
Содержание главы 18

18	<u>Функции для работы с блоками</u>	4
18.1	<u>Функции для работы с кодовыми блоками</u>	4
18.1.1	<u>Вызовы блоков: общая информация</u>	5
18.1.2	<u>Блочный элемент вызова</u>	6
18.1.3	<u>Катушка / блочный элемент CALL</u>	9
18.1.4	<u>Функция завершения блока</u>	10
18.1.5	<u>Временные локальные данные</u>	11
18.1.6	<u>Статические локальные данные</u>	14
18.2	<u>Функции блоков для блоков данных</u>	19
18.2.1	<u>Два регистра блоков данных</u>	19
18.2.2	<u>Доступ к операндам данных</u>	20
18.2.3	<u>Открытие блока данных</u>	22
18.2.4	<u>Особые замечания по адресации данных</u>	24
18.3	<u>Системные функции для блоков данных</u>	26
18.3.1	<u>Создание блока данных</u>	26
18.3.2	<u>Удаление блока данных</u>	26
18.3.3	<u>Тестирование блоков данных</u>	27

18 Функции для работы с блоками

В этой главе рассказывается о том, как вызывать и завершать кодовые блоки и как работать с операндами из блоков данных. Затем в следующей главе вы познакомитесь с использованием параметров блоков.

18.1 Функции для работы с кодовыми блоками

Функции для работы с кодовыми блоками (code blocks) включают в себя инструкции для вызова и завершения блоков (рисунок 18.1). Кодовые блоки вызываются с помощью блочного элемента вызова (call box). Если функции или системные функции не имеют параметров блока, они также могут быть вызваны с использованием катушки или блочного элемента CALL. В обоих случаях допустима предшествующая логическая операция, разрешающая условный вызов (вызов, зависимый от условий). Функция завершения блока RET всегда зависит от RLO (результата предшествующей логической операции).

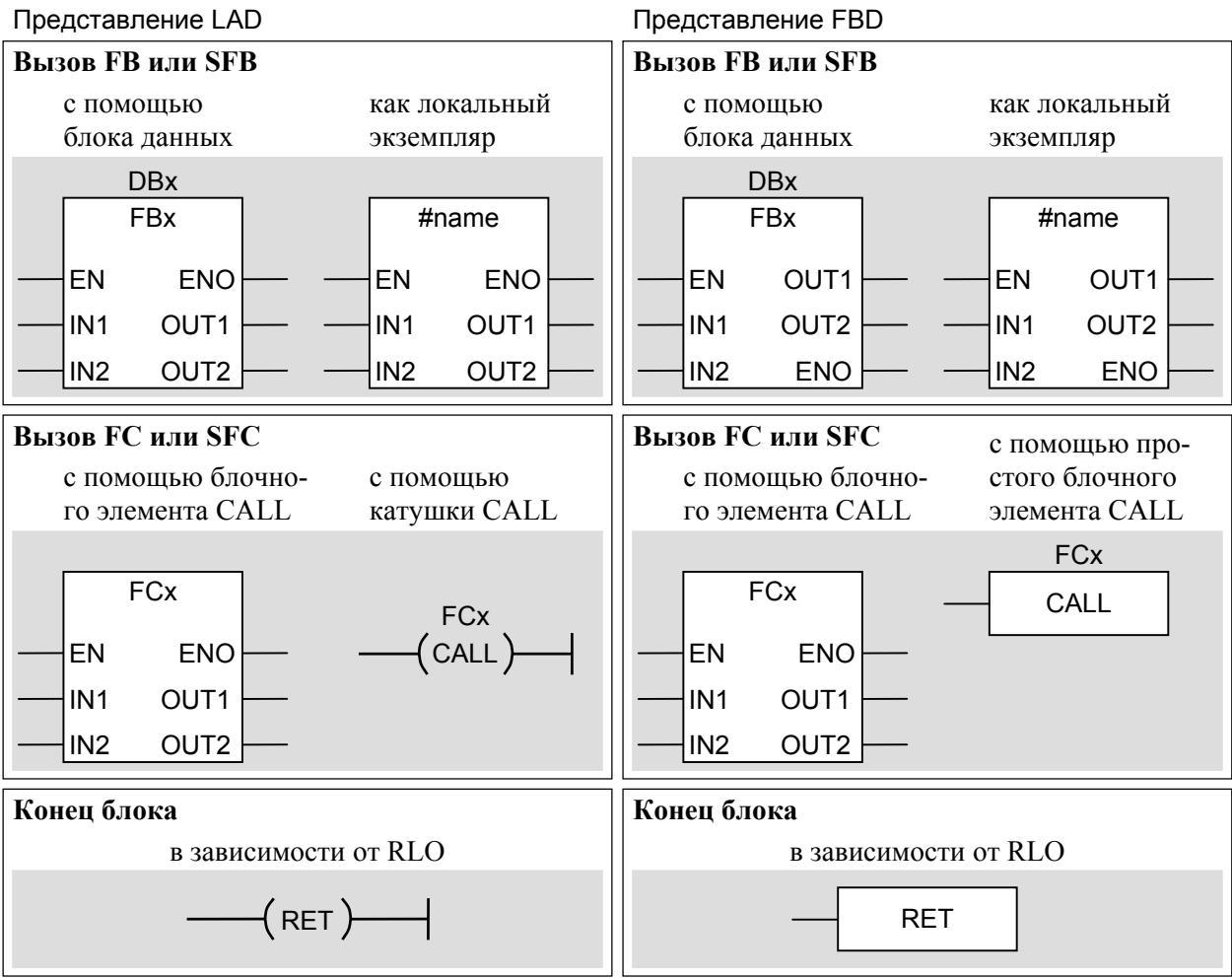


Рисунок 18.1 Функции для работы с кодовыми блоками

Кроме смены блока блочный элемент вызова также содержит пересылку параметров блока. Когда вызываются функциональные блоки, он также открывает экземпляр блока данных. Катушка / блочный элемент CALL есть ничто иное, как переход к другому блоку, и имеет значение (и допустим) только в случае функций и системных функций.

После завершения блока, вслед за функцией вызова CPU продолжает выполнение программы в блоке, который осуществил вызов (вызывающий блок). Если организационный блок завершен, то CPU передает управление операционной системе.

В случае пошагового программирования вы сможете найти катушки / блочные элементы CALL и RET в каталоге программных элементов (Program Elements Catalog), вызвав его командой меню View → Catalog (Вид → Каталог) [Ctrl – K], а также применив команду Insert → Program Elements (Вставка → Программные элементы), раздел «Program Control» («Программное управление»). Вы можете вставлять в программу вызовы блоков с использованием блочных элементов вызова при выборе блоков из «FC/FB/SFC/SFB blocks» («Блоки FC/FB/SFC/SFB»), «Multiple Instances» («Мультиэкземпляры») или «Libraries» («Библиотеки»).

18.1.1 Вызовы блоков: общая информация

Если кодовый блок предназначен для обработки, он должен быть «вызван». На рисунке 18.2 представлен пример вызова функции FC 10 в организационном блоке OB1.

Вызов блока состоит из блочного элемента вызова, который содержит адрес вызываемого блока (здесь: FC 10), разрешающий вход (enable input) EN, разрешающий выход (enable output) ENO и любые параметры блока. После обработки функции вызова CPU продолжает выполнение программы в вызванном блоке. Блок обрабатывается до конца или до встречи функции завершения блока. Затем CPU возвращается к вызывающему блоку (здесь: OB 1) и продолжает обработку этого блока после блочного элемента вызова.

Информация, необходимая CPU для нахождения обратного пути в вызывающий блок, сохраняется в стеке блоков (В-стек). С каждым новым вызовом блока создается новый элемент стека, включающий в себя адрес возврата, содержимое регистра блока данных и адрес локального стека данных вызывающего блока.

Если CPU в результате ошибки переходит в состояние STOP, то вы можете, используя программирующее устройство, увидеть по содержимому В-стека, какой блок обрабатывался до возникновения ошибки.

Вы можете переслать в вызываемый блок и из него данные для обработки. Эти данные передаются через параметры блока. Посредством блочного элемента вызова можно также вызывать блоки без параметров.

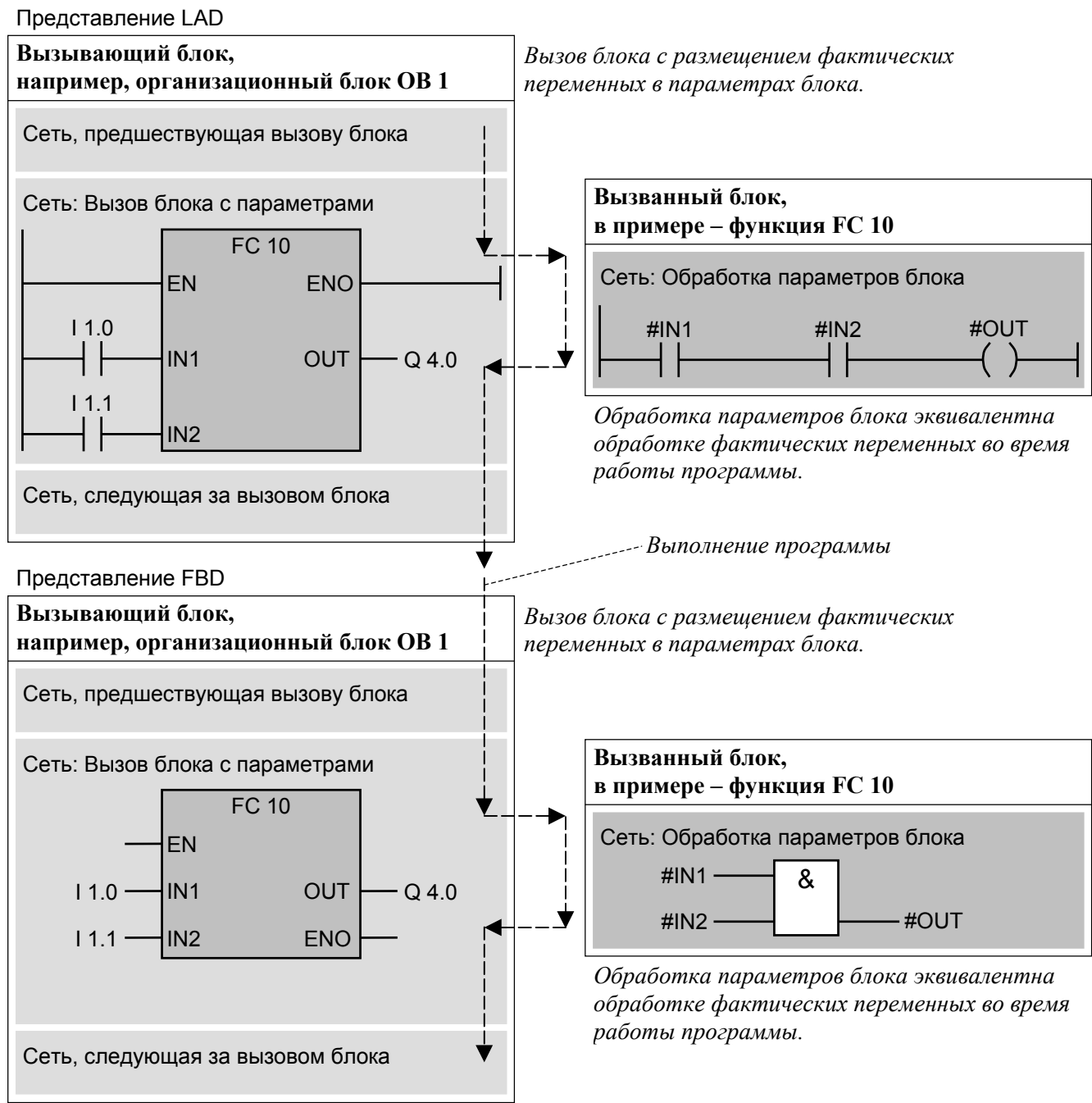


Рисунок 18.2 Пример вызова блока

18.1.2 Блочный элемент вызова

Блочный элемент вызова используется для вызова блоков FB, FC, SFB и SFC. (Вызывать организационные блоки нельзя, так как они управляются событиями и запускаются операционной системой.)

Для введения условий вызова можно воспользоваться входом EN блочного элемента вызова. Если вход EN подключен непосредственно к левой направляющей (шине питания), то такой вызов будет абсолютным, и он выполняется всегда. Если перед входом EN имеется логическая операция, то блочный элемент вызова выполняется,

только если предшествующая логическая операция выполнена. Выход ENO имеет то же сигнальное состояние, что и бинарный результат BR при осуществлении выхода из вызванного блока.

ЕСЛИ EN == “1” или не используется		ИНАЧЕ Вызванный блок не обрабатывается ENO := “0”
ТО		
Вызванный блок обрабатывается		
ЕСЛИ вызванный блок возвращает BR = “1”		
ТО	ИНАЧЕ	
ENO := “1”	ENO := “0”	ENO := “0”

Вы должны промаркировать (назначить) параметры вызванного блока с использованием абсолютных или символических операндов. Если тип параметра BOOL, то этому параметру должны предшествовать

- контакт или цепь (LAD) или
- двоичная переменная или бинарная логическая операция (FBD).

Логический выходной параметр в дальнейшем не может быть скомбинирован.

LAD: Вы можете скомпоновать несколько блочных элементов вызова последовательно, соединив их друг с другом через EN или ENO. Блочный элемент вызова может быть вставлен в параллельную цепь, если только он непосредственно соединен с левой направляющей (шиной питания).

FBD: Блочные элементы вызова можно соединить последовательно, подключив выход ENO элемента к входу EN следующего. Выходы ENO нескольких блочных элементов могут быть объединены по AND или OR.

Когда блок вызывается, MCR-зависимость деактивируется. MCR отменяется в вызванном блоке независимо от того, было MCR разрешено или заблокировано до вызова блока. При выходе из блока MCR-зависимость принимает установки, которые были у нее перед вызовом блока.

В зависимости от параметров блока вы можете изменить содержимое регистров блока данных при осуществлении смены блока. Если *вызванный* блок является функциональным блоком, то экземплярный блок всегда открывается в этом блоке через регистр DI. Если *вызывающий* блок относится к функциональным блокам, то содержимое регистра DI (экземплярный блок данных) после вызова блока сохраняется. Содержимое регистра DB зависит, помимо прочего, от передаваемых параметров блока.

Вызов функциональных блоков

Вызвать функциональный блок вы можете, выбрав соответствующий функциональный блок из раздела «FB Blocks» («Блоки FB») каталога программных элементов (Program Elements Catalog). Необходимым условием является наличие вызываемого функционального блока в пользовательской программе. Экземплярный блок данных, принадлежащий вызову, вы должны записать над блочным элементом. Оба блока (функциональный блок и экземпляр блока данных) могут иметь абсолютные и символические адреса.

В случае функциональных блоков при вызове не требуется инициализировать все параметры блока. Инициализируемые параметры блока сохраняют свои текущие значения.

Вы также можете вызвать «функциональные блоки с возможностью мультиэкземпляльности» в качестве локального экземпляра из других «функциональных блоков с возможностью мультиэкземпляльности». При этом вызываемый функциональный блок использует экземпляр блока данных вызывающего функционального блока как хранилище для своих локальных данных. Перед вызовом вы должны объявить локальный экземпляр в статических локальных данных вызывающего функционального блока (блока, который вы в настоящий момент программируете). Локальный экземпляр вызывается путем выбора одного из доступных локальных экземпляров в разделе «Multiple Instances» («Мультиэкземпляры») в каталоге программных элементов; экземплярный блок данных определять необязательно (см. также параграф 18.1.6 «Статические локальные данные»).

Вызов функций

Вызов функций осуществляется путем выбора соответствующей функции из раздела «FC Blocks» («Блоки FC») в каталоге программных элементов. Функция должна иметь абсолютный или символический адрес.

Когда вы вызываете функции, вы должны инициализировать все доступные параметры.

Вызов функций со значением функции принимает точно такую же форму, как и вызов функций без значения функции. Только первый выходной параметр, соответствующий значению функции, имеет имя RET_VAL.

Вызов системных блоков

Операционная система CPU содержит системные функции (SFC) и системные функциональные блоки (SFB), которыми вы можете воспользоваться. Количество и тип системных блоков зависит от CPU. Все системные блоки вы можете вызвать с помощью блочного элемента вызова.

Вызвать системный блок вы можете тем же способом, что и блок, написанный вами; настройте соответствующий экземпляр блока данных в пользовательской памяти с тем же типом данных, который имеет SFB.

Системная функция и функция, созданная вами, вызываются одинаково.

Системные блоки существуют только в операционной системе CPU. Если вы хотите вызвать системные блоки во время автономного программирования, программному редактору требуется описание интерфейса вызова, чтобы он мог инициализировать параметры. Вы найдете это описание интерфейса в разделе «System Function Blocks» («Системные функциональные блоки») в библиотеке «Standard Library» («Стандартная библиотека»). Отсюда программный редактор копирует описание интерфейса в контейнер автономных блоков, когда вы вызываете системный блок. Скопированное таким образом описание интерфейса затем появляется как «нормальный» блочный объект.

Каталог программных элементов предоставляет системные блоки, доступные в настоящий момент в пользовательской программе, в разделе «SFC Blocks» («Блоки SFC») или «SFB Blocks» («Блоки SFB»). Вы можете, к примеру, выбрать мышью системный блок из каталога программных элементов и перетащить его в блок, обрабатываемый в настоящий момент, где он позже вызывается. Одновременно этот блок (или точнее описание его интерфейса) копируется в контейнер блоков.

18.1.3 Катушка / блочный элемент CALL

Вы можете вызывать функции и системные функции с использованием катушки / блочного элемента CALL. При этом вызываемые блоки не должны иметь параметров. Если блок очень длинный или не достаточно понятен для вас, можно использовать катушку / блочный элемент CALL, просто «разбив» блок на разделы и вызывая разделы один за другим. В одной сети допускается одна катушка или один блочный элемент CALL.

LAD: Если катушка CALL соединена непосредственно с левой направляющей (шиной питания), то вызов выполняется всегда (безусловный вызов).

Если имеется логическая операция, предшествующая катушке CALL, то вызов осуществляется, только если выполняется предшествующая логическая операция, то есть если в катушке CALL течет ток. Если указанная логическая операция не выполнена, то вызова не происходит, и обрабатывается следующая сеть.

FBD: Если блочному элементу CALL предшествует логическая операция, то вызов выполняется, только когда успешно обработана логическая операция, то есть когда $RLO = \langle 1 \rangle$ присутствует на блочном элементе CALL. Если предыдущая логическая операция не выполнена, вызов не осуществляется, немедленно начинает обрабатываться следующая сеть.

Когда производится смена блоков, сбрасывается бит состояния OS; биты состояния CC0, CC1 и OV не затрагиваются.

При вызове блока деактивируется MCR-зависимость. В вызванном блоке MCR отключается, несмотря на то, был ли MCR разрешен или отменен перед вызовом блока. Когда осуществляется выход из блока, MCR-зависимость приобретает те же установки, которые она имела до вызова блока.

Вызов блока с применением катушки / блочного элемента CALL сохраняет регистры блока данных в В-стеке; завершение блока восстанавливает их содержимое, когда производится выход из блока. Текущий перед вызовом блока глобальный блок данных и экземплярный блок данных также открываются вслед за вызовом блока. Если до вызова блока не был открыт блок данных (например, в ОБ 1 нет экземпляра блока данных), то и после вызова блока блок данных также не открыт, несмотря на это, блоки данных могут открываться в вызванном блоке.

18.1.4 Функция завершения блока

Вы можете преждевременно завершить обработку в блоке с помощью функции окончания блока RET.

Условное завершение блока

Представление LAD: 

Представление FBD: 

Функция завершения блока представляется в виде катушки или блочного элемента, для которых требуется предшествующая логическая операция. В сети катушка / блочный элемент RET должен быть единственным.

Если предшествующая логическая операция выполнена, то производится выход из блока. Возвратный переход осуществляется в предыдущий обрабатываемый блок, в котором имел место вызов блока. Если завершается организационный блок, то CPU передаст управление системной программе.

Если предшествующая логическая операция не реализована, обрабатывается следующая сеть в блоке.

ЕСЛИ предшествующая логическая операция == "1"	
ТО	ИНАЧЕ
Выполняется выход из блока	Обрабатывается следующая сеть
BR := "1"	BR := "0"

Катушка / блочный элемент RET одновременно сохраняет RLO (протекал ли ток или нет) в бинарном результате BR независимо от того, была выполнена логическая операция или нет. Бинарный результат играет решающую роль в управлении выходом ENO блочного элемента вызова (см. также главу 15 «Биты состояния»).

18.1.5 Временные локальные данные

Вы можете использовать временные локальные данные для буферизации результатов, получаемых при обработке блока. Временные локальные данные доступны только во время обработки блока; по завершению обработки блока ваши буферизованные данные теряются.

Временные локальные данные – это операнды, которые располагаются в стеке локальных данных (L-стеке) в системной памяти. Операционная система CPU делает временные локальные данные доступными для кодового блока, когда этот блок вызывается. При вызове блока значения в L-стеке фактически случайные. Чтобы использование локальных данных имело смысл, вы перед чтением должны их заполнить. Когда блок завершается, L-стек назначается следующему вызываемому блоку.

Количество байт временных локальных данных, требующееся блоку, записано в заголовке блока. Прочитав заголовок, операционная система узнает, сколько байт должно быть зарезервировано в L-стеке при вызове блока. Вы также можете увидеть из записи в заголовке блока, сколько байт локальных данных необходимо блоку. Для этого в редакторе при открытом блоке выбирается команда File → Properties (Файл → Свойства), или пункт меню SIMATIC-менеджера Edit → Object Properties (Правка → Свойства объекта), в каждом случае используется вкладка «General – Part 2» («Общие – часть 2»).

Объявление временных локальных данных

Вы должны объявить временные локальные данные в разделе описаний кодового блока:

- под «temp» («временные») в случае пошагового программирования или
- между VAR_TEMP и END_VAR в случае программирования, ориентированного на источник (исходные файлы).

Рисунок 18.3 демонстрирует пример объявления временных локальных данных. Переменная *temp1* располагается во временных локальных данных и имеет тип INT; переменная *temp2* имеет тип REAL.

Временные локальные данные хранятся в L-стеке в порядке их описания и в соответствии с их типом данных.

Address (Адрес)	Declaration (Описание)	Name (Имя)	Type (Тип)	Initial value (Начальное значение)	Comment (Комментарий)
0.0	in	Man_on	BOOL	FALSE	Input parameter (Входной параметр)
2.0	out	Switch_on	BOOL	FALSE	Output parameter (Выходной параметр)
4.0	in_out	Length	INT	0	I/O parameter (Входной / выходной параметр)
6.0	stat	Total	INT	0	Static local data (Статические локальные данные)
8.0	stat	Setpoint	DINT	L#0	
0.0	temp	Deviation	INT		Temporary local data (Временные локальные данные)
2.0	temp	Intermediate	REAL		

Рисунок 18.3 Пример объявления локальных данных в функциональном блоке

Символическая адресация временных локальных данных

Ссылка на временные локальные данные происходит по их символическим именам. Вы должны назначать эти имена в соответствии с правилами для внутриблочных (локальных) символов.

Все операции, разрешенные для меркеров, также допустимы для временных локальных данных. Обратите внимание, однако, что бит временных локальных данных не подходит для использования в качестве меркера фронта, потому что он не сохраняет свое сигнальное состояние вне соответствующего блока.

Вы можете адресовать временные локальные данные для блока только внутри этого блока (исключение: временные локальные данные для вызывающего блока могут быть доступны через параметры блока)

Размер L-стека

Общий размер L-стека зависит от версии CPU. Доступное количество байтов временных локальных данных в приоритетном классе, то есть в программе организационного блока, также predetermined. В S7-300 объем временных локальных данных фиксирован, например, в CPU 314 на приоритетный класс выделено 256 байтов. В S7-400 вы можете при инициализации CPU задать требуемое количество байтов временных локальных данных. Эти байты должны быть поделены блоками, вызываемыми в соответствующих организационных блоках, а также блоками, которые они вызывают.

Обратите внимание, что редактор также использует временные локальные данные, к примеру, для передачи параметров блока, этот факт проходит незамеченным в ходе программирования интерфейса.

Первичная информация

Когда вызывается организационный блок, операционная система CPU передает первичную (стартовую) информацию во временные локальные данные. Эта первичная информация включает 20 байтов для каждого организационного блока и почти идентична для всех блоков ОВ. О первичной информации для различных организационных блоков подробно рассказывается в главах 20 «Главная программа», 21 «Обработка прерываний», 22 «Характеристики рестарта» и 23 «Обработка ошибок».

Эти 20 байтов должны быть всегда доступны в каждом используемом приоритетном классе. Если вы программируете процедуру для обнаружения синхронных ошибок (программные ошибки и ошибки доступа), то вы должны установить отдельно дополнительные 20 байтов (по крайней мере) для первичной информации этих организационных блоков обработки ошибок, так как эти ОВ обработки ошибок принадлежат одному приоритетному классу.

Вы описываете стартовую информацию для организационного блока при программировании этого блока. Эти данные являются обязательными. Простые описания на английском языке находятся в *Standard Library* (Стандартная библиотека) в разделе *Organization Blocks* (Организационные блоки). Если первичная информация вам не нужна, просто объявите первые 20 байтов как что-то другое, например, как массив (как показано на рисунке 18.4).

Address (Адрес)	Declaration (Описание)	Name (Имя)	Type (Тип)
0.0	temp	SINFO	ARRAY [1..20]
*1.0	temp		BYTE
20.0	temp	LByte	ARRAY [1..16]
*1.0	temp		BYTE

Рисунок 18.4 Пример объявления временных локальных данных в организационном блоке

Абсолютная адресация временных локальных данных

Обычно ссылка на временные локальные данные происходит по их символическим именам. Использование абсолютных адресов является исключением. Познакомившись со способом хранения данных в L-стеке, вы можете самостоятельно вычислить адреса статических локальных данных. Также вы можете увидеть список адресов в таблице описания (объявления) переменных компилированного блока.

Идентификатор операнда временных локальных данных – L; бит адресуется с L, байт – с LB, слово – с LW, а двойное слово – с LD.

Пример: Вам требуется зарезервировать 16 байт временных локальных данных для абсолютной адресации, и вы хотите обращаться к значениям в этих байтах по байтам и битам. Для этого создайте массив в начале области локальных данных, так чтобы

адресация начиналась с 0. В организационном блоке в должны поместить объявление этого массива сразу после описания первичной информации, в этом случае адресация начнется с 20.

Тип данных ANY

Переменная во временных локальных данных может быть – хотя это исключение – объявлена как переменная типа данных ANY (Любой). Вы можете использовать это свойство для изменения указателя ANY во время исполнения программы (см. параграф 24.2.5 «Переменный» указатель ANY»).

18.1.6 Статические локальные данные

Статические локальные данные – это операнды, которые функциональный блок хранит в своем экземпляре блока данных.

Статические локальные данные являются «памятью» функционального блока. Они сохраняют свои значения до изменения этих значений программой, как в случае операндов данных в глобальных блоках данных.

Число байт статических локальных данных ограничено типами данных переменных и определяемым версией CPU размером блока данных.

Объявление статических локальных данных

Вы можете объявить статические локальные данные в разделе описаний функционального блока:

- с типом объявления переменной (declaration) «stat» в случае пошагового программирования или
- между VAR и END_VAR в случае программирования, ориентированного на источник (исходные файлы).

Рисунок 18.3 параграфа 18.1.5 «Временные локальные данные» содержит пример объявления переменной в функциональном блоке. Параметры блока объявляются первыми, затем статические локальные данные и, наконец, временные локальные данные.

Статические локальные данные хранятся в экземпляре блока данных после параметров блока в порядке их описаний и в соответствии с их типами данных.

Символическая адресация статических локальных данных

Вы можете обращаться к статическим локальным данным с помощью символических имен. Назначаются эти имена в соответствии с правилами для внутриблочных (локальных) символов.

Все операции, которые могут адресовать операнды данных в глобальных блоках данных, могут также адресовать статические локальные данные.

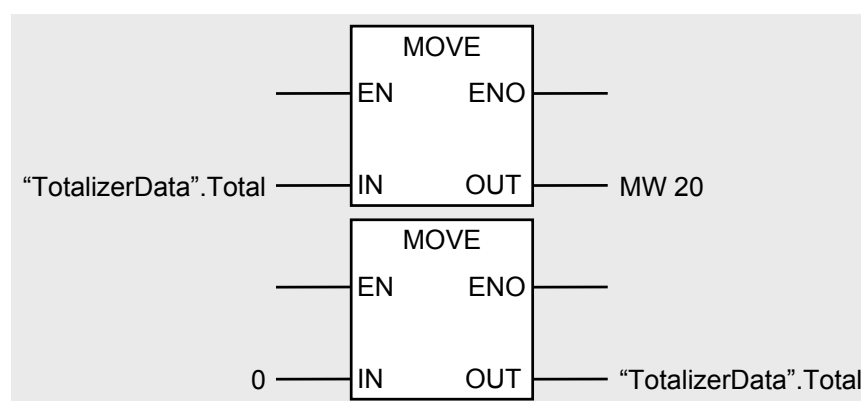
Пример: Функциональный блок «Totalizer» прибавляет входное значение к значению, хранящемуся в статических локальных данных, и сохраняет сумму в статических локальных данных. При следующем вызове входное значение вновь складывается с этой суммой и так далее (см. верхнюю часть рисунка 18.5).

Total – переменная в блоке данных «TotalizerData», экземпляром блока данных для функционального блока данных «Totalizer» (вы можете определить имена всех блоков самостоятельно в таблице символов, но при этом вы должны придерживаться применяемым правилам). Экземпляр блока данных имеет структуру данных функционального блока; в примере он содержит две переменных *In* и *Total* типа INT.

Доступ к статическим локальным данным вне функционального блока

Как правило, статические локальные данные обрабатываются в самом функциональном блоке, так как они хранятся в блоке данных. Однако, вы можете получить доступ к статическим локальным данным в любой момент времени с помощью «*Data Block Name*». *Operand Name* («Имя блока данных». *Имя операнда*), так же как к переменной в глобальном блоке данных.

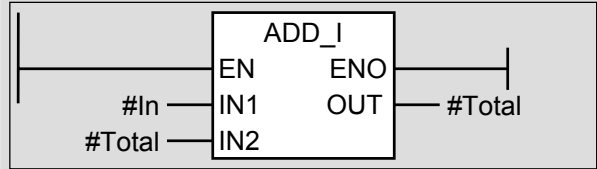
В нашем небольшом примере блок данных именуется *TotalizerData*, а операнд данных – *Total*. Применяемые инструкции доступа могут быть следующие.



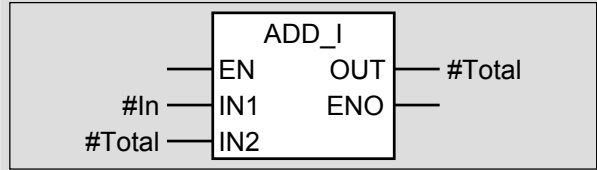
FB “Totalizer”

Address	Declaration	Name	Type
+ 0.0	in	In	INT
+ 2.0	stat	Total	INT

Представление LAD



Представление FBD



DB “TotalizerData”

Address	Declaration	Name	Type
+ 0.0	in	In	INT
+ 2.0	stat	Total	INT

В режиме просмотра данных (окне просмотра *data view*) блок данных отображает каждую отдельную переменную, так что переменные локального экземпляра выводятся с их полными именами.

В то же время вы можете видеть абсолютный адрес каждой переменной.

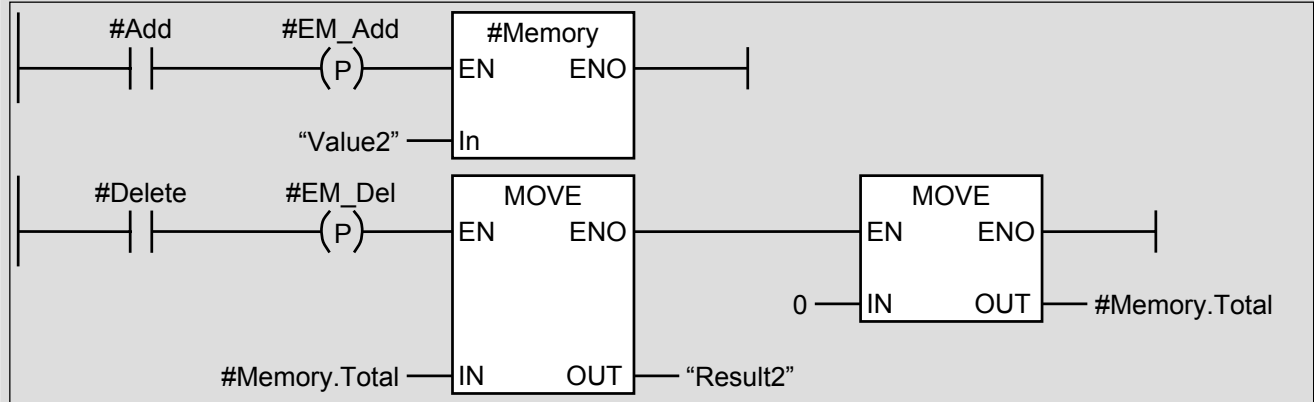
FB “Evaluation”

Address	Declaration	Name	Type
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

DB “EvaluationData”

Address	Declaration	Name	Type
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

Представление LAD



Представление FBD

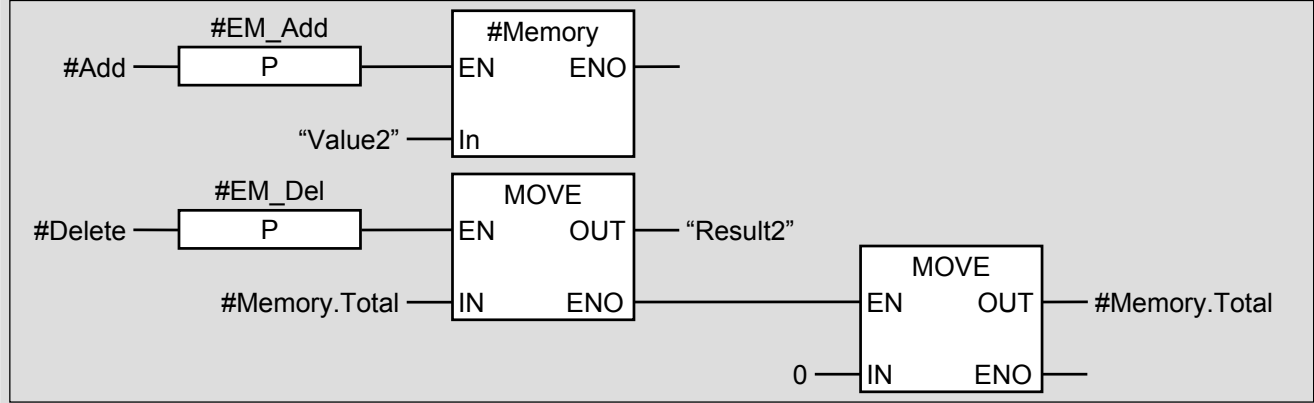


Рисунок 18.5 Пример статических локальных данных и локальных экземпляров

Локальные экземпляры

Когда вызывается функциональный блок, обычно определяется экземпляр блока данных для этого вызова. Затем функциональный блок сохраняет в этом экземпляре блока данных свои параметры блока и статические локальные данные.

Начиная со STEP 7 V2, имеется возможность генерирования «мультиэкземпляры», то есть вы можете вызывать функциональный блок как локальный экземпляр из другого функционального блока. Статические локальные данные (и параметры блока) вызванного функционального блока становятся подмножеством статических локальных данных вызывающего блока. Обязательным условием является то, что вызывающий и вызываемый блоки должны быть версии 2, то есть оба эти блока должны иметь «возможность мультиэкземпляльности». Это позволяет вам «вкладывать» вызовы блоков до глубины вложения 8.

Пример (рисунок 18.5, нижняя часть): В статических локальных данных функционального блока «Evaluation» объявляется переменная *Memory*, которая соответствует функциональному блоку «Totalizer», и имеет собственную структуру. Теперь вы можете вызвать функциональный блок «Totalizer» через переменную *Memory*, но без определения блока данных, так как данные для *Memory* хранятся «внутриблочно» («локально») в статических локальных данных (*Memory* является локальным экземпляром функционального блока «Totalizer»).

Вы можете обращаться к статическим локальным данным *Memory* в программе в функциональном блоке «Evaluation» точно так же, как и к компонентам структуры, то есть путем указания имени структуры (*Memory*) и имени компонента (*Total*).

Экземплярный блок данных «EvaluationData», таким образом, содержит переменные *Memory.In* и *Memory.Total*, которые также можно адресовать как глобальные переменные, например, как «EvaluationData». *Memory.Total*.

Приведенный пример по использованию локального экземпляра находится в функциональных блоках FB 6, FB 7 и FB 8 в программе «Program Flow Control» («Управление программным потоком») на прилагаемой к книге дискете.

В примере «Устройство подачи» параграфа 19.5.3 «Пример устройства подачи» показаны дополнительные способы применения локальных экземпляров.

Абсолютная адресация статических локальных данных

Как правило, статические локальные данные адресуются символически. Абсолютная адресация является исключением. В функциональном блоке экземпляр блока данных открывается через регистр DI. DI, следовательно, служит операндам в этом блоке данных, и сюда включаются статические локальные данные и параметры блока, в качестве идентификатора операнда. Бит адресуется с DIX, байт – с DIB, слово – с DIW и двойное слово – с DID.

Узнав, как данные хранятся в блоке данных, вы сами можете вычислить абсолютные адреса статических локальных данных. Вы также можете найти адреса в скомпилированном блоке в таблице описания переменных. Но будьте очень осторожны! *Эти адреса являются адресами, соответствующими началу экземпляра.* Они применимы, только когда вы вызываете функциональный блок с блоком данных. Если функциональный блок вызывается как локальный экземпляр, то локальные данные локального экземпляра располагаются в середине экземплярного блока данных вызывающего функционального блока. Вы можете просмотреть абсолютные адреса, например, в скомпилированном экземпляре блока данных, который содержит все локальные экземпляры. Если вы хотите прочитать адреса отдельных операндов локальных данных, выберите команду меню View → Data View (Вид → Просмотр данных).

Используя наш пример в качестве основы, переменная *Total* в функциональном блоке «Totalizer» может быть адресована с помощью DIW 2, если функциональный блок «Totalizer» был вызван с блоком данных (см. также операнды в блоке данных «TotalizerData»), и при помощи DIW 6, если функциональный блок «Totalizer» был вызван как локальный экземпляр в функциональном блоке «Evaluation» (см. также операнды в блоке данных «EvaluationData»).

18.2 Функции блоков для блоков данных

Данные программы хранятся в блоках данных. В принципе, для хранения данных вы также можете использовать область памяти меркеров; однако, блоки данных дают значительно больше возможностей благодаря объему данных, структурированию данных и типам данных. В этом параграфе демонстрируется

- как работать с операндами данных,
- как вызывать блоки данных и
- как создавать, удалять и тестировать блоки данных во время исполнения программы.

Вы можете использовать блоки данных в двух видах: как *глобальные блоки данных* (*global data blocks*), которые не назначаются кодовым блокам, и как *экземплярные блоки данных* (*instance data blocks*), которые назначаются функциональному блоку. Данные в глобальных блоках данных являются «свободными» данными, которыми может воспользоваться любой кодовый блок. Вы сами можете определить их объем и структуру непосредственно при программировании глобального блока данных. Экземплярный блок данных содержит только данные, с которыми работает соответствующий функциональный блок. Этот функциональный блок также определяет структуру и место хранения данных в «своем» экземпляре блока данных.

Количество и размер блоков данных зависит от версии CPU. Нумерация блоков данных начинается с 1; нет блока данных DB 0. Каждый блок данных может использоваться либо как глобальный блок данных, либо экземплярный блок данных.

Сначала вы должны создать блоки данных, используемые в вашей программе, либо запрограммировав, как кодовые блоки, либо при выполнении программы с использованием системной функции SFC 22 CREAT_DB.

18.2.1 Два регистра блоков данных

Каждый CPU S7 оснащен двумя регистрами блоков данных. Эти регистры содержат номера текущих блоков данных; это блоки данных, операнды которых в настоящий момент времени обрабатываются. Перед доступом к операнду блока данных в должны открыть блок данных, содержащий операнд. Если вы используете доступ по полному адресу (*fully-addressed access*) к операндам данных (со спецификацией блока данных см. ниже), то вам не нужно заботиться об открытии блоков данных или о содержимом регистров блоков данных. Редактор генерирует необходимые инструкции из ваших спецификаций.

Программный редактор использует первый регистр блоков данных главным образом для доступа к глобальным блокам данных, а второй регистр блоков данных – для обращения к экземплярам блоков данных. По этой причине указанные регистры называются «Регистр глобальных блоков данных» («Global Data Block Register», для краткости – регистр DB) и «Регистр экземплярных блоков данных» («Instance Data

Block Register», кратко – регистр DI). CPU обрабатывает регистры абсолютно идентичным образом. Каждый блок данных может быть открыт посредством одного из регистров (или с использованием двух одновременно).

Существенной особенностью является то, что вы можете воздействовать с помощью LAD или FBD только на регистр DB. Если вы открываете блок данных катушкой / блочным элементом OPN (см. ниже), то вы всегда делаете это через регистр DB. В LAD и FBD экземпляр блока данных всегда открывается через регистр DI; это относится к блочному элементу вызова, когда вызывается блок.

Когда вы загружаете слово данных, вы должны указать, какой из двух возможных открытых блоков данных содержит слово данных. Если блок данных открыт через регистр DB, то слово данных именуется DBW; если слово данных находится в блоке данных, открытом посредством регистра DI, оно называется DIW. Остальные операнды данных именуются соответственно (таблица 18.1).

Таблица 18.1 Операнды данных

Операнды данных	Расположены в блоке данных, открытом через	
	регистр DB	регистр DI
Бит данных	DBX y.x	DIX y.x
Байт данных	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 Доступ к операндам данных

Вы можете использовать следующие методы доступа к операндам данных:

- Символическая адресация по полному адресу,
- Абсолютная адресация по полному адресу и
- Абсолютная адресация по частичному адресу.

Символический доступ к операндам данных в глобальных блоках данных требует минимум системных знаний. Для абсолютного обращения или для применения обоих регистров блоков данных вам необходимо изучить материалы, приведенные ниже.

Символическая адресация операндов данных

Рекомендуется адресовать операнды данных символически всякий раз, когда это возможно. Символическая адресация

- делает программу более читаемой и понятной (если в качестве символов используются значащие термины),
- сокращает количество ошибок при написании программ (программный редактор сравнивает термины, используемые в таблице символов и в программе; «ошибки смены номера», например, DBB 156 и DBB 165, которые могут возникнуть при использовании абсолютных адресов, здесь не происходят) и
- не требуется знание программирования на уровне машинного кода (какой блок данных CPU открыл в настоящий момент?).

Символическая адресация использует доступ по полному адресу (блок данных вместе с операндом данных), поэтому операнд данных всегда имеет уникальный адрес.

Определить символический адрес операнда данных вы можете в два шага:

- 1) Назначение блока данных в таблице символов
Блоки данных являются глобальными данными, имеющими в программе уникальные адреса. В таблице символов вы должны назначить символ (например, Motor1) абсолютному адресу блока данных (например, DB 51).
- 2) Назначение операнда данных в блоке данных
Вы должны указать имена операндов данных (и типы данных) при программировании блока данных. Имя применяется только в соответствующем блоке (оно «внутриблочное» или локальное). Вы можете также использовать это же имя в другом блоке для другой переменной.

Доступ к операндам данных по полному адресу

В случае обращения по полному адресу вы определяете блок данных вместе с операндом данных. Этот метод адресации может быть символическим или абсолютным:

```
"MOTOR1".ACTVAL
DB 51.DBW 20
```

MOTOR1 – это символический адрес, назначенный блоку данных в таблице символов. ACTVAL является операндом данных, определенным при программировании блока данных. Символическое имя "MOTOR1".ACTVAL представляет собой такую же уникальную спецификацию (описание) операнда данных, как и DB 51.DBW 20.

Доступ по полному адресу возможен только в сочетании с регистром глобальных блоков данных (регистром DB). В случае полностью адресованных операндов данных редактор программ сначала открывает блок данных через регистр DB, а затем обращается к операнду данных.

Вы можете применять доступ по полному адресу вместе со всеми функциями, доступными для типа данных адресованного операнда данных. Тогда, используя параметры блока, к примеру, вы можете указать полностью адресованный операнд в качестве фактического параметра.

Абсолютная адресация операндов данных

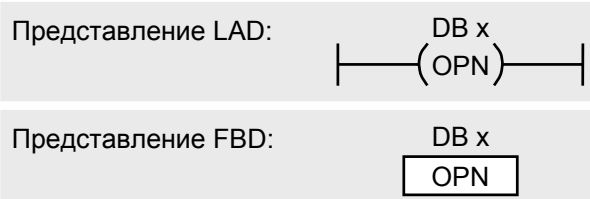
Для работы с абсолютной адресацией вы должны знать адреса, по которым редактор программ размещает операнды данных при их установке. Адреса вы можете узнать путем их вывода после программирования и компиляции блока данных: в столбце адресов вы можете увидеть абсолютный адрес, с которого начинается соответствующая переменная. Эта процедура подходит для всех блоков данных, как для используемых вами в качестве глобальных блоков данных, так и для экземплярных блоков данных (о локальных экземплярах рассказывается в параграфе 18.2.4 «Особые замечания по адресации данных»). Таким образом, вы можете также увидеть, где программный редактор хранит параметры блока и статические локальные данные для функциональных блоков.

Операнды данных адресуются побайтно (байт за байтом), как, например память меркеров; функции, работающие с операндами данных, те же, что и для памяти меркеров.

18.2.3 Открытие блока данных

Для того, чтобы открыть блок данных через регистр DB используются катушка OPN (LAD) или блочный элемент OPN (FBD).

Открытие блока данных



Катушка / блочный элемент OPN соединен непосредственно с левой направляющей (шиной питания) и является единственным в цепи. Указываемый блок данных всегда открывается через регистр DB. В LAD или FBD невозможно открыть блок данных катушкой / блочным элементом OPN посредством регистра DI. (Регистр DI использует блочный элемент вызова для открытия текущего экземпляра блока данных.)

Для пошагового программирования катушку / блочный элемент OPN вы можете найти в каталоге программных элементов (Program Elements Catalog) в разделе «DB Call» («Вызов DB»).

Открытый блок данных во время исполнения программы должен находиться в пользовательской памяти.

В сетях, следующих за катушкой / блочным элементом OPN, частичную адресацию вы можете использовать для доступа к тем операндам данных, которые расположены в открытом блоке данных. Если вы хотите скопировать из одного блока данных в другой, то вы можете, например, обратиться к временным локальным данным через промежуточный буфер или (лучше) использовать полностью адресованные операн-

ды данных. Замечание: полностью адресованные операнды данных перезаписывают регистр DB «своим» блоком данных.

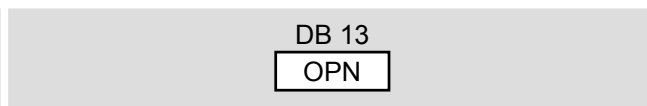
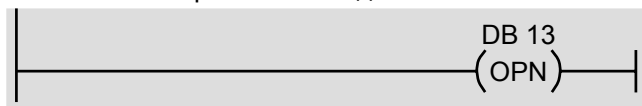
Пример: Значение слова данных DBW 10 блок данных DB 13 должно быть передано в слово данных DBW 16 блока данных DB 14. С помощью блочного элемента MOVE можно копировать частично адресованные операнды данных только внутри открытого в текущий момент блока данных. Для копирования между блоками данных вам потребуется промежуточный буфер, например, слово локальных данных (см. рисунок 18.6). Удобнее использовать полную адресацию.

Когда блок данных открыт, он остается «действующим» до открытия другого блока данных. По определенным обстоятельствам вы не сможете это увидеть в программном редакторе, к примеру, если блок меняется блочным элементом вызова (обратитесь к параграфу 18.2.4 «Особые замечания по адресации данных»).

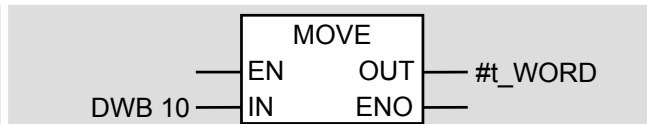
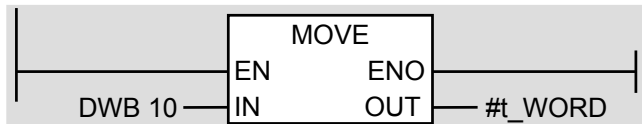
При смене блока с использованием катушки / блочного элемента CALL содержимое регистров блоков данных сохраняется. По возврату в вызывающий блок операция смена блока восстанавливает старое содержимое регистров.

Пример: Частично и полностью адресованные операнды данных

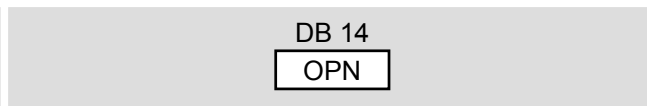
Network 5: Открытие блока данных DB 13



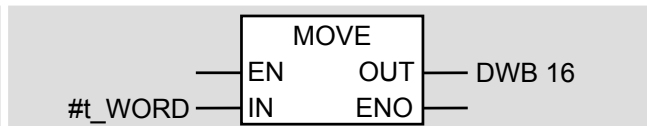
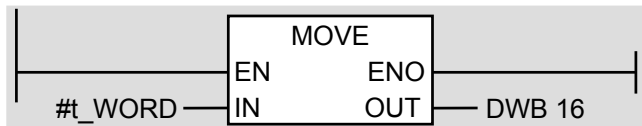
Network 6: Загрузка из DB 13 и копирование в промежуточный буфер



Network 7: Открытие блока данных DB 14



Network 8: Копирование из промежуточного буфера и передача в DB 14



Network 9: Копирование полностью адресованных операндов данных

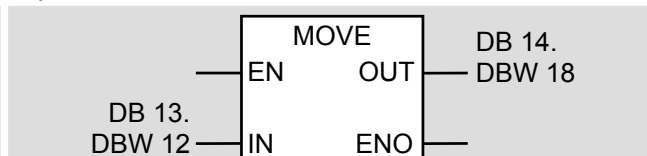
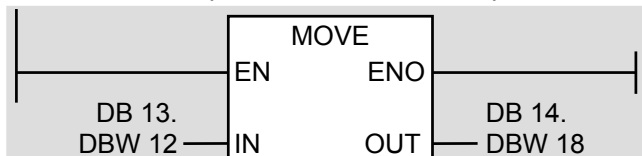


Рисунок 18.6 Пример частично и полностью адресованных данных

18.2.4 Особые замечания по адресации данных

Изменения содержимого регистров DB

Используя следующие функции, редактор программ генерирует дополнительные функции, которые могут внести изменения в содержимое двух регистров DB:

Полная адресация операндов данных

Каждый раз при полной адресации операндов данных редактор программ сначала открывает блок данных, затем обращается к операнду данных. Регистр DB перезаписывается каждый раз. Это также относится к поддержке параметров блока с использованием полностью адресуемых операндов данных.

Доступ к параметрам блоков

Доступ к следующим параметрам блоков изменяет содержимое регистра DB: все параметры блоков сложных типов данных в случае функций и входные/выходные (in/out) параметры сложных типов данных в случае функциональных блоков.

Вызов функционального блока с помощью блочного элемента вызова

Перед фактическим вызовом функционального блока блочный элемент вызова записывает номер текущего экземпляра блока данных в регистр DB (путем свопинга регистров блоков данных, то есть обмена их содержимым) и открывает экземплярный блок данных для вызываемого функционального блока. Таким образом, экземплярный блок данных, ассоциированный с вызываемым функциональным блоком, всегда открыт. После фактического вызова блока блочный элемент вызова снова проводит свопинг регистров блоков данных, поэтому текущий экземплярный блок данных снова доступен в вызывающем функциональном блоке. Таким способом блочный элемент вызова изменяет содержимое регистра DB.

Регистр DI в функциональных блоках

В функциональных блоках регистр DI всегда содержит номер текущего экземплярного блока данных. Весь доступ к параметрам блока и статическим локальным данным осуществляется через регистр DI. Адрес локальных данных, заданный в таблице описаний (declaration table) функционального блока, действует, только когда вы открываете функциональный блок с экземплярным блоком данных, в этом случае операнды данных начинаются с нулевого байта.

Когда вы вызываете функциональный блок как локальный экземпляр, его данные находятся «в середине» экземплярного блока данных вызывающего блока данных. Они содержат абсолютный адрес локальных данных, когда вы выведете экземплярный блок данных в формате просмотра данных. При этом каждая отдельная пере-

менная отобразится с именем и адресом, включая переменные вызванного локального экземпляра.

Когда вы программируете функциональный блок, который позже вы хотите также использовать как локальный экземпляр, вам еще не известен абсолютный адрес переменной во время написания программы. В этом случае – как поступает программный редактор при символическом программировании – содержимое адресного регистра AR2 прибавляется к адресу переменной. Однако, это возможно только в языке программирования STL.

Изменение содержимого блоков данных в более позднее время

В окне свойств (Properties) контейнера автономных объектов *Blocks* (Блоки) в регистре «Blocks» вы можете указать, абсолютный или символический адрес операнда данных будет иметь приоритет для последующих отображений и операций сохранения при изменении в содержимом блоков данных кодовых блоков, которые уже сохранены.

По умолчанию установлено «Absolute address has priority» («Приоритет имеет абсолютный адрес») (так же как в предыдущих версиях STEP 7). Эта установка по умолчанию означает, что когда происходит изменение в описании, абсолютный адрес сохраняется в программе, пока символический адрес соответственно изменяется. Если выбрано «Symbolic address has priority» («Приоритет имеет символический адрес»), абсолютный адрес меняется, в то время как символический адрес остается неизменным.

Пример: Допустим, что слово данных DBW 10 в блоке данных DB 1 содержит символический адрес *ActValue*. В программе вы можете адресовать это слово данных так:

```
"Data".ActValue      DB1.DBW 10
```

Здесь "Data" – это символический адрес блока данных DB 1. Если вы теперь добавите дополнительное слово данных с символическим адресом *MaxCurrent* сразу перед словом данных DBW 10, то результат при следующем открытии (или сохранении) кодового блока будет следующим:

в случае «Absolute address has priority» («Приоритет имеет абсолютный адрес»):

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

в случае «Symbolic address has priority» («Приоритет имеет символический адрес»):

```
"Data".ActValue      DB1.DBW 12
```

Сказанное о доступе к операндам данных в глобальных блоках данных относится и к доступу к глобальным операндам (входам экземпляров), для которых символические адреса были введены в таблице символов. Подробную информацию по этому предмету вы можете найти в параграфе 2.5.5 «Приоритет адреса».

18.3 Системные функции для блоков данных

Существуют три системных функции для работы с блоками данных. Их параметры описаны в таблице 18.2. Это функции

- **SFC 22 CREAT_DB**
Создает блок данных;
- **SFC 23 DEL_DB**
Удаляет блок данных;
- **SFC 24 TEST_DB**
Тестирует блок данных.

18.3.1 Создание блока данных

Системная функция **SFC 22 CREAT_DB** создает блок данных в пользовательской памяти. В качестве номера блока данных системная функция берет наименьший свободный номер из диапазона номеров, заданного входными параметрами **LOW_LIMIT** и **UP_LIMIT**. Номера, указанные в этих параметрах, включаются в диапазон номеров. Если эти значения одинаковы, блок данных генерируется с указанным номером. Выходной параметр **DB_NUMBER** предоставляет фактический номер созданного блока. Во входном параметре **COUNT** задается размер создаваемого блока. Размер соответствует количеству байт данных и должен быть четным числом.

Создание блока не то же самое, что его вызов. Текущий блок данных все еще действителен. Созданный с использованием рассматриваемой системной функции блок данных содержит случайные данные. Для осмысленного использования созданный таким образом блок данных сначала должен быть заполнен данными, и после этого может быть прочитан.

При возникновении ошибки блок данных не создается, содержимое выходного параметра неопределено, и в качестве значения функции возвращается номер ошибки.

18.3.2 Удаление блока данных

Системная функция **SFC 23 DEL_DB** удаляет блок данных, находящийся в памяти RAM (рабочая и загрузочная память), номер которого указан во входном параметре **DB_NUMBER**. Блок данных не должен быть открыт в момент удаления, иначе CPU переходит в состояние STOP.

Блоки данных, созданные с указанием ключевого слова **UNLINKED** (Неприсоединенный), и блоки данных на карте памяти **FEPR0M** не могут быть удалены системной функцией **SFC 23**.

В случае возникновения ошибки блок данных не удаляется, и в качестве значения функции возвращается номер ошибки.

18.3.3 Тестирование блоков данных

Системная функция **SFC 24 TEST_DB** возвращает количество доступных байтов для блока данных в пользовательской памяти (в выходном параметре DB_LENGTH) и состояние защиты от записи (в выходном параметре WRITE_PROT). Вы можете задать номер выбранного блока данных во входном параметре DB_NUMBER.

В случае ошибки содержимое выходных параметров неопределено, и в качестве значения функции возвращается номер ошибки.

Таблица 18.2 Функции SFC для управления блоками данных

SFC	Имя	Объявление	Тип данных	Назначение, описание
22	LOW_LIMIT	INPUT	WORD	Наименьший номер создаваемого блока данных
	UP_LIMIT	INPUT	WORD	Наибольший номер создаваемого блока данных
	COUNT	INPUT	WORD	Размер блока данных в байтах (четное число)
	RET_VAL	OUTPUT	INT	Информация об ошибке
	DB_NUMBER	OUTPUT	WORD	Номер созданного блока данных
23	DB_NUMBER	INPUT	WORD	Номер удаляемого блока данных
	RET_VAL	OUTPUT	INT	Информация об ошибке
24	DB_NUMBER	INPUT	WORD	Номер тестируемого блока данных
	RET_VAL	OUTPUT	INT	Информация об ошибке
	DB_LENGTH	OUTPUT	WORD	Размер блока данных (в байтах)
	WRITE_PROT	OUTPUT	BOOL	«1» = защищен от записи