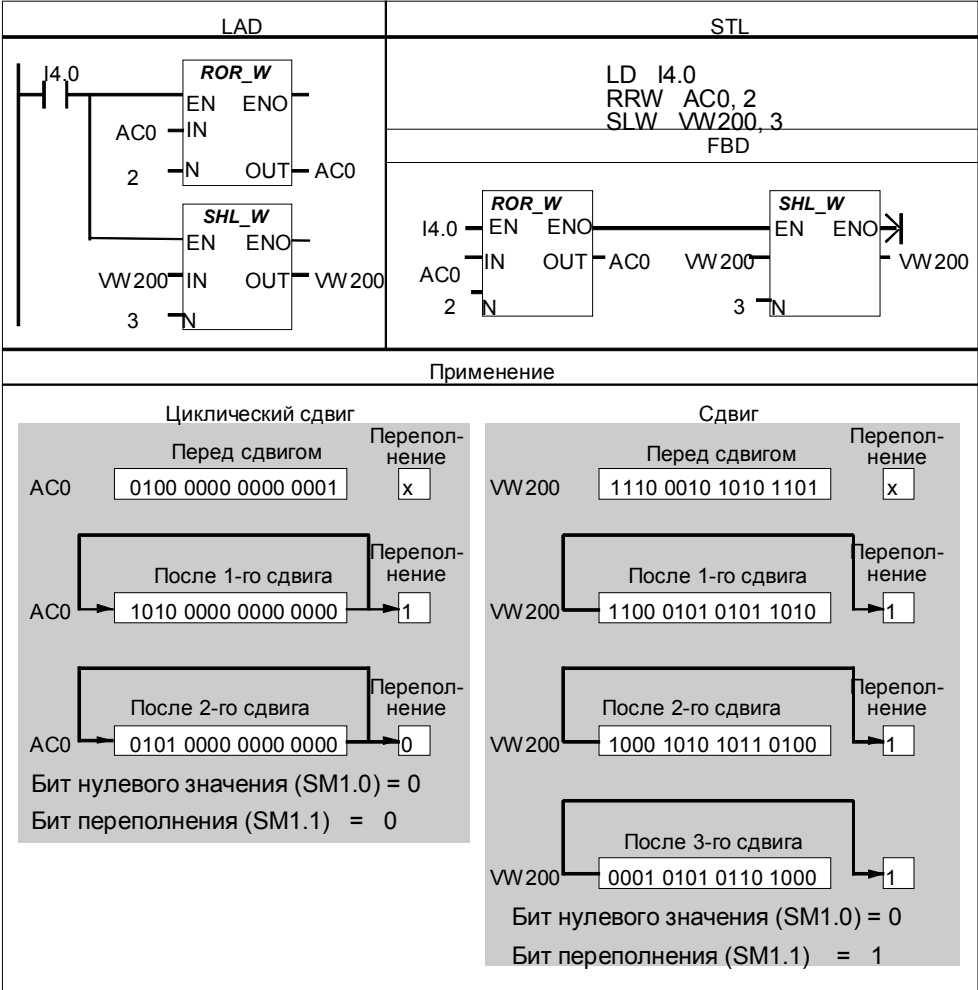
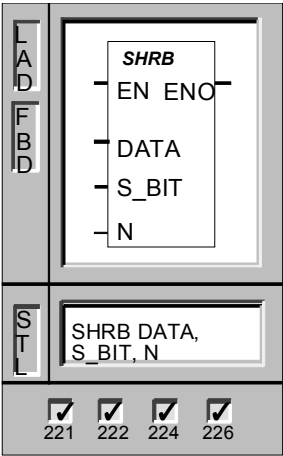


Рис. 9-39. Пример команд сдвига и циклического сдвига для SIMATIC LAD, STL и FBD

Вдвигание бита в регистр сдвига



Команда **Вдвинуть бит в регистр сдвига** (SHRB) вдвигает значение DATA в регистр сдвига. S_BIT указывает младший бит регистра сдвига. N задает длину регистра сдвига и направление сдвига (положительный сдвиг = N, отрицательный сдвиг = -N).

Каждый бит, выдвинутый из регистра командой SHRB, помещается в бит переполнения (SM1.1).

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация), 0091 (выход операнда за пределы допустимого диапазона), 0092 (ошибка в поле счета)

Эта команда влияет на следующий бит специальной памяти: SM1.1 (переполнение)

Входы/выходы	Операнды	Типы данных
DATA, S_BIT	I, Q, M, SM, T, C, V, S, L	BOOL
N	VB, IB, QB, MB, SMB, LB, AC, константа, *VD, *AC, SB, *LD	BYTE

Описание команды вдвигания бита в регистр сдвига

Команда **Вдвинуть бит в регистр сдвига** предоставляет в распоряжение простой метод упорядочения и управления потоком изделий или данных. С помощью этой команды вы можете сдвигать весь регистр на один бит каждый цикл. Команда **Вдвинуть бит в регистр сдвига** определяется младшим битом регистра сдвига (S_BIT) и количеством битов, задаваемых параметром (N). На рис. 9–41 показан пример команды вдвигания бита в регистр сдвига.

Адрес старшего бита регистра сдвига (MSB.b) можно рассчитать с помощью следующего уравнения:

$$\text{MSB.b} = [(\text{байт параметра S_BIT}) + ([N] - 1 + (\text{бит параметра S_BIT})) / 8] \text{. [остаток от деления на 8]}$$

Вы должны вычесть 1 бит, так как S_BIT – это один из битов регистра сдвига.

Например, если S_BIT – это V33.4, и N = 14, то MSB.b – это V35.1, или:

$$\begin{aligned} \text{MSB.b} &= \text{V33} + ([14] - 1 + 4) / 8 \\ &= \text{V33} + 17 / 8 \\ &= \text{V33} + 2 \text{ с остатком } 1 \\ &= \text{V35.1} \end{aligned}$$

При отрицательном сдвиге, который задается отрицательным значением длины (N), входное значение (DATA) вдвигается на место старшего бита регистра сдвига и выдвигает из регистра младший бит (S_BIT).

При положительном сдвиге, который задается положительным значением длины (N), входное значение (DATA) вдвигается на место младшего бита регистра сдвига (S_BIT) и выдвигает из регистра сдвига старший бит.

Данные, выдвинутые из регистра, помещаются в бит переполнения (SM1.1). Максимальная длина регистра сдвига (положительная или отрицательная) равна 64 битам. На рис. 9–40 показан сдвиг битов при отрицательном и положительном значении N.

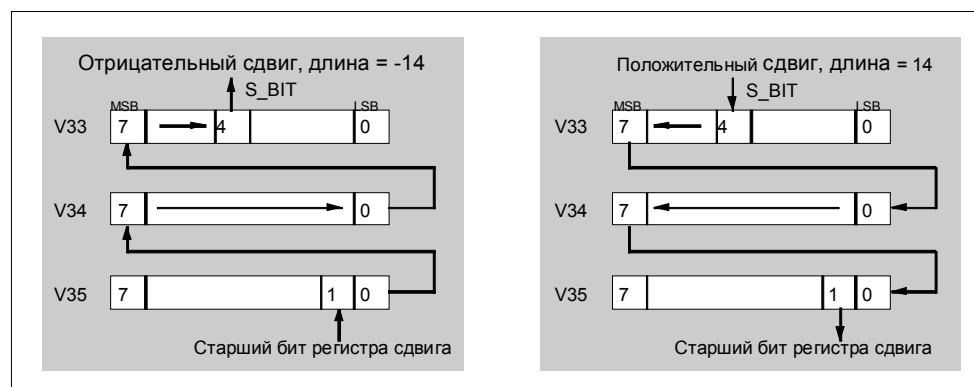


Рис. 9-40. Положительный и отрицательный сдвиг в регистре сдвига

Пример команды вдвигания бита в регистр сдвига

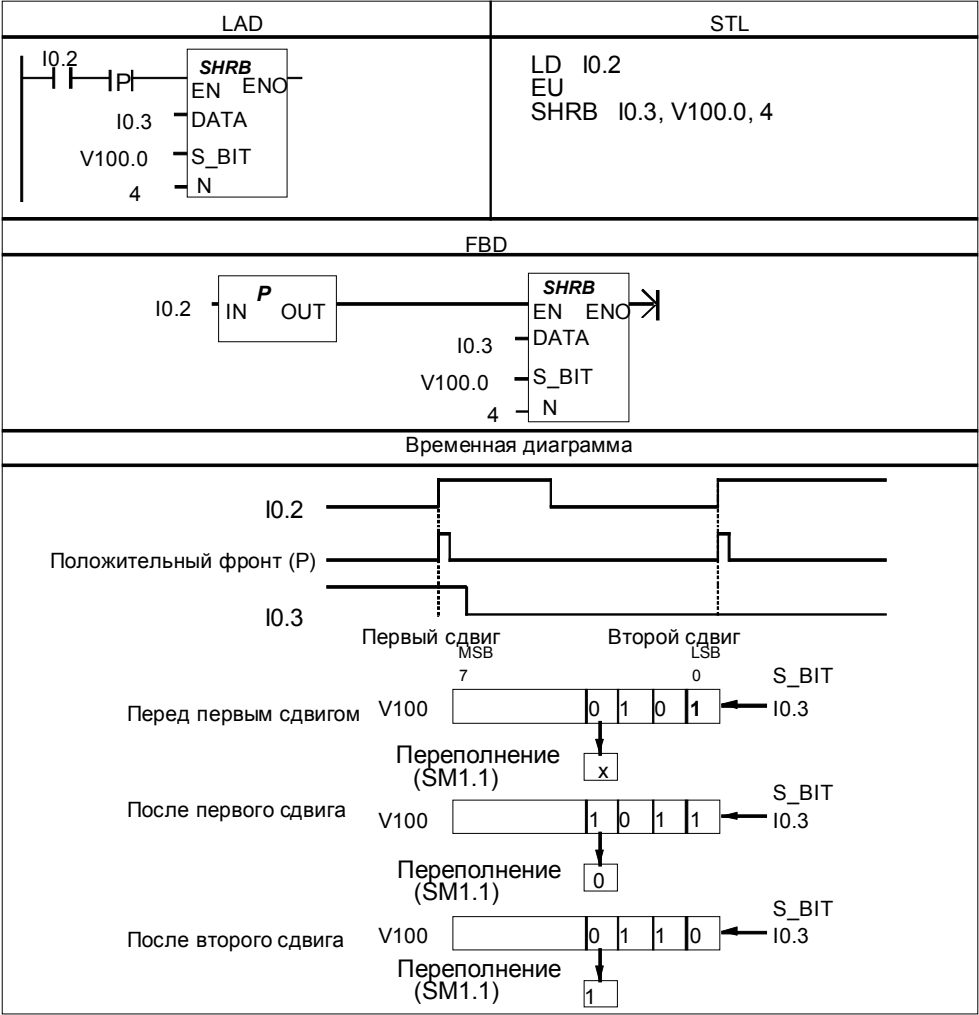
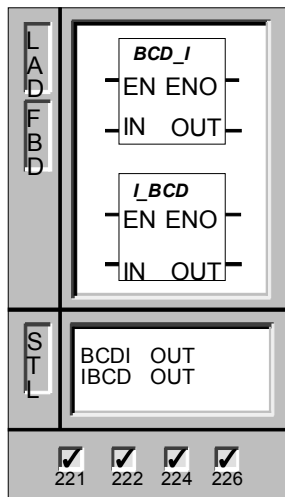


Рис. 9-41. Пример вдвигания бита в регистр сдвига для SIMATIC LAD, STL и FBD

9.15 Команды SIMATIC для выполнения преобразований

Преобразование BCD в целое и целого в BCD



Команда **BCD в целое** преобразует входное число, представленное в двоично-десятичном формате, (IN) в целое число и загружает результат в переменную, указанную в OUT. Допустимый диапазон для IN: от 0 до 9999 (BCD).

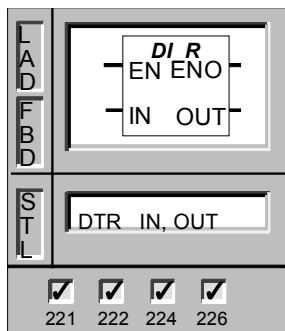
Команда **Целое в BCD** преобразует входное целое число (IN) в двоично-десятичный формат и загружает результат в переменную, указанную в OUT. Допустимый диапазон для IN: от 0 до 9999 (целое).

Ошибки, устанавливающие ENO в 0: SM1.6 (ошибка BCD), SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эти команды влияют на следующие биты специальной памяти: SM1.6 (недопустимое значение BCD)

Входы/выходы	Операнды	Типы данных
IN	VW, T, C, IW, QW, MW, SMW, LW, AC, AIW, константа, *VD, *AC, SW, *LD	WORD
OUT	VW, T, C, IW, QW, MW, SMW, LW, AC, *VD, *AC, SW, *LD	WORD

Преобразование двойного целого в вещественное

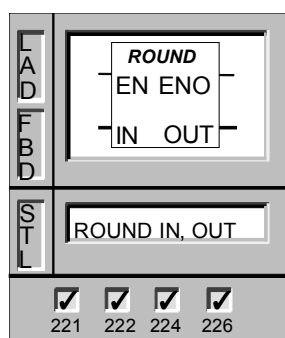


Команда **Двойное целое в вещественное** преобразует 32-битовое целое со знаком (IN) в 32-битовое вещественное число и помещает результат в переменную, указанную в OUT.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Входы/выходы	Операнды	Типы данных
IN	VD, ID, QD, MD, SMD, AC, LD, HC, константа, *VD, *AC, SD, *LD	DINT
OUT	VD, ID, QD, MD, SMD, LD, AC, *VD, *AC, SD, *LD	REAL

Округление



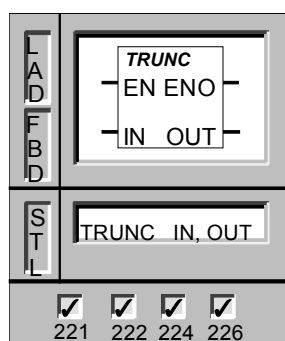
Команда **Округлить** преобразует вещественное число (IN) в двойное целое число и помещает результат в переменную, указанную в OUT. Если дробная часть равна 0,5 или больше, то число округляется в большую сторону.

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эта команда влияет на следующий бит специальной памяти: SM1.1 (переполнение)

Входы/выходы	Операнды	Типы данных
IN	VD, ID, QD, MD, SMD, AC, LD, константа, *VD, *AC, SD, *LD	REAL
OUT	VD, ID, QD, MD, SMD, LD, AC, *VD, *AC, SD, *LD	DINT

Округление отбрасыванием



Команда **Округлить отбрасыванием** преобразует 32-битовое вещественное число (IN) в 32-битовое целое число со знаком и помещает результат в переменную, указанную в OUT. Преобразуется только целая часть вещественного числа, а дробная часть отбрасывается.

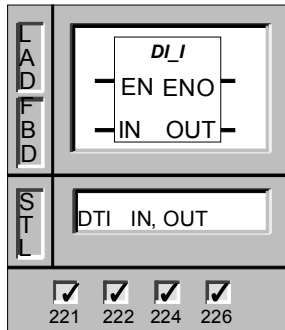
Если преобразованное вами значение не является допустимым вещественным числом или слишком велико, чтобы быть представленным на выходе, то устанавливается бит переполнения, а выход не меняется.

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эта команда влияет на следующий бит специальной памяти: SM1.1 (переполнение)

Входы/выходы	Операнды	Типы данных
IN	VD, ID, QD, MD, SMD, LD, AC, константа, *VD, *AC, SD, *LD	REAL
OUT	VD, ID, QD, MD, SMD, LD, AC, *VD, *AC, SD, *LD	DINT

Преобразование двойного целого числа в целое



Команда **Двойное целое в целое** преобразует двойное целое число (IN) в целое число и помещает результат в переменную, указанную в OUT.

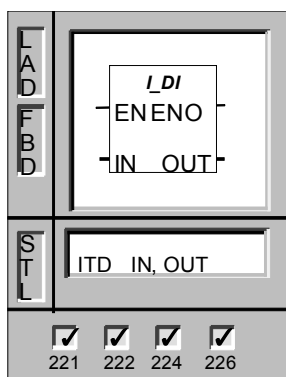
Если преобразованное вами значение слишком велико, чтобы быть представленным на выходе, то устанавливается бит переполнения, а выход не изменяется.

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эта команда влияет на следующий бит специальной памяти: SM1.1 (переполнение)

Входы/выходы	Операнды	Типы данных
IN	VD, ID, QD, MD, SMD, AC, LD, HC, константа, *VD, *AC, SD, *LD	DINT
OUT	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, *VD, *LD, *AC	INT

Преобразование целого числа в двойное целое число



Команда **Целое в двойное целое** преобразует целое число (IN) в двойное целое число и помещает результат в переменную, указанную в OUT. Знак распространяется.

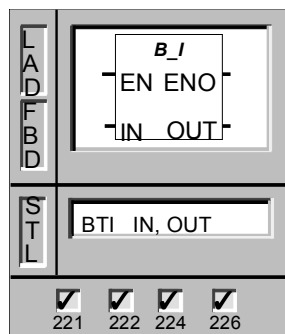
Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Входы/выходы	Операнды	Типы данных
IN	VW, IW, QW, MW, SW, SMW, LW, T, C, AIW, AC, константа, *AC, *VD, *LD	INT
OUT	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *LD, *AC	DINT

Преобразование целого числа в вещественное

Для преобразования целого числа в вещественное используйте команду «Целое в двойное целое», а затем команду «Двойное целое в вещественное» (стр. 9–133). См. рис. 9–42.

Преобразование байта в целое число

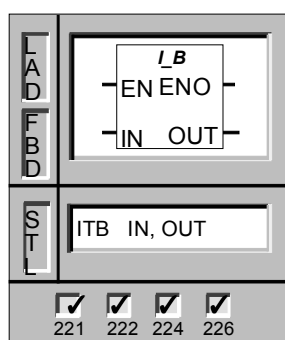


Команда **Байт в целое** преобразует байт (IN) в целое число и помещает результат в переменную, указанную в OUT. Байт не имеет знака, поэтому распространение знака не происходит.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Входы/выходы	Операнды	Типы данных
IN	VB, IB, QB, MB, SB, SMB, LB, AC, константа, *AC, *VD, *LD	BYTE
OUT	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, *VD, *LD, *AC	INT

Преобразование целого числа в байт



Команда **Целое в байт** преобразует слово (IN) в байт и помещает результат в переменную, указанную в OUT.

Преобразуются значения от 0 до 255. Все остальные значения приводят к переполнению и не влияют на выход.

Ошибки, устанавливающие ENO в 0: SM1.1 (переполнение), SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Эта команда влияет на следующий бит специальной памяти: SM1.1 (переполнение)

Входы/выходы	Операнды	Типы данных
IN	VW, IW, QW, MW, SW, SMW, LW, T, C, AIW, AC, константа, *VD, *LD, *AC	INT
OUT	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD	BYTE

Пример преобразований

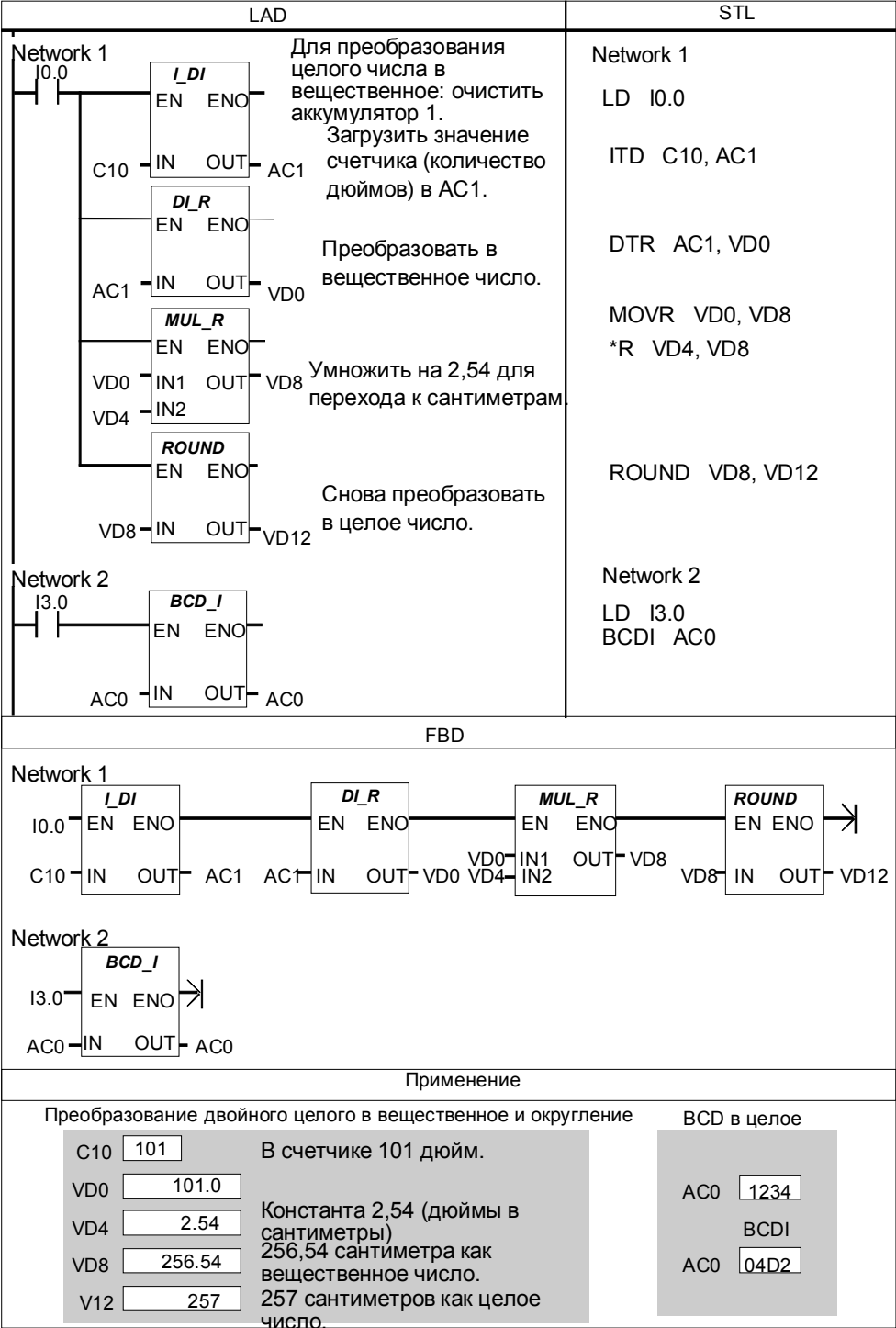
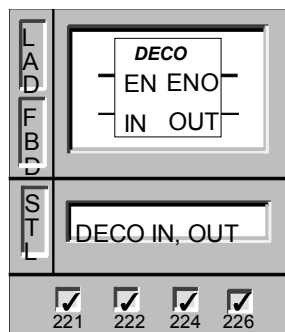


Рис. 9-42. Пример команд преобразования для SIMATIC LAD, STL и FBD

Декодирование

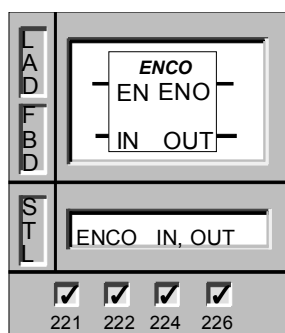


Команда **Декодировать** устанавливает в выходном слове (OUT) бит, соответствующий номеру бита, представленному младшим полубайтом (4 бита) входного байта (IN). Все остальные биты выходного слова устанавливаются в 0.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Входы/выходы	Операнды	Типы данных
IN	VB, IB, QB, MB, SMB, LB, SB, AC, константа, *VD, *AC, *LD	BYTE
OUT	VW, IW, QW, MW, SMW, LW, SW, AQW, T, C, AC, *VD, *AC, *LD	WORD

Кодирование



Команда **Закодировать** записывает номер самого младшего установленного бита входного слова (IN) в младший полубайт (4 бита) выходного байта (OUT).

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Входы/выходы	Операнды	Типы данных
IN	VW, T, C, IW, QW, MW, SMW, AC, LW, AIW, константа, *VD, *AC, SW, *LD	WORD
OUT	VB, IB, QB, MB, SMB, LB, AC, *VD, *AC, SB, *LD	BYTE

Примеры декодирования и кодирования

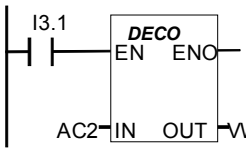
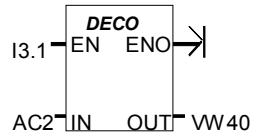
LAD	STL
 <p>Установить бит, соответствующий коду ошибки в AC2.</p>	<pre>LD I3.1 DECO AC2, VW40</pre> <p>FBD</p> 
Применение	
<p>AC2 содержит код ошибки 3. Команда DECO устанавливает в VW40 бит, соответствующий этому коду ошибки.</p>	

Рис. 9-43. Пример установки бита ошибки с помощью команды декодирования для LAD, STL и FBD

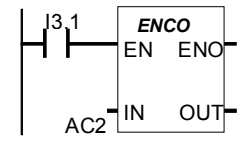
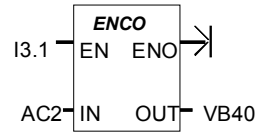
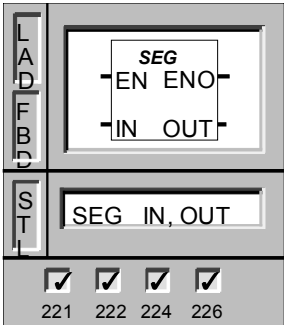
LAD	STL
 <p>Преобразовать бит ошибки в AC2 в код ошибки в VB40.</p>	<pre>LD I3.1 ENCO AC2, VB40</pre> <p>FBD</p> 
Применение	
<p>AC2 содержит код ошибки. Команда ENCO преобразует самый младший установленный бит в код ошибки, который сохраняется в VB40.</p>	

Рис. 9-44. Пример преобразования бита ошибки в код ошибки с помощью команды кодирования для LAD, STL и FBD

Сегмент



Команда **Сегмент** использует символ, заданный в IN, для генерирования двоичного кода (OUT), с помощью которого высвечиваются сегменты семисегментного дисплея. Высвечиваемые сегменты представляют символ младшей цифры входного байта (IN).

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

На рис. 9–45 показано кодирование семисегментного дисплея с помощью команды Сегмент.

Входы/выходы	Операнды	Типы данных
IN	VB, IB, QB, MB, SMB, LB, AC, константа, *VD, *AC, SB, *LD	BYTE
OUT	VB, IB, QB, MB, SMB, LB, AC, *VD, *AC, SB, *LD	BYTE

(IN) LSD	Отображение сегментов	(OUT) - g f e d c b a		(IN) LSD	Отображение сегментов	(OUT) - g f e d c b a
0	0	0 0 1 1 1 1 1 1		8	8	0 1 1 1 1 1 1 1
1	1	0 0 0 0 0 1 1 0		9	9	0 1 1 0 0 1 1 1
2	2	0 1 0 1 1 0 1 1		A	A	0 1 1 1 0 1 1 1
3	3	0 1 0 0 1 1 1 1		B	B	0 1 1 1 1 1 0 0
4	4	0 1 1 0 0 1 1 0		C	C	0 0 1 1 1 0 0 1
5	5	0 1 1 0 1 1 0 1		D	D	0 1 0 1 1 1 1 0
6	6	0 1 1 1 1 1 0 1		E	E	0 1 1 1 1 0 0 1
7	7	0 0 0 0 0 1 1 1		F	F	0 1 1 1 0 0 0 1

Рис. 9-45. Кодирование семисегментного дисплея

Пример команды «Сегмент»

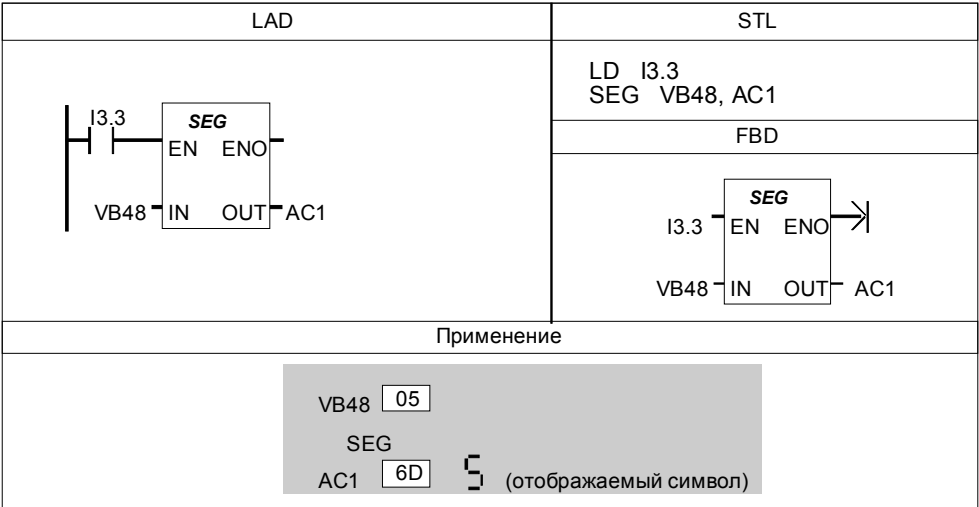
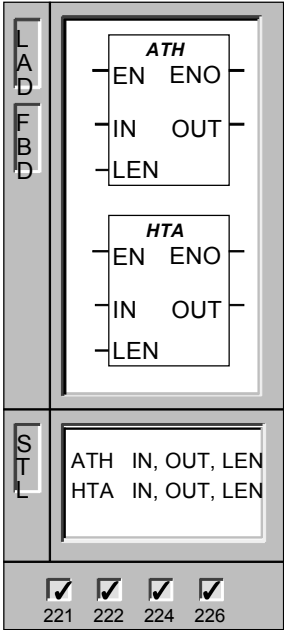


Рис. 9-46. Пример команды «Сегмент» для SIMATIC LAD, STL и FBD

Преобразование ASCII в 16-ричный код и 16-ричного кода в ASCII



Команда **Преобразовать ASCII в 16-ричный код** преобразует строку символов ASCII длиной (LEN), начинающуюся с адреса IN, в шестнадцатеричные цифры, начиная с адреса OUT. Максимальная длина строки ASCII составляет 255 символов.

Команда **Преобразовать 16-ричный код в ASCII** преобразует шестнадцатеричные цифры, начинающиеся с входного байта (IN), в строку символов ASCII, которая начинается по адресу OUT. Количество шестнадцатеричных цифр, подлежащих преобразованию, задается длиной (LEN). Максимальное количество шестнадцатеричных цифр, которое может быть преобразовано, равно 255.

Допустимым символам ASCII соответствуют шестнадцатеричные значения от 30 до 39 и от 41 до 46.

Преобразование ASCII в 16-ричный код: Ошибки, устанавливающие ENO в 0: SM1.7 (недопустимый символ ASCII), SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация), 0091 (операнд вне допустимого диапазона)

Преобразование 16-ричного кода в: Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация), 0091 (операнд вне допустимого диапазона)

Эти команды влияют на следующий бит специальной памяти: SM1.7 (недопустимый символ ASCII)

Входы/выходы	Операнды	Типы данных
IN, OUT	VB, IB, QB, MB, SMB, LB, *VD, *AC, SB, *LD	BYTE
LEN	VB, IB, QB, MB, SMB, LB, AC, константа, *VD, *AC, SB, *LD	BYTE

Пример преобразования ASCII в 16-ричный код

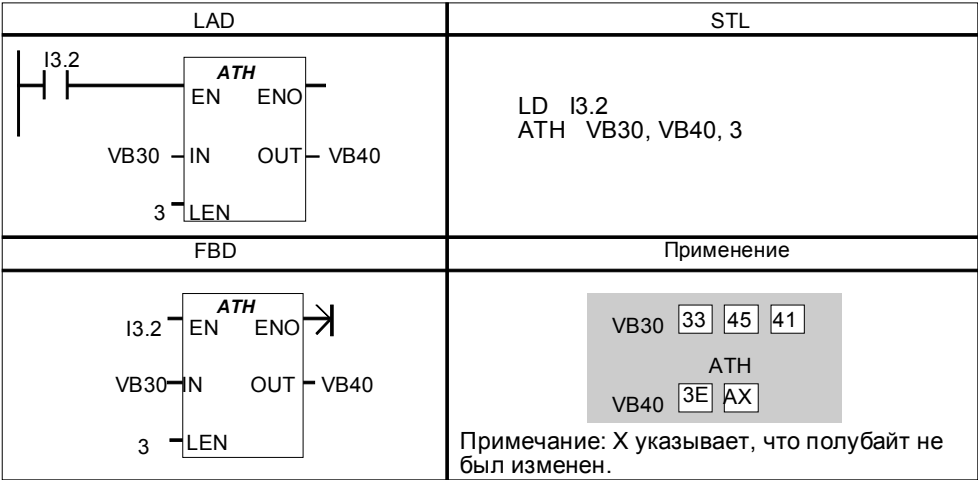


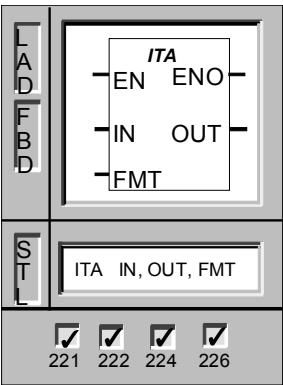
Рис. 9-47. Пример преобразования ASCII в 16-ричный код для SIMATIC LAD, STL и FBD

Преобразование целого числа в строку ASCII

LAD

FBD

STL



Команда **Преобразовать целое число в строку ASCII** преобразует целое число, содержащееся в слове (IN), в строку ASCII. Операнд формат (FMT) определяет точность преобразования справа от десятичной точки, а также форму представления десятичной точки – в виде запятой или точки. Результат преобразования помещается в 8 последовательных байтов, начиная с адреса OUT. Строка ASCII всегда содержит 8 символов.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация), нет выхода (недопустимый формат)

Входы/выходы	Операнды	Типы данных
IN	VW, IW, QW, MW, SW, SMW, LW, AIW, T, C, AC, константа, *VD, *AC, *LD	INT
FMT	VB, IB, QB, MB, SMB, LB, AC, константа, *VD, *AC, SB, *LD	BYTE
OUT	VB, IB, QB, MB, SMB, LB, *VD, *AC, SB, *LD	BYTE

Операнд формата (FMT) для команды ITA (Integer to ASCII [Целое в ASCII]) описан на рис. 9–48. Размер выходного буфера всегда равен 8 байтам. Количество цифр справа от десятичной точки в выходном буфере определяется полем nnn. Допустимый диапазон поля nnn составляет от 0 до 5. Задание 0 цифр справа от десятичной точки приводит к тому, что число отображается без десятичной точки. Для значений nnn, больших 5, выходной буфер заполняется пробелами ASCII. Бит с определяет использование запятой (с=1) или десятичной точки (с=0) в качестве разделителя целой и дробной части. Старшие 4 бита должны быть нулями.

Выходной буфер форматируется в соответствии со следующими правилами:

- 1. Положительные числа записываются в выходной буфер без знака.
- 2. Отрицательные числа записываются в выходной буфер с ведущим знаком минус (-).
- 3. Нули в старших разрядах слева от десятичной точки (кроме цифры, смежной с десятичной точкой) подавляются.
- 4. Числа в выходном буфере выравниваются вправо.

Рис. 9–48 дает примеры чисел, форматированных с помощью десятичной точки (с = 0), с тремя цифрами справа от десятичной точки (nnn = 011).

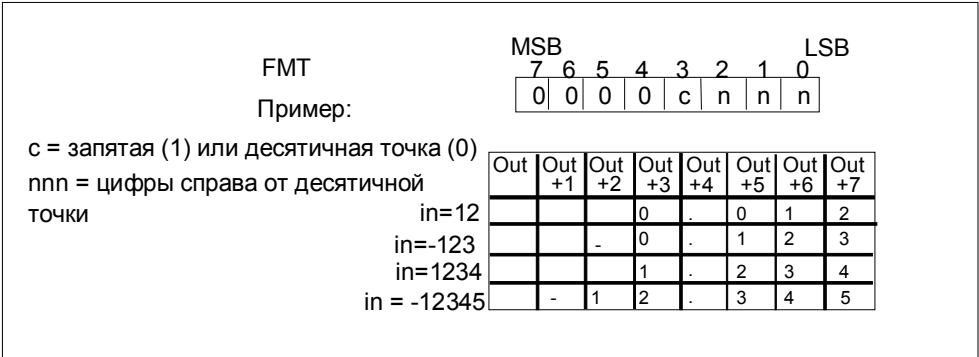
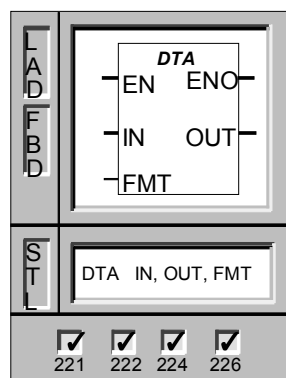


Рис. 9–48. Операнд FMT для команды ITA

Преобразование двойного целого числа в строку ASCII



Команда **Преобразовать двойное целое число в строку ASCII** преобразует двойное слово (IN) в строку ASCII. Операнд формат (FMT) определяет точность преобразования справа от десятичной точки, а также форму представления десятичной точки – в виде запятой или точки. Результат преобразования помещается в 12 последовательных байтов, начиная с адреса OUT.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация), нет выхода (недопустимый формат)

Входы/выходы	Операнды	Типы данных
IN	VD, ID, QD, MD, SD, SMD, LD, HC, константа, AC, *VD, *AC, *LD	DINT
FMT	VB, IB, QB, MB, SMB, LB, AC, константа, *VD, *AC, SB, *LD	BYTE
OUT	VB, IB, QB, MB, SMB, LB, *VD, *AC, SB, *LD	BYTE

Операнд формата (FMT) для команды DTA описан на рис. 9–49. Размер выходного буфера всегда равен 12 байтам. Количество цифр справа от десятичной точки в выходном буфере определяется полем ппп. Допустимый диапазон поля ппп составляет от 0 до 5. Задание 0 цифр справа от десятичной точки приводит к тому, что число отображается без десятичной точки. Для значений ппп, больших 5, выходной буфер заполняется пробелами ASCII. Бит с определяет использование запятой (с=1) или десятичной точки (с=0) в качестве разделителя целой и дробной части. Старшие 4 бита должны быть нулями. Выходной буфер форматируется в соответствии со следующими правилами:

1. Положительные числа записываются в выходной буфер без знака.
2. Отрицательные числа записываются в выходной буфер с ведущим знаком минус (-).
3. Нули в старших разрядах слева от десятичной точки (кроме цифры, смежной с десятичной точкой) подавляются.
4. Числа в выходном буфере выравниваются вправо.

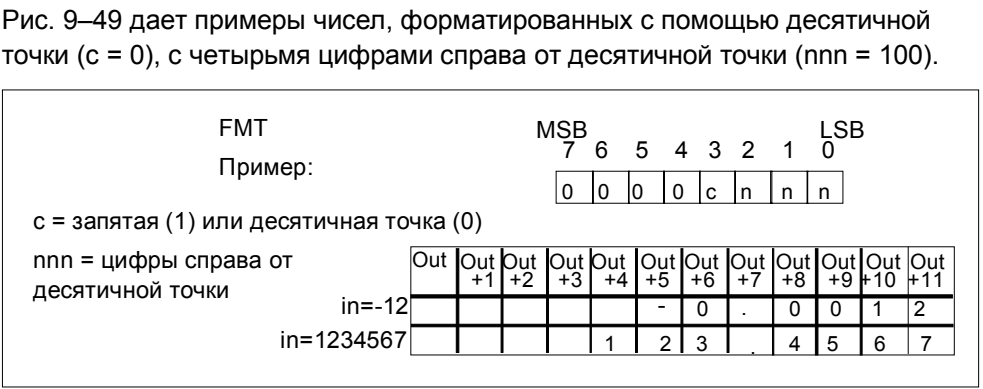
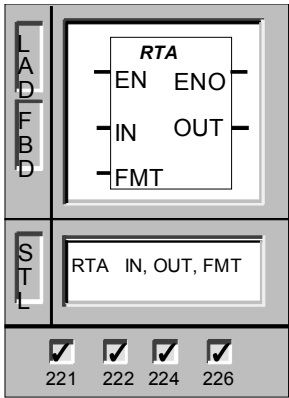


Рис. 9–49. Операнд FMT для команды DTA

Преобразование вещественного числа в строку ASCII



Команда **Преобразовать вещественное число в строку ASCII** преобразует число с плавающей точкой (IN) в строку ASCII. Формат (FMT) определяет точность преобразования справа от десятичной точки, а также форму представления десятичной точки – в виде запятой или точки и размер выходного буфера. Результат преобразования помещается в выходном буфере, начиная с адреса OUT. Длина результирующей строки ASCII определяется размером выходного буфера и может быть задана в диапазоне от 3 до 15.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация), нет выхода (недопустимый формат или буфер слишком мал)

Входы/выходы	Операнды	Типы данных
IN	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD	REAL
FMT	VB, IB, QB, MB, SMB, LB, AC, константа, *VD, *AC, SB, *LD	BYTE
OUT	VB, IB, QB, MB, SMB, LB, *VD, *AC, SB, *LD	BYTE

Операнд формата (FMT) для команды RTA описан на рис. 9–50. Размер выходного буфера определяется полем ssss. Размеры 0, 1 или 2 байта недопустимы. Количество цифр справа от десятичной точки в выходном буфере определяется полем nnn. Допустимый диапазон поля nnn составляет от 0 до 5. Задание 0 цифр справа от десятичной точки приводит к тому, что число отображается без десятичной точки. Выходной буфер заполняется пробелами ASCII для значений nnn, больших 5, или если заданный выходной буфер слишком мал для хранения преобразованного значения. Бит c определяет использование запятой (c=1) или десятичной точки (c=0) в качестве разделителя целой и дробной части. Выходной буфер форматируется в соответствии со следующими правилами:

1. Положительные числа записываются в выходной буфер без знака.
2. Отрицательные числа записываются в выходной буфер с ведущим знаком минус (-).
3. Нули в старших разрядах слева от десятичной точки (кроме цифры, смежной с десятичной точкой) подавляются.
4. Значение справа от десятичной точки округляется в соответствии с заданным количеством цифр справа от десятичной точки.
5. Размер выходного буфера должен по крайней мере на три байта превышать количество цифр справа от десятичной точки.
6. Числа в выходном буфере выравниваются вправо.

Рис.9–50 дает примеры чисел, форматированных с помощью десятичной точки (c = 0), с одной цифрой справа от десятичной точки (nnn=001) и размером буфера, равным 6 байтам (ssss=0110).

MSB	7	6	5	4	3	2	1	0	LSB
	s	s	s	s	c	n	n	n	

ssss = размер выходного буфера	in = 1234.5
c = запятая (1) или десятичная точка (0)	in = -0.0004
nnn = цифры справа от десятичной точки	in = -3.67526
	in = 1.95

Out	Out +1	Out +2	Out +3	Out +4	Out +5
1	2	3	4	.	5
			0	.	0
		-	3	.	7
			2	.	0

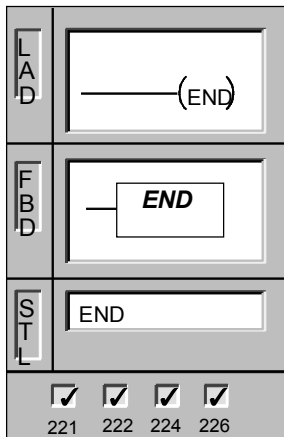
Рис. 9–50. Операнд FMT для команды RTA

Примечание

Формат чисел с плавающей точкой, используемый CPU S7-200, поддерживает не более значащих цифр. Попытка отобразить более 7 значащих цифр приводит к ошибке округления.

9.16 Команды SIMATIC для управления программой

Условное завершение



Команда **Условное завершение** завершает главную программу пользователя в зависимости от результата предшествующей логической операции.

Операнды: Нет

Типы данных: Нет

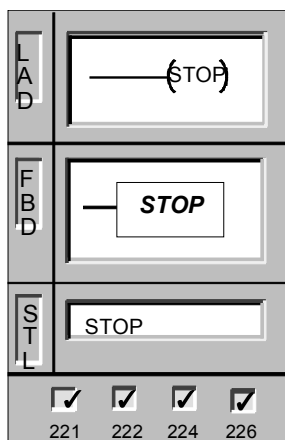
Примечание

Вы можете использовать команду «Условное завершение» в главной программе, но ее нельзя использовать в подпрограммах и программах обработки прерываний.

Примечание

STEP 7-Micro/WIN 32 автоматически добавляет к главной программе пользователя команду безусловного завершения.

STOP

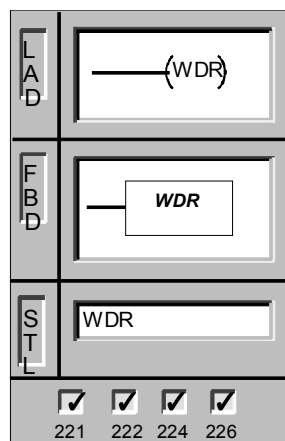


Команда **STOP** завершает выполнение программы, вызывая переход CPU из RUN в STOP.

Операнды: Нет

Если команда STOP выполняется в программе обработки прерывания, то эта программа завершается немедленно, а все прерывания, стоящие в очереди, игнорируются. Оставшиеся действия в текущем цикле обработки программы завершаются, включая выполнение главной программы пользователя, а переход из RUN в STOP производится в конце текущего цикла.

Сброс контроля времени



Команда **Сбросить контроль времени** позволяет перезапустить системный таймер контроля времени CPU. Это увеличивает время, которое может занимать цикл обработки программы, не вызывая ошибки контроля времени.

Операнды:

Нет

Рекомендации по использованию команды WDR для сброса таймера контроля времени

Команду сброса контроля времени следует использовать с осторожностью. Если вы с помощью программных циклов препятствуете завершению цикла обработки программы или существенно задерживаете его завершение, то следует иметь в виду, что до завершения цикла обработки программы запрещены следующие процессы.

- связь (за исключением режима свободно программируемой связи)
- актуализация входов и выходов (кроме входов и выходов с непосредственным доступом)
- актуализация принудительно задаваемых значений
- актуализация битов специальной памяти (SM0, SM5 ÷ SM29 не обновляются)
- диагностика в режиме реального времени
- 10–миллисекундные и 100–миллисекундные таймеры не накапливают время должным образом для циклов обработки программы, превышающих 25 мс
- команда STOP при использовании в программе обработки прерывания

Примечание

Команду WDR следует использовать для перезапуска таймера контроля времени, если вы ожидаете, что время цикла обработки программы превысит 300 мс, или вы ожидаете увеличения активности прерываний, что может воспрепятствовать возвращению в главный цикл более чем на 300 мс.

Перевод переключателя CPU в положение STOP вызывает переход CPU в режим STOP в течение 1,4 секунды.

Пример команд STOP, END и WDR

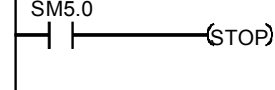
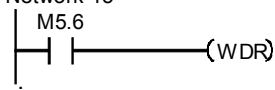
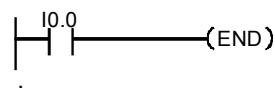
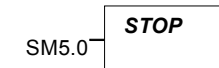
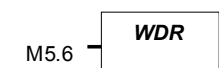
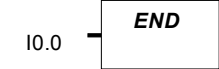
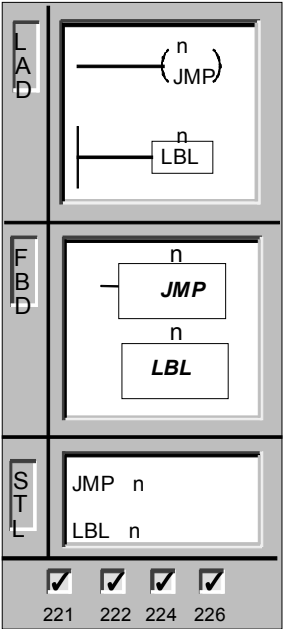
LAD	STL
<div>Network 1</div> <div></div> <div>При обнаружении ошибки ввода/вывода перейти принудительно в режим STOP.</div> <div>...</div> <div>Network 15</div> <div></div> <div>При включении M5.6 перезапустить контроль времени цикла (WDR), чтобы разрешить увеличение времени цикла.</div> <div>...</div> <div>Network 78</div> <div></div> <div>При включении IO.0 завершить главную программу.</div> <div>...</div>	<div>Network 1</div> <div>LD SM5.0</div> <div>STOP</div> <div>...</div> <div>Network 15</div> <div>LD M5.6</div> <div>WDR</div> <div>...</div> <div>Network 78</div> <div>LD IO.0</div> <div>END</div>
FBD	
<div>Network 1</div> <div></div> <div>При обнаружении ошибки ввода/вывода перейти принудительно в режим STOP.</div> <div>Network 15</div> <div></div> <div>При включении M5.6 перезапустить контроль времени цикла (WDR), чтобы разрешить увеличение времени цикла.</div> <div>Network 78</div> <div></div> <div>При включении IO.0 завершить главную программу.</div>	

Рис. 9-51. Пример команд STOP, END и WDR для SIMATIC LAD, STL и FBD

Переход на метку, метка



Команда **Перейти на метку** осуществляет переход к указанной метке (n) внутри программы. При выполнении перехода вершина стека всегда имеет значение 1.

Команда **Метка** отмечает положение цели перехода (n).

Операнды: n: константа (от 0 до 255)

Типы данных: WORD

Команда перехода и соответствующая метка должны находиться в главной программе, подпрограмме или программе обработки прерывания. Вы не можете перейти из главной программы на метку в подпрограмме или в программе обработки прерывания. Аналогично, вы не можете перейти из подпрограммы или программы обработки прерывания на метку вне этой подпрограммы или программы обработки прерывания.

Пример перехода на метку

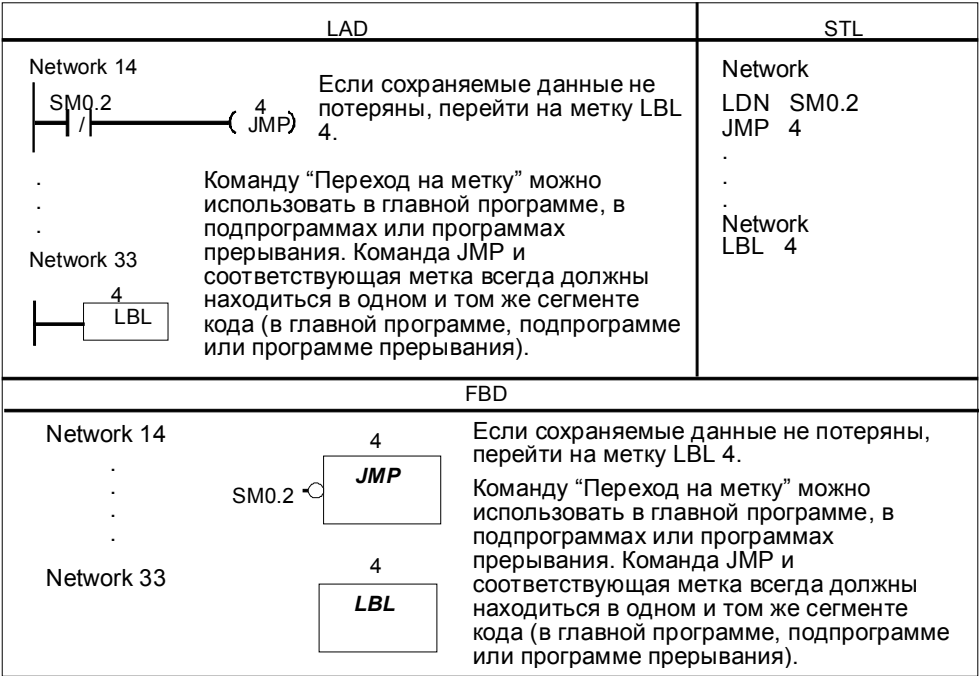
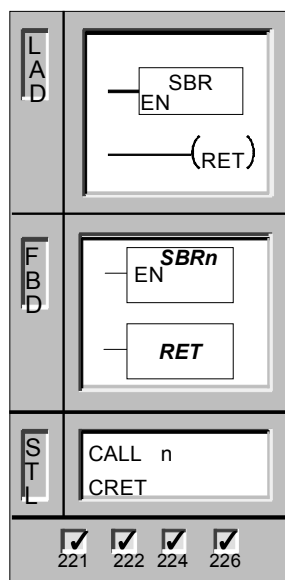


Рис. 9-52. Пример команд «Переход на метку» и «Метка» для SIMATIC LAD, STL и FBD

Подпрограмма, возврат из подпрограммы



Команда **Вызвать подпрограмму** передает управление подпрограмме (n). Команду Вызвать подпрограмму можно использовать с параметрами или без них. Для добавления подпрограммы выберите из меню **Edit → Insert → Subroutine [Редактировать → Вставить → Подпрограмма]**.

Команда **Условный возврат из подпрограммы** используется для завершения подпрограммы в зависимости от предшествующей логической операции.

Операнды: Нет

Типы данных: Нет

Как только исполнение подпрограммы завершается, управление возвращается команде, следующей за вызовом подпрограммы.

На рис. 9–55 показан пример команд «Вызвать подпрограмму» и «Вернуться из подпрограммы».

Ошибки, устанавливающие ENO в 0 для вызова подпрограммы с параметрами: SM4.3 (ошибка этапа выполнения), 0008 (превышена максимальная вложенность для подпрограмм)

Примечание

STEP 7-Micro/WIN 32 автоматически добавляет команду возврата из каждой подпрограммы.

В главной программе вы можете вкладывать подпрограммы друг в друга (помещать вызов подпрограммы внутри другой подпрограммы) на глубину до восьми уровней. В программе обработки прерывания вложение подпрограмм друг в друга невозможно. Подпрограмма не может быть помещена ни в какую другую подпрограмму, вызываемую из программы обработки прерывания. Рекурсия (вызов подпрограммы, вызывающей саму себя) не запрещена, но с подпрограммами ее следует использовать с осторожностью.

Когда вызывается подпрограмма, весь логический стек сохраняется, вершина стека устанавливается в единицу, все остальные ячейки стека устанавливаются в ноль и управление передается вызываемой подпрограмме. Когда эта подпрограмма завершается, стек восстанавливается со значениями, сохраненными в точке вызова, а управление возвращается в вызывающую программу.

Аккумуляторы являются общими для подпрограмм и вызывающей программы. При использовании подпрограммы операции сохранения и восстановления к аккумуляторам не применяются.

Вызов подпрограммы с параметрами

Подпрограмма может содержать передаваемые параметры. Параметры определяются в таблице локальных переменных подпрограммы (рис. 9–53). Параметру должно быть назначено символическое имя (не более 8 символов), тип переменной и тип данных. В подпрограмму и из нее может быть передано шестнадцать параметров.

Поле типа переменной в таблице локальных переменных определяет, передается ли переменная в подпрограмму (IN), в подпрограмму и из нее (IN_OUT), или она передается из подпрограммы (OUT). Типы параметров имеют следующие характеристики:

- **IN:** параметры передаются в подпрограмму. Если параметр является прямым адресом (например, VB10), то переменная с указанным адресом передается в подпрограмму. Если параметр является косвенным адресом (например, *AC1), то в подпрограмму передается значение, расположенное по указанному адресу. Если параметр является константой (16#1234) или адресом (VB100), то в подпрограмму передается значение константы или адреса.
- **IN_OUT:** значение, находящееся по указанному адресу параметра, передается в подпрограмму, а результирующее значение возвращается по тому же самому адресу. Константы (например, 16#1234) и адреса (например, &VB100) не могут быть параметрами типа IN_OUT.
- **OUT:** результирующее значение из подпрограммы возвращается по указанному адресу параметра. Константы (например, 16#1234) и адреса (например, &VB100) не могут быть параметрами типа OUT.
- **TEMP:**
Локальная память, не используемая для передаваемых параметров, может использоваться для временного хранения данных внутри подпрограммы.

Для добавления параметра поместите курсор на поле того типа параметров (IN, IN_OUT, OUT), который вы хотите добавить. Щелкните правой кнопкой мыши, чтобы вызвать меню опций. Выберите опцию Insert [Вставить] и опцию Row Below [Строка снизу]. Под текущей записью появится место для записи еще одного параметра выбранного типа.

	Name [Имя]	Var. Type [Тип пер.]	Data Type [Тип данн.]	Comment [Комментарий]
	EN	IN	BOOL	
L0.0	IN1	IN	BOOL	
LB1	IN2	IN	BYTE	
LB2.0	IN3	IN	BOOL	
LD3	IN4	IN	DWORD	
LW7	IN/OUT1	IN_OUT	WORD	
LD9	OUT1	OUT	DWORD	
		TEMP		

Рис. 9–53. Таблица локальных переменных STEP 7-Micro/WIN 32

Поле типа данных в таблице локальных переменных определяет размер и формат параметра. Типами данных являются:

- Power Flow [Поток сигнала]: Булев поток сигнала разрешен только для битовых (булевых) входов. Это описание сообщает STEP 7–Micro/WIN 32, что этот входной параметр является результатом достижения подпрограммы потоком сигнала, основанным на комбинации битовых логических операций. Входы с булевым потоком сигнала должны находиться в таблице локальных переменных перед любым другим типом входов. Таким способом можно использовать только входные параметры. Разрешающий вход (EN) и вход IN1 на рис. 9–54 используют булеву логику.
- BOOL – Этот тип данных используется для отдельных битовых входов и выходов. IN2 на рис. 9–54 является булевым входом.
- BYTE, WORD, DWORD – Эти типы данных определяют входной или выходной параметр без знака размером 1, 2 или 4 байта соответственно.
- INT, DINT - Эти типы данных определяют входной или выходной параметр со знаком размером 2 или 4 байта соответственно.
- REAL – Этот тип данных определяет число с плавающей точкой IEEE однократной точности (4 байта).

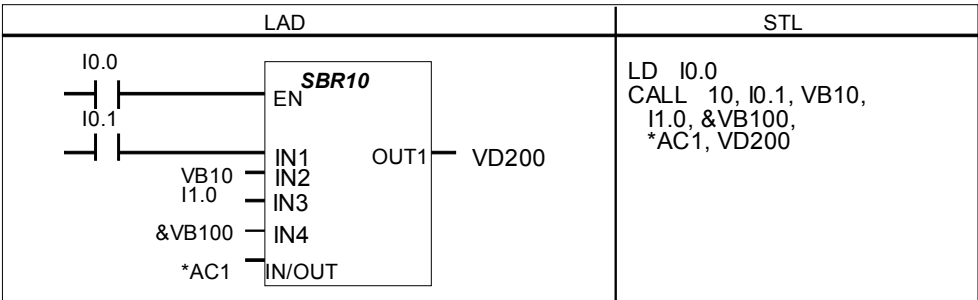


Рис. 9–54. Вызов подпрограммы в LAD и STL

Адресные параметры, например, IN4 на рис. 9–54 (&VB100), передаются в подпрограмму как DWORD (двойное слово без знака). Тип постоянного параметра должен быть указан для параметра в вызывающей программе с помощью описателя константы перед значением константы. Например, чтобы передать в качестве параметра константу, имеющую размер двойного слова без знака, со значением 12 345, постоянный параметр должен быть задан как DW#12345. Если описатель константы для параметра опущен, то константа может быть воспринята как имеющая другой тип.

Автоматическое преобразование типов для входных и выходных параметров не производится. Например, если таблица локальных переменных указывает, что параметр имеет тип данных REAL, а вызывающая программа задает для этого параметра двойное слово (DWORD), то это значение в подпрограмме будет рассматриваться как двойное слово.

Когда значения передаются в подпрограмму, они помещаются в локальную память подпрограммы. Самый левый столбец таблицы локальных переменных (см. рис. 9–53) показывает адрес в локальной памяти для каждого передаваемого параметра. Значения входных параметров копируются в локальную память подпрограммы, когда подпрограмма вызывается. Значения выходных параметров копируются из локальной памяти подпрограммы в указанные адреса выходных параметров, когда исполнение подпрограммы завершается.

Размер и тип элемента данных представляются в коде параметра. Значения параметров ставятся в соответствие локальной памяти в подпрограмме следующим образом:

- Значения параметров ставятся в соответствие локальной памяти в порядке, задаваемом командой вызова подпрограммы с параметрами, начиная с L.0.
- От одного до восьми последовательных битовых значений параметров ставятся в соответствие отдельному байту, начиная с Lx.0 и вплоть до Lx.7.
- Значения, имеющие тип байт, слово или двойное слово ставятся в соответствие локальной памяти на границах байтов (LBx, LWx или LDx).

В команде вызова подпрограммы с параметрами параметры должны расположены быть расположены в следующем порядке: сначала входные параметры, за ними параметры типа IN_OUT, а затем выходные параметры.

Если вы программируете на STL, то формат команды CALL имеет вид:

CALL номер подпрограммы, параметр 1, параметр 2, ... , параметр n

Ошибки, устанавливающие ENO в 0 для вызова подпрограммы с параметрами: SM4.3 (ошибка этапа выполнения), 0008 (превышена максимальная вложенность для подпрограмм)

Пример подпрограммы и возвращения из подпрограммы


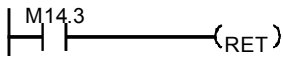
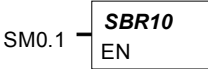
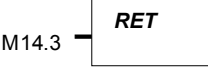
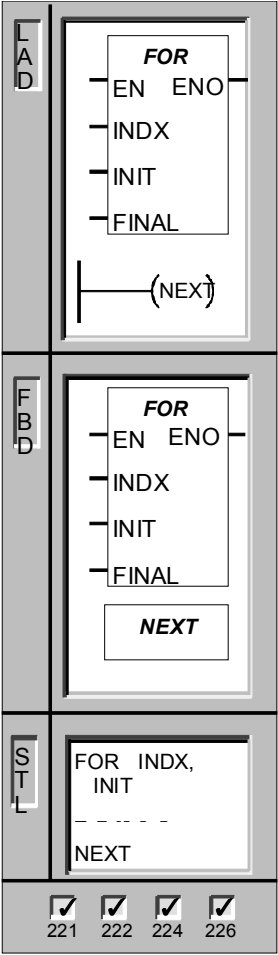
LAD		STL
MAIN [Главная программа]		
<div>Network 1</div> <div></div> <div>В первом цикле: Вызвать SBR10 для инициализации.</div>		<div>Network 1</div> <div>LD SM0.1</div> <div>CALL 10</div> <div>.</div>
SUBROUTINE 10 [Подпрограмма 10]		
<div>.</div> <div>.</div> <div>Запуск подпрограммы 10</div> <div>.</div> <div>Network 6</div> <div></div> <div>Может быть использован условный возврат (RET) из подпрограммы 10.</div> <div>.</div> <div>.</div> <div>.</div>		<div>.</div> <div>.</div> <div>Network 6</div> <div>LD M14.3</div> <div>CRET</div> <div>.</div> <div>.</div> <div>.</div>
FBD		
MAIN [Главная программа]		
		
SUBROUTINE 10 [Подпрограмма 10]		
		

Рис. 9-55. Пример команд вызова и возврата из подпрограммы для SIMATIC LAD, STL и FBD

FOR и NEXT



Команда **FOR** выполняет команды, расположенные между операторами FOR и NEXT. Вы должны задать значение индекса или счетчик цикла (INDX), начальное значение (INIT) и конечное значение (FINAL).

Команда **NEXT** отмечает конец цикла FOR и устанавливает вершину стека в 1.

Например, если значение INIT равно 1, значение FINAL равно 10, то команды между FOR и NEXT исполняются 10 раз, причем значение INDX каждый раз увеличивается на единицу: 1, 2, 3, ...10.

Если начальное значение больше конечного, то цикл не выполняется. После каждого исполнения команд между FOR и NEXT значение INDX увеличивается, а результат сравнивается с конечным значением. Если INDX больше конечного значения, то цикл завершается.

FOR: Ошибки, устанавливающие ENO в 0:
SM4.3 (ошибка этапа выполнения), 0006 (косвенная адресация)

Входы/выходы	Операнды	Типы данных
INDX	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, *VD, *AC, *LD	INT
INIT	VW, IW, QW, MW, SW, SMW, T, C, AC, LW, AIW, константа, *VD, *AC, *LD	INT
FINAL	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, AIW, константа, *VD, *AC, *LD	INT

Вот некоторые указания по использованию цикла FOR/NEXT:

- Если вы разблокируете FOR/NEXT, то процесс циклического выполнения продолжается, пока не закончатся итерации, если только вы не измените конечное значение изнутри самого цикла. Вы можете изменять эти значения, пока FOR/NEXT выполняют циклическую обработку.
- Когда цикл снова разблокирован, он копирует начальное значение в индекс (счетчик цикла). Команда FOR/NEXT сбрасывает себя каждый раз, когда она разблокируется.

Используйте команды FOR/NEXT для описания цикла, который повторяется заданное количество раз. Каждая команда FOR требует наличия команды NEXT. Вы можете вкладывать циклы FOR/NEXT друг в друга (помещать цикл FOR/NEXT внутри другого цикла FOR/NEXT). Глубина вложения не может превышать восьми.

Пример цикла FOR/NEXT

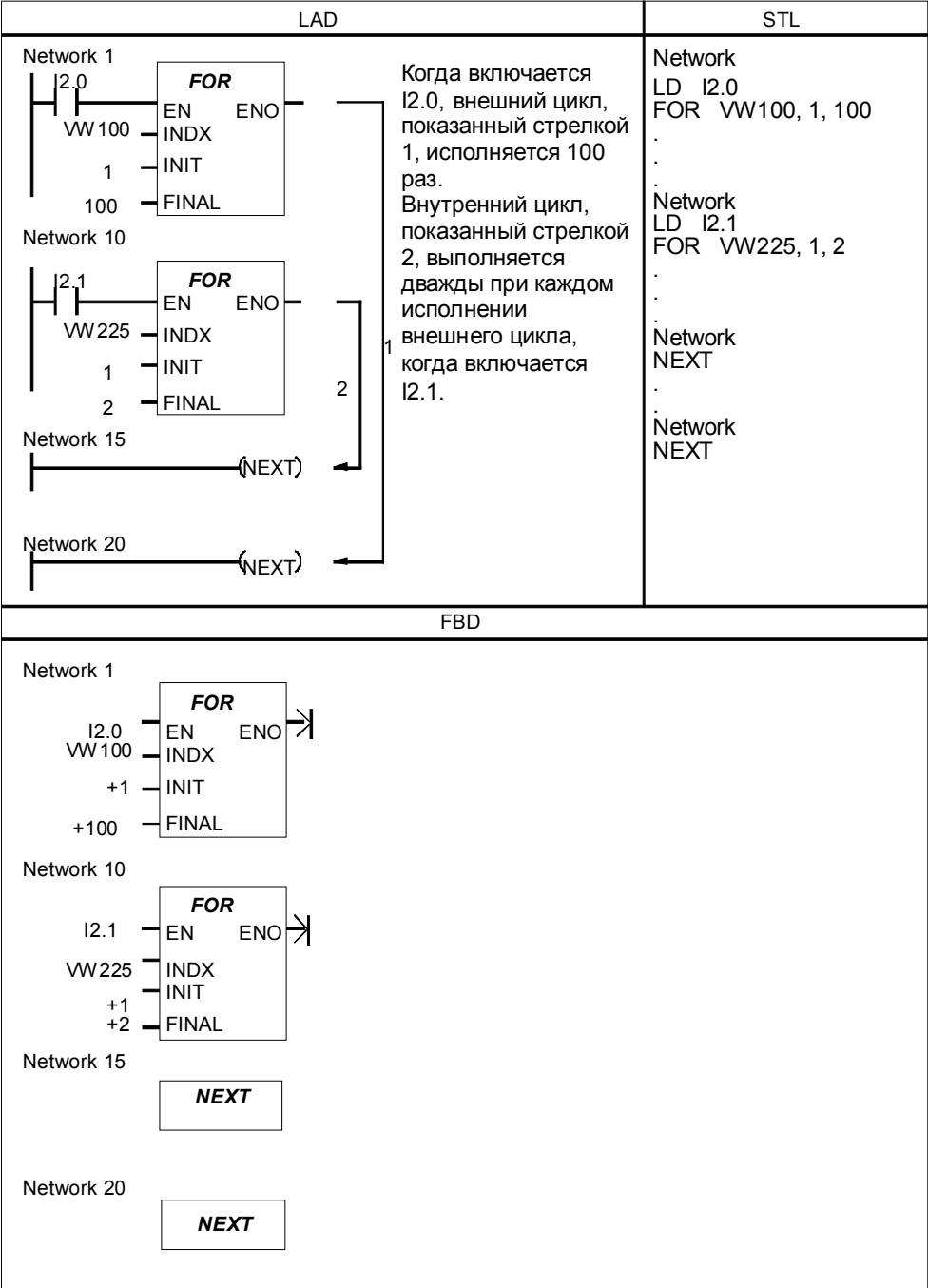
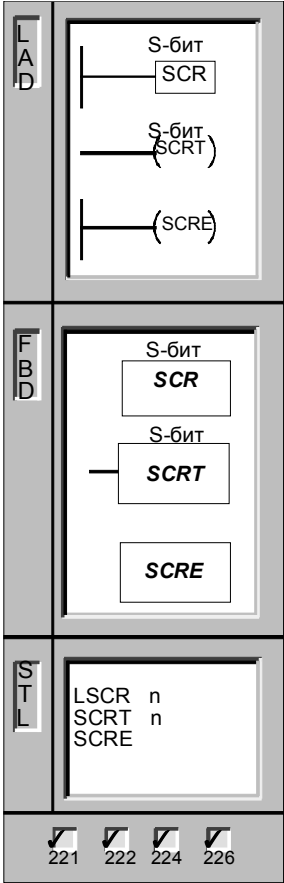


Рис. 9-56. Пример команд FOR/NEXT для SIMATIC LAD, STL и FBD

Реле управления последовательностью



Команда **Загрузить реле управления последовательностью (LSCR)** отмечает начало сегмента SCR. Когда n равно 1, поток сигнала пропускается к сегменту SCR. Сегмент SCR должен завершаться командой SCRE.

Команда **Перейти к следующему реле управления последовательностью (SCRT)** определяет бит SCR, который должен быть разблокирован (следующий S-бит, который должен быть установлен). Когда поток сигнала достигает катушки или блока FBD, S-бит, к которому производится обращение, устанавливается, а S-бит команды LSCR (который разблокировал этот сегмент SCR) сбрасывается.

Команда **Конец реле управления последовательностью (SCRE)** отмечает конец сегмента SCR.

Входы/выходы	Операнды	Типы данных
n	S	BOOL

Описание команд SCR

В LAD, FBD и STL реле управления последовательностью (Sequence Control Relay, SCR) используются для организации машинных операций или шагов в эквивалентные сегменты программы. SCR делают возможной логическую сегментацию программы управления.

Команда LSCR загружает стек SCR и логический стек значением S-бита, к которому обращается команда. Сегмент SCR активизируется или деактивизируется результирующим значением стека SCR. Вершина логического стека загружается в S-бит, к которому производится обращение, так что блоки или выходные катушки могут быть непосредственно связаны с левой силовой шиной без промежуточных контактов. На рис. 9–57 показаны S-стек и логический стек и влияние выполнения команды LSCR.

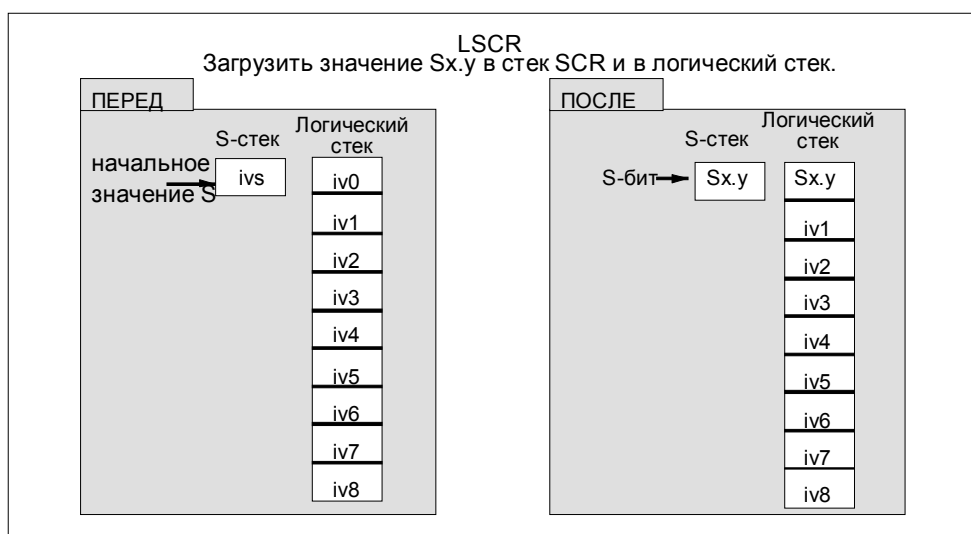


Рис. 9-57. Влияние LSCR на логический стек

Относительно команд реле управления последовательностью имеет силу следующее:

- Вся логика между командами LSCR и SCRE образует сегмент SCR, и ее исполнение зависит от значения S-стека. Логика между SCRE и следующей командой LSCR не зависит от значения S-стека.
- Команда SCRT устанавливает S-бит, чтобы разблокировать следующее SCR, а также сбрасывает S-бит, который был загружен для разблокирования данного раздела сегмента SCR.

Ограничения

Использование SCR имеет следующие ограничения:

- Нельзя использовать один и тот же S-бит более чем в одной программе. Например, если вы используете S0.1 в главной программе, не используйте его в подпрограмме.
- В сегменте SCR нельзя использовать команды JMP и LBL. Это значит, что не разрешаются переходы в, внутри и из сегмента SCR. Команды перехода и метки можно использовать для обхода сегментов SCR.
- В сегменте SCR нельзя использовать команды FOR, NEXT и END.

Пример SCR

На рис. 9–58 показан пример работы SCR.

- В этом примере бит первого цикла SM0.1 используется для установки S0.1, который в первом цикле будет в активном состоянии 1.
- После 2-секундной задержки T37 вызывает переход к состоянию 2. Этот переход деактивирует сегмент SCR для состояния 1 (S0.1) и активизирует сегмент SCR для состояния 2 (S0.2).

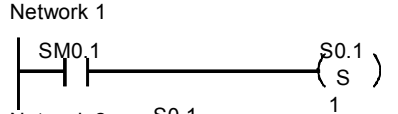
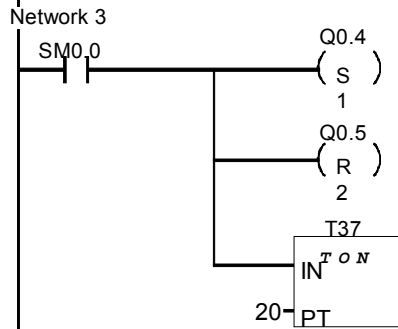
LAD		STL
Network 1		Network 1 LD SM0.1 S S0.1, 1
Network 2		Network 2 LSCR S0.1
Network 3		Network 3 LD SM0.0 S Q0.4, 1 R Q0.5, 2 TON T37, 20
Network 4		Network 4 LD T37 SCRT S0.2
Network 5		Network 5 SCRE
(Продолжение программы на следующей странице)		

Рис. 9-58. Пример реле управления последовательностью (SCR) для SIMATIC LAD, STL и FBD

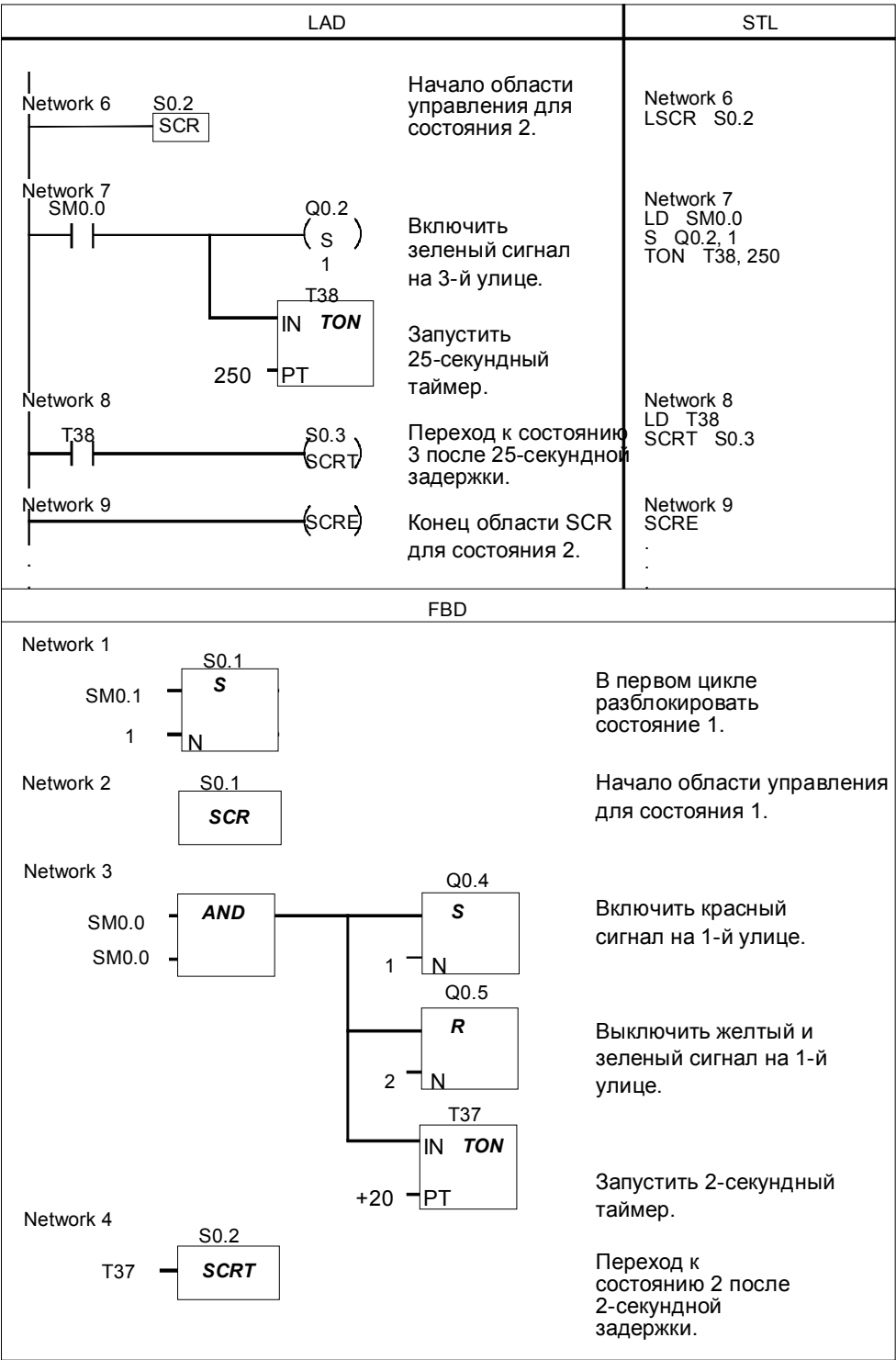


Рис. 9-58. Пример реле управления последовательностью (SCR) для SIMATIC LAD, STL и FBD (продолжение)

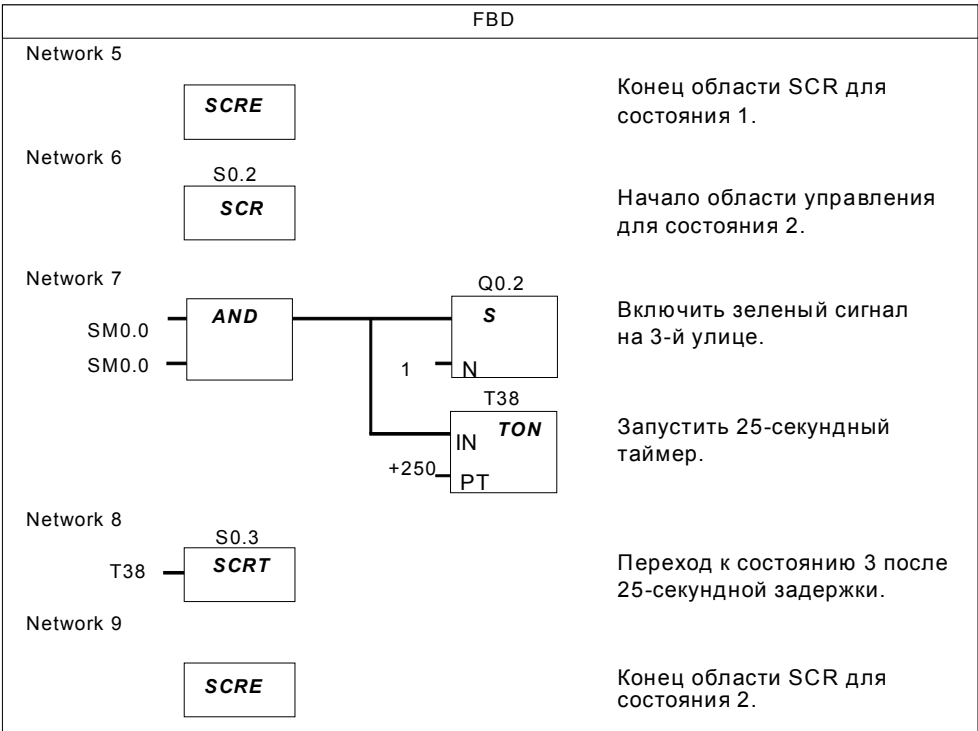


Рис. 9-58. Пример реле управления последовательностью (SCR) для SIMATIC LAD, STL и FBD (продолжение)

Разделение потока управления

Во многих приложениях единый поток состояний последовательности исполнения должен быть разделен на два или более различных потоков. Когда поток управления разделяется на несколько потоков, все выходящие потоки должны активизироваться одновременно. Это показано на рис. 9-59.

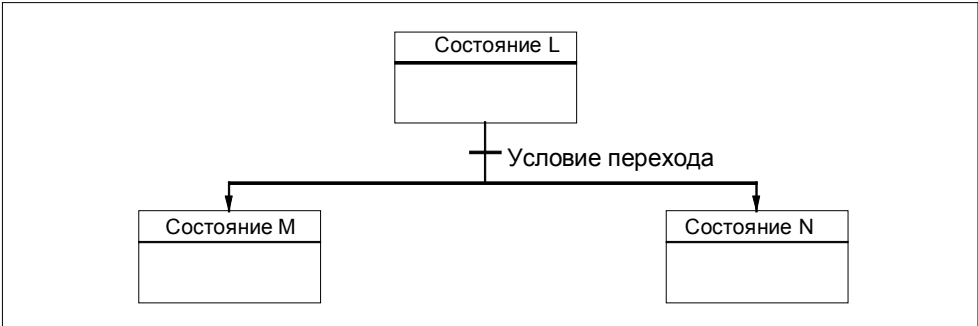


Рис. 9-59. Разделение потока управления

Разделение потоков управления может быть реализовано в программе SCR путем использования нескольких команд SCRT, разблокируемых одним и тем же условием перехода, как показано на рис. 9–60.

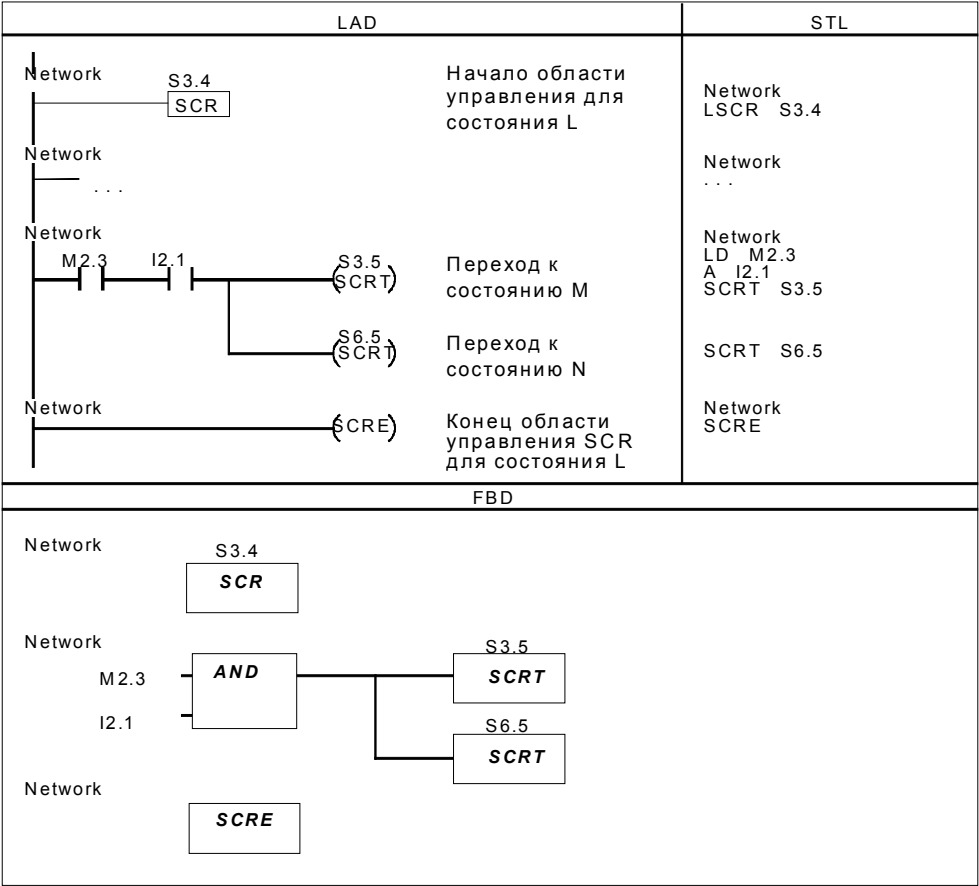


Рис. 9-60. Пример разделения потоков управления для LAD, STL и FBD

Управление слиянием

Аналогичная ситуация возникает, когда два или более потоков последовательных состояний должны быть объединены в один поток. Когда несколько потоков объединяются в один поток, говорят, что они сливаются. Когда несколько потоков сливаются в один поток, все входящие потоки должны быть завершены до того, как будет выполняться следующее состояние. На рис. 9-61 изображено слияние двух потоков управления.

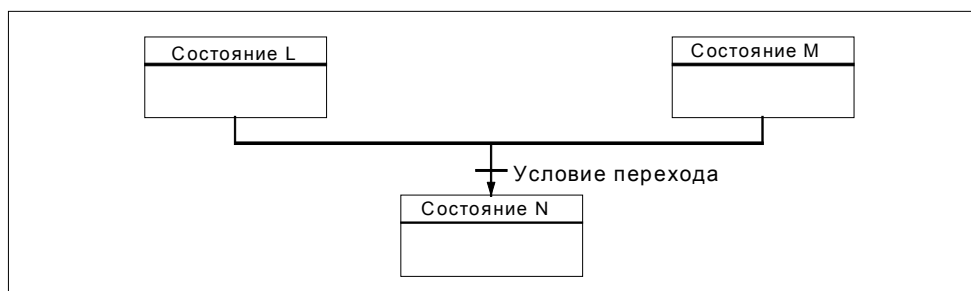


Рис. 9-61. Слияние потоков управления

Слияние потоков управления может быть реализовано в программе SCR путем создания перехода от состояния L к состоянию L' и перехода от состояния M к состоянию M'. Когда биты SCR, представляющие L' и M', установлены, состояние N может быть разблокировано, как показано на рис. 9–62.

LAD		STL
<div>Network</div> <div><div>S3.4</div><div>SCR</div></div>	Начало области управления для состояния L	Network LSCR S3.4
<div>Network</div> <div>...</div>		Network ...
<div>Network</div> <div><div>V100.5</div><div><div>S3.5</div><div>SCRT</div></div></div>	Переход к состоянию L'.	Network LD V100.5 SCRT S3.5
<div>Network</div> <div><div><div>SCR</div></div></div>	Конец области SCR для состояния L	Network SCRE
<div>Network</div> <div><div>S6.4</div><div>SCR</div></div>	Начало области управления для состояния M	Network LSCR S6.4
<div>Network</div> <div>...</div>		Network ...
<div>Network</div> <div><div>C50</div><div><div>S6.5</div><div>SCRT</div></div></div>	Переход к состоянию M'.	Network LD C50 SCRT S6.5
<div>Network</div> <div><div><div>SCR</div></div></div>	Конец области SCR для состояния M.	Network SCRE
<div>Network</div> <div><div>S3.5</div><div>S6.5</div><div><div>S5.0</div><div>(S)</div><div>1</div></div></div>	Разблокировать состояние N.	Network LD S3.5 A S6.5 S S5.0, 1
<div><div><div>S3.5</div><div>(R)</div><div>1</div></div></div>	Сбросить состояние L'.	R S3.5, 1
<div><div><div>S6.5</div><div>(R)</div><div>1</div></div></div>	Сбросить состояние M'.	R S6.5, 1

Network

S3.5

SCRE

Network

S6.4

SCR

Network

...

Network

C50

S6.5

SCRT

Network

S6.5

SCRE

Network

S3.5

S6.5

S5.0

S

1

Network

S3.5

R

1

Network

S6.5

R

1

Начало области управления для состояния L

Переход к состоянию L'.

Конец области SCR для состояния L

Начало области управления для состояния M

Переход к состоянию M'.

Конец области SCR для состояния M.

Разблокировать состояние N.

Сбросить состояние L'.

Сбросить состояние M'.

Рис. 9-62. Пример слияния потоков управления для LAD, STL и FBD

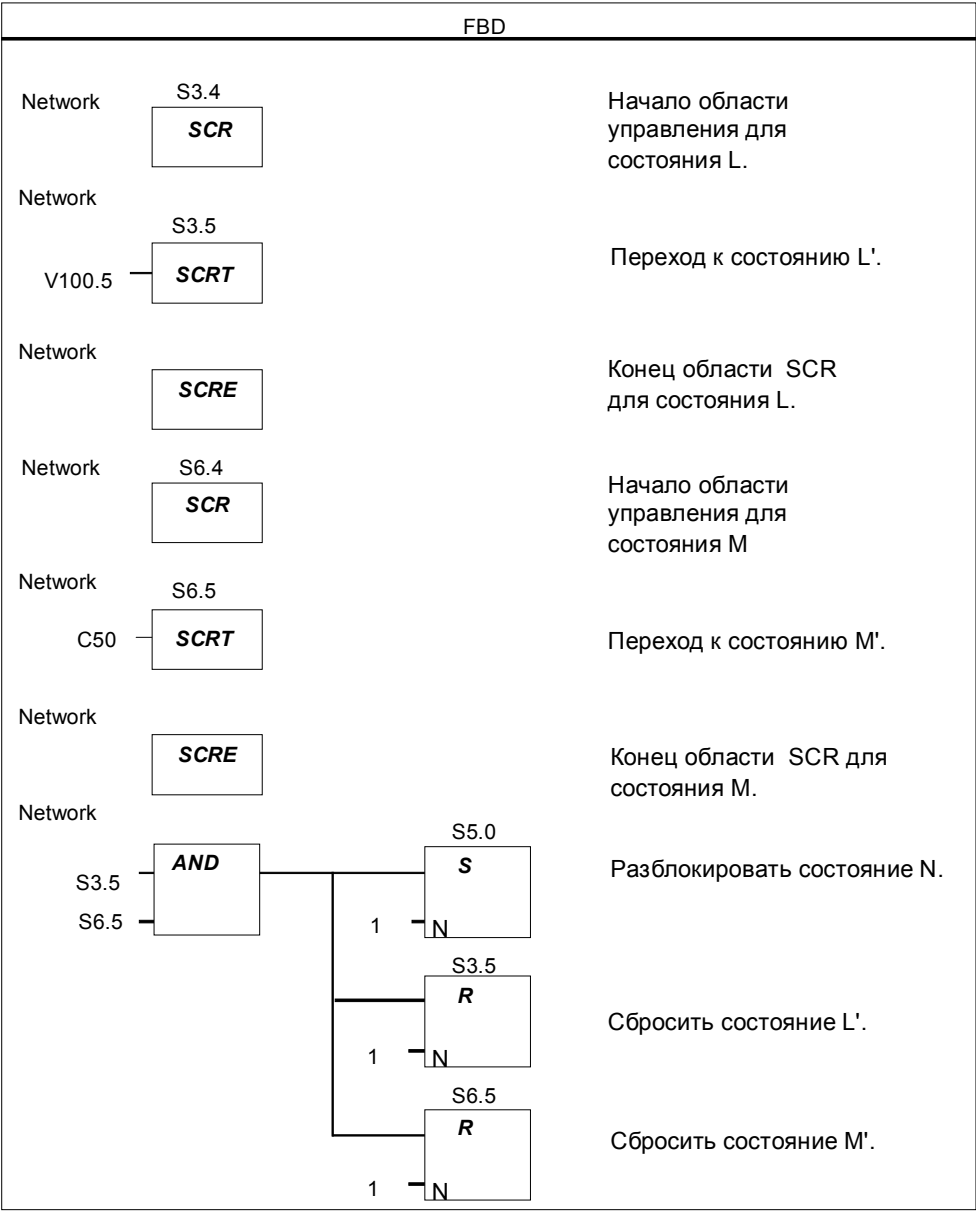


Рис. 9-62. Пример слияния потоков управления для LAD, STL и FBD (продолжение)

В других ситуациях поток управления может быть направлен в один из нескольких возможных потоков управления в зависимости от того, какое условие перехода выполнилось первым. Такая ситуация изображена на рис. 9–63.

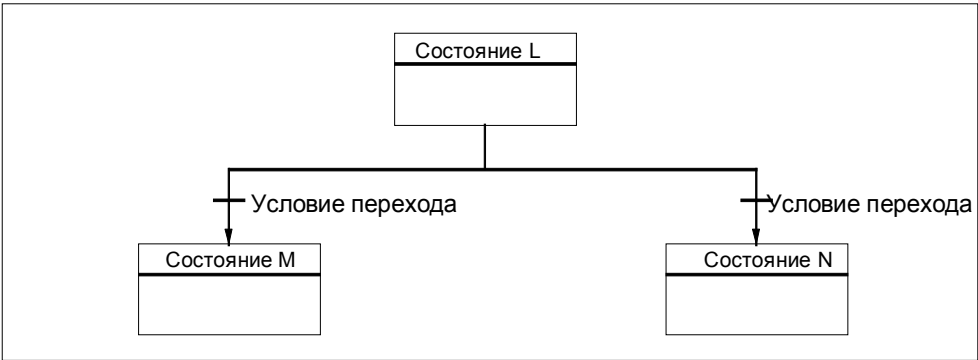


Рис. 9-63. Перенаправление потока управления в зависимости от условия перехода

Эквивалентная программа SCR показана на рис. 9–64.

LAD		STL
Network	S3.4 [SCR]	Network LSCR S3.4
Network	...	Network ...
Network	M2.3 ----- S3.5 (SCRT)	Network LD M2.3 SCRT S3.5
Network	I3.3 ----- S6.5 (SCRT)	Network LD I3.3 SCRT S6.5
Network	----- (SCRE)	Network SCRE

Рис. 9-64. Пример условных переходов для LAD, STL и FBD

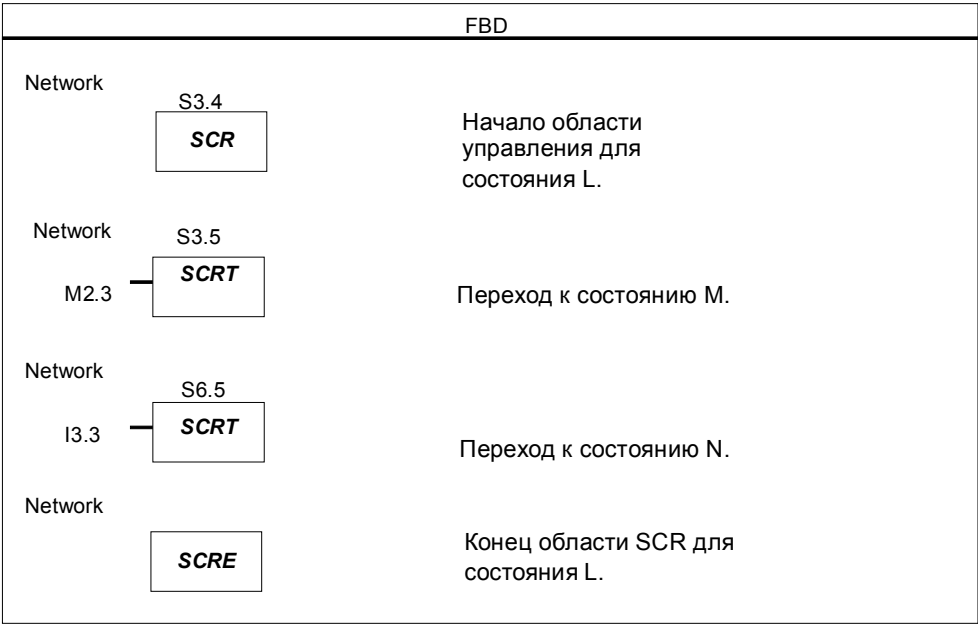
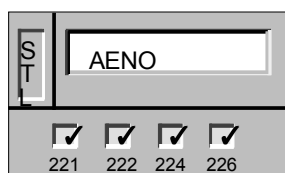


Рис. 9-64. Пример условных переходов для LAD, STL и FBD (продолжение)

ENO



ENO – это булев выход для блоков в LAD и FBD. Если блок имеет поток сигнала на входе EN и выполняется без ошибок, то выход ENO передает поток сигнала следующему элементу. ENO может быть использован как разблокирующий бит, указывающий на успешное завершение команды.

Бит ENO используется с вершиной стека для воздействия на поток сигнала для выполнения последующих команд.

У команд STL нет входа EN; чтобы команда исполнялась, вершина стека должна быть равна 1.

В STL нет выхода ENO, но команды STL, соответствующие командам LAD и FBD с выходами ENO обязательно устанавливают специальный бит ENO. Доступ к этому биту осуществляется с помощью команды **И ENO** (And ENO, AENO). AENO можно использовать для создания такого же эффекта, что и бит ENO в блоке. Команда AENO доступна только в STL.

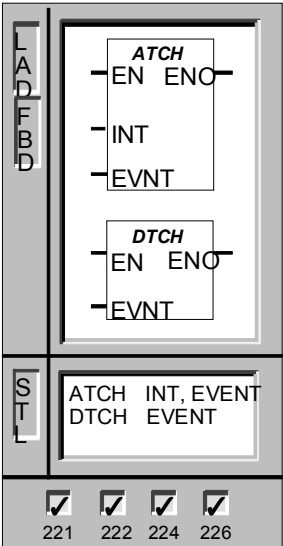
AENO выполняет логическое И бита ENO и вершины стека. Результатом операции И является новая вершина стека.

Операнды: Нет

Типы данных: Нет

9.17 Команды SIMATIC для организации прерываний и связи

Назначение и отсоединение прерывания



Команда **Назначить прерывание** связывает прерывающее событие (EVNT) с номером программы обработки прерывания (INT) и разблокирует прерывающее событие.

Команда **Отсоединить прерывание** разрывает связь прерывающего события (EVNT) со всеми программами обработки прерываний и блокирует прерывающее событие.

Назначить прерывание: Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0002 (конфликт при назначении входов для HSC).

Входы/выходы	Операнды	Типы данных
INT	константа	BYTE
EVNT	константа (CPU 221/222: 0-12, 19-23, 27-33; CPU 224: 0-23, 27-33; CPU 226: 0 - 33)	BYTE

Описание команд назначения и отсоединения прерываний

Прежде чем программа обработки прерывания может быть вызвана, должно быть установлено соответствие между прерывающим событием и сегментом программы, который вы хотите выполнить, когда это событие происходит. Для организации связи между прерывающим событием (задаваемым номером прерывающего события) и сегментом программы (задаваемым номером программы обработки прерывания) используйте команду «Назначить прерывание» (ATCH). Одной программе обработки прерываний можно поставить в соответствие несколько прерывающих событий, но одно событие не может быть одновременно поставлено в соответствие нескольким программам обработки прерываний. Когда происходит событие при разблокированных прерываниях, то исполняется только последняя программа обработки прерывания, поставленная в соответствие этому событию.

Когда вы назначаете прерывающее событие программе обработки прерывания, это прерывание автоматически разблокируется. Если вы заблокировали все прерывания с помощью команды глобального блокирования прерываний, то каждое возникновение прерывающего события ставится в очередь, пока прерывания не будут снова разблокированы с помощью глобального разблокирования прерываний.

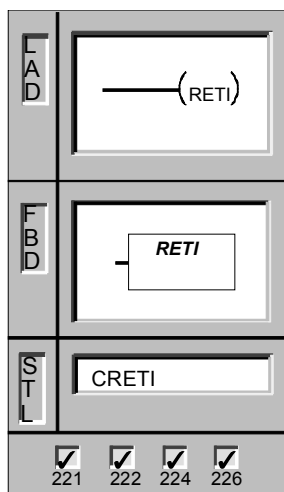
Отдельные прерывающие события можно заблокировать разрывом связи между этим прерывающим событием и программой обработки прерывания с помощью команды «Отсоединить прерывание» (DTCH). Команда отсоединения возвращает прерывание в неактивное или игнорируемое состояние.

Таблица 9–20 перечисляет различные типы прерывающих событий.

Таблица 9–20. Прерывающие события

Номер события	Описание прерывания	CPU 221	CPU 222	CPU 224	CPU 226
0	Нарастающий фронт, I0.0	Да	Да	Да	Да
1	Падающий фронт, I0.0	Да	Да	Да	Да
2	Нарастающий фронт, I0.1	Да	Да	Да	Да
3	Падающий фронт, I0.1	Да	Да	Да	Да
4	Нарастающий фронт, I0.2	Да	Да	Да	Да
5	Падающий фронт, I0.2	Да	Да	Да	Да
6	Нарастающий фронт, I0.3	Да	Да	Да	Да
7	Падающий фронт, I0.3	Да	Да	Да	Да
8	Порт 0: символ принят	Да	Да	Да	Да
9	Порт 0: передача завершена	Да	Да	Да	Да
10	Управляемое временем прерывание 0, SMB34	Да	Да	Да	Да
11	Управляемое временем прерывание 1, SMB35	Да	Да	Да	Да
12	HSC0: CV=PV (текущее значение = предустановленному)	Да	Да	Да	Да
13	HSC1: CV=PV (текущее значение = предустановленному)			Да	Да
14	HSC1: направление изменено			Да	Да
15	HSC1: внешний сброс			Да	Да
16	HSC2: CV=PV (текущее значение = предустановленному)			Да	Да
17	HSC2: направление изменено			Да	Да
18	HSC2: внешний сброс			Да	Да
19	PLS0: отсчет количества импульсов завершен	Да	Да	Да	Да
20	PLS1: отсчет количества импульсов завершен	Да	Да	Да	Да
21	Таймер T32: прерывание CT=PT	Да	Да	Да	Да
22	Таймер T96: прерывание CT=PT	Да	Да	Да	Да
23	Порт 0: прием сообщения завершен	Да	Да	Да	Да
24	Порт 1: прием сообщения завершен				Да
25	Порт 1: символ принят				Да
26	Порт 1: передача завершена				Да
27	HSC0: направление изменено	Да	Да	Да	Да
28	HSC0: внешний сброс	Да	Да	Да	Да
29	HSC4: CV=PV (текущее значение = предустановленному)	Да	Да	Да	Да
30	HSC4: направление изменено	Да	Да	Да	Да
31	HSC4: внешний сброс	Да	Да	Да	Да
32	HSC3: CV=PV (текущее значение = предустановленному)	Да	Да	Да	Да
33	HSC5: CV=PV (текущее значение = предустановленному)	Да	Да	Да	Да

Условный возврат из программы обработки прерывания



Команда **Условный возврат из программы обработки прерывания** может быть использована для возврата из программы обработки прерывания в зависимости от условия, задаваемого предшествующей логикой.

Операнды: Нет

Типы данных: Нет

На экране STEP 7-Micro/WIN 32 возврат из программ обработки прерываний обозначается отдельными программными метками.

Программы обработки прерываний

Программа обработки прерывания выполняется в ответ на соответствующее внутреннее или внешнее событие. После выполнения последней команды программы обработки прерывания управление возвращается в главную программу. Вы можете покинуть эту программу, выполнив команду «Условный возврат из прерывания» (CRETI).

Рекомендации по использованию прерываний

Обработка прерываний обеспечивает быструю реакцию на определенные внутренние или внешние события. Вам следует оптимизировать программы обработки прерываний, чтобы выполнить конкретную задачу, а затем вернуть управление главной программе. Если программа обработки прерывания спроектирована короткой с точными спецификациями, то она будет быстро выполняться и не будет задерживать другие процессы на длительные промежутки времени. Если этого не сделать, то неожиданные условия могут вызвать ненормальную работу оборудования, управляемого главной программой. Для прерываний безусловно верна аксиома «чем короче, тем лучше».

Ограничения

В программе обработке прерывания нельзя использовать команды DISI, ENI, HDEF, LSCR и END.

Системная поддержка прерываний

Так как прерывания могут оказывать влияние на контакты, катушки и аккумуляторы, то система сохраняет и перезагружает логический стек, аккумуляторные регистры и биты специальной памяти (SM), которые отображают состояние аккумуляторов и команд. Это позволяет избежать искажения главной программы пользователя из-за перехода в программу обработки прерывания и возвращения из нее.

Вызов подпрограммы из программы обработки прерывания

Из программы обработки прерывания можно вызвать только один уровень вложенности подпрограмм. Аккумуляторы и логический стек совместно используются программой обработки прерывания и вызываемой подпрограммой.

Совместное использование данных главной программой и программами обработки прерываний

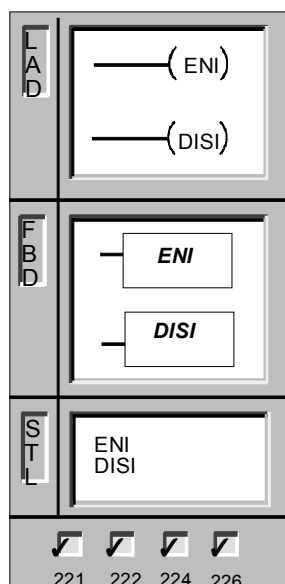
Данные могут совместно использоваться главной программой и одной или несколькими программами обработки прерываний. Например, часть вашей программы может предоставить данные для использования программой обработки прерываний и наоборот. Если в вашей программе применяется совместное использование данных, то вы должны также принять в расчет влияние асинхронной природы прерывающих событий, которые могут возникнуть в любой момент при исполнении вашей главной программы. Проблемы непротиворечивости совместно используемых данных могут возникнуть в результате действий программ обработки прерываний, когда выполнение команд вашей главной программы прерывается событиями, вызывающими прерывания.

Существует ряд методов программирования, которые вы можете использовать, чтобы обеспечить корректное разделение данных между вашей главной программой и программами обработки прерываний. Эти методы или ограничивают способ доступа к совместно используемым ячейкам памяти, или препятствуют прерыванию последовательностей команд, использующих разделяемые ячейки памяти.

- Для программы на STL, которая совместно использует единственную переменную: Если разделяемые данные представляют собой единственную переменную в виде байта, слова или двойного слова и ваша программа написана на STL, то правильный совместный доступ может быть обеспечен сохранением промежуточных значений от операций над совместно используемыми данными только в неразделяемых ячейках памяти или аккумуляторах.
- Для программы на LAD, которая совместно использует единственную переменную: Если разделяемые данные представляют собой единственную переменную в виде байта, слова или двойного слова и ваша программа написана на LAD, то правильный совместный доступ может быть обеспечен установлением соглашения, что доступ к разделяемым ячейкам памяти может осуществляться только с помощью команд передачи (MOVB, MOVW, MOVD, MOVR). В то время как многие команды LAD составлены из непрерываемых последовательностей команд STL, команды передачи состоят из единственной команды STL, на исполнение которой не могут влиять прерывающие события.

- Для программы STL или LAD, совместно использующей несколько переменных: Если разделяемые данные составлены из ряда связанных байтов, слов или двойных слов, то для управления исполнением программ обработки прерываний могут быть использованы команды блокировки/ разблокировки прерываний (DISI и ENI). В той точке вашей программы, где должны начаться операции с разделяемыми ячейками памяти, заблокируйте прерывания. Как только все действия, влияющие на совместно используемые ячейки памяти, завершены, вновь разблокируйте прерывания. В течение времени, когда прерывания заблокированы, программы обработки прерываний не могут выполняться и, следовательно, не имеют доступа к разделяемым ячейкам памяти; однако такой подход может привести к запаздыванию реакции на прерывающие события.

Блокировка и разблокировка прерываний



Команда **Разблокировать прерывания** глобально разблокирует обработку всех назначенных прерывающих событий.

Команда **Заблокировать прерывания** глобально блокирует обработку всех прерывающих событий.

Операнды: Нет

Типы данных: Нет

Когда вы переходите в режим RUN, прерывания первоначально заблокированы. Находясь в режиме RUN, вы можете разблокировать все прерывания, выполнив глобальную команду «Разблокировать прерывания». Глобальная команда «Заблокировать прерывания» дает возможность ставить прерывания в очередь, но не позволяет вызывать программы обработки прерываний.

Ошибки, устанавливающие ENO в 0: SM4.3 (ошибка этапа выполнения), 0004 (попытка исполнения команд ENI, DISI или HDEF в программе обработки прерываний).

Прерывания коммуникационных портов

Программы, написанные на LAD и STL, могут управлять последовательным коммуникационным портом программируемого логического контроллера. Этот режим работы коммуникационного порта называется свободно программируемой связью (Freepoint mode). В этом режиме ваша программа определяет скорость передачи, количество битов на символ, контроль по четности и протокол. Для облегчения программно управляемой связи в вашем распоряжении имеются прерывания приема и передачи. За дополнительной информацией обратитесь к командам приема и передачи.

Прерывания от ввода/вывода

К прерываниям от ввода/вывода относятся прерывания при нарастающем/ падающем фронте, прерывания скоростных счетчиков и прерывания от последовательности импульсов. CPU может генерировать прерывание при нарастающем и/или падающем фронте на входе. Входы, доступные для этих прерываний, приведены в таблице 9–21. Появления нарастающего и падающего фронта могут быть распознаны для любой из этих точек ввода. Эти события могут использоваться для отображения условия, которое немедленно должно быть принято во внимание, когда это событие происходит.

Таблица 9–21. Поддерживаемые прерывания при нарастающем/падающем фронте

Прерывания от ввода/вывода	CPU S7–200
Входы и выходы	от I0.0 до I0.3

Прерывания скоростных счетчиков дают вам возможность реагировать на такие условия, как достижение текущим значением предустановленного значения, изменение направления счета, которое может соответствовать реверсированию направления, в котором вращается вал, или внешний сброс счетчика. Каждое из этих прерываний дает возможность предпринимать в реальном времени действия в ответ на быстрые события, которыми нельзя управлять при скоростях, определяемых временем цикла программируемого логического контроллера.

Прерывания от последовательности импульсов немедленно извещают о завершении вывода предписанного количества импульсов. Импульсные последовательности часто используются для управления шаговыми двигателями.

Каждое из вышеописанных прерываний может быть разблокировано назначением программы обработки прерывания соответствующему событию ввода/вывода.

Прерывания, управляемые временем

К прерываниям, управляемым временем, относятся циклические прерывания и прерывания, вызываемые таймерами T32 и T96. CPU может поддерживать циклические прерывания. С помощью циклических прерываний вы можете задать действия, которые должны выполняться циклически. Время цикла устанавливается в пределах от 1 до 255 мс шагами по 1 мс. Вы должны записать время цикла в SMB34 для циклического прерывания 0 и в SMB35 для циклического прерывания 1.

Событие, вызывающее циклические прерывания, передает управление соответствующей программе обработки прерываний каждый раз, как истекает время работы таймера. Обычно циклические прерывания используются для управления опросом аналоговых входов через регулярные интервалы времени или для организации работы PID-регулятора.

Циклическое прерывание разблокируется, и начинается отсчет времени, когда вы назначаете программу обработки прерывания событию, вызывающему циклическое прерывание. При этом система воспринимает значение времени цикла, и последующие изменения на это время цикла влияния не оказывают. Чтобы изменить время цикла, вы должны задать для него новое значение, а затем снова назначить программу обработки прерывания событию, вызывающему циклическое прерывание. Когда происходит повторное назначение, функция циклического прерывания сбрасывает все накопленное время от предыдущего назначения и начинает отсчет времени с новым значением.

Будучи разблокированным, циклическое прерывание работает постоянно, выполняя назначенную программу обработки прерывания при каждом истечении заданного временного интервала. Если вы выйдете из режима RUN или отсоедините циклическое прерывание, то циклическое прерывание блокируется. Если выполняется глобальная команда блокирования прерываний, то циклические прерывания продолжают происходить. Каждое появление циклического прерывания ставится в очередь (пока прерывания не будут разблокированы или не заполнится очередь). Пример использования циклического прерывания показан на рис. 9–66.

Прерывания от таймера T32 или T96 позволяет своевременно реагировать на завершение заданного интервала времени. Эти прерывания поддерживаются только для таймеров с задержкой включения (TON) и с задержкой выключения (TOF) с разрешением 1 мс T32 и T96. Таймеры T32 и T96 в противном случае ведут себя нормально. Когда прерывание разблокировано, назначенная программа обработки прерывания исполняется, когда текущее значение активного таймера становится равным предустановленному значению во время нормального обновления 1-миллисекундного таймера, выполняемого в CPU. Эти прерывания разблокируются назначением программы обработки прерывания событиям, вызывающим прерывание от таймера T32/T96.

Описание приоритета прерываний и постановки их в очередь

Прерывания получают приоритеты в соответствии с фиксированной схемой приоритетов, показанной ниже:

- связь (наивысший приоритет)
- прерывания от ввода/вывода
- прерывания, управляемые временем (самый низкий приоритет).

В пределах соответствующего приоритета прерывания обслуживаются CPU по принципу «первым пришел – первым обслужен». В любой момент времени выполняется только одна программа обработки прерывания. Когда исполнение программы обработки прерывания начинается, программа выполняется до своего завершения. Она не может быть выгружена другой программой обработки прерывания, даже если последняя имеет более высокий приоритет. Прерывания, возникающие во время обработки другого прерывания, ставятся в очередь для последующей обработки.

Три очереди прерываний и максимальное количество прерываний, которое они могут хранить, показаны в таблице 9–22.

Таблица 9–22. Очереди прерываний и максимальное количество записей на очередь

Очередь	CPU 221	CPU 222	CPU 224	CPU 226
Очередь коммуникационных прерываний	4	4	4	8
Очередь прерываний от ввода/вывода	16	16	16	16
Очередь циклических прерываний	8	8	8	8

В принципе может возникнуть больше прерываний, чем может содержать очередь. Поэтому системой поддерживаются биты переполнения очереди (обозначающие тип прерывающих событий, которые были потеряны). Биты переполнения очереди прерываний показаны в таблице 9–23. Вам следует использовать эти биты только в программе обработки прерывания, так как они сбрасываются, когда очередь опустошается и управление возвращается главной программе.

Таблица 9–23. Определение битов специальной памяти в качестве битов переполнения очереди прерываний

Описание (0 = нет переполнения, 1 = переполнение)	SM-бит
Переполнение очереди коммуникационных прерываний	SM4.0
Переполнение очереди прерываний от ввода/вывода	SM4.1
Переполнение очереди циклических прерываний	SM4.2

В табл. 9–24 представлены события, вызывающие прерывания, приоритеты и соответствующие номера событий.

Таблица 9–24. События, вызывающие прерывания, в порядке убывания приоритета

Номер события	Описание прерывания	Группа приоритета	Приоритет в группе
8	Порт 0: символ принят	Коммуникации (наивысшая)	0
9	Порт 0: передача завершена		0
23	Порт 0: прием сообщения завершен		0
24	Порт 1: прием сообщения завершен		1
25	Порт 1: символ принят		1
26	Порт 1: передача завершена		1
19	Прерывание при завершении РТО 0	Дискретные операции (средняя)	0
20	Прерывание при завершении РТО 1		1
0	Нарастающий фронт, I0.0		2
2	Нарастающий фронт, I0.1		3
4	Нарастающий фронт, I0.2		4
6	Нарастающий фронт, I0.3		5
1	Падающий фронт, I0.0		6
3	Падающий фронт, I0.1		7
5	Падающий фронт, I0.2		8
7	Падающий фронт, I0.3		9
12	HSC0: CV=PV (текущее значение = предустановленному)		10
27	HSC0: направление изменено		11
28	HSC0: внешний сброс		12
13	HSC1: CV=PV (текущее значение = предустановленному)		13
14	HSC1: направление изменено		14
15	HSC1: внешний сброс		15
16	HSC2: CV=PV (текущее значение = предустановленному)		16
17	HSC2: направление изменено		17
18	HSC2: внешний сброс		18
32	HSC3: CV=PV (текущее значение = предустановленному)		19
29	HSC4: CV=PV (текущее значение = предустановленному)		20
30	HSC4: направление изменено		21
31	HSC4: внешний сброс		22
33	HSC5: CV=PV (текущее значение = предустановленному)		23
10	Циклическое прерывание 0	Управление Временем (низшая)	0
11	Циклическое прерывание 1		1
21	Прерывание от таймера T32 CT=PT		2
22	Прерывание от таймера T96 CT=PT		3

Примеры прерываний

На рис. 9–65 показан пример команд с программами обработки прерываний.

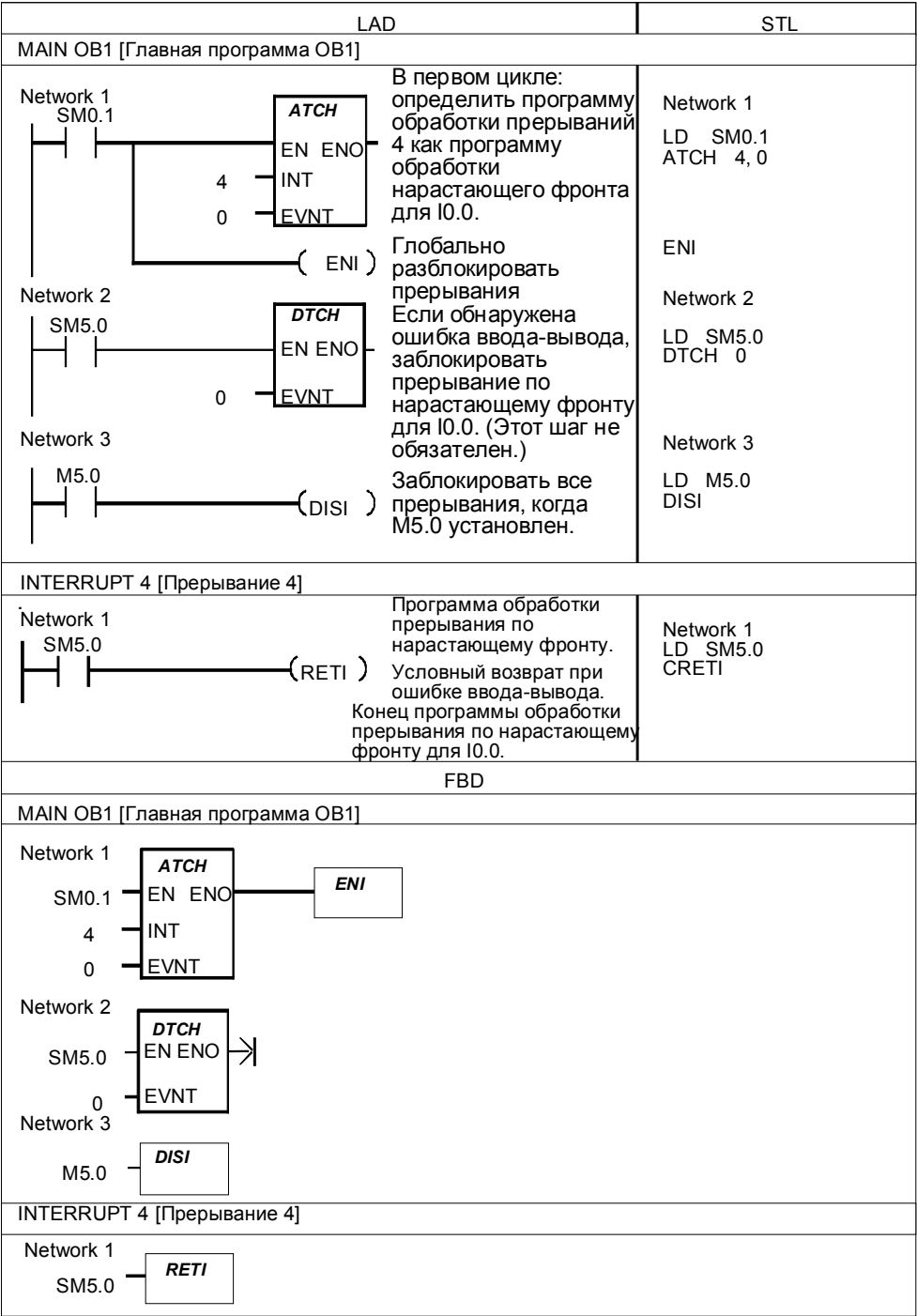


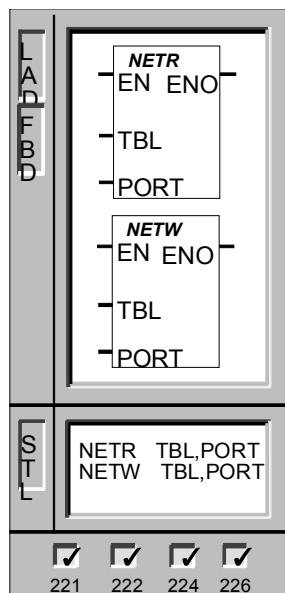
Рис. 9-65. Пример команд прерывания для SIMATIC LAD, STL и FBD

На рис. 9–66 показано, как организовать циклическое прерывание для считывания значений аналогового входа.

LAD		STL
MAIN [Главная программа]		
<p>Network 1</p>	<p>Бит первого цикла: Вызвать подпрограмму 0.</p>	<p>Network 1</p> <pre>LD SM0.1 CALL 0</pre>
SUBROUTINE 0		
<p>Network 1</p>	<p>Начало подпрограммы 0. Бит памяти постоянно установлен: Установить интервал для циклического прерывания 0 на 100 мс. Глобально разблокировать прерывания. Назначить циклическое прерывание 0 программе обработки прерываний 0.</p>	<p>Network 1</p> <pre>LD SM0.0 MOVB 100, SMB34 ATCH 0, 10 ENI</pre>
INTERRUPT 0 [Прерывание 0]		
<p>Network 1</p>	<p>Начало программы обработки прерываний 0. Опросить AIW4. Завершить программу обработки прерываний.</p>	<p>Network 1</p> <pre>LD SM0.0 MOVW AIW4, VW100</pre>
FB		
MAIN [Главная программа]		
<p>Network 1</p>	<p>*См. стр. 9-150</p>	
SUBROUTINE 0 [Подпрограмма 0]		
<p>Network 1</p>		
INTERRUPT 0 [Прерывание 0]		
<p>Network 1</p>		

Рис. 9-66. Пример организации циклического прерывания для чтения значения аналогового входа

Чтение из сети, запись через сеть



Команда **Читать из сети** инициирует коммуникационную операцию для получения данных из удаленного устройства через указанный порт (PORT), как указано в таблице (TBL).

Команда **Записать через сеть** инициирует коммуникационную операцию для записи данных в удаленное устройство через указанный порт (PORT), как указано в таблице (TBL).

Команда NETR может прочитать до 16 байтов информации из удаленной станции, а команда NETW может записать до 16 байтов информации в удаленную станцию. В программе можно иметь любое количество команд NETR/NETW, но одновременно можно активизировать не более восьми команд NETR и NETW. Например, в данном S7-200 в одно и то же время у вас могут активными четыре NETR и четыре NETW или два NETR и шесть NETW.

На рис. 9–67 определена таблица, к которой обращается параметр TBL в командах NETR и NETW.

NETR: Ошибки, устанавливающие ENO в 0: SM4.3 (этап исполнения), 0006 (косвенная адресация)

NETW: Ошибки, устанавливающие ENO в 0: SM4.3 (этап исполнения), 0006 (косвенная адресация)

Входы/выходы	Операнды	Типы данных
TBL	VB, MB, *VD, *AC, *LD	BYTE
PORT	константа	BYTE

		D – выполнена (функция завершена): 0 = не завершена; 1 = завершена			
		A – активна (функция поставлена в очередь): 0 = не активна; 1 = активна			
		E – ошибка (функция вернула ошибку): 0 = нет ошибки; 1 = ошибка			
Байтовое смещение	7	0			
	0	D	A	E	0 Код ошибки
	1	Адрес удаленной станции			
	2	Указатель на область			
	3	данных в			
	4	удаленной станции			
	5	(I, Q, M или V)			
	6	Длина данных			
	7	Байт данных 0			
	8	Байт данных 1			
		⋮			
	22	Байт данных 15			
		Адрес удаленной станции: адрес ПЛК, к данным которого нужно получить доступ.			
		Указатель на область данных в удаленной станции: косвенный указатель на данные, к которым нужно получить доступ.			
		Длина данных: количество данных, к которым нужно получить доступ в удаленной станции (от 1 до 16 байтов).			
		Область данных для приема или передачи: от 1 до 16 байтов, зарезервированных для данных, как описано ниже:			
		Для NETR это область данных, где хранятся значения, считанные из удаленной станции, после выполнения NETR.			
		Для NETW это область данных, где хранятся значения, подлежащие передаче в удаленную станцию, перед выполнением NETW.			
Код ошибки		Определение			
0		Нет ошибки			
1		Ошибка истечения времени ожидания, удаленная станция не отвечает.			
2		Ошибка приема; ошибка четности, кадрирования или контрольной суммы в ответе			
3		Ошибка отключения: коллизии, вызванные двойным адресом станции или неисправностью аппаратуры			
4		Ошибка переполнения очереди; активизировано более 8 блоков NETR/NETW			
5		Нарушение протокола; попытка выполнить NETR/NETW без разблокировки PPI+ в SMB30			
6		Недопустимый параметр; таблица NETR/NETW содержит недопустимое или ошибочное значение			
7		Нет ресурса; удаленная станция занята (идет процесс загрузки)			
8		Ошибка 7-го уровня; нарушение протокола приложения			
9		Ошибка сообщения; неверный адрес данных или неправильная длина данных			
A-F		Не используются (зарезервированы для использования в будущем)			

Рис. 9-67. Определение TABLE для NETR и NETW

Пример чтения из сети и записи через сеть

На рис. 9–68 показан пример, иллюстрирующий использование команд NETR и NETW. Для этого примера рассмотрим автоматическую линию, где коробочки заполняются маслом и передаются одной из четырех упаковочных машин. Упаковочная машина пакует по восемь коробочек с маслом в одну картонную коробку. Направляющее устройство управляет потоком коробочек с маслом, направляя их к той или иной упаковочной машине. Для управления упаковочными машинами используются четыре модуля CPU 221, а для управления направляющим устройством используется модуль CPU 222, оборудованный интерфейсом оператора TD 200. Структура сети показана на рис. 9–68.

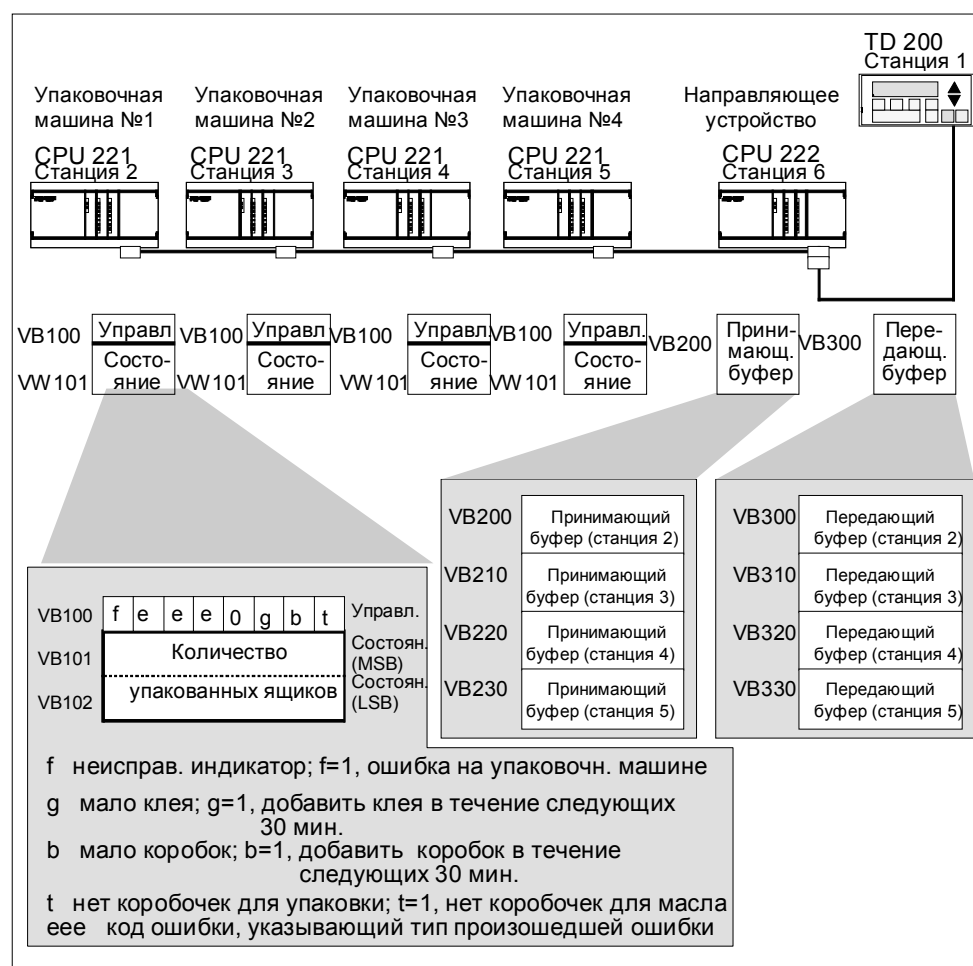


Рис. 9-68. Пример команд NETR и NETW

Приемный и передающий буфера для доступа к данным в станции 2 (размещенные соответственно в VB200 и VB300) подробно показаны на рис. 9–69.

CPU 224 использует команду NETR для регулярного чтения управляющей информации и информации о состоянии из каждой упаковочной машины. Каждый раз, когда упаковочная машина упаковывает 100 коробок, направляющее устройство замечает это и с помощью команды NETW передает сообщение для сброса слова состояния.

Программа, необходимая для чтения управляющего байта, подсчета количества упакованных ящиков и сброса количества упакованных ящиков для каждой упаковочной машины в отдельности (упаковочная машина №1), показана на рис. 9–70.

Приемный буфер направляющего устройства для чтения из упаковочной машины №1					Передающий буфер направляющего устройства для сброса подсчитанного количества коробок в упаковочной машине №1						
	7			0		7			0		
VB200	D	A	E	0	Код ошибки	VB300	D	A	E	0	Код ошибки
VB201	Адрес удаленной станции					VB301	Адрес удаленной станции				
VB202	Указатель на					VB302	Указатель на				
VB203	область данных					VB303	область данных				
VB204	в удаленной					VB304	в удаленной				
VB205	станции = (&VB100)					VB305	станции = (&VB101)				
VB206	Длина данных = 3 байта					VB306	Длина данных = 2 байта				
VB207	Управление					VB307	0				
VB208	Состояние (MSB)					VB308	0				
VB209	Состояние (LSB)										

Рис. 9-69. Образец данных TABLE для примера команд NETR и NETW

LAD	STL
<p>Network 1</p> <p>SM0.1</p> <p>2</p> <p>MOV B</p> <p>EN ENO</p> <p>IN OUT</p> <p>SMB30</p> <p>Network 2</p> <p>V200.7 VV208</p> <p>==I</p> <p>100</p> <p>2</p> <p>MOV B</p> <p>EN ENO</p> <p>IN OUT</p> <p>VB301</p> <p>&VB101</p> <p>MOV D</p> <p>EN ENO</p> <p>IN OUT</p> <p>VD302</p> <p>2</p> <p>MOV B</p> <p>EN ENO</p> <p>IN OUT</p> <p>VB306</p> <p>0</p> <p>MOV W</p> <p>EN ENO</p> <p>IN OUT</p> <p>VW307</p> <p>VB300</p> <p>0</p> <p>NETW</p> <p>EN ENO</p> <p>TBL</p> <p>PORT</p> <p>Network 3</p> <p>V200.7</p> <p>VB207</p> <p>MOV B</p> <p>EN ENO</p> <p>IN OUT</p> <p>VB400</p> <p>Network 4</p> <p>SM0.1 V200.6 V200.5</p> <p>/ / /</p> <p>2</p> <p>MOV B</p> <p>EN ENO</p> <p>IN OUT</p> <p>VB201</p> <p>&VB100</p> <p>MOV D</p> <p>EN ENO</p> <p>IN OUT</p> <p>VD202</p> <p>3</p> <p>MOV B</p> <p>EN ENO</p> <p>IN OUT</p> <p>VB206</p> <p>VB200</p> <p>0</p> <p>NETR</p> <p>EN ENO</p> <p>TBL</p> <p>PORT</p>	<p>В первом цикле разблокировать протокол PPI+.</p> <p>Очистить все принимающие и передающие буфера.</p> <p>Если бит завершения NETR установлен и 100 коробок упаковано, загрузить адрес станции упаковочной машины №1. Загрузить указатель на данные в удаленной станции. Загрузить длину данных для передачи. Загрузить данные, подлежащие передаче.</p> <p>Сбросить количество коробок, упакованных упаковочной машиной №1. Если бит завершения установлен, сохранить управляющие данные упаковочной машиной №1. Если NETR не активна и нет ошибок, загрузить адрес станции упаковочной машиной №1. Загрузить указатель на данные в удаленной станции. Загрузить длину данных, подлежащих приему. Прочитать управляющие данные и данные о состоянии в упаковочной машине №1.</p> <p>Network 1</p> <p>LD SM0.1</p> <p>MOVB 2, SMB30</p> <p>FILL 0, VW200, 68</p> <p>Network 2</p> <p>LD V200.7</p> <p>AW= VW208, 100</p> <p>MOVB 2, VB301</p> <p>MOVD &VB101, VD302</p> <p>MOVB 2, VB306</p> <p>MOVW 0, VW307</p> <p>NETW VB300, 0</p> <p>Network 3</p> <p>LD V200.7</p> <p>MOVB VB207, VB400</p> <p>Network 4</p> <p>LDN SM0.1</p> <p>AN V200.6</p> <p>AN V200.5</p> <p>MOVB 2, VB201</p> <p>MOVD &VB100, VD202</p> <p>MOVB 3, VB206</p> <p>NETR VB200, 0</p>

Рис. 9-70. Пример команд NETR и NETW для SIMATIC LAD и STL

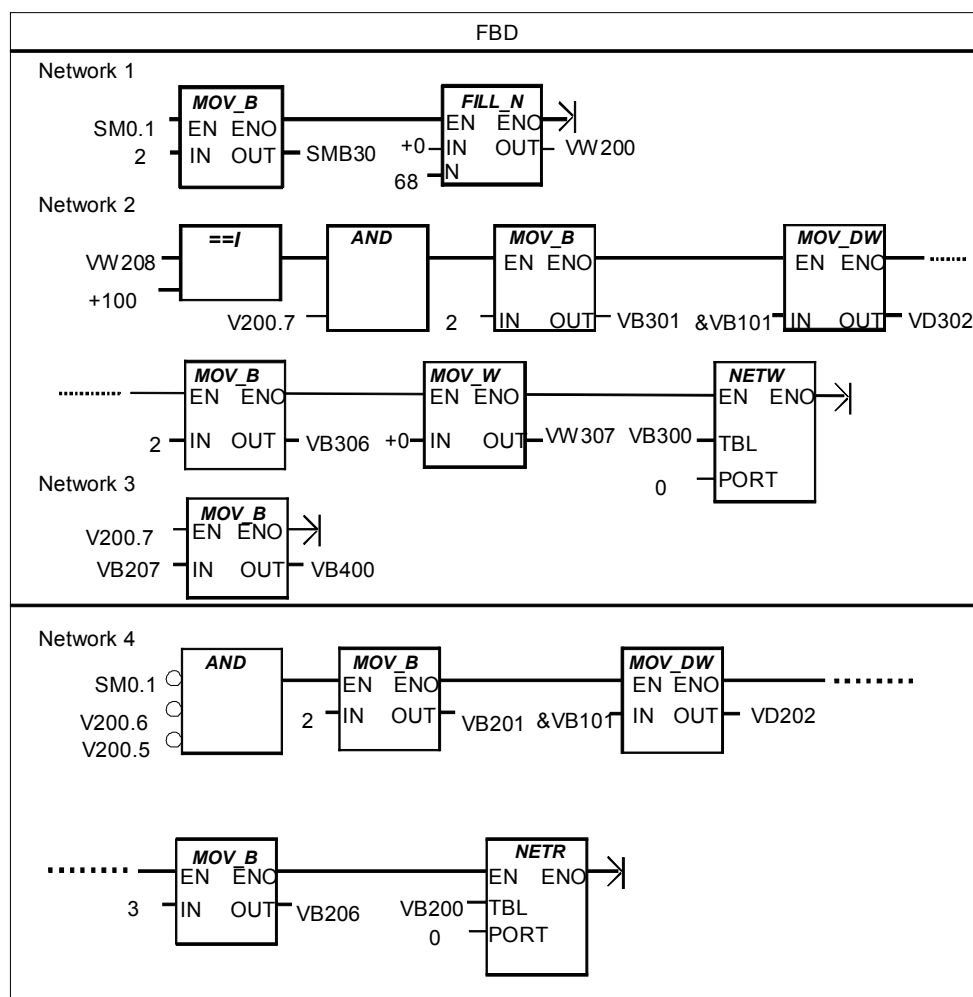
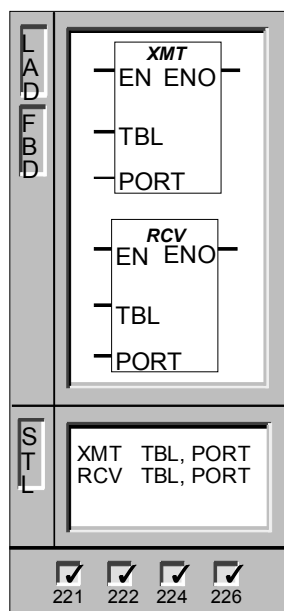


Рис. 9–71. Пример команд NETR и NETW для SIMATIC FBD

Передача, прием сообщения



Команда **Передать сообщение** вызывает передачу буфера данных (TBL). Первая запись в буфере данных определяет количество байтов, подлежащих передаче. PORT задает коммуникационный порт, который должен быть использован для передачи.

Команда XMT использует в режиме свободно программируемой связи для передачи данных через коммуникационный порт(ы).

Команда **Принять сообщение** инициирует или завершает обслуживание приема сообщения. Чтобы блок приема мог работать, вы должны указать условия начала и конца сообщения. Сообщения, получаемые через указанный порт (PORT), сохраняются в буфере данных (TBL). Первая запись в буфере данных указывает количество принятых байтов.

Передать сообщение: Ошибки, устанавливающие ENO в 0: SM4.3 (этап исполнения), 0006 (косвенная адресация), 0009 (одновременно XMT/RCV в порту 0), 000B (одновременно XMT/RCV в порту 1), CPU не в режиме свободного программирования связи.

Принять сообщение: Ошибки, устанавливающие ENO в 0: SM86.6 и SM186.6 (RCV – ошибка параметризации), SM4.3 (этап исполнения), 0006 (косвенная адресация), 0009 (одновременно XMT/RCV в порту 0), 000B (одновременно XMT/RCV в порту 1), CPU не в режиме свободного программирования связи.

Входы/выходы	Операнды	Типы данных
TABLE	VB, IB, QB, MB, SB, SMB, *VD, *AC, *LD	BYTE
PORT	константа (0 для CPU 221, CPU 222, CPU 224; 0 или 1 для CPU 226)	BYTE

Описание режима свободно программируемой связи

Режим свободно программируемой связи можно выбрать для управления последовательным коммуникационным портом CPU с помощью программы пользователя. Если вы используете режим свободно программируемой связи, то программа на LAD управляет работой коммуникационного порта путем использования прерываний приема, прерываний передачи, команды передачи (XMT) и команды приема (RCV). Во время свободно программируемой связи протокол связи полностью управляется программой LAD. Для выбора скорости передачи и контроля четности используются SMB30 (для порта 0) и SMB130 (для порта 1, если ваш CPU имеет два порта).

Режим свободно программируемой связи блокируется, и восстанавливается обычная связь (например, доступ через устройство программирования), когда CPU находится в состоянии STOP.

В простейшем случае вы можете послать сообщение на принтер или дисплей, используя только команду «Передать сообщение» (XMT). К другим примерам относятся связь с устройством для считывания штрихового кода, весами или сварочным аппаратом. В каждом случае вы должны написать программу для поддержки протокола, используемого устройством, с которым CPU обменивается данными при нахождении в режиме свободно программируемой связи.

Свободно программируемая связь возможна только тогда, когда CPU находится в режиме RUN. Разблокируйте режим свободно программируемой связи установкой значения 01 в поле выбора протокола SMB30 (порт 0) или SMB130 (порт 1). В режиме свободно программируемой связи обмен данными с устройством программирования невозможен.

Примечание

Вход в режим свободно программируемой связи может управляться с помощью бита специальной памяти SM0.7, который отражает текущее положение переключателя режимов работы. Если SM0.7 равен 0, то переключатель находится в положении TERM; если SM0.7 = 1, то переключатель режимов работы находится в положении RUN. Если вы разблокируете режим свободно программируемой связи только тогда, когда переключатель находится в положении RUN, то вы можете использовать устройство программирования для контроля и управления работой CPU путем перевода переключателя в любое другое положение.

Инициализация свободно программируемой связи

SMB30 и SMB130 конфигурируют коммуникационные порты 0 и 1 соответственно для свободно программируемой связи и обеспечивают выбор скорости передачи, контроля четности и количество битов данных. Описание управляющих байтов свободно программируемой связи представлено в таблице 9–25.

Таблица 9–25. Байты специальной памяти SMB30 и SMB130

Порт 0	Порт 1	Описание
Формат SMB30	Формат SMB130	<div style="display: flex; align-items: center;"> <div style="margin-right: 20px;"> <div style="text-align: center;">MSB 7</div> <div style="display: flex; justify-content: space-around; width: 100px;"> <div style="border: 1px solid black; padding: 2px;">p</div> <div style="border: 1px solid black; padding: 2px;">p</div> <div style="border: 1px solid black; padding: 2px;">d</div> <div style="border: 1px solid black; padding: 2px;">b</div> <div style="border: 1px solid black; padding: 2px;">b</div> <div style="border: 1px solid black; padding: 2px;">b</div> <div style="border: 1px solid black; padding: 2px;">m</div> <div style="border: 1px solid black; padding: 2px;">m</div> </div> <div style="text-align: center;">LSB 0</div> </div> <div>Управляющий байт режима свободно программируемой связи</div> </div>
SM30.6 и SM30.7	SM130.6 и SM130.7	pp: выбор контроля четности 00 = нет контроля четности 01 = контроль по четности 10 = нет контроля четности 11 = контроль по нечетности
SM30.5	SM130.5	d: Количество битов данных на символ 0 = 8 битов на символ 1 = 7 битов на символ
SM30.2 ÷ SM30.4	SM130.2 ÷ SM130.4	bbb: скорость передачи в режиме свободно программируемой связи 000 = 38 400 Бод 001 = 19 200 Бод 010 = 9 600 Бод 011 = 4 800 Бод 100 = 2 400 Бод 101 = 1 200 Бод 110 = 600 Бод 111 = 300 Бод
SM30.0 и SM30.1	SM130.0 и SM130.1	mm: Выбор протокола 00 = протокол интерфейса точка-точка (PPI/непривилегированный режим) 01 = протокол свободно программируемой связи 10 = PPI/привилегированный режим 11 = резерв (по умолчанию PPI/непривилегированный режим)

Примечание: для всех конфигураций генерируется один стоповый бит.

Использование команды ХМТ для передачи данных

Команда ХМТ дает возможность передать буфер, состоящий из одного или нескольких символов (не более 255). После того как передан последний символ буфера, генерируется прерывание (прерывающее событие 9 для порта 0 и прерывающее событие 26 для порта 1), если событию «Передача завершена» поставлена в соответствие программа обработки прерывания. Вы можете производить передачу и без использования прерываний (например, посылая сообщение на принтер), контролируя SM4.5 или SM4.6, сигнализирующих о завершении передачи.

Команда ХМТ может быть использована для генерирования условия BREAK путем установки количества символов в ноль, а затем исполнения команды ХМТ. Это генерирует условие BREAK 16 раз с текущей скоростью передачи. Передача BREAK обрабатывается так же, как передача любого другого сообщения, причем прерывание ХМТ генерируется, когда передача BREAK завершена, а SM4.5 или SM4.6 отображают текущее состояние ХМТ.

Формат буфера ХМТ показан на рис. 9–72.

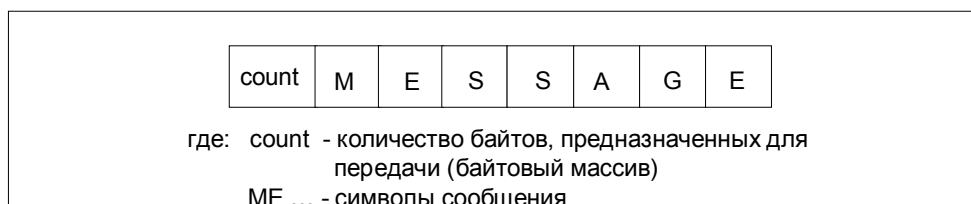


Рис. 9–72. Формат буфера ХМТ

Использование команды RCV для приема данных

Команда RCV дает возможность принять буфер, состоящий из одного или нескольких символов (не более 255). После того как принят последний символ буфера, генерируется прерывание (прерывающее событие 23 для порта 0 и прерывающее событие 24 для порта 1), если событию «Прием сообщения завершен» поставлена в соответствие программа обработки прерывания.

Вы можете принимать сообщения и без использования прерываний, контролируя SMB86. SMB86 (или SMB186) не равен нулю, если блок RCV неактивен или завершил работу. Он равен нулю, когда происходит прием.

Команда RCV позволяет выбирать условие начала и окончания сообщения. Описание этих условий см. в таблице 9–26 (от SM86 до SM94 для порта 0 и от SM186 до SM194 для порта 1). Формат буфера RCV показан на рис. 9–73.

Примечание

Функция приема сообщений автоматически завершается при превышении количества символов или ошибке четности. Вы должны определить условие начала (x или z) и условие конца (y, t или максимальное количество символов), чтобы функция приема сообщений могла работать.

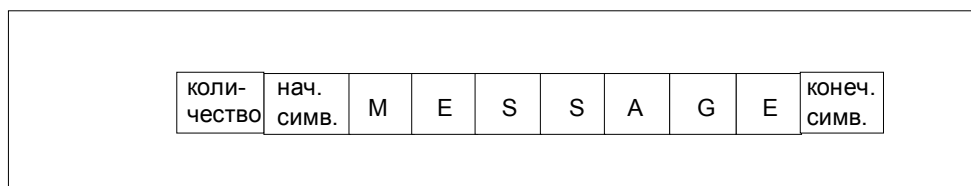


Рис. 9–73. Формат буфера RCV


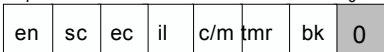
Порт 0	Порт 1	Описание
SMB86	SMB186	<p>  </p> <p>Байт состояния принимаемого сообщения</p> <p> n: 1 = прием сообщения завершен командой блокировки от пользователя r: 1 = прием сообщения завершен: ошибка во входных параметрах или отсутствие условия начала или конца e: 1 = получен символ конца t: 1 = прием сообщения завершен: истекло время таймера c: 1 = прием сообщения завершен: достигнуто макс. количество символов p: 1 = прием сообщения завершен из-за ошибки четности </p>
SMB87	SMB187	<p>  </p> <p>Управляющий байт для приема сообщения</p> <p> en: 0 = функция приема сообщений заблокирована. 1 = функция приема сообщений разблокирована. Бит разблокировки/блокировки приема сообщений проверяется при каждом исполнении команды RCV. sc: 0 = игнорировать SMB88 или SMB188. 1 = использовать значение SMB88 или SMB188 для обнаружения начала сообщения. ec: 0 = игнорировать SMB89 или SMB189. 1 = использовать значение SMB89 или SMB189 для обнаружения конца сообщения. il: 0 = игнорировать SMW90 или SMB90. 1 = использовать значение SMW90 для обнаружения бездействия линии c/m: 0 = таймер измеряет время между символами 1 = таймер измеряет время сообщения. tmr: 0 = игнорировать SMW92 или SMW192. 1 = завершить прием, если превышен интервал времени, указанный в SMW92 или SMW192. bk: 0 = игнорировать условия прерывания 1 = использовать условие прерывания как начало обнаружения сообщения. </p> <p> Биты байта управления прерыванием сообщения используются для определения критериев, с помощью которых распознается сообщение. Определяются критерии начала и конца сообщения. Для определения начала сообщения должен принимать значение «истина» любой из двух наборов логически соединенных по И критериев начала сообщения, которые должны выполняться последовательно (бездействующая линия, после чего следует символ начала, или разрыв сообщения, за которым следует символ начала). Для завершения сообщения разрешенные критерии конца сообщения логически комбинируются по ИЛИ. Ниже приведены уравнения для критериев начала и конца: </p> <p> Начало сообщения = $il * sc + bk * sc$ Конец сообщения = $ec + tmr$ достигнуто максимальное количество символов. </p> <p>Программирование критериев начала сообщения:</p> <ol style="list-style-type: none"> обнаружение бездействующей линии: $il=1, sc=0, bk=0, SMW90>0$ обнаружение символа начала: $il=0, sc=1, bk=0, SMW90$ не имеет значения обнаружение разрыва: $il=0, sc=0, bk=1, SMW90$ не имеет значения любой ответ на запрос: $il=0, sc=0, bk=1, SMW90=0$ <p>(Для завершения приема может быть использован таймер сообщения, если ответ отсутствует.)</p> <ol style="list-style-type: none"> разрыв и символ начала: $il=0, sc=1, bk=1, SMW90$ не имеет значения бездействующая линия и символ начала: $il=1, sc=1, bk=0, SMW90 >0$ бездействующая линия и символ начала (недопустимый): $il=1, sc=1, bk=0, SMW90=0$ <p>Примечание: прием будет автоматически завершен при превышении количества символов или ошибке четности (если разблокирована)</p>
SMB88	SMB188	Символ начала сообщения
SMB89	SMB189	Символ конца сообщения
SMB90	SMB190	Время бездействия линии в миллисекундах. Первый символ.

Таблица 9–26. Байты специальной памяти SMB86 ÷ SMB94 и SMB186 ÷ SMB194

Порт 0	Порт 1	Описание
SMB91	SMB191	принятый по истечении времени бездействия линии, является началом нового сообщения. SMB90 (или SMB190) – это старший байт, а SMB91 (или SMB191) – младший байт.
SMB92 SMB93	SMB192 SMB193	Значение контроля времени при измерении времени между символами и времени сообщения в миллисекундах. Если этот интервал времени истек, то прием сообщения завершается. SMB92 (или SMB192) – это старший байт, а SMB93 (или SMB193) – младший байт.
SMB94	SMB194	Максимальное количество символов, которое должно быть принято (от 1 до 255 байтов). Примечание: Этот диапазон должен быть установлен на ожидаемый максимальный размер буфера, даже если завершение сообщения с помощью подсчета символов не используется.

Прием данных с помощью прерываний от символов

Для достижения полной гибкости в поддержке протоколов вы также можете принимать данные, управляя прерываниями от приема символов. Каждый принимаемый символ генерирует прерывание. Принятый символ помещается в SMB2, а результат контроля четности (если активизирован) помещается в SMB3 непосредственно перед исполнением программы обработки прерывания, назначенной событию «Символ принят».

- SMB2 – это буфер для принятого символа при свободно программируемой связи. Каждый символ, принятый в режиме свободно программируемой связи, помещается по этому адресу для облегчения доступа к нему из программы пользователя.
- SMB3 используется для режима свободно программируемой связи и содержит бит ошибки четности, который устанавливается, когда в принятом символе обнаруживается ошибка четности. Все остальные биты этого байта зарезервированы. Используйте этот бит для отклонения сообщения или для генерирования отрицательного квитирования этого сообщения.

Примечание

SMB2 и SMB3 совместно используются портами 0 и 1. Когда прием символа в порт 0 приводит к исполнению программы обработки прерывания, назначенной этому событию (прерывающее событие 8), SMB2 содержит символ, принятый портом 0, а SMB3 содержит результат контроля четности этого символа. Когда прием символа в порт 1 приводит к исполнению программы обработки прерывания, назначенной этому событию (прерывающее событие 25), SMB2 содержит символ, принятый портом 1, а SMB3 содержит результат контроля четности этого символа.

Пример приема и передачи

Программа этого примера (рис. 9–74) показывает использование команд приема и передачи. Эта программа принимает строку символов, пока не будет принят символ перевода строки. Это сообщение затем передается обратно отправителю.

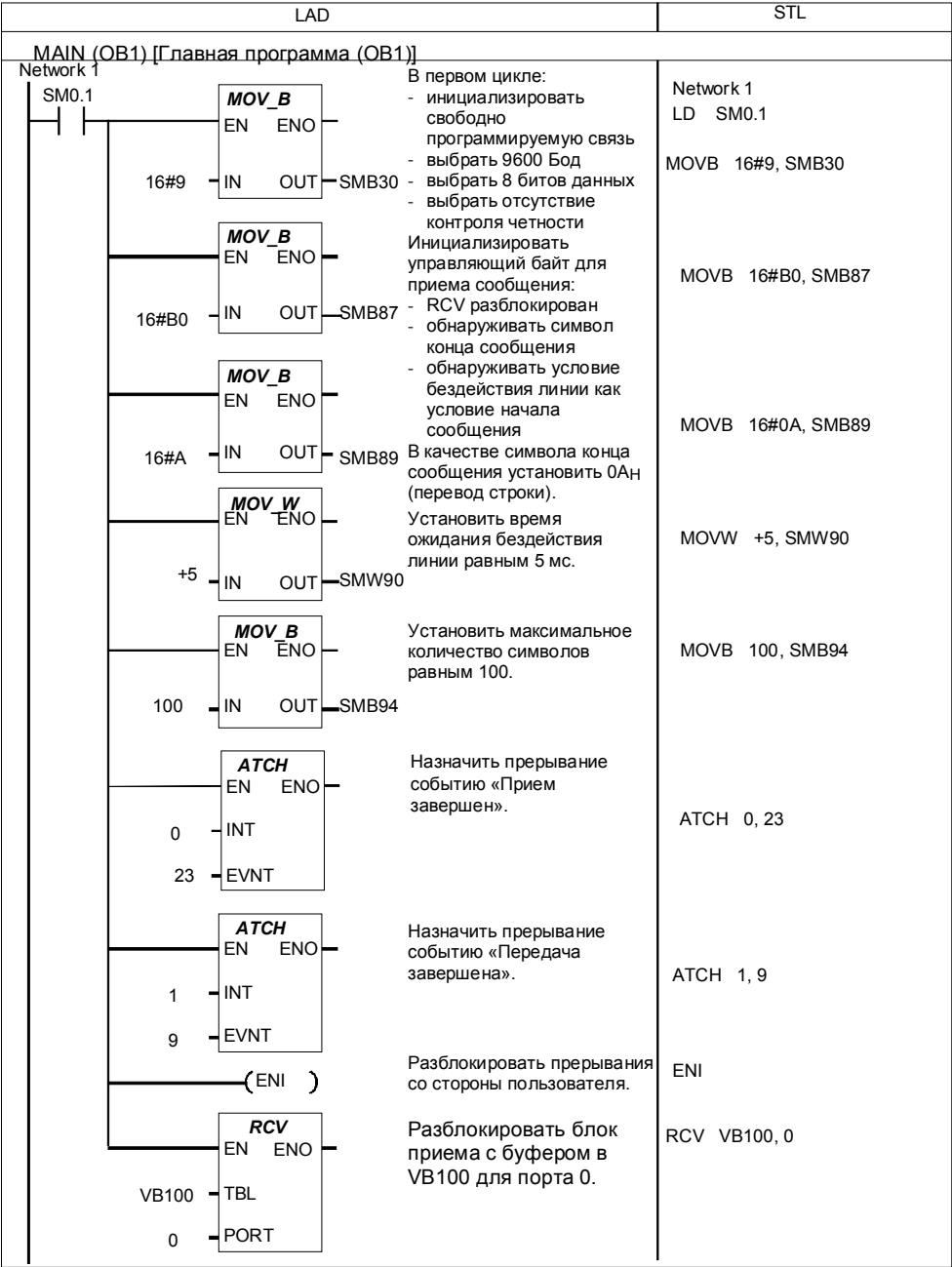


Рис. 9-74. Пример команды передачи для SIMATIC LAD, STL и FBD

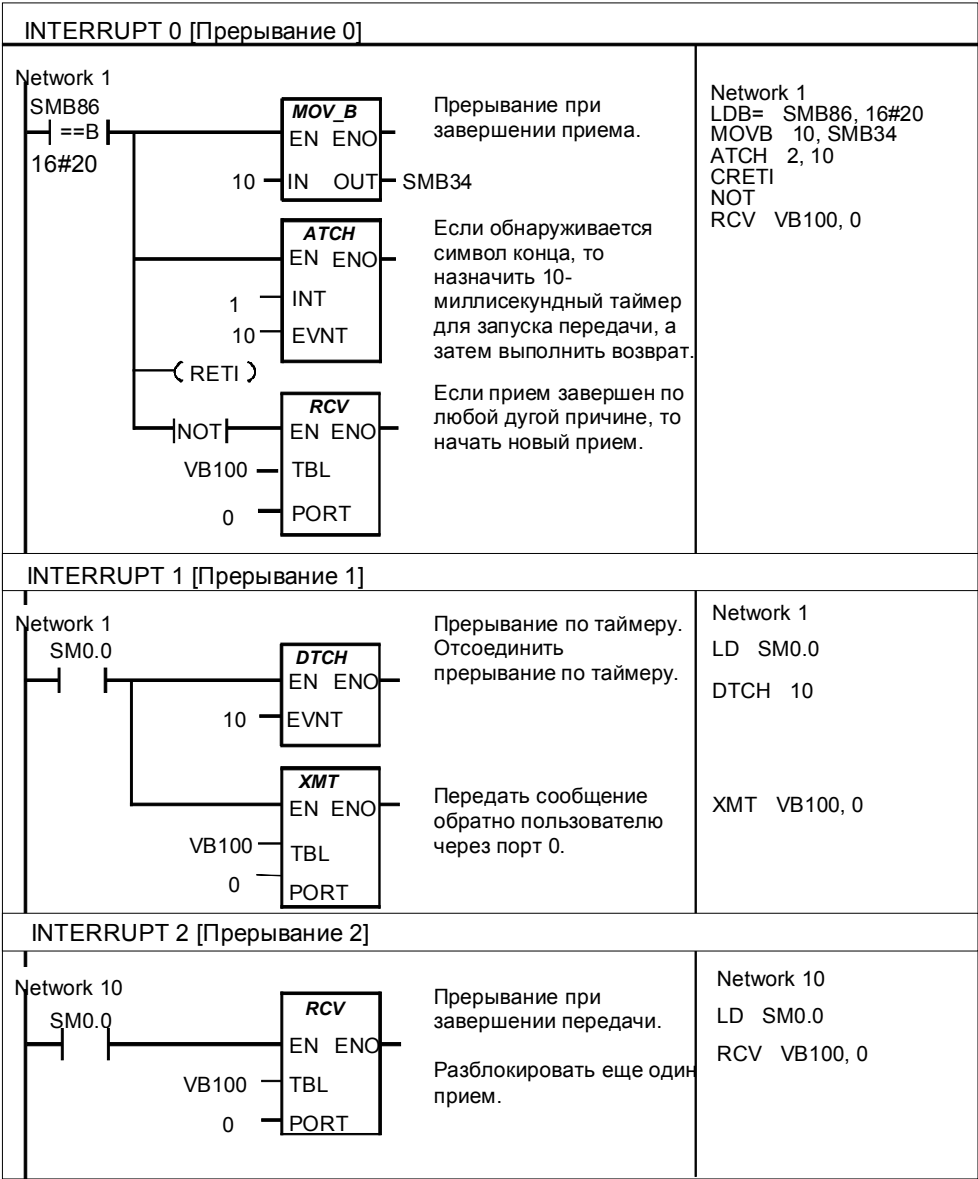


Рис. 9-74. Пример команды передачи для SIMATIC LAD, STL и FBD (продолжение)

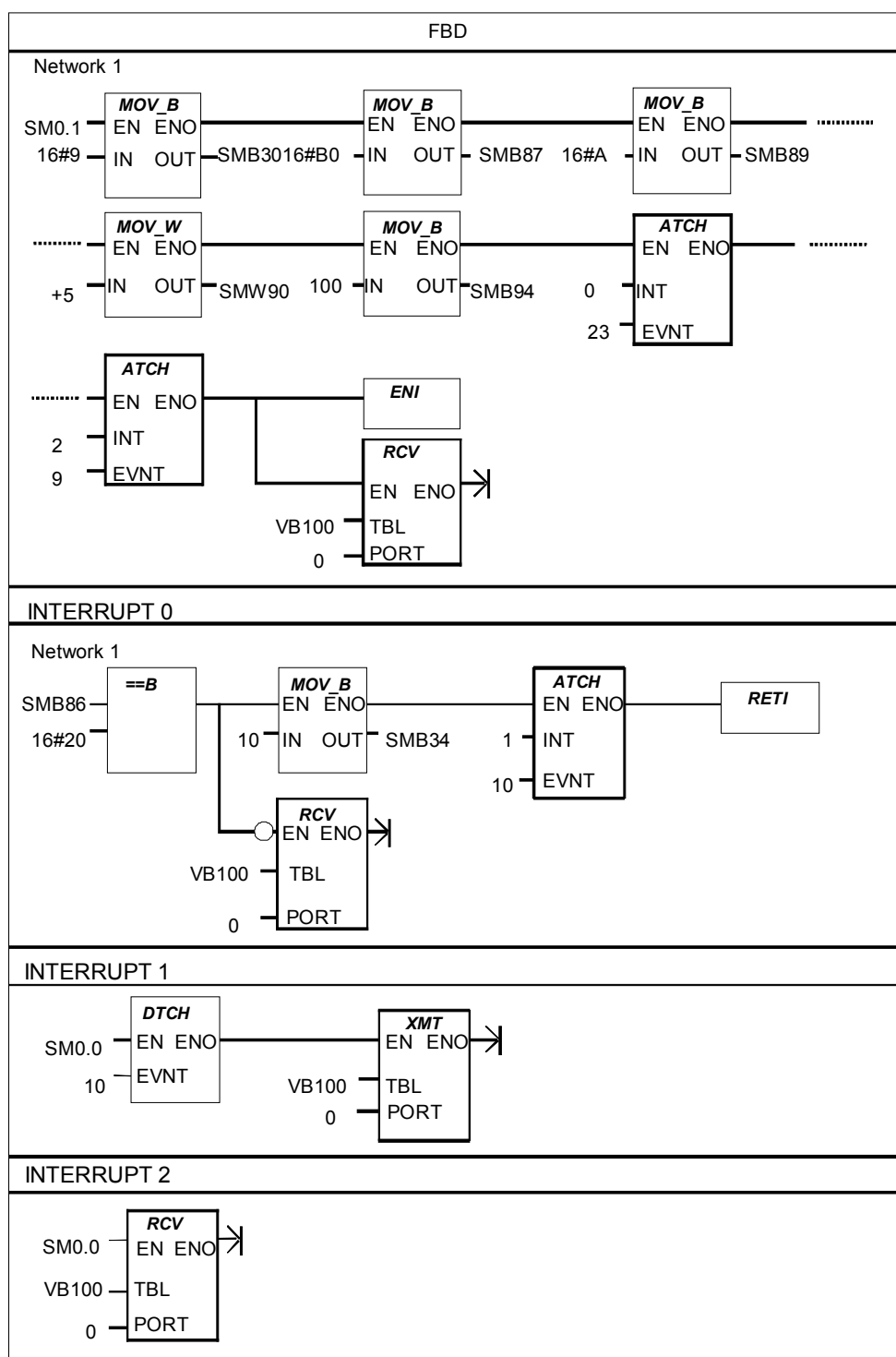
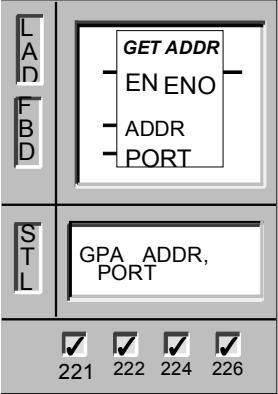


Рис. 9-74. Пример команды передачи для SIMATIC LAD, STL и FBD
(продолжение)

Получение адреса порта

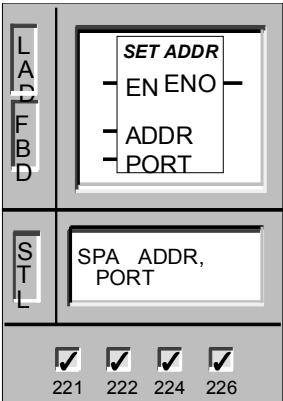


Команда **Получить адрес порта** считывает адрес станции из потока CPU, указанного в параметре PORT, и помещает значение по адресу, указанному в ADDR.

Получить адрес порта: Ошибки, устанавливающие ENO в 0: SM4.3 (этап исполнения), 0006 (косвенная адресация)

Входы/выходы	Операнды	Типы данных
ADDR	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD	BYTE
PORT	константа	BYTE

Установить адрес порта



Команда **Установить адрес порта** адрес порта станции (PORT) на значение, указанное в ADDR.

Установить адрес порта: Ошибки, устанавливающие ENO в 0: SM4.3 (этап исполнения), 0006 (косвенная адресация)

Новый адрес не сохраняется постоянно. По завершении цикла порт возвращается к последнему адресу (к тому, который был загружен системным блоком).

Входы/выходы	Операнды	Типы данных
ADDR	VB, IB, QB, MB, SB, SMB, LB, AC, константа, *VD, *AC, *LD	BYTE
PORT	константа	BYTE

9.18 Команды SIMATIC, выполняемые над логическим стеком

Логическое сопряжение первого и второго уровня по И

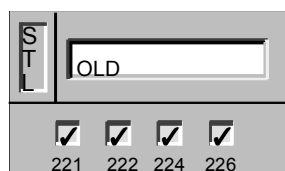


Команда **Выполнить логическое сопряжение первого и второго уровня по И** комбинирует значения в первом и втором уровне стека, используя логическое И.

Результат загружается в вершину стека. После выполнения AND глубина стека уменьшается на единицу.

Операнды: Нет

Логическое сопряжение первого и второго уровня по ИЛИ



Команда **Выполнить логическое сопряжение первого и второго уровня стека по ИЛИ** комбинирует значения в первом и втором уровне стека, используя логическое ИЛИ.

Результат загружается в вершину стека. После выполнения AND глубина стека уменьшается на единицу.

Операнды: Нет

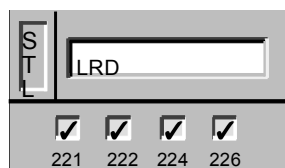
Дублирование вершины логического стека



Команда **Дублировать вершину логического стека** дублирует значение вершины стека и помещает это значение в стек. Дно стека выталкивается и теряется.

Операнды: Нет

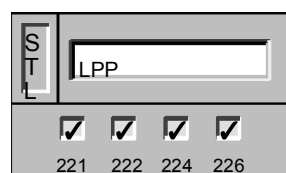
Копирование второго уровня стека



Команда **Копировать второй уровень стека** копирует второй уровень стека в его вершину. В стек ничего не помещается и из него ничего не извлекается, но его вершина замещается копией.

Операнды: Нет

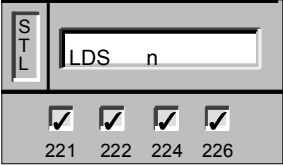
Извлечение вершины стека



Команда **Извлечь вершину стека** извлекает одно значение из стека. Второй уровень становится новой вершиной стека.

Операнды: Нет

Дублирование n-го бита стека



Команда **Дублировать n-ый бит стека** дублирует n-ый бит стека и помещает это значение в вершину стека. Дно стека выталкивается и теряется.

Операнды: n (от 1 до 8)

Логические операции со стеком

Рис. 9–75 иллюстрирует действие логического сопряжения первого и второго уровня стека по И и ИЛИ.



Рис. 9-75. Команды логического сопряжения по И и ИЛИ двух верхних уровней стека

Рис. 9–76 иллюстрирует действие дублирования вершины логического стека, копирования второго уровня стека и извлечения вершины стека.

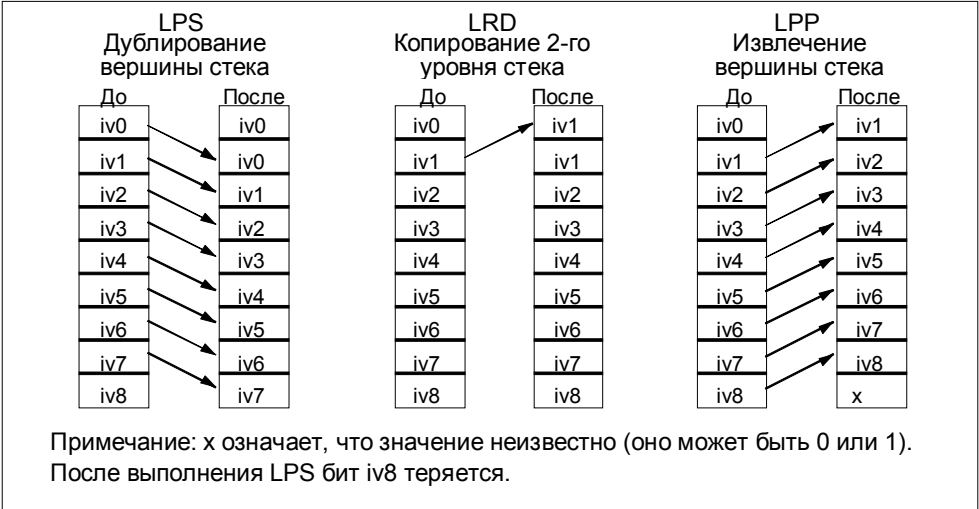


Рис. 9-76. Дублирование вершины стека, копирование второго уровня стека и извлечение вершины стека

Рис. 9–77 иллюстрирует действие дублирования n-го бита стека.

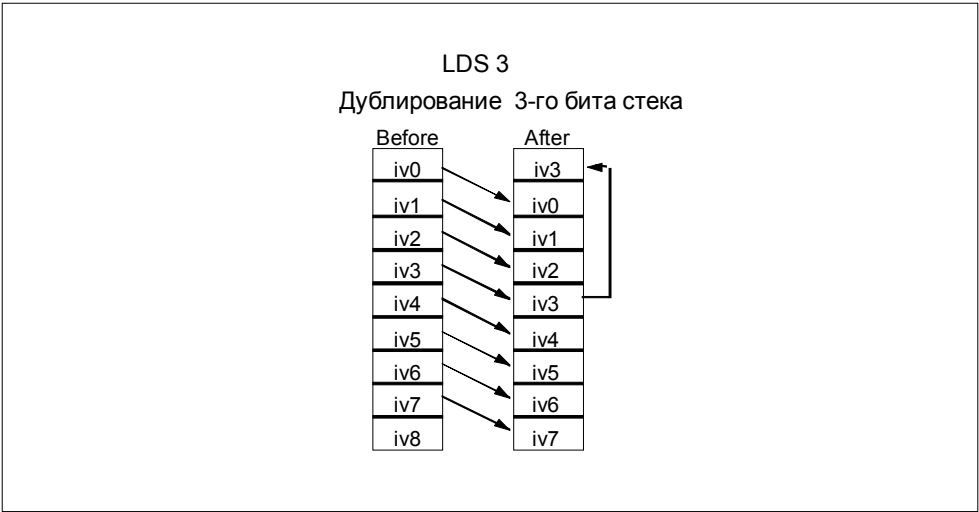


Рис. 9-77. Дублирование n-го бита стека

Пример логического стека

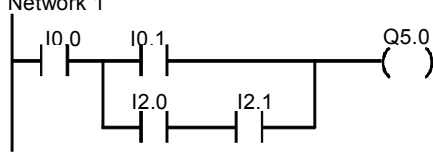
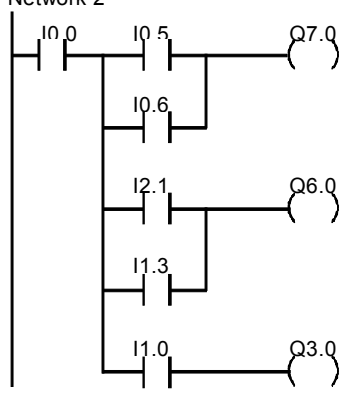
LAD	STL
<p>Network 1</p> 	<p>NETWORK 1</p> <pre>LD I0.0 LD I0.1 LD I2.0 A I2.1 OLD ALD = Q5.0</pre>
<p>Network 2</p> 	<p>NETWORK 2</p> <pre>LD I0.0 LPS LD I0.5 O I0.6 ALD = Q7.0 LRD LD I2.1 O I1.3 ALD = Q6.0 LPP A I1.0 = Q3.0</pre>

Рис. 9-78. Пример команд, выполняемых над логическим стеком для SIMATIC LAD и STL

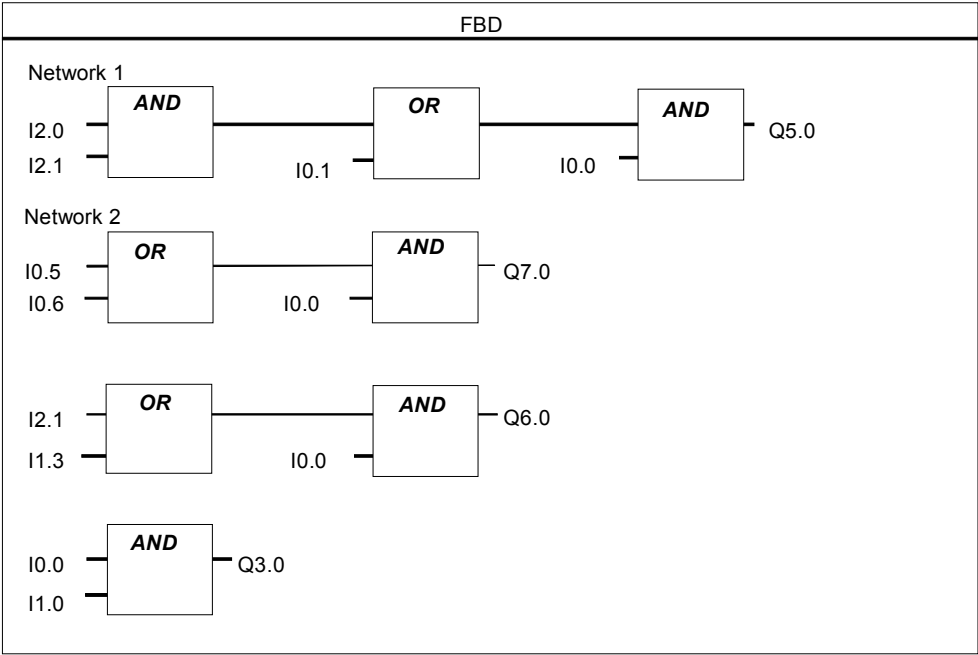


Рис. 9–79. Пример команд, выполняемых над логическим стеком для SIMATIC FBD