# Базовые функции

В данной части книги описываются функции языка программирования STL, представляющие собой "базовые функции" ("basic functionality") STEP 7. Эти функции позволят Вам запрограммировать PLC как систему управления, построенную на контакторах и реле.

**Двоичные логические операции** (**binary logic operations**) используются для моделирования параллельных и последовательных цепей в схемах или для выполнения функций AND (И) и OR (ИЛИ) в электронных переключающих системах. Использование приема вложения ("комбинирование") функций делает возможным выполнение сложных двоичных логических операций.

**Операции с памятью** (**memory functions**) используются для сохранения результата логической операции (RLO - result of logic operation). Таким образом, этот результат может быть в дальнейшем проверен и обработан в другом месте программы.

**Функции пересылки** (**transfer functions**) используются для обработки дискретных значений. Эти функции также используются, например, для передачи таймеру значения времени.

**Таймеры** (**timer**) используются для программирования PLC на выполнение функций включения/выключения, управляемых по времени, и для обеспечения функции задержки в переключающих электронных системах "electronic switching systems". Таймеры, встроенные в CPU, позволяют Вам программировать такие параметры как время ожидания или время контроля (monitoring time).

**Счетчики** (**counter**) двух видов - прямого и обратного счета - используются для счета в пределах от 0 до 999.

В данной части книги описываются функции, использующие адресные области входов, выходов и меркеров. Входы и выходы связаны с процессом или установкой. Меркеры играют роль дополнительных реле, сохраняющих двоичные состояния. В последующих разделах книги рассматриваются другие адресные области, которые могут использоваться в двоичных логических операциях. Одними из наиболее важных являются биты данных в блоках глобальных данных, а также биты данных во временных и статических локальных данных.

Глава 5 "Операции с памятью" (memory functions) содержит примеры программирования двоичных логических операций и операций с памятью (memory functions); глава 8 "Функции счетчиков" предоставляет Вашему вниманию примеры использования таймеров и счетчиков. Каждый пример содержится в FC функции без параметров.

**4 Двоичные логические операции (binary logic operations)**

Функции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ); считывание состояния сигнала "1" и "0"; выполнение двоичных логических операций; вложение (комбинирование) функций.

**5 Операции с памятью (memory functions)**

Функции Assign (присвоение), Set (установка) и Reset (сброс); RS-триггер; проверка наличия фронта сигнала; пример управления ленточным транспортером.

**6 Функции пересылки (transfer functions)**

Функции Load (загрузка) и Transfer (выгрузка); функции аккумулятора.

**7 Таймеры (timer)**

Запуск 5 различных типов таймеров; сброс, разблокировка и считывание значения таймера; значение времени.

**8 Счетчики (counter)**

Установка счетчика; прямой и обратный счет; сброс разблокировка и считывание значения счетчика; значение счетчика; пример счетчика числа деталей.

# 4 Двоичные логические операции

В этой главе обсуждаются функции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ), также как и их комбинации для языка программирования STL. Функции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ) используются для проверки состояния сигналов двоичных разрядов и связи их друг с другом.

Двоичный разряд может быть проверен (просканирован) на наличие сигнала "1" или "0". С помощью инвертирования результата логической операции и использования комбинации логических операций Вы можете запрограммировать сложные логические операции без необходимости хранения промежуточного результата.

Примеры, показанные в данном разделе, представлены на дискете, приложенной к данной книге, в библиотеке "STL_Book" в разделе "Basic Functions" в функциональном блоке FB 104 и в исходном файле Chap_4.

## 4.1 Структура программируемого контроллера

На рис. 4.1 показана общая схема выполнения двоичной логической операции. Входной модуль выбирает датчик посредством адреса, например, датчик на входе I 1.2. CPU проверяет состояние сигнала ("status" - статус) датчика связывает его на входе блока логической операции с результатом логической операции, сохраненным после выполнения логической операции в предыдущий раз. Результат текущей логической операции (RLO) запоминается и хранится как "новый результат логической операции". Затем CPU обрабатывает следующее выражение программы, например, обеспечивающее сохранение результата логической операции в специальной ячейке памяти.

**Состояние сигнала (статус)**

Состояние (статус) бита эквивалентно состоянию (статусу) его сигнала и может иметь значения "0" или "1". В SIMATIC S7 состояние сигнала имеет значение "1" при наличии напряжения на входе, например, ~ 230 В или = 24 В (в зависимости от модуля); с другой стороны, если напряжение на входе отсутствует, то состояние сигнала входа соответствует "0".

Выражение, содержащее оператор проверки (check statement), инициирует проверку состояния бита. В то же время это выражение содержит правило логической операции - алгоритм, согласно которому результат проверки состояния бита будет сравниваться с сохраненным в процессоре результатом логической операции. Например выражение:

```
A   I 17.1
```

проверяет вход I 17.1 на состояние "1" и связывает с RLO по логике AND (И).

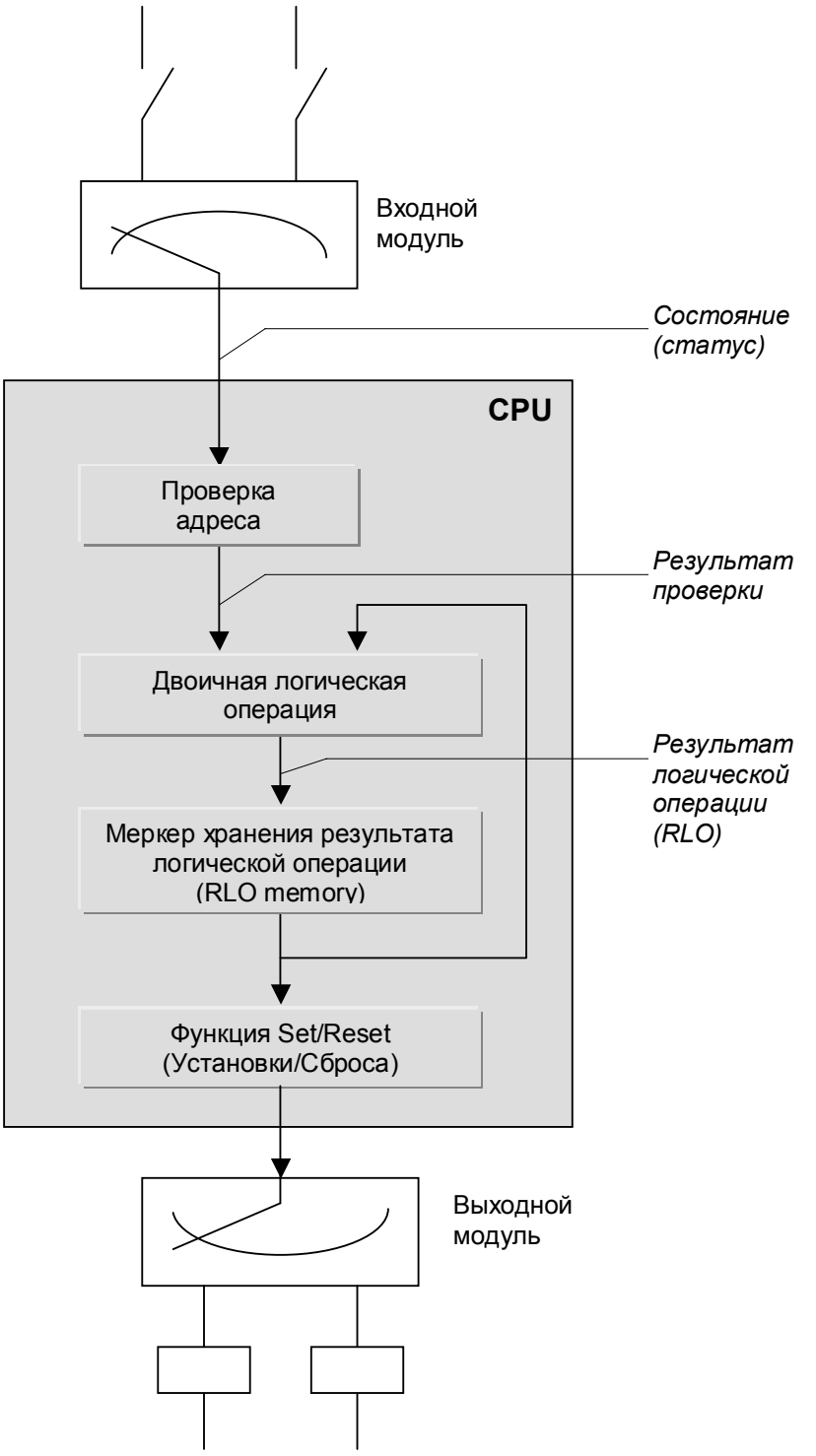



Рис. 4.1 Схема выполнения двоичной логической операции

Выражение:

ON M 20.5

проверяет меркер M 20.5 на состояние "0" и связывает (links) с RLO по правилу логической операции OR (ИЛИ).

**Результат проверки**

Строго говоря, CPU не связывает (link) состояние сигнала проверенного бита, а скорее формирует результат проверки: в случаях проверки на состояние сигнала "1" результат проверки идентичен состоянию сигнала проверяемого бита; в случаях проверки на состояние сигнала "0" результат проверки инвертируется относительно состояния сигнала проверяемого бита.

**Результат логической операции**

Результат логической операции (RLO) - это состояние сигнала в CPU, которое CPU в дальнейшем использует для обработки двоичных сигналов. Результат логической операции формируется и модифицируется с помощью операторов проверки (check statement). Если RLO имеет значение "1", то это означает, что условие двоичной логической операции выполнено, если RLO имеет значение "0", то это означает, что условие двоичной логической операции не выполнено. Биты устанавливаются или сбрасываются в соответствии с результатом логической операции.

**Логический шаг (logic step)**

Так же как можно выделить последовательный шаг в последовательной системе управления (sequential control system), так и в логической системе управления (logic control system) можно выделить логический шаг (logic step). На каждом таком логическом шаге формируется и оценивается (т.е. подвергается последующей обработке) результат логической операции. Логический шаг состоит из выражений с операторами проверки (также называемыми операторами сканирования - "scan statements") и выражений с условными операторами. Первый оператор проверки, следующий за условным оператором, называется "первичным опросом" ("first check").

В нижеследующей схеме логической системы управления выделен логический шаг:

```
…                …
=  Q 4.0        условный оператор (conditional statement)
A  I 2.0        первичный опрос (first check)
A  I 2.1        оператор проверки (check statement)
…               …
A  I 1.7        оператор проверки (check statement)
=  Q 5.1        условный оператор (conditional statement)
…               …
=  Q 4.3        условный оператор (conditional statement)
O  I 2.6        первая проверка (first check)
O  I 2.5        оператор проверки (check statement)
…               …
```

**Первичный опрос (First check)**

Первый оператор проверки, следующий за условным оператором, называется "первичным опросом" ("first check"). Он имеет особенное значение, потому что CPU имеет прямой доступ к результату этого оператора, как результату логической операции. Таким образом "старое значение" ("old") результата логической операции RLO теряется. Первичный опрос всегда соответствует началу логической операции. Правило (алгоритм) первичного опроса (AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ)) не играет при этом никакой роли.

**Оператор проверки (Check statement)**

Результат логической операции RLO формируется с помощью операторов проверки. Эти операторы проверяют сигнал бита на состояние "1" или "0" и связывают (link) его в соответствии с правилом AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ). Затем CPU сохраняет результат данной логической операции как новый результат логической операции RLO.

На рис. 4.2 показано, как программируется проверка сигнала бита на состояние "1" и "0". В случаях проверки на состояние сигнала "1" результат проверки идентичен состоянию сигнала проверяемого бита. В случаях проверки на состояние сигнала "0" результат проверки инвертируется относительно состояния сигнала проверяемого бита.

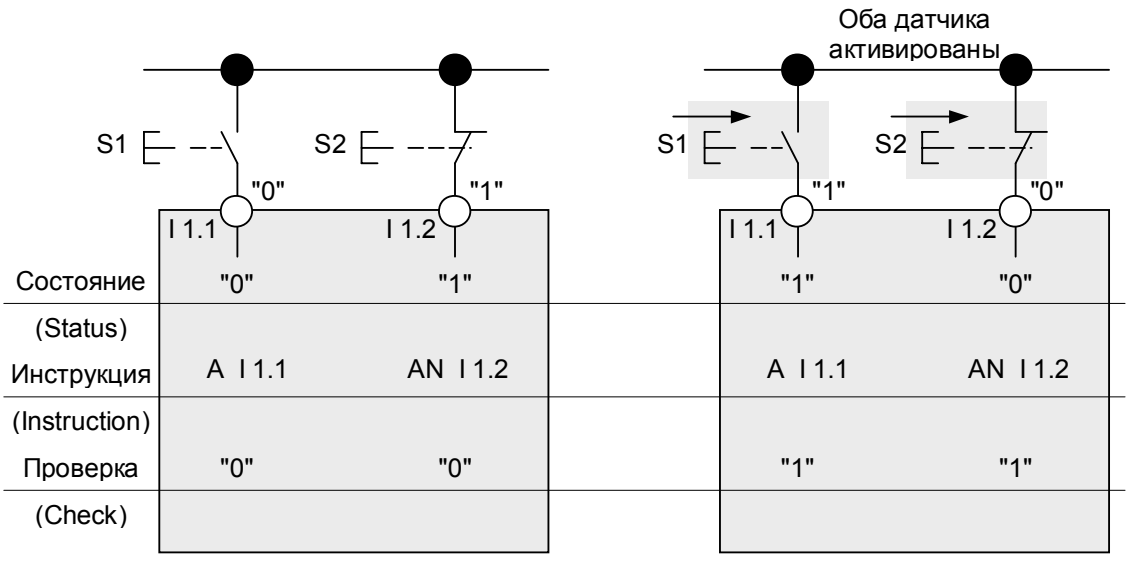



Рис. 4.2 Проверка состояния сигнала "1" и "0"

**Условные операторы (Condition statement)**

Условные операторы (condition statement) - это такие операторы, выполнение которых зависит от результата логической операции RLO. Эти операторы включают операции присвоения (назначения [assign]), установки (set) и сброса (reset) двоичных разрядов, а также запуск таймеров и счетчиков и т.п.

Условные операторы (за редким исключением) выполняются, когда результат логической операции RLO имеет состояние "1", и не выполняются, когда RLO имеет состояние "0". Условные операторы (за редким исключением) не влияют на результат логической операции RLO, и, таким образом, RLO на протяжении последовательного выполнения нескольких операторов остается неизменным.

**Правильный подход в программировании**

Правило (алгоритм) первичного опроса не имеет значения, так как результат этой проверки берется непосредственно как результат логической операции. В целях грамотного программирования правило (алгоритм) первичного опроса должно быть идентично требуемой функции. Например, последовательность операторов

```
…
=   Q 15.3
O   I 18.5      первая AND функция
A   I 21.7
=   Q 15.4

A   I 18.4      вторая AND функция
A   I 21.6
=   Q 15.5
…
```

представляет собой две AND-функции, из которых вторая функция (в которой обе проверки имеют логику AND) запрограммирована более правильным и более предпочтительным способом.

В случае отдельных операторов проверки, как например, в следующем фрагменте:

```
…
=   Q 10.0
A   I 20.1      присвоение
=   Q 10.1      I 20.1 -> Q 10.1
…
```

функция AND более предпочтительна.

## 4.2 Элементарные двоичные логические операции

В языке программирования STL используются элементарные двоичные функции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ). Эти функции связаны с проверкой сигнала на состояние "1" и "0".

A *адрес бита*
проверка сигнала на состояние "1" и
комбинирование в соответствии с функцией AND (И)

AN *адрес бита*
проверка сигнала на состояние "0" и
комбинирование в соответствии с функцией AND (И)

O *адрес бита*
проверка сигнала на состояние "1" и
комбинирование в соответствии с функцией OR (ИЛИ)

ON *адрес бита*
проверка сигнала на состояние "0" и
комбинирование в соответствии с функцией OR (ИЛИ)

X *адрес бита*
проверка сигнала на состояние "1" и
комбинирование в соответствии с Exclusive OR (Исключающее ИЛИ)

XN *адрес бита*
проверка сигнала на состояние "0" и
комбинирование в соответствии с Exclusive OR (Исключающее ИЛИ)

Проверка сигнала на состояние "1" дает результат проверки "1", если состояние сигнала бита соответствует "1". Проверка сигнала на состояние "0" дает результат проверки "1", если состояние сигнала бита соответствует "0".

CPU комбинирует результат проверки бита с текущим результатом логической операции RLO в соответствии с определенной функцией и формирует новое значение RLO. Если двоичная логическая операция следует сразу же за операцией с памятью, результат проверки вводится в RLO-буфер без выполнения логической операции.

Число двоичных функций и диапазон их действия теоретически произвольны. На практике, однако, ограничения на использование этих функций накладываются из-за конечных размеров блока и рабочей памяти CPU.

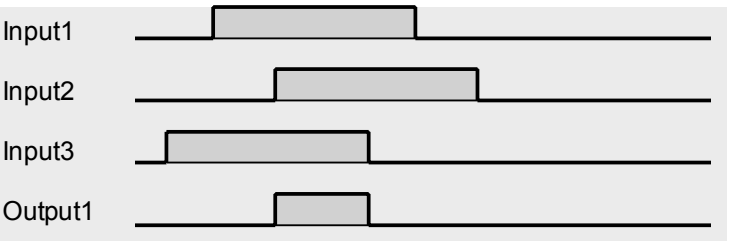
### 4.2.1 Функция AND (И)

Функция AND (И) связывает два двоичных сигнала и возвращает результат RLO, равный "1", если оба эти сигнала (оба результата проверки) равны "1". Если функция AND (И) встречается несколько раз подряд, то для того, чтобы общий результат RLO был равен "1", нужно, чтобы все промежуточные RLO были равны "1", иначе общий результат будет равен "0". На рис. 4.3 показан

Функция AND (И)

**Network 1 (Сегмент 1)**

```
A   Input1;
A   Input2;
A   Input3;
=   Output1;
```



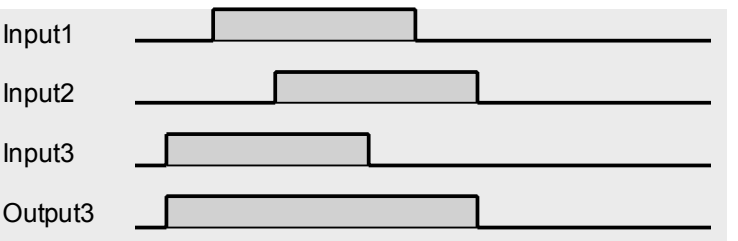

**Network 2 (Сегмент 2)**

```
A   Input4;
AN  Input5;
=   Output2;
```


Функция OR (ИЛИ)

**Network 3 (Сегмент 3)**

```
O   Input1;
O   Input2;
O   Input3;
=   Output3;
```




**Network 4 (Сегмент 4)**

```
O   Input4;
ON  Input5;
=   Output4;
```


Функция Exclusive OR (Исключающее ИЛИ)

**Network 5 (Сегмент 5)**

```
X   Input1;
X   Input2;
=   Output5;
```


**Network 6 (Сегмент 6)**

```
X   Input4;
XN  Input5;
=   Output6;
```

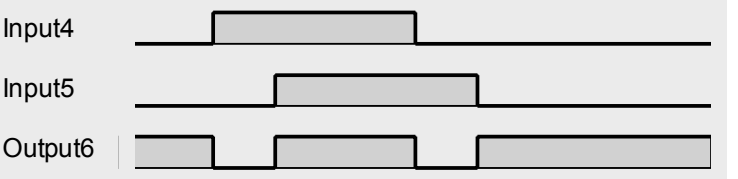



Рис. 4.3 Элементарные двоичные функции

пример действия функции AND (И). В сегменте 1 функция AND (И) имеет три входа (Input1 ... Input3) и один выход (Output1); за этими идентификаторами могут стоять произвольные адреса битов. На всех трех входах сигнал проверяется на состояние "1", так что сигналы этих битов связываются непосредственно по схеме AND (И). Когда все три входных бита дают при проверке состояние сигнала, равное "1", оператор присваивания устанавливает бит Output1 в состояние "1". Во всех других случаях условие AND (И) не выполняется, и бит Output1 находится в состоянии "0".

В сегменте 2 функция AND (И) имеет два входа, один из которых (Input5) инвертирующий, и один выход (Output2). Суть инвертирования для входа Input5 заключается в том, что сигнал на этом входе проверяется на состояние "0", т.е. результат проверки равен "1" при сигнале, равном "0". Поэтому условие AND (И) выполняется, когда бит Input4 находится в состоянии "1", а бит Input5 находится в состоянии "0".

### 4.2.2 Функция OR (ИЛИ)

Функция OR (ИЛИ) связывает два двоичных сигнала и возвращает результат RLO, равный "1", если хоть один из этих сигналов (один из результатов проверки) равен "1". Если функция OR (ИЛИ) встречается несколько раз подряд, то для того, чтобы общий результат RLO был равен "1", нужно, чтобы хоть один промежуточный RLO был равен "1". Если все промежуточные RLO равны "0", то общий результат будет равен "0". На рис. 4.3 показан пример действия функции OR (ИЛИ). В сегменте 3 функция OR (ИЛИ) имеет три входа (Input1 ... Input3) и один выход (Output3); за этими идентификаторами могут стоять произвольные адреса битов. На всех трех входах сигнал проверяется на состояние "1", так что сигналы этих битов связываются непосредственно по схеме OR (ИЛИ). Если хоть один из входных битов дает при проверке состояние сигнала, равное "1", следующий оператор присваивания устанавливает бит Output3 в состояние "1". Если все промежуточные RLO равны "0", то условие OR (ИЛИ) не выполняется, и бит Output3 устанавливается в состояние "0".

В сегменте 4 функция OR (ИЛИ) имеет два входа, один из которых (Input5) инвертирующий, и один выход (Output4). Суть инвертирования для входа Input5 заключается в том, что сигнал на этом входе проверяется на состояние "0", т.е. результат проверки равен "1" при сигнале, равном "0". Поэтому условие OR (ИЛИ) выполняется, когда бит Input4 находится в состоянии "1", или когда бит Input5 находится в состоянии "0".

### 4.2.3 Функция Exclusive OR (Исключающее ИЛИ)

Функция Exclusive OR (Исключающее ИЛИ) связывает два двоичных сигнала и возвращает результат RLO, равный "1", если оба эти сигнала (оба результата проверки) имеют разные значения; с другой стороны, RLO равен "0", если оба эти сигнала имеют одинаковое значение.

На рис. 4.3 показан пример действия функции Exclusive OR (Исключающее ИЛИ). В сегменте 5 функция Exclusive OR (Исключающее ИЛИ) имеет два входа (Input1 и Input2) и один выход (Output5); за этими именами могут стоять

произвольные адреса битов. На обоих входах сигнал проверяется на состояние "1", так что сигналы этих битов связываются непосредственно по схеме Exclusive OR (Исключающее ИЛИ). Если только один из входных битов дает при проверке состояние сигнала, равное "1", бит Output5 устанавливается в состояние "1". Если оба входных бита дают при проверке состояния сигнала значения, равные "1", или оба дают значения, равные "0", то бит Output5 устанавливается в состояние "0".

В сегменте 6 функция Exclusive OR (Исключающее ИЛИ) имеет два входа, один из которых (Input5) инвертирующий, и один выход (Output6). Суть инвертирования для входа Input5 заключается в том, что сигнал на этом входе проверяется на состояние "0", т.е. результат проверки равен "1" при сигнале, равном "0". Поэтому условие Exclusive OR (Исключающее ИЛИ) выполняется, когда оба входных бита находятся в одинаковом состоянии, т.е. имеют одинаковое состояние сигнала.

Если функция Exclusive OR (Исключающее ИЛИ) встречается несколько раз подряд, то для того, чтобы общий результат RLO был равен "1", нужно, чтобы нечетное число проверяемых входных битов дало результат проверки, равный "1".

### 4.2.4 Допущения, принимаемые по отношению к типам датчиков

Элементарные двоичные функции AND (И), OR (ИЛИ), Exclusive OR (Исключающее ИЛИ), описываемые в предыдущих разделах данной главы, рассматривались как входные модули с нормальноразомкнутыми контактами на входах (т.е. датчик имеет нормальноразомкнутые контакты, которые замыкаются при активации датчика, и при этом датчик возвращает значение сигнала, равное "1"). При выполнении функции управления, однако, не всегда возможно использовать только датчики, имеющие нормальноразомкнутые контакты. Во многих случаях, например, в случае "закороченных" цепей (имеющих короткое замыкание), использование нормальнозамкнутых контактов совершенно необходимо (нормальнозамкнутые контакты датчика обеспечивают возвращение сигнала, равного "0", при активации датчика).

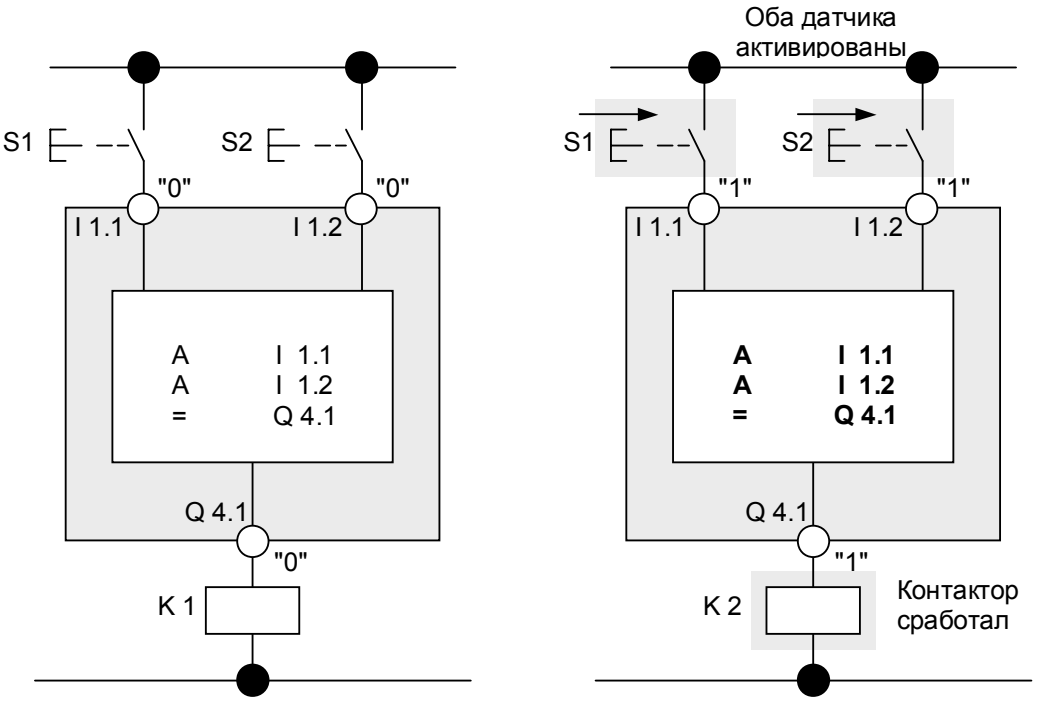
Если датчик, подключенный ко входу, имеет нормальноразомкнутые контакты, то при активации датчика на входе возникает значение сигнала, равное "1". Если датчик, подключенный ко входу, имеет нормальнозамкнутые контакты, то в неактивном состоянии датчика на входе сохраняется значение сигнала, равное "1". CPU "не знает" к какому типу относится датчик на том или ином входе (нормальнозамкнутый или нормальноразомкнутый). Он может только различить состояния сигналов "1" или "0".

При разработке программы, следовательно, необходимо учитывать тип используемого датчика.

Перед вводом программы Вы должны определиться с типом используемого датчика (с нормальнозамкнутыми или с нормальноразомкнутыми контактами). Это необходимо, потому что в отдельных частях программы учитывается состояние датчиков ("Sensor activated" [Датчик активирован], "Sensor not activated" [Датчик не активен]), а, следовательно, Вы должны проверять вход на состояние сигнала "1" или на состояние "0", в зависимости от типа используемого датчика.

На рис. 4.4 показаны варианты программы в зависимости от типа датчика.

Случай 1: оба датчика имеют нормальноразомкнутые контакты:




Случай 2: один нормальноразомкнутый и один нормальнозамкнутый датчики:

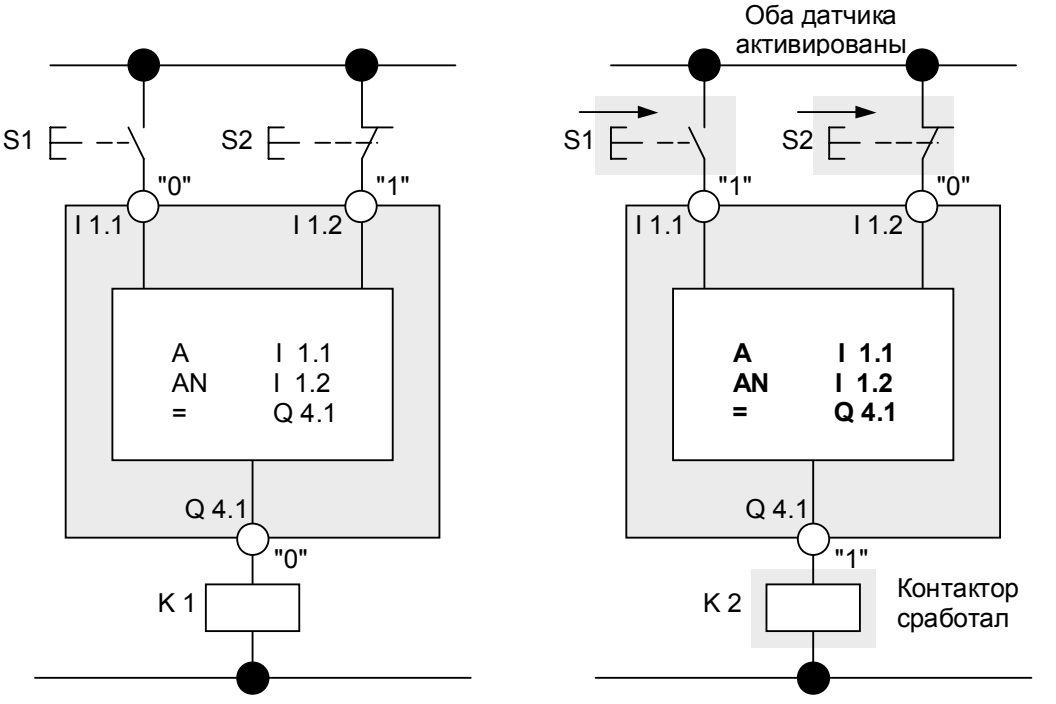



Рис. 4.4 Применение различных типов датчиков

Из вышеприведенного рисунка следует, что благодаря программам, учитывающим тип датчика, и в первом случае (когда оба подключенных к PLC датчика имеют нормальноразомкнутые контакты), и во втором случае (когда к PLC подключены нормальноразомкнутый и нормальнозамкнутый датчики) контакторы, подключенные к выходам, срабатывают при активации двух датчиков на входах.

Если оба подключенных ко входам датчика с нормальноразомкнутыми контактами активируются, сигналы на входах принимают значение "1". Для выполнения условия функции AND (И) с результатом проверки "1", входы функции проверяются на состояние "1". Если активируется датчик с нормальнозамкнутыми контактами, сигнал на входе принимает значение "0". Для выполнения в данном случае условия функции AND (И) с результатом проверки "1", такой вход функции должен проверяться на состояние "0".

## 4.3 Инвертирование результата логической операции

Оператор NOT (НЕ) инвертирует результат логической операции. Вы можете использовать оператор NOT (НЕ), например, для инвертирования результата выполнения функции AND (И) (см. рис.4.5. Сегмент 7. Функция NAND [НЕ-И]). На этом же рисунке в сегменте 8 показано инвертирование функции OR (ИЛИ), вызванное функцией NOR (НЕ-ИЛИ).

Вы можете найти дополнительные примеры использования оператора NOT (НЕ) в разделе 4.4.6 "Инвертирование во вложенных операторах".

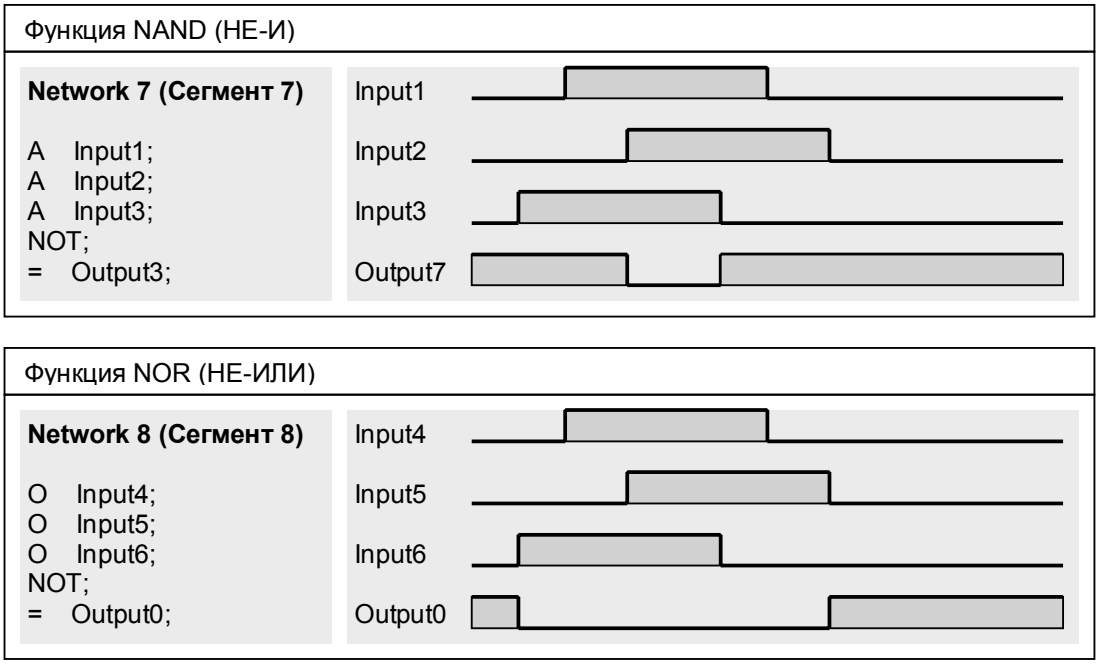



Рис. 4.5 Примеры применения функции NOT (НЕ)

## 4.4 Сложные двоичные логические операции

Двоичные логические функции могут быть объединены, например, функции AND (И) и OR (ИЛИ) могут стоять в программе в любом порядке. Если такие функции стоят в программе в произвольном порядке, нелегко разобраться в обработке их CPU. Поэтому лучше использовать для иллюстрации решения задачи, например, программирование с помощью функциональных блок-схем, чем программирование на STL.

При программировании сложных двоичных логических операций STL одинаково рассматривает операторы OR (ИЛИ) и Exclusive OR (Исключающее ИЛИ) как операторы с одинаковым приоритетом. Оператор AND (И) имеет более высокий приоритет и выполняется "перед" операторами OR (ИЛИ) и Exclusive OR (Исключающее ИЛИ).

Чтобы функции выполнялись в требуемом порядке CPU иногда должен временно сохранять значение функции (результат логической операции RLO, рассчитанный в некоторой точке программы). Для этой цели (для временного хранения результата) могут использоваться вложенные выражения (вложенные операторы). Как и в случае используемых в булевой алгебре записей вложенные операторы обеспечивают выполнение одних функций раньше других. Вложенные выражения (операторы) могут также включать в себя функцию OR (ИЛИ).

Язык программирования STL позволяет использование следующих двоичных вложенных выражений (вложенных операторов):

| | |
|---|---|
| O | функция OR (ИЛИ) для функций AND (И) |
| A( | открывающая скобка с функцией AND (И) |
| O( | открывающая скобка с функцией OR (ИЛИ) |
| X( | открывающая скобка с функцией Exclusive OR (Исключающее ИЛИ) |
| AN( | открывающая скобка с функцией NOT-AND (НЕ-И) |
| ON( | открывающая скобка с функцией NOT-OR (НЕ-ИЛИ) |
| XN( | открывающая скобка с функцией NOT-Exclusive OR (НЕ-Исключающее ИЛИ) |
| ) | закрывающая скобка. |

Правило логики для выражения с открывающей скобкой показывает, как результат вложенного выражения должен быть связан с текущим значением RLO в момент обработки закрывающей скобки. До этой логической операции результат выполнения вложенного выражения инвертируется, если присутствует символ операции инвертирования.

### 4.4.1 Обработка вложенных выражений (вложенных операторов)

В языке программирования STL двоичные вложенные выражения используются для определения порядка выполнения двоичных логических операций. В процессе выполнения программы CPU первыми обрабатывает выражения, заключенные в скобки, то есть до выполнения выражений, находящихся за скобками.

Когда CPU встречает открывающую скобку, запоминает текущее значение RLO и затем обрабатывает выражение в скобках (вложенное выражение), когда CPU встречает закрывающую скобку, он связывает значение RLO для вложенного выражения с ранее запомненным значением RLO в соответствии с функцией, определенной при открывающей скобке (Рис. 4.6).

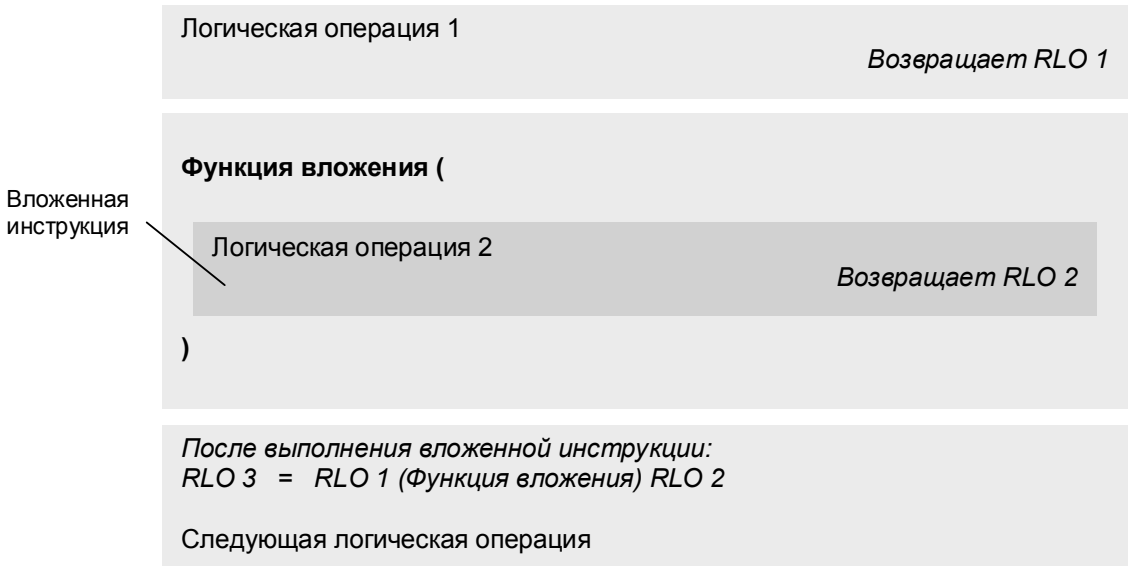



Рис. 4.6 Обработка вложенных выражений

Оператор проверки, следующий за открытой скобкой, всегда является первичным опросом, т.к. CPU всегда создает новый результат логической операции RLO для вложенного выражения. Оператор проверки, следующий за закрытой скобкой, никогда не является первичным опросом, т.к. первой инструкцией является вложенное выражение. CPU обрабатывает значение RLO для вложенного выражения как результат первичного опроса.

Вложенные выражения могут включать в себя другие вложенные выражения (см. рис. 4.7). Глубина вложения равна 7, что означает, что Вы можете 7 раз включать новые вложенные выражения в выражения, которые уже сами по себе являются вложенными, не завершая последних. Обработка внутренних скобок ведется, как описано выше.

**Сохранение промежуточных результатов с помощью стека вложения (nesting stack)**

При обработке вложенных выражений в CPU заполняется "стек вложения" ("nesting stack") в порядке обработки вложенных функций. В данном стеке хранится следующая информация:

- результат логической операции (RLO) предыдущих скобок;
- двоичный результат (BR "binary result") предыдущих скобок;
- бит состояния (OR) (показывающий, было ли уже выполнено условие функции OR [ИЛИ]);
- функция вложения (для запоминания функции, с которой необходимо связать результат вложенного выражения).

```
X(   ;              // функция XOR (6)
A(   ;              // функция AND (4)
O   Input0;         // функция OR  (1)
O   Input1;
)    ;
A(   ;
O   Input2;         // функция OR  (2)
O   Input3;
)    ;
A   Input4;
)    ;
X(   ;              // функция XOR (6)
A   Input5;         // функция AND (5)
A(   ;
O   Input6;         // функция OR  (3)
O   Input7;
)    ;
)    ;
=   Output20;
```

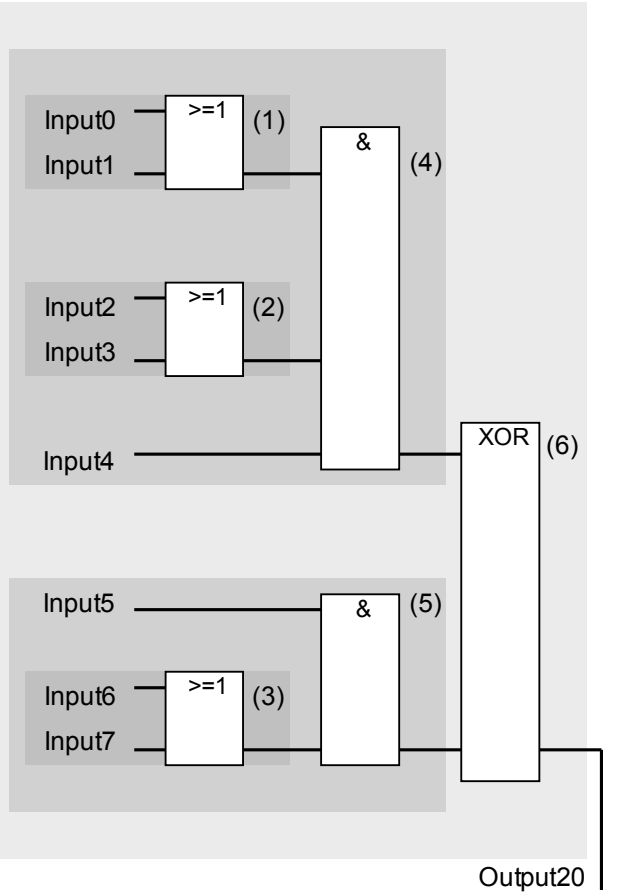



Рис. 4.7 Пример реализации вложенных выражений (операторов)

CPU устанавливает двоичный результат (BR) после закрывающей скобки в состояние, которое имело место к началу обработки вложенного выражения. Во вложенных выражениях Вы можете использовать не только двоичные логические операторы, но и все остальные выражения языка программирования STL. Однако, от программиста требуется внимание при завершении сложного вложенного выражения закрывающей скобкой. Так, возможно, например, во вложенном выражении запрограммировать несколько логических шагов, или операций с памятью, или функций сравнения.

## 4.4.2 Объединение AND-функций (И) в операторе OR (ИЛИ)

Эти логические операции, представляющие собой комбинации функций OR (ИЛИ) и AND (И), могут быть записаны в булевой алгебре без использования скобок. Здесь действует правило, согласно которому функции AND (И) выполняются в первую очередь. Затем результаты, если есть функция OR (ИЛИ), обрабатываются (link - связываются) этой функцией.

Пример:

```
A     Input0;
A     Input1;
O     ;
A     Input2;
A     Input3;
=     Output8;
```

В данном примере единственный оператор O (соответствующий функции OR (ИЛИ)) находится между двумя функциями AND (И).

*В примере Output8 устанавливается, если { Input0* И *Input1}* ИЛИ *{ Input2* И *Input3}* установлены (имеют значение "1").

### 4.4.3 Объединение OR (ИЛИ) и Exclusive OR (Исключающее ИЛИ) в операторе AND (И)

Эти логические операции, представляющие собой комбинации функций OR (ИЛИ) и AND (И), должны быть записаны в булевой алгебре с использованием скобок, с помощью которых указывается, что функции OR (ИЛИ) выполняются в первую очередь, раньше функции AND (И).

Пример:

```
A(    ;
O     Input0;
O     Input1;
)     ;
A(    ;
O     Input2;
O     Input3;
)     ;
=     Output10;
```

В данном примере оператор открытой скобки объединен с функцией AND (И). Функция OR (ИЛИ) находится во вложенном выражении. Закрывающая скобка в данном случае связывает (link) результат функции OR (ИЛИ) (результат логической операции, которая заключена в скобки) с дополнительными проверками, если они есть, по логике функции AND (И).

*В примере Output10 устанавливается, если { Input0* ИЛИ *Input1}* И *{ Input2* ИЛИ *Input3}* установлены (имеют значение "1").

Обработка функций Exclusive OR (Исключающее ИЛИ) в операторе AND (И) производится и записывается точно таким же образом. В вышеуказанном примере функции OR (ИЛИ) должны быть заменены на функции Exclusive OR (Исключающее ИЛИ), так как они имеют одинаковый приоритет.

### 4.4.4 Объединение функций AND (И) в операторе Exclusive OR (Исключающее ИЛИ)

Функции AND (И), выполняющиеся перед функцией Exclusive OR (Исключающее ИЛИ), должны быть записаны в скобках. С помощью скобок CPU сохраняет результат выполнения AND (И) функции и затем их комбинирует, возможно с дополнительными проверками, в соответствии с правилами функции Exclusive OR (Исключающее ИЛИ).

Пример:

```
X(   ;
A    Input0;
A    Input1;
)    ;
X(   ;
A    Input2;
A    Input3;
)    ;
=    Output12;
```

В данном примере первая функция AND (И) не нуждается в скобках, так как функция AND (И) имеет более высокий приоритет, чем функция Exclusive OR (Исключающее ИЛИ). Тем не менее, при наличии скобок программа лучше читается.

*В примере Output12 устанавливается, если только в одной из скобок {Input0 И Input1} и { Input2 И Input3}* выполнено условие функции AND (И).

### 4.4.5 Объединение функций OR (ИЛИ) в операторе Exclusive OR (Исключающее ИЛИ)

Функции OR (ИЛИ), выполняющиеся перед функцией Exclusive OR (Исключающее ИЛИ), должны быть записаны в скобках. С помощью скобок CPU сохраняет результат выполнения OR (ИЛИ) функции и затем их комбинирует, возможно с дополнительными проверками, в соответствии с правилами функции Exclusive OR (Исключающее ИЛИ).

Пример:

```
X(   ;
O    Input0;
O    Input1;
)    ;
X(   ;
O    Input2;
O    Input3;
)    ;
=    Output14;
```

*В примере Output14 устанавливается, если только в одной из скобок {Input0 ИЛИ Input1} и { Input2 ИЛИ Input3} выполнено условие функции OR (ИЛИ).*

Обработка функций Exclusive OR (Исключающее ИЛИ) в операторе OR (ИЛИ) производится и записывается точно таким же образом. Для такого случая в вышеуказанном примере функции OR (ИЛИ) должны быть заменены на функции Exclusive OR (Исключающее ИЛИ) и наоборот, так как они имеют одинаковый приоритет.

### 4.4.6 Инвертирование вложенных выражений

Точно также как Вы можете проверить бит на состояние сигнала "0" (по сути инвертировать состояние бита), Вы также можете инвертировать вложенное выражение (точнее - инвертировать результата этого выражения). Это означает, что CPU после вычисления вложенного выражения должен "взять" его результат в инвертированной форме. Признаком инвертирования результата операции является наличие дополнительного символа N в выражении открывающей скобки.

Пример:

```
AN(   ;
O    Input0;
O    Input1;
)    ;
AN(   ;
X    Input2;
X    Input3;
)    ;
=    Output16;
```

*В примере Output16 устанавливается, если ни в одной из скобок - ни в {Input0 ИЛИ Input1}, ни в { Input2 Исключающее ИЛИ Input3} - не выполняется условие вложенных функций.*

Второй способ инвертирования вложенных выражений возможен с помощью использования выражения (оператора) NOT. Оператор NOT, записанный перед закрывающей скобкой, инвертирует результат вложенного выражения перед последующей обработкой.

Пример:

```
A(    ;
O    Input0;
O    Input1;
NOT ;
)    ;
```

```
A(    ;
X    Input2;
X    Input3;
NOT ;
)    ;
=    Output17;
```

В данном примере имеет место такая же логическая операция, как и в предыдущем примере. Инвертирование вложенных выражений обеспечивается с помощью использования выражения (оператора) NOT внутри вложенных выражений.

# 5 Операции с памятью (memory functions)

В данной главе рассматриваются операции с памятью, применяемые в языке программирования STL. К операциям с памятью относятся функции Assign (Присвоение) для "динамичного" управления битами, а также функции Set (Установка бита) и Reset (Сброс бита) для "статического" управления битами. Также к операциям с памятью относится функция проверки наличия фронта сигнала.

Операции с памятью используются в сочетании с двоичными логическими операциями, чтобы воздействовать на значения сигналов (состояния) битов с использованием "результата логической операции" RLO, который генерируется CPU.

Вы можете использовать функции работы с памятью для управления всеми битовыми адресами: адресами области отображения входов/выходов процесса, адресами меркеров, адресами глобальных данных, адресами статических и временных локальных данных.

Примеры, рассмотренные в данной главе, содержатся на дискете, прилагающейся к данной книге в библиотеке STL_Book в разделе "Basic functions" ("Базовые функции") в функциональном блоке FB 105 и исходном файле Chap_5.

## 5.1 Функция Assign (Присвоение)

Синтаксис:

= *Bit*

Функция присваивает биту результат логической операции

Выражение, содержащее символ присвоения "=", назначает результат логической операции RLO, содержащийся в CPU, биту, указанному в выражении. Если результат логической операции имеет значение "1", то бит устанавливается; если результат логической операции имеет значение "0", то бит сбрасывается (см. рис. 5.1. Сегмент 1).

Если необходимо установить бит, в то время когда RLO имеет значение "0", то значение RLO должно быть сначала инвертировано с помощью оператора NOT до момента выполнения функции присвоения (см. рис. 5.1. Сегмент 2).

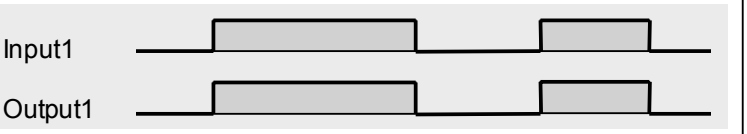
Вы можете найти дополнительные примеры применения функции присвоения в главе 4 "Двоичные логические операции".

**Одновременное выполнение нескольких функций присвоения**

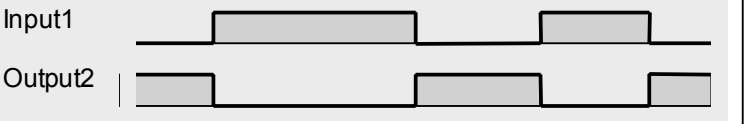
Также допускается выполнять одновременное присваивание результата логической операции различным битам, запрограммировав подряд несколько операторов присваивания соответствующим битам (см. рис. 5.1. Сегмент 3).

Функция Assign (Присвоение значения биту)

**Network 1 (Сегмент 1)**
A Input1;
= Output1;

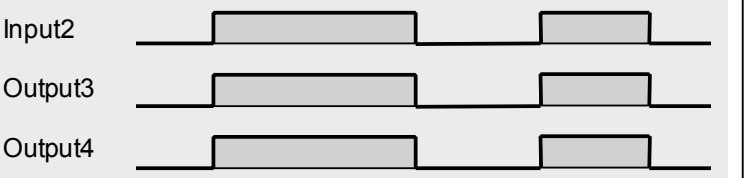



**Network 2 (Сегмент 2)**
A Input1;
NOT;
= Output2;



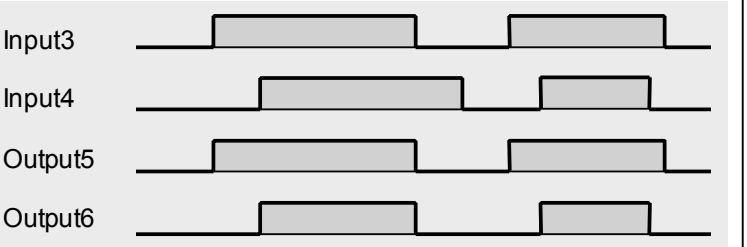

**Network 3 (Сегмент 3)**

A Input2;
= Output3;
= Output4;




**Network 4 (Сегмент 4)**

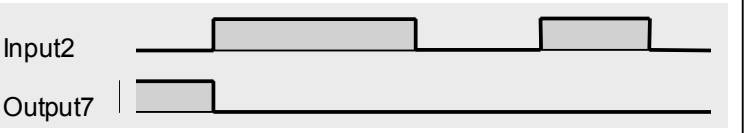
A Input3;
= Output5;
A Output5;
A Input4;
= Output6;




Функции Set и Reset (Установка и сброс бита)

**Network 5 (Сегмент 5)**
A Input1;
S Output7;


**Network 6 (Сегмент 6)**
A Input2;
R Output7;




**Network 7 (Сегмент 7)**

A Input3;
A Input4;
S Output8;
R Output9;
S Output10;
= Output11;

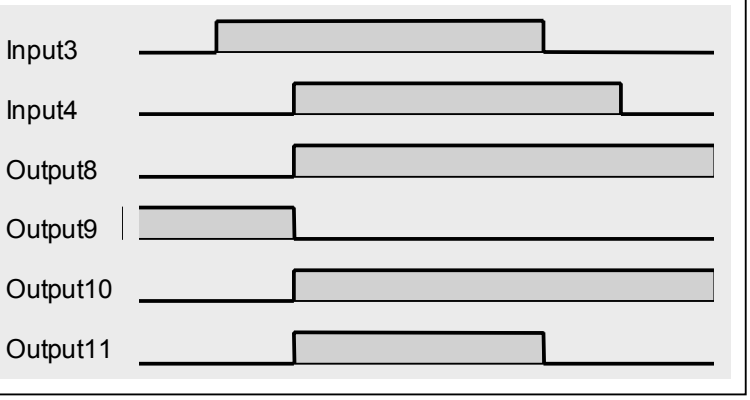
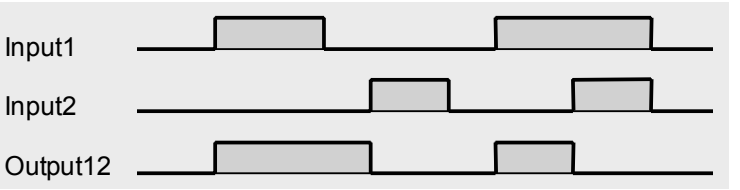



Рис. 5.1 Функции Assign (Присвоение), Set (Установка) и Reset (Сброс)

Функции Set и Reset (Установка и сброс бита)

**Network 8 (Сегмент 8)**
A Input1;
S Output12;
A Input2;
R Output12;




**Network 9 (Сегмент 9)**
A Input3;
R Output13;
A Input4;
S Output13;

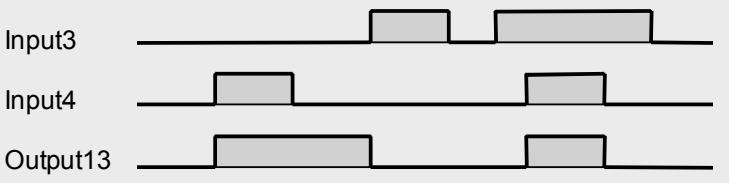



Рис. 5.1 (*продолжение*) Функции Set (Установка) и Reset (Сброс)

Все биты, определенные в функции присвоения Assign, реагируют на эту функцию одинаково, так как инструкции, используемые для управления битами, не влияют на RLO. CPU не генерирует новый результат логической операции RLO до появления следующего оператора проверки.

Если необходимо проверить состояние сигнала выхода в другой логической операции просто проверьте этот выход с помощью соответствующей функции проверки (см. рис. 5.1. Сегмент 4).

## 5.2 Функции Set (Установка бита) и Reset (Сброс бита)

Синтаксис:

S *Bit*

Функция устанавливает бит, в случае если результат логической операции равен "1".

R *Bit*

Функция сбрасывает бит, в случае если результат логической операции равен "1".

Функция Set (Установка бита) устанавливает бит, а функция Reset (Сброс бита) сбрасывает бит только в случае, если результат логической операции RLO имеет значение "1". Если результат логической операции имеет значение "0", то инструкции Set и Reset не изменяют состояния бита, определенного как операнд в этих инструкциях (см. рис. 5.1. Сегменты 5 и 6).

**Одновременное выполнение нескольких операций с памятью**

Также допускается использование нескольких операций с памятью одновременно и в любой комбинации, а также совместно с функциями присвоения Assign с использованием одного и того же результата логической

операции. Просто запишите в программе подряд несколько операторов установки или сброса (Set/Reset) для соответствующих битов (см. рис. 5.1. Сегмент 7). Во время действия функции присвоения Assign, установки Set и сброса Reset результат логической операции не изменяет своего значения. При этом CPU не генерирует новый результат логической операции RLO до появления следующего оператора проверки. Также Вы можете использовать оператор NOT (НЕ) для инвертирования значения RLO в одном ряду с операциями с памятью.

Для большей ясности и лучшей читаемости программы рекомендуется использовать операторы Set (Установка бита) или Reset (Сброс бита) парами для определенного бита и только один раз.

## 5.3 Функции RS Flipflop (RS-триггер)

Функция RS Flipflop (RS-триггер) состоит из одного выражения с оператором Set (Установка бита) и одного выражения, содержащего оператор Reset (Сброс бита); данная функция не имеет специального идентификатора в языке программирования STL. Функция RS Flipflop (RS-триггер) реализуется при последовательной записи выражения с оператором Set (Установка бита) и выражения с оператором Reset (Сброс бита) с общим для этих операторов операндом - одним и тем же битом. С точки зрения функциональных свойств триггера очень важным является порядок, в котором записываются вышеуказанные выражения.

Надо отметить, что биты, используемые в операциях с памятью, обычно сбрасываются при запуске (при полном перезапуске). В особых случаях состояния битов, используемых в операциях с памятью, сохраняют свои значения; это может зависеть от типа запуска (например, "теплый" перезапуск), от используемого бита (например, бит расположен в области статических локальных данных) и от установок CPU (имеются в виду установки свойства реманентности).

### 5.3.1 Операции с памятью при установленном приоритете функции Reset (Сброс бита)

Приоритет функции Reset (Сброс бита) здесь означает, что при одновременном управлении битом с помощью инструкций Set (Установка бита) и Reset (Сброс бита) и при одновременной выработке этими инструкциями управляющего сигнала "1", указанный бит будет сброшен (значение его сигнала будет равно "0"). Таким образом, здесь имеет место случай, когда инструкция Reset (Сброс бита) имеет приоритет над инструкцией Set (Установка бита) (см. рис. 5.1. Сегмент 8).

Так как выражения обрабатываются последовательно, CPU сначала устанавливает бит, первой выполняя инструкцию Set (Установка бита), затем сбрасывает этот бит, выполняя инструкцию Reset (Сброс бита). После этого до конца цикла сканирования программы выход остается в сброшенном состоянии.

Если рассматриваемый бит является выходом, то столь короткое по времени

его пребывание в установленном состоянии имеет место лишь в области отображения процесса, на этом промежутке времени состояние соответствующего связанного выхода (внешнего) выходного блока не изменяется. До конца цикла сканирования программы CPU не пересылает в выходные модули таблицу выходов образа процесса.

Фактически приоритет функции Reset (Сброс бита) является "стандартной" формой использования данной функции, так как состояние сброса (состояние "0") является, как правило, более безопасным.

### 5.3.2 Операции с памятью при установленном приоритете функции Set (Установка бита)

Приоритет функции Set (Установка бита) здесь означает, что при одновременном управлении битом с помощью инструкций Set (Установка бита) и Reset (Сброс бита) и при одновременной выработке этими инструкциями управляющего сигнала "1", указанный бит будет установлен (значение его сигнала будет равно "1"). Таким образом, здесь имеет место случай, когда инструкция Set (Установка бита) имеет приоритет над инструкцией Reset (Сброс бита) (см. рис. 5.1. Сегмент 9).

Так как выражения обрабатываются последовательно, CPU сначала сбрасывает этот бит, первой выполняя инструкцию Reset (Сброс бита), выполняя инструкцию Set (Установка бита), затем устанавливает этот бит. После этого до конца цикла сканирования программы выход остается в установленном состоянии.

Если рассматриваемый бит является выходом, то его пребывание в установленном состоянии имеет место в области отображения процесса до конца цикла сканирования программы, на этом промежутке времени состояние соответствующего связанного выхода (внешнего) выходного блока не изменяется. До конца цикла сканирования программы CPU не пересылает в выходные модули таблицу выходов образа процесса.

Фактически приоритет функции Set (Установка бита) является скорее исключением, чем правилом, как форма использования данной функции. Обычно приоритет функции Set (Установка бита) используется, например, для выставления сигнала отказа, в случае когда несмотря на подтверждение на входе Reset (Сброс), текущий сигнал отказа должен поддерживать определенный бит в установленном состоянии с использованием операций с памятью.

### 5.3.3 Операции с памятью в сочетании с двоичными логическими функциями

В языке программирования STL Вы можете свободно использовать операции с памятью. С их помощью можно сохранять результат логической операции RLO в любом месте программы, чтобы в дальнейшем можно было его использовать.

На рис. 5.2 показано, как можно вместо использования вложенных выражений использовать временное хранение результата логической операции RLO.

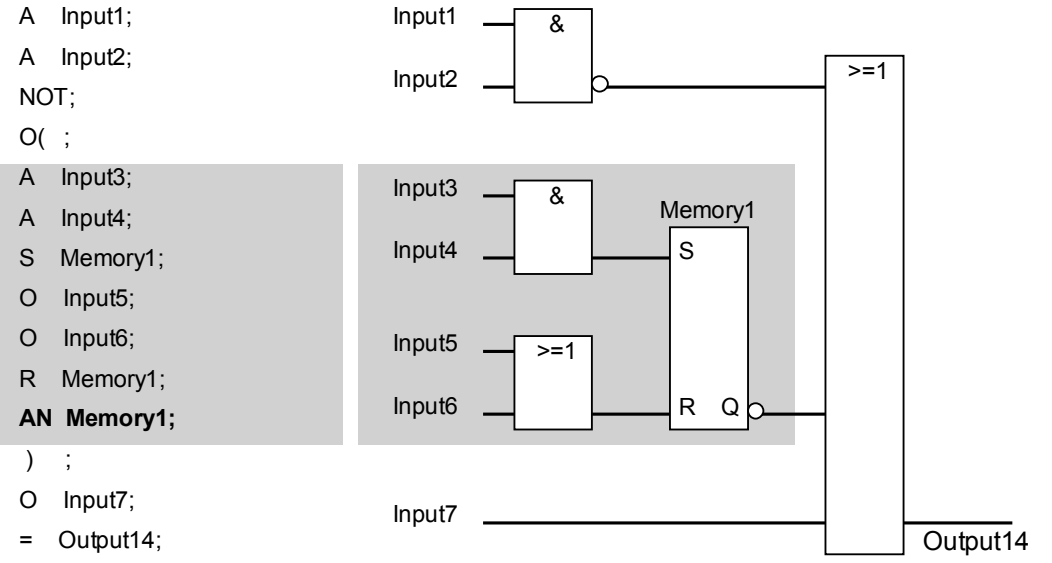



Рис. 5.2 Пример: промежуточный буфер для хранения значения бита

Здесь внутри вложенного выражения используется RS-триггер.

Состояние его сигнала используется в дальнейшем. Поэтому необходимо опросить состояние RS-триггера в конце вложенного выражения для получения состояния сигнала RS-триггера перед оператором закрытия скобки. Если этот оператор отсутствует, то состояние сигнала логической операции перед входом сброса будет в этом случае обрабатываться в дальнейшем.

Внутри скобок Вы можете запрограммировать любое STL-выражение; однако, перед закрытием скобок Вы должны убедиться в том, что Вы получили во вложенном выражении требуемое значение результата логической операции RLO.

**Промежуточные двоичные результаты**

Почти все биты могут использоваться для временного хранения двоичных результатов, в том числе:

- Биты из области временных локальных данных, являющиеся наиболее подходящим вариантом, если Вам необходимо сохранять промежуточные результаты только внутри блока. Все кодовые блоки имеют области временных локальных данных.
- Биты из области статических локальных данных, доступные только в функциональных блоках и при этом сохраняющие заданные состояния вплоть до нового их определения.

- Биты из области меркеров, предоставляющие повсеместный доступ. Количество меркеров определяется типом CPU. Для четкости работы программы избегайте "множественного" использования меркеров (т.е. избегайте использования одних и тех же меркеров для разных целей).
- Биты в блоках глобальных данных, также предоставляющие повсеместный доступ в программе. Но для того, чтобы их можно было использовать, необходимо сначала открыть соответствующий блок данных (даже если для этого используется полная адресация).

Примечание:

Вы можете заменить "сверхоперативную память" ("scratch-pad memory"), используемую в STEP 5, на область временных локальных данных, которая предоставляется в каждом блоке.

## 5.4 Функция Edge Evaluation (Проверка наличия фронта сигнала)

Синтаксис:

FP *Bit*

Функция проверки наличия положительного или переднего (возрастающего) фронта сигнала

FN *Bit*

Функция проверки наличия отрицательного или заднего (убывающего) фронта сигнала

Функция проверки наличия переднего или заднего фронта сигнала есть функция для определения наличия изменений в состоянии сигнала (изменения его уровня). Наличие переднего фронта сигнала свидетельствует о переходе сигнала от уровня "0" к уровню "1". Соответственно, наличие заднего фронта сигнала свидетельствует о переходе сигнала от уровня "1" к уровню "0".

В логических переключающих схемах эквивалентом функции проверки наличия фронта сигнала является "контактный формирователь импульса" ("pulse contact element"). При включении реле этот формирователь импульса генерирует импульс, свидетельствующий о наличии возрастающего фронта сигнала. При выключении реле этот формирователь импульса генерирует импульс, свидетельствующий о наличии убывающего фронта сигнала.

К биту, указанному как операнд в функции проверки наличия фронта сигнала, обращаются как к "меркеру фронта" ("edge memory bit") (хотя фактически этот бит может быть и не из области меркеров). Тем не менее, он должен быть таким битом, состояние сигнала которого может быть проверено в любой момент в последующих циклах сканирования программы, и который не используется в программе каким-либо другим образом. Битом в функции проверки наличия фронта сигнала может быть меркер, бит из блока глобальных данных или бит из статических локальных данных в функциональных блоках. (В дальнейшем рассматриваемый бит именуется для определенности "меркером фронта").

Итак, вышеупомянутый меркер фронта сохраняет "старое" значение RLO, которое CPU использовал при последней проверке наличия фронта сигнала.

При каждой новой проверке наличия фронта сигнала CPU сравнивает текущее значение RLO с состоянием меркера фронта. Фронт сигнала будет обнаружен, если эти два сигнала будут иметь различные состояния. В случае обнаружения фронта сигнала CPU обновляет состояние меркера фронта, посредством назначения последнему текущего значения RLO, и устанавливает для RLO значение "1" после положительного или отрицательного фронта сигнала, в зависимости от инструкции функции проверки наличия фронта сигнала. В том случае, если CPU не определяет присутствия фронта сигнала, он устанавливает для RLO значение "0".

Таким образом, состояние бита, равное "1", означает факт обнаружения фронта сигнала. Это состояние бита сохраняется короткое время, как правило, в течение одного цикла сканирования программы. Это происходит из-за того, что CPU не обнаруживает фронта сигнала при следующей проверке наличия фронта сигнала (если значение проверяемого бита не изменяется). Поэтому CPU вновь возвращает RLO значение "0" при следующей проверке наличия фронта.

Вы можете использовать RLO сразу после выполнения проверки наличия фронта сигнала или можете сохранить его значение в бите, называемом "меркер импульса" ("pulse memory bit"). Используйте для сохранения RLO меркер импульса, если Вы должны обработать значение RLO в другом месте программы; это удобный буфер хранения сигнала о наличии фронта сигнала. В качестве "меркера импульса" также может использоваться собственно меркер, бит из блока глобальных данных, а также бит из временных или бит из статических локальных данных.

Непосредственно после выполнения функции обнаружения фронта сигнала Вы можете использовать полученное значение RLO с помощью функций проверки AND (И), OR (ИЛИ) или Exclusive OR (Исключающее ИЛИ).

Проверьте реакцию функции обнаружения фронта сигнала после включения CPU. Для обнаружения фронта сигнала нужно, чтобы RLO до выполнения функции обнаружения фронта и состояние меркера фронта были одинаковыми. При определенных обстоятельствах меркер фронта должен быть сброшен при запуске (в зависимости от требуемой реакции функции проверки наличия фронта, а также от используемого бита).

В следующих примерах показано, как выполняется функция проверки наличия фронта сигнала. На упрощенной схеме показаны вход, начиная с момента времени до начала выполнения проверки наличия фронта синала и меркер импульса ("pulse memory bit"). Понятно, что функции проверки наличия фронта сигнала могут предшествовать, а также после нее могут выполняться двоичные логические операции.

### 5.4.1 Положительный фронт сигнала

CPU определяет положительный (возрастающий) фронт сигнал, когда до начала выполнения функции проверки наличия фронта результат логической операции изменяется от уровня "0" к уровню "1". Процесс обработки сигналов показан на верхней половине рис. 5.3.

На рисунке 5.3 меркеры импульса имеют имена *PulseMerker*X, а меркеры фронта, соответственно, имеют имена *FrontMerker*X.

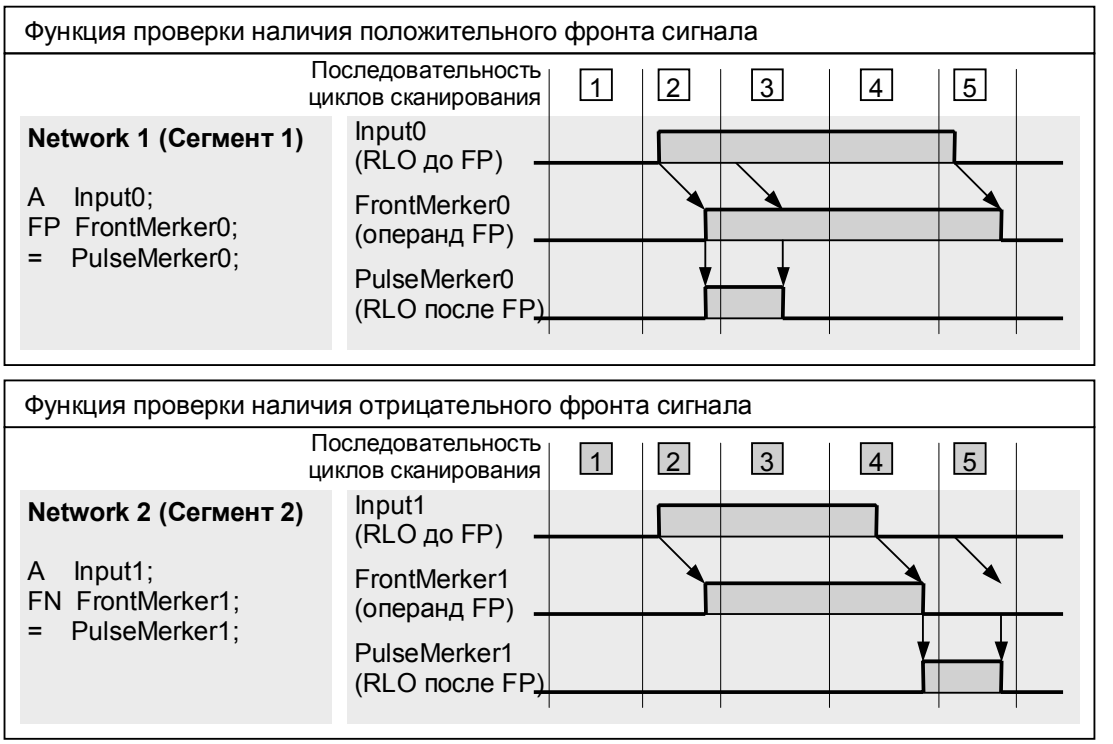



Рис. 5.3 Функция проверки наличия фронта сигнала

На рис. 5.3 показана следующая последовательность циклов сканирования:

1 Сначала состояния и входа Input0, и меркера фронта FrontMerker0 соответствуют уровню "0". Меркер импульса PulseMerker0 также сброшен, т.е. уровень его сигнала равен "0".

2 На 2-ом цикле сканирования состояние входа Input0 изменяется с "0" на "1". CPU обнаруживает это изменение при сравнении текущего значения RLO с состоянием меркера фронта FrontMerker0. Если Input0 равен "1", а меркер фронта FrontMerker0 равен "0", то значение меркера фронта устанавливается (становится равным "1"). Текущее значение PulseMerker0 также равен "1".

3 На 3-ем цикле сканирования при сравнении текущего значения RLO с состоянием меркера фронта FrontMerker0 CPU обнаруживает, что они имеют один уровень. Поэтому он устанавливает для PulseMerker0 значение "0".

4 На 4-ом цикле при оставшихся с предыдущего цикла сканирования значениях текущего RLO и сигнала меркера фронта FrontMerker0 CPU оставляет для PulseMerker0 значение "0", а для меркера фронта FrontMerker0 значение "1".

5 На 5-ом цикле сканирования состояние входа Input0 изменяется с "1" на "0". CPU обнаруживает это изменение и изменяет состояние меркера фронта FrontMerker0 с "1" на "0". При этом не изменяется значение PulseMerker0 (PulseMerker0 остается равным "0"). Таким образом, восстановлено исходное состояние рассматриваемых битов.

### 5.4.2 Отрицательный фронт сигнала

CPU определяет отрицательный (убывающий) фронт сигнал, когда до начала выполнения функции проверки наличия фронта результат логической операции изменяется от уровня "1" к уровню "0". Процесс обработки сигналов показан на рис. 5.3.

На на нижней половине рис. 5.3 показана следующая последовательность циклов:

1 Сначала состояния входа Input1 и меркера фронта FrontMerker1 соответствуют уровню "0". Меркер импульса PulseMerker1 также сброшен, т.е. уровень его сигнала равен "0".

2 На 2-ом цикле сканирования состояние входа Input1 изменяется с "0" на "1". CPU обнаруживает это изменение при сравнении текущего значения RLO с состоянием меркера фронта FrontMerker1. Если Input1 равен "1", а меркер фронта равен "0", то меркер фронта устанавливается (становится равным "1"). PulseMerker1 остается равным "0".

3 На 3-ем цикле сканирования пока нет разницы между значениями уровней Input1 и FrontMerker1, PulseMerker1 сохраняет значение "0", а меркер фронта FrontMerker1 имеет состояние "1".

4 На 4-ом цикле состояние входа Input1 изменяется с "1" на "0". Обнаружив это, CPU изменяет состояние меркера фронта FrontMerker1 с "1" на "0" и устанавливает для PulseMerker1 значение "1".

5 На 5-ом цикле сканирования не изменяется состояние Input1 и FrontMerker1. CPU устанавливает значение "0" для PulseMerker1. Таким образом, восстановлено исходное состояние битов.

### 5.4.3 Проверка меркера импульса

Наблюдение за состоянием меркеров импульсов (pulse memory bits) с помощью средств тестирования (тест-функций) программатора PG является трудной задачей, так как эти меркеры остаются в установленном состоянии (состояние сигнала "1") в течение только одного цикла сканирования программы.

Выход также не подходит для такой роли как меркер импульса, так как усилители сигнала выходного модуля или приводы не способны достаточно быстро реагировать на изменения входного сигнала.

С помощью "схемы быстрого перезапуска" ("flying restart circuit"), тем не менее, Вы можете записывать чрезвычайно короткие изменения сигналов меркеров импульсов, используя RS-триггер. Меркер импульса устанавливает RS-триггер, тем самым фиксируя факт прихода фронта сигнала, т.е. RS-триггер запоминает сигнал "Фронт сигнала обнаружен". После проверки этого сигнала Вы можете "сбросить" триггер.

```
O   Pmembit0;
O   Pmembit1;
S   Flipflop2;
```

```
A   Input2;
R   Flipflop2;
```

Таким образом, после проверки "сохраненного фронта" Вы можете вновь "сбросить" триггер.

### 5.4.4 Проверка наличия фронта в двоичной логической операции

Проверка наличия фронта в двоичной логической операции может служить для решения практической задачи, если Вы используете сигнал, полученный в результате проверки ("импульс") для управления таймером, счетчиком или операцией с памятью. Двоичные операции проверки могут располагаться между операцией проверки наличия фронта и запускаемой функцией.

```
O   Input3;
O   Input4;
FP  EMembit2;
A   Input5;
S   Output15;
A   Input6;
FN  EMembit3;
R   Output15;
```

В примере выход *Output15* устанавливается в момент, когда выполняется OR- (ИЛИ-) условие (когда бит в OR- [ИЛИ-] выражении переходит от состояния "0" к состоянию "1") и вход *Input5* установлен (равен "1"). Выход *Output15* сбрасывается в момент, когда приходит отрицательный фронт на вход *Input6.*

Функция проверки наличия фронта сигнала является "первичным опросом" ("first check"), так как результат логической операции RLO, который генерируется функцией, может быть использован для последующей обработки. Это также означает, что логическая операция до момента начала проверки наличия фронта считается "завершенной" ("completed") (выполненное OR- [ИЛИ-] условие не сохранено). Функция проверки наличия фронта сигнала не влияет на обработку вложенных инструкций.

### 5.4.5 Двоичный делитель (Binary Scaler)

Двоичный делитель (Binary Scaler) имеет один вход и один выход. Если сигнал на входе двоичного делителя меняет свое состояние, например, с состояния "0" на состояние "1", то выходной сигнал также будет менять свое состояние в соответствии с рис. 5.4.

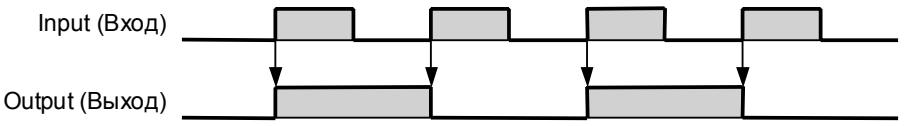



Рис. 5.4 Схема, поясняющая принцип работы двоичного делителя

В нашем примере новое состояние выходной сигнал сохраняет до появления нового положительного фронта сигнала на входе делителя. Только при этом условии выходной сигнал делителя изменяет свое состояние. Это означает, что в этом случае частота выходного сигнала составляет половину частоты переменного входного сигнала.

Существуют различные методы решения такой задачи, два из которых будут рассмотрены ниже.

В первом варианте используется меркер импульса, который устанавливает выход (задает уровень "1"), если он был сброшен (имел состояние "0"), и сбрасывает его (задает уровень "0"), если выход был установлен (имел состояние "1").

При программировании такого решения необходимо помнить, что важно сразу после установки выхода сбросить меркер импульса (иначе выход сразу же будет сброшен вновь).

В нижеследующей программе для первого варианта приняты обозначения:
Input - вход, Output - выход,
EMembit - меркер фронта, PMembit - меркер импульса.

```
      A    Input_1;
      FP   EMembit_1;
      =    PMembit_1;
      A    PMembit_1;
      AN   Output_1;
      S    Output_1;
      R    PMembit_1;
      A    PMembit_1;
      A    Output_1;
      R    Output_1;
```

Второе решение использует условный переход JCN для проверки наличия фронта. Если CPU не обнаруживает фронта сигнала, RLO имеет значение"0" и сканирование программы продолжается с метки перехода.

В случае обнаружения положительного фронта сигнала CPU не выполняет переход к метке, а выполняет следующие два выражения. Здесь, если выход был сброшен (имел состояние "0"), то он устанавливается (получает уровень "1"), если выход был установлен (имел состояние "1"), то он сбрасывается (получает уровень "0). Хотя оператор присвоения управляет выходом, эти последние операторы как бы обладают "свойством запоминания", так как выполняются только при обнаружении положительного фронта сигнала.

```
      A      Input_2;
      FP     EMembit_2;
      JCN    M1
      AN     Output_2;
      =      Output_2;
M1:   ... ;
```

## 5.5 Пример системы управления ленточным конвейером

Очень простая система управления ленточным конвейером используется в этом разделе в качестве примера для демонстрации двоичных логических операций и операций с памятью для входов, выходов и меркеров.

**Описание функций**

Детали должны переноситься лентой конвейера; один поддон на ленте.

Особые характеристики представлены ниже:

- Если на ленте конвейера нет деталей, контроллер сообщает об этом с помощью сигнала "readyload" ("готов к загрузке").
- Сигнал "Start" ("запуск") включает движение ленты конвейера для транспортировки деталей.
- Если на оконечном участке конвейера датчик "end-of-belt" ("конец конвейера"), например, фотоэлемент, обнаруживает присутствие деталей, то контроллер сообщает об этом сигналом "ready_rem" ("готов к выгрузке") и останавливает мотор транспортера конвейера.
- По сигналу "Continue" ("продолжить") лента конвейера с деталями продолжает движение, пока датчик "end-of-belt" ("конец конвейера") не обнаружит присутствие деталей.

Функциональная блок-схема системы управления ленточным конвейером показана на рис. 5.5. В примере запрограммированы входы, выходы и меркеры. Он может быть загружен в любое место любого блока. В примере функция без функционального значения была выбрана в качестве блока.

В главе 19 "Параметры блока" такой же пример запрограммирован в функциональном блоке с параметрами; функциональный блок может вызываться многократно (в том числе для нескольких ленточных конвейеров).

**Сигналы и символы**

Система управления ленточным конвейером использует множество дополнительных сигналов:

- Basic_st
  устанавливает контроллер в исходное состояние.
- Man_on
  активирует движение ленты транспортера, несмотря ни на какие условия.
- /Stop
  останавливает движение ленты транспортера, как только появляется сигнал "0" (датчик в виде нормальнозамкнутого контакта, "zero active" ["активный ноль"]).
- Light_barrier1
  обеспечивает сигнал о достижении деталью конца транспортера.
- /Mfault
  аварийный сигнал от двигателя привода транспортера (например, предохранительный выключатель двигателя); конструкция датчика "zero active" ("активный ноль") обеспечивает аварийный сигнал также и при других видах сбоев, таких, например, как обрыв провода (датчика).

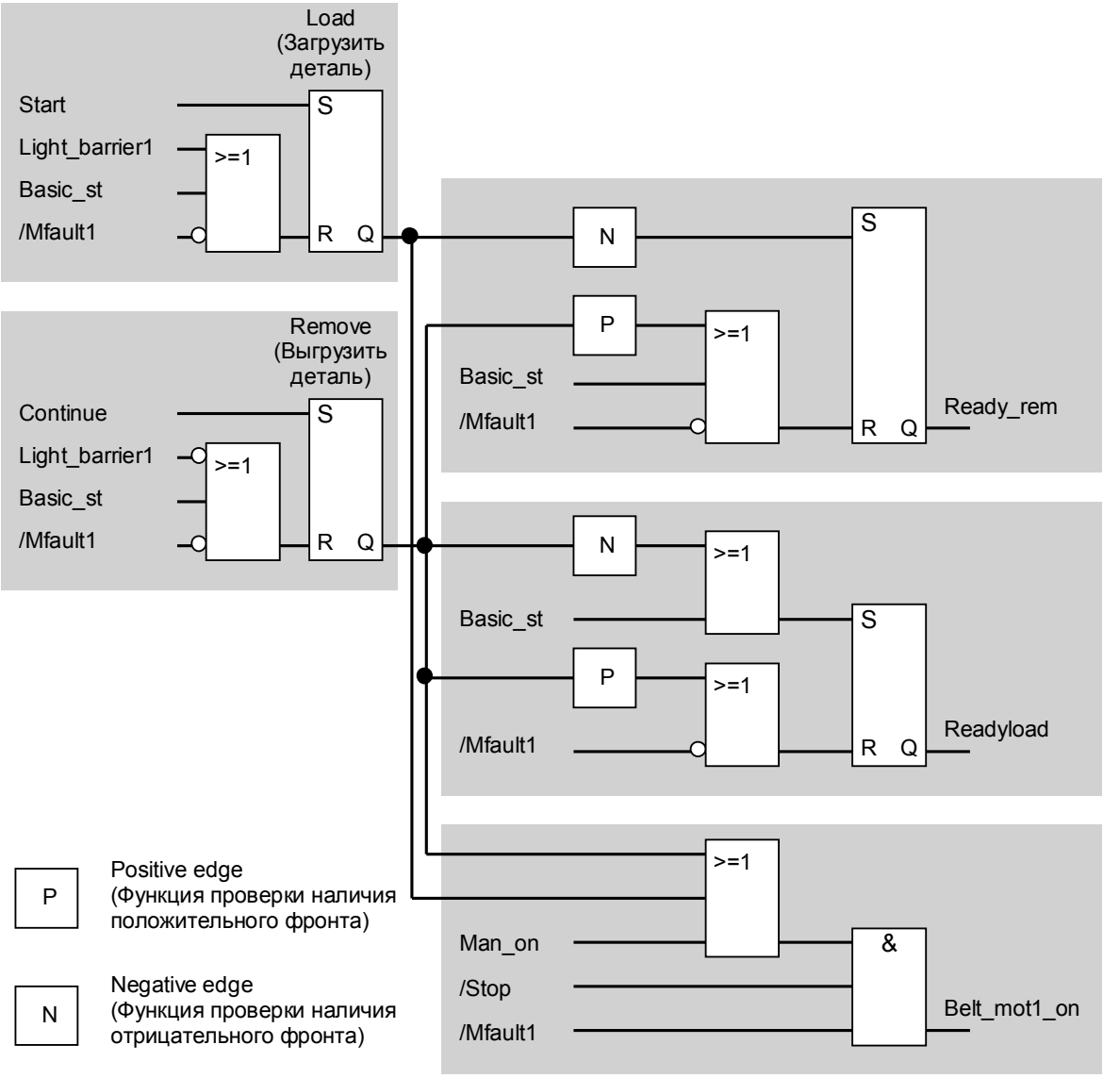



Рис. 5.5 Пример системы управления ленточным конвейером

Если необходимо использовать символьную адресацию, т.е. адресацию посредством символьных имен (символов), то необходимо сгенерировать таблицу символов Symbol Table (см. табл. 5.1) до момента компиляции или инкрементного ввода программы. Эта таблица должна содержать используемые в программе входы, выходы, меркеры и блоки.

Таблица 5.1 Таблица символов Symbol Table для примера системы управления ленточным конвейером

| Symbol (Символ) | Address (Адрес) | Data Type (Тип) | Comment (Комментарий) |
|---|---|---|---|
| Belt_control | FC11 | FC11 | Система управления конвейером |
| Basic_st | I 0.0 | BOOL | Установка контроллеров в исходное состояние |
| Man_on | I 0.1 | BOOL | Включение двигателя привода конвейера |
| /Stop | I 0.2 | BOOL | Остановка двигателя привода конвейера ("zero-active" ["активный ноль"]) |
| Start | I 0.3 | BOOL | Запуск конвейера |
| Continue | I 0.4 | BOOL | Подтверждение того, что деталь была снята с конвейера |
| Light_barrier1 | I 1.0 | BOOL | Фотоэлемент, датчик "End of belt" ("Конец конвейера") для конвейера №1 |
| /Mfault1 | I 2.0 | BOOL | Предохранительный выключатель двигателя привода ("zero-active" ["активный ноль"]) для конвейера №1 |
| Readyload | Q 4.0 | BOOL | Загрузка новых деталей на транспортер ("Готов к загрузке") |
| Ready_rem | Q 4.1 | BOOL | Выгрузка деталей с транспортера ("Готов к выгрузке") |
| Belt_mot1_on | Q 5.0 | BOOL | Управляющий сигнал на включение двигателя конвейера №1 |
| Load | M 2.0 | BOOL | Команда загрузки новых деталей на транспортер |
| Remove | M 2.1 | BOOL | Команда выгрузки деталей с транспортера |
| EM_Rem_N | M 2.2 | BOOL | Меркер фронта (отрицательного) "remove" ("выгрузка детали") |
| EM_Rem_P | M 2.3 | BOOL | Меркер фронта (положительного) "remove" ("выгрузка детали") |
| EM_Loa_N | M 2.4 | BOOL | Меркер фронта (отрицательного) "load" ("загрузка детали") |
| EM_Loa_P | M 2.5 | BOOL | Меркер фронта (положительного) "load" ("загрузка детали") |

**Программа**

Пример программы для системы управления ленточным конвейером расположен в функции без параметров. Эту функцию можно вызывать, например, в организационном блоке OB1 следующим образом:

CALL Belt_control;

Ниже на рис. 5.6 представлен исходный текст программы с символьной адресацией для нашего примера системы управления конвейером.

```
FUNCTION Belt_control :  VOID
TITLE = Control of a conveyor belt
//Пример двоичных логических операций и операций с памятью (без параметров)
NAME    :    Belt1
AUTHOR  :    Berger
FAMILY  :    STL_Book
VERSION :    01.00
BEGIN
NETWORK
TITLE = Load parts
//В этом сегменте выполняется команда "Load", которая начинает перенос
//деталей транспортером
     A    Start;             //Запуск конвейера
     S    Load;
     O    Light_barrier1;    //Детали достигают конца ленты
     O    Basic_st;
     ON   "/Mfault1";        //Предохранительный выключатель мотора
     R    Load;
NETWORK
TITLE = Parts ready for removal
//Когда детали достигли конца конвейера, они могут быть сняты
     A    Load;              //При достижении конца конвейера
     FN   EM_Loa_N;          //"Load" сбрасывается
     S    Ready_rem;         //Детали могут быть сняты
     A    Remove;
     FP   EM_Rem_P;          // Детали сняты
     O    Basic_st;
     ON   "/Mfault1";
     R    Ready_rem;
NETWORK
TITLE = Remove parts
//Команда "Remove" инициирует снятие деталей с транспортера
     A    Continue;          //Включатель реверса конвейера
     S    Remove;
     ON   Light_barrier1;    //Детали сняты с ленты
     O    Basic_st;
     ON   "/Mfault1";        //Предохранительный выключатель мотора
     R    Remove;
NETWORK
TITLE = Belt ready for loading
//Когда детали сняты с конвейера, можно поставить на ленту новые
     A    Remove;
     FN   EM_Rem_N;          //Детали сняты
     O    Basic_st;
     S    Readyload;         //Лента конвейера пуста
     A    Load;
     FP   EM_Loa_P;          //Транспортер запущен
     ON   "/Mfault1";
     R    Readyload;
NETWORK
TITLE = Control belt motor
//Двигатель привода включается и выключается в данном сегменте
     A(;
     O    Load;              //Загрузка деталей на транспортер
     O    Remove;            //Удаление деталей с транспортера
     O    Man_on;            //Запуск с помощью "Man_on" (без реманентности)
     );
```

(см. продолжение программы на следующей странице)

```
     A    "/Stop";          //Остановка двигателя транспортера
     ON   "/Mfault1";       //Предохранительный выключатель мотора
     =    Belt_motor1;
NETWORK
TITLE = Block end
     BE
END_FUNCTION
```

Рис. 5.6 Пример программы для системы управления ленточным конвейером

Глобальные символы могут также использоваться без кавычек (без апострофа), если они не содержат специальных символов. Если же символ (символьное имя) содержит специальный символ (например, пробел [space]), то такое имя должно быть заключено в кавычки. В компилированных блоках редактор STL отображает все глобальные символы в кавычках.

Представленная программа разделена на сегменты с целью большей ясности и лучшей читаемости. Последний сегмент, имеющий заголовок BLOCK END (конец блока), не является необходимым, а служит лишь для обозначения окончания блока. Такой прием бывает очень полезно использовать в случаях, когда блоки имеют чрезмерно большой размер.

# 6 Функции пересылки данных (move functions)

В данной главе рассматриваются функции для языка программирования STL, с помощью которых выполняются операции обмена данными с помощью аккумуляторов (регистров). К ним относятся следующие функции:

- Функции Load (функции загрузки данных в аккумулятор)
  Функции загрузки используются для загрузки данных в аккумуляторы для последующей обработки (выполнение функций обработки чисел - "digital functions" - операций сравнения, арифметических операций и т.д.)
- Функции Transfer (функции выгрузки данных из аккумулятора) Функции выгрузки используются для выгрузки численных результатов из аккумулятора accumulator1 в память CPU, например, в область меркеров.
- Accumulator functions (функции аккумуляторов) Функции аккумуляторов позволяют передавать информацию из одного аккумулятора в другой или перемещать информацию внутри аккумулятора accumulator1.

Функции Load (функции загрузки данных в аккумулятор) Вам также могут потребоваться для задания начальных значений для таймеров и счетчиков или для обработки текущих значений таймеров и счетчиков.

Системные функции SFC 20 BLKMOV, SFC 81 UBLKMOV и SFC 21 FILL используются для копирования больших объемов информации в памяти или в заданные области данных.

Функции Load (функции загрузки данных в аккумулятор) и функции Transfer (функции выгрузки данных из аккумулятора) также могут потребоваться для адресации модулей с использованием области данных пользователя; если для адресации модулей Вы используете область системных данных, Вы должны использовать системные функции для передачи записей данных.

Эти системные функции также можно использовать для параметризации модулей.

Примеры для этой главы Вы найдете на прилагаемой дискете в библиотеке STL_Book в разделе "Basic Functions" в функциональном блоке FB 106 или в исходном файле Chap_6.

## 6.1 Общие замечания по поводу операций загрузки и выгрузки данных

Функции Load (функции загрузки данных в аккумулятор) и функции Transfer (функции выгрузки данных из аккумулятора) позволяют производить обмен между различными областями памяти. Такой обмен данными не может производиться непосредственно, а только с использованием аккумулятора

accumulator 1. Аккумулятор - это специальный регистр в процессоре, который выполняет функции промежуточного буфера.

Во время обмена информацией направление, в котором происходит передача данных, указывается в используемой для передачи инструкции. Данные, направляемые из памяти в аккумулятор accumulator 1, называются *загружаемыми* (*loading*), тогда как данные, пересылаемые в обратном направлении, называются *выгружаемыми* (*transferring*) (содержимое аккумулятора "выгружается" ["transferred"] в область памяти).

Операции загрузки данных в аккумулятор и выгрузки данных из аккумулятора являются предопределенными операциями для выполнения *функций обработки чисел (digital functions)*, с помощью которых осуществляется управление численными значениями (digital value) (например, операция сдвига или преобразования) или комбинирование двух чисел (например, операция сложения или сравнения). Для одновременной обработки двух численных значений требуются два промежуточных буфера. В роли таких буферов выступают аккумулятор accumulator 1 и аккумулятор accumulator 2. Все CPU имеют такие специальные регистры. Кроме того, S7-400 CPU имеют два дополнительных промежуточных буфера - аккумулятор accumulator 3 и аккумулятор accumulator 4, которые используются преимущественно в арифметических операциях.

Несколько функций, называемых функциями аккумуляторов (accumulator functions), используются для копирования содержимого одного аккумулятора в другой.

На рис. 6.1 графически показаны соотношения между функциями пересылки данных и области их применения.

Функции загрузки (load) пересылают информацию из системной памяти (system memory), рабочей памяти (work memory) и периферии (I/O) в аккумулятор accumulator 1, смещая при этом "старое" (точнее сказать, "текущее") значение аккумулятора accumulator 1 в аккумулятор accumulator 2.

Функции обработки чисел (digital functions) позволяют управлять содержимым аккумулятора accumulator 1 или комбинировать численные значения, содержащиеся в аккумуляторах accumulator 1 и accumulator 2, с последующей записью результата в аккумулятор accumulator 1.

Функции аккумуляторов (accumulator functions) позволяют получить доступ к содержимому всех аккумуляторов. Источником для пересылки (transfer) информации в системную память (system memory), рабочую память (work memory) и в периферию (I/O) может служить лишь только аккумулятор accumulator 1.

Каждый аккумулятор содержит 32 разряда, тогда как все области памяти имеют байтовую структуру (byte-oriented). Обмен информацией между областями памяти и аккумулятором accumulator 1 может происходить побайтно, по 1 машинному слову и по 1 двойному машинному слову.

В данной главе функции Load (функции загрузки данных в аккумулятор) и функции Transfer (функции выгрузки данных из аккумулятора) обсуждаются в применении к адресным областям входов, выходов, меркеров, периферии (I/O) и для загрузки констант.

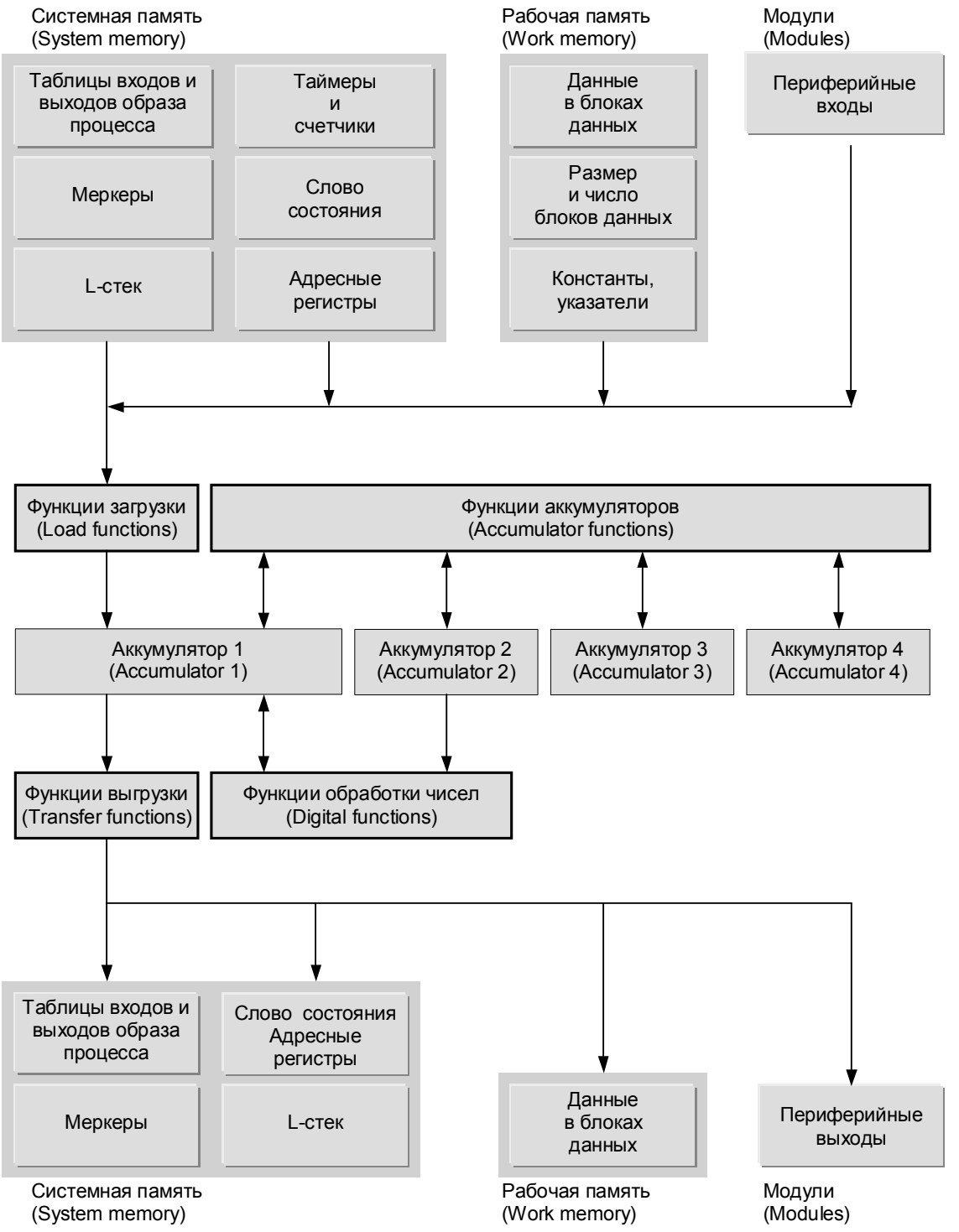



Рис. 6.1 Области памяти для функций загрузки и выгрузки данных

Функции Load (функции загрузки данных в аккумулятор) и функции Transfer (функции выгрузки данных из аккумулятора) могут быть также применены со следующими адресными областями:

- области таймеров и счетчиков
  (глава 7 "Функции таймеров", глава 8 "Функции счетчиков")
- слово состояния
  (глава 15 "Биты состояния")
- области временных локальных данных
  (L-стек, раздел 18.1.5 "Временные локальные данные")
- области адресов данных, длина и число блоков данных
  (глава 18.2 "Функции для блоков данных")
- адресные регистры и указатели
  (глава 25 "Косвенная адресация")
- адреса переменных
  (глава 26 "Прямой доступ к переменным")

## 6.2 Функции Load (функции загрузки данных в аккумулятор)

### 6.2.1 Общее представление о функциях загрузки Load

Функция загрузки состоит из оператора L (код операции загрузки) и константы, переменной или адреса с идентификатором адреса, содержимое которого функция будет загружать в аккумулятор accumulator 1.

```
L   +1200       Константа
                 (прямая адресация)
L   IW  16       Местоположение числа
                 (прямая адресация)
L   ActValue     Переменная
                 (символьная адресация)
```

CPU выполняет функцию загрузки независимо от результата логической операции RLO и битов состояния. Функция загрузки не влияет на результат логической операции и не влияет на биты состояния.

**Влияние на аккумулятор accumulator 2**

Функция Load (функция загрузки данных в аккумулятор) влияет на содержимое аккумулятора accumulator 2. В то время как значение адреса, константы или переменной, определенное в операторе загрузки загружается в аккумулятор accumulator 1, текущее содержимое аккумулятора accumulator 1 пересылается в аккумулятор accumulator 2. Функция загрузки Load полностью пересылает содержимое аккумулятора accumulator 1 в аккумулятор accumulator 2. Предыдущее содержимое аккумулятора accumulator 2 при этом теряется.

При использовании S7-400 CPU функция загрузки данных не влияет на содержимое аккумуляторов accumulator 3 и accumulator 4.

**Общая информация об операции загрузки**

Адрес числа, определенный в функции загрузки Load, может иметь размер байта (byte), слова (word) или двойного слова (duoble word) (см. рис. 6.2).

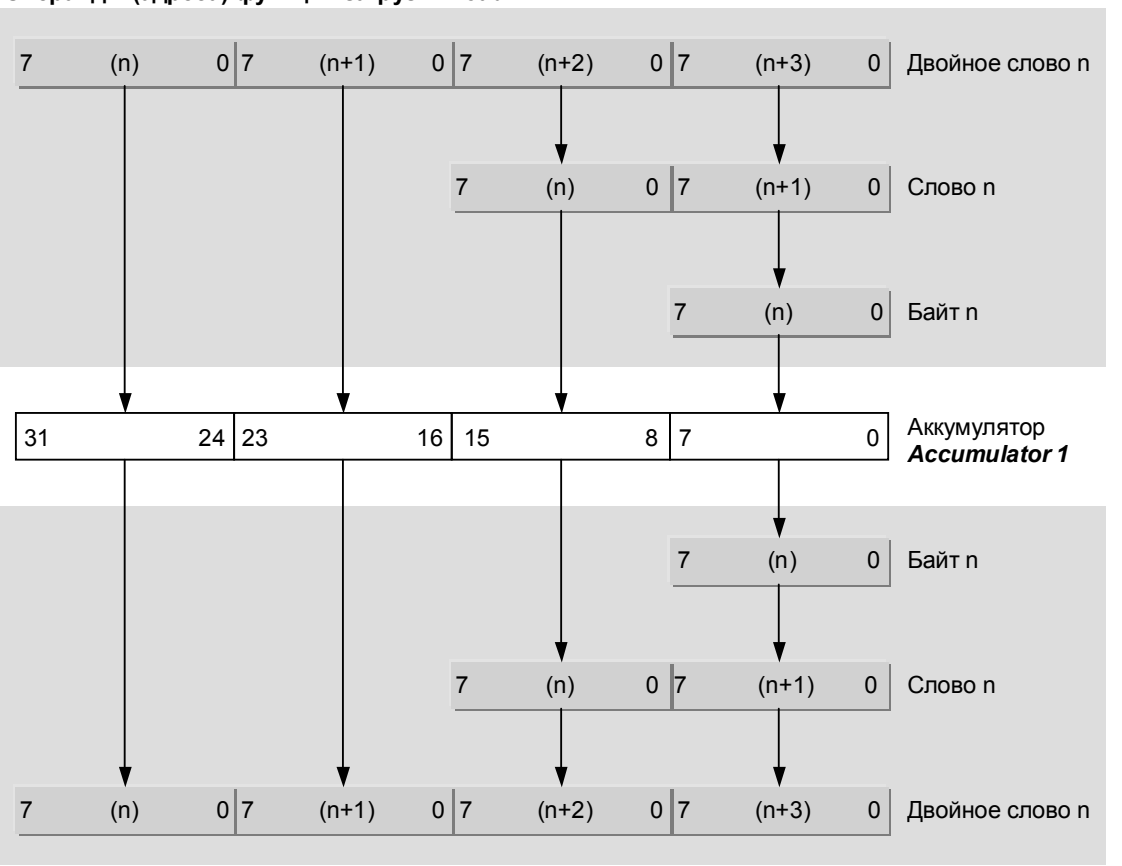



Рис. 6.2 Загрузка и выгрузка байтов, слов, двойных слов

*Загрузка байта*

При загрузке байта его содержимое считывается в аккумулятор accumulator 1 с выравниванием вправо. Неиспользуемые байты заполняются нулями.

*Загрузка слова*

При загрузке слова его содержимое считывается в аккумулятор accumulator 1 с выравниванием вправо. При этом старший байт слова (n+1) располагается в аккумуляторе с выравниванием вправо, тогда как младший байт слова (n) располагается в аккумуляторе вплотную слева от байта (n+1). Неиспользуемые байты заполняются нулями.

*Загрузка двойного слова*

При загрузке двойного слова его содержимое считывается в аккумулятор accumulator 1 с выравниванием вправо. При этом младший байт двойного

слова (n) занимает крайний левый байт аккумулятора, а самый старший байт двойного слова (байт n+3) занимает крайний правый байт аккумулятора.

## 6.2.2 Загрузка в аккумулятор из памяти

**Загрузка данных из области входов**

L IB n Загрузка данных из входного байта

L IW n Загрузка данных из входного слова

L ID n Загрузка данных из входного двойного слова

В некоторых типах CPU загрузка данных от входов разрешена, только если к соответствующим входным модулям имеется доступ (см. разд. 1.5.2 "Отображение процесса").

**Загрузка данных из области выходов**

L QB n Загрузка данных из выходного байта

L QW n Загрузка данных из выходного слова

L QD n Загрузка данных из выходного двойного слова

В некоторых типах CPU загрузка данных от выходов разрешена, только если к соответствующим выходным модулям имеется доступ (см. разд. 1.5.2 "Отображение процесса").

**Загрузка данных из периферии (I/O)**

L PIB n Загрузка данных из периферийного входного байта

L PIW n Загрузка данных из периферийного входного слова

L PID n Загрузка данных из периферийного входного двойного слова

При загрузке данных от области I/O входные модули адресуются как периферийные входы (PI). При этом доступны только существующие входные модули.

Надо отметить, что при непосредственной загрузке данных от I/O модуля пересылаемое значение может отличаться от значения, загружаемого из области входов образа процесса с тем же адресом, так как эти значения одинаковы лишь в начале цикла сканирования программы (когда CPU обновляет отображение процесса). Непосредственная же загрузка данных от I/O модулей позволяет записать в аккумулятор "текущие" значения сигналов входов.

**Загрузка данных из меркеров**

L MB n Загрузка данных из байта меркеров

L MW n Загрузка данных из слова меркеров

L MD n Загрузка данных из двойного слова меркеров

Загрузка данных из области меркеров всегда разрешена, так как такая область целиком расположена в CPU. Надо отметить, однако, что разные типы CPU могут иметь разные по размерам области меркеров.

### 6.2.3 Загрузка констант в аккумулятор

**Загрузка констант простых типов**

Вы можете загружать константу или фиксированное значение непосредственно в аккумулятор. При этом для улучшения читаемости программы Вы можете выбирать для констант подходящее представление, использовать различные форматы. В главе 3 "SIMATIC S7-программа" Вы можете найти обзор разрешенных форматов для констант. Все константы, которые могут быть загружены в аккумулятор являются константами простых типов.

| | | |
|---|---|---|
| L | B#16#F1 | Загрузка двухразрядного шестнадцатеричного числа |
| L | -1000 | Загрузка целого числа (INT) |
| L | 5.0 | Загрузка действительного числа (REAL) |
| L | S5T#2s | Загрузка данных таймера формата S5 |
| L | C#250 | Загрузка числа формата BCD (значение счетчика) |
| L | TOD#8:30:00 | Загрузка времени суток |

В главе 24 "Типы данных" описывается назначение битов констант (показана битовая структура данных).

**Загрузка указателей**

Указатели - это специальная форма констант, которая используется для вычисления позиции в памяти. Вы можете загружать следующие указатели в аккумулятор:

| | | |
|---|---|---|
| L | P#1.0 | Загрузка внутризонного счетчика |
| L | P#M2.1 | Загрузка межзонного счетчика |
| L | P#*name* | Загрузка адреса локальной переменной |

Вы не сможете загрузить указатель DB (DB pointer) или указатель ANY (ANY pointer) в аккумулятор, так как длина этих указателей превышает 32 бита.

В главе 25 "Косвенная адресация" и в главе 26 "Прямой доступ к переменным" Вы найдете дополнительную информацию по данной теме.

## 6.3 Функции Transfer (функции выгрузки данных из аккумулятора)

### 6.3.1 Общее представление о функциях выгрузки Transfer

Функция выгрузки состоит из оператора T (код операции выгрузки) и адреса в области памяти, по которому должны быть отправлены данные из аккумулятора accumulator 1.

| | | |
|---|---|---|
| T | MW120 | Переносит содержимое аккумулятора по определенному адресу в память (абсолютная адресация) |
| T | Setpoint | Переносит содержимое аккумулятора в переменную (символьная адресация) |

CPU выполняет функцию выгрузки независимо от результата логической операции RLO и битов состояния. Функция выгрузки не влияет на результат логической операции и не влияет на биты состояния.

Функция выгрузки пересылает содержимое аккумулятора accumulator 1 по одному байту, или по одному слову, или по одному двойному слову в заданный адрес. При этом содержимое аккумулятора accumulator 1 остается неизменным, что делает возможным многократную пересылку данных из аккумулятора accumulator 1.

Функция выгрузки возможна только с помощью аккумулятора accumulator 1. Если есть необходимость передать данные из другого аккумулятора, то Вы должны сначала передать эти данные в аккумулятор accumulator 1, используя функции аккумуляторов, и лишь затем переслать их с помощью функции выгрузки Transfer по требуемому адресу в память.

**Общая информация об операции выгрузки**

Адрес числа, определенный в функции выгрузки Transfer, может иметь размер байта (byte), слова (word) или двойного слова (duoble word) (см. рис. 6.2).

*Выгрузка байта*

При выгрузке байта содержимое крайнего правого байта аккумулятора accumulator 1 пересылается в байт по указанному в выражении адресу.

*Выгрузка слова*

При выгрузке слова содержимое крайнего правого слова аккумулятора accumulator 1 пересылается в слово по указанному в выражении адресу. При этом содержимое старшего байта слова (n+1) в аккумуляторе переносится в слово назначения, где заполняет старший байт (n+1), тогда как младший байт слова (n) в аккумуляторе переносится в слово назначения, где заполняет младший байт (n).

*Выгрузка двойного слова*

При выгрузке двойного слова его содержимое считывается из аккумулятора accumulator 1 пересылается в двойное слово по указанному в выражении адресу. При этом содержимое младшего байта двойного слова (n) из аккумулятора занимает младший байт (n) двойного слова назначения, а самый старший байт двойного слова (n+3) из аккумулятора занимает самый старший байт двойного слова назначения (n+3).

## 6.3.2 Выгрузка данных из аккумулятора в различные области памяти

**Выгрузка данных во входы**

| | | |
|---|---|---|
| T | IB n | Выгрузка данных во входной байт |
| T | IW n | Выгрузка данных во входное слово |
| T | ID n | Выгрузка данных во входное двойное слово |

В некоторых типах CPU выгрузка данных в входы разрешена, только если к соответствующим входным модулям имеется доступ (см. разд. 1.5.2 "Отображение процесса").

Пересылка во входы влияет только на биты в области отображения процесса, также как и операции установки и сброса этих битов (входов "inputs"). Возможное применение для таких операций заключается в инициализации значений входов в целях отладки или для запуска с определенными начальными значениями: если Вы запускаете программу на выполнение с заданными значениями входов, то программа начинает выполняться именно с заданными Вами новыми значениями, а не со значениями, полученными от входных модулей.

**Выгрузка данных в выходы**

T QB n Выгрузка данных в выходной байт

T QW n Выгрузка данных в выходное слово

T QD n Выгрузка данных в выходное двойное слово

В некоторых типах CPU выгрузка данных в выходы разрешена, только если к соответствующим выходным модулям имеется доступ (см. разд. 1.5.2 "Отображение процесса").

**Выгрузка данных в периферийные выходы**

T PQB n Выгрузка данных в периферийный выходной байт

T PQW n Выгрузка данных в периферийное выходное слово

T PQD n Выгрузка данных в периферийное выходное двойное слово

При выгрузке данных в область I/O выходные модули адресуются как периферийные выходы (PQ). При этом доступны только существующие выходные модули.

Надо отметить, что при непосредственной выгрузке данных в область I/O выходных модулей, связанных с таблицей выходов образа процесса, данные в этой таблице обновляются, ибо нет никакой разницы (в рассматриваемом контексте) между выходом (из области отображения процесса) и периферийным выходом с одинаковым адресом.

**Выгрузка данных в область меркеров**

T MB n Выгрузка данных в байт меркеров

T MW n Выгрузка данных в слово меркеров

T MD n Выгрузка данных в двойное слово меркеров

Выгрузка данных в область меркеров всегда разрешена, так как такая область целиком расположена в CPU. Надо отметить, однако, что разные типы CPU могут иметь разные по размерам области меркеров.

## 6.4 Функции аккумуляторов (Accumulator Functions)

Функции аккумуляторов (Accumulator Functions) позволяют пересылать значения из одного аккумулятора в другой или перемещать байты внутри аккумулятора accumulator 1. На выполнение функций аккумуляторов не влияют ни результат логической операции RLO ни биты состояния. Также и результат выполнения этих функций не оказывает влияния ни на RLO, ни на биты состояния.

## 6.4.1 Прямая пересылка данных между аккумуляторами

| | |
|---|---|
| PUSH | Сдвиг содержимого аккумулятора "вперед" |
| POP | Сдвиг содержимого аккумулятора "назад" |
| TAK | Обмен содержимым между аккумуляторами accumulator 1 и accumulator 2 |
| ENT | Сдвиг содержимого аккумулятора "вперед" (без аккумулятора accumulator 1) |
| LEAVE | Сдвиг содержимого аккумулятора "назад" (без аккумулятора accumulator 1) |

Первые три функции PUSH, POP и TAK используются в CPU, имеющих только два аккумулятора (S7-300 CPU). Все пять функций используются в CPU, имеющих четыре аккумулятора (S7-400 CPU). Схематически выполнение перечисленных выше функций показано на рис. 6.3.

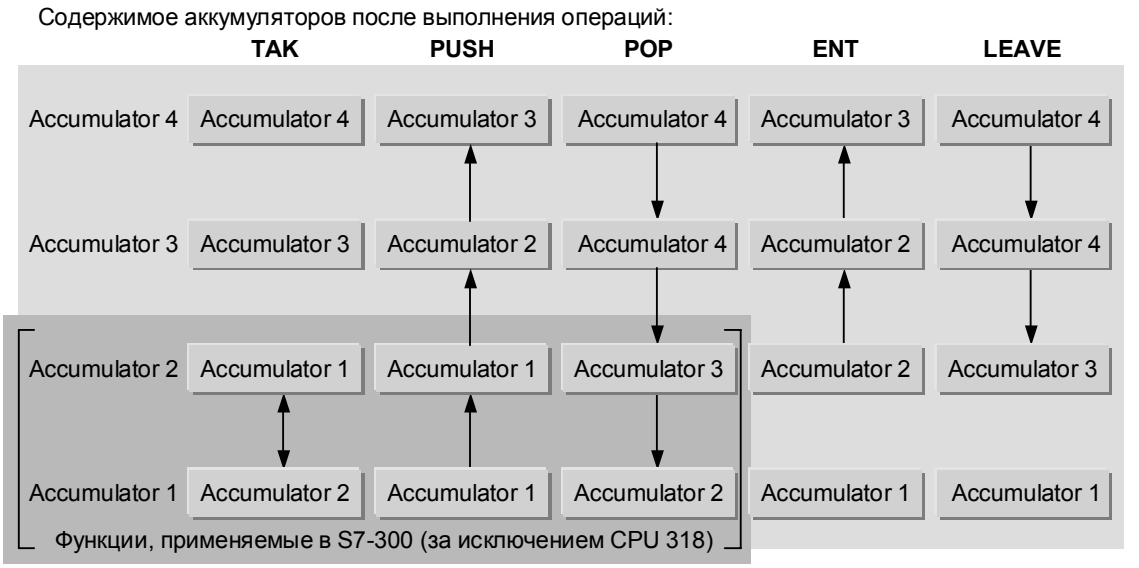



Рис. 6.3 Прямая пересылка данных между аккумуляторами в CPU S7-300 и S7-400

**PUSH**

Функция PUSH вызывает сдвиг содержимого аккумуляторов вдоль цепочки accumulator 1, accumulator 2, accumulator 3 и accumulator 4. При этом происходит перемещение данных из аккумулятора accumulator 3 в accumulator 4, данных из аккумулятора accumulator 2 в accumulator 3 и копирование данных из аккумулятора accumulator 1 в accumulator 2 (содержимое аккумулятора accumulator 1 остается неизменным).

Таким образом, Вы можете использовать функцию PUSH для внесения одного и того же значения в несколько аккумуляторов.

**POP**

Функция POP вызывает сдвиг содержимого вдоль цепочки аккумуляторов accumulator 4, accumulator 3, accumulator 2 и accumulator 1. При этом происходит перемещение данных из аккумулятора accumulator 4 в accumulator 3, данных из аккумулятора accumulator 3 в accumulator 2 и данных из аккумулятора accumulator 2 в accumulator 1 (содержимое аккумулятора accumulator 4 при этом остается неизменным).

Таким образом, Вы можете использовать функцию POP для переноса значений из аккумуляторов accumulator 4, accumulator 3 и accumulator 2 в accumulator 1, откуда эти значения могут быть пересланы в память в требуемые адреса.

**TAK**

Функция TAK вызывает обмен содержимым между аккумуляторами accumulator 1 и accumulator 2. При этом содержимое аккумуляторов accumulator 3 и accumulator 4 остается неизменным.

**ENT**

Функция ENT вызывает сдвиг содержимого аккумуляторов вдоль цепочки accumulator 2, accumulator 3 и accumulator 4. При этом происходит перемещение данных из аккумулятора accumulator 3 в accumulator 4 и копирование данных из аккумулятора accumulator 2 в accumulator 3. При этом содержимое аккумуляторов accumulator 1 и accumulator 2 остается неизменным.

Если сразу за функцией ENT следует функция загрузки Load, то последняя вызывает вызывает сдвиг содержимого аккумуляторов вдоль цепочки accumulator 1, accumulator 2, accumulator 3 и accumulator 4 (аналогично функции PUSH); при этом новое значение будет располагаться в аккумуляторе accumulator 1.

**LEAVE**

Функция LEAVE вызывает сдвиг содержимого аккумуляторов вдоль цепочки accumulator 4, accumulator 3 и accumulator 2. При этом происходит перемещение данных из аккумулятора accumulator 3 в accumulator 2 и копирование данных из аккумулятора accumulator 4 в accumulator 3. При этом содержимое аккумуляторов accumulator 4 и accumulator 1 остается неизменным.

Арифметические функции используют функцию LEAVE. Используя данную функцию Вы можете также смоделировать некоторые функциональные возможности других логических операций (например, в логических операциях с операндами формата слова [Word]).

Будучи записанной после логической операции с числами (digital logic operation), функция LEAVE помещает содержимое аккумуляторов accumulator 4 и accumulator 3 соответственно в аккумуляторы accumulator 3 и accumulator 2. При этом результат логической операции с числами (digital logic operation) остается неизменным в аккумуляторе accumulator 1.

## 6.5 Функции обмена байтами в аккумуляторе accumulator 1

| | |
|---|---|
| CAW | Обмен местами между байтами младшего слова в аккумуляторе accumulator 1 |
| CAD | Обмен местами между всеми байтами в аккумуляторе accumulator 1 |

Функция CAW меняет местами два байта в младшем слове в аккумуляторе accumulator 1. При этом байты старшего слова аккумулятора остаются неизменными.

Функция CAD меняет местами все байты в аккумуляторе accumulator 1. При этом самый старший байт становится самым младшим по номеру, а средние два байта меняются местами.

Схематически результат выполнения перечисленных выше функций показано на рис. 6.4.

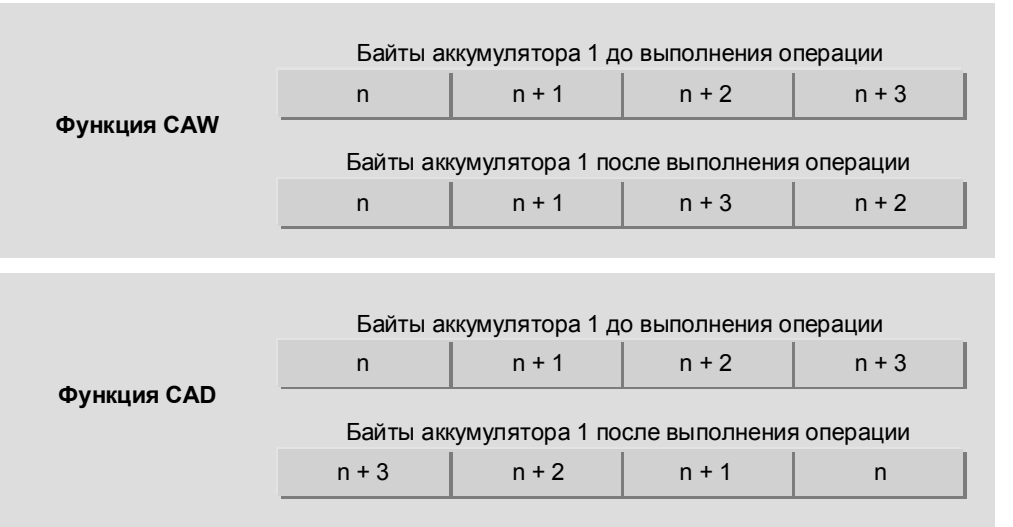



Рис. 6.4. Обмен байтов местами в аккумулятре accumulator 1

## 6.6 Системные функции для пересылки данных

- SFC 20 BLKMOV
  Системная функция копирования области данных
- SFC 21 FILL
  Системная функция вставки данных в область назначения
- SFC 81 UBLKMOV
  Системная функция непрерывного копирования области данных

Каждая из этих системных функций имеет два параметра типа ANY (см. табл. 6.1). Теоретически каждый из этих параметров может определять произвольный адрес, переменную или абсолютный адрес в памяти.

Таблица 6.1 Параметры для системных функций SFC 20, 21 и 81

| SFC | Параметр | Назначение | Тип данных | Содержание, описание |
|---|---|---|---|---|
| SFC 20 | SRCBLK | INPUT | ANY | Область-источник, из которой копируются данные |
| | RET_VAL | OUTPUT | INT | Информация об ошибках |
| | DSTBLK | OUTPUT | ANY | Область-приемник, в которую копируются данные |
| SFC 21 | BVAL | INPUT | ANY | Область-источник, из которой копируются данные |
| | RET_VAL | OUTPUT | INT | Информация об ошибках |
| | BLK | OUTPUT | ANY | Область-приемник, в которую копируются данные (включая повторное копирование) |
| SFC 81 | SRCBLK | INPUT | ANY | Область-источник, из которой копируются данные |
| | RET_VAL | OUTPUT | INT | Информация об ошибках |
| | DSTBLK | OUTPUT | ANY | Область-приемник, в которую копируются данные |

Если Вы определяете переменную сложного типа, она должна быть определена целиком ("complete variable"); поэтому компоненты переменной (например, отдельные компоненты массива или структуры) не допускаются (как параметры). Для абсолютной адресации Вы можете использовать указатель ANY; такие указатели рассматриваются в разделе 25.1 "Указатели".

Вы можете также использовать функции для копирования отдельных переменных типа STRING. Тем не менее, в этом случае редактор STL-программ и редактор SCL-программ ведут себя по-разному (см. разд. 6.6.4 "Копирование переменных типа STRING").

Если Вы используете временные локальные данные в виде фактического параметра блока типа ANY, редактор принимает такой параметр, как параметр со структурой указателя ANY. В таком случае Вы можете создать указатель ANY на временные локальные данные, которые могут быть изменены во время работы программы, что означает, что Вы можете создать "область переменной" ("variable area"). В главе 26 "Прямой доступ к переменным" показано, как использовать такие "указатели ANY переменной".

### 6.6.1 Копирование области данных

Системная функция SFC 20 BLKMOV позволяет выполнить копирование содержания области данных в направлении возрастания (инкрементного) их адресов (параметр SRCBLK) в область назначения (параметр DSTBLK).

При этом могут быть определены следующие фактические параметры:

- любые переменные из адресных областей входов (I), выходов (Q), меркеров (M) и блоков данных (переменные блоков глобальных данных и экземплярных блоков данных),
- переменные из области временных локальных данных (особо для данных типа ANY),
- области данных с абсолютной адресацией посредством указателя ANY.

Используя функцию SFC 20 BLKMOV Вы не можете копировать значения таймеров и счетчиков или передавать информацию как из модулей, так и в модули (область I/O), а также с помощью этой функции нельзя передавть информацию из/в системные блоки данных (SDB).

Что касается входов и выходов (области образа процесса), заданные области могут копироваться вне зависимости от того, определены ли здесь адреса входных или выходных модулей. Вы можете также определить переменную или адрес области в блоке данных в загрузочной (load) памяти (блок данных с ключевым словом UNLINKED [несвязанный]).

При использовании функции SFC 20 BLKMOV область-источник и область-приемник не должны перекрываться. Если область-источник и область-приемник имеют разную длину, то наименьшая по размеру из них будет определять объем данных, переданный в область-приемник.

Пример: Переменная *Frame* (например, структурированная переменная пользовательского типа) в блоке данных "Rec_mailb" должна быть скопирована в переменную *Frame1* (такого же типа как и переменная *Frame*) в блоке данных "Buffer". Значение функции должно быть введено в переменную *Copyerror* в блоке данных "Evaluation".

```
CALL BLKMOV  (
   SRCBLK   := Rec_mailb.Frame,
   RET_VAL  := Evaluation.Copyerror,
   DSTBLK   := Buffer.Frame1);
```

## 6.6.2 Непрерывное копирование из области данных

Системная функция SFC 81 UBLKMOV позволяет копировать содержимое исходной области данных (параметр SRCBLK) в область назначения (параметр DSTBLK) в направлении возрастания (инкрементного) адресов данных. Операция копирования не может прерываться, поэтому в соответствующих условиях время ожидания обработки прерывания может возрастать. Объем копируемых данных может достигать 512 байт.

Для этой функции могут быть определены следующие фактические параметры:

- любые переменные из адресных областей входов (I), выходов (Q), меркеров (M) и блоков данных (переменные блоков глобальных данных и экземплярных блоков данных),

- переменные из области временных локальных данных (особо для данных типа ANY),
- области данных с абсолютной адресацией посредством указателя ANY.

Используя функцию SFC 81 BLKMOV Вы не можете копировать значения таймеров и счетчиков или передавать информацию как из модулей, так и в модули (область I/O). Также с помощью этой функции нельзя передавть информацию из/в системные блоки данных (SDB) и из/в блоки данных в загрузочной (load) памяти (блок данных с ключевым словом UNLINKED [несвязанный]).

Что касается входов и выходов (области образа процесса), заданные области могут копироваться вне зависимости от того, определены ли здесь адреса входных или выходных модулей.

При использовании функции SFC 81 BLKMOV область-источник и область-приемник не должны перекрываться. Если область-источник и область-приемник имеют разную длину, то наименьшая по размеру из них будет определять объем данных, переданный в область-приемник.

Пример: Из блока данных "Buffer" первый компонент массива *Data* должен быть скопирован в переменную *Frame* в блоке данных "Send_mailb". Значение функции должно быть сохранено в переменной *Copyerror* в блоке данных "Evaluation".

```
CALL UBLKMOV  (
   SRCBLK  := Buffer.Data[1],
   RET_VAL := Evaluation.Copyerror,
   DSTBLK  := Send_mailb.Frame1);
```

### 6.6.3 Вставка данных в область назначения

Системная функция SFC 21 FILL позволяет копировать содержимое исходной области данных (параметр BVAL) в область назначения (параметр BLK) в направлении возрастания (инкрементного) адресов данных. Операция копирования продолжается до полного заполнения области назначения (при условии превышения размера области назначения над областью-источником возможно многократное копирование исходных данных).

Для этой функции могут быть определены следующие фактические параметры:

- любые переменные из адресных областей входов (I), выходов (Q), меркеров (M) и блоков данных (переменные блоков глобальных данных и экземплярных блоков данных),
- переменные из области временных локальных данных (особо для данных типа ANY),
- области данных с абсолютной адресацией посредством указателя ANY.

Используя функцию SFC 21 FILL Вы не можете копировать значения таймеров и счетчиков или передавать информацию как из модулей, так и в модули (область I/O). Также с помощью этой функции нельзя передавть информацию из/в системные блоки данных (SDB).

Что касается входов и выходов (области образа процесса), заданные области могут копироваться вне зависимости от того, определены ли здесь адреса входных или выходных модулей.

При использовании функции SFC 21 FILL область-источник и область-приемник не должны перекрываться. Если область-источник больше области-приемника, то объем данных, переданный в область-приемник будет соответствовать размеру последнего; если же область-источник меньше области-приемника, то в область-приемник будут записываться (возможно многократно) копируемые данные до его полного заполнения (даже, если размеры рассматриваемых областей не кратны друг другу).

Пример: Блок данных DB 13 содержит 128 байт. Необходимо скопировать во все эти байты содержимое байта меркеров MB 80.

```
CALL SFC 21  (
  BVAL     := MB 80,
  RET_VAL  := MW 32,
  BLK      := P#DB13.DBX0.0 BYTE 128);
```

### 6.6.4 Копирование переменных типа STRING

Вы можете копировать отдельные переменные типа STRING с использованием системных функций SFC 20 BLKMOV и SFC 81 UBLKMOV. Редакторы STL-программ и SCL-программ ведут себя по-разному при этом.

Редактор STL-программ обрабатывает переменную типа STRING как массив байтов, так что SFC передает отдельные байты один к одному (включая два первые байта со спецификацией размера). Если, например, Вы пересылаете массив байтов в переменную типа STRING, то Вы должны задать правильную длину в первых двух байтах STRING-переменной ("length bytes") в соответствии с передаваемым массивом.

Редактор SCL-программ записывает переменную типа STRING в указатель ANY. При этом SFC передает только соответствующие позиции символов строки STRING-переменной. Если STRING-переменная является областью назначения, то при необходимости фактическая длина ее корректируется. Таким образом Вы можете, например, легко пересылать STRING-переменную в массив данных типа CHAR и наоборот.

Тем не менее, оба редактора - и для STL-программ, и для SCL-программ - копируют STRING-переменную в другую STRING-переменную вполне корректно.

# 7 Функции таймеров (Timer Functions)

Функции таймеров используются для управления по времени, например, для обеспечения заданного времени ожидания (waiting) или мониторинга (monitoring time), для измерения отрезков времени или для генерации импульсов

В данной главе рассматриваются выражения, содержащие функции таймеров для использования в языке программирования STL. Для языка SCL функции таймеров включены в состав стандартных функций (см. разд. 30.1 "Функции таймеров" ["Timer Functions"]).

Пользователю доступны следующие типы таймеров:

- Таймер с управляемым импульсом (Pulse timer)
- Таймер с расширенным импульсом (Extended pulse timer)
- Таймер с задержкой включения (On-delay timer)
- Таймер с задержкой включения с памятью (Retentive On-delay timer)
- Таймер с задержкой выключения (Off-delay timer)

При запуске таймера Вы можете задавать динамические характеристики (dynamic response), длительность (duration), длительность задержки запуска/выключения таймера. Вы также можете выключить или включить функцию перезапуска ("retrigger") таймеров. Для опроса таймеров используются двоичные логические операции. Функции загрузки (Load) используются для пересылки текущего значения времени в двоичном или BCD-коде в аккумулятор accumulator 1.

Примеры, рассматриваемые в данной главе, и вызовы IEC-таймеров Вы можете найти на прилагаемой дискете в библиотеке STL_Book library в разделе "Basic Functions" в функциональном блоке FB 107 или в исходном файле Chap_7.

## 7.1 Программирование функций таймеров

### 7.1.1 Запуск таймера

Таймер запускается (т.е. начинает выполняться функция таймера), если перед переходом CPU к инструкции запуска таймера произошло изменение значения результата логической операции RLO. При этом для таймера с задержкой выключения ("Off-delay timer") RLO должен изменить свое состояние со значения "1" на "0", а для всех остальных типов таймеров RLO должен изменить свое состояние со значения "0" на "1".

Вы можете запустить на выполнение любой из пяти возможных таймеров (см. рис. 7.1).

| Таймер запускается как: | На STL | На SCL | |
|---|---|---|---|
| | | | Сигнал запуска |
| Таймер с управляемым импульсом (импульсный) (Pulse timer) | SP | S_PULSE | t |
| Таймер с расширенным импульсом (Extended pulse timer) | SE | S_PEXT | t |
| Таймер с задержкой включения (On-delay timer) | SD | S_ODT | t |
| Таймер с задержкой включения с памятью (Retentive on-delay timer) | SS | S_ODTS | t |
| Таймер с задержкой выключения (Off-delay timer) | SF | S_OFFDT | t |

Рис. 7.1 Инструкции запуска для таймеров

### 7.1.2 Задание временных параметров таймера

При запуске таймера из аккумулятора accumulator 1 выбирается время запуска (running time) или длительность работы (duration). Как и когда в accumulator 1 поступают временные параметры таймера не имеет значения.

Для обеспечения лучшей читаемости программы наилучшим будет вариант, когда эти параметры загружаются в аккумулятор accumulator 1 непосредственно перед запуском функции таймера или в виде константы (непосредственно заданная величина), или в виде переменной (например, слово в памяти, содержащее значение).

Примечание:

аккумулятор accumulator 1 должен содержать корректное значение, даже если отсчет времени не начинается при выполнении инструкции запуска.

*Определение длительности работы (импульса) таймера с помощью константы*

```
L     S5TIME#10s;          //Длительность 10 с
L     S5T#1m10ms;          //Длительность 1 мин + 10 мс
```

В базовых языках STL, LAD и FBD параметры "время запуска" (running time) и "длительность работы (импульса)" (duration) может быть определены в часах, минутах, секундах и миллисекундах. Диапазон задания временных параметров простирается от S5TIME#10ms до S5TIME#2h46m30s (что соответствует диапазону времени, равному 9990 с). При этом Вы можете использовать как формат S5TIME#, так и формат S5T# для определения временных параметров с помощью константы.

*Определение длительности работы (импульса) таймера с помощью переменной*

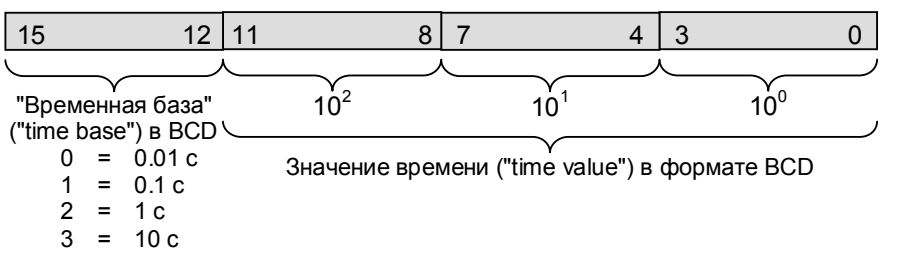
```
L     S5T#10m;             //Длительность (duration) = 10 мин
T     MW20;                //Сохранить "длительность работы"
…     ;
L     MW20;                //Загрузить "длительность работы"
```

*Структура параметра "длительность работы (импульса) таймера" (duration)*

Внутренняя структура временного параметра "длительность работы (импульса)" ("duration") состоит из "значения времени" ("time value") и "временной базы" ("time base"). Длительность работы (импульса) таймера равна произведению этих величин:

"длительность работы (импульса)" = "значение времени" × "временная база".

Длительность работы (импульса) таймера - это период времени, в течение которого таймер находится в активном состоянии ("timer running"). Значение времени ("time value") представляет собой число отрезков времени в сумме дающих полный период времени, в течение которого таймер должен находиться в активном состоянии ("timer running"). Величина такого отрезка времени равна значению "временной базы" ("time base") и является величиной шага по времени, которая используется операционной системой CPU для "декрементирования" таймера (рис. 7.2).




7.2 Назначение битов параметра "длительность работы" ("duration")

Вы можете непосредственно задать параметр "длительность работы" ("duration") в машинном слове. Наименьшее возможное значение для "временной базы" ("time base") обеспечивает более точное исчисление промежутков времени с помощью таймера. Например, если необходимо задать для таймера отрезок времени, равный 1 с, то можно при этом использовать одно из трех значений для "временной базы" ("time base") и, соответственно, для каждого из этих значений Вы получите свое значение длительности работы функции таймера (фактически, значение функции таймера):

"Временная база" ("time base") = 1 с; "Длительность работы" ("duration") = $2001_{hex}$

"Временная база" ("time base") = 100 мс; "Длительность работы" ("duration") = $1010_{hex}$

"Временная база" ("time base") = 10 мс; "Длительность работы" ("duration") = $0100_{hex}$

Последний из трех вариантов является предпочтительным для данного случая.

При запуске таймера CPU использует заданное значение времени ("time value") как период времени, в течение которого таймер находится в активном состоянии ("timer running"). Операционная система обновляет таймеры через фиксированные интервалы времени независимо от процесса сканирования программы пользователя. То есть, CPU производит ступенчатое уменьшение значения функции таймера в соответствии с заданным значением для "временной базы" ("time base").

Когда значение функции таймера достигает уровня "0", это означает, что отсчитываемое таймером время истекло. При этом CPU изменяет состояние (статус) таймера (состояние сигнала в "0" или "1", в зависимости от выбранного типа таймера), при этом функция таймера перестает быть активной до следующего запуска таймера.

Если для функции таймера было задано значение времени ("time value"), равное нулю (0), то таймер остается активным, пока при обработке функции таймера CPU не обнаружит, что время заданное для таймера "истекло".

Таймеры обновляются асинхронно по отношению к процессу сканирования программы пользователя. Следовательно, возможно, что состояние таймера в начале цикла сканирования отличается от его состояния в конце цикла. Если Вы используете функцию таймера только в одном месте программы и придерживаетесь нижеследующих рекомендаций, то не возникнет ошибок из-за асинхронного обновления таймера в программе.

### 7.1.3 Сброс таймера (Resetting a timer)

Следующая инструкция

R T n вызывает сброс таймера.

Таймер сбрасывается, при результате логической операции RLO, равном "1", при появлении вышеуказанной инструкции. Пока RLO равен "1", проверки таймера на состояние "1" возвращают результат проверки "0"; проверки таймера на состояние "0" возвращают результат проверки "1".

Сброс таймера устанавливает для значения времени ("time value") и для "временной базы" ("time base") нулевые значения (0).

Примечание:

Сброс функции таймера не сбрасывает внутренний меркер фронта для запуска. Для повторного запуска CPU должен обработать инструкцию запуска таймера при RLO, равном "0", и только после этого при появлении фронта сигнала в соответствующем меркере фронта функция таймера сможет стартовать.

## 7.1.4 Разблокировка таймера (Enabling a timer)

Следующая инструкция

| | | |
|---|---|---|
| FR | T n | позволяет перезапуск таймера. |

Инструкция FR используется для перезапуска таймера, находящегося в активном состоянии (т.е., когда функция таймера находится в активном состоянии).

Таймер перезапускается, если инструкция FR обрабатывается при положительном (возрастающем) фронте сигнала. При этом внутренний меркер фронта сбрасывается для обеспечения возможности запуска таймера. Если после этого результат логической операции RLO становится равным "1" перед появлением вышеуказанной инструкции, то таймер запускается, даже если при этом не определяется фронт сигнала.

Сброс таймера вызывает изменение величин времени ("time value") и "временной базы" ("time base") на нулевые значения (0).

Примечание:

Данная инструкция разблокировки функции таймера не требуется для запуска или сброса таймера, т.е., инструкция не является необходимой для обычных условий работы с таймером.

## 7.1.5 Проверка (опрос) таймера (Checking a timer)

**Проверка (опрос) состояния таймера**

| | | |
|---|---|---|
| A | T n | проверка сигнала на состояние "1" и комбинирование с RLO в соответствии с логикой AND (И). |
| O | T n | проверка сигнала на состояние "1" и комбинирование с RLO в соответствии с логикой OR (ИЛИ). |
| X | T n | проверка сигнала на состояние "1" и комбинирование с RLO в соответствии с логикой Exclusive OR (Исключающее ИЛИ). |
| AN | T n | проверка сигнала на состояние "0" и комбинирование с RLO в соответствии с логикой AND (И). |

| | | |
|---|---|---|
| ON | T n | проверка сигнала на состояние "0" и комбинирование с RLO в соответствии с логикой OR (ИЛИ). |
| XN | T n | проверка сигнала на состояние "0" и комбинирование с RLO в соответствии с логикой Exclusive OR (Исключающее ИЛИ). |

Вы можете опрашивать таймер, как если бы это был, например, вход (input), и в дальнейшем использовать результат этой проверки. В зависимости от типа таймера проверка (опрос) сигнала на состояние "1" вызывает разное поведение сигнала (импульса) таймера во времени (см. далее описание динамических характеристик).

Как и в случае с опросом входа (input) проверка (опрос) сигнала на состояние "0" возвращает результат проверки таймера с точностью до наоборот по сравнению с опросом на состояние "1" - результат проверки инвертируется.

**Проверка (опрос) значения таймера (time value)**

Инструкция

| | | |
|---|---|---|
| L | T n | загружает двоичное значение времени ("time value") таймера, |
| LC | T n | загружает значение времени ("time value") таймера в двоично-десятичном формате (BCD). |

Функции загрузки L T и LC T считывают определенное значение времени ("time value") таймера и пересылают его в аккумулятор accumulator 1. При этом инструкция L загружает значение времени ("time value") в двоичном, а инструкция LC загружает значение времени таймера в двоично-десятичном формате. Значение времени ("time value") таймера, пересылаемое в аккумулятор accumulator 1, представляет собой текущее значение, соответствующее моменту времени, когда производится опрос таймера (если функция таймера находится в активном состоянии, то значение времени, которое загружается в аккумулятор, будет одним из значений, полученным CPU при декрементировании заданного исходного значения в сторону нулевого значения).

*Непосредственная загрузка значения таймера (time value)*

Значение времени ("time value"), определенное с помощью инструкции таймера, имеет двоичный формат, и может быть переслано в аккумулятор accumulator 1 в этом же (двоичном) формате. В этом случае "временная база" ("time base") будет потеряна и на ее месте в аккумуляторе accumulator 1 будут записаны нули "0".

Следовательно, в аккумуляторе будет значение, которое соответствует положительному целому (INT) числу и которое может быть обработано в дальнейшем с помощью, например, функций сравнения.

Примечание:

Это значение таймера не есть "заданная длительность работы" ("duration").

Пример:

```
L    T 15;      //загрузка текущего значения времени ("time value")
                //таймера
T    MW 34;     //сохранение текущего значения времени ("time value")
```

*Загрузка значения таймера (загрузка в формате BCD ["coded load"])*

Вы можете также использовать инструкцию для т.н. "кодированной" пересылки ("coded load") в аккумулятор двоичного значения времени ("time value") таймера. В этом случае и значение времени ("time value") и "временная база" ("time base") будут иметь формат BCD (двоично-десятичный). При этом также как и в предыдущем случае в содержимом аккумулятора accumulator 1 в левое машинное слово (в старшее слово) будут записаны нули "0".

Пример:

```
LC   T 16;      //загрузка текущего значения времени ("time value")
                //таймера в BCD-формате
T    MW 122;    //сохранение текущего значения времени ("time value")
```

### 7.1.6 Последовательность инструкций при использовании функций таймера

При программировании таймера Вам нет необходимости использовать все возможные выражения для активации функций таймеров. Вы должны использовать только те из функций, которые Вам необходимо выполнить. Обычно используется таймер с заданной длительностью импульса и двоичный опрос состояния таймера.

Чтобы выполнить функцию таймера в соответствии с описанием в предыдущих разделах необходимо соблюдать определенный порядок при программировании соответствующих операторов.

В таблице 7.1 показан оптимальный порядок для всех операторов при программировании функций таймера.

Таблица 7.1 Последовательность операторов для таймера

| Функция таймера: | Примеры: |
|---|---|
| Разблокировка таймера (Enable timer) | `A  I  16.5`<br>`FR T  5` |
| Запуск таймера (Start timer) | `A  I  17.5`<br>`L  S5T#1s`<br>`SP T  5` |
| Сброс таймера (Reset timer) | `A  I  18.0`<br>`R  T  5` |
| Опрос численного значения таймера (Digital timer check) | `L  T  5`<br>`T  MW20`<br>`LC T  5`<br>`T  MW22` |
| Двоичный опрос таймера (Binary timer check) | `A  T  5`<br>`=  Q  2.0` |

При использовании операторов выбирайте те, которые необходимы Вам, и просто пропускайте ненужные.

Если таймер запускается и сбрасывается "одновременно" (как в показанной последовательности операторов), то функция таймера сначала будет запущена на выполнение, а следующее выражение немедленно сбросит таймер. Следовательно, если после этого таймер будет проверен (опрошен), то факт запуска таймера останется незамеченным.

### 7.1.7 Пример часового генератора (генератора часов)

Пример показывает, как запрограммировать часовой генератор с разным отношением импульс/пауза с помощью отдельного таймера.

Со входа *Start_Input* обеспечивается запуск часового генератора. Если еще не начался отсчет времени или отсчет закончился, генератор запускается в режиме расширенного импульса (extended pulse). При каждом запуске двоичный делитель *Output* меняет состояние сигнала и при этом определяется длительность работы (duration).

```
     AN    Start_input;
     R     Timer;
     R     Output;
     JC    M1;
     A     Timer;
     JC    M1;
     AN    Output;
     =     Output;
     L     Pulse_duration;
     JC    M2;
     L     Pulse_duration;
M2:  SE    Timer;
M1:  ;     //Остальная часть программы
```

## 7.2 Таймер с управляемым импульсом (Pulse timer)

Ниже представлен пример законченной программы на STL для запуска таймера в режиме "управляемого импульса" (Pulse timer):

```
     A     Enable_input;
     FR    Timer;
     A     Start_input;
     L     Duration;
```

```
SP      Timer;
A       Reset_input;
R       Timer;
L       Timer;
T       Binary_time_value;
LD      Timer;
T       BCD_time_value;
A       Timer;
=       Timer_status;
```

Программа на SCL для вызова таймера в режиме "управляемого импульса" (Pulse timer):

```
BCD_time_value   := S_PULSE (
     T_NO        := Timer,
     S           := Start_input,
     TV          := Duration,
     R           := Reset_input,
     Q           := Timer_status,
     BI          := Binary_time_value);
```

**Запуск таймера с управляемым импульсом (Starting a pulse timer)**

На рисунке 7.3 показаны динамические характеристики таймера, запускаемого в режиме управляемого импульса (Pulse), и реакция на сброс.

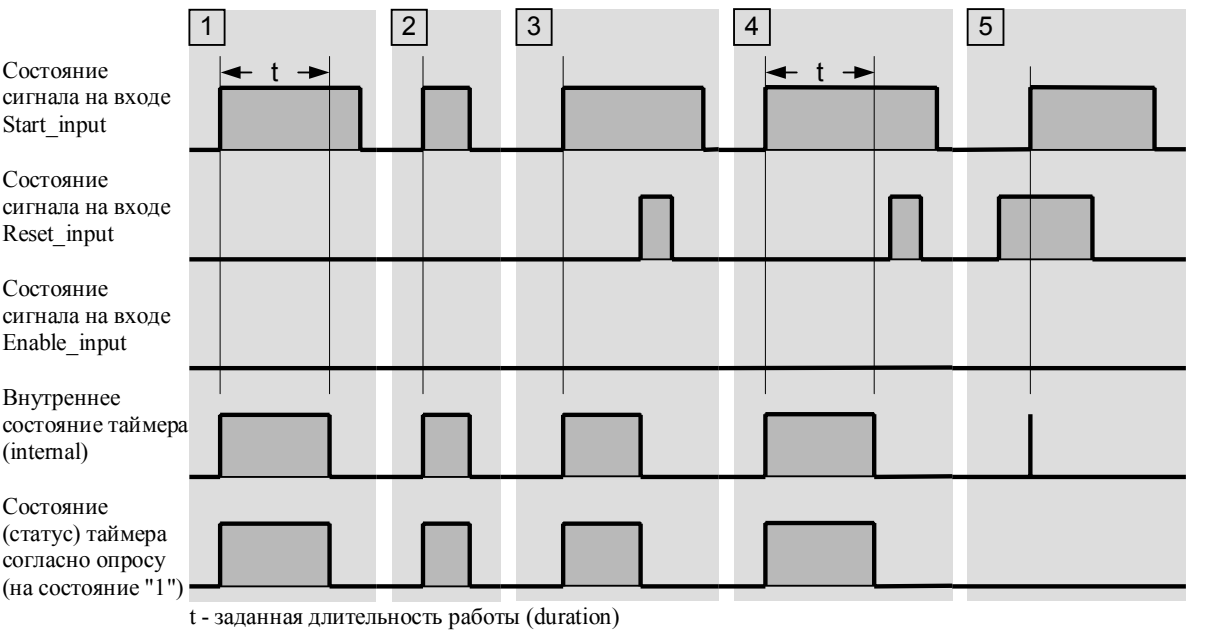



Рис. 7.3 Отклик таймера с управляемым импульсом на сигналы запуска и сброса

Показанное на рис. 7.3 поведение таймера будет соответствовать действительности, если Вы будете придерживаться рекомендованной выше последовательности операторов для STL (сначала запуск, затем сброс, затем опрос таймера). Обычно операция разблокирования (Enabling a timer) не требуется, и для программы на SCL она также не является необходимой.

1 Функция таймера запускается, когда состояние сигнала на входе Start_input меняется от состояния "0" к состоянию "1" (положительный фронт). Таймер работает, пока состояние сигнала на входе Start_input остается равным "1". Проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "1", пока функция таймера активна.

Убывающее значение времени (как значение таймера) отсчитывается от некоторого заданного начального (initial) значения с заданным шагом, равным заданной величине "временной базы" (time base).

2 Функция таймера перестает быть активной, когда состояние сигнала на входе Start_input меняется от состояния "1" к состоянию "0" (отрицательный фронт), если даже это происходит до момента истечения заданного времени работы (длительности - "duration"). После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0". Значение времени (как значение таймера) показывает время, оставшееся до окончания заданного периода работы (длительности - "duration"), обозначая точку на временной оси, в которой произошло преждевременное прерывание работы таймера.

**Сброс таймера с управляемым импульсом (Resetting a pulse timer)**

Операция сброса (Resetting a pulse timer) имеет статический эффект и имеет приоритет перед запуском таймера (см. рис. 7.3).

3 Состояние сигнала на входе Reset_input, равное "1", вызывает сброс таймера, если он был до этого активен. После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0". Значение времени (time value) и значение "временной базы" (time base) также сбрасываются и становятся равными "0". Отрицательный фронт сигнала (т.е. переход его от состояния "1" к "0") на входе Reset_input в то время, пока на входе Start_input присутствует состояние "1", никак не сказывается на режиме таймера.

4 Если функция таймера не активна, то присутствие состояния "1" на входе Reset_input также никак не сказывается на режиме таймера.

5 Если сигнал на входе Start_input изменяет свое состояние с "0" на "1" (присутствует положительный фронт сигнала), в то время, как на входе Reset_input присутствует состояние "1", то таймер запускается, но последующая инструкция сброса немедленно его сбрасывает (на рис. 7.3 это показано жирной вертикальной чертой). Если проверки (опросы) таймера на состояние "1" (timer status) следуют по времени после сброса таймера, то короткое время работы таймера после его запуска не скажется на результатах проверки - результат проверки будет равен "0".

**Разблокировка таймера с управляемым импульсом (Enabling a pulse timer)**

На рисунке 7.4 показана функция разблокировки таймера, запускаемого в режиме управляемого импульса (Pulse).

Функция разблокировки таймера (Enabling a pulse timer) позволяет вновь запускать (в том числе и уже активный таймер) посредством приложения ко входу Enabling_input положительного фронта сигнала (переход сигнала от состояния "0" к состоянию "1").

Функция разблокировки таймера (Enabling a pulse timer) доступна для использования только в языке программирования STL.

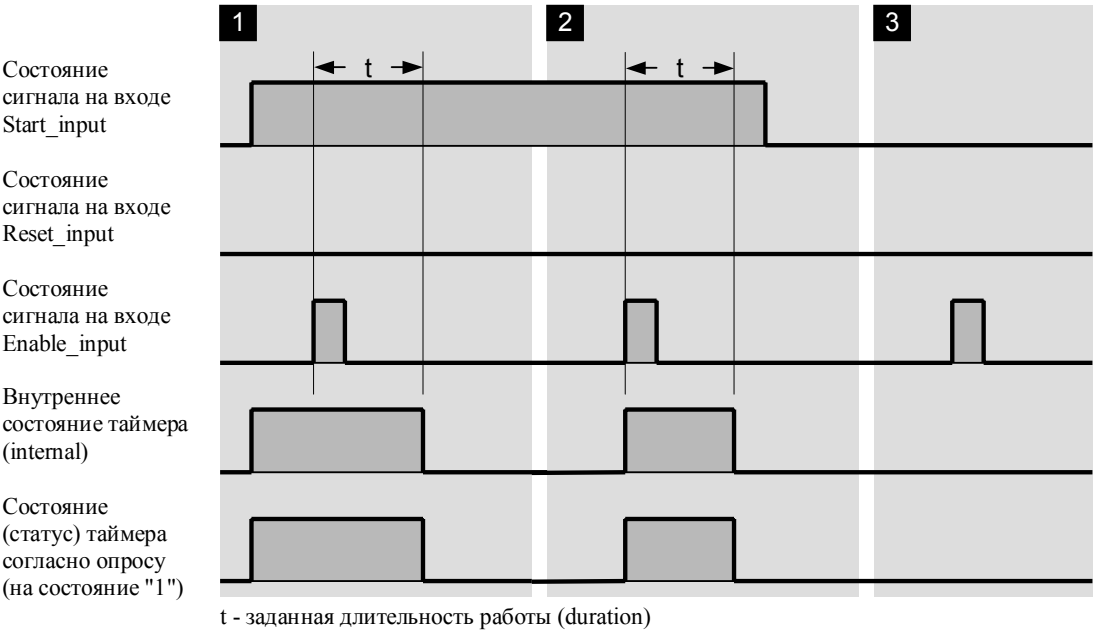



Рис. 7.4 Функция разблокировки (Enabling) таймера с управляемым импульсом

**1** Если функция таймера активна и состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт), то таймер будет перезапущен, если состояние сигнала на входе Start_input остается равным "1". При этом заданное значение длительности работы ("duration") будет вновь взято как исходное значение в момент перезапуска таймера. Последующее изменение сигнала на входе Enable_input от состояния "1" к состоянию "0" (отрицательный фронт) не изменяет режима таймера.

**2** Если состояние сигнала на входе Start_input все еще остается равным "1", а функция таймера пассивна, то в момент изменения сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) таймер будет вновь запущен в режиме управляемого импульса (pulse timer). При этом заданное значение длительности работы ("duration") будет взято как исходное значение.

**3** Если состояние сигнала на входе Start_input равно "0", то изменение сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) не будет иметь никакого эффекта.

## 7.3 Таймер с расширенным импульсом (Extended pulse timer)

Ниже представлен пример законченной программы на STL для запуска таймера в режиме "расширенного импульса" (Extended pulse timer):

```
A       Enable_input;
FR      Timer;
A       Start_input;
L       Duration;
SE      Timer;
A       Reset_input;
R       Timer;
L       Timer;
T       Binary_time_value;
LD      Timer;
T       BCD_time_value;
A       Timer;
=       Timer_status;
```

Программа на SCL для вызова таймера в режиме "расширенного импульса" (Extended pulse timer):

```
BCD_time_value   := S_PEXT (
     T_NO        := Timer,
     S           := Start_input,
     TV          := Duration,
     R           := Reset_input,
     Q           := Timer_status,
     BI          := Binary_time_value);
```

**Запуск таймера с расширенным импульсом (Starting an extended pulse timer)**

На рисунке 7.5 показаны динамические характеристики таймера, запускаемого в режиме "расширенного импульса" (Extended pulse timer), и его реакция на сигнал сброса.

Показанное на рис. 7.5 поведение таймера будет соответствовать действительности, если Вы будете придерживаться рекомендованной выше последовательности операторов для STL (сначала запуск, затем сброс, затем опрос таймера). Обычно операция разблокирования (Enabling a timer) не требуется для работы таймера в программе STL, и для программы на SCL она также не является необходимой.

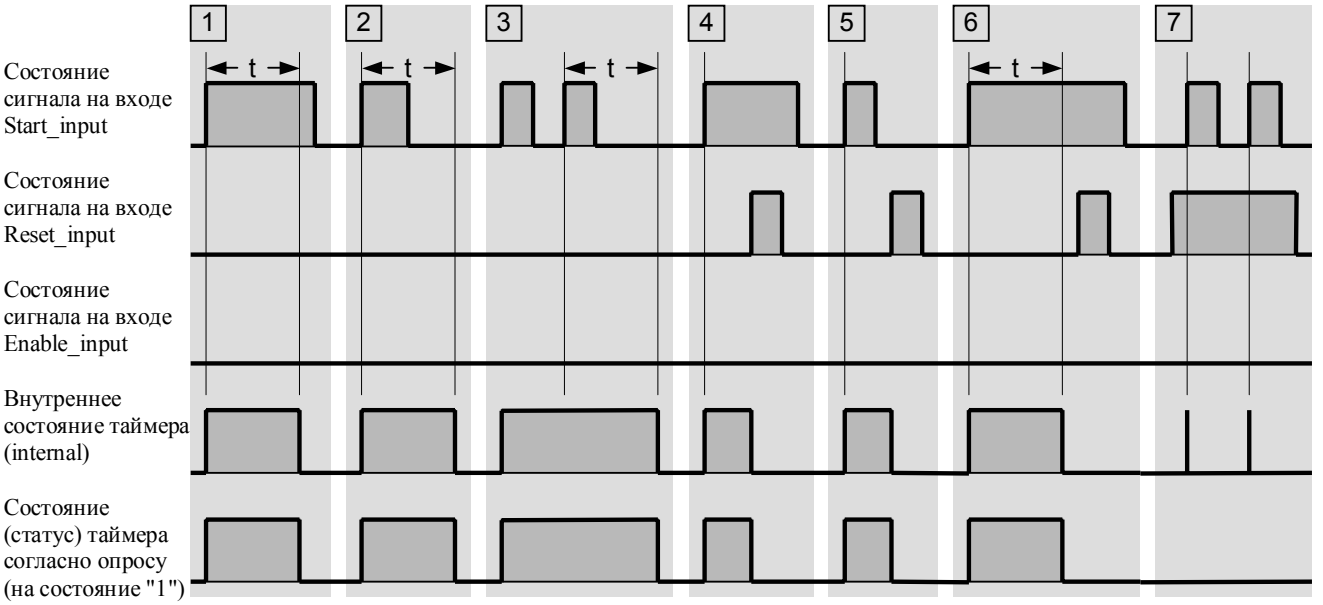



Рис. 7.5 Отклик таймера в режиме "расширенного импульса" (Extended pulse timer) на сигналы запуска и сброса

1 2 Функция таймера запускается, когда состояние сигнала на входе Start_input меняется от состояния "0" к состоянию "1" (положительный фронт). Таймер работает до момента истечения заданного времени работы (длительности - "duration"), даже если до этого момента состояние сигнала на входе Start_input изменится снова от состояния "1" к состоянию "0". Проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "1", пока функция таймера активна.

Убывающее значение времени (как значение таймера) отсчитывается от некоторого заданного начального (initial) значения с заданным шагом, равным заданной величине "временной базы" (time base).

3 Если состояние сигнала на входе Start_input меняется от состояния "0" к состоянию "1" (положительный фронт), в то время, пока функция таймера активна, то таймер будет перезапущен. При этом заданное значение длительности работы ("duration") будет вновь взято как исходное значение в момент перезапуска таймера.

Таким образом таймер может быть перезапущен столько раз, сколько требуется для нормальной работы программы, невзирая на время, оставшееся в предыдущем рабочем периоде таймера.

**Сброс таймера с режимом "расширенного импульса" (Extended pulse timer)**

Операция сброса таймера с режимом "расширенного импульса" (Extended pulse timer) имеет статический эффект и имеет приоритет перед запуском таймера (см. рис. 7.5).

4 5 Состояние сигнала на входе Reset_input, равное "1", вызывает сброс таймера, если он был до этого активен. После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0". Значение времени (time value) и значение "временной базы" (time base) также сбрасываются и становятся равными "0". Отрицательный фронт сигнала (т.е. переход его от состояния "1" к "0") на входе Reset_input в то время, пока на входе Start_input присутствует состояние "1", никак не сказывается на режиме таймера.

6 Если функция таймера не активна, то присутствие состояния "1" на входе Reset_input также никак не сказывается на режиме таймера.

7 Если сигнал на входе Start_input изменяет свое состояние с "0" на "1" (присутствует положительный фронт сигнала), в то время, как на входе Reset_input присутствует состояние "1", то таймер запускается, но последующая инструкция сброса немедленно его сбрасывает (на рис. 7.5 это показано жирной вертикальной чертой). Если проверки (опросы) таймера на состояние "1" (timer status) следуют по времени после сброса таймера, то короткое время работы таймера после его запуска не скажется на результатах проверки - результат проверки будет равен "0".

**Разблокировка таймера с режимом "расширенного импульса" (Extended pulse timer)**

Функция разблокировки таймера (Anabling extended pulse timer) позволяет вновь запускать (в том числе и уже активный таймер) посредством приложения ко входу Enabling_input положительного фронта сигнала (переход сигнала от состояния "0" к состоянию "1").

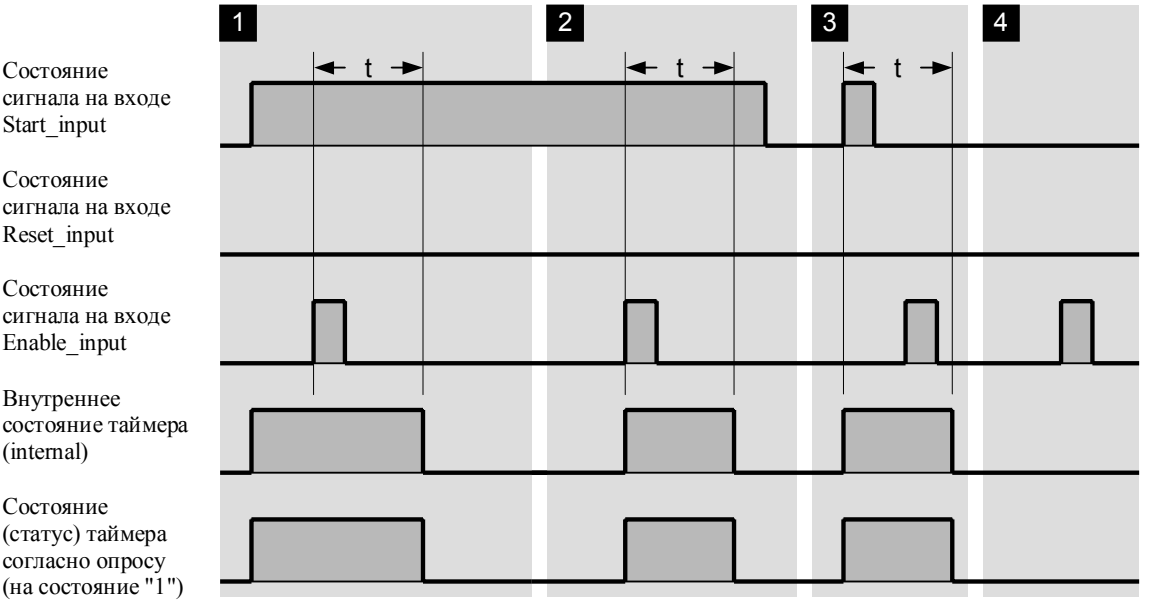
Функция разблокировки таймера (Anabling a pulse timer) доступна для использования только в языке программирования STL.

1 Если функция таймера активна и состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт), то таймер будет перезапущен, если состояние сигнала на входе Start_input остается равным "1". При этом заданное значение длительности работы ("duration") будет вновь взято как исходное значение в момент перезапуска таймера. Последующее изменение сигнала на входе Enable_input от состояния "1" к состоянию "0" (отрицательный фронт) не изменяет режима таймера.

2 Если состояние сигнала на входе Start_input все еще остается равным "1", а функция таймера пассивна, то в момент изменения сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) таймер будет вновь запущен в режиме "расширенного импульса" (extended pulse timer). При этом заданное значение длительности работы ("duration") будет взято как исходное значение.

3 4 Если состояние сигнала на входе Start_input равно "0", то изменение сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) не будет иметь никакого эффекта.

На рисунке 7.6 показана функция разблокировки таймера, запускаемого в режиме "расширенного импульса" (Extended pulse timer).




t - заданная длительность работы (duration)

Рис. 7.6 Функция разблокировки (Enabling) таймера с режимом "расширенного импульса" (Extended pulse timer).

## 7.4 Таймер с задержкой включения (On-delay timer)

Ниже представлен пример законченной программы на STL для запуска таймера с задержкой включения (On-delay timer):

```
A       Enable_input;
FR      Timer;
A       Start_input;
L       Duration;
SD      Timer;
A       Reset_input;
R       Timer;
L       Timer;
T       Binary_time_value;
LD      Timer;
```

```
T       BCD_time_value;
A       Timer;
=       Timer_status;
```

Программа на SCL для вызова таймера с задержкой включения (On-delay timer):

```
BCD_time_value   := S_ODT (
     T_NO        := Timer,
     S           := Start_input,
     TV          := Duration,
     R           := Reset_input,
     Q           := Timer_status,
     BI          := Binary_time_value);
```

**Запуск таймера с задержкой включения (On-delay timer)**

На рисунке 7.7 показаны динамические характеристики таймера, запускаемого в режиме с задержкой включения (On-delay timer), и его реакция на сброс.

Показанное на рис. 7.7 поведение таймера будет соответствовать действительности, если Вы будете придерживаться рекомендованной выше последовательности операторов для STL (сначала запуск, затем сброс, затем опрос таймера). Обычно операция разблокирования (Enabling a timer) не требуется для работы таймера в программе STL, она также не является необходимой и для программы на SCL.

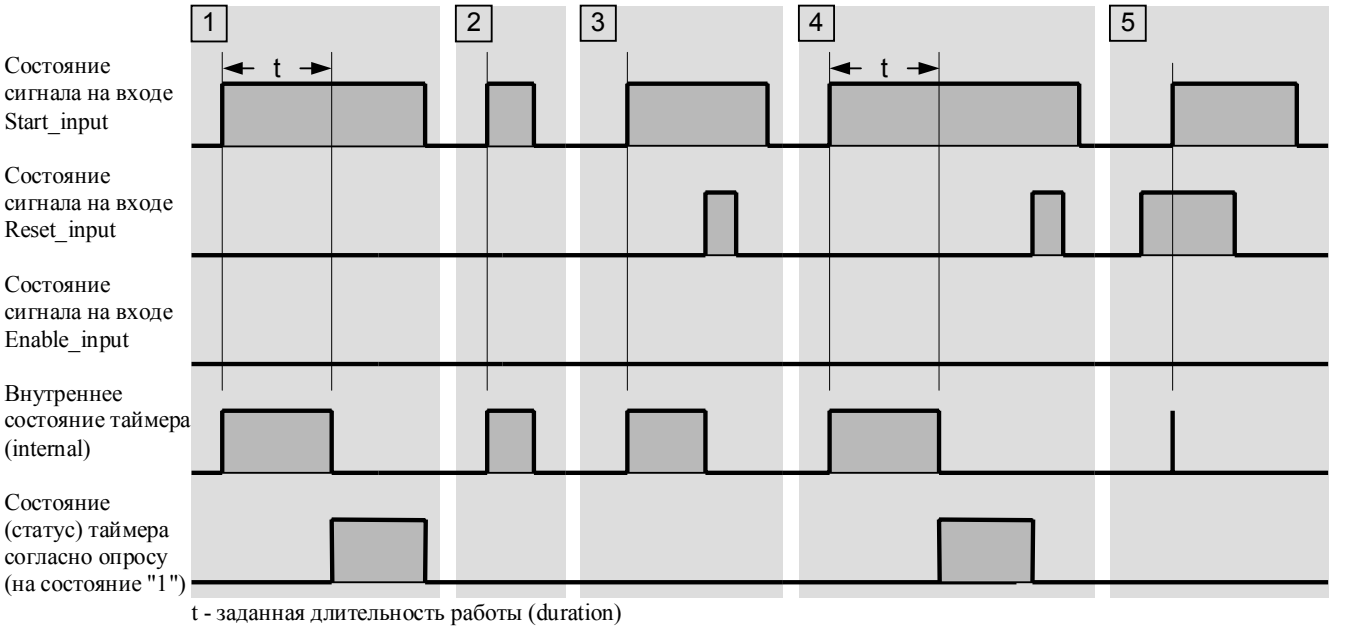



Рис. 7.7 Отклик таймера с задержкой включения (On-delay timer) на сигналы запуска и сброса

1 Функция таймера запускается, когда состояние сигнала на входе Start_input меняется от состояния "0" к состоянию "1" (положительный фронт). Таймер работает, до момента истечения заданного времени работы (длительности - "duration"). Проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "1", если до момента истечения заданного времени работы таймера (длительности - "duration") не сбрасывался в состояние "0" сигнал на входе Start_input, и пока состояние сигнала на входе Start_input остается равным "1", не появится сигнал, равный "1", на входе Reset_input, вызывающий сброс таймера.

Убывающее значение времени (как значение таймера) отсчитывается от некоторого заданного начального (initial) значения с заданным шагом, равным заданной величине "временной базы" (time base).

2 Функция таймера перестает быть активной, когда состояние сигнала на входе Start_input меняется от состояния "1" к состоянию "0" (отрицательный фронт), если даже это происходит до момента истечения заданного времени работы (длительности - "duration"). После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0". Значение времени (как значение таймера) показывает время, оставшееся до окончания заданного периода работы (длительности - "duration"), обозначая точку на временной оси, в которой произошло преждевременное прерывание работы таймера.

**Сброс таймера с задержкой включения (Resetting an on-delay timer)**

Операция сброса таймера с задержкой включения (Resetting an on-delay timer) имеет статический эффект и имеет приоритет перед запуском таймера (см. рис. 7.7).

3 4 Состояние сигнала на входе Reset_input, равное "1", вызывает сброс функции таймера, кончился ли отсчет заданного времени таймера или нет. После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0", даже если отсчет заданного времени таймера закончился и на входе Start_input присутствует состояние "1". Значение времени (time value) и значение "временной базы" (time base) также сбрасываются и становятся равными "0".
Отрицательный фронт сигнала (т.е. переход его от состояния "1" к "0") на входе Reset_input в то время, пока на входе Start_input присутствует состояние "1", никак не сказывается на режиме таймера.
Результат логической операции RLO, равный "1", на входе Reset_input также сбрасывает таймер, даже если отсчет заданного времени таймера закончился. После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0"

5 Если сигнал на входе Start_input изменяет свое состояние с "0" на "1" (положительный фронт сигнала), в то время, как на входе Reset_input присутствует состояние "1", то таймер запускается, но последующая инструкция сброса немедленно его сбрасывает (на рис. 7.7 это показано жирной вертикальной чертой). Если проверки (опросы) таймера на состояние "1" (timer status) следуют по времени после сброса таймера, то короткое время работы таймера после его запуска не скажется на результатах проверки - результат проверки будет равен "0".

**Разблокировка таймера с задержкой включения (Anabling an on-delay timer)**

Функция разблокировки таймера (Anabling an on-delay timer) позволяет вновь запускать отсчет времени (или перезапускать сначала отсчет времени уже активного таймера) посредством приложения ко входу Enabling_input положительного фронта сигнала (переход сигнала от состояния "0" к состоянию "1").

Функция разблокировки таймера (Anabling an on-delay timer) доступна для использования только в языке программирования STL.

На рисунке 7.8 показана функция разблокировки таймера, запускаемого в режиме с задержкой включения (On-delay timer).

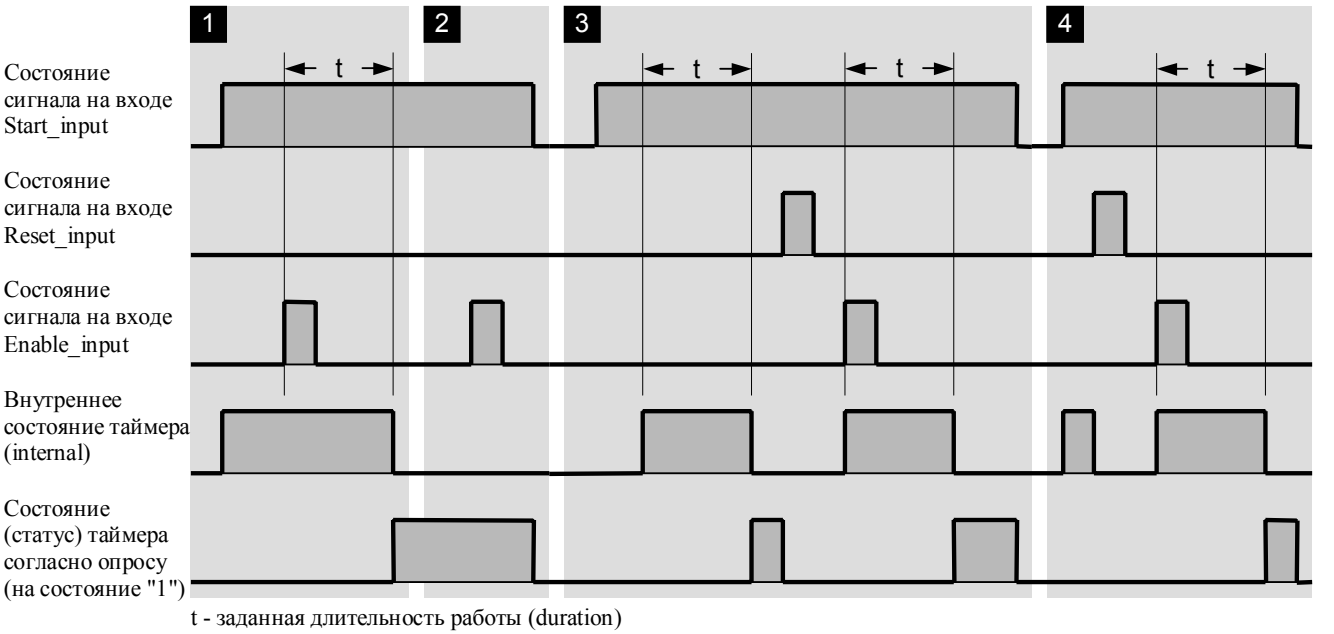



Рис. 7.8 Функция разблокировки (Enabling) таймера с задержкой включения (On-delay timer)

1 Если функция таймера активна (идет отсчет времени таймера) и состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт), то отсчет времени таймера будет перезапущен сначала, если состояние сигнала на входе Start_input остается равным "1". При этом заданное значение длительности работы ("duration") будет вновь взято как исходное значение в момент перезапуска таймера. Последующее изменение сигнала на входе Enable_input от состояния "1" к состоянию "0" (отрицательный фронт) не изменяет режима таймера.

2 Если состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт) после того, как отсчет заданного времени таймера закончился без прерывания, на режим работы таймера этот положительный фронт не окажет никакого влияния.

3 4 Если состояние сигнала на входе Start_input все еще остается равным "1", а функция таймера была деактивирована сигналом на входе сброса Reset_input, то в момент изменения сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) таймер будет вновь запущен в режиме с задержкой включения (On-delay timer). При этом заданное значение длительности работы ("duration") будет взято как исходное значение.

Если состояние сигнала на входе Start_input равно "0", то изменение сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) не будет иметь никакого эффекта.

## 7.5 Таймер с задержкой включения с памятью (Retentive On-delay timer)

Ниже представлен пример законченной программы на STL для запуска таймера с задержкой включения с памятью (Retentive On-delay timer):

```
A       Enable_input;
FR      Timer;
A       Start_input;
L       Duration;
SS      Timer;
A       Reset_input;
R       Timer;
L       Timer;
T       Binary_time_value;
LD      Timer;
T       BCD_time_value;
A       Timer;
=       Timer_status;
```

Программа на SCL для вызова таймера с задержкой включения с памятью (Retentive On-delay timer):

```
BCD_time_value   := S_ODTS (
     T_NO        := Timer,
     S           := Start_input,
     TV          := Duration,
     R           := Reset_input,
     Q           := Timer_status,
     BI          := Binary_time_value);
```

**Запуск таймера с задержкой включения с памятью (Retentive On-delay timer)**

На рисунке 7.9 показаны динамические характеристики таймера, запускаемого в режиме с задержкой включения с памятью (Retentive On-delay timer), и его реакция на сброс.

Показанное на рис. 7.9 поведение таймера будет соответствовать действительности, если Вы будете придерживаться рекомендованной выше последовательности операторов для STL (сначала запуск, затем сброс, затем опрос таймера). Обычно операция разблокирования (Enabling a timer) не требуется для работы таймера в программе STL, и для программы на SCL она также не является необходимой.

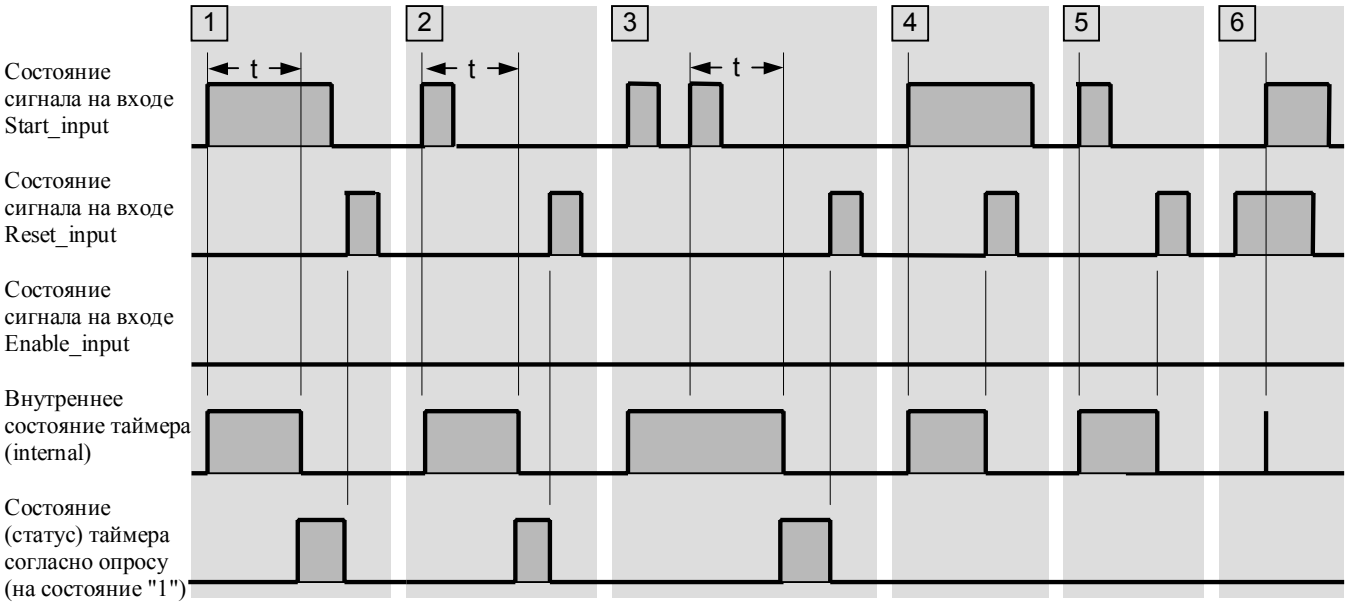



Рис. 7.9 Отклик таймера с задержкой включения с памятью (Retentive On-delay timer) на сигналы запуска и сброса

1 2 Функция таймера запускается, когда состояние сигнала на входе Start_input меняется от состояния "0" к состоянию "1" (положительный фронт). Отсчет времени таймера происходит до момента истечения заданного времени работы (длительности - "duration"), даже если до этого момента состояние сигнала на входе Start_input изменится снова от состояния "1" к состоянию "0". Если отсчет времени завершился без прерывания, то в дальнейшем проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "1" вне зависимости от состояния сигнала на входе Start_input. При этом проверки (опросы) таймера на состояние "1" (timer status) будут возвращать результат проверки "1" до момента сброса таймера вне зависимости от состояния сигнала на входе Start_input.

Убывающее значение времени (как значение таймера) отсчитывается от некоторого заданного начального (initial) значения с заданным шагом, равным заданной величине "временной базы" (time base).

3 Если состояние сигнала на входе Start_input меняется от состояния "0" к состоянию "1" (положительный фронт), в то время, пока функция таймера активна, то таймер будет перезапущен. При этом заданное значение длительности работы ("duration") будет вновь взято как исходное значение в момент перезапуска таймера.

Таким образом таймер может быть перезапущен столько раз, сколько требуется для нормальной работы программы, невзирая на время отсчета, оставшееся в предыдущем рабочем периоде таймера.

**Сброс таймера с задержкой включения с памятью (Retentive On-delay timer)**

Операция сброса таймера с задержкой включения с памятью (Retentive On-delay timer) имеет статический эффект и имеет приоритет перед запуском таймера (см. рис. 7.9).

4 5 Состояние сигнала на входе Reset_input, равное "1", вызывает сброс таймера, если он был до этого активен и вне зависимости от состояния сигнала на входе Start_input. После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0". Значение времени (time value) и значение "временной базы" (time base) также сбрасываются и становятся равными "0".

6 Если сигнал на входе Start_input изменяет свое состояние с "0" на "1" (положительный фронт сигнала), в то время, как на входе Reset_input присутствует состояние "1", то таймер запускается, но последующая инструкция сброса немедленно его сбрасывает (на рис. 7.9 это показано жирной вертикальной чертой). Если проверки (опросы) таймера на состояние "1" (timer status) следуют по времени после сброса таймера, то короткое время работы таймера после его запуска не скажется на результатах проверки - результат проверки будет равен "0".

**Разблокировка таймера с задержкой включения с памятью (Anabling a retentive on-delay timer)**

Функция разблокировки таймера (Anabling a retentive on-delay timer) позволяет вновь запускать отсчет времени (или перезапускать сначала отсчет времени уже активного таймера) посредством приложения ко входу Enabling_input положительного фронта сигнала (переход сигнала от состояния "0" к состоянию "1").

Функция разблокировки таймера (Anabling a retentive on-delay timer) доступна для использования только в языке программирования STL.

На рисунке 7.10 показана функция разблокировки таймера, запускаемого в режиме с задержкой включения с памятью (Retentive On-delay timer).

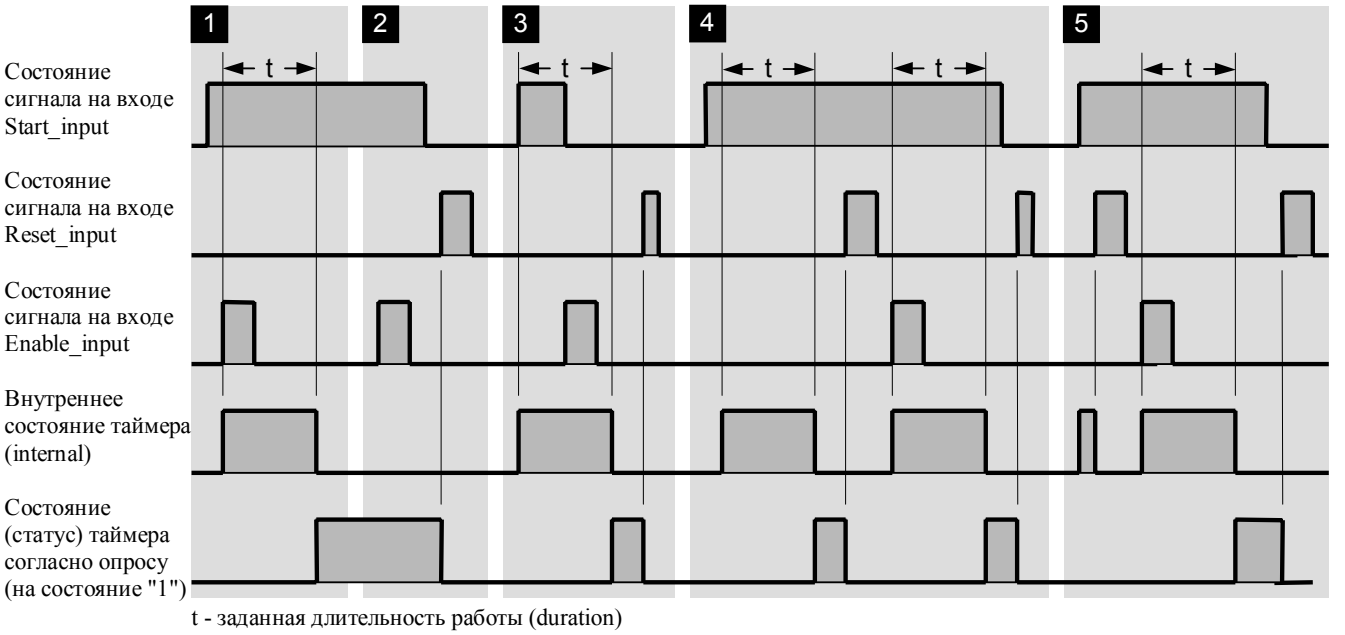



Рис. 7.10 Функция разблокировки (Enabling) таймера с задержкой включения с памятью (Retentive On-delay timer)

**1** Если функция таймера активна (идет отсчет времени таймера) и состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт), то отсчет времени таймера будет перезапущен сначала, если состояние сигнала на входе Start_input остается равным "1". При этом заданное значение длительности работы ("duration") будет вновь взято как исходное значение в момент перезапуска таймера. Последующее изменение сигнала на входе Enable_input от состояния "1" к состоянию "0" (отрицательный фронт) не изменяет режима таймера.

**2** Если состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт) после того, как отсчет заданного времени таймера закончился без прерывания, на режим работы таймера этот положительный фронт не окажет никакого влияния.

**3** Если состояние сигнала на входе Start_input равно "0", то изменение сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) не будет иметь никакого эффекта.

**4** **5** Если состояние сигнала на входе Start_input все еще остается равным "1", а функция таймера была деактивирована сигналом на входе сброса Reset_input, то в момент изменения сигнала на входе Enable_input от состояния "0" к состоянию "1" (положительный фронт) таймер будет вновь запущен в режиме с задержкой включения с памятью (Retentive on-delay timer). При этом заданное значение длительности работы ("duration") будет взято как исходное значение.

## 7.6 Таймер с задержкой выключения (Off-delay timer)

Ниже представлен пример законченной программы на STL для запуска таймера с задержкой выключения (Off-delay timer):

```
A       Enable_input;
FR      Timer;
A       Start_input;
L       Duration;
SF      Timer;
A       Reset_input;
R       Timer;
L       Timer;
T       Binary_time_value;
LD      Timer;
T       BCD_time_value;
A       Timer;
=       Timer_status;
```

Программа на SCL для вызова таймера с задержкой выключения (Off-delay timer):

```
BCD_time_value   := S_OFFDT (
     T_NO        := Timer,
     S           := Start_input,
     TV          := Duration,
     R           := Reset_input,
     Q           := Timer_status,
     BI          := Binary_time_value);
```

### Запуск таймера с задержкой выключения (Off-delay timer)

На рисунке 7.11 показаны динамические характеристики таймера, запускаемого в режиме с задержкой выключения (Off-delay timer), и его реакция на сброс.

Показанное на рис. 7.11 поведение таймера будет соответствовать действительности, если Вы будете придерживаться рекомендованной выше последовательности операторов для STL (сначала запуск, затем сброс, затем опрос таймера). Обычно операция разблокирования (Enabling a timer) не требуется для работы таймера в программе STL, и для программы на SCL она также не является необходимой.

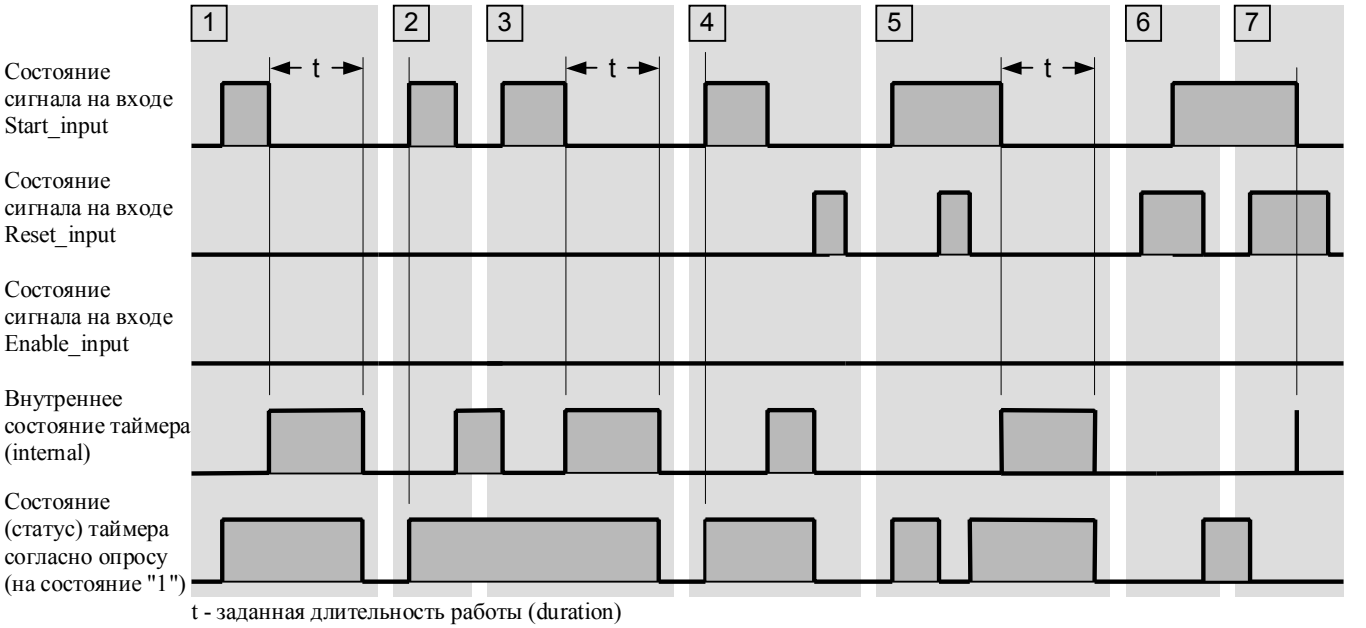



Рис. 7.11 Отклик таймера с задержкой выключения (Off-delay timer) на сигналы запуска и сброса

1 2 Функция таймера запускается, когда состояние сигнала на входе Start_input меняется от состояния "1" к состоянию "0" (отрицательный фронт). Таймер работает, до момента истечения заданного времени работы (длительности - "duration"). Проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "1", если до момента истечения заданного времени работы таймера (длительности - "duration") на входе Reset_input не появлялся сигнал "1", вызывающий сброс таймера, и если состояние сигнала на входе Start_input остается равным "1".

Убывающее значение времени (как значение таймера) отсчитывается от некоторого заданного начального (initial) значения с заданным шагом, равным заданной величине "временной базы" (time base).

3 Функция таймера перестает быть активной, когда состояние сигнала на входе Start_input меняется от состояния "0" к состоянию "1" (положительный фронт), если это происходит до момента истечения заданного времени работы (длительности - "duration"). При этом таймер сбрасывается, и после этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0". Таймер не запускается до появления нового отрицательного фронта сигнала на входе Start_input.

**Сброс таймера с задержкой выключения (Resetting an off-delay timer)**

Операция сброса таймера с задержкой выключения (Resetting an off-delay timer) имеет статический эффект и имеет приоритет перед запуском таймера (см. рис. 7.11).

4 Состояние сигнала на входе Reset_input, равное "1", вызывает сброс функции таймера. После этого проверки (опросы) таймера на состояние "1" (timer status) возвращают результат проверки "0". Значение времени (time value) и значение "временной базы" (time base) также сбрасываются и становятся равными "0".

5 6 Если сигнал на входе Reset_input изменяет свое состояние с "0" на "1" (положительный фронт сигнала), в то время, как на входе Start_input присутствует состояние "1", то выход таймера сбрасывается (проверки [опросы] таймера на состояние "1" [timer status] после сброса выхода таймера, дают результат проверки, равный "0").

Отрицательный фронт сигнала (т.е. переход его от состояния "1" к "0") на входе Reset_input в то время, пока на входе Start_input присутствует состояние "1", возвращает выход таймера в состояние "1".

Если сигнал на входе Start_input изменяет свое состояние с "1" на "0" (отрицательный фронт сигнала), в то время, как на входе Reset_input присутствует состояние "1", то таймер запускается, но последующая инструкция сброса немедленно его сбрасывает (на рис. 7.11 это показано жирной вертикальной чертой). Если проверки (опросы) таймера на состояние "1" (timer status) следуют по времени после сброса таймера, то короткое время работы таймера после его запуска не скажется на результатах проверки - результат проверки будет равен "0".

**Разблокировка таймера с задержкой выключения (Anabling an off-delay timer)**

Функция разблокировки таймера (Anabling an off-delay timer) позволяет вновь запускать отсчет времени (или перезапускать сначала отсчет времени уже активного таймера) посредством приложения ко входу Enabling_input положительного фронта сигнала (переход сигнала от состояния "0" к состоянию "1").

Функция разблокировки таймера (Anabling an off-delay timer) доступна для использования только в языке программирования STL.

На рисунке 7.12 показана функция разблокировки таймера, запускаемого в режиме с задержкой выключения (Off-delay timer).

1 Если функция таймера неактивна (нет отсчета времени таймера), состояние сигнала на входе Start_input остается равным "1", а состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт), то на режим работы таймера ни этот положительный фронт, ни последующий отрицательный фронт сигнала на входе Enable_input не окажут никакого влияния.

2 Если состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт) в то время, когда функция таймера активна (запущен отсчет заданного времени таймера), отсчет времени таймера будет перезапущен сначала. При этом заданное значение длительности работы ("duration") будет вновь взято как исходное значение в момент перезапуска таймера.

**3** Если состояние сигнала на входе Enable_input меняется от состояния "0" к состоянию "1" (положительный фронт) или от состояния "1" к состоянию "0" (отрицательный фронт) в то время, когда функция таймера неактивна (нет отсчета времени таймера), то на режим работы таймера этот положительный или отрицательный фронт сигнала не окажут никакого влияния.

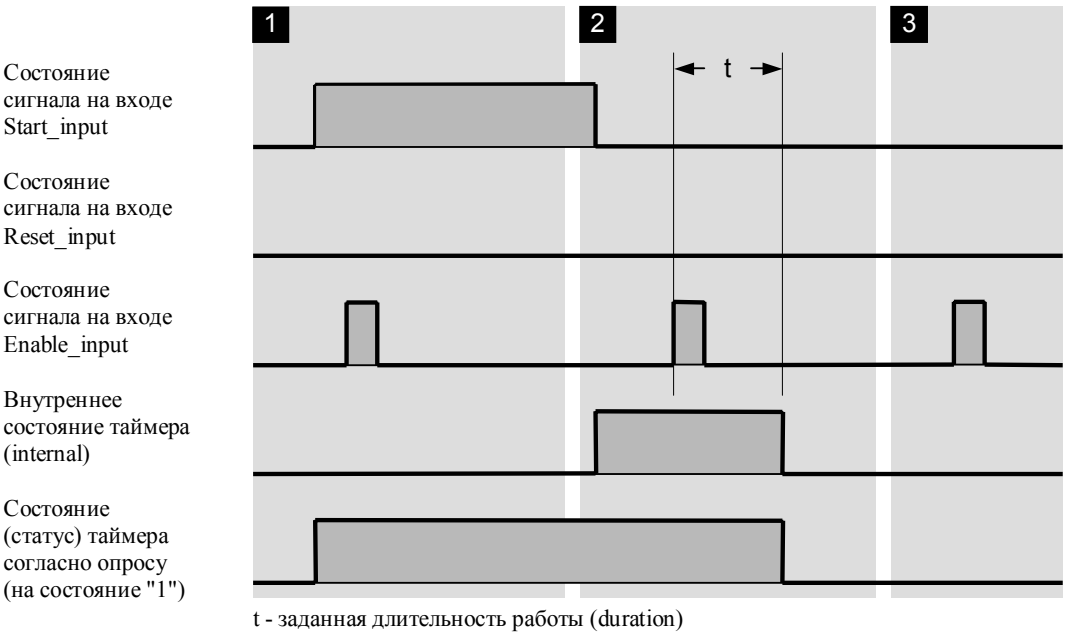



Рис. 7.12 Функция разблокировки (Enabling) таймера с задержкой выключения (Off-delay timer)

## 7.7 IEC-функции таймеров (IEC Timer Functions)

IEC-функции таймеров (IEC Timer Functions) встроены в операционную систему CPU как системные функциональные блоки SFB.

В соответствующим образом оснащенных CPU могут быть доступны следующие функции таймеров:

- SFB 3 TP

  Генератор импульсов
- SFB 4 TON

  Генерация импульса с задержкой включения
- SFB 5 TOF

  Генерация импульса с задержкой выключения

На рис. 7.13 представлены динамические характеристики этих таймеров.

Вы можете вызывать эти SFB с экземплярными блоками данных или использовать эти SFB как локльные экземпляры в функциональном блоке.

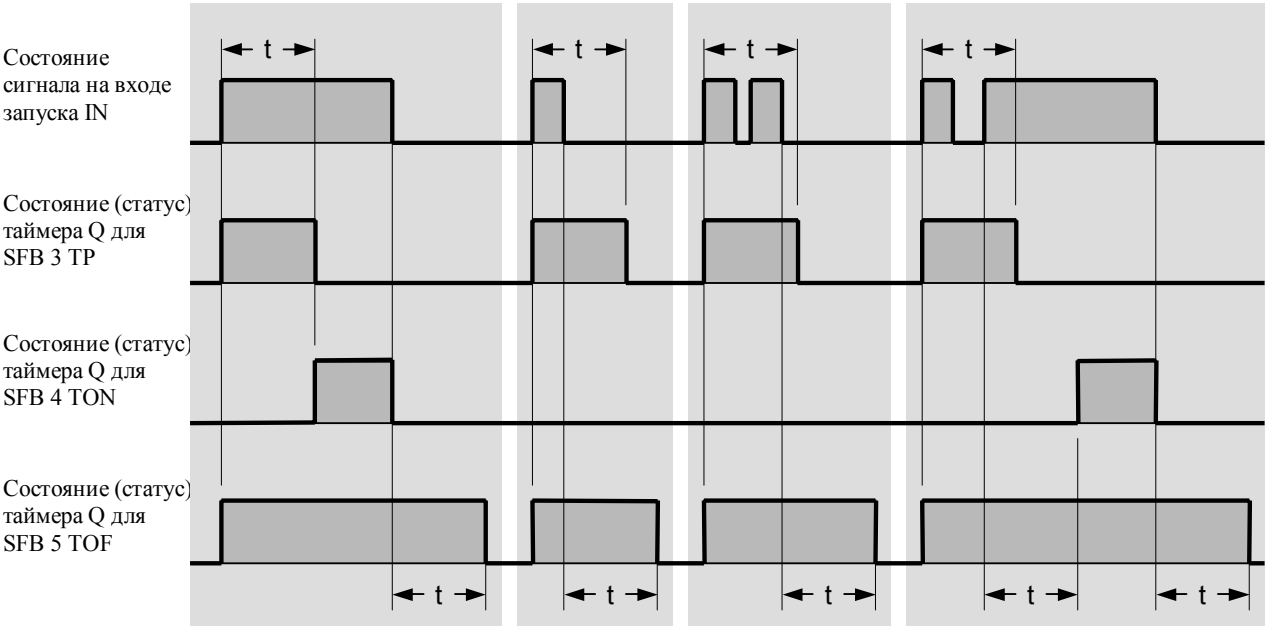



Рис. 7.13 Динамические характеристики IEC-функций таймеров

Вы можете найти описание интерфейса функций для программирования в автономном режиме (offline) в "стандартной библиотеке" *Standard Library* в *System Function Blocks*.

Примеры вызовов этих функций находятся на прилагаемой дискете в библиотеке STL_Book в программе "Basic Functions" в функциональном блоке FB 107 или в исходном файле Chap_7 или в библиотеке SCL_Book в программе "30 SCL Functions".

Таблица 7.2 Параметры IEC-функций таймеров

| Название (Name) | Объявление (Declaration) | Тип (Data type) | Описание (Description) |
|---|---|---|---|
| IN | INPUT | BOOL | Вход запуска (Start input) |
| PT | INPUT | TIME | Длина импульса (Pulse length) или продолжительность задержки включения (Delay duration) |
| Q | INPUT | BOOL | Состояние таймера (Timer status) |
| ET | INPUT | TIME | Истекшее время (Elapsed time) |

### 7.7.1 Генератор импульсов SFB 3 TP

Параметры IEC-таймера SFB 3 TP показаны в таблице 7.2.

Если RLO на входе запуска IN таймера изменяет свое состояние с "0" на "1", то функция таймера активизируется, т.е. запускается отсчет времени в течение заданного периода времени (duration), который не зависит от изменений состояния RLO на входе запуска. При этом выход Q возвращает сигнал "1" все то время, пока идет отсчет времени.

Выход ET возвращает длительность времени (duration), в течение которого выход Q находится в установленном состоянии. Этот период времени (duration) начинается в момент T#0s и заканчивается в заданное пользователем время PT. Если время PT истекло, то ET сохраняет это значение, пока состояние сигнала на входе IN не станет равным "0". Если состояние сигнала на входе IN станет равным "0" до того, как истекло заданное значение времени PT (duration), то на выходе ET значение изменится на T#0s сразу же после того, как закончится отсчет времени PT.

Если необходимо снова инициализировать функцию таймера, запустите ее с заданной величиной PT = T#0s.

IEC-таймер SFB 3 TP работает в рабочих режимах RESTART и RUN. IEC-таймер SFB 3 TP сбрасывается (инициализируется) при холодном перезапуске.

### 7.7.2 Генератор импульсов с задержкой включения SFB 4 TON

Параметры IEC-таймера с задержкой включения SFB 4 TON показаны в таблице 7.2

Если RLO на входе запуска IN таймера изменяет свое состояние с "0" на "1", то функция таймера активизируется, т.е. запускается отсчет времени в течение заданного периода времени (duration). При этом выход Q возвращает сигнал "1" после того, как отсчет времени завершился без досрочного прерывания. Если до истечения заданного периода времени (duration) RLO на входе запуска изменяет свое состояние с "1" на "0", то функция таймера сбрасывается. Отсчет времени запускается снова, если на входе запуска вновь появляется положительный фронт сигнала.

Выход ET возвращает длительность времени (duration), в течение которого функция таймера активна. Этот период времени (duration) начинается в момент T#0s и заканчивается в заданное пользователем время PT. Если время PT истекло, то ET сохраняет это значение, пока состояние сигнала на входе IN вновь не станет равным "0". Если состояние сигнала на входе IN станет равным "0" до того, как истекло заданное значение времени PT (duration), то на выходе ET значение изменится на T#0s немедленно.

Если необходимо снова инициализировать функцию таймера, запустите ее с заданной величиной PT = T#0s.

IEC-таймер SFB 4 TON работает в рабочих режимах RESTART и RUN. IEC-таймер SFB 4 TON сбрасывается (инициализируется) при холодном перезапуске.

### 7.7.3 Генератор импульсов с задержкой выключения SFB 5 TOF

Параметры IEC-таймера с задержкой выключения SFB 5 TOF показаны в таблице 7.2

Как только RLO на входе запуска IN таймера изменяет свое состояние с "0" на "1", при проверке выход таймера Q возвращает сигнал "1". После того как RLO на входе запуска изменяет свое состояние с "1" на "0", функция таймера активизируется, т.е. запускается отсчет времени в течение заданного периода времени (duration). При этом выход Q возвращает сигнал, равный "1", пока отсчет времени не завершится без досрочного прерывания. Если до истечения заданного периода времени (duration) RLO на входе запуска опять изменяет свое состояние с "0" на "1", то функция таймера сбрасывается, при этом выход Q сохраняет значение сигнала, равное "1". Отсчет времени запускается снова, если на входе запуска вновь появляется отрицательный фронт сигнала.

Выход ET возвращает длительность времени (duration), в течение которого функция таймера активна. Этот период времени (duration) начинается в момент T#0s и заканчивается в заданное пользователем время PT. Если время PT истекло, то ET сохраняет это значение, пока состояние сигнала на входе IN вновь не станет равным "1". Если состояние сигнала на входе IN станет равным "1" до того, как истекло заданное значение времени PT (duration), то на выходе ET значение изменится на T#0s немедленно.

Если необходимо снова инициализировать функцию таймера, запустите ее с заданной величиной PT = T#0s.

IEC-таймер SFB 5 TOF работает в рабочих режимах RESTART и RUN. IEC-таймер SFB 5 TOF сбрасывается (инициализируется) при холодном перезапуске.

# 8 Функции счетчиков (Counter Functions)

Функции счетчиков позволяют решать задачи счета непосредственно в CPU. Счетчики позволяют выполнять прямой и обратный счет и используют при этом "трехдекадный" формат значения счетчика и диапазон значений от 000 до 999.

В данной главе рассматриваются выражения, содержащие функции счетчиков для использования в языке программирования STL. Для языка SCL функции счетчиков включены в состав стандартных функций (см. разд. 30.2 "Функции счетчиков" ["Counter Functions"]).

Скорость счета счетчиков зависит от времени сканирования Вашей программы! Для обеспечения процесса счета CPU должен обнаруживать на входе счетчика изменения входного сигнала, иначе говоря, входной импульс (или междуимпульсная пауза) на входе должна присутствовать по крайней мере один цикл сканирования программы. Чем продолжительнее цикл сканирования программы, тем медленнее скорость счета счетчика.

Примечание:

В S7-300 CPU со встроенными функциями (CPU 3xxIFM) имеются встроенные функции счетчика, которые обеспечивают счет с использованием специального входа счетчика с частотой следования импульсов до 10 кГц.

Значения счетчиков, описанных в данной главе, сохраняются в системной памяти CPU. Вы можете задавать для счетчика начальное значение (initial value). Вы также можете сбрасывать счетчик, включать режим прямого или обратного счета счетчика. Существует возможность определения состояния счетчика (содержит ли счетчик нулевое или ненулевое значение). Функции загрузки (load) используются для пересылки текущего значения счетчика в двоичном или BCD-коде в аккумулятор accumulator 1.

Примеры, рассматриваемые в данной главе, и вызовы IEC-счетчиков Вы можете найти на прилагаемой дискете в библиотеке STL_Book library в разделе "Basic Functions" в функциональном блоке FB 108 или в исходном файле Chap_8.

## 8.1 Установка и сброс счетчиков

### Установка счетчика (Setting a counter)

S C n Установка счетчика

Счетчик устанавливается, если RLO переходит от "0" к "1" перед операцией установки S счетчика (Set). Т.е. для установки счетчика всегда требуется положительный фронт.

Установить счетчик - это значит загрузить в счетчик начальное значение.

Начальное значение, которое загружается в счетчик, должно находиться в аккумуляторе accumulator 1 (см. ниже). Оно должно быть в диапазоне от 0 до 999.


Рис. 8.1 Назначение битов для значения счетчика ("counter value")

### Спецификация счетчика (Specifying a counter)

Функция установки счетчика ("set counter") использует значение в аккумуляторе accumulator 1 как "значение счетчика" ("count value"). Как и когда это значение появляется в аккумуляторе accumulator 1 не имеет значения.

Для обеспечения лучшей читаемости программы наилучшим будет вариант, когда эти параметры загружаются в аккумулятор accumulator 1 непосредственно перед запуском функции счетчика или в виде константы (т.е., в виде непосредственно заданной величины), или в виде переменной (например, через слово в памяти, содержащее значение счетчика).

Примечание:

аккумулятор accumulator 1 должен содержать корректное значение, даже если счетчик не установлен во время выполнения инструкции.

*Определение (спецификация) счетчика с помощью константы*

```
L     C#100;              //Значение счетчика 100
L     W#16#0100;          //Значение счетчика 100
```

Значения счетчика лежат в диапазоне 000 … 999. При этом используются три декады. Значение счетчика может быть только в формате BCD. При этом счетчики не могут работать с отрицательными числами.

Для задания константы Вы можете использовать C# или W#16# (и только с десятичными числами).

*Определение (спецификация) счетчика с помощью переменной*

```
L     C#200;          //Значение счетчика 200
T     MW 56;          //Сохранить значение счетчика
…     ;
L     MW 56;          //Загрузить значение счетчика
```

Для выполнения операции Set предполагается наличие значения счетчика в аккумуляторе accumulator 1, состоящего из трех декад и расположенного в нем с выравниванием вправо. Назначение битов в счетчике (тип C#) подробно описано в главе 24 "Типы данных".

**Сброс счетчика (Resetting a counter)**

R C n Сброс счетчика

Счетчик сбрасывается, если RLO имеет значение "1" перед тем, как в программе встретится операция сброса R счетчика (Reset). Пока RLO равен "1", проверки счетчика на состояние "1" возвращают результат проверки "0"; проверки счетчика на состояние "0" возвращают результат проверки "1". Сброс устанавливает для значения счетчика ("count value") нулевое значение (0).

Примечание:

Сброс функции счетчика не сбрасывает внутренний меркер фронта для установки счетчика, для включения режима прямого счета и для включения режима обратного счета. Для повторного запуска CPU должен обработать инструкцию установки счетчика или повторного запуска на счет при RLO, равном "0", и только после этого при появлении фронта сигнала в соответствующем меркере фронта счетчик может быть установлен или запущен. Также для повторной активации функций счетчика Вы можете использовать операцию разблокировки счетчика.

## 8.2 Счет (Counting)

**Прямой счет (Counting up)**

Инструкция

CU C n вызывает процесс прямого счета.

Счетчик выполняет прямой счет ("инкрементируется"), если инструкция CU обрабатывается при положительном (возрастающем) фронте сигнала RLO (RLO меняет свое состояние с "0" на "1").

Каждый положительный фронт сигнала, предшествующий операции CU, увеличивает значение счетчика ("count value") на единицу, пока не будет достигнут верхний предел, равный 999. После этого положительный фронт сигнала RLO перед вводом CU никак не будет влиять на состояние счетчика.

**Обратный счет (Counting Down)**

Инструкция

| | | |
|---|---|---|
| CD | C n | вызывает процесс обратного счета. |

Счетчик выполняет обратный счет ("декрементируется"), если инструкция CD обрабатывается при положительном (возрастающем) фронте сигнала RLO (RLO меняет свое состояние с "0" на "1").

Каждый положительный фронт сигнала, предшествующий операции CD, уменьшает значение счетчика ("count value") на единицу, пока не будет достигнут нижний предел, равный 0. После этого положительный фронт сигнала RLO перед вводом CD никак не будет влиять на состояние счетчика.

Значения счетчика отрицательными быть не могут.

## 8.3 Проверка (опрос) счетчика (Checking a Counter)

**Проверка (опрос) состояния счетчика (binary counter check)**

| | | |
|---|---|---|
| A | C n | проверка сигнала на состояние "1" и комбинирование с RLO в соответствии с логикой AND (И). |
| O | C n | проверка сигнала на состояние "1" и комбинирование с RLO в соответствии с логикой OR (ИЛИ). |
| X | C n | проверка сигнала на состояние "1" и комбинирование с RLO в соответствии с логикой Exclusive OR (Исключающее ИЛИ). |
| AN | C n | проверка сигнала на состояние "0" и комбинирование с RLO в соответствии с логикой AND (И). |
| ON | C n | проверка сигнала на состояние "0" и комбинирование с RLO в соответствии с логикой OR (ИЛИ). |
| XN | C n | проверка сигнала на состояние "0" и комбинирование с RLO в соответствии с логикой Exclusive OR (Исключающее ИЛИ). |

Вы можете опрашивать счетчик, как если бы это был, например, вход (input), и в дальнейшем использовать результат этой проверки. Проверка (опрос) сигнала на состояние "1" вызывает результат "1", если значение счетчика больше 0 и результат, равный "0", если значение счетчика равно 0.

**Непосредственная загрузка значения счетчика (direct loading of a count value)**

Инструкция

L C n непосредственно загружает двоичное значение счетчика ("count value").

Функция загрузки L C пересылает определенное значение ("count value") счетчика, определенного в инструкции, в аккумулятор accumulator 1 в форме двоичного числа. Значение счетчика, пересылаемое в аккумулятор accumulator 1, представляет собой текущее значение, соответствующее моменту времени, когда производится опрос. Теперь значение в аккумуляторе accumulator 1 соответствует положительному числу целого типа (INT) и может использоваться для дальнейшей обработки, например, с помощью арифметических функций.

Пример:

```
L   C 99;     //загрузка текущего значения счетчика
T   MW 76;    //сохранение текущего значения счетчика
```

**Загрузка значения счетчика в BCD-формате ("coded loading")**

Инструкция

LD C n загружает двоично-десятичное ("кодированное") значение счетчика ("count value").

Вы можете также использовать инструкцию для т.н. "кодированной" пересылки ("coded load") в аккумулятор accumulator 1 в формате двоично-десятичного числа значения счетчика, определенного в инструкции. Значение счетчика, пересылаемое в аккумулятор accumulator 1, представляет собой текущее значение, соответствующее моменту времени, когда производится опрос. Содержимое аккумулятора accumulator 1 доступно для дальнейшего использования в виде числа в формате BCD-числа, выровненного вправо. Оно имеет такую же структуру как и заданное значение счетчика.

Пример:

```
LD  C 99;     //загрузка текущего значения счетчика в BCD-формате
T   MW 50;    //сохранение текущего значения счетчика
```

## 8.4 Разблокировка счетчика (Enabling a counter)

Инструкция

FR C n позволяет выполнить переустановку (перезапуск) счетчика.

При использовании инструкции FR Вы можете установить (Set) счетчик или запустить на счет без предшествующего операции положительного фронта сигнала RLO. Тем не менее, выполнение этих операций со счетчиком будет возможно только, пока RLO имеет значение "1".

Функция разблокировки активна, если перед тем, как она встретится, RLO переходит от состояния "0" к состоянию "1". Положительный фронт сигнала RLO - это всегда необходимое условие для выполнения разблокировки счетчика.

Данная инструкция разблокировки функции счетчика не требуется для установки, запуска на счет или сброса счетчика (т.е., инструкция не является необходимой для обычных условий работы со счетчиком).

Примечание:

Инструкция разблокировки функций счетчика воздействует на функции установки, запуска на счет и сброса счетчика одновременно! Положительный фронт сигнала RLO перед выполнением разблокировки счетчика вызывает все последующие инструкции (S, CU и CD), которые имеют сигнал запуска "1".

Ниже представлен пример, показывающий принципы работы инструкции разблокировки (соответствующие диаграммы показаны на рис. 8.2):

```
A       "Enable";
FR      "Counter";
A       "Count up";
CU      "Counter";
A       "Count down";
CD      "Counter";
A       "Set";
L       C#020;
S       "Counter";
A       "Reset";
R       "Counter";
A       "Counter";
=       "Counter status";
```

Ниже рассмотрены фазы соответствующих диаграмм, показанных на рис. 8.2:

1 Положительный фронт сигнала на входе установки счетчика (Set) устанавливает счетчик на начальное значение 20.

2 Положительный фронт сигнала на входе CU инкрементирует счетчик (его значение увеличивается на 1).

3 Так как сигнал на входе Set имеет значение "1", инструкция разблокирования счетчика инкрементирует счетчик - его значение увеличивается на 1.

4 Положительный фронт сигнала на входе сброса (Reset) декрементирует счетчик (его значение уменьшается на 1).

5 Инструкция разблокирования счетчика создает условий для выполнения прямого и обратного счета: сигнал "1" присутствует на обоих входах.

6 Положительный фронт сигнала на входе установки счетчика (Set) устанавливает счетчик на начальное значение 20.

7 Сигнал на входе Reset, имеющий значение "1", сбрасывает счетчик. Проверка его на состояние "1" возвращает результат "0".

8 Так как состояние сигнала на входе Set все еще равно "1", инструкция разблокирования устанавливает счетчик с начальным значением 20. Проверка счетчика на состояние "1" возвращает результат "1".

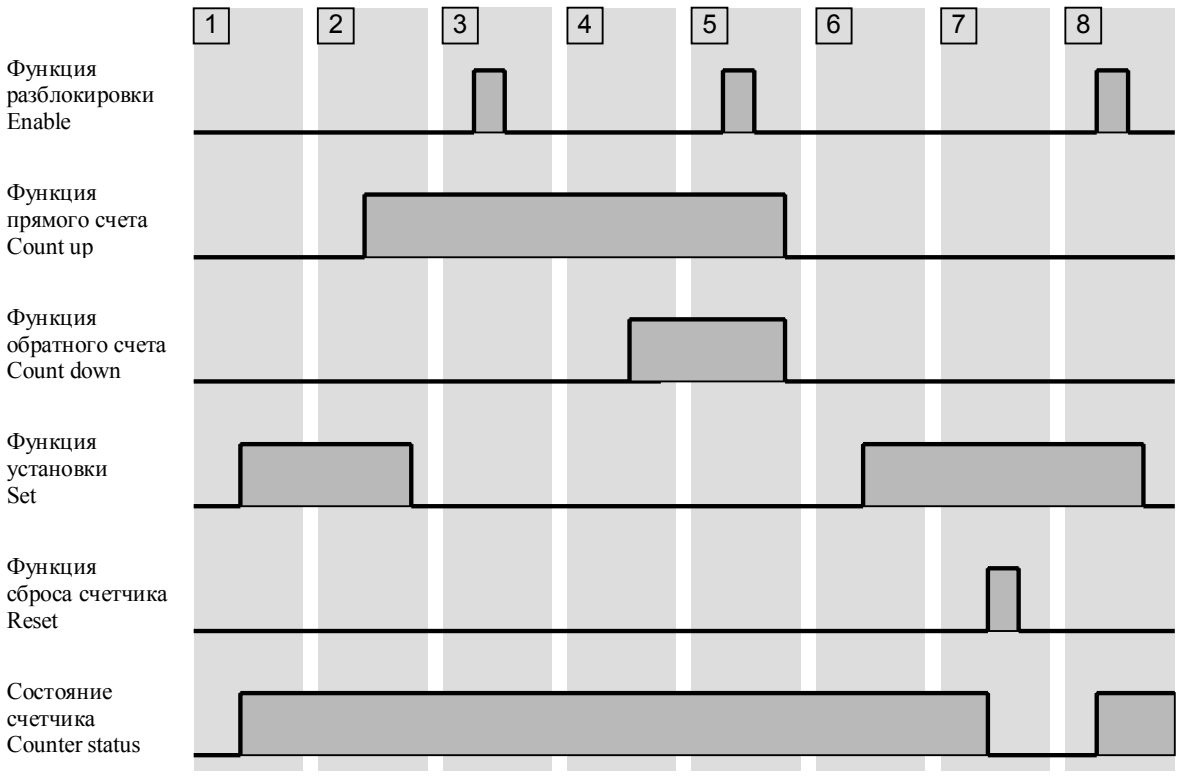



Рис. 8.2 Принцип работы инструкции разблокировки Enable

## 8.5 Последовательность инструкций при использовании функций счетчика

При программировании счетчика Вам нет необходимости использовать все возможные выражения для активации функций счетчиков. Вы должны использовать только те из функций, которые Вам необходимо выполнить. Например, для выполнения функции обратного счета обычно используются операция установки для счетчика заданного значения (initial value), операция обратного счета и двоичный опрос счетчика на состояние "0".

Чтобы выполнить функцию счетчика в соответствии с описанием в предыдущих разделах необходимо соблюдать определенный порядок при программировании соответствующих операторов.

В таблице 8.1 показан оптимальный порядок для всех операторов при программировании функций счетчика. Вы можете просто пропустить ненужные операторы, когда Вы будете записывать программу, ориентируясь на рекомендуемую последовательность операторов, например, пропустите функцию разблокировки Enable.

Таблица 8.1 Последовательность операторов для счетчика

| Функция счетчика: | Примеры: |
|---|---|
| Разблокировка счетчика (Enable counter) | `A  I  22.0`<br>`FR C  17` |
| Функция прямого счета (Count up) | `A  I  22.1`<br>`CU C  17` |
| Функция обратного счета (Count down) | `A  I  22.2`<br>`CD C  17` |
| Установка счетчика (Set counter) | `A  I  22.3`<br>`L  C#500`<br>`S  C  17` |
| Сброс счетчика (Reset counter) | `A  I  22.4`<br>`R  C  17` |
| Проверка численного значения счетчика (Digital counter check) | `L  C  17`<br>`T  MW 30`<br>`LC C  17`<br>`T  MW 32` |
| Двоичный опрос счетчика (Binary counter check) | `A  C  17`<br>`=  Q  13.0` |

Если функция сброса счетчика Reset должна иметь статическое ("static") влияние на операции CU, CD и S и не должна зависеть от результата логической операции (RLO), то Вы должны записать выражение с операцией Reset для счетчика после выражений с вышеуказанными операциями, но перед операцией проверки счетчика.

Если при этом счетчик устанавливается (set) и сбрасывается (reset) "одновременно", то счетчик сначала получит заданное значение, а затем немедленно будет сброшен операцией Reset. Таким образом, последующая проверка счетчика не позволит установить тот факт, что счетчик кратковременно находился в установленном состоянии.

Если функция установки счетчика Set должна иметь статическое ("static") влияние на операции счета и не должна зависеть от результата логической операции (RLO), то Вы должны записать выражение с операцией Set после выражений с операциями счета.

Если при этом счетчик устанавливается (set) и сбрасывается (reset) "одновременно", то операции счета вначале будут изменять состояние счетчика, а затем счетчик немедленно будет установлен операцией Set и получит заданное значение, которое и сохранится в нем до окончания сканирования программы.

Таким образом, последовательность выражений с операциями прямого и обратного счета не будет иметь влияние на счетчик.

## 8.6 IEC-функции счетчиков (IEC Counter Functions)

IEC-функции счетчиков (IEC Counter Functions) встроены в операционную систему CPU как системные функциональные блоки SFB.

В соответствующим образом оснащенных CPU могут быть доступны следующие функции счетчиков:

- SFB 0 CTU

  Функция прямого счета
- SFB 1 CTD

  Функция обратного счета
- SFB 2 CTUD

  Функция прямого и обратного счета

Вы можете вызывать эти SFB с экземплярными блоками данных или использовать эти SFB как локальные экземпляры в функциональном блоке.

Вы можете найти описание интерфейса функций для программирования в автономном режиме (offline) в "стандартной библиотеке" *Standard Library* в разделе *System Function Blocks.*

Примеры вызовов этих функций находятся на прилагаемой дискете в библиотеке STL_Book в разделе "Basic Functions" в функциональном блоке FB 108 или в исходном файле Chap_8 или в библиотеке SCL_Book в разделе "30 SCL Functions".

Параметры вышеуказанных IEC-функций счетчиков предлагаются Вашему вниманию в таблице 8.2.

Таблица 8.2 Параметры IEC-функций счетчиков

| Название (Name) | Представление в SFB | | | Объявление (Declaration) | Тип (Data type) | Описание (Description) |
|---|---|---|---|---|---|---|
| CU | 0 | - | 2 | INPUT | BOOL | Вход прямого счета (Up count input) |
| CD | - | 1 | 2 | INPUT | BOOL | Вход обратного счета (Down count input) |
| R | 0 | - | 2 | INPUT | BOOL | Вход сброса счетчика (Reset input) |
| LOAD | - | 1 | 2 | INPUT | BOOL | Вход загрузки (Load input) |
| PV | 0 | 1 | 2 | INPUT | INT | Заданное значение счетчика (Preset value) |
| Q | 0 | 1 | - | OUTPUT | BOOL | Состояние счетчика (Counter status) |
| QU | - | - | 2 | OUTPUT | BOOL | Состояние счетчика в режиме прямого счета (Count counter status up) |
| QD | - | - | 2 | OUTPUT | BOOL | Состояние счетчика в режиме обратного счета (Count counter status down) |
| CV | 0 | | 2 | OUTPUT | INT | Текущее значение счетчика (Current counter value) |

### 8.6.1 Функция прямого счета SFB 0 CTU

Параметры IEC-счетчика SFB 0 CTU показаны в таблице 8.2.

Если сигнал на входе прямого счета счетчика (up counter input) CU изменяет свое состояние с "0" на "1" ("прямой фронт"), тогда текущее значение счетчика инкрементируется (увеличивается на единицу) и отображается на выходе CV. При первом вызове (при состоянии сигнала "0" на входе сброса R) значение счетчика соответствует заранее заданному значению на входе PV (Preset value).

Если текущее значение достигает верхнего предела, равного 32767, значение счетчика больше не увеличивается. При этом сигнал на входе CU игнорируется.

Если сигнал на входе сброса R принимает значение "1", то счетчик сбрасывается в 0. При этом положительный фронт на входе счетчика CU игнорируется, пока состояние сигнала на входе сброса R равно "1". На выход Q счетчика будет выводиться значение "1", если значение на выходе CV будет больше или равно заранее заданного значения счетчика на входе PV.

IEC-счетчик SFB 0 CTU работает в рабочих режимах RESTART и RUN. IEC-счетчик SFB 0 CTU сбрасывается при холодном перезапуске.

### 8.6.2 Функция обратного счета SFB 1 CTD

Параметры IEC-счетчика SFB 1 CTD показаны в таблице 8.2.

Если сигнал на входе обратного счета счетчика (down counter input) CD изменяет свое состояние с "0" на "1" ("прямой фронт"), тогда текущее значение счетчика декрементируется (уменьшается на единицу) и отображается на выходе CV. При первом вызове (при состоянии сигнала "0" на входе LOAD) значение счетчика соответствует заранее заданному значению на входе PV (Preset value).

Если текущее значение достигает нижнего предела, равного -32768, значение счетчика далее не уменьшается. При этом сигнал на входе CD игнорируется.

Если сигнал на входе LOAD имеет значение "1", то счетчик сбрасывается и принимает заранее заданное значение (на входе PV). При этом положительный фронт на входе счетчика CD игнорируется, пока состояние сигнала на входе LOAD равно "1". На выход Q счетчика будет выводиться значение "1", если значение на выходе CV будет меньше или равно нулю.

IEC-счетчик SFB 1 CTD работает в рабочих режимах RESTART и RUN. IEC-счетчик SFB 1 CTD сбрасывается при холодном перезапуске.

### 8.6.3 Функция прямого и обратного счета SFB 2 CTUD

Параметры IEC-счетчика SFB 2 CTUD показаны в таблице 8.2.

Если сигнал на входе прямого счета счетчика (up counter input) CU изменяет свое состояние с "0" на "1" ("прямой фронт"), тогда текущее значение счетчика инкрементируется (увеличивается на единицу) и отображается на выходе CV. Если сигнал на входе обратного счета счетчика (down counter input) CD изменяет свое состояние с "0" на "1" ("прямой фронт"), тогда текущее значение счетчика декрементируется (уменьшается на единицу) и отображается на выходе CV. Если сигнал на обоих входах счетчика CU и CD изменяет свое состояние с "0" на "1" ("прямой фронт"), тогда текущее значение счетчика не изменяется.

Если текущее значение достигает верхнего предела, равного 32767, значение счетчика больше не увеличивается. При этом в дальнейшем сигнал на входе CU игнорируется.

Если текущее значение достигает нижнего предела, равного -32768, значение счетчика далее не уменьшается. При этом в дальнейшем сигнал на входе CD игнорируется.

Если сигнал на входе LOAD имеет значение "1", то счетчик сбрасывается и принимает заранее заданное значение (на входе PV). При этом положительные фронты сигналов на входах счетчика игнорируются, пока

состояние сигнала на входе LOAD равно "1".

Если состояние сигнала на входе сброса R принимает значение "1", то счетчик сбрасывается в 0. При этом положительные фронты сигналов на входах счетчика и состояние сигнала "1" на входе LOAD игнорируются, пока состояние сигнала на входе сброса R равно "1". На выход QU счетчика будет выводиться значение "1", если значение на выходе CV будет больше или равно значению на входе PV. На выход QD счетчика будет выводиться значение "1", если значение на выходе CV будет меньше или равно нулю.

IEC-счетчик SFB 2 CTUD работает в рабочих режимах RESTART и RUN. IEC-счетчик SFB 2 CTUD сбрасывается при холодном перезапуске.

## 8.7 Пример счетчика деталей

В данном разделе представлен пример, с помощью которого иллюстрируется работа с таймерами и счетчиками. В этом примере запрограммированы входы, выходы и меркеры, так что данная программа может быть включена в любое место любого блока. Пример выполнен как функция без параметров.

**Описание функций**

Детали должны переноситься лентой конвейера. Для обнаружения и подсчета деталей используется фотодатчик. После того, как подсчитанное число деталей становится равным заданному максимальному количеству, счетчик посылает сигнал окончания работы "Finished". Счетчик снабжен цепью слежения. Если состояние сигнала от фотодатчика не меняется в течение заданного времени, эта цепь слежения генерирует соответствующий сигнал.

Вход "Set" обеспечивает передачу счетчику начального значения (число деталей, которое должно быть сосчитано). Положительный фронт сигнала от фотодатчика вызывает уменьшение на единицу значения счетчика. Когда значение счетчика достигнет значения 0, счетчикает посылает сигнал окончания работы "Finished". Должно выполняться условие, согласно которому детали на ленте конвейера лежат отдельно (с интервалом между отдельными деталями).

Вход "Set" обеспечивает также установку сигнала "Active". Контроллер отслеживает изменение состояния сигнала, поступающего от фотодатчика, только во время установления сигнала "Active". Сигнал "Active" сбрасывается при завершении счета, когда последняя деталь минует фотодатчик.

В активном состоянии положительный фронт сигнала от фотодатчика запускает таймер со значением времени "Dura1" в режиме таймера с памятью. Если на входе Start таймера "0" в следующем цикле сканирования таймер в дальнейшем не продолжает отсчет времени. Новый положительный фронт сигнала от фотодатчика перезапускает таймер. Следующий положительный фронт сигнала от фотодатчика, перезапускающий таймер, генерируется, после того как фотодатчик выдаст отрицательный фронт сигнала. Тогда таймер запустится со значением времени "Dura2".

Если фотодатчик обнаружит деталь через промежуток времени, больший чем значение "Dura1" или будет свободен ("free") в течение промежутка времени, большего, чем значение "Dura2", то время заканчивается и таймер сигнализирует о сбое - выдает сигнал "Fault".

Первый сигнал "Active" запускает таймер со значением времени "Dura2". Сигнал "Set" запускает счетчик со схемой слежения. Положительные и отрицательные фронты сигнала, поступающие от фодатчика, используются для управления счетчиком, для выбора промежутка времени и для запуска (перезапуска) таймера (watchdog timer).

Проверка наличия положительного и отрицательного фронта сигнала требуется часто, и, поэтому, в качестве "сверхоперативной памяти" можно использовать область временных локальных данных для запоминания результата проверки. Временные локальные данные - это "внутриблочные" переменные (переменные, объявленные в блоке, а не в таблице символов). В данном примере результаты проверки хранятся в меркерах импульса ("pulse memory bits") в области временных локальных даных. Сигналы от меркеров фронта ("edge memory bits") требуются также в последующих циклах сканирования программы, поэтому эти меркеры не должны располагаться в области временных локальных даных.

Данная программа-пример выполнен как функция без параметров. Вы можете вызывать эту функцию (например, из OB 1) следующим образом:

```
Call "Counter_control";
```

В конце данной главы представлен исходный текст программы-примера с символьной адресацией таймеров, счетчиков и меркеров.

Глобальные символы могут также использоваться без кавычек (без апострофа), если они не содержат специальных символов. Если же символ (символьное имя) содержит специальный символ (например, умляут или пробел [space]), то такое имя должно быть заключено в кавычки. В компилированных блоках редактор STL отображает все глобальные символы в кавычках.

Для большей ясности и лучшей читаемости представленная программа разделена на сегменты. Последний сегмент, имеющий заголовок BLOCK END (конец блока), не является необходимым, а служит лишь для обозначения окончания блока. Такой прием бывает очень полезно использовать в случаях, когда блоки имеют чрезмерно большой размер.

Вы можете найти таблицу символов (symbol table) на прилагаемой к данной книге дискете в библиотеке STL_Book в разделе "Conveyor Example" в объекте *Symbol*, в исходной программе "Conveyor" в разделе исходных файлов *Source Files* и в скомпилированной программе в разделе *Blocks,* в функции FC 12.

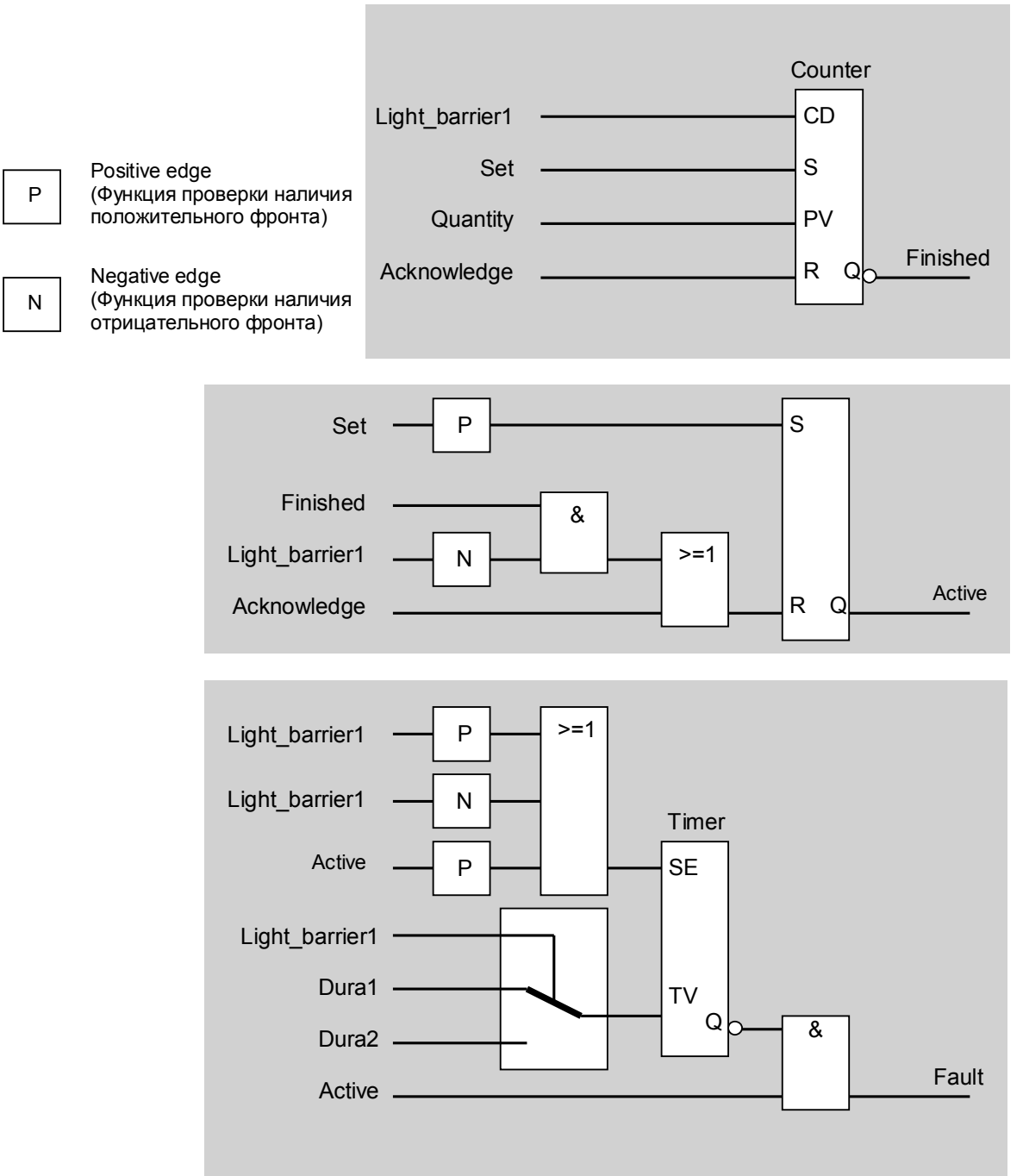



Рис. 8.3 Пример счетчика деталей

```
FUNCTION  "Counter_control" : VOID
//TITLE = Счетчик деталей с системой слежения
//Данный пример иллюстрирует работу таймеров и счетчиков
NAME    : Count
AUTHOR  : Berger
FAMILY  : STL_Book
Version : 01.00
VAR_TEMP
    PM_LB_P : BOOL;        //Положительный фронт импульса от фотодатчика
    PM_LB_N : BOOL;        //Отрицательный фронт импульса от фотодатчика
END_VAR
BEGIN
NETWORK
TITLE = Counter_Control
    A   Light_barnerl;      //При срабатывании фотодатчика
    CD  Count;              //декрементировать счетчик на 1
    A   Set;
    L   Quantity;           //Загрузить в счетчик заданное количество
//                            "Quantity"
    S   Count;
    A   Acknowledge;
    R   Count;
    AN  Count;              //Когда значение счетчика достигнет 0,
    =   Finished;           //генерируется сигнал окончания работы
NETWORK
TITLE = Activate monitor
    A   Light_barrierl;
    FP  EM_LB_P;            //Генерировать меркер импульса
    =   PM_LB_P;            //при положительном фронте сигнала от датчика
    A   Light_barrierl;
    FN  EM_LB_N;            //Генерировать меркер импульса
    =   PM_LB_N;            //при отрицательном фронте сигнала от датчика
    A   Set;
    FP  EM_ST_P;
    S   Active;             //Активировать систему слежения
    A   Finished;
    A   PM_LB_N;
    0   Acknowledge;
    R   Active;             //Деактивировать систему слежения
NETWORK
TITLE = Monitoring circuit
    L   Dura1;              //Если фотодатчик в состоянии "1",
    A   Light_barrierl;     //то выполняется переход JC к Dl и
    JC  D1;                 //аккумулятор получает значение "Dura1",
    L   Dura2;              //иначе - значение "Dura2"
Dl: A   Active;
    FP  EM_AC_P;            //Если положительный фронт в "Active",
    0   PM_LB_P;            //или положительный фронт от датчика,
    0   PM_LB_N;            //или отрицательный фронт от датчика, то
    SE  Monitor;            //таймер запускается или перезапускается
    AN  Monitor;
    A   Active;             //Если время заканчивается во время
//                            действия сигнала "Active",
    =   Fault;              //то генерируется сигнал сбоя "Fault"
NETWORK
TITLE = Block End
    BE;
END_FUNCTION
```