Управление выполнением программы

STEP 7 обеспечивает пользователя различными средствами для управления ходом выполнения программы. Вы можете выходить из выполняемой линейной программы внутри блока, Вы можете также создать определенную структуру программы посредством вызовов параметрируемых блоков. Вы можете воздействовать на выполнение программы, исходя из текущих значений параметров, рассчитанных во время работы программы, в зависимости параметров процесса или в связи с состоянием установки.

Биты состояния (status bits) призваны обеспечивать информацией, зависящей от результатов выполнения арифметических и математических функций, а также от возникающих ошибок (например, из-за нарушения границ диапазона допустимых значений при расчетах). Вы можете использовать состояния сигналов этих битов в Вашей программе непосредственно, проверяя их с помощью двоичных логических операций.

Функции перехода (jump functions) могут использоваться для организации ветвления программы пользователя. Функции перехода могут быть либо безусловные, либо условные, (по условию определяемому отдельным битом состояния, результатом логической операции RLO или двоичным результатом). В STL можно легко запрограммировать переходы в рассчетные точки программы (распределитель переходов) или выполнить программные циклы (циклический переход).

Главное управляющее реле (Master Control Relay - MCR) обеспечивает еще один способ управления выполнением программы. Изначально разработанный для систем управления с помощью реле, язык STL обеспечивает программную (компьютерную) версию программируемого способа управления.

Функции блоков (block functions) позволяют пользователю структурировать программу. Вы можете снова и снова использовать функции и функциональные блоки, определяя **параметры блоков**.

Более подробную информацию о программировании блоков на STL Вы можете найти в разделе 3.4 "Программирование кодовых блоков на STL". Главы 18 "Функции блоков" и 19 "Параметры блоков" продолжают эту тему. Информацию, касающуюся этих же вопросов для языка SCL, Вы можете отыскать в разделе 3.5 "Программирование кодовых блоков на SCL" и в главе 29 "SCL блоки". Глава 26 "Прямой доступ к переменным" содержит дополнительную информацию по параметрам блоков, такую, например, как сохранение параметров в памяти и использование параметров для сложных типов данных.

15 Биты состояния (Status bits)

Биты состояния RLO, BR, CC0, CC1 и превышения (overflow); проверка битов состояния; слово состояния (status word); EN/ENO.

16 Функции перехода (Jump functions)

Безусловные переходы; переходы по условию для RLO, BR, CC0, CC1 и превышения (overflow); распределитель переходов ("jump distributor"); циклический переход (loop jump).

17 Главное управляющее реле (Master Control Relay - MCR)

Зависимость от MCR; MCR-диапазон; MCR-зона.

18 Функции блоков

Тип блоков, вызовы блоков, окончания блоков; статические локальные данные; обработка блоков данных, регистр блока данных, обработка адресов данных.

19 Параметры блоков

Объявление параметров; формальные параметры и фактические параметры; передача параметров в вызываемые блоки; примеры: ленточный конвейер, счетчик деталей и загрузка.

15 Биты состояния (Status Bits)

Биты состояния - это двоичные флаги (индикаторные биты). CPU использует их для управления двоичными логическими операциями. CPU устанавливает биты состояния при выполнении операций обработки чисел. Вы можете проверять состояние этих битов (например, для выяснения результата выполнения вычислений); Вы можете также влиять на отдельные биты. Биты состояния скомпонованы в слово, в "слово состояния" ("status word").

Примеры, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book library в разделе "Program Flow Control" ("Управление выполнением программы") в функциональном блоке FB 115 или в исходном файле Chap_15.

15.1 Описание битов состояния

В таблице 15.1 представлены биты состояния, доступные при STL-программировании.

Таблица 15.1 Биты состояния

Бит	**Двоичные флаги (binary flags)**	
0	/FC	Первичный опрос (first check)
1	RLO	Результат логической операции
2	STA	Состояние (статус - "status")
3	OR	Бит состояния OR (OR status bit)
8	BR	Двоичный результат (binary result)
Бит	**Числовые флаги (digital flags)**	
4	OS	Для сохранения информации о переполнении (stored overflow)
5	OV	Переполнение (overflow)
6	CC0	Условный код (condition code)
7	CC1	Условный код (condition code)

Первый столбец показывает номер бита в слове состояния. CPU использует "двоичные флаги" ("binary flags") для управления двоичными функциями; "числовые флаги" ("digital flags") используются прежде всего для индикации результатов арифметических и математических функций.

Первичный опрос (first check)

Бит состояния /FC управляет двоичными логическими операциями внутри логического устройства управления (часть АЛУ). Двоичный логический шаг всегда начинается с двоичной инструкции проверки (первичного опроса) при /FC = "0", как показано в описании двоичных логических операций. Первичный опрос устанавливает /FC в состояние "1" (/FC = "1"). Двоичный логический шаг заканчивается присвоение двоичного значения, или условным переходом, или изменением блока. Это сбрасывает бит состояния: /FC = "0". Следующая двоичная проверка (опрос) начинается с новой двоичной логической функции.

Результат логической операции (RLO)

Бит состояния RLO является промежуточным буфером в двоичных логических операциях. При первичном опросе CPU передает результат опроса в RLO, комбинирует результат опроса с хранящимся в RLO значением при каждом последующем опросе и затем сохраняет результат в RLO (как описано в главе 4 "Двоичные логические операции"). Вы можете также устанавливать, сбрасывать или инвертировать значение в RLO непосредственно или сохранять его в BR. Функции счетчика, таймера и операции с памятью управляются с использованием RLO, как и функции перехода.

Состояние (status)

Состояние бита STA соответствует состоянию сигнала указанного двоичного разряда (двоичного адреса) или проверяемого "условного бита" для двоичной логической операции (A, AN, O, ON, X, XN).

В случае операций с памятью (S, R, =) значение бита состояния STA соответствует записанному в память значению или (в случае отсутствия операции записи в память, например, при RLO = "0" или когда главное управляющее реле MCR активно) значение бита STA соответствует значению адресованного (но неизмененного) двоичного адреса.

При проверке наличия фронта сигнала FP или FN значение RLO до операции проверки фронта сохраняется в бите состояния STA. Все остальные двоичные операции устанавливают STA (STA = "1"), как и переходы, зависящие от состояния двоичного флага: JC, JCN, JBI, JNBI (исключение: CLR сбрасывает бит STA: STA = "0").

Бит состояния STA не влияет на обработку операторов STL-операторов. Состояние этого бита отображается тестовыми функциями с помощью программатора PG (такой функцией, например, как функция отображения состояния программы "program status"). Таким образом, Вы можете использовать этот бит состояния для трассировки двоичных логических последовательностей или в целях отладки программы.

Бит состояния OR (OR status bit)

Бит OR status сохраняет результат выполненной (имеется в виду, что условие функции выполнено) двоичной логической операции AND (И) ("1")

и показывает последующей операции OR (ИЛИ), что результат уже зафиксирован (при комбинированной операции OR (ИЛИ) внутри операции AND (И) перед выполнением операции OR (ИЛИ)). Все остальные двоичные операторы сбрасывают бит OR status.

В таблице 15.2 в графах "Двоичные флаги (Binary flags)" приводится пример двоичного логического шага для иллюстрации действия двоичных флагов. Двоичный логический шаг начинается с первичного опроса после операции с памятью и заканчивается с последней операцией с памятью до функции проверки.

Таблица 15.2 Пример влияния битов состояния

STL-операторы	Двоичные флаги (binary flags)				Примечания
	/FC	RLO	STA	OR	
...					
= M 10.0	0	x	x		
A I 4.0	1	1	1	0	I 4.0 сод. "1" Окрашенная
A I 4.1	1	1	0	0	I 4.1 сод. "0" область
O	1	1	1	1	соответствует
O I 4.2	1	1	0	0	I 4.2 сод. "0" двоичному
ON I 4.3	1	1	1	0	I 4.3 сод. "1" логическому
= Q 8.0	0	1	1	0	Q 8.0 --> "1" шагу
R Q 8.1	0	1	0	0	Q 8.1 --> "0"
S Q 8.2	0	1	1	0	Q 8.2 --> "1"
A I 5.0	1	x	x		
...					

STL-операторы	Числовые флаги (digital flags)				Примечания
	CC0	CC1	OV	OS	
...					
T MW 10	x	x	x	x	
L +12	x	x	x	x	
L +15	x	x	x	x	
-I	1	0	0	0	результат отрицательный
L +20000	1	0	0	0	
*I	0	1	1	1	переполнение: OV и OS -> "1"
L +20	0	1	1	1	
+I	0	1	0	1	OV --> "0" OS без изм.="1"
T MW 30	0	1	0	1	
L MW 40	1	1	0	1	
...					

Переполнение (Overflow)

Бит состояния OV индицирует факт превышения диапазона допустимых численных значений (переполнения) или факт использования некорректных действительных (REAL) чисел. На состояние бита OV влияют следующие функции: арифметические, математические, отдельные функции преобразования, функции сравнения действительных (REAL) чисел.

Вы можете проверить состояние бита OV с помощью операций проверки или посредством оператора перехода JO.

Сохранение информации о превышении граничных значений (Stored overflow)

Бит состояния OS фиксирует факт превышения (дублирует и сохраняет установленное состояние бита OV). Когда CPU устанавливает бит состояния OV, он также устанавливает бит состояния OS. При этом, если бит состояния OV в дальнейшем может быть сброшен при выполнении соответствующей операции, то бит OS сохранит свое состояние, т.е. информацию о состоявшемся факте превышения. Это позволит Вам выполнять альтернативную проверку бита состояния, в том числе и отложенную во времени (в более поздней точке программы), для определения факта превышения диапазона допустимых численных значений или факта использования некорректных действительных (REAL) чисел.

Вы можете проверять состояние бита OS посредством операторов проверки (опроса) или с помощью оператора перехода JOS. Оператор перехода JOS или смена блока сбрасывает бит состояния OS.

Биты состояния CC0, CC1 (биты "условных кодов") (condition codes)

Биты состояния CC0, CC1 (условные биты) призваны обеспечивать информацией о результатах выполнения функций сравнения, арифметических и математических функций, логических операций для слов данных или о состоянии "вытолкнутых" битов в случае выполнения операций сдвига.

Вы можете проверять состояние всех числовых флагов с помощью операторов перехода или посредством операторов проверки (опроса) (см. далее в этой главе). В нижней части таблицы 15.2 показаны примеры установки числовых флагов (digital flags).

Двоичный результат (binary result)

Бит состояния BR (двоичный результат) помогает реализовать механизм EN/ENO для вызовов блоков (в сочетании с графическими языками). В разделе 15.4 "Использование двоичного результата" показано, как STEP 7 использует бит двоичный результат. Вы можете также установить или сбросить бит состояния BR или проверить его состояние с помощью операторов проверки (опроса) или посредством операторов перехода.).

Слово состояния (status word)

Слово состояния (status word) содержит в себе все биты состояния. Вы можете загружать слово состояния в аккумулятор accumulator 1, а также считать его значение из этого аккумулятора.

```
L    STW;    //загрузить слово состояния
T    STW;    //загрузить в слово состояния
```

См. в главе 6 "Функции пересылки данных" о том, как используются операторы загрузки (load) и выгрузки (transfer). В таблице 15.1 представлены биты состояния в слове состояния с указанием их назначения.

Вы можете использовать слово состояния для проверки битов состояния или для их установки в соответствии с Вашими требованиями. Таким образом, Вы можете сохранить текущее значение слова состояния или начать выполнение программного блока с требуемыми значениями битов состояния.

Надо учитывать, что S7-300 CPU (кроме CPU 318) не загружают биты состояния /FC, STA и OR в аккумулятор; аккумулятор в соответствующих этим битам позициях содержит "0".

15.2 Описание битов состояния

Функции обработки чисел влияют на биты состояния CC0, CC1, OV и OS, как показано в таблице 15.3. Отдельные операторы STL влияют на биты состояния RLO и BR.

Биты состояния при вычислениях с INT- и DINT-числами

Арифметические функции при использовании данных форматов INT и DINT могут устанавливать все числовые флаги ("digital flags" - биты состояния). В случае результата, равного нулю, биты CC0 и CC1 сбрасываются в "0". Сочетание CC0 = "0", а CC1 = "1" сообщает о положительном результате. Сочетание CC0 = "1", а CC1 = "0" сообщает об отрицательном результате. Превышение (overflow) пределов диапазона допустимых значений устанавливает биты OV и OS (примечание: при других значениях битов состояния CC0 и CC1). Деление на ноль приводит к установлению в состояние "1" всех "числовых битов состояния" ("digital status bits").

Биты состояния при вычислениях с REAL-числами

Арифметические функции при использовании данных формата REAL могут устанавливать все "числовые биты состояния" ("digital status bits"). В случае результата, равного нулю, биты CC0 и CC1 сбрасываются в "0". Сочетание CC0 = "0", а CC1 = "1" сообщает о положительном результате. Сочетание CC0 = "1", а CC1 = "0" сообщает об отрицательном результате. Превышение (overflow) пределов диапазона допустимых значений устанавливает биты OV и OS (примечание: при других значениях битов состояния CC0 и CC1). При обработке некорректного действительного (REAL) числа происходит установка в состояние "1" всех числовых битов состояния.

Действительное (REAL) число относится к "ненормированным" (или "ненормализованным" - "denormalized"), если оно представляется с уменьшенной точностью. Абсолютное значение "ненормированного" ("denormalized") числа формата REAL меньше чем $1.175494 \cdot 10^{-38}$ (см. главу 24 "Типы данных"). S7-300 интерпретируют ненормированные числа формата REAL как числа, равные нулю.

Таблица 15.3 Установка битов состояния

Вычисления с INT-данными				
Результат	CC0	CC1	OV	OS
<-32768 (+I, -I)	0	1	1	1
<-32768 (*I)	1	0	1	1
-32768 ... -1	1	0	0	-
0	0	0	0	-
+1 ... +32767	0	1	0	-
>+32767 (+I, -I)	1	0	1	1
>+32767 (*I)	0	1	1	1
32768 (/I)	0	1	1	1
(-)65536	0	0	1	1
Деление на 0	1	1	1	1

Вычисления с DINT-данными				
Результат	CC0	CC1	OV	OS
<-2147483648 (+D, -D)	0	1	1	1
<-2147483648 (*D)	1	0	1	1
-2147483648 ... -1	1	0	0	-
0	0	0	0	-
+1... +2147483647	0	1	0	-
>+2147483647 (+D, -D)	1	0	1	1
>+2147483647 (*D)	0	1	1	1
2147483648 (/D)	0	1	1	1
(-)4294967296	0	0	1	1
Деление на 0 (/D, MOD)	1	1	1	1

Вычисления с REAL-данными				
Результат	CC0	CC1	OV	OS
+ нормирован	0	1	1	1
± ненормирован	1	0	1	1
± ноль	1	0	0	-
- нормирован	0	0	0	-
+ бесконечность (деление на 0)	0	1	0	-
- бесконечность (деление на 0)	1	0	1	1
± некорр. REAL	0	1	1	1

Операции сравнения				
Результат	CC0	CC1	OV	OS
равен	0	0	0	-
больше чем	0	1	0	-
меньше чем	1	0	0	-
некорр. REAL	1	1	1	1

Преобразование данных NEG_I				
Результат	CC0	CC1	OV	OS
+1 ... +32767	0	1	0	-
0	0	0	0	-
-32768 ... -1	1	0	0	-
(-) 32768	1	0	1	1

Преобразование данных NEG_D				
Результат	CC0	CC1	OV	OS
+1... +2147483647	0	1	0	-
0	0	0	0	-
-2147483648 ... -1	1	0	0	-
(-) 2147483648	1	0	1	1

Функции сдвига				
Результат	CC0	CC1	OV	OS
"0"	0	0	0	-
"1"	0	1	0	-
с числом поз. 0	-	-	-	-

Логические операции для слов данных				
Результат	CC0	CC1	OV	OS
ноль	0	0	0	-
не ноль	0	1	0	-

Биты состояния при работе с функциями преобразования

Из функций преобразования функции инвертирования влияют на все "числовые биты состояния" ("digital status bits"). Кроме того, следующие функции преобразования устанавливают биты состояния OV и OS в случае появления ошибок: (выход за пределы диапазона допустимых значений и обработка некорректного числа формата REAL):

- ITB и DTB: преобразование числа формата INT в формат BCD
- RND+, RND-, RND, TRUNK: преобразование числа формата REAL в формат DINT

Биты состояния при работе с функциями сравнения

Функции сравнения устанавливают биты состояния CC0 и CC1. Флаги устанавливаются независимо от типа выполняемой функции сравнения; зависят они только от отношения между двумя значениями, указанными в выражении функции сравнения. Функция сравнения чисел формата REAL производит проверку корректности данных формата REAL.

Биты состояния при работе с логическими операциями для слов и с функциями сдвига

Логические операции для слов и функции сдвига устанавливают биты состояния CC0 и CC1. Бит OV находится в сброшенном состоянии ("0").

Установка и сброс RLO

Оператор SET устанавливает RLO в состояние "1", а оператор CLR сбрасывает RLO в состояние "0". Одновременно происходит установка в "1" или сброс в "0" бита состояния STA. Рассматриваемые операторы выполняются без всяких условий.

Операторы SET и CLR устанавливают также биты состояния OR и /FC, что означает, что после операторов SET или CLR новая логическая операция начинается со следующей операции проверки (опроса).

Вы можете запрограммировать сброс или установку бита (двоичного адреса) с помощью оператора SET:

```
SET   ;
S     M 8.0;     //Установка меркера
R     M 8.1;     //Сброс меркера
CLR   ;
S     C 1;       //Сброс меркера фронта для установки
                 //счетчика "Set counter"
```

Прямая установка и сброс RLO также полезны при использовании с таймерами и счетчиками. Для запуска таймера и счетчика Вам необходимо изменить RLO со значения "0" на значение "1" (помните также, что Вам еще требуется для этого положительный фронт сигнала разрешения). В разделе программы, в котором преобладают числовые логические операции, RLO обычно не определяется, например, при последующей функции перехода для проверки "числовых флагов" (битов состояния -"digital flags"). При этом Вы можете использовать операторы SET и CLR для определенных установок и сбросов RLO или для программного изменения RLO.

Об инвертировании RLO с NOT см. гл. 4 "Двоичные логические операции".

Установка и сброс BR

С помощью оператора SAVE Вы можете сохранить в бите состояния "двоичный результат" (BR - "binary result"). Оператор SAVE пересылает состояние сигнала RLO в бит состояния BR. Оператор SAVE выполняются без всяких условий; при этом он не влияет на другие биты состояния.

```
SET        ;
SAVE       ;       //Установка бита BR в "1"
...
AN         OV;
SAVE       ;       //Сброс бита BR в "0" при превышении
...
```

15.3 Проверка битов состояния

Биты состояния RLO и BR и все числовые флаги (digital flags) могут быть проверены с помощью операций двоичного опроса и функций перехода. Также можно обрабатывать биты состояния после загрузки в аккумулятор слова состояния.

Проверка битов состояния с помощью операции двоичного опроса

Вы можете использовать все операторы опроса (проверки), описанные в главе 4 "Двоичные логические операции", для проверки числовых флагов (digital flags) и двоичного результата BR (см. ниже).

A	-	Проверка (опрос) выполнения условия и логическое AND (И)
O	-	Проверка (опрос) выполнения условия и логическое OR (ИЛИ)
X	-	Проверка (опрос) выполнения условия и Exclusive OR (исключающее ИЛИ)
AN	-	Проверка (опрос) невыполнения условия и логическое AND (И)
ON	-	Проверка (опрос) невыполнения условия и логическое OR (ИЛИ)
XN	-	Проверка (опрос) невыполнения условия и Exclusive OR (исключающее ИЛИ)

>0	Результат больше 0	[(CC0=0) & (CC1=1)]
>=0	Результат больше или равен 0	[(CC0=0)]
<0	Результат меньше 0	[(CC0=1) & (CC1=0)]
<=0	Результат меньше или равен 0	[(CC1=0)]
<>0	Результат не равен 0	[(CC0=1) & (CC1=0)] v [(CC0=0) & (CC1=1)]
==0	Результат равен 0	[(CC0=0) & (CC1=0)]
UO	Результат не верен	[(CC0=0) & (CC1=1)]
OV	Переполнение (overflow)	[(OV=1)]
OS	Сохраненное переполнение	[(OS=1)]
BR	Двоичный результат	

При этом принципы использования операторов опроса (проверки) такие же, как и при проверке, например, входов.

Проверка битов состояния с помощью функций перехода

Вы можете проверять биты состояния RLO и BR, все комбинации CC0 и CC1, а также биты состояния OV и OS с помощью соответствующих функций перехода (табл. 15.4). В главе 16 "Функции перехода" содержится подробное описание.

Таблица 15.4 Проверка битов состояния с помощью функций перехода

RLO	BR	CC0	CC1	OV	OS	Функции перехода
1	-	-	-	-	-	JC, JCB
0	-	-	-	-	-	JCN, JNB
-	1	-	-	-	-	JBI
-	0	-	-	-	-	JNBI
-	-	0	0	-	-	JZ, JMZ, JPZ
-	-	0	1	-	-	JN, JP, JPZ
-	-	1	0	-	-	JN, JM, JMZ
-	-	1	1	-	-	JUO
-	-	-	-	1	-	JO
-	-	-	-	-	1	JOS

Замечания по проверке битов состояния OV и OS

Если результат вычисления выходит за пределы диапазона допустимых значений для определенного типа данных, то бит состояния OV устанавливается в "1"; одновременно устанавливается в "1" бит состояния OS (бит для сохранения информации о том, что имело место нарушение границы диапазона допустимых значений).

Если результат следующей далее функции (например, в цепочке вычислений) находится в пределах допустимых значений, флаг OV будет сброшен. Однако, при этом флаг OS останется установленным, так что тот факт, что имело место превышения границ разрешенного диапазона значений, останется сохраненным, при этом он может быть обнаружен с помощью проверки (опроса) в конце вычислений.

Флаг OS не может быть сброшен, пока не встретится функция перехода JOS или не сменится блок (т.е., пока не произойдет новый вызов блока или пока не закончится обработка текущего блока).

Далее представлены два варианта программного решения задачи проверки состояния флагов для определения того факта, что имело место превышение пределов диапазона допустимых значений - в первой программе выполняется двоичный опрос (проверка) флагов, а во второй - выполняется проверка флагов с помощью условных переходов.

```
Двоичный опрос
L     Value1;
L     Value2;
+I    ;
A     OV;           //Отдельная проверка
=     Status1;
L     Value3;
+I    ;
A     OV;           //Отдельная проверка
=     Status2;
L     Value4;
+I    ;
A     OS;           //Отдельная проверка
=     Status_overall;
T     Result;

Функции перехода
L     Value1;
L     Value2;
+I    ;
JO    ST1;          //Отдельная проверка
L     Value3;
+I    ;
JO    ST2;          //Отдельная проверка
L     Value4;
+I    ;
JOS   STOV;         //Общая проверка
T     Result;
```

Как видно из последних примеров, опрос (проверку) флагов можно производить либо после каждой операции вычисления (опрос бита состояния OV), либо в конце цепочки вычислений (опрос бита состояния OS).

15.4 Использование двоичного результата (бита состояния BR)

STEP 7 использует бит BR (двоичный результат) для реализации механизма EN/ENO при использовании языков программирования LAD ("контактный план" - ["ladder diagram"]) и FBD ("Функциональная блок-схема" - ["functional block diagram"]).

Вы можете пропустить эту информацию, если Вы используете только язык программирования STL; тогда в Вашем распоряжении двоичный результат BR как дополнительный бит RLO.

Тем не менее, для индикации ошибок при обработке блоков (как это используется в системных блоках SFB и SFC и некоторых стандартных блоках) Вы можете использовать BR как флаг групповой ошибки, даже если Вы используете исключительно программирование на STL.

Механизм EN/ENO

При использовании языков программирования LAD и FBD все функциональные элементы имеют разрешающие входы EN и разрешающие выходы ENO. Если на разрешающем входе EN присутствует единица "1", то функциональный элемент активизируется (выполняется его функция). Если функция выполняется корректно, то на разрешающем выходе ENO также присутствует единичный сигнал "1". Если происходит ошибка во время выполнения функции (например, переполнение при вычислении арифметической функции), выход ENO сбрасывается в состояние "0". Если на разрешающем входе EN присутствует сигнал "0", то на разрешающем выходе ENO также присутствует сигнал "0".

Вы можете использовать такие свойства разрешающих входов/выходов (EN/ENO) при связывании нескольких функциональных элементов в единую цепь; при этом разрешающий выход ENO предыдущего элемента будет управлять разрешающим входом EN последующего функционального элемента (см. рис. 15.1). Такая конфигурация позволит сделать возможным, чтобы происходил разрыв ("switch off") всей цепи (чтобы не был активен ни один функциональный элемент, если в примере на входе I 1.0 присутствует сигнал "0") или чтобы при появлении ошибки в одном из функциональных элементов последующие за ним элементы не могли быть активизированы.

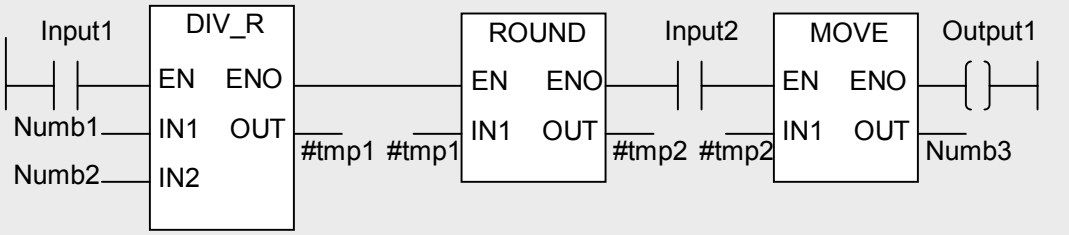
Вход EN и выход ENO не являются параметрами блока, однако, инструкции программы приводят к тому, что LAD/FBD редактор сам генерирует предыдущие и последующие функциональные элементы (даже в случае применения функций и функциональных блоков). В этом случае LAD/FBD редактор использует двоичный результат для хранения состояния сигнала на разрешающем входе EN во время обработки блока или в качестве сигнального бита для индикации ошибки в функциональном элементе.

Вы можете найти программу, указанную на рис. 15.1, в сегменте Network 8 блока FB 115 в программе "Program Flow Control" (в библиотеке STL_Book). Если Вы видите указанный сегмент блока FB 115 на экране монитора, Вы можете переключиться на LAD-представление, выбрав опции: View -> LAD. После этого редактор включит графическое отображение LAD-программы.

Если Вы пишите свои собственные функции или функциональные блоки и хотите их использовать, например, с LAD- или FBD-представлением, Вы должны таким образом управлять BR, чтобы этот бит сбрасывался в "0" при обнаружении ошибки (см. ниже).

Сообщение о групповой ошибке в блоках

Вы можете использовать двоичный результат в качестве сигнального бита для сообщения о групповой ошибке в блоках. Если блок обрабатывается без ошибок, бит BR устанавливается в состояние "1", иначе - BR сбрасывается в состояние "0".




```
        A(
        A(
        A(
        A     Input1
        JNB   M001     //Опрос входа EN
        L     Numb1
        L     Numb2    //функционального элемента
        /R
        T     tmp1
        AN    OV       //Проверка ошибки в элементе
        SAVE           //и установка BR
        CLR
M001:   A     BR       //Проверка BR для установки ENO
        )
        JNB   M002     //Вход EN следующего элемента
        L     tmp1
        RND
        T     tmp2
        AN    OV
        SAVE
        CLR
M002:   A     BR
        )
        A     Input2
        JNB   M003     //Вход EN последнего элемента
        L     tmp2
        T     Numb3
        SET
        SAVE
        CLR
M003:   A     BR
        )
        =     Output1
```

Рис 15.1 Пример реализации механизма EN/ENO

Пример: В начале обработки блока BR устанавливается в состояние "1". На тот случай, если после этого при обработке блока будет обнаружена ошибка, например, результат выходит за рамки определенного диапазона (после чего дальнейшая обработка блока должна быть остановлена) Вы должны запрограммировать сброс BR в состояние "0" с помощью JNB и переход на конец блока, например, (в случае ошибки условие должно выдать сигнал "0").

```
SET   ;
SAVE  ;           //BR = "1"
...
L     10000;
L     Result;     //Если Result > 10000,
<=I   ;           //то BR = "0" и выполнить
JNB   ERR;        //переход к ERR
...
```

В примере "Clock entry" в разделе 26.4 "Краткое описание примера фрейма сообщения" также используется BR для сообщения о групповой ошибке.

16 Функции перехода

С помощью функций перехода (jump functions) Вы можете прервать линейное выполнение программы и продолжить ее выполнение в другой точке блока. Такое ветвление программы может быть организовано либо с проверкой выполнения определенного условия, либо без проверки каких-либо условий (соответственно в программе - либо условный, либо безусловный переход).

Распределитель переходов (распределение переходов с проверкой параметра) и циклический переход также доступны пользователю как особые формы функций перехода.

Обзор

JU	*метка*	Безусловный переход
JC	*метка*	Переход, если RLO = "1"
JCN	*метка*	Переход, если RLO = "0"
JCB	*метка*	Переход, если RLO = "1", и сохранение RLO
JNB	*метка*	Переход, если RLO = "0", и сохранение RLO
JBI	*метка*	Переход, если BR = "1"
JBI	*метка*	Переход, если BR = "0"
JZ	*метка*	Переход, если результат равен "0"
JN	*метка*	Переход, если результат не равен "0"
JP	*метка*	Переход, если результат больше "0"
JPZ	*метка*	Переход, если результат больше или равен "0"
JM	*метка*	Переход, если результат меньше "0"
JMZ	*метка*	Переход, если результат меньше или равен "0"
JUO	*метка*	Переход, если результат неверен
JO	*метка*	Переход при переполнении (при проверке бита OV)
JOS	*метка*	Переход при переполнении (при проверке бита OS)
JL	*метка*	Переход в распределителе переходов
LOOP	*метка*	Переход циклический

В этой главе будут рассмотрены функции перехода, используемые в языке программирования STL. В языке программирования SCL выполнение переходов обеспечивается с помощью различных методов ветвления программы, например, с помощью оператора IF (см. главу 28 "Операторы управления").

Примеры, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book в разделе "Program Flow Control" ("Управление выполнением программы") в функциональном блоке FB 116 или в исходном файле Chap_16.

16.1 Программирование функций перехода

Инструкция для каждой функции перехода содержит оператор перехода, определяющий проверяемое условие, и метку перехода, которая в свою очередь показывает, в какой точке программы следует продолжать ее обработку в случае, когда условие для перехода выполняется.

Метка перехода содержит до 4 символов алфавита и числового ряда, а также символ подчеркивания. Метка перехода не может начинаться с цифры. После метки перехода должно следовать двоеточие. Метка перехода указывает на выражение (строку), которая должна обрабатываться после выполнения операции перехода.

На рис. 16.1 показан пример ветвления программы.

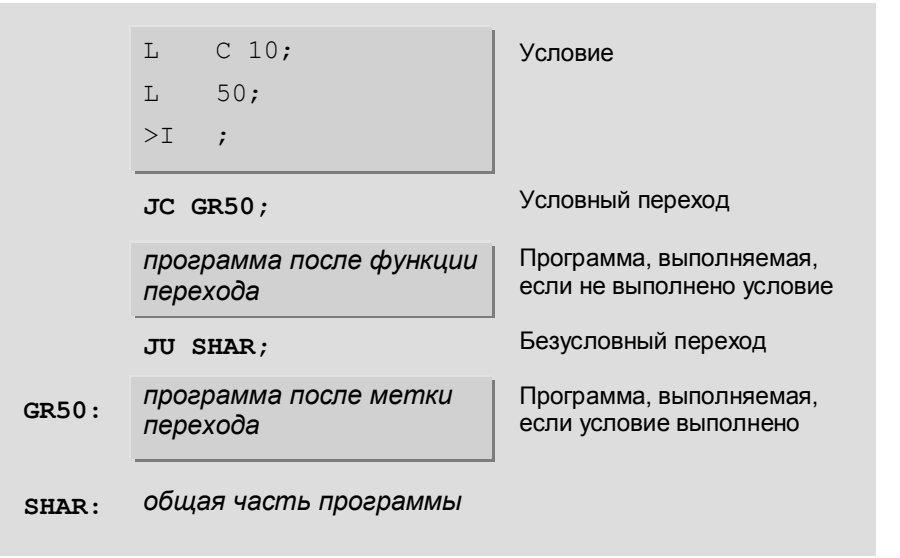


Рис.16.1 Пример ветвления программы.

В примере условием перехода является положительный результат операции сравнения; операция сравнения формирует результат логической операции RLO. Собственно значение RLO и является условием перехода для выражения с JC. При положительном результате операции сравнения (условие операции сравнения выполняется) RLO = "1", при этом выполняется переход к метке GR50. Выполнение программы продолжается с этой метки. При отрицательном результате операции сравнения (условие операции сравнения не выполняется) RLO = "0" и перехода к GR50 нет, программа выполняется со следующего за оператором условного перехода выражения.

Переход может выполняться как вперед (в направлении выполнения программы, т.е., в направлении увеличения номеров строк), так и назад. Переход может выполняться только внутри блока; то есть, назначение точек перехода должно задаваться в том же блоке, где находятся выражения с оператором перехода. Разбиение блока на сегменты не влияет на функцию перехода.

Точки перехода должны иметь уникальный идентификатор, что означает, что Вы можете назначать любую метку перехода в блоке, но только один раз с таким именем.

К одной точке (метке) переход может осуществляться из нескольких точек программы блока. Если Вы используете главное управляющее реле MCR, то метки перехода должны быть в той же MCR-зоне или в той же MCR-области, что и выражение с оператором перехода.

При программировании на языке STL идентификаторы меток сохраняются в соответствующей неисполняемой части блока на носителях данных программатора PG. Только величина перехода хранится в рабочей (work) памяти CPU (в скомпилированном блоке). По этой причине изменения в программе блока, сделанные в интерактивном режиме в CPU, должны также быть выполнены и на дисках программатора PG для сохранения исходных назначений. Если обновления программы не сделаны или если блоки были перенесены из CPU в программатор PG, то соответствующие неисполняемые части блоков переписываются или уничтожаются. Затем редактор сгенерирует свои собственные назначения меток (M001, M002 и т.д.) на экране или в распечатке.

16.2 Безусловный переход

Функция безусловного перехода JU выполняется всегда, т.е. не зависит ни от каких условий. Функция JU прерывает последовательное выполнение программы и продолжает это выполнение с другой точки программы - с метки перехода, указанной в инструкции.

Функция безусловного перехода JU не влияет на биты состояния. Если встречаются операторы проверки (опроса), такие, например, как A I, O I и т.д., располагающиеся непосредственно перед функцией перехода и сразу после метки перехода, то они воспринимаются как отдельные логические операции.

16.3 Функции перехода в зависимости от состояния RLO и BR

Переход в программе может быть сделан зависящим от состояния сигналов битов RLO и BR (см. табл. 16.1). Кроме того, одновременно можно сохранить результаты проверки битов состояния RLO и BR.

Таблица 16.1 Функции перехода с RLO и BR

RLO	BR	Выполняемые переходы	
"1"	-	JC	Переход, если RLO = "1"
"1"	--> "1"	JCB	Переход, если RLO = "1" и сохранить RLO
"0"	-	JCN	Переход, если RLO = "0"
"0"	--> "0"	JNB	Переход, если RLO = "0" и сохранить RLO
-	"1"	JBI	Переход, если BR = "1"
-	"0"	JNBI	Переход, если BR = "0"

Установка битов состояния

Функции перехода, зависящие от RLO, устанавливают биты состояния STA и RLO в "1", а OR и /FC в "0", и при выполнении условия для перехода, и при невыполнении этого условия.

Это приводит к нижеуказанным последствиям при использовании этих функций перехода: RLO всегда устанавливается в состояние "1". Если выражения содержат операции, зависящие от RLO, следующие сразу за такими функциями перехода, то они будут выполняться, если не будет выполнен переход. Если встречаются операторы проверки (опроса), такие, например, как A I, O I и т.д., располагающиеся непосредственно за такими функциями перехода, то эти операторы проверки (опроса) воспринимаются как операции первичного опроса, что означает, что начинается новый логический шаг (новая логическая операция).

Функции перехода, зависящие от BR, устанавливают бит состояния STA в "1", а OR и /FC в "0", и при выполнении условия для перехода, и при невыполнении этого условия. Биты состояния BR и RLO остаются неизменными. Это приводит к нижеуказанным последствиям при использовании таких функций перехода: такие функции перехода завершают логическую операцию; новая логическая операция начинается после таких функций перехода или с метки перехода в программе при выполнении условия для перехода. Бит состояния RLO остается и может быть проверен с помощью операций с памятью сразу после функции перехода.

Переход при RLO = "1"

Функция перехода JC выполняется только в том случае, если RLO = "1" в момент обработки функции. Если RLO = "0", то переход не выполняется и программа продолжает выполняться со следующей инструкции.

Переход при RLO = "0"

Функция перехода JCN выполняется только в том случае, если RLO = "0" в момент обработки функции. Если RLO = "1", то переход не выполняется и программа продолжает выполняться со следующей инструкции.

Переход при RLO = "1" и сохранение RLO

Функция перехода JCB выполняется только в том случае, если RLO = "1" в момент обработки функции. Одновременно функция JCB устанавливает двоичный результат BR в состояние "1". Если RLO = "0", то переход не выполняется и программа продолжает выполняться со следующей инструкции. В этом случае функция JCB устанавливает двоичный результат BR в состояние "0" (т.е., в любом случае RLO переносится в двоичный результат BR).

Переход при RLO = "0" и сохранение RLO

Функция перехода JNB выполняется только в том случае, если RLO = "0" в момент обработки функции. Одновременно функция JNB устанавливает двоичный результат BR в состояние "0". Если RLO = "1", то переход не выполняется и программа продолжает выполняться со следующей инструкции. В этом случае функция JNB устанавливает двоичный результат BR в состояние "1" (т.е., в любом случае RLO переносится в двоичный результат BR).

Переход при BR = "1"

Функция перехода JBI выполняется только в том случае, если двоичный результат BR = "1" в момент обработки функции. Если двоичный результат BR = "0", то переход не выполняется и программа продолжает выполняться со следующей инструкции.

Переход при BR = "0"

Функция перехода JBIN выполняется только в том случае, если двоичный результат BR = "0" в момент обработки функции. Если двоичный результат BR = "1", то переход не выполняется и программа продолжает выполняться со следующей инструкции.

16.4 Функции перехода в зависимости от состояния CC0 и CC1

Переход в программе может быть сделан зависящим от состояния сигналов битов CC0 и CC1 (см. табл. 16.2). Это позволяет, например, проверять, положителен ли результат вычисления, равен ли он нулю или меньше нуля (отрицательный результат). Подробную информацию о том, как устанавливаются биты состояния CC0 и CC1, Вы можете найти в главе 15 "Биты состояния".

Таблица 16.2 Функции перехода с CC0 и CC1

CC0	CC1	Выполняемые переходы	
0	0	JZ	Переход, если результат = 0
		JMZ	Переход, если результат = 0 или < 0
		JPZ	Переход, если результат = 0 или > 0
1	0	JM	Переход, если результат < 0
		JMZ	Переход, если результат = 0 или < 0
		JN	Переход, если результат <> 0
0	1	JP	Переход, если результат > 0
		JPZ	Переход, если результат = 0 или > 0
		JN	Переход, если результат <> 0
1	1	JUO	Переход, если результат неверен

Установка битов состояния

Функции перехода, зависящие от CC0 и CC1, не изменяют никаких битов состояния. Если переход выполнен, значение RLO остается при продолжении программы со строки с меткой перехода и может использоваться далее в программе (без изменения в /FC).

Двоичный опрос является другим способом проверки битов состояния (см. главу 15 "Биты состояния").

Переход при условии, что результат равен нулю

Функция перехода JZ выполняется только в том случае, если CC0 = "0" и CC1 = "0". Это случается, если

- аккумулятор accumulator 1 содержит ноль после арифметической или математической операции;
- аккумулятор accumulator 2 содержит то же значение, что и аккумулятор accumulator 1, при выполнении операции сравнения;
- аккумулятор accumulator 1 содержит ноль после выполнения логической операции для чисел;
- значение сдвинутого последним бита содержит ноль после выполнения операции сдвига.

Во всех остальных случаях условие для перехода для функции JZ не выполняется и выполнение программы продолжается со следующей инструкции.

Переход при условии, что результат не равен нулю

Функция перехода JN выполняется только в том случае, если биты состояния CC0 и CC1 имеют разные значения. Этот случай имеет место, если

- аккумулятор accumulator 1 не содержит ноль после арифметической или математической операции;
- аккумулятор accumulator 2 не содержит то же значение, что и аккумулятор accumulator 1, при выполнении операции сравнения;
- аккумулятор accumulator 1 не содержит ноль после выполнения логической операции для чисел;
- значение сдвинутого последним бита содержит "1" после выполнения операции сдвига.

Во всех остальных случаях условие для перехода для функции JN не выполняется и выполнение программы продолжается со следующей инструкции.

Переход при условии, что результат больше нуля

Функция перехода JP выполняется только в том случае, если CC0 = "0" и CC1 = "1". Этот случай имеет место, если

- содержимое аккумулятора accumulator 1 находится внутри диапазона допустимых положительных значений после арифметической или математической операции (проверить нарушение границ диапазона можно с помощью операторов JO или JOS);
- содержимое аккумулятора accumulator 2 больше значения, находящегося в аккумуляторе accumulator 1, при выполнении операции сравнения;
- аккумулятор accumulator 1 не содержит нуля после выполнения логической операции для чисел;
- значение сдвинутого последним бита содержит "1" после выполнения функции сдвига.

Во всех остальных случаях условие для перехода для функции JP не выполняется и выполнение программы продолжается со следующей инструкции.

Переход при условии, что результат больше нуля или равен нулю

Функция перехода JPZ выполняется только в том случае, если CC0 = "0". Этот случай имеет место, если

- содержимое аккумулятора accumulator 1 находится внутри диапазона допустимых положительных значений или равно нулю после арифметической или математической операции (проверить нарушение границ диапазона можно с помощью операторов JO или JOS);
- содержимое аккумулятора accumulator 2 больше значения, находящегося в аккумуляторе accumulator 1, или равно ему при выполнении операции сравнения;
- после выполнения каждой логической операции для чисел;
- после выполнения каждой функции сдвига.

Во всех остальных случаях условие для перехода для функции JPZ не выполняется и выполнение программы продолжается со следующей инструкции.

Переход при условии, что результат меньше нуля

Функция перехода JM выполняется только в том случае, если CC0 = "1" и CC1 = "0". Этот случай имеет место, если

- содержимое аккумулятора accumulator 1 находится внутри диапазона допустимых отрицательных значений после арифметической или математической операции (проверить нарушение границ диапазона можно с помощью операторов JO или JOS);
- содержимое аккумулятора accumulator 2 меньше значения, находящегося в аккумуляторе accumulator 1, при выполнении операции сравнения.

Во всех остальных случаях условие для перехода для функции JM не выполняется и выполнение программы продолжается со следующей инструкции.

Переход при условии, что результат меньше нуля или равен нулю

Функция перехода JMZ выполняется только в том случае, если CC1 = "0". Этот случай имеет место, если

- содержимое аккумулятора accumulator 1 находится внутри диапазона допустимых отрицательных значений или равно нулю после арифметической или математической операции (проверить нарушение границ диапазона можно с помощью операторов JO или JOS);
- содержимое аккумулятора accumulator 2 меньше значения, находящегося в аккумуляторе accumulator 1, или равно ему при выполнении операции сравнения;

Во всех остальных случаях условие для перехода для функции JMZ не выполняется и выполнение программы продолжается со следующей инструкции.

Переход при условии, что результат некорректен

Функция перехода JUO выполняется только в том случае, если CC0 = "1" и CC1 = "1". Этот случай имеет место, если

- если совершается попытка деления на ноль при арифметической операции;
- если некорректное действительное (REAL) число было определено как исходное значение или было получено при выполнении операции.

Во всех остальных случаях условие для перехода для функции JUO не выполняется и выполнение программы продолжается со следующей инструкции.

16.5 Функции перехода в зависимости от состояния OV и OS

Переход в программе может быть сделан зависящим от состояния сигналов битов OV и OS. Это позволяет проверять, находится ли все еще результат вычисления в диапазоне допустимых значений. Подробную информацию о том, как устанавливаются биты состояния OV и OS, Вы можете найти в главе 15 "Биты состояния".

Переход при условии, что результат вышел за пределы диапазона (переполнение) (проверка бита OV)

Функция перехода JO выполняется только в том случае, если OV = "1". Это случается, если результат вычисления выходит за пределы допустимых значений в предыдущей операции. Бит состояния OV может быть установлен такими функциями, как:

- арифметические функции;
- математические функции;
- функции инвертирования числа;
- функции сравнения для чисел формата REAL;
- функции преобразования чисел форматов INT/DINT в формат BCD и чисел формата REAL в формат DINT.

Если бит состояния OV = "0", то условие для перехода для функции JO не выполняется и выполнение программы продолжается со следующей инструкции.

Если выполняется цепочка последовательных вычислений, то состояние бита OV должно проверяться после выполнения каждого вычисления, так как бит состояния OV вновь будет сброшен после выполнения операции вычисления, результат которой будет находится в диапазоне допустимых значений.

Бит состояния OS может быть опрошен после выполнения цепочки вычислений для проверки, не вышел ли за пределы допустимых значений результат одного из вычислений в рассматриваемой последовательности операций.

Переход при условии, что результат вышел за пределы диапазона (переполнение) (проверка бита OS)

Функция перехода JOS выполняется только в том случае, если OS = "1". Это происходит во всех случаях, когда результат вычисления выходит за пределы допустимых значений, что вызывает установку бита состояния OV (см. выше). В отличие от OV, бит OS сохраняет свое состояние, даже если после его установки результат одной из последующих операций будет в диапазоне допустимых значений. Бит состояния OS может быть сброшен при следующих обстоятельствах:

- при вызове блока и при завершении блока;
- при выполнении перехода JOS.

Если бит состояния OS = "0", то условие для перехода для функции JOS не выполняется и выполнение программы продолжается со следующей инструкции.

16.6 Распределитель переходов (Jump Distributor)

Распределитель переходов (Jump Distributor) JL позволяет определять (вычислять) переходы в разделе программы в блоке к разным точкам, отмеченным метками.

Функция JL работает вместе с набором функций перехода JU. Последовательность выражений с операторами перехода JU следует сразу после функции JL и может содержать до 255 строк (входов). В инструкции после оператора JL следует метка перехода, указывающая на конец списка функций перехода JU (на первую инструкцию, следующую за набором функций перехода JU).

Вы можете запрограммировать распределитель переходов (Jump Distributor) JL в соответствии со следующей общей схемой:

```
    L    Number_of_positions;   //номер позиции
    JL   End;
    JU   M0;
    JU   M1;
    JU   M2;
      ...
    JU   Mx;
End:  ...
```

В этом примере переменная *Number_of_positions* содержит число, загружаемое в аккумулятор accumulator 1. Вслед за операцией загрузки следует распределитель переходов JL с меткой, указывающей на конец списка функций перехода JU.

Номер перехода, который должен быть выполнен, содержится в правом байте аккумулятора accumulator 1. Если этот аккумулятор содержит 0, то выполняется первая функция перехода. Если этот аккумулятор содержит 1, то выполняется вторая функция перехода и так далее. Если число в аккумуляторе превышает размер списка операторов перехода, то

происходит переход на конец списка (на первую инструкцию, следующую за набором функций перехода JU).

JL не зависит ни от каких условий и не изменяет битов состояния.

При этом только выражения с оператором JU, располагающиеся без пробелов, допускаются в списке операторов перехода функции JL. Назначая произвольные метки для этих операторов перехода, Вы должны придерживаться общих правил для меток.

16.7 Циклический переход (Loop Jump)

Функция циклического перехода (Loop Jump) LOOP позволяет просто реализовать в программе программные циклы.

Функция LOOP интерпретирует число в правом слове (word) аккумулятора accumulator 1 как беззнаковое 16-разрядное число из диапазона от 0 до 65535.

При обработке функция LOOP сначала декрементирует (уменьшает) содержимое аккумулятора accumulator 1 на 1. Если значение при этом не равно нулю, то выполняется переход к указанной метке.

Если значение после декрементирования равно нулю, то выполняется следующее за телом цикла выражение.

Значение в аккумуляторе accumulator 1, таким образом, соответствует числу циклов, которые должны быть выполнены. Вы должны сохранить это число в счетчике цикла. Вы можете использовать любое число (адрес) как счетчик цикла.

Вы можете запрограммировать циклический переход LOOP в соответствии со следующей общей схемой:

```
          L     Number;
Next:     T     Counter;
          ...
          ...
          ...
          L     Counter;
          LOOP  Next;
          ...
```

В этом примере переменная *Number* содержит общее число циклов, которое необходимо выполнить. Переменная *Counter* содержит число циклов, которое осталось выполнить.

При первом проходе цикла значению *Counter* задается число циклов, которое необходимо выполнить. В конце программного цикла содержимое переменной *Counter* загружается в аккумулятор и декрементируется с помощью оператора LOOP. Если после этого аккумулятор не содержит нуля, то выполняется переход к метке цикла - здесь к метке Next. После этого происходит обновление переменной *Counter.*

Функция циклического перехода LOOP не изменяет битов состояния.

17 Главное управляющее реле MCR

Управляя методом переключения, главное управляющее реле (Master Control Relay - MCR) активирует или деактивирует отдельные фрагменты схемы управления, которая может состоять из одного или нескольких уровней. При деактивации уровня:

- выключаются все переманентные контакторы и
- остаются неизменными состояния всех реманентных контакторов.

Вы можете вновь изменить состояния этих контакторов только тогда, когда главное управляющее реле MCR будет активировано.

Эти свойства главного управляющего реле MCR определяют его использование особенно в LAD-программах.

В этой главе рассматриваются операторы, необходимые для выполнения функций главного управляющего реле MCR в языке программирования STL. Вы можете использовать эти операторы для эмуляции свойств главного управляющего реле MCR в STL.

Примеры использования главного управляющего реле MCR, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book в разделе "Program Flow Control" ("Управление выполнением программы") в функциональном блоке FB 117 или в исходном файле Chap_17.

Необходимо отметить, что выключение с помощью программного варианта главного управляющего реле (Master Control Relay - MCR) не заменяет антиаварийных или предохранительных устройств. Вы должны трактовать включение с помощью главного управляющего реле MCR, как включение посредством операций с памятью (memory functions)!

Язык программирования STL предоставляет следующие операторы для выполнения функций главного управляющего реле MCR:

- MCRA оператор активации области MCR
- MCR(оператор открытия зоны MCR
-)MCR оператор закрытия зоны MCR
- MCRD оператор деактивации области MCR

Операторы MCRA и MCRD определяют область в Вашей программе, на которую распространяется действие MCR. Внутри этой области Вы можете использовать операторы MCR(и)MCR для задания одной или нескольких зон действия MCR, в которых зависимость от MCR может включаться или выключаться.

Вы можете также использовать вложенные MCR зоны. Результат логической операции RLO, выполняемой непосредственно перед зоной MCR включает или выключает MCR-зависимость внутри этой зоны.

17.1 MCR-зависимость (MCR Dependency)

Главное управляющее реле (MCR) влияет на все операции, которые возвращают (записывают) значения в память. Такие операции, зависящие от MCR, при включении MCR-зависимости, независимо от результатов любых предыдущих двоичных логических операций или логических операций с числами, реагируют следующим образом:

- оператор присвоения (=):
  содержимое адреса сбрасывается в состояние "0"
- операторы установки Set (S) и сброса Reset (R):
  содержимое адреса остается неизменным
- оператор пересылки Transfer (T):
  пересылается ноль "0"

Некоторые функции STL используют операторы пересылки (незаметно для пользователя) для того, например, чтобы записать значение в адресный регистр. Так как оператор пересылки записывает значение нуля "0", если MCR-зависимость включена, то выполнение соответствующей функции (использующей оператор пересылки) не может быть гарантировано.

Поэтому, чтобы во время работы избежать переход CPU в состояние STOP или в неопределенное состояние, Вы должны исключить возможность влияния MCR на соответствующие части программы, содержащие:

- вызовы блоков с параметрами
- операции доступа к параметрам блоков, которые относятся к типам параметров (например, BLOCK_DB)
- операции доступа к параметрам блоков, которые являются компонентами или элементами сложных типов данных или типов, определенных пользователем UDT.

Если MCR-зависимость выключена, то операции, зависящие от MCR, выполняются в "нормальном" режиме, в соответствии с описанием в соответствующих разделах данной книги.

Вы включите MCR-зависимость в зоне, если RLO равен "0" непосредственно перед открытием зоны (аналогично выключению главного управляющего реле MCR). Если открытие MCR-зоны происходит при RLO, равном "1" (главное управляющее реле MCR включается), то обработка программы в такой MCR-зоне выполняется без MCR-зависимости.

Пример:

```
MCRA    ;          //Активация MCR
A       Input0;
MCR(    ;          //Открытие MCR-зоны
A       Input1;
A       Input2;
=       Output0;
)MCR    ;          //Закрытие MCR-зоны
```

```
MCRD    ;           //Деактивация MCR
```

В этом примере переменная *Input0* = "0" сбрасывает содержимое адреса *Output0* в "0". Если *Input0* имеет состояние "1", то содержание адреса *Output0* будет зависеть только от *Input1* и *Input2*.

17.2 MCR-область (MCR Area)

Для того, чтобы использовать функции главного управляющего реле (MCR), Вы должны определить область его действия - MCR-область (MCR Area) с помощью операторов MCRA и MCRD. MCR-зависимость активна внутри MCR-области (но пока еще не включена)

```
MCRA    ;           //Активация MCR
...
...                 //MCR-область
...
MCRD    ;           //Деактивация MCR
```

Оператор MCRA определяет начало MCR-области, а оператор MCRD закрывает эту область. Если Вы вызываете блок внутри MCR-области, то MCR-зависимость является деактивированной в вызванном блоке (см. рис. 17.1). MCR-область начинается снова только с появлением нового оператора MCRA. Когда по окончании обработки вызванного блока управление возвращается в вызывающий блок, MCR-зависимость вновь будет находиться в том состоянии, в котором она находилась до вызова блока, независимо от состояния MCR-зависимости при окончании обработки этого блока.

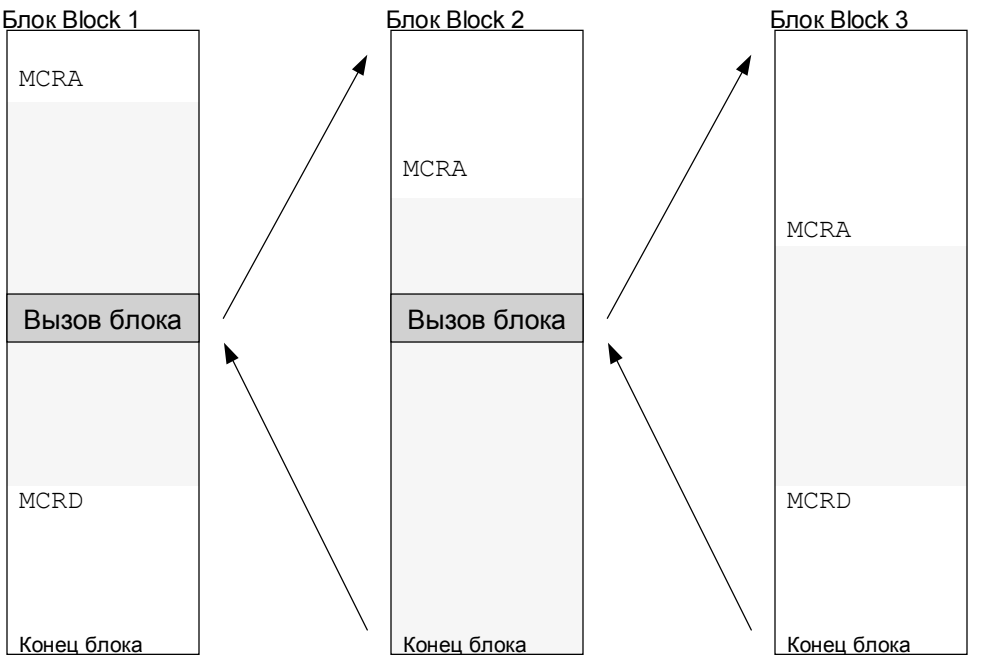


Рис. 17.1 MCR-область в случае вызовов блоков

17.3 MCR-зона (MCR Zone)

Вы можете определить MCR-зону (MCR Zone) с помощью операторов MCR(и)MCR. Внутри MCR-зоны Вы можете включать MCR-зависимость при RLO = "0" и выключать при RLO = "1".

```
...                 //Включение MCR
...                 //при "0"
A     Input3;
MCR(  ;             //Включение MCR-зависимости
...
...                 //MCR-зона
...
)MCR  ;             //Выключение MCR-зависимости
...
```

Операторы MCR(и)MCR заканчивают двоичную логическую операцию (комбинацию).

Вы можете открыть другую MCR-зону внутри MCR-зоны. Глубина вложения для MCR-зон может достигать 8, что означает, что Вы можете открыть до 8 зон перед тем, как должны будете закрыть зону.

Если MCR-зона открыта, Вы можете управлять MCR-зависимостью включенной MCR-зоны с помощью RLO. Однако, если MCR-зависимость включена в MCR-зоне верхнего уровня, Вы не сможете выключить MCR-зависимость в MCR-зоне нижнего уровня. Главное управляющее реле (MCR) первой MCR-зоны управляет MCR-зависимостью во всех включенных зонах (см. рис. 17.2).

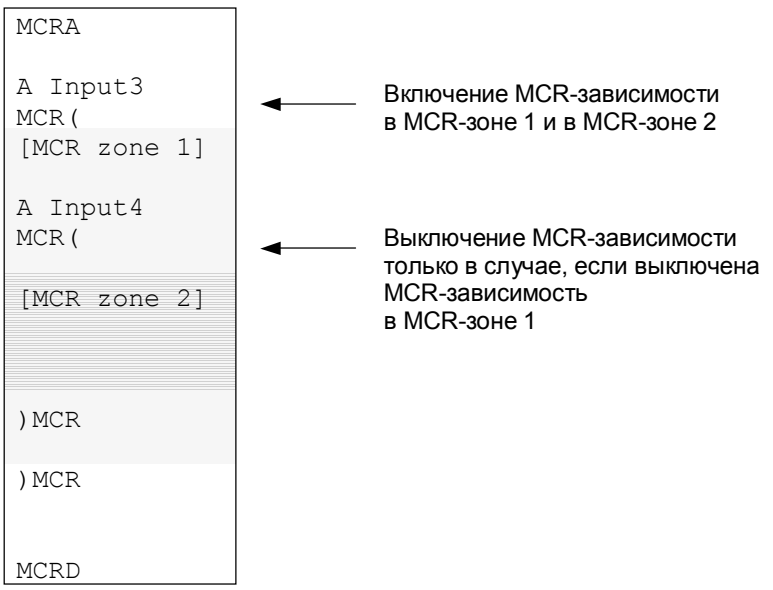


Рис. 17.2 MCR-зависимость в случае вложенных MCR-зон

Вызов блока внутри MCR-зоны не изменяет глубины вложения MCR-зоны. Программа в вызванном блоке находится все еще в MCR-зоне, которая была в открытом состоянии, когда был вызван блок (и управляется из нее). Тем не менее, Вы должны вновь активировать MCR-зависимость в вызванном блоке с помощью открытия MCR-области оператором MCRA (см. рис. 17.3).

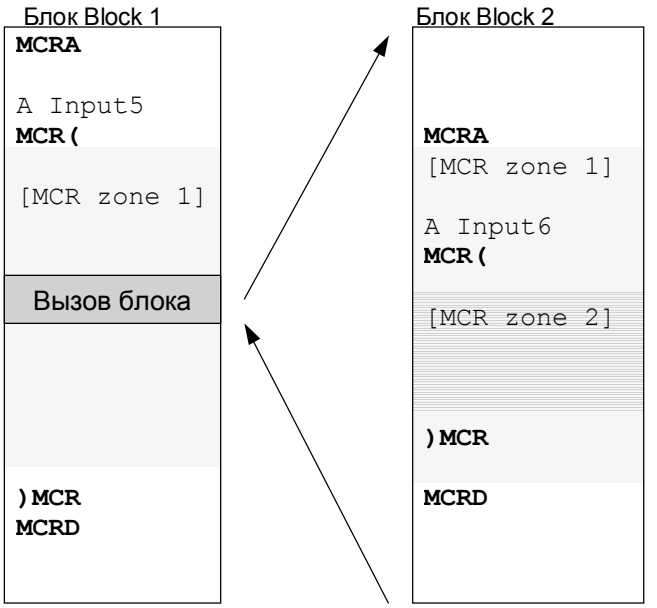


Рис. 17.3 MCR-зоны в случае вызова блока

На рис. 17.3 адреса *Input5* и *Input6* управляют MCR-зависимостью. Посредством адреса *Input5* Вы можете включить MCR-зависимость в обоих зонах (сигналом "0"), независимо от состояния адреса *Input6*. Если MCR-зависимость зоны 1 выключается посредством *Input5* = "1", то Вы можете управлять MCR-зависимостью в зоне 2 посредством адреса *Input6* (см. табл. 17.1).

Таблица 17.1 Пример управления MCR-зависимостью в случае вложенных MCR-зон

Вход *Input5*	Вход *Input6*	Зона Zone 1	Зона Zone 2
"1"	"1"	Нет MCR-зависимости	
"1"	"0"	Нет MCR-зависимости	MCR-зависимость включена
"0"	"1" или "0"	MCR-зависимость включена	

17.4 Установка и сброс битов периферии (I/O битов)

При включенной MCR-зависимости Вы можете устанавливать (Set) и сбрасывать (Reset) биты в области I/O с помощью системных функций.

Для этого требуется, чтобы биты, которыми необходимо управлять, были в области отображения выходов процесса.

Системная функция **SFC 79 SET** предназначена для установки I/O битов, а системная функция **SFC 80 RSET** предназначена для их сброса (см. табл. 17.2). Вы можете вызывать эти функции в MCR-зоне. Системные функции работают только в том случае, когда MCR-зависимость включена; если MCR-зависимость выключена, то вызовы указанных системных функций останутся без всяких последствий.

Таблица 17.2 Параметры SFC для управления I/O-битами

SFC	Параметр	Объявление	Тип	Назначение, описание
79	N	INPUT	INT	Число устанавливаемых битов
	RET_VAL	OUTPUT	INT	Информация об ошибках
	SA	OUTPUT	POINTER	Указатель на первый бит, который должен быть установлен
80	N	INPUT	INT	Число сбрасываемых битов
	RET_VAL	OUTPUT	INT	Информация об ошибках
	SA	OUTPUT	POINTER	Указатель на первый бит, который должен быть сброшен

При выполнении установки и сброса I/O-битов одновременно обновляются выходы из области отображения процесса по выходам.

Периферийные входы/выходы (I/O) управляются побайтно (байт за байтом). Биты, не выбранные с помощью SFC-функций, (в первом и в последнем байте) сохраняют состояние сигнала, так как они доступны в области отображения процесса.

Пример:

```
CALL SFC 79 (
  N         := 8,
  RET_VAL   := MW 10,
  SA        := P#12.0);
CALL SFC 80 (
  N         := 16,
  RET_VAL   := MW 12,
  SA        := P#13.5);
```

В данном примере вызов функции SFC 79 SET устанавливает I/O-биты, соответствующие выходам Q 12.0 ... Q 12.7; вызов функции SFC 80 RSET сбрасывает I/O-биты, соответствующие выходам Q 13.5 ... Q 15.5.

Параметр N определяет число битов, которые должны быть обработаны, а параметр SF определяет первый бит (тип "указатель" - POINTER) из числа битов, которые должны быть обработаны. Указанные функции SFC используют также параметр RET_VAL, в котором они возвращают информацию об ошибках, которые возникли при выполнении функций.

18 Функции блоков (Block Functions)

Из этой главы Вы узнаете, как вызывать и завершать обработку кодовых блоков, а также, как работать с адресами из блоков данных при использовании языка программирования STL. В следующей главе будут рассмотрены параметры блоков и использование этих параметров. Данная глава является продолжением раздела 3.4 "Программирование кодовых блоков на STL" и раздела 3.6 "Программирование блоков данных".

Как вызываются блоки при использовании языка программирования SCL, описано в главе 29 "SCL блоки".

Примеры, рассматриваемые в данной главе, Вы можете найти на прилагаемой дискете в библиотеке STL_Book в разделе "Program Flow Control" ("Управление выполнением программы") в функциональном блоке FB 118 или в исходном файле Chap_18.

18.1 Функции для кодовых блоков

К функциям для кодовых блоков относятся инструкции для вызова и завершения обработки блоков (см. табл. 18.1).

Таблица 18.1 Функции для кодовых блоков

Вызов функционального блока		
С блоком данных и с параметрами блока	Как локальный экземпляр и с параметрами блока	Без параметров блока безусловный вызов и вызов по условию

CALL FB 10, DB 10 (
` In1 := Number1;`
` In2 := Number2;`
` Out := Number3);`
	CALL *name* (
	`  In1 := Number1;`
	`  In2 := Number2;`
	`  Out := Number3);`
		UC FB 11;
		CC FB 11;

Вызов функции		
Со значением функции и с параметрами блока	Без значения функции, но с параметрами блока	Без параметров блока безусловный вызов и вызов по условию

CALL FB 10 (
` In1 := Number1;`
` In2 := Number2;`
` **Ret_Val** := Number3);`
	CALL FB 10 (
	`  In1 := Number1;`
	`  In2 := Number2;`
	`  Out := Number3);`
		UC FB 11;
		CC FB 11;

Операторы завершения блока		
Условное завершение обработки блока	Безусловное завершение обработки блока	Конец блока
BEC	**BEU**	**BE**

Кодовые блоки вызываются для обработки с помощью оператора CALL. Вы можете передавать данные в вызываемый блок и можете также получать данные от вызываемого блока. Такая передача данных выполняется с помощью параметров блока. С оператором CALL в вызываемый блок пересылаются параметры блока, а также открывается экземплярный блок данных в случае вызова функционального блока. Если кодовые блоки не имеют параметров, то их можно вызвать посредством операторов UC или CC. Обработка блока прекращается посредством оператора конца блока.

18.1.1 Вызов блока: общая информация

Если кодовый блок должен быть обработан, он должен быть "вызван".

На рис. 18.1 показан пример вызова функции FC 10 в организационном блоке OB 1.

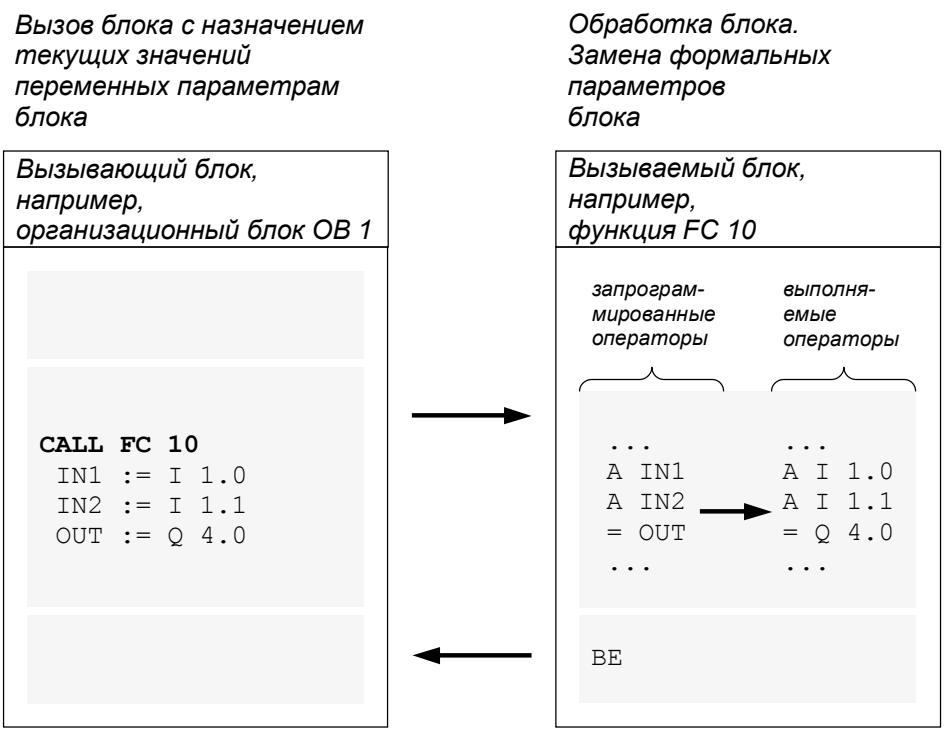


Рис. 18.1 Пример вызова блока

Вызов блока состоит из оператора вызова (в примере: CALL FC 10) и списка параметров. Если вызываемый блок не имеет параметров, инструкция вызова не будет иметь списка параметров. После выполнения оператора вызова CPU продолжает выполнение программы в вызываемом блоке (в примере: FC 10). Программа блока обрабатывается, пока не встретится оператор окончания блока. По окончании вызванного блока CPU возвращается к выполнению программы в вызывающем блоке (в примере: OB 1); выполнение этой программы продолжается со следующего оператора после оператора вызова блока. Если завершается выполнение программы организационного блока, CPU передает

управление операционной системе.

Информация, которая требуется CPU для возврата в вызывающий блок, сохраняется в стеке блоков (B-стек [B stack]). При каждом вызове блока в B-стеке генерируется новый элемент стека, который содержит адрес возврата, содержимое регистра данных и адрес стека локальных данных вызывающего блока. Если CPU переходить в состояние STOP в результате ошибки, Вы можете использовать программатор для того, чтобы увидеть в B-стеке информацию о том, какие блоки обрабатывались до момента возникновения ошибки.

Параметры блока являются интерфейсом данных для вызываемого блока. Рекомендуется избегать передачи данных посредством внутренних регистров (например, посредством аккумуляторов, адресных регистров, RLO), так как содержимое этих регистров может быть изменено при смене текущего блока (в результате "скрытых" операторов, инициированных редактором).

18.1.2 Оператор вызова блока CALL

С помощью оператора вызова блока CALL Вы можете вызывать функциональные блоки FB, функции FC, системные функциональные блоки SFB, системные функции SFC. Оператор вызова блока CALL является безусловным вызовом, что означает, что заданный блок всегда вызывается и обрабатывается, несмотря ни на какие условия. (Вы не можете вызывать организационные блоки; организационные блоки вызываются операционной системой, в зависимости от события).

Вызов функциональных блоков

Вы можете вызывать функциональные блоки FB, задав идентификатор блока после оператора CALL, и отделенный запятой идентификатор экземплярного блока данных, связанного с этим вызовом. Вы можете адресовать оба блока или абсолютным способом, или символьным. Назначение абсолютного адреса символьному адресу выполняется в таблице символов (Symbol Table) для экземплярного блока данных, имеющего связанный функциональный блок как тип данных.

В инструкции вызова может быть список параметров блока. При вводе исходного текста программы список параметров блока помещается между круглыми скобками; в списке параметры блока должны быть разделены запятыми.

При вызове функционального блока нет необходимости инициализировать все параметры вызываемого блока. Неинициализированные параметры блока сохраняют свои текущие значения. Если Вы не задаете никаких параметров, то скобки также не вводятся при вводе исходного текста программы.

Если Вы создаете функциональные блоки с атрибутом "мультиэкземплярные" ("multi-instance-capable"), Вы можете также вызывать эти блоки, как локальные экземпляры внутри других "мультиэкземплярных" функциональных блоков. Имеется в виду, что вызванный функциональный блок использует экземплярный блок данных вызывающего функционального блока для сохранения своих собственных локальных данных.

В таком случае Вы должны объявить локальный экземпляр в статических локальных данных вызывающего функционального блока, и после этого Вы можете вызывать функциональный блок в программе (без определения экземплярного блока данных). Локальный экземпляр трактуется как сложный тип данных внутри функционального блока "верхнего уровня". Более подробно материал изложен в разделе 18.1.6 "Статические локальные данные".

Вызов функций

Вы можете вызывать функции FC, задав идентификатор функции после оператора CALL, или абсолютным способом, или символьным. Далее в инструкции вызова следует список параметров. При вводе исходного текста программы список параметров блока помещается между круглыми скобками; в списке параметры блока должны быть разделены запятыми.

При вызове функции Вы должны инициализировать все параметры; тем не менее, параметры могут следовать в любом порядке. Вызываемые функции с функциональным значением имеют точно такую же форму, как и функции без функционального значения. Единственный выходной параметр, соответствующий функциональному значению, имеет имя RET_VAL.

Вызов системных блоков

Операционная система CPU содержит системные функции SFC и системные функциональные блоки SFB, доступные пользователю. Число и тип системных блоков определяется типом CPU. Все системные блоки вызываются оператором CALL.

Вы можете вызывать системные функциональные блоки таким же образом, как и те блоки, которые Вы можете написать сами; Вы должны расположить связанный экземплярный блок данных в пользовательской памяти (user memory) с типом данных SFB. Вы можете также вызывать системные функции таким же образом, как и функции, которые Вы можете написать сами.

Системные блоки доступны только в операционной системе CPU. При вызове системных блоков во время программирования в автономном (offline) режиме, редактору требуется описание интерфейса вызова для того, чтобы было возможно инициализировать параметры. Описание интерфейса расположено в стандартной библиотеке *Standard library* в системных функциональных блоках *System Function Blocks*. Отсюда редактор копирует описание интерфейса в папку (раздел) автономного режима "Blocks", когда Вы вызываете системный блок. После этого скопированное описание интерфейса вызова появляется как "нормальный" объект блока.

18.1.3 Операторы вызова UC и CC

Вы можете вызывать функциональные блоки и функции с помощью операторов вызова UC и CC. При этом требуется, чтобы вызываемые функции не имели параметров блоков и вызываемые функциональные блоки не имели экземплярного блока данных и, следовательно, не имели параметров блоков и статических локальных данных. Тем не менее, редактор не проверяет, выполняются ли эти условия.

Для вызова Вы можете использовать операторы UC и CC, если блок слишком большой и недостаточно удобен для понимания. При этом блок

просто разбивается на отдельные фрагменты, после чего эти фрагменты вызываются последовательно. Операторы вызова UC и CC не различают функции FC и функциональные блоки FB. Оба типа блоков обрабатываются одинаковым образом.

Оператор вызова UC является безусловным оператором, что означает, что UC вызывает блок вне зависимости от любых условий.

Оператор вызова CC является условным оператором, что означает, что CC вызывает блок только при условии, что результат логической операции (RLO) равен "1". Если RLO равен "0", то блок не вызывается, но RLO устанавливается в "1". После этого выполняется следующий за операцией вызова оператор.

Влияние на индикаторные биты (биты условного кода "condition code"): бит состояния OS сбрасывается при смене текущего блока; на биты состояния CC0, CC1 и OV смена блока не влияет, тогда как бит состояния /FC при смене текущего блока сбрасывается, что означает, что новая логическая операция начинается с операции первичного опроса в новом блоке или следует за вызовом блока.

Влияние смены текущего блока на "стек скобок" ("binary nesting stack"): Вы можете вызывать кодовый блок внутри "двоичного выражения вложения". Текущая глубина стека скобок не изменяется при смене блока. Возможная глубина стека в блоке, который может быть вызван внутри двоичного вложения (binary nest), следовательно, равен разности между максимально возможной глубиной и текущей глубиной вложения при вызове блока.

Влияние смены текущего блока на главное управляющее реле MCR: MCR-зависимость деактивируется при вызове блока. MCR выключается в вызванном блоке, независимо от того, было ли включено или не было включено MCR перед вызовом блока. При окончании обработки вызванного блока MCR-зависимость имеет то состояние, в котором она находилась до вызова блока.

Влияние смены текущего блока на аккумуляторы и адресные регистры: содержимое аккумуляторов и адресных регистров не изменяется при вызове блока с помощью операторов вызова UC и CC.

Влияние смены текущего блока на блоки данных: при вызове блока регистр блока данных сохраняется в B-стеке; оператор окончания блока сохраняет его содержимое при завершении обработки вызванного блока. Блок глобальных данных и экземплярный блок, бывшие текущими перед вызовом блока также открываются после операции вызова блока. Если перед вызовом блока нет открытого блока данных (например, нет экземплярного блока данных в OB 1), то не будет также открытих блоков данных после операции вызова блока, независимо от того, какие блоки данных открыты в вызванном блоке.

Дополнительные возможности:

- косвенная адресация вызовов блоков FB и FC с помощью операторов вызова UC и CC;
- организация вызова с параметрами блока с помощью оператора вызова UC;

- организация вызова с параметрами блока с помощью оператора вызова CC также и в функциональных блоках.

18.1.4 Функции окончания блока (Block End Functions)

Оператор BEC завершает обработку программы в блоке, при этом зависит от состояния RLO, а операторы BEU и BE заканчивают блок независимо от условий.

Оператор завершения блока по условию BEC

Выполнение оператора BEC, завершающего обработку программы в блоке, зависит от состояния RLO. Если результат логической операции RLO = "1" при обработке оператора BEC, то функция окончания блока выполняется и обрабатываемый блок закрывается. При этом выполняется переход в ранее обрабатывавшийся блок, из которого был сделан вызов только что завершенного блока.

Если RLO = "0" при обработке оператора BEC, то функция окончания блока не выполняется. При этом CPU устанавливает RLO в состояние "1" и выполняется следующая за оператором BEC инструкция. Следующий запрограммированный оператор проверки (опроса) в любом случае является первичным опросом.

Оператор безусловного завершения блока BEU

Выполнение оператора BEU не зависит ни от каких условий; оператор BEU выполняется и обрабатываемый блок закрывается. При этом выполняется переход в ранее обрабатывавшийся блок, из которого был сделан вызов только что завершенного блока.

В отличие от оператора BE оператор BEU может быть использован внутри одного блока несколько раз. При этом отдельные части программы, следующие за оператором BEU, могут быть обработаны только в случае выполнения операции перехода к ним.

Оператор безусловного завершения блока BE

Выполнение оператора BE не зависит ни от каких условий; оператор BE выполняется и обрабатываемый блок закрывается. При этом выполняется переход в ранее обрабатывавшийся блок, из которого был сделан вызов только что завершенного блока.

Оператор BE всегда является последним оператором в блоке.

Использование оператора BE является предметом выбора. При "инкрементном" программировании Вы завершаете программирование блока, закрыв блок; при программировании, ориентированном на ввод исходного текста программы, конец блока отмечается ключевым словом, например, END_FUNCTION_BLOCK вместо оператора BE.

18.1.5 Временные локальные данные

Временные локальные данные используются для промежуточного

хранения результатов, получающихся при обработке программы блока. Временные локальные данные доступны только во время обработки блока; после завершения блока эти данные теряются.

Временные локальные данные соответствуют адресам, которые расположены в стеке локальных данных (в L-стеке) в системной памяти CPU. Операционная система CPU обеспечивает размещение временных локальных данных для каждого кодового блока при вызове этого блока. Первоначально при вызове блока значения в L-стеке носят "псевдослучайный" характер. Для того, чтобы эти значения отвечали реалиям, до того, как они будут считаны, необходимо их записать. После завершения обработки блока, L-стек назначается следующему вызываемому блоку.

Объем памяти, требуемый для размещения локальных данных для блока, указывается в заголовке блока. Таким образом операционная система узнает, сколько байт в L-стеке необходимо отвести для вызываемого блока. Вы также можете узнать из заголовка блока, сколько байт локальных данных требуется для блока (если блок открыт, в редакторе с помощью опций меню: *File -> Properties (Файл -> Свойства),* или в оболочке SIMATIC Manager при выбранном блоке с помощью опций меню: *Edit -> Object Properties (Правка -> Свойства объекта),* информация и в том и в другом случае находится на вкладке "General - Part 2" ("Общие - Часть 2")).

Объявление временных локальных данных

Объявление временных локальных данных производится в разделе объявлений кодового блока:

- при инкрементном программировании временные локальные данные помечаются в столбце "Declaration" ("Объявление") словом "temp" (временный);
- при программировании, ориентированном на создание программы путем ввода исходного текста, временные локальные данные объявляются между ключевыми словами: VAR_TEMP и END_VAR.

На рис. 18.2 представлен пример объявления временных локальных данных. В примере переменная *temp1*, расположенная во временных локальных данных относится к типу INT, а переменная *temp2* относится к типу REAL.

Временные локальные данные сохраняются в L-стеке в порядке их объявления в соответствии с типами данных.

Более подробно материал о хранении данных в L-стеке изложен в разделе 26.2 "Хранение переменных".

Символьная адресация временных локальных данных

Адресацию временных локальных данных можно производить с использованием символьных имен. Для этого данным должны быть назначены имена в соответствии с правилами, принятыми для символьных имен (символов) локальных для блока данных.

Все операции, которые действительны для меркеров, разрешены для временных локальных данных. Тем не менее, надо отметить, что биты из временных локальных данных непригодны для использования в качестве

меркеров фронта, так как они не сохраняют состояние своего сигнала после окончания обработки блока.

Инкрементное программирование

Address (адрес)	Declaration (объявление)	Name (имя)	Type (тип)	Initial value (начальное значение)
0.0	in	In	INT	0
	out			
	inout			
2.0	stat	Total	INT	0
0.0 2.0	temp temp	temp1 temp2	INT REAL	

Программирование, ориентирование на создание исходных текстов программы

```
VAR_INPUT
  In   : INT := 0;
END_VAR
VAR_OUTPUT ... END_VAR
VAR_IN_OUT ... END_VAR
VAR
  Total : INT := 0;
END_VAR
VAR_TEMP
  Temp1 : INT;
  Temp2 : REAL;
END_VAR
```

Рис. 18.2 Пример объявления локальных данных в функциональном блоке.

Вы можете получить доступ к временным локальным данным блока только непосредственно в самом блоке. (Исключение: к временным локальным данным вызывающего блока можно получить доступ через параметры блока.)

Размер L-стека

Предельный размер для L-стека определяется типом CPU. Количество байтов временных локальных данных доступное в приоритетном классе, к которому относится программа организационного блока, также фиксировано. Для S7-300 размер L-стека фиксирован, например, 256 байт на приоритетный класс для CPU 314. При работе с S7-400 размер L-стека может настраиваться пользователем. Необходимое число байт отводится для временных локальных данных при параметризации CPU. Эта область памяти должна быть общедоступной для блоков, вызов которых производятся из соответствующего организационного блока и для блоков, которые вызываются, в свою очередь, из этих блоков.

Необходимо заметить в этой связи, что редактор также использует временные локальные данные, например, при передаче параметров блока. Вы не видите эти временные локальные данные в интерфейсе программирования.

Стартовая информация (Start information)

Операционная система CPU передает стартовую информацию (start information) во временных локальных данных при вызове организационного блока. Эта стартовая информация занимает размер памяти 20 байт в каждом организационном блоке и имеет почти идентичную структуру в каждом блоке. В главах 20 "Главная программа", 21 "Управление прерываниями", 22 "Параметры перезапуска" и 23

"Обработка ошибок" описывается назначение стартовой информации для отдельных организационных блоков.

Упомянутые 20 байт стартовой информации должны быть всегда доступны в каждом используемом приоритетном классе. Если Вы программируете обработку синхронных ошибок (ошибки программирования и доступа), необходимо обеспечить дополнительные 20 байт, по крайней мере, для стартовой информации этих организационных блоков обработки ошибок, так как эти OB ошибок должны обрабатываться в том же приоритетном классе.

Вы должны объявить эту стартовую информацию, при программировании организационного блока. Это обязательное условие. В стандартной библиотеке *Standard Library* имеются шаблоны для объявления на английском языке в *Organization Blocks (Организационные блоки).* Если Вам не требуется стартовая информация, то достаточно объявить первые 20 байт, например, как поле (как показано на рис. 18.3).

Инкрементное программирование

Address (адрес)	Declaration (объявление)	Name (имя)	Type (тип)	Initial value (начальное значение)
0.0	temp	SINFO	INT	ARRAY [1..20]
*1.0	temp			BYTE
20.0	temp	Lbyte	INT	ARRAY [1..16]
*1.0	temp		BYTE	BYTE

Программирование, ориентирование на создание исходных текстов программы

```
VAR_TEMP
  SINFO : ARRAY[1..20]
  OF BYTE;
  Lbyte : ARRAY[1..16]
  OF BYTE;
END_VAR
```

Рис. 18.3 Пример объявления локальных данных в организационном блоке.

Абсолютная адресация временных локальных данных

Обычно адресация временных локальных данных производится с использованием символьных имен, а абсолютная адресация используется в исключительных случаях. Если Вы знакомы с тем, как хранятся данные в L-стеке, Вы можете выработать для себя способ адресации статических локальных данных. Вы также можете посмотреть адреса в таблице объявления переменных в скомпилированном блоке.

Идентификатором адреса для временных локальных данных является L; при этом биты адресуются идентификатором L, байты - с помощью идентификатора LB, машинные слова - с помощью идентификатора LW, а двойные слова - с помощью идентификатора LD.

Пример:

Для абсолютной адресации Вы планируете использовать 16 байтов временных локальных данных, к отдельным значениям которых в дальнейшем Вы желаете иметь доступ как в формате байта, так и побитно. Создайте данную область как поле прямо с начала области

локальных данных с начальным адресом, равным 0.

В организационном блоке Вы должны поместить объявление этого поля сразу же за объявлением поля для стартовой информации, таким образом, адресация Вашего поля должна начинаться с адреса 20.

Примечание:

Абсолютная адресация временных локальных данных возможна только при использовании базовых языков программирования STL, LAD и FBD. При использовании языка SCL адресация временных локальных данных возможна только символьным способом.

В главе 26 "Прямой доступ к переменным" показано, как узнать адреса переменных во временных локальных данных во время выполнения программы.

Тип данных ANY

Переменные временных локальных данных могут быть объявлены, как исключение, с типом ANY.

При использовании языка STL Вы имеете возможность сгенерировать указатель типа ANY, который может изменяться во время выполнения программы. Более подробную информацию Вы можете найти в разделе 26.3.3 ""Переменный" указатель ANY".

При использовании языка SCL Вы можете временной переменной типа ANY назначать адрес другой (сложной) переменной во время выполнения программы. Более подробную информацию Вы можете найти в разделе 29.2.4 "Временные локальные данные".

18.1.6 Статические локальные данные

Статические локальные данные - это те адреса, которые функциональный блок сохраняет в своем экземплярном блоке данных.

Статические локальные данные - это "память" функционального блока. Эти данные сохраняются до тех пор, пока программа не изменит их, так же как адреса данных в блоках глобальных данных.

Объем памяти, требуемый для размещения статических локальных данных, ограничивается типом данных используемых переменных и максимальным размером блока, определяемым типом используемого CPU.

Объявление статических локальных данных

Объявление статических локальных данных производится в разделе объявлений функционального блока:

- при инкрементном программировании статические локальные данные помечаются в столбце "Declaration" ("Объявление") словом "stat" (статический);
- при программировании, ориентированном на создание программы путем ввода исходного текста, статические локальные данные объявляются между ключевыми словами: VAR и END_VAR.

На рис. 18.2 в разделе 18.1.5 "Временные локальные данные" представлен пример объявления переменных в функциональном блоке. Сначала объявляются параметры блока, затем объявляются статические локальные данные и, наконец, временные локальные данные.

Статические локальные данные сохраняются в экземплярном блоке после параметров блока в порядке их объявления и в соответствии с типами данных.

Более подробно материал о хранении данных в блоках данных изложен в разделе 26.2 "Хранение переменных".

Символьная адресация статических локальных данных

Адресацию статических локальных данных можно производить с использованием символьных имен. Для этого данным должны быть назначены имена в соответствии с правилами, принятыми для символьных имен (символов) локальных для блока данных.

Имеется возможность доступа к статическим локальным данным с помощью тех операций, которые действительны для использования в сочетании с адресами данных в блоках глобальных данных.

Пример:

Функциональный блок "Totalizer" прибавляет входное значение (input value) к значению, сохраненному в статических локальных данных, (stored value) и после этого сохраняет сумму опять в статических локальных данных. При следующем вызове блока новое входное значение вновь будет прибавлено к сохраненному в статических локальных данных значению, после чего результат вновь будет сохранен, и так далее (см. рис. 18.4. (вверху)).

Total - это переменная в блоке данных "TotalizerData", который является экземплярным блоком данных для функционального блока "Totalizer" (Вы можете самостоятельно задавать имена всех блоков в таблице символов Symbol Table в соответствии с правилами). Экземплярный блок данных имеет структуру данных функционального блока. В данном примере экземплярный блок данных содержит две переменные формата INT с именами *In* и *Total.*

Доступ к статическим локальным данным со стороны функционального блока

Статические локальные данные обычно обрабатываются только в функциональном блоке. Тем не менее, хранятся они в блоке данных. блока данных. Вы можете получить доступ к статическим локальным данным в любое время тем же образом, каким Вы имеете доступ к переменным в блоке глобальных данных с помощью следующей формы адресации: *"ИмяБлокаДанных"."ИдентификаторАдреса".*

В нашем маленьком примере блок данных имеет имя *"TotalizerData",* а переменная адресуется как *Total*. Доступ к переменной может иметь следующую форму:

```
L   "TotalizerData".Total;
T   MW 20;
L   0;
T   "TotalizerData".Total;
```

FB "Totalizer" ("Сумматор")

Address адрес	Declaration объявление	Name имя	Type тип
+0.0	in	In	INT
+2.0	stat	Total	INT

```
      L    #In;
      L    #Total;
      +I   ;
      T    #Total;
```

DB "TotalizerData"

Address адрес	Declaration объявление	Name имя	Type тип
+0.0	in	In	INT
+2.0	stat	Total	INT

FB "Evaluation" ("Опрос")

Address адрес	Declaration объявление	Name имя	Type тип
0.0	in	Add	BOOL
0.1	in	Delete	BOOL
2.0	stat	EM_Add	BOOL
2.1	stat	EM_Del	BOOL
4.0	stat	Memory	Totalizer

```
      A    #Add;
      FP   #EM_Add;
      JCN  M1;
      CALL #Memory
       (In := "Value2");
M1:   A    #Delete;
      FP   #EM_Del;
      JCN  End;
      L    #Memory.Total;
      T    "Result";
      L    0;
      T    #Memory.Total;
```

DB "EvaluationData"

Address адрес	Declaration объявление	Name имя	Type тип
0.0	in	Add	BOOL
0.1	in	Delete	BOOL
2.0	stat	EM_Add	BOOL
2.1	stat	EM_Del	BOOL
4.0	stat:in	Memory.In	INT
6.0	stat	Memory. total	INT

*Выбрав опцию меню: **Data view** (Просмотр данных) для блока данных, можно просмотреть отдельные переменные с их полными именами.*

Одновременно Вы можете видеть соответствующие абсолютные адреса.

Рис. 18.4 Пример статических локальных данных и локальных экземпляров.

Локальные экземпляры

При вызове функционального блока обычно для вызова назначается также экземплярный блок. Функциональный блок сохраняет свои параметры и свои статические локальные данные в этом экземплярном блоке.

Начиная с STEP 7 V2, Вы можете создавать "мультиэкземпляры", что означает, что Вы можете вызывать одни функциональные блоки в других функциональных блоках. Статические локальные данные (и параметры блока) вызванного функционального блока являются подмножеством статических локальных данных вызывающего блока. Для этого требуется, чтобы вызываемый и вызывающий функциональные блоки имели версию 2, чтобы они могли быть обработаны в режиме "мультиэкземпляра". Таким путем можно организовывать вложение вызовов функциональных блоков с глубиной вложения до 8 вызовов.

Пример (рис. 18.4 (внизу)): В статических локальных данных функционального блока "Evaluation" объявлена переменная *Memory*, которая соответствует функциональному блоку "Totalizer" и имеет такую же структуру. Теперь Вы можете вызывать функциональный блок "Totalizer" посредством переменной *Memory*, без определения блока данных, так как данные для *Memory* размещены внутриблочно ("block-local") в статических локальных данных (*Memory* является "локальным экземпляром" блока "Totalizer").

Доступ к статическим локальным данным *Memory* в программе функционального блока "Evaluation" выполняется таким же способом, как и к компонентам структуры с определением имени структуры (*Memory*) и имени компонента (*Total*).

Экземплярный блок данных "EvaluationData", следовательно, содержит переменные *Memory.In* и *Memory.Total*, к которым Вы можете также обращаться как к глобальным переменным, например, следующим образом: *"EvaluationData".Memory.Total*.

Вы можете найти пример использования локального экземпляра в функциональных блоках FB 6, 7 и 8 в программе "Program Flow Control" на прилагаемой дискете. Пример в разделе 19.5.3 "Пример установки" содержит дополнительные варианты применения локальных экземпляров.

Абсолютная адресация статических локальных данных

Обычно адресация статических локальных данных производится с использованием символьных имен, а абсолютная адресация используется в исключительных случаях. Внутри функционального блока экземплярный блок данных открывается посредством регистра DI. Идентификатором для адресов в блоке данных, для статических локальных данных также как параметров блока является идентификатор DI. Биты адресуются идентификатором DIX, байты - с помощью идентификатора DIB, слова - с помощью идентификатора DIW, а двойные слова - с помощью идентификатора DID.

Если Вы знакомы с тем, как хранятся данные блоке данных, Вы можете выработать для себя способ адресации статических локальных данных. Вы также можете посмотреть адреса в таблице объявления переменных в скомпилированном блоке. Но будьте внимательны! *Эти адреса связаны с запуском экземпляра.* Они действительны только в случае, если Вы вызываете функциональный блок с блоком данных. Если Вы вызываете

функциональный блок как локальный экземпляр, соответствующие локальные данные этого локального экземпляра располагаются внутри экземплярного блока данных вызывающего функционального блока. Вы можете просмотреть абсолютные адреса, например, в скомпилированном экземплярном блоке данных, в котором содержатся все локальные экземпляры. Для просмотра адресов отдельных локальных данных выберите опции меню: *Select View -> Data View (Выбор просмотра -> Просмотр данных).*

Вернемся к нашему примеру. В функциональном блоке "Totalizer" переменная *Total* может быть адресована с помощью DIW 2, если FB "Totalizer" вызывается с блоком данных (см. назначение адресов в DB "TotalizerData"), и с помощью DIW 6, если FB "Totalizer" вызывается как локальный экземпляр в блоке FB "Evaluation" (см. назначение адресов в DB "EvaluationData").

Тем не менее, при программировании функционального блока, если пока неизвестно, будет ли он вызываться с блоком данных или будет вызываться как локальный экземпляр, то возникает вопрос: как в этом случае назначать абсолютные адреса статическим локальным данным? Коротко говоря, в этом случае к адресу переменной прибавляется смещение локального экземпляра из адресного регистра AR2. (См. главу 25 "Косвенная адресация" и главу 26 "Прямой доступ к переменным" для получения подробной информации по данному вопросу).

Примечание:

Абсолютная адресация статических локальных данных возможна только при использовании базовых языков программирования STL, LAD и FBD. При использовании языка SCL адресация статических локальных данных возможна только символьным способом.

18.2 Функции для блоков данных

Вы можете хранить данные Вашей программы в блоках данных. В принципе, для хранения данных Вы можете также использовать область меркеров (bit memory area). Тем не менее, используя блоки данных, Вы имеете значительно больше возможностей, в плане объема данных, структурирования данных и типов данных.

В данной главе будет показано

- как работать с адресацией данных,
- как вызывать блоки данных,
- как создавать, удалять и тестировать блоки данных при выполнении программы.

Существует возможность использовать блоки двух видов: *блоки глобальных данных (global data blocks),* которые не назначаются никакому кодовому блоку, и *экземплярные блоки (instance data blocks)*, которые назначаются функциональным блокам. Данные блоков глобальных данных являются, попросту говоря, "свободными" данными, которые может использовать каждый кодовый блок. Вы сами определяете объем, который занимают эти данные, их структуру непосредственно при программировании блоков глобальных данных.

Экземплярные блоки данных содержат только те данные, которые использует связанный функциональный блок; этот функциональный блок определяет структуру данных и место хранения данных в "своем" экземплярном блоке данных.

Число и размер блоков данных определяются типом CPU. Нумерация блоков данных начинается с 1; не может существовать блока данных с именем DB 0. Вы можете использовать каждый блок данных или как блок глобальных данных или как экземплярный блок данных.

Вы должны сначала создать блоки данных, которые Вы будете использовать в Вашей программе или запрограммировав их, так же как кодовые блоки, или используя системную функцию SFC 22 CREAT_DB при выполнении программы.

Блоки данных должны быть сохранены в рабочей (work) памяти для того, чтобы к ним был возможен доступ чтения/записи из пользовательской программы. Вы можете также оставить блоки данных в загрузочной (load) памяти, используя атрибут блока "Unlinked" (ключевое слово UNLINKED используется при создании программы путем ввода исходного текста программы).

Такие блоки данных не занимают место в рабочей (work) памяти. Тем не менее, Вы сможете только считывать блоки данных в загрузочной (load) памяти с помощью системной функции SFC 20 BLCMOV. Такая процедура пригодна для блоков данных с данными параметризации или технологическими (recipe) данными, которые требуются относительно редко для управления установкой или процессом.

Если Вы установили атрибут "The data block is writeprotected in the programmable controller" ("Блок данных защищен от записи в PLC") в свойствах блока (что соответствует использованию ключевого слова READ_ONLY при создании программы путем ввода исходного текста программы), в дальнейшем Вы сможете только считывать данные из этого DB.

18.2.1 Два регистра блоков данных

В CPU есть два "регистра блоков данных" (data block register) для обработки адресованных данных. В этих регистрах содержатся номера обрабатываемых в текущем времени блоков; имеются в виду блоки, в которых находятся обрабатываемые в настоящий момент данные. Перед тем, как получить доступ к адресу, содержащемуся в блоке данных, Вы должны сначала открыть этот блок данных. Если Вы при адресации данных указываете полный их адрес (о способах спецификации блоков данных см. ниже), то Вам нет необходимости заботиться об открытии блока данных и об использовании регистров блоков данных. Редактор сам создаст необходимые инструкции из Ваших данных.

Редактор использует один из регистров блоков данных преимущественно для доступа к блокам глобальных данных, а второй регистр блоков данных используется для доступа к экземплярным блокам данных. В соответствии с назначением регистров, они имеют следующие идентификаторы: регистр для доступа к блокам глобальных данных DB-регистр ("Global data block register") и регистр для доступа к экземплярным блокам данных DI-регистр ("Instance data block register").

Обработка данных регистров CPU происходит абсолютно одинаково. Каждый блок данных может быть открыт с помощью одного из двух регистров (или же с помощью двух регистров одновременно). При загрузке в аккумулятор слова данных, Вы должны задать, какой из двух возможных открытых блоков содержит это слово данных. Если обрабатываемый блок данных открыт с помощью DB-регистра, то слово данных должно содержать DBW; если обрабатываемый блок данных открыт с помощью DI-регистра, то слово данных должно содержать DIW. Остальные форматы данных также имеют соответствующие обозначения (см. табл. 18.2).

Таблица 18.2 Адресация данных

Адресуемые данные	Блок данных открыт с помощью	
	DB-регистра	DI-регистра
Бит данных	DBX y.x	DIX y.x
Байт банных	DBB y	DIB y
Слово данных	DBW y	DIW y
Двойное слово данных	DBD y	DID y

x = адрес бита,
y = адрес байта

18.2.2 Адресация данных

Вы можете использовать следующие способы организации доступа к данным:

- символьная адресация с указанием полного адреса,
- абсолютная адресация с указанием полного адреса,
- абсолютная адресация с указанием неполного адреса.

Дополнительную информацию по способам адресации Вы можете найти в главе 25 "Косвенная адресация".

Символьная адресация с указанием полного адреса глобальных данных в блоке данных требуют минимальной информации для системы. Для использования абсолютной адресации или для использовании сразу двух регистров блоков данных, Вам необходимо ознакомиться с материалом, изложенным ниже.

Символьная адресация данных

Автор рекомендует пользователям использовать символьную адресацию настолько широко, насколько это возможно.

Символьная адресация

- делает более простым чтение и понимание программы (если в качестве символьных имен используются осмысленные обозначения),

- уменьшает ошибки программирования (редактор сравнивает термы, использованные в таблице символов и в программе; "ошибки перестановки чисел", например, DBB 156 и DBB 165, которые могут происходить при использовании абсолютной адресации, не могут произойти при символьной адресации),
- не требует умения программирования на уровне машинных кодов (не требует знания, какой из блоков данных открыт CPU для обработки в настоящий момент).

При символьной адресации указывается полный адрес (указывается и идентификатор блока, и адрес данных), поэтому адрес данных всегда уникален.

Символьный адрес данных формируется в два этапа:

- Назначение блока данных в таблице символов.

  Блоки данных являются глобальными данными, имеющими уникальные адреса в программе. В таблице символов Вы должны назначить символьное имя (например, Motor1) абсолютному адресу блока данных (например, DB 51).

- Назначение адресов данных в блоке данных.

  Вы должны назначить символьные имена адресам данных (и типам данных) во время программирования блока данных. Имена назначаются только в связанном ("associated") блоке (такие имена являются "внутриблочными" - "block-local"). Вы можете назначать такие же имена другим переменным в другом блоке.

Доступ к адресам данных с указанием полного адреса

При организации доступа к адресам данных с указанием полного адреса Вы должны задавать адрес данных совместно с адресом блока. Такой способ указания адреса может быть использован как с символьными, так и с абсолютными адресами.

```
L   MOTOR1.ACTVAL;
L   DB 51.DBW 20;
```

MOTOR1 является символьным адресом, который Вы должны назначить блоку данных в таблице символов. ACTVAL - это адрес данных, который Вы назначаете во время программирования блока данных. Символьное имя MOTOR1.ACTVAL - это такое же уникальное символьное обозначение адреса данных, как и обозначение DB 51.DBW 20.

Доступ к данным посредством полного адреса возможен только при использовании вместе с регистром блоков глобальных данных (с DB-регистром). При адресации данных с указанием полного адреса редактор выполняет две операции: сначала открывает блок данных с помощью DB-регистра, а затем выполняет операцию доступа к адресам данных.

Вы можете использовать доступ к данным посредством полного адреса при всех разрешенных операциях с адресуемыми типами данных. К таким операциям относятся двоичные логические операции, операции с памятью для адресуемых битов, а также операции загрузки (load) и пересылки (transfer) для адресуемых численных данных. Также Вы можете задавать полный адрес для параметров блоков (настоятельно рекомендуется изучить материал главы 19 "Параметры блоков").

Абсолютная адресация данных

При организации доступа к адресам данных посредством абсолютной адресации Вы должны знать адреса, назначенные редактором для данных, при установке блока (setting up). Вы можете найти эти адреса, выведя их после программирования и компилирования блока данных. При этом в колонке адресов Вы увидите значения абсолютных адресов, с которых начинаются соответствующие переменные.

Такая процедура может быть выполнена для всех блоков данных - тех, которые Вы используете как блоки глобальных данных, и тех, которые Вы используете как экземплярные блоки данных. Таким образом Вы также можете видеть, где редактор хранир параметры блоков и статические локальные данные (в случае функциональных блоков).

Если необходимо вычислить адрес, соответствующую информацию Вы можете найти в разделе 26.2 "Хранение переменных".

Организация доступа к адресам данных, имеющих байтовый размер, одинакова, например, с доступом к меркерам; и в том, и в другом случае используются одинаковые операции (см. табл. 18.3), которые выполняются одинаковым способом.

Таблица 18.3 Операции с блоками данных

Оператор	Значение
A –	Проверка на состояние "1", комбинирование по логике AND (И) с адр.
O –	Проверка на состояние "1", комбинирование по логике OR (ИЛИ) с адр.
X –	Проверка на "1", комбинирование по логике Excl.OR (искл.ИЛИ) с адр.
AN –	Проверка на состояние "0", комбинирование по логике AND (И) с адр.
ON –	Проверка на состояние "0", комбинирование по логике OR (ИЛИ) с адр.
XN –	Проверка на "0", комбинирование по логике Excl.OR (искл.ИЛИ) с адр.
= –	Назначение для адр.
S –	Установка адр.
R –	Сброс адр.
FP –	Проверка наличия фронта (переднего) адр.
FN –	Проверка наличия фронта (заднего) адр.
DBXy.x	Адресация бита с помощью DB-регистра
DIXy.x	Адресация бита с помощью DI-регистра
DBz.DBXy.x	Адресация бита с указанием полного адреса
L –	Загрузка из адр.
T –	Пересылка в адр.
DBBy	Адресация байта с помощью DB-регистра
DBWy	Адресация слова с помощью DB-регистра
DBDy	Адресация двойного слова с помощью DB-регистра
DIBy	Адресация байта с помощью DI-регистра
DIWy	Адресация слова с помощью DI-регистра
DIDy	Адресация двойного слова с помощью DI-регистра
DBz.DBBy	Адресация байта с указанием полного адреса
DBz.DBWy	Адресация слова с указанием полного адреса
DBz.DBDy	Адресация двойного слова с указанием полного адреса

x = адрес бита, y = адрес байта, z = номер блока данных

Если Вы намереваетесь назначать исключительно абсолютные адреса данным в блоке данных, Вы должны зарезервировать требуемое количество байтов посредством объявления поля.

18.2.3 Открытие блока данных

OPN DBx	открытие блока данных с помощью DB-регистра с использованием абсолютной адресации;
OPN DB*name*	открытие блока данных с помощью DB-регистра с использованием символьной адресации;
OPN DIx	открытие блока данных с помощью DI-регистра с использованием абсолютной адресации;
OPN DI*name*	открытие блока данных с помощью DI-регистра с использованием символьной адресации;

Блоки данных открываются независимо от любых условий. Открытие блоков не действует ни на RLO, ни на содержимое аккумуляторов; глубина вложения вызовов блоков также не изменяется.

Открываемый блок должен быть в рабочей (work) памяти.

Пример:

Значение слова данных DBW 10 из блока данных DB 12 должно быть перенесено в слово данных DBW 12 из блока данных DB 13 (см. рис 18.5). Значение слов данных DBW 14 из блоков данных DB 12 и DB 13 должны быть сложены; сумма должна быть сохранена в слове данных DBW 14 из блока данных DB 14.

Вы можете запрограммировать данный пример следующими способами организации доступа к данным: с неполной адресацией и с полной адресацией данных (см. таблицу к рис. 18.5).

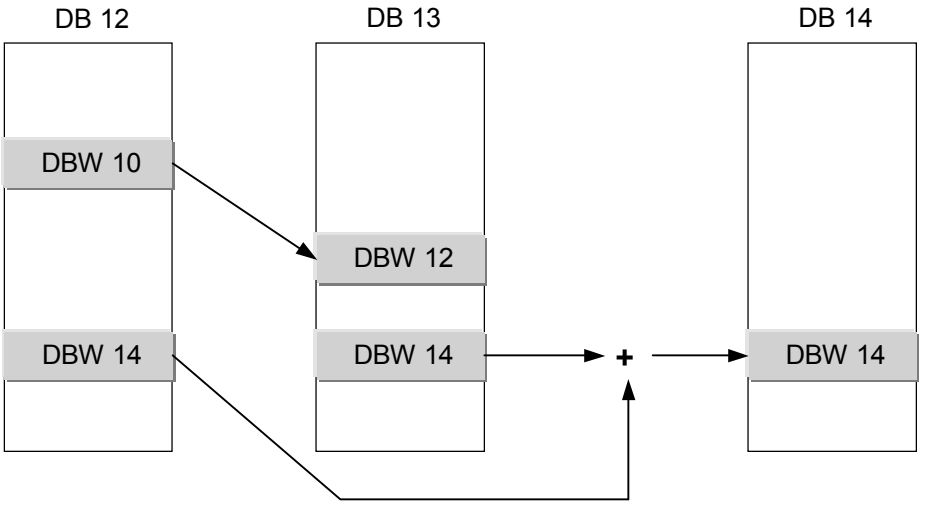


Рис. 18.5 Открытие блоков данных

Таблица к рис. 18.5

Программирование с неполной адресацией данных	Программирование с полной адресацией данных
OPN DB 12; L DBW 10; OPN DB 13; T DBW 12;	L DB 12.DBW 10; T DB 13.DBW 12;
OPN DB 12; L DBW 14; OPN DB 13; L DBW 14; +I ; OPN DB 14; T DBW 14;	L DB 12.DBW 14; L DB 13.DBW 14; +I ; T DB 14.DBW 14;

Когда блок данных открыт, он остается "доступным" ("valid"), до тех пор, пока не будет открыт другой блок данных. При определенных условиях незаметно для пользователя это может обеспечиваться редактором (см. "Особенности, имеющие место при адресации данных"). Например, вызов блока посредством оператора CALL с одновременной передачей параметра может изменять содержимое регистров блоков данных.

При смене обрабатываемого блока с помощью операторов UC и CC содержимое регистров блоков данных сохраняется. При возвращении в вызывающий блок оператор завершения блока восстанавливает содержимое регистров.

18.2.4 Обмен содержимым между регистрами блоков данных

CDB Обмен содержимым между регистрами блоков данных.

Оператор CDB позволяет выполнить обмен содержимым между регистрами блоков данных. Операция обмена данными выполняется независимо от любых условий и не действует ни на биты состояния, ни на другие регистры.

Пример:

С помощью оператора CDB Вы можете "обойти" вызов через регистр DB посредством регистра DI; блок данных пересылается как параметр блока (что невозможно выполнить напрямую).

```
CDB  ;
OPN  #Data2;
CDB  ;
```

В примере с помощью оператора CDB содержимое DB-регистра пересылается в DI-регистр. Затем с помощью параметра блока `#Data2`

открывается блок данных, пересланный как фактический параметр; т.е., номер блока записан в DB-регистр. После нового обмена содержимого регистров, старое значение вновь возвращается в DB-регистр, а DI-регистр содержит параметризированного блока данных.

18.2.5 Размер блока данных и его номер

L	DBLG	Загрузка (Load) размера блока данных, открытого с помощью DB-регистра;
L	DBNO	Загрузка (Load) номера блока данных, открытого с помощью DB-регистра;
L	DILG	Загрузка (Load) размера блока данных, открытого с помощью DB-регистра;
L	DINO	Загрузка (Load) номера блока данных, открытого с помощью DB-регистра;

Инструкция L DBLG загружает (load) размер блока данных, открытого с помощью DB-регистра в аккумулятор accumulator 1. Этот размер представляет собой определенное количество байтов. Инструкция L DILG выполняет аналогичную операцию, только для блока данных, открытого с помощью DI-регистра.

Инструкция L DBNO загружает (load) номер блока данных, открытого с помощью DB-регистра в аккумулятор accumulator 1. Инструкция L DINO выполняет аналогичную операцию, только для блока данных, открытого с помощью DI-регистра.

При выполнении данных операций по внесению номера и размера блока данных в аккумулятор accumulator 1, его предыдущее содержимое пересылается в аккумулятор accumulator 2, как при "нормальном" выполнении операции загрузки (load). Если перед операцией загрузки (load) номера или размера блока данных в аккумулятор не было открыто ни одного блока данных, то в результате в аккумулятор accumulator 1 будут загружены нулевые (0) значения соответственно в качестве номера или размера блока данных.

Невозможно выполнить запись номера блока в регистр блоков данных прямой операцией загрузки; Вы можете влиять на содержимое регистра блоков данных только посредством операций OPN DB, OPN DI и CDB (обмен содержимым между регистрами блоков данных).

18.2.6 Особенности, имеющие место при адресации данных

Изменение назначения в DB-регистре

С помощью следующих функций редактор создает дополнительные операторы, которые могут влиять на содержимое одного или двух регистров блоков данных:

Использование полной адресации данных

Каждый раз, когда при организации доступа к данным используется полная адресация, редактор сначала открывает блок данных с помощью операции OPN DB, затем обеспечивает доступ к адресованным данным. При этом каждый раз производится обновление (переписывание) содержимого DB-регистра. Это же происходит также при инициализации параметров блоков с использованием полной адресации данных.

Операция доступа к параметрам блока

Содержимое DB-регистра также меняется при операциях доступа к следующим параметрам блоков: для функций, для всех параметров блоков сложных типов и функциональных блоков, входных/выходных параметров сложных типов данных.

Операция вызова блока CALL FB

Перед тем как выполнить вызов блока, инструкция CALL FB инициирует сохранение номера текущего экземплярного блока данных в DB-регистре (с помощью обмена содержимого в регистрах блоков данных) и открывает экземплярный блок данных для вызванного функционального блока. Таким образом, связанный экземплярный блок данных всегда открыт в вызванном функциональном блоке. Последующие вызовы блока с помощью инструкции CALL FB вновь изменяют содержимое DB-регистра так, что текущий экземплярный блок данных снова доступен для вызванного функционального блока. Таким образом инструкции CALL FB изменяют содержимое DB-регистра.

Значение DI-регистра для функциональных блоков

Для функциональных блоков данных DI-регистр всегда содержит номер соответствующего экземплярного блока данных. Все операции доступа к параметрам блока или к статистическим локальным данным выполняются с помощью DI-регистра и также посредством адресного регистра AR2 в случае "мультиэкземплярных" функциональных блоков.

Примечание: необходимо учитывать указанное выше постоянное назначение DI-регистра, если Вы изменяете содержимое DI-регистра посредством операций CDB или OPN DI.

Если, к примеру, Вы желаете использовать оба регистра блоков данных одновременно для замены данных, то Вы должны сначала сохранить содержимое регистров, чтобы в последствии восстановить эти значения. Пример, представленный на рис. 18.6, показывает применение соответствующего приема программирования.

Внесение изменений в назначения блоков данных на поздних стадиях

На вкладке "Blocks" ("Блоки") в окне свойств для папки автономных объектов *Blocks* Вы можете определить, какой из способов адресации данных будет иметь преимущество: абсолютная или символьная - при выполнении изменений в назначениях блока данных для уже сохраненных кодовых блоков.

По умолчанию действует установка "Absolute address has priority" ("Абсолютная адресация имеет приоритет") (такие же параметры, как и в предыдущих версиях STEP 7). Такая, принятая по умолчанию, установка означает, что при изменении в разделе объявлений в блоке, абсолютный адрес сохраняется в программе, а символьный адрес соответственно

изменяется. При задании установки "Symbol has priority" ("Символьная адресация имеет приоритет") означает, что при изменении в разделе объявлений в блоке, символьный адрес сохраняется в программе, а абсолютный адрес соответственно изменяется.

```
Var_Temp
   ZW_DB : WORD;  //Промежуточный буфер для блока гло-
                  //бальных данных
   ZW_DI : WORD;  //Промежуточный буфер для экземпярного
                  //блока данных
END_VAR
//Сохранение регистров блоков данных
   L  DBNO;       //Буфер для № блока глобальных данных
   T  ZW_DB;
   L  DINO;       //Буфер для № экземплярн. блока данных
   T  ZW_DI;
//При использовании неполной адресации данных и обоих
//регистров блоков данных
   OPN  DB 12;    //Открыть DB 12 с помощью DB-регистра
   OPN  DI 13;    //Открыть DB 13 с помощью DI-регистра
   L    DBW 16;   //##########
   T    DIW 28;   //# Будьте осторожны с символьной адр.
   L    DID 30;   //# в этой части программы, напр. при
   L    DBD 30;   //# использовании параметров блока,
   +R   ;         //# "внутриблочных" переменных и при
                  //# полной адресации данных
   T    DID 26;   //##########
//Восстановление регистров блоков данных
   OPN  DB[ZW_DB];//Открыть исходный блок глобальных
                  //данных
   OPN  DI[ZW_DI];//Открыть исходный экземплярный блок
```

Рис. 18.6 Пример непосредственного использования двух регистров блоков данных

Пример:

В блоке данных DB 1 слово данных DBW 10 назначено символьному имени *Actual_value*. Если "Data" является символьным именем для блока данных DB 1, то в программе Вы можете загрузить данное слово, например, посредством операции:

```
L    "Data".Actual_value      DB1.DBW 10
```

Если теперь Вы добавите дополнительное слово данных с помощью символьного имени *MaxCurrent* сразу перед словом данных DBW 10, тогда программа будет иметь следующее содержание при последующем открытии (и сохранении) кодового блока:

если "Absolute address has priority" ("Абсолютная адресация имеет приоритет"):

```
L    "Data".MaxCurrent        DB1.DBW 10
```

если "Symbol has priority" ("Символьная адресация имеет приоритет"):

```
L    "Data".Actual_value      DB1.DBW 12
```

Такие же условия имеют место при организации доступа к адресам в блоках глобальных данных (например, доступ к адресам входов), если для этих адресов назначены символьные имена в таблице символов. Подробную информацию по данному вопросу Вы можете найти в разделе 2.5.6 "Приоритет адресов".

18.3 Системные функции для блоков данных

Существуют три системные функции для обработки блоков данных. Параметры этих функций приведены в таблице 18.4.

- SFC 22 CREAT_DB
  С помощью этой функции выполняется создание блока данных
- SFC 23 DEL_DB
  С помощью этой функции выполняется удаление блока данных
- SFC 24 TEST_DB
  С помощью этой функции выполняется тестирование блока данных

Таблица 18.4 Параметры системных функций для обработки блоков данных

SFC	Имя	Объявл.	Тип	Назначение, описание
22	LOW_LIMIT	INPUT	WORD	Нижн. граница для № создаваемого DB
	UP_LIMIT	INPUT	WORD	Верхн. граница для № создаваемого DB
	COUNT	INPUT	WORD	Длина DB в байтах (четное число)
	RET_VAL	OUTPUT	INT	Информация об ошибках
	DB_NUMBER	OUTPUT	WORD	Номер созданного блока данных
23	DB_NUMBER	INPUT	WORD	Номер удаляемого блока данных
	RET_VAL	OUTPUT	INT	Информация об ошибках
24	DB_NUMBER	INPUT	WORD	Номер проверяемого блока данных
	RET_VAL	OUTPUT	INT	Информация об ошибках
	DB_LENGTH	OUTPUT	WORD	Длина DB в байтах
	WRITE_PROT	OUTPUT	BOOL	Если = "1", то DB защищен от записи

18.3.1 Создание блока данных

С помощью системной функции SFC 22 CREAT_DB выполняется создание блока данных в рабочей (work) памяти. В качестве номера блока данных системная функция принимает наименьшее свободное число в диапазоне чисел, заданных входными параметрами LOW_LIMIT (нижний предел) и UP_LIMIT (верхний предел). Граничные значения сами входят в диапазон возможных значений для номера блока. Если оба граничных значения одинаковы, то создаваемый блок будет иметь данное значение. Выходной параметр DB_NUMBER содержит фактический номер созданного блока данных. С помощью входного параметра COUNT Вы задаете размер (длину) создаваемого блока данных.

Длина блока данных соответствует числу байтов данных и при этом должна быть четным числом.

Создание блока данных и вызов блока данных не одно и то же. Текущий (обрабатываемый блок) остается доступным ("valid") блоком. Блок данных, созданный с помощью системной функции, содержит случайные данные. Для использования блока данных, созданного с помощью системной функции, необходимо сначала определить значения данных, содержащихся в блоке. Только после этого можно будет считывать эти данные в программе.

Если при выполнении системной функции происходит ошибка, то блок данных не создается, выходной параметр DB_NUMBER остается неопределенным, а номер ошибки выдается как значение функции.

18.3.2 Удаление блока данных

С помощью системной функции SFC 23 DEL_DB выполняется удаление блока данных в RAM памяти (рабочей [work] и загрузочной [load] памяти). Номер удаляемого блока данных определен в инструкции во входном параметре DB_NUMBER. При выполнении данной системной функции блок данных не должен быть открыт, иначе CPU перейдет в состояние STOP.

Блок данных, созданный с ключевым словом UNLINKED (несвязанный), а также блок данных, расположенный в модуле памяти FEPROM не может быть удален с помощью системной функции SFC 23.

Если при выполнении системной функции происходит ошибка, то блок данных не удаляется, а номер ошибки выдается как значение функции.

18.3.3 Тестирование блока данных

С помощью системной функции SFC 24 TEST_DB выдает число байтов для блока данных в рабочей (work) памяти в выходном параметре DB_LENGTH, а также ID защиты от записи в выходном параметре

WRITE_PROT. Номер тестируемого блока данных Вы задаете в параметре DB_NUMBER.

Если при выполнении системной функции происходит ошибка, то блок данных не создается, выходной параметр остается неопределенным, а номер ошибки выдается как значение функции.

18.4 Null-операции (нуль-операции)

Null-операции (нуль-операции) не создают никакого воздействия при выполнении программы. В языке программирования STL есть операторы NOP 0, NOP 1 и BLD, используемые в качестве Null-операций (нуль-операций).

18.4.1 Операторы NOP

Вы можете использовать операторы NOP 0 (набор битов 16 x "0") и NOP 1 (набор битов 16 x "1") для использования в инструкциях, которые не выполняют никакого действия.

Примечание:

Необходимо отметить, что Null-операции (нуль-операции) занимают в памяти определенное пространство (2 байта) и для выполнения инструкции требуют определенного времени.

Пример:

Необходимо, чтобы в строке с меткой перехода присутствовала инструкция. Если необходимо использовать операцию перехода, но не нужно выполнять никаких операций в строке с меткой перехода, то Вы можете в этой строке использовать оператор NOP 0.

```
          A    I 1.0
          JC   MXX1
          ...
MXX1:     NOP  0
```

Вы можете ввести пустую строку для лучшей читаемости программы простым вводом строки комментария (пустой строки) (это не требует использования пространства памяти пользователя и не добавляет потерь в общее время выполнения программы, так как не содержит кода).

18.4.2 Оператор отображения программы BLD

Редактор использует инструкцию отображения программы BLD nnn для включения информации декомпиляции в программу.

Сами операторы BLD не отображаются.

19 Параметры блоков

Из этой главы Вы узнаете, как использовать параметры блоков; кроме того, Вы узнаете, как

- объявлять (декларировать) параметры блока,
- работать с параметрами блока,
- инициализировать параметры блока,
- последовательно передавать ("pass on") параметры блока.

Параметры блока представляют собой интерфейс передачи данных между вызывающим и вызванным блоками. Все функции, в частности, функции блоков могут выполняться посредством параметров блоков.

19.1 Параметры блока: общая информация

19.1.1 Определение параметров блока

Параметры блока дают возможность передать данные для выполнения инструкций и функций блока.

Пример: необходимо записать блок с функцией сумматора (adder), который Вы хотите использовать в своей программе несколько раз для различных переменных. Переменные будут передаваться как параметры блока; в нашем примере используются три входных параметра и один выходной (см. рис. 19.1). Так как для функционирования сумматора нет необходимости сохранения внутренних значений, то для данной задачи может быть использована функция.

Параметр блока необходимо определять как "входной" параметр (input parameter), если требуется только проверить (Check - т.е., опросить) или загрузить (Load) его значение в программе блока. Если значение параметра только записывается (операции Set, Reset, Assign, Transfer), то Вы имеете дело с "выходным" параметром (output parameter). Если же параметр может быть как записан, так и считан, то Вы должны использовать параметр типа "входной/выходной" (in/out parameter). Редактор не проверяет вариант использования параметров.

19.1.2 Обработка параметров блока

В программе сумматора имена параметров блока являются как бы "вместилищем" для содержимого переменных, которые в дальнейшем будут передаваться в блок для обработки.

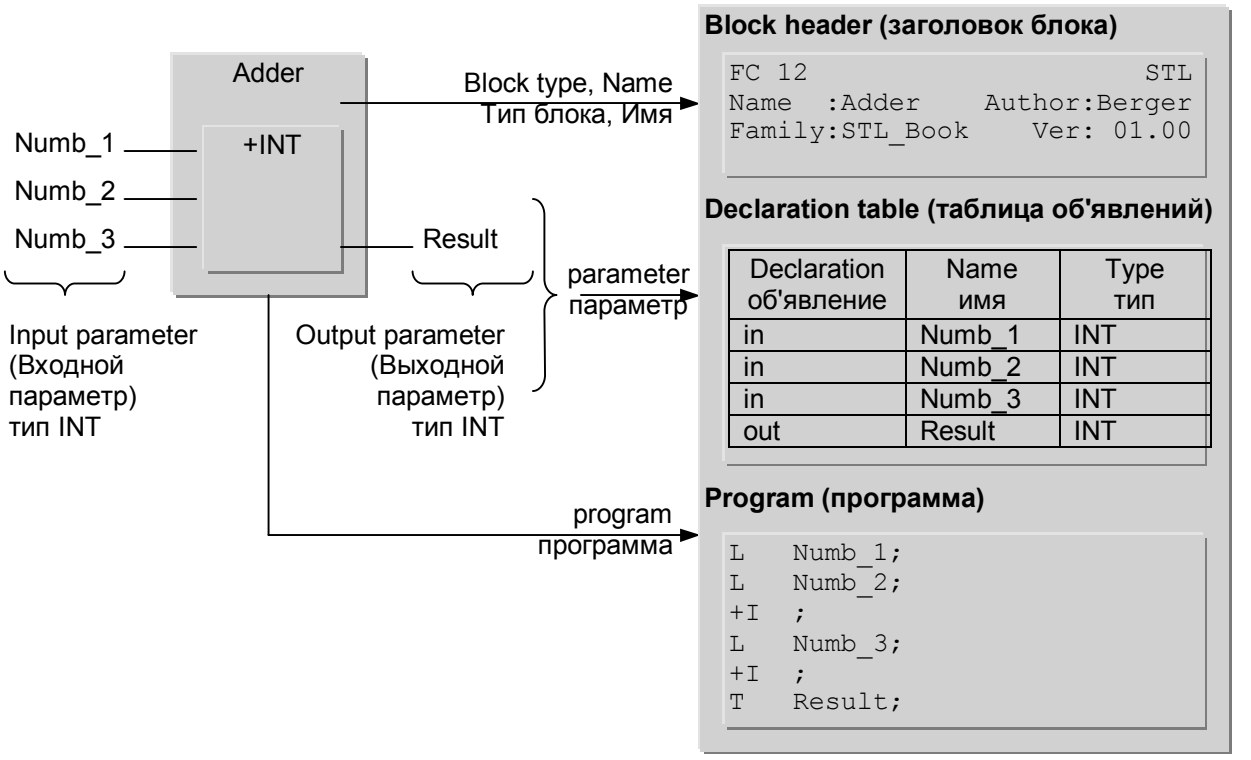


Рис. 19.1 Пример использования параметров блока

Вы можете использовать параметры блока также, как при символьной адресации переменных; в программе они называются *формальными параметрами (Formal parameters).*

Функция "Adder" может быть вызвана в Вашей программе несколько раз. При каждом вызове Вы можете передавать различные значения для операции суммирования с помощью параметров блока (см. рис. 19.2). Значения могут быть константами, адресами или переменными; они называются *фактическими параметрами (Actual parameters).*

Во время выполнения программы CPU заменяет формальные параметры фактическими параметрами. При первом вызове в примере (рис. 19.2) складываются значения из слов MW 30, MW 32 и MW 34, после чего результат сложения записывается в слово MW 40. Другой блок с другими фактическими параметрами вызывает функцию "Adder" второй раз, в результате чего происходит суммирование данных, заключенных в словах DBW 30, DBW 32 и DBW 34 блока данных DB 10, и сохранение суммарного значения в слове данных DBW 40 блока данных DB 10.

19.1.3 Объявление (declaration) параметров блока

Параметры блока должны быть определены в разделе объявлений блока, при программировании этого блока.

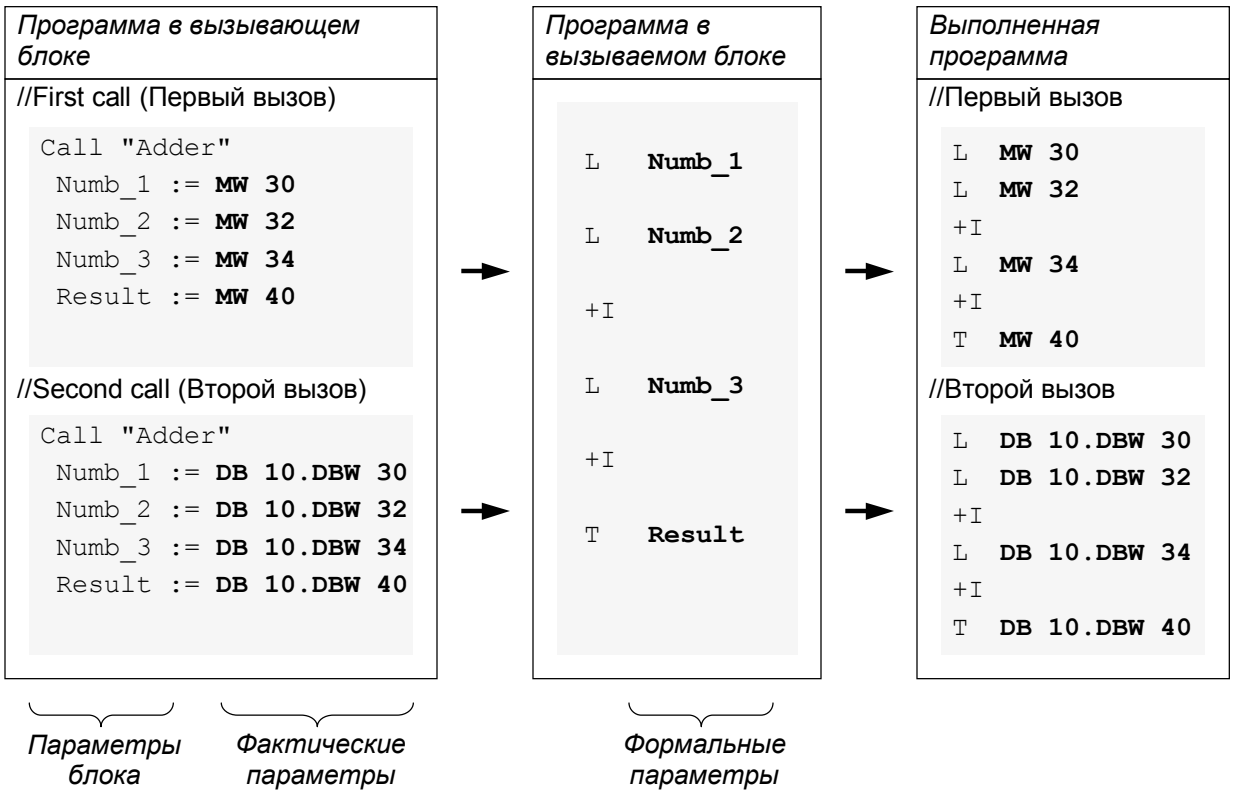


Рис. 19.2 Вызов блока с параметрами

При инкрементном способе программирования Вы просто дополняете список параметров. При способе программирования, ориентированном на создание исходных текстов, Вы определяете параметры блока в особых разделах (рис. 19.3). При этом используются следующие ключевые слова: VAR_INPUT - для входных параметров, VAR_OUTPUT - для выходных параметров и VAR_IN_OUT - для входных/выходных параметров.

Предопределение параметров является необязательным (опциональным) моментом при программировании и имеет смысл только для функциональных блоков, если значение параметра должно быть сохранено. Это касается всех параметров блоков простых типов, а также входных и выходных параметров сложных типов. Заполнение столбца комментариев для параметров необязательно и может быть выполнено в любой момент.

Имя параметра (*Block parameter name*) может содержать до 24 символов. При этом оно должно состоять только из алфавитно-цифровых символов (без национальных элементов, таких как умляут из немецкого алфавита) и символа подчеркивания. Символы в верхнем и нижнем регистрах различаются. Имя не может совпадать ни с одним из ключевых слов.

Символы в верхнем и нижнем регистрах не различаются при вводе имени параметра. При выводе программы редактор использует регистр символов, установленный при объявлении имени параметра блока.

При указании типа данных *Data type* для параметров разрешены все типы: простые *(elementary)*, сложные *(complex)* и пользовательские *(user-defined)*.

Инкрементное программирование

Address (адрес)	Declaration (объявление)	Name (имя)	Type (тип)	Initial value (начальное значение)	Comment (комментарий)
0.0	in	Manual	BOOL	TRUE	Ручное управление
2.0	in	Setpoint	INT	10000	Заданная скорость двигателя Motor 1
4.0	in	Characteristic	ANY		Указатель Point на область данных
14.0	out	Actual_value	INT	0	Скорость двигателя Motor 1
16.0	out	Temperature	REAL	0.000000e+00	Температура двигателя Motor 1
20.0	out	Message	WORD	W#16#0	Сообщение о сбое
22.0	in_out	EM	ARRAY[1..16]		Меркер фронта
*0.1	in_out		BOOL		
*28.0	in_out	Interface	DWORD	DW#16#0	Интерфейс для двигателя Motor 2

Программирование, ориентирование на создание исходных текстов программы

```
VAR_INPUT
  Manual         : BOOL  := TRUE;        //Ручное управление
  Setpoint       : INT   := 10000;       //Заданная скорость для Motor 1
  Characteristic : ANY;                  //Указатель на область данных
END_VAR
```

```
VAR_OUTPUT
  Actual_value   : INT   := 0;           //Скорость двигателя Motor 1
  Temperature    : REAL  := 0.0;         //Температура двигателя Motor 1
  Message        : WORD  := 16#0000;     //Сообщение о сбое
END_VAR
```

```
VAR_IN_OUT
  EM             : ARRAY[1..8] OF BOOL;  //Меркер фронта
  Interface      : DWORD := 16#00000000; //Интерфейс для двиг. Motor 2
END_VAR
```

Рис. 19.3 Пример объявления параметров блока для инкрементного способа программирования и для создания исходных текстов программы

Кроме того, Вы можете задавать типы параметров для параметров блоков.

STEP 7 сохраняет имена параметров блоков в невыполняемых разделах блоков на носителях данных в программаторе PG. В рабочей (work) памяти CPU (в скомпилированном блоке) содержатся только объявленные типы (в столбце *Declaration*) и типы данных (в столбце *Type*). Поэтому изменения в программе блока, которые делаются в интерактивном (online) режиме в памяти CPU, должны быть дополнительно продублированы и на носителях данных в программаторе PG, чтобы сохранить заданные пользователем имена.

Если не обновить данные в PG или блоки перенести из CPU в программатор PG, то соответствующие неисполняемые разделы блока будут либо удалены, либо переписаны. Затем редактор перезаписывает имена, заменяя их, для отображения на экране дисплея или для печати (входные параметры при этом получают имена вида INn, выходные параметры получают имена вида OUTn и входные/выходные параметры получают имена вида INOUTn, где n является порядковым номером, начинающимся с 1).

19.1.4 Объявление (declaration) значения функции

Такой параметр блока как значение функции (в случае применения блока-функции) является особым выходным параметром. Он имеет имя RET_VAL (или ret_val) и определяется как первый выходной параметр.

Тип значения функции может относиться к любому из простых типов данных. Кроме того, к допустимым типам в этом случае также относятся следующие типы данных: DATE_AND_TIME, STRING, POINTER, ANY и пользовательский тип UDT. Однако, тип значения функции не может относиться к таким типам данных, как ARRAY и STRUCT.

В рассмотренном выше примере функция "Adder" может быть также запрограммирована с присвоением значения параметра Result значению функции.

Программирование, ориентированное на создание исходных файлов программы

При программировании, ориентированном на создание исходных файлов программы, Вы должны определить функциональное значение (значение функции), задав тип данных для значения функции после типа блока и разделив эти идентификаторы двоеточием:

```
FUNCTION FC 12 : INT
VAR_INPUT
  Numb_1 : INT;
  Numb_2 : INT;
  Numb_3 : INT;
END_VAR
BEGIN
  L   Numb_1;
  L   Numb_2;
```

```
  +I  ;
  L   Numb_3;
  +I  ;
  T   RET_VAL;
END_FUNCTION
```

В примере значение функции RET_VAL принадлежит к типу INT. Посредством инструкции "T RET_VAL" значение функции принимается равным итоговой сумме параметров Numb_1, Numb_2 и Numb_3.

Инкрементное программирование

При инкрементном программировании Вы должны назначить имя RET_VAL *первому* выходному параметру в списке выходных параметров в разделе объявления переменных. Этим самым Вы определите этот выходной параметр как значение функции для функции FC.

В программе Вы должны рассматривать значение функции как выходной параметр. В примере посредством инструкции "T RET_VAL" значение функции принимается равным итоговой сумме параметров *Numb_1, Numb_2* и *Numb_3.*

19.1.5 Инициализация (Initialization) параметров блока

При вызове блока Вы инициализируете параметры блока значениями фактических параметров. Эти параметры могут быть константами, абсолютными адресами, данными с полной адресацией или переменными с символьной адресацией. Фактические параметры должны относиться к такому же типу данных, что и параметры блока (см. разд. 19.3 "Фактические параметры").

Начиная со STEP 7 V5.1 Вы должны определять параметры блока в исходной программе точно в том порядке, в котором они определены в разделе объявления блока во время программирования. Вы должны инициализировать все параметры функции при каждом ее вызове. В случае использования функциональных блоков инициализация отдельных или всех параметров блока носит опциональный (т.е., выборочный) характер.

19.2 Формальные параметры

В данном разделе Вы узнаете, как получить доступ к параметрам блока внутри блока. В табл. 19.1 показано, что нет ограничений для доступа к параметрам простых типов, к компонентам полей или структур, к функциям таймеров и счетчиков.

Доступ к параметрам сложных типов и к параметрам типов POINTER и ANY в языке программирования STL в настоящее время не поддерживается. Тем не менее, Вы можете инициализировать соответствующие блоки или системные блоки, которые имеют такие параметры, с помощью соответствующих переменных.

Тем не менее, в главе 26 "Прямой доступ к переменным" показано, как Вы можете использовать параметры таких типов данных в блоках, которые написали Вы сами.

Таблица 19.1 Доступ к параметрам блока (общий вид)

Тип данных	Допускается для			Доступ к блоку	
	IN	I_O	OUT	возможен	с помощью
Простые типы данных:					
BOOL	x	x	x	x	двоичного опроса, операции с памятью
BYTE, WORD, DWORD, CHAR, INT, DINT, REAL, S5TIME, TIME, DOT, DATE	x	x	x	x	операции загрузки (load) и пересылки (transfer)
Сложные типы данных:					
DT, STRING	x	x	x	-	Невозможно в STL непосредственно
ARRAY, STRUCT					
Отдельные двоичные компоненты	x	x	x	x	двоичного опроса, операции с памятью
Отдельные двоичные компоненты	x	x	x	x	операции загрузки (load) и пересылки (transfer)
Полные переменные	x	x	x	-	Невозможно в STL непосредственно
Типы параметров:					
TIMER	x	-	-	x	всех операций таймера
COUNTER	x	-	-	x	всех операций счетчика
BLOCK_FC, BLOCK_FB	x	-	-	x	вызова с UC и CC[2)]
BLOCK_DB	x	-	-	x	открытия с OPN DB
BLOCK_SDB	x	-	-	-	Невозможно[3)]
POINTER, ANY	x	x	x[1)]	-	Невозможно в STL непосредственно

[1)] Только для функций

[2)] Оператор CC не применим для функций

[3)] Имеет значение только для системных блоков

Параметры блоков типа BOOL

Параметры блоков типа BOOL могут быть отдельными двоичными переменными или двоичными компонентами полей или структур. Вы можете проверять (опрашивать) входные параметры и входные/выходные параметры с помощью подключения (with contacts) ко входам двоичных функций и влиять на выходные параметры и входные/выходные параметры с помощью операций с памятью.

При использовании FC функций-блоков Вы должны назначать значение двоичному выходному параметру и значению функции в блоке или устанавливать (сбрасывать) его.

В табл. 19.2 показаны разрешенные операции с данными типа BOOL. При программировании функции Вы должны использовать формальные параметры, вместо параметров блока xxxx.

После того, как CPU использовал фактический параметр, определенный посредством параметра блока, он обрабатывает соответствующий оператор (см. табл. 19.2) в соответствии с материалом из глав 4 "Двоичные логические операции" и 5 "Операции с памятью".

Таблица 19.2 Доступ параметрам блоков типа BOOL

A	-	Логическая операция AND (И) при проверки на состояние сигнала "1"
AN	-	Логическая операция AND (И) при проверки на состояние сигнала "0"
O	-	Логическая операция OR (ИЛИ) при проверки на состояние сигнала "1"
ON	-	Логическая операция OR (ИЛИ) при проверки на состояние сигнала "0"
X	-	Логическая операция Exclusive OR (исключающее ИЛИ) при проверки на состояние сигнала "1"
XN	-	Логическая операция Exclusive OR (исключающее ИЛИ) при проверки на состояние сигнала "0"
-	xxxx	входного или входного/выходного параметра типа BOOL
-	xxxx	входного параметра типа TIMER
-	xxxx	входного параметра типа COUNTER
S	-	Операция SET (Установка)
R	-	Операция RESET (Сброс)
=	-	Операция присвоения
-	xxxx	выходного или входного/выходного параметра типа BOOL
FP	-	Операция проверки наличия положительного фронта сигнала
FN	-	Операция проверки наличия отрицательного фронта сигнала
-	xxxx	входного/выходного параметра типа BOOL

Параметры блоков числовых типов

Параметры блоков числовых типов занимают 8, 16 или 32 бита (они могут относиться к любым простым типам, кроме типа BOOL). Они могут быть отдельными численными переменными или численными компонентами полей или структур. Вы можете считывать входные параметры и входные/выходные параметры с помощью функции загрузки (load) ко входам двоичных функций, можете записать выходные параметры и входные/выходные параметры с помощью функций передачи (transfer).

При использовании FC Вы должны передать (transfer) значение в численный выходной параметр и в значение функции. До этого Вы не должны выходить из функции.

```
L   xxxx   Загрузка входного или вх/вых параметра
T   xxxx   Передача в выходной или вх/вых параметр
```

При программировании функции Вы должны использовать формальные параметры, вместо параметров блока xxxx.

После того, как CPU использовал фактический параметр, определенный посредством параметра блока, он обрабатывает соответствующий оператор в соответствии с материалом из главы 6 "Функции пересылки данных".

Параметры блоков типов DT и STRING

Прямой доступ к параметрам блоков типов DT и STRING в настоящее время не поддерживается. Однако, при работе с функциональными блоками Вы можете передавать параметры типов DT и STRING в параметры вызываемых блоков.

В главе 26 "Прямой доступ к переменным" показано, как пользователь может сам запрограммировать операции доступа к параметрам указанных выше типов данных.

Параметры блоков типов ARRAY и STRUCT

Прямой доступ к параметрам блоков типов ARRAY и STRUCT возможен в "покомпонентном" режиме, т.е., доступ обеспечивается к отдельным двоичным или численным компонентам данных вышеуказанных сложных типов посредством соответствующих операций (двоичных логических операций, операций с памятью, функций загрузки [load] или пересылки [transfer]).

Доступ к полным переменным (к целым полям или к целым структурам) в настоящее время не поддерживается, а также нет доступа к отдельным компонентам комбинированного или пользовательского типа. Однако, при работе с функциональными блоками Вы можете последовательно передавать ("pass on") параметры типов ARRAY и STRUCT в параметры вызываемых блоков.

В главе 26 "Прямой доступ к переменным" показано, как пользователь может сам запрограммировать операции доступа к параметрам указанных выше типов данных.

Параметры блоков пользовательского типа

Вы можете работать с параметрами пользовательского типа таким же образом, как и с параметрами типа STRUCT.

Прямой доступ обеспечивается к отдельным двоичным или численным компонентам данных пользовательского типа UDT посредством соответствующих операций (двоичных логических операций, операций с памятью, функций загрузки [load] или пересылки [transfer]).

Доступ к полным переменным в настоящее время не поддерживается и также доступ к отдельным компонентам комбинированного или пользовательского типа невозможен. При работе с функциональными блоками Вы можете последовательно передавать ("pass on") параметры типа UDT в параметры вызываемых блоков.

В главе 26 "Прямой доступ к переменным" показано, как пользователь может сам запрограммировать операции доступа к параметрам указанных выше типов данных.

Параметры блоков типа TIMER

В добавление к функциям проверки (опроса), указанным в табл. 19.2, Вы можете программировать обработку параметров блока типа TIMER с помощью следующих операторов:

SP	-	запуск таймера в режиме "импульса" ("pulse")
SD	-	запуск таймера в режиме "задержки включения" ("ON delay")
SE	-	запуск таймера в режиме "расширенного импульса" ("extended pulse")
SS	-	запуск таймера в режиме "задержки включения с памятью" ("retentive ON delay")
SF	-	запуск таймера в режиме "задержки выключения" ("OFF delay")
R	-	функция сброса ("reset")
FR	-	функция разрешения ("enable")
-	xxxx	входной параметр типа TIMER

При программировании функции Вы должны использовать формальные параметры, вместо параметров блока xxxx.

После того, как CPU использовал фактический параметр, определенный посредством параметра блока, он обрабатывает соответствующий STL оператор точно также, как описано в главе 7 "Функции таймера". При запуске функции таймера значение времени "time value" может также быть параметром формата S5TIME.

Параметры блоков типа COUNTER

В добавление к функциям проверки (опроса), указанным в табл. 19.2, Вы можете программировать обработку параметров блока типа COUNTER с помощью следующих операторов:

S	-	установка счетчика ("set")
CU	-	запуск счетчика в режиме "прямого счета" ("count up")
CD	-	запуск счетчика в режиме "обратного счета" ("count down")
R	-	функция сброса ("reset")
FR	-	функция разрешения ("enable")
-	xxxx	входной параметр типа COUNTER

При программировании функции Вы должны использовать формальные параметры, вместо параметров блока xxxx.

После того, как CPU использовал фактический параметр, определенный посредством параметра блока, он обрабатывает соответствующий STL оператор точно также, как описано в главе 8 "Функции счетчика". При запуске функции счетчика значение счетчика "count value" может также быть, например, параметром типа WORD.

Параметры блоков типа BLOCK_xx

OPN	-	функция открытия блока данных (параметр типа BLOCK_DB)
UC	-	вызов функции (параметр типа BLOCK_FC)
UC	-	вызов функционального блока (параметр типа BLOCK_FB)
CC	-	условный вызов функции (параметр типа BLOCK_FC)
CC	-	условный вызов функционального блока (параметр типа BLOCK_FB) (см. текст)
-	xxxx	входной параметр

При программировании функции Вы должны использовать формальные параметры, вместо параметров блока xxxx.

При открытии блока данных посредством параметра блока, CPU всегда использует регистр блоков глобальных данных (DB-регистр).

Функции и функциональные блоки, которые пересылаются с параметрами блоков, сами не должны содержать параметров блоков. Условный вызов блока посредством параметра возможен, только если это параметр функционального блока.

Вы можете также использовать блок данных, который переслали как параметр блока, в качестве экземплярного блока. Так как редактор не проверяет тип блока данных, используемого при выполнении программы, Вы должны сами обеспечивать, чтобы пересылаемый блок данных также соответствовал экземплярному блоку данных для вызванного функционального блока.

Пример:

Вы можете определить параметр блока типа BLOCK_DB с помощью имени *#Data* как экземплярный блок данных для вызванного функционального блока:

```
CALL FB 10, #Data
```

Параметры блоков типов POINTER и ANY

Прямой доступ к параметрам блоков типов POINTER и ANY невозможен.

В главе 26 "Прямой доступ к переменным" показано, как пользователь может сам запрограммировать операции доступа к параметрам блоков типов POINTER и ANY.

19.3 Фактические параметры

При вызове блока Вы инициализируете параметры блока значениями, которые могут быть константами, адресами или переменными, с которыми программа блока должна быть выполнена. Это фактические параметры. При частых вызовах блока в программе, обычно используются

различные значения фактических параметров.

Фактический параметр должен относиться к такому же типу данных, что и соответствующий параметр блока. Вы можете поставить в соответствие двоичному фактическому параметру (например, меркеру) только параметр блока типа BOOL; Вы можете инициализировать параметр блока типа ARRAY только переменной, занимающей такое же поле в памяти. В табл. 19.3 представлен обзор адресуемых данных, которые Вы можете применять как фактические параметры соответствующих типов.

Табл. 19.3 Инициализация фактических параметров

Тип параметра блока	Допустимые фактические параметры
Простой тип	• Простые адреса, полные адреса, константы • Компоненты полей или структур простых типов • Параметр вызывающего блока • Компоненты параметров блока вызывающего блока простых типов
Сложный тип	• Переменные или параметры вызывающего блока
TIMER, COUNTER, BLOCK_xx	• Таймеры, счетчики и блоки
POINTER	• Простые адреса, полные адреса • Указатель области или DB-указатель
ANY	• Переменные любого типа • Указатель ANY

При вызове функции Вы должны инициализировать все параметры блока фактическими параметрами.

При вызове функциональных блоков нет необходимости инициализировать параметры блока. STEP 7 сохраняет все параметры блока простых типов, входные и выходные параметры сложных типов и входные параметры типов TIMER, COUNTER и BLOCK_xx как значения или как числа. Вх/вых параметры сложных типов и входные параметры типов POINTER и ANY сохраняются как указатели на фактические параметры. Так как при этом вводится значащее значение, Вы должны инициализировать по крайней мере параметры с неполной адресацией, по крайней мере при первом вызове. Имеется также возможность прямого доступа к параметрам блока. Так как параметры блока располагаются в блоке данных, Вы можете обрабатывать их как адреса данных.

Пример:

Функциональный блок с экземплярным блоком "Lift_stat_1" управляет двоичным выходным параметром с именем *Up.* Выполняя обработку программы функционального блока после его вызова, Вы можете проверить (опросить) параметр без инициализации выходного параметра:

```
A   "Lift_stat_1".Up
```

Вы можете запрограммировать такую инструкцию проверки (опроса) параметра вместо его инициализации.

Инициализация параметров блоков простых типов данных

Параметры, перечисленные в табл. 19.4, являются допустимыми для использования в качестве фактических параметров простых типов.

Таблица 19.4 Фактические параметры простых типов данных

Адреса	Допускается для			Адрес двоичных данных или символьное имя	Адрес числовых данных или символьное имя
	IN	I_O	OUT		
Входы (образ процесса)	x	x	x	I y.x	IB y, IW y, ID y
Выходы (образ процесса)	x	x	x	Q y.x	QB y, QW y, QD y
Меркеры	x	x	x	M y.x	MB y, MW y, MD y
Периферийные входы	x	-	-	-	PIB y, PIW y, PID y
Периферийные выходы	-	-	x	-	PQB y, PQW y, PQD y
Глобальные данные неполная адресация полная адресация	 x x	 x x	 x x	 DBX y.x DBz.DBX y.x	 DBB y, DBW y, DBD y DBz.DBB y, DBz.DBW y, DBz.DBD y
Временные локальные данные	x	x	x	L y.x	LB y, LW y, LD y
Статические локальные данные	x	x	x	DIX y.x	DIB y, DIW y, DID y
Константы	x	-	-	TRUE, FALSE	Все числовые константы
Компоненты массивов и структур	x	x	x	Полное имя компонента	Полное имя компонента

x = адрес бита, y = адрес байта, z = номер блока данных

Вы можете назначить или абсолютный, или символьный адрес входу, выходу и меркеру. Входные адреса обычно назначаются входным параметрам; выходные адреса обычно назначаются выходным параметрам (тем не менее, это не обязательно). Адреса меркеров допустимы для параметров, объявленных как входныые, выходные или вх/вых параметры.

При использовании неполной адресации данных Вы должны обеспечить, чтобы соблюдалась "корректность" в открытом блоке данных при обращении к параметру блока (в вызванном блоке). Так как редактор может в отдельных случаях заменить блок данных при вызове блока, то применение неполной адресации данных не рекомендуется. Поэтому используйте только полную адресацию данных.

Для временных локальных данных обычно используется символьная адресация. Они размещаются в L-стеке вызывающего блока (и объявлены в вызывающем блоке).

Если вызывающий блок является функциональным блоком, Вы можете также использовать его статические локальные данные как фактические параметры (см. раздел 19.4 "Передача "Pass On" параметров блока").

Для статических данных обычно используется символьная адресация. Если Вы используете абсолютную адресацию с указанием DI-регистра (DI-адресация), то Вы должны обеспечить "корректность" в открытом блоке данных с указанием DI-регистра при обращении к параметру блока (в вызванном блоке).

Примечание: в связи с вышеизложенным надо заметить, что при использовании вызываемых блоков как локальных экземпляров, абсолютные адреса локальных (block-local - внутриблочных) переменных зависит от объявления локального экземпляра в вызванном блоке.

В качестве параметра блока типа BOOL Вы можете применять константу TRUE (ИСТИНА - для состояния сигнала "1") или константу FALSE (ЛОЖЬ - для состояния сигнала "0"). В качестве параметра блока численных типов Вы можете применять любые константы, относящиеся к численным типам. Инициализация посредством присвоения констант имеет смысл только для входных параметров.

Вы можете также инициализировать параметры блока простых типов посредством присвоения компонентов полей или структур при условии, что компоненты полей или структур имеют соответствущий тип данных.

Инициализация параметров блоков сложных типов данных

Каждый параметр блока может быть сложного типа или пользовательского (UDT) типа. Этим параметрам могут быть поставлены в соответствие адреса переменных соответствующих типов, то есть, эти переменные могут выступать как фактические параметры.

Вы можете для инициализации параметров блока типа DT и STRING использвать отдельные переменные или компоненты полей или структур при условии, что они имеют соответствущий тип данных. Инициализация таких параметров не допустима в STL.

Если Вы инициализируете параметры функционального блока переменной STRING, эта переменная должна иметь такую же длину как и параметр блока STRING.

Когда создается переменная STRING во временных локальных данных, предопределение ее значение невозможно, поэтому при этих условиях содержимое данной переменной содержит "случайные" значения. При использовании такой переменной в качестве фактического параметра для IEC-функции Вы должны сначала определить ее соответствующим корректным значением в программе (перед записью значения в STRING-переменную IEC-функция проверяет его на соответствие типу переменной).

Вы можете для инициализации параметров блока типа ARRAY и STRUCT использвать переменные с точно такой же структурой как и параметры блока.

Назначение параметров сложных типов описано в разделе 26.4 "Краткое описание примера фрейма сообщения" в примерах "Создание фрейма сообщения" и "Считывание времени суток (считывание данных TOD)".

Инициализация параметров блоков пользовательских типов данных

Для сложных и громоздких структур данных рекомендуется ввести и использовать пользовательский (UDT) тип данных. Сначала Вы должны определить UDT-тип данных, затем - использовать его, например, для создания переменной в блоке данных или для объявления параметра

блока. После этого Вы можете использовать переменную при инициализации параметра блока. В этом случае фактический параметр (переменная) должна быть точно такого же типа (такой же UDT-структуры) как и параметр блока.

Назначение параметров пользовательских типов описано в разделе 26.4 "Краткое описание примера фрейма сообщения" в примере "Данные фрейма сообщения".

Инициализация параметров блоков типов TIMER, COUNTER и BLOCK_xx

Вы должны инициализировать параметр блока типа TIMER для функции таймера, параметр блока типа COUNTER для функции счетчика. Для параметров типа BLOCK_FC и BLOCK_FB Вы можете использовать только блоки без своих собственных параметров. После этого эти блоки могут быть вызваны посредством оператора UC (а также CC для функциональных блоков). Инициализировать параметр блока типа BLOCK_DB можно блоком данных, открытом в вызванном блоке посредством DB-регистра.

Назначение параметров пользовательских типов описано в разделе 26.4 "Краткое описание примера фрейма сообщения" в примере "Данные фрейма сообщения".

Параметры блоков типов TIMER, COUNTER и BLOCK_xx могут быть только входными параметрами.

Инициализация параметров блоков типа POINTER

Для параметра блока типа POINTER допустимо использовать только указатели (константы). Эти указатели могут являться указателями либо для области данных (32-разрядные), либо для блока данных DB (48-разрядные).

При этом возможна адресация данных простых типов; также возможна полная адресация.

Для функциональных блоков не допускаются выходные параметры типа POINTER.

Инициализация параметров блоков типа ANY

Для параметра блока типа ANY допустимо использовать переменные любых типов. При программировании вызываемого блока Вы должны определить, какие переменные (адреса или типы данных) должны быть

применены в параметрах блока, какие переменные могут примяться. Вы также можете определить константы в формате указателя ANY

"p#[Data_block.]Address Data_type Number"

и таким образом определить область с помощью абсолютной адресацией.

Исключением является инициализация параметра ANY временными локальными данными типа ANY. В этом случае редактор скорее допустит, что указатель типа ANY уже существует во временных локальных данных, чем создаст указатель на переменную. Это дает Вам возможность применить для параметра ANY указатель ANY, которым можно манипулировать во время выполнения программы. "Переменный указатель ANY" может быть, отчасти, полезен при работе с системной функцией SFC 20 BLKMOV (см. пример "Буфер входа" в разделе 26.4 "Краткое описание примера фрейма сообщения").

Для функциональных блоков не допускаются выходные параметры типа ANY.

19.4 Последовательная передача ("Pass On") параметров блока

Последовательная передача ("Pass On") параметров блока - это особая форма доступа и инициализации параметров блока. При такой форме доступа параметры блока вызывающего передаются ("passed on") в параметры вызванного блока. При этом формальный параметр вызывающего блока становится фактическим параметром вызванного блока.

В общем случае, при этом фактический параметр должен быть того же типа, что и формальный параметр (то есть, параметры блока должны соответствовать типам передаваемых данных). Вдобавок, Вы можете применить входной параметр вызывающего блока только как входной параметр вызываемого блока, и, аналогично, использовать выходные параметры. Вы можете применить вх/вых параметр вызывающего блока во всех (входных, выходных и вх/вых) параметрах вызываемого блока.

Существуют ограничения, зависящие от типов данных, которые возникают из-за различий в способе хранения параметров блоков между функциями и функциональными блоками. Параметры блоков могут последовательно передаваться ("pass on") без ограничений и в соответствии с информацией, изложенной в предыдущих параграфах. Сложные типы данных для входов и выходных параметров могут передаваться, если только вызывающий блок является функциональным блоком. Параметры блоков типов TIMER, COUNTER и BLOCK_xx могут передаваться от одного входного параметра к другому, только если вызывающий блок является функциональным блоком. В табл. 19.5 представлены разрешенные и запрещенные пути передачи данных между входными и выходными параметрами для функций и функциональных блоков в зависимости от типа передаваемых данных.

Вы можете последовательно передавать ("pass on") данные типов TIMER, COUNTER и BLOCK_xx при работе с функциями при использовании косвенной адресации. Соответствующему параметру сначала должен быть присвоен тип данных WORD или INT; затем Вы инициализируете его константой или переменной, которая содержит номер таймера, счетчика или передаваемого блока. Вы можете последовательно передавать ("pass on") этот параметр, так как он относится к параметрам простого типа. В последнем (в цепочке передачи - "last") блоке Вы можете использовать функцию загрузки "load" для передачи содержимого параметра в слово временных локальных данных, после чего может быть выполнена функция таймера, счетчика или блока.

Табл. 19.5 Разрешенные комбинации для последовательной передачи ("pass on") параметров

Вызывающий ---> вызываемый (объявленный тип)	FC вызывает FC			FB вызывает FC			FC вызывает FB			FB вызывает FB		
	E	C	P	E	C	P	E	C	P	E	C	P
Input -> Input (Вх -> Вх)	x	-	-	x	x	-	x	-	x	x	x	x
Output -> Output (Вых -> Вых)	x	-	-	x	x	-	x	-	-	x	x	-
In/Out -> Input (Вх/Вых -> Вх)	x	-	-	x	-	-	x	-	-	x	-	-
In/Out -> Output (Вх/Вых -> Вых)	x	-	-	x	-	-	x	-	-	x	-	-
In/Out -> In/Out (Вх/Вых -> Вх/Вых)	x	-	-	x	-	-	x	-	-	x	-	-

E = простые типы данных
C = сложные типы данных
P = параметрические типы TIMER, COUNTER и BLOCK_xx

19.5 Примеры

19.5.1 Пример: ленточный конвейер

Пример показывает передачу состояний сигналов с помощью параметров. Для этой цели мы будем использовать функцию управления ленточным конвейером, работа которой была объяснена в главе 5 "Операции с памятью". Функция управления ленточным конвейером должна быть размещена в функциональном блоке, и все входы и выходы должны быть запрограммированы как параметры блока, таким образом, что функция управления ленточным конвейером может быть использована многократно (для нескольких конвейеров). На рис. 19.4 показаны входные и выходные параметры блока, так же как и использованные статические локальные данные.

В данном блоке параметры распределяются достаточно просто: все двоичные адреса, соответствующие входам должны стать входными параметрами, все двоичные адреса, соответствующие выходам должны стать выходными параметрами, все меркеры должны стать статическими локальными данными.

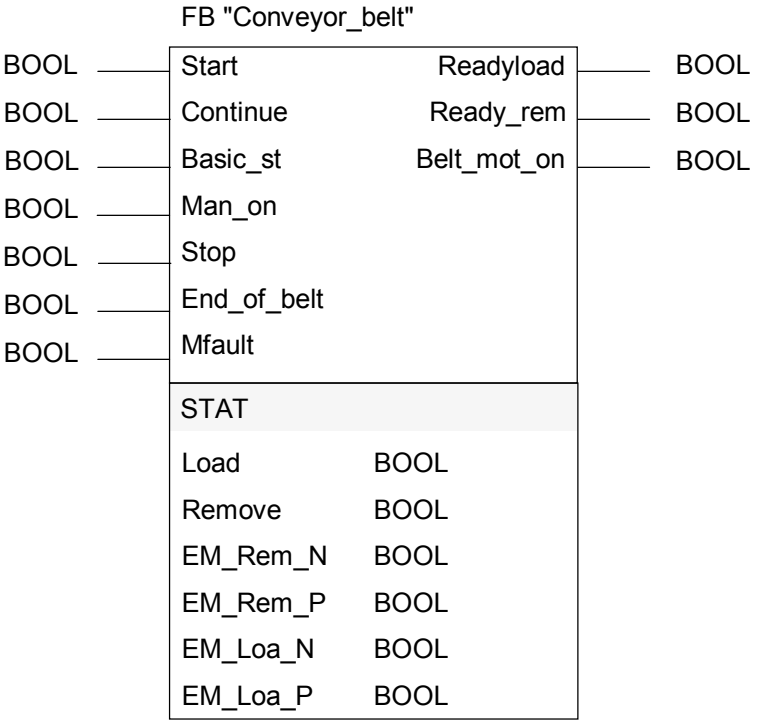


Рис. 19.4 Функциональный блок для примера ленточного конвейера

Вы можете заметить, что имена также слегка изменены, так как при написании имен переменных могут использоваться только алфавитно-цифровые символы, а также символ подчеркивания.

Функциональный блок "Conveyor_belt" предназначен для управления двумя ленточными конвейерами. Поэтому он вызываеися дважды: первый раз - для обработки входов и выходов конвейера 1, второй раз - для обработки входов и выходов конвейера 2. Для каждого вызова функционального блока требуется экземплярный блок данных, в котором хранятся данные конвейера для соответствующего случая. Блок данных для конвейера 1 называется "Belt_data1", блок данных для конвейера 2 называется "Belt_data2" и т.д.

Вы можете найти пример исполняемой программы в библиотеке STL_Book с названием "Conveyor Example" на дискете, приложенной к книге. Исходная программа содержит код функционального блока с входными параметрами, с выходными параметрами и статическими локальными данными. За этой программой следуют программы экземплярных блоков данных; в них достаточно определить функциональный блок в разделе объявлений. Вы можете использовать любой блок данных в качестве экземплярного, например, DB 21 "Belt_data1" и DB 22 "Belt_data2". В таблице символов эти блоки данных имеют тип данных функционального блока.

В конце исходной программы Вы можете увидеть другие два вызова функциональных блоков, такие, например, как в OB 1. Входы и выходы из таблицы символов используются как фактические параметры. В тех случаях, когда такие глобальные символьные имена содержат особые

символы, в программе эти имена должны быть заключены в кавычки.

19.5.2 Пример: счетчик деталей

Пример демонстрирует обработку параметров блока простых типов. Пример "Parts Counter" ("счетчик деталей") из главы 8 "Функции счетчиков" взят за основу для нашей функции. В нашем случае функция выполняется как функциональный блок со всеми глобальными переменными, объявленными или как параметры блока, или как статические локальные данные (Рис. 19.5).

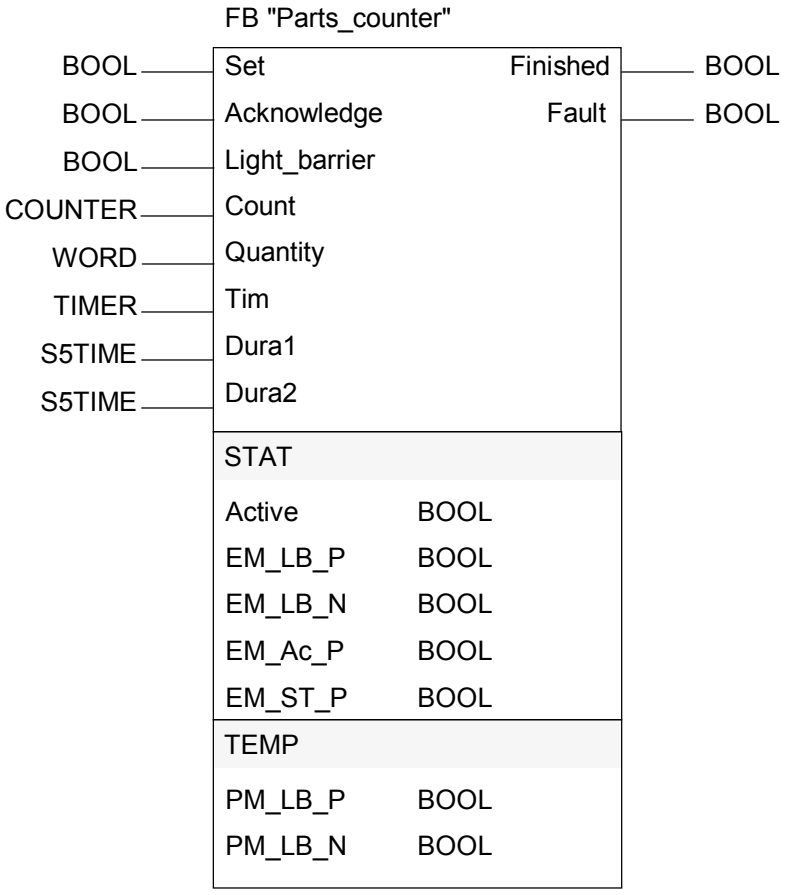


Рис. 19.5 Функциональный блок для примера счетчика деталей

Функции таймера и счетчика передаются с помощью параметров блока типов TIMER и COUNTER. Эти параметры блока должны быть входными параметрами. Начальные значения для функции счетчика (Quantity) и для функции таймера (Dura1 и Dura2) могут также быть переданы как параметры блока; тип данных параметров блока здесь соответствует фактическим параметрам.

Меркеры фронта сохраняются в статических локальных данных, а меркеры импульса сохраняются во временных локальных данных.

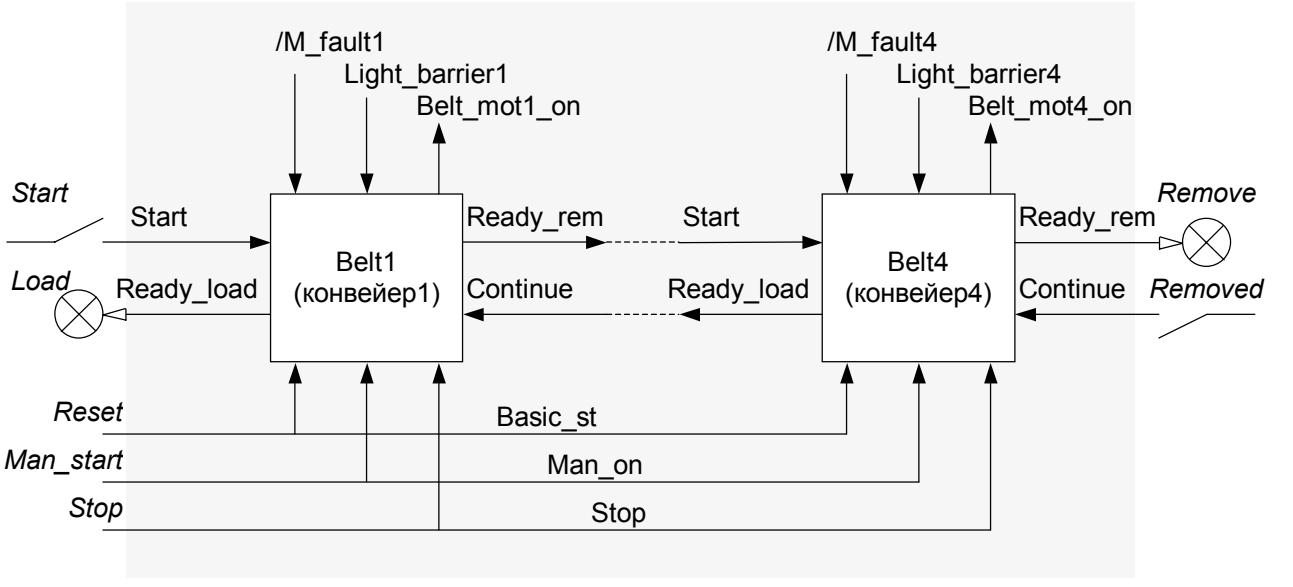
Вы можете найти пример исполняемой программы в библиотеке STL_Book с названием "Conveyor Example" на дискете, приложенной к книге. Исходная программа содержит функциональный блок "Parts_counter", связанный с ним экземплярный блок данных "Count_Dat", а также вызов функционального блока с экземплярным блоком данных.

19.5.3 Пример: подающий механизм

Такие же функции как те, что описаны в предыдущих двух примерах, могут также быть вызваны как локальные экземпляры. В нашем примере это означает, что нам предстоит запрограммировать функциональный блок "Feed" ("Подающий механизм") для управления четырьмя ленточными конвейерами и для подсчета деталей, переданных с помощью этих конвейеров. В данном функциональном блоке блок FB "Conveyor_Belt" ("Ленточный конвейер") вызывается четыре раза, а блок FB "Parts_Counter" ("Счетчик деталей") - только один раз. В нашем случае не для каждого вызова FB вызывается свой экземплярный блок, а все вызываемые функциональные блоки хранят свои данные в экземплярном блоке функционального блока "Feed".

На рис. 19.6 показано, как отдельные блоки управления конвейерами соединяются в единую систему управления (блок FB "Parts_Counter" не показан на рисунке).

Функциональный блок "Feed"



Belt1, ... , Belt4 - функциональные блоки "Conveyor_Belt"

Рис. 19.6 Пример функционального блока программы "Feed"

Запускающий (Start) сигнал подается на вход *Start* блока управления конвейера Belt 1, выход *Ready_rem* подается на вход *Start* блока управления конвейера Belt 2, и т.д. И, наконец, *Ready_rem* от блока управления Belt 4 подается на выход *Remove* системы управления "Feed". Такая же цепочка сигналов проходит в обратном направлении *Removed -> Continue -> Ready_load -> ... -> Load.*

Сигналы *Belt_motX_on, Light_barrierX* и */M_faultX* (отказ мотора) - это отдельные сигналы в каждом из блоков управления конвейерами. Входы *Reset, Man_start* и *Stop* позволяют управлять всеми блоками посредством сигналов *Basic_st, Man_on* и *Stop* соответственно.

Следующий раздел программы функционального блока "Feed" создан таким же путем. Входные и выходные параметры функционального блока показаны на рис. 19.5. Кроме того, числовые значения для счетчика деталей Quantity и таймера Dura1 и Dura2 используются здесь как входные параметры. Мы объявляем данные отдельных блоков управления конвейерами и данные счетчика деталей в статических локальных данных таким же образом, как объявляются данные пользовательского типа UDT, т.е., с указанием имени и типа данных.

Переменная *Belt1* должна содержать структуру данных функционального блока "Conveyor_Belt", так же как и переменные *Belt2, ..., Belt4;* переменная *Check* должна содержать структуру данных функционального блока "Parts_counter".

Программа функционального блока начинается с инициализации общих сигналов для всех блоков управления конвейерами. Здесь мы используем тот факт, что параметры блока функциональных блоков, вызываемых как локальные экземпляры, являются статическими локальными данными в текущем блоке и, поэтому, могут обрабатываться как таковые. Параметр блока *Man_start* текущего функционального блока управляет входным параметром *Man_on* всех четырех блоков управления конвейерами посредством простой операции присвоения. Таким же образом проводится инициализация входных параметров *Basic_st* и *Stop* с помощью параметров блока *Reset* и *Stop* соответственно. Теперь общие сигналы для блоков управления конвейерами инициализированы. (Конечно же Вы можете инициализировать эти сигналы и при вызове функционального блока).

Последовательные вызовы функциональных блоков управления конвейерами содержат параметры блока только для отдельных сигналов соответствующего конвейера, связанные с параметрами функционального блока "Feed". Эти отдельные сигналы представляют собой сигналы от фотодатчиков (*Light_barrierX*), сигналы для и от моторов приводов конвейеров (соответственно сигналы для аварийной остановки двигателей */M_faultX* и сигналы о состоянии моторов приводов *Belt_motX_on*). (Здесь имеется в виду, что при вызове функционального блока не все параметры блока должны быть инициализированы).

Программирование связи между отдельными блоками управления конвейерами производится с использованием операций присвоения.

Функциональный блок FB "Parts Counter" ("счетчик деталей") вызывается как локальный экземпляр, даже несмотря на то, что он не связан с сигналами управления конвейерными лентами. Экземплярный блок данных функционального блока "Feed" содержит в себе данные FB.

Входные параметры для счетчика деталей *Quantity* и таймера *Dura1* и *Dura2* функционального блока "Feed" должны быть установлены только один раз. Это может быть сделано путем задания значений, принимаемых по умолчанию (как в данном примере), или путем прямого присвоения значений при перезапуске программы в OB 100 (если, к примеру, эти три параметра трактуются как глобальные данные).

Данный функциональный блок "Feed" и связанный с ним экземплярный блок "FeedDat" содержатся в библиотеке STL_Book в программе с именем "Conveyor Example". Ниже представлен функциональный блок с экземплярным блоком данных для основной программы.

```
FUNCTION_BLOCK "Feed"
TITLE = Control of several conveyor belts //Управление несколькими
                                          //ленточными конвейерами
//Пример локальных экземпляров "local instances"
//declaration (объявления), calls (вызовы)
NAME        : Feed
AUTHOR      : Berger
FAMILY      : STL_Book
VERSION     : 01.00
VAR_INPUT
  Start     : BOOL      := FALSE;    //Запуск лент конвейеров
  Removed   : BOOL      := FALSE;    //Детали удалены с ленты
  Man_start : BOOL      := FALSE;    //Ручной запуск конвейеров
  Stop      : BOOL      := FALSE;    //Остановка лент конвейеров
  Reset     : BOOL      := FALSE;    //Установка в основное
                                     //  состояние (basic_st)
  Count     : COUNTER;               //Счетчик деталей
  Quantity  : WORD      := W#16#0200;//Число деталей
  Tim       : TIMER;                 //Функция таймера
  Dural     : S5TIME    := S5T#5s;   //Контрольное время д/деталей
  Dura2     : S5TIME    := S5T#10s;  //Контрольное время д/перерыва
END_VAR
VAR_OUTPUT
  Load      : BOOL      := FALSE;    //Загрузка ленты деталями
  Remove    : BOOL      := FALSE;    //Удаление деталей с ленты
END_VAR
VAR
  Beltl : "Conveyor_belt";    //Управление лентой belt 1
  Belt2 : "Conveyor_belt";    //Управление лентой belt 2
  Belt3 : "Conveyor_belt";    //Управление лентой belt 3
  Belt4 : "Conveyor_belt";    //Управление лентой belt 4
  Check : "Parts_counter";    //Управление подсчетом деталей
                              //и контролем времени
END_VAR

BEGIN

NETWORK
TITLE = Initializing the common signals //инициализация общих
                                        //сигналов
  A   Man_start;
  =   Beltl.Man_on;
  =   Belt2.Man_on;
  =   BeIt3.Man_on;
  =   Belt4.Man_on;

  A   Stop;
  =   Belt1.Stop;
  =   Belt2.Stop;
  =   Belt3.Stop;
  =   Belt4.Stop;

  A   Reset;
  =   Beltl.Basic_state;
  =   Belt2.Basic_state;
  =   Belt3.Basic_state;
  =   Belt4.Basic_state;
```

(продолжение на следующей странице)

```
NETWORK
TITLE = Calling the conveyor belt controls //Вызов блоков управления
                                           //отдельными конвейерами
CALL Beltl (
   Start          := Start,
   Readyload      := Load,
   End_of_belt    := Light_barrierl,
   Mfault         := "/Mfaultl",
   Belt_mot_pn    := Belt_motl_on);
A  Belt2.Readyload;
=  Beltl.Continue;
A  BeIt1.Ready_rem;
=  Belt2.Start;

CALL Belt2 (
   End_pf_belt    := Light_barrier2,
   Mfault         := "/Mfault2",
   Belt_mot_on    := Belt_mot2_on);
A  Belt3.Readyload;
=  Belt2.Continue;
A  Belt2.Ready_rem;
=  Belt3.Start;

CALL Belt3 (
   End_of_belt    := Light__barrier3,
   Mfault         := "/Mfault3",
   Belt_mot_on    := Belt_mot3_on);
A  Belt4.Readyload;
=  Belt3.Continue;
A  Belt3.Ready_rem;
=  Belt4.Start;

CALL Belt4 (
   Continue       := Removed,
   Ready_rem      := Remove,
   End_of_belt    := Light_barrier4,
   Mfault         := "/Mfault4",
   Belt_mot_on    := Belt_mot4_on);

NETWORK
TITLE = Call for counting and monitoring //Контроль времени/режима
CALL Check (
    Set            := Start,
    Acknowledge    := "Acknowledge";
    Light_barrier  := Light barrier 1,
    Count          := #Count,
    Quantity       := #Quantity,
    Tim            := #Tim,
    Dura1          := #Dural,
    Dura2          := #Dura2,
    Finished       := Finished,
    Fault          := "Fault");

NETWORK
TITLE = Block end
   BE
END FUNCTION BLOCK
```