SIEMENS

SIMATIC

STL для S7-300 и S7-400, Программирование

Справочное руководство

Это руководство является частью пакета документации с заказным номером:

6ES7810-4CA05-8BR0

C79000-G7000-C562-01

Указания по безопасности

Это руководство содержит указания, которые вы должны соблюдать для обеспечения собственной безопасности, а также защиты продукта и подключенного оборудования. Эти указания выделены в руководстве предупреждающим треугольником и помечены следующим образом в соответствии с уровнем опасности:

Опасность
Указывает, что несоблюдение надлежащих предосторожностей приведет к смерти, тяжким телесным повреждениям или существенному повреждению имущества.

Предупреждение
Указывает, что несоблюдение надлежащих предосторожностей может привести к смерти, тяжким телесным повреждениям или существенному повреждению имущества.

Предостережение
Указывает, что несоблюдение надлежащих предосторожностей может привести к небольшим телесным повреждениям или порче имущества.

Замечание
Привлекает ваше внимание к особенно важной информации о продукте, обращении с продуктом или к определенной части документации.

Квалифицированный персонал

К установке и работе на данном оборудовании должен допускаться только квалифицированный персонал. К квалифицированному персоналу относятся лица, имеющие право пускать в эксплуатацию, заземлять и маркировать электрические цепи, оборудование и системы в соответствии с установленным порядком и стандартами.

Правильное использование

Примите во внимание следующее:

Предупреждение
Это устройство и его компоненты могут быть использованы только для приложений, описанных в каталоге или технических описаниях, и только в соединении с устройствами или компонентами других производителей, которые были одобрены или рекомендованы фирмой Siemens.

Этот продукт может правильно и безопасно функционировать только при правильной транспортировке, хранении, установке и инсталляции, а также эксплуатации и обслуживании в соответствии с рекомендациями.

Торговые марки

SIMATIC®, SIMATIC HMI® и SIMATIC NET® являются зарегистрированными торговыми марками SIEMENS AG.

Некоторые из других обозначений, использованных в этих документах, также являются зарегистрированными торговыми марками; права собственности могут быть нарушены, если эти обозначения используются третьей стороной для своих собственных целей.

Siemens AG
Департамент техники автоматизации и приводов
Сфера деятельности: промышленные системы автоматизации
п/я 4848, D- 90327 Нюрнберг

Предисловие

Назначение

Это руководство является вашим путеводителем при создании программ пользователя на языке программирования *Список операторов*, часто сокращенно называемом STL (от англ. Statement List) или AWL (от нем. Anweisungsliste). В дальнейшем будет преимущественно использоваться сокращенное обозначение AWL.

Это руководство включает также справочный раздел, описывающий синтаксис и функции элементов AWL.

Круг читателей

Это руководство предназначено для программистов, операторов и обслуживающего персонала систем S7. Очень важно практическое знакомство с процедурами автоматизации.

Область применения

Это руководство действительно для версии 5.0 пакета программного обеспечения STEP 7.

Соответствие стандартам

AWL соответствует языку «Instruction List [Список команд]», определенному в стандарте Международной электротехнической комиссии IEC 1131–3, хотя имеются существенные отличия в отношении операций. Дополнительные подробности вы найдете в таблице стандартов в файле NORM_TBL.WRI пакета STEP 7.

Требования

Для эффективного использования данного руководства вы должны быть уже знакомы с теорией, на которую опирается программирование для S7 и которая задокументирована в оперативной помощи для STEP 7.

Языковые пакеты используют также стандартное программное обеспечение STEP 7, так что вы должны быть знакомы с тем, как обращаться с этим программным обеспечением, и прочитать сопроводительную документацию.

Документация	**Назначение**	**Номер для заказа**
Базовая информация о STEP 7, включающая в себя: • Working with STEP 7 V5.0, Getting Started Manual [Работа со STEP 7 версии 5.0. Введение в STEP 7] • Programming with STEP 7 V5.0 [Программирование с помощью STEP 7 версии 5.0] • Configuring Hardware and Communication Connections, STEP 7 V5.0 [Конфигурирование аппаратуры и проектирование соединений с помощью STEP 7 v5.0] • From S5 to S7, Converter Manual [От S5 к S7. Руководство по конвертированию]	Базовая информация для технического персонала, описывающая методы реализации задач управления с помощью STEP 7 и программируемых контроллеров S7-300/400.	6ES7810–4CA04–8BA0
Справочники по STEP 7, в том числе • Руководства Ladder Logic (LAD) /Function Block Diagram (FBD) /Statement List (STL) for S7-300/400 [Контактный план (LAD, КОР) /Функциональный план (FBD, FUP)/ Список операторов (STL, AWL) для S7-300/400] • Standard and System Functions for S7-300/400 [Стандартные и системные функции для S7-300/400]	Предоставляется справочная информация и описываются языки программирования LAD (контактный план, КОР), FBD (функциональный план, FUP) и STL (список операторов, AWL) и стандартные и системные функции, расширяя объем базовой информации о STEP 7.	6ES7810–4CA04–8BR0

Оперативные справки	**Назначение**	**Номер для заказа**
Справочная информация STEP 7	Базовая информация о программировании и конфигурировании аппаратуры с помощью STEP 7 в виде оперативной справки (online).	Часть стандартного программного обеспечения STEP 7.
Справочная информация о STL/LAD/FBD Справочная информация о SFB/SFC Справочная информация об организационных блоках	Контекстно-чувствительная справочная информация	Часть стандартного программного обеспечения STEP 7.

Доступ к оперативной помощи

Оперативную помощь вы можете отобразить следующими способами:

- Контекстно-чувствительная справочная информация о выделенном объекте с помощью команды меню **Help > Context–Sensitive Help [Помощь > Контекстно-чувствительная помощь],** с помощью **функциональной клавиши F1** или щелчком на символе вопросительного знака на панели инструментов.
- Справочная информация по STEP 7 через команду меню **Help > Contents [Помощь > Содержание].**

Ссылки

Ссылки на другую документацию даются с помощью номеров, заключенных между косыми чертами /.../. Используя эти номера, вы можете проверить точное название документа в разделе Литература в конце данного руководства.

Оперативные службы поддержки клиентов SIMATIC

Бригада поддержки клиентов SIMATIC предлагает существенную дополнительную информацию о продуктах SIMATIC через свои оперативные службы:

- Общая текущая информация может быть получена:
  - в **Internet** под `http://www.ad.siemens.de/simatic/html_00/simatic`
  - через **Fax-Polling** номер 08765–93 02 77 95 00
- Текущие данные о продукте и загрузки, которые вы, возможно, найдете полезными, доступны:
  - в **Internet** через http://www.ad.siemens.de/support/html_00/
  - через **Bulletin Board System** (BBS) в Нюрнберге (*SIMATIC Customer Support Mailbox)* под номером +49 (911) 895–7100.

Для набора почтового ящика используйте модем с протоколом до V.34 (28.8 кБод) со следующей настройкой параметров: 8, N, 1, ANSI; или наберите через ISDN (x.75, 64 кБод).

Дополнительная помощь

Если у вас есть другие вопросы, обращайтесь, пожалуйста, к представителю фирмы Siemens в вашем регионе. Адреса перечислены, например, в каталогах и в Compuserve (`go autforum`).

Наша горячая линия **SIMATIC Basic Hotline** тоже готова помочь:

- в Нюрнберге, Германия
  - с понедельника по пятницу с 07:00 до 17:00 (местное время): телефон: +49 (911) 895-7000
  - или E–mail: simatic.support@nbgm.siemens.de
- в Джонсон-Сити (TN), USA
  - с понедельника по пятницу с 08:00 до 17:00 (местное время): телефон: +1 423 461-2522
  - или E–mail: simatic.hotline@sea.siemens.com
- в Сингапуре
  - с понедельника по пятницу с 08:30 до 17:30 (местное время): телефон: +65 740-7000
  - или E–mail: simatic@singet.com.sg

Платная горячая линия SIMATIC Premium Hotline доступна круглосуточно по всему миру с помощью SIMATIC card (telephone: +49 (911) 895–7777).

Курсы по продуктам SIMATIC

Фирма Siemens предлагает ряд учебных курсов для ознакомления с системой автоматизации SIMATIC S7. Для получения более подробной информации обращайтесь в свой региональный учебный центр или в центральный учебный центр в Нюрнберге, Германия:
Телефон: +49 (911) 895–3154.

Содержание

1 Обзор продукта

Что такое Список операторов?

Список операторов (англ. Statement List, STL; нем. Anweisungsliste, AWL) – это текстовый язык программирования, который может быть использован для создания операторной части логического блока. Синтаксис его операторов похож на язык ассемблера и состоит из команд, за которыми следуют адреса (операнды), на которые команда действует. В дальнейшем будет обычно использоваться сокращение AWL.

Язык программирования AWL

Из языков программирования, с помощью которых можно программировать контроллеры S7, AWL наиболее близок к машинному коду MC7 процессора S7. Это значит, что при его использовании для программирования контроллеров S7, вы можете оптимизировать время исполнения и использование памяти.

Язык программирования AWL имеет все необходимые элементы для создания всей программы пользователя. Он содержит обширный набор команд. В вашем распоряжении имеется свыше 130 различных основных команд, а также широкий набор адресов. Функции и функциональные блоки позволяют структурировать вашу программу на AWL, делая ее более обозримой.

Программный пакет

Программный пакет AWL – это составная часть стандартного программного обеспечения STEP 7. Это значит, что после установки программного обеспечения STEP 7 вам доступны все функции редактирования, компиляции и тестирования/отладки для AWL.

Используя AWL, вы можете создать свою собственную пользовательскую программу:

- с помощью редактора пошагового ввода; при этом ввод структуры локальных данных облегчается с помощью табличных редакторов.
- с помощью исходного файла в текстовом редакторе; при этом ввод текста облегчается с помощью шаблонов блоков.

В стандартном программном обеспечении имеется три языка программирования: STL (AWL), FBD (FUP) и LAD (KOP). Вы можете переходить от одного языка программирования к другому почти без ограничений, выбирая наиболее подходящий язык для конкретного блока, который вы программируете.

Если вы пишете программу в LAD или FBD, то вы всегда можете перейти к представлению STL. Если вы преобразуете программу на языке LAD в программу на языке FBD и наоборот, то элементы программы, которые не могут быть представлены на целевом языке, отображаются на STL.

2 Структура и компоненты команд и операторов

Обзор главы

2.1 Структура оператора

Компоненты оператора

В зависимости от своей структуры оператор относится к одной из двух следующих основных групп (см. также рис. 2–1):

- оператор, состоящий только из команды (например, NOT, см. раздел 5.9)
- оператор, состоящий из команды и операнда (адреса) (см. таблицы с 2–1 по 2–5 и табл. 2–9)

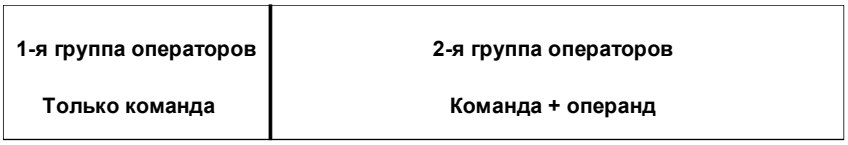


Рис. 2–1. Основные группы операторов

Операнд команды

Операнд команды задает константу или адрес, по которому команда находит значение (объект данных), с которым она должна выполнить операцию. Операнд может иметь символическое имя или абсолютное обозначение. Он может указывать на следующие элементы (см. также таблицы с 2–1 по 2–9):

- Константа, значение таймера или счетчика или строка символов ASCII, которые должны загружаться в аккумулятор 1 (например, L +27, см. табл. 2–1)
- Бит слова состояния программируемого логического контроллера (например, A UO, см. табл. 2–2)
- Символическое имя (например, A Motor.On, см. табл. 2–3)
- Блок данных и адрес внутри области этого блока данных (например, L DB4.DBD10, см. табл. 2–4)
- Функция (FC), функциональный блок (FB), встроенная системная функция (SFC) или встроенный системный функциональный блок (SFB) и номер функции или функционального блока (см. табл. 2–5)
- Идентификатор операнда и адрес внутри области памяти, задаваемой идентификатором операнда (например, A I 1.0 или A I [AR1,P#4.3], см. табл. 2–9)

Таблицы с 2–1 по 2–9 показывают различные операторы, каждый из которых состоит из команды и операнда.

Постоянные значения

Таблица 2–1 показывает, как можно использовать постоянное значение в качестве операнда команды.

Таблица 2–1. Операнды, указывающие на значение или строку символов

Оператор		**Описание**
Команда	**Операнд Константа**	
L	+27	Загрузить целое число 27 в аккумулятор 1.
L	'END'	Загрузить символы ASCII 'END' в аккумулятор 1.

Биты слова состояния

Операнд команды списка операций может обращаться к одному или более битам слова состояния программируемого логического контроллера (см. раздел 2.2). Команда опрашивает состояние сигнала отдельного бита слова состояния (например, A BR) и реагирует на него или интерпретирует комбинацию из двух битов (например, A UO).

Таблица 2–2. Операнды, ссылающиеся на бит слова состояния

Оператор		**Описание**
Команда	**Операнд Бит слова состояния**	
A	BR	1 или 0 в бите 8 слова состояния включается в булеву логическую комбинацию.
A	UO	Команда интерпретирует комбинацию, которую она находит в битах CC 1 и CC 0 слова состояния, чтобы выяснить, выполняется ли определенное условие. Например, комбинация из 1 и 1 означает «недопустимо», т.е. одно из значений в операции с плавающей точкой не было в действительности числом с плавающей точкой.

Символическое имя

Таблица 2–3 показывает, как использовать символическое имя в качестве операнда команды. Вы можете использовать символические имена в операторах AWL только после того, как вы их описали: глобальные символические имена должны быть введены в таблицу символов, а локальные имена – в блок.

Таблица 2–3. Операнды, указывающие на символическое имя

Оператор		**Описание**
Команда	**Операнд**	
	Символ	
A	Motor.On	Выполнить логическую операцию И с битом, имеющим символическое имя «Motor.On». В этом примере символическое имя «Motor.On» может представлять только один бит из области памяти блока данных (D) или элемент структуры «MOTOR» .
L	SPEED	Загрузить значение в виде байта, слова или двойного слова с символическим именем SPEED в аккумулятор 1 .

Блок данных и адрес в блоке данных

Таблица 2–4 показывает, как использовать блок данных и адрес в этом блоке данных в качестве операнда команды.

Таблица 2–4. Операнды, указывающие на блок данных или адрес в блоке данных

Оператор		**Описание**
Команда	**Операнд**	
	Блок данных и адрес	
L	DB4.DBD10	Загрузить двойное слово данных DBD10 из блока данных DB4 в аккумулятор 1.
A	DB10.DBX4.3	Выполнить логическую операцию И с битом данных DBX4.3 из блока данных DB10.

FC, FB, SFC и SFB

Таблица 2–5 показывает, как использовать функцию (FC), функциональный блок (FB), встроенную системную функцию (SFC), встроенный системный функциональный блок (SFB) и номер функции или функционального блока в качестве операнда команды.

Таблица 2–5. Операнды, указывающие на функцию, функциональный блок, системную функцию или системный функциональный блок

Оператор		**Описание**
Команда	**Операнд FC, FB, SFC, SFB и номер**	
CALL	FB10, DB10	Вызвать функциональный блок FB10 с экземплярным блоком данных DB10.
CALL	SFC43	Вызвать встроенную системную функцию SFC43.

Идентификаторы операндов

Некоторые операнды включают в себя идентификатор операнда и адрес внутри области памяти, указанной в идентификаторе операнда. Идентификатор операнда может быть одного из следующих трех основных типов (см. таблицы с 2–6 по 2–8):

- Идентификатор операнда, указывающий область памяти и размер объекта данных в этой области следующим образом (см. табл. 2–6):

- Область памяти, в которой команда находит значение (объект данных), с которым она должна выполнить операцию (например, «I» для области входов образа процесса)
- Размер значения (объекта данных), с которым команда должна выполнить операцию (например, B для «байта,» W для «слова» и D для «двойного слова»)

- Идентификатор операнда, который указывает область памяти, но не указывает размер объекта данных в этой области (например, идентификатор, указывающий область T для «таймера», C для «счетчика» или DB или DI для «блока данных» плюс номер этого таймера, счетчика или блока данных, см. табл. 2–7)
- Идентификатор операнда, который указывает размер объекта данных, но не область памяти. Область памяти закодирована в адресе, который следует за идентификатором операнда (см. табл. 2–8).

Таблица 2–6. Идентификатор операнда, задающий область памяти и размер объекта данных

Тип адресации	**Команда**	**Идентификатор операнда**		**Адрес операнда**
		Область памяти	**Размер объекта данных (если размер не указан, то подразумевается бит)**	
Прямая	A	I		0.0
Прямая	L	I	B	10
Косвенная через память	A	I		[MD2]
Косвенная через память	L	I	B	[DID4]
Косвенная внутри области через регистр	A	I		[AR1, P#4.3]
Косвенная внутри области через регистр	T	L	D	[AR2, P#53.0]

Таблица 2–7. Идентификатор операнда, задающий область памяти, но не задающий размер объекта данных

Тип адресации	**Команда**	**Идентификатор операнда: Область памяти**	**Номер или адрес номера**
Прямая	OPN	DB	5
Прямая	SP	T	7
Косвенная через память	OPN	DB	[LW2]
Косвенная через память	S	C	[MW44]

Таблица 2–8. Идентификатор операнда, задающий размер объекта данных, но не задающий область памяти

Тип адресации	**Команда**	**Размер объекта данных (если размер не указан, то подразумевается бит.)**	**Адрес операнда**
Косвенная с указанием области через регистр	A		[AR1, P#4.3]
Косвенная с указанием области через регистр	L	B	[AR1, P#100.0]

Таблица 2–9. Операнды, состоящие из идентификатора операнда и адреса операнда

Оператор			**Описание**
Команда	**Операнд**		
	Идентифи-катор операнда	**Адрес в области памяти или регистре**	
A	I	1.0	Выполнить логическую операцию И с входным битом I 1.0.
A	I	[MD2]	Выполнить логическую операцию И с входным битом, точный адрес которого находится в двойном слове памяти MD2.
L	C	1	Загрузить счетное значение счетчика 1 в аккумулятор 1.

Работа со словом или двойным словом как с объектом данных

Если вы работаете с командой, идентификатор операнда которой задает область памяти вашего программируемого логического контроллера, и с объектом данных, который по размеру является словом или двойным словом (см. табл. 2–6), то вы должны знать, что на адрес памяти всегда ссылаются как на адрес байта. Этот адрес байта является номером самого старшего байта слова или двойного слова. Например, операнд оператора, показанного на рис. 2–2, обращается к четырем следующим друг за другом байтам в области памяти M, начинающимся байтом 10 (MB10) и заканчивающимся байтом 13 (MB13).

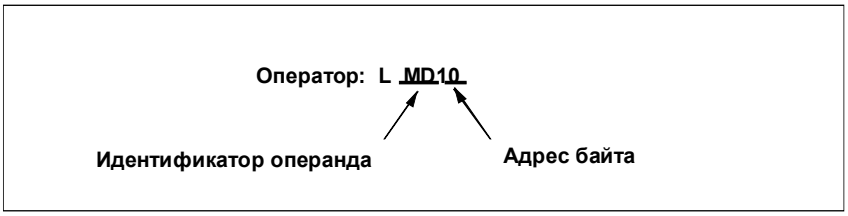


Рис. 2–2. Пример адреса памяти, к которому обращаются как к адресу байта

Рис. 2–3 иллюстрирует объекты данных следующих размеров:

- двойное слово: двойное слово памяти MD10
- слово: слова памяти MW10, MW11 и MW13
- байт: байты памяти MB10, MB11, MB12 и MB13

Если вы используете абсолютные операнды, являющиеся словом или двойным словом, то убедитесь, что вам удалось избежать перекрытия байтов разных слов.

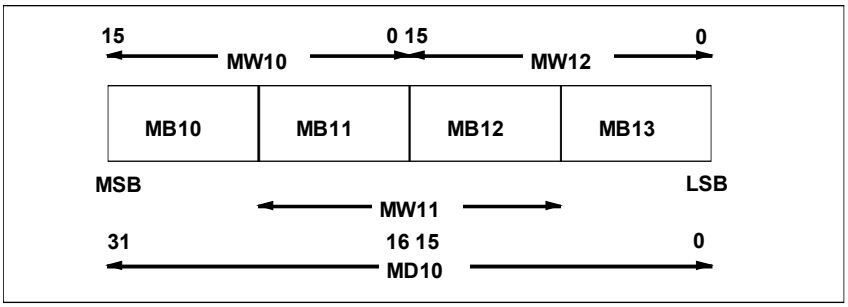


Рис. 2–3. Обращение к адресу памяти как к байтовому адресу

Области памяти и их функции

Большинство операндов в AWL обращаются к областям памяти. В следующей таблице перечислены области памяти и описаны функции каждой области.

Таблица 2–10. Области памяти и их функции

Имя области	**Функция области**	**Доступ к области**	
		через единицы следующего размера:	**Сокращение**
Вход образа процесса	В начале цикла сканирования операционная система считывает входы из процесса и записывает эти значения в эту область. Программа может использовать эти значения при циклической обработке.	Входной бит Входной байт Входное слово Входное двойное слово	I IB IW ID
Выход образа процесса	При выполнении цикла сканирования программа рассчитывает выходные значения и помещает их в эту область. В конце цикла сканирования операционная система считывает рассчитанные выходные значения из этой области и передает их на выходы процесса.	Выходной бит Выходной байт Выходное слово Выходное двойное слово	Q QB QW QD
Битовая память	Эта область предоставляет место для хранения промежуточных результатов, рассчитанных в программе.	Бит памяти (меркер) Байт памяти (меркерный байт) Слово памяти (меркерное слово) Двойное слово памяти (двойное меркерное слово)	M MB MW MD
Периферия: внешний вход Периферия: внешний выход	Эта область обеспечивает прямой доступ вашей программе к модулям ввода и вывода (т.е., к периферийным входам и выходам).	Периферийный входной байт Периферийное входное слово Периферийное входное двойное слово Периферийный выходной байт Периферийное выходное слово Периферийное выходное двойное слово	PIB PIW PID PQB PQW PQD
Таймеры	Таймеры – это функциональные элементы программирования на AWL. Эта область предоставляет место для хранения таймерных ячеек. В этой области генератор тактовых импульсов обращается к ячейкам времени, чтобы обновить их путем уменьшения значения времени, а таймерные команды получают доступ к ячейкам времени.	Таймер (T)	T
Счетчики	Счетчики – это функциональные элементы программирования на AWL. Эта область предоставляет место для хранения счетчиков. Операции счета получают доступ к ним здесь.	Счетчик (C)	C
Блок данных	Эта область содержит данные, к которым можно обратиться из любого блока. Если вам нужно иметь два различных блока данных, открытых одновременно, то один из них можно открыть оператором «OPN DB», а второй – оператором «OPN DI». Благодаря этому CPU может отличить, к какому из двух блоков данных хочет обратиться ваша программа, когда открыты оба блока. Хотя вы можете использовать оператор «OPN DI» для открытия любого блока данных, этот оператор в основном используется для открытия экземплярных блоков данных, связанных с функциональными блоками (FB) и системными функциональными блоками (SFB). За дополнительной информацией о FB, SFB и экземплярных блоках данных обращайтесь к оперативной справке *STEP 7.*	Блок данных, открытый оператором «OPN DB»: Бит данных Байт данных Слово данных Двойное слово данных	 DBX DBB DBW DBD

Таблица 2–10. Области памяти и их функции

Имя области	Функция области	Доступ к области через единицы следующего размера:	Сокращение
		Блок данных, открытый оператором «OPN DI»:	
		Бит данных	DIX
		Байт данных	DIB
		Слово данных	DIW
		Двойное слово данных	DID
Локальные данные	Эта область содержит временные данные, используемые внутри логического блока (OB, FB или FC). Эти данные называются также динамическими локальными данными. Они служат в качестве промежуточного буфера. Когда логический блок завершает работу, эти данные теряются. Эти данные хранятся в стеке локальных данных (L-стек).	Бит временных локальных данных	L
		Байт временных локальных данных	
		Слово временных локальных данных	LB
		Двойное слово временных локальных данных	LW
			LD

Таблица 2–11 перечисляет максимальные диапазоны адресов различных областей памяти. Диапазон адресов, возможный в вашем CPU, вы найдете в технических данных CPU. Функции областей памяти объясняются в таблице 2–10.

Таблица 2–11. Области памяти и их диапазоны адресов

Имя области	Доступ к области через единицы следующего размера:	Сокращение	Максимальный диапазон адресов
Вход образа процесса	Входной бит	I	от 0.0 до 65,535.7
	Входной байт	IB	от 0 до 65,535
	Входное слово	IW	от 0 до 65,534
	Входное двойное слово	ID	от 0 до 65,532
Выход образа процесса	Выходной бит	Q	от 0.0 до 65,535.7
	Выходной байт	QB	от 0 до 65,535
	Выходное слово	QW	от 0 до 65,534
	Выходное двойное слово	QD	от 0 до 65,532
Битовая память	Бит памяти (меркер)	M	от 0.0 до 255.7
	Байт памяти (меркерный байт)	MB	от 0 до 255
	Слово памяти (меркерное слово)	MW	от 0 до 254
	Двойное слово памяти (двойное меркерное слово)	MD	от 0 до 252
Периферия: внешний вход	Периферийный входной байт	PIB	от 0 до 65,535
	Периферийное входное слово	PIW	от 0 до 65,534
	Периферийное входное двойное слово	PID	от 0 до 65,532
Периферия: внешний выход	Периферийный выходной байт	PQB	от 0 до 65,535
	Периферийное выходное слово	PQW	от 0 до 65,534
	Периферийное выходное двойное слово	PQD	от 0 до 65,532
Таймеры	Таймер (T)	T	от 0 до 255
Счетчики	Счетчик (C)	C	от 0 до 255
Блок данных	Блок данных, открытый оператором «OPN DB»:		
	Бит данных	DBX	от 0.0 до 65,535.7
	Байт данных	DBB	от 0 до 65,535
	Слово данных	DBW	от 0 до 65, 534
	Двойное слово данных	DBD	от 0 до 65,532
	Блок данных, открытый оператором «OPN DI»:		
	Бит данных	DIX	от 0.0 до 65,535.7
	Байт данных	DIB	от 0 до 65,535
	Слово данных	DIW	от 0 до 65, 534
	Двойное слово данных	DID	от 0 до 65,532

Таблица 2–11. Области памяти и их диапазоны адресов

Имя области	Доступ к области через единицы следующего размера:	Сокращение	Максимальный диапазон адресов
Локальные данные	Бит временных локальных данных	L	от 0.0 до 65,535.7
	Байт временных локальных данных	LB	от 0 до 65,535
	Слово временных локальных данных	LW	от 0 до 65, 534
	Двойное слово временных локальных данных	LD	от 0 до 65,532

2.2 Значение регистров CPU в операторах

Аккумуляторы

Два 32–битных аккумулятора – это регистры общего назначения, используемые для обработки байтов, слов и двойных слов. Вы можете загружать константы и значения из памяти в качестве операндов в аккумулятор и выполнять над ними логические операции. Вы можете также передать результат операции из аккумулятора 1 по указанному адресу. Рис. 2–4 показывает области аккумулятора.

Стековый механизм для управления аккумулятором выглядит следующим образом:

- Команда загрузки всегда воздействует на аккумулятор 1 и сохраняет старое содержимое в аккумуляторе 2.
- Команда передачи не изменяет аккумуляторы (за исключением команд TAR1 и TAR2).
- Команда TAK обменивает содержимое аккумуляторов 1 и 2.

За информацией об управлении аккумуляторами для арифметических команд обратитесь к разделу 9.1.

Рис. 2–4. Области аккумулятора

Скобочный стек

Скобочный стек – это область памяти, имеющая ширину один байт. Эта область памяти используется скобочными командами A(, O(, X(, AN(, ON(, XN(. Эти команды сохраняют текущий результат логической операции (RLO) в скобочном стеке и начинают новую логическую цепочку.

Скобочный стек может принять семь записей. Запись скобочного стека состоит из битов RLO, BR и OR слова состояния и кода функции, указывающего, какая из булевых логических операций должна использоваться (A, AN, O, ON, X или XN).

Команда «)»завершает скобочное выражение, выполняя следующие функции:

- Извлекает запись из скобочного стека
- Восстанавливает биты OR и BR
- Определяет новый RLO, логически комбинируя текущий RLO (т.е. RLO из выражения, заключенного в скобки) с RLO записи стека в соответствии с кодом функции (см. раздел 5.4)

Рисунок 2–5 показывает структуру записи в скобочном стеке. Под этим рисунком вы найдете объяснение битов байта скобочного стека.

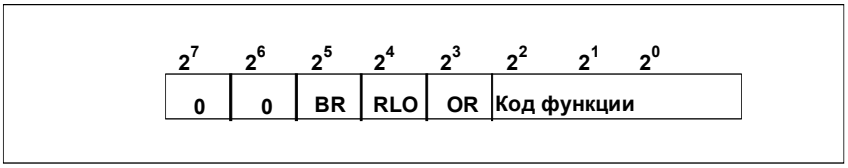


Рис. 2–5. Структура записи в скобочном стеке

Байт скобочного стека содержит следующие биты (см. рис. 2–5):

- неназначенные биты (биты 7 и 6 с состоянием сигнала «0»)
- сохраняемый двоичный результат (BR)
- сохраняемый результат логической операции (RLO)
- сохраняемый бит OR в функциях A(и AN(
  Во всех остальных функциях сохраняется ноль
- код функции (в битах 2, 1 и 0)

Код функции

С помощью кода функции команда «)» определяет функцию, которая должна использоваться для комбинирования текущего RLO (т.е. RLO из выражения, заключенного в скобках) с RLO записи скобочного стека. Таблица 2–12 показывает комбинации битов кода функции для каждого вида функции:

Таблица 2–12. Код функции из байта скобочного стека

Команда	Код функции 2	Код функции 1	Код функции 0
A(	0	0	0
AN(	0	0	1
O(	0	1	0
ON(	0	1	1
X(	1	0	0
XN(	1	0	1

Скобочный стек с записями и указателем

Скобочный стек и указатель этого стека должны сохраняться в стеке прерываний или извлекаться из него при переходе от одного уровня вложения к другому. Число в указателе скобочного стека показывает, количество записей, имеющихся в этом стеке (см. рис. 2–6).

Рис. 2–6. Структура скобочного стека с записями и указателем

Слово состояния

Слово состояния содержит биты, к которым вы можете обратиться в операндах логических команд, выполняемых с битами и словами.
Рис. 2–7 показывает структуру слова состояния. Разделы, следующие за этим рисунком, объясняют значение битов с 0 по 8.

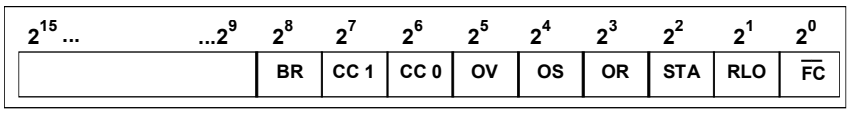


Рис. 2–7. Структура слова состояния

Первичный опрос

Бит 0 слова состояния называется битом первичного опроса (бит FC, см. рис. 2–7). Состояние сигнала 0 в бите FC указывает, что за этой точкой в вашей программе следующая логическая команда начинает новую цепь логических операций. (Черта над FC показывает, что этот бит берется с отрицанием).

Каждая логическая команда опрашивает состояние сигнала бита FC, а также состояние сигнала операнда, к которому она обращается. Если бит FC равен 0, то команда сохраняет результат опроса состояния сигнала в бите результата логической операции слова состояния (бит RLO, см. следующий раздел) и устанавливает бит FC в 1. Этот процесс называется первичным опросом (см. рис. 2–8 и раздел 5.6).

Если состояние сигнала бита FC равно 1, то команда логически сопрягает результат опроса состояния сигнала операнда, к которому она обращается, со значением, хранящимся в предыдущем бите RLO (см. рис. 2–8).

Цепь логических команд всегда заканчивается командой вывода (S, R или =, см. разделы 5.7 и 5.8), командой перехода, связанной с результатом логической операции (JC, см. раздел 16) или одной из скобочных команд A(, O(, X(, AN(, ON(или XN((см. раздел 5.4). Каждая такая команда сбрасывает бит FC в 0 (см. рис. 2–8).

Результат логической операции

Бит 1 слова состояния называется битом RLO (RLO означает «result of logic operation [результат логической операции]», см. рис. 2–7). Этот бит хранит результат битовой логической операции или операции сравнения.

Например, вторая команда в цепи битовых логических команд опрашивает состояние сигнала операнда и дает результат 1 или 0. Затем эта команда логически сопрягает этот результат со значением, хранящимся в бите RLO слова состояния в соответствии с правилами булевой логики (см. выше *Первичный опрос* и главу 5). Результат этой логической операции сохраняется в бите RLO слова состояния, заменяя предыдущее значение бита RLO. Каждая последующая команда в цепи выполняет логическую операцию с двумя значениями: результатом, полученным при опросе командой операнда, и текущим значением RLO.

Вы можете устанавливать RLO в 1 безусловно с помощью команды SET; вы можете сбрасывать RLO в 0 безусловно с помощью команды CLR. Вы можете использовать булеву битовую логическую команду при первичном опросе для присваивания RLO состояния содержимого ячейки битовой памяти. Вы можете использовать RLO для запуска команд перехода.

Программа на AWL	Состояние сигнала входа (I) или выхода (Q)	Результат опроса	Бит RLO	Бит $\overline{FC}$	Объяснение
				0	Бит $\overline{FC}$= 0 указывает, что следующая команда начинает логическую цепь.
A I 1.0	1	1	1	1	Результат первичного опроса __ сохраняется в бите RLO. Бит $\overline{FC}$ устанавливается в 1.
AN I 1.1	0	1	1	1	Результат опроса сопрягается с предыдущим RLO в соответствии с таблицей истинности для И. Бит $\overline{FC}$ остается равным 1.
= Q 4.0	1			0	Значение RLO присваивается выходу. Бит $\overline{FC}$ сбрасывается в 0.

Рис. 2–8. Влияние состояния сигнала бита $\overline{FC}$ на логические операции

Бит состояния

Бит состояния (бит STA) сохраняет значение бита, к которому происходит обращение. Состояние битовой команды, которая имеет доступ к памяти на чтение (A, AN, O, ON, X, XN), всегда равно значению бита, опрашиваемого этой командой (т.е. бита, с которым она выполняет логическую операцию). Состояние битовой команды, которая имеет доступ к памяти на запись (S, R, =), равно значению бита, в который команда производит запись, или, если запись не производится, то оно равно значению бита, к которому команда обращается. Бит состояния не имеет значения для битовых команд, не обращающихся к памяти. Такие команды устанавливают бит состояния в 1 (STA=1). Бит состояния не опрашивается командой. Он интерпретируется только при тестировании программы (статус программы).

Бит OR

Бит OR необходим, если вы используете команду O при выполнении логического И перед операцией ИЛИ. Функция И может содержать следующие команды: A, AN A(, AN(,) и NOT. Бит OR показывает этим командам, что ранее выполненная функция И дала значение 1, предвосхищая тем самым результат логической операции ИЛИ. Любая другая команда, обрабатывающая биты, сбрасывает бит OR (см. раздел 5.4).

Бит переполнения

Бит переполнения (бит OV) указывает на ошибку. Он устанавливается арифметической командой или командой сравнения чисел с плавающей точкой после возникновения ошибки (переполнение, недопустимая операция, недопустимое число с плавающей точкой). Этот бит устанавливается в соответствии с результатом следующей арифметической команды или команды сравнения.

Бит сохраняемого переполнения

Бит сохраняемого переполнения (бит OS) устанавливается вместе сбитом OV, когда происходит ошибка. Так как бит OS остается установленным после устранения ошибки, то он сохраняет состояние бита OV и указывает, появлялась ли ошибка в одной из ранее выполненных команд. Следующие команды сбрасывают бит OS: JOS (переход после сохраняемого переполнения), команды вызова блока и команды конца блока.

Код условия 1 и код условия 0

Биты CC 1 и CC 0 (коды условий) предоставляют информацию о следующих результатах или битах:

- Результат арифметической операции
- Результат операции сравнения
- Результат цифровой операции
- Биты, выдвинутые командой сдвига или циклического сдвига

Таблицы с 2–13 по 2–18 перечисляют значения CC 1 и CC 0 после выполнения вашей программой определенных команд.

Таблица 2–13. CC 1 и CC 0 после арифметических операций без переполнения

CC 1	CC 0	Объяснение
0	0	Результат = 0
0	1	Результат < 0
1	0	Результат > 0

Таблица 2–14. CC 1 и CC 0 после операций с целыми числами, с переполнением

CC 1	CC 0	Объяснение
0	0	Переполнение отрицательного диапазона при +I и +D
0	1	Переполнение отрицательного диапазона при *I и *D Переполнение положительного диапазона при +I, -I, +D, -D, NEGI и NEGD
1	0	Переполнение положительного диапазона при *I, *D, /I и /D Переполнение отрицательного диапазона при +I, -I, +D и -D
1	1	Деление на 0 в /I, /D и MOD

Таблица 2–15. CC 1 и CC 0 после арифметических операций с плавающей точкой, с переполнением		
CC 1	**CC 0**	**Объяснение**
0	0	Ступенчатая потеря значимости
0	1	Переполнение отрицательного диапазона
1	0	Переполнение положительного диапазона
1	1	Недопустимое число с плавающей точкой

Таблица 2–16. CC 1 и CC 0 после операций сравнения		
CC 1	**CC 0**	**Объяснение**
0	0	Аккумулятор 2 = аккумулятору 1
0	1	Аккумулятор 2 < аккумулятора 1
1	0	Аккумулятор 2 > аккумулятора 1
1	1	Аккумулятор 1 или аккумулятор 2 – недопустимое число с плавающей точкой

Таблица 2–17. CC 1 и CC 0 после операций сдвига и циклического сдвига		
CC 1	**CC 0**	**Объяснение**
0	0	Последний выдвинутый бит = 0
1	0	Последний выдвинутый бит = 1

Таблица 2–18. CC 1 и CC 0 после цифровых логических операций		
CC 1	**CC 0**	**Объяснение**
0	0	Результат = 0
1	0	Результат <> 0

Бит двоичного результата

Бит двоичного результата (бит BR) образует связь между обработкой битов и слов. Это эффективное средство интерпретации результата операции со словами как двоичного результата и встраивания этого результата в двоичную логическую цепь. С этой точки зрения бит BR представляет собой внутримашинный бит памяти, в котором сохраняется RLO перед выполнением операции со словами, которая изменяет RLO, так что RLO после этой операции снова доступен для продолжения прерванной битовой цепи.

Например, бит BR дает возможность записать функциональный блок (FB) или функцию (FC) в виде списка операторов (AWL), а затем вызвать FB или FC из контактного плана (КОР, см. *Справочное руководство* ***I*233*I***).

При записи функционального блока или функции, которые вы хотите вызывать из КОР, независимо о того, пишете вы FB или FC на AWL или КОР, вы отвечаете за управление битом BR. Бит BR соответствует разрешающему выходу (ENO) блока КОР. Вы должны использовать команду SAVE (в AWL, см. раздел 5.9) или катушку ---(SAVE) (в КОР) для сохранения RLO в бите BR в соответствии со следующими критериями:

- Сохраняйте RLO в бите BR со значением 1, если FB или FC выполняется без ошибок.
- Сохраняйте RLO в бите BR со значением 0, если FB или FC выполняется с ошибкой

Вы должны запрограммировать эти команды в конце FB или FC, чтобы они были последними командами, выполняемыми в блоке.

Когда вы вызываете в своей программе системный функциональный блок (SFB) или системную функцию (SFC), SFB или SFC показывает, смог ли CPU выполнить функцию без ошибок или нет, предоставляя в бите двоичного результата следующую информацию:

- Если при выполнении произошла ошибка, то бит BR равен 0.
- Если функция была исполнена без ошибок, то бит BR равен 1.

3 Адресация

Обзор главы

3.1 Непосредственная адресация

Описание

При непосредственной адресации операнд кодируется непосредственно в команде; то есть за командой непосредственно следует значение, с которым должна работать данная команда (например, загрузка). Команда также может предоставить свое собственное значение (например, SET, см. табл. 3–1).

Примеры

Таблица 3–1. Непосредственная адресация

SET	Установить RLO в 1.
OW W#16#A320	Поразрядное ИЛИ со словом.
L 27	Загрузить целое число 27 в аккумулятор 1.
L 'ABCD'	Загрузить символы ASCII ABCD в аккумулятор 1.
L B#(100,12)	Загрузить два байта 100 и 12 в аккумулятор 1.
L C#100	Загрузить значение 100 в формате BCD в аккумулятор 1.

3.2 Прямая адресация

Описание

Команда, использующая прямую адресацию, имеет операнд, указывающий местоположение значения, которое команда будет обрабатывать, и состоящий из следующих двух частей:

- идентификатора операнда (например, «IB» для «входного байта»)
- точного адреса внутри области памяти, указанной идентификатором операнда

Операнд прямо указывает на адрес значения.

Примеры

Таблица 3–2. Прямая адресация	
A I 0.0	Выполнить логическую операцию И с входным битом I 0.0.
S L 20.0	Установить бит локальных данных L 20.0.
= M 115.4	Присвоить RLO биту памяти M 115.4
L IB0	Загрузить входной байт IB0 в аккумулятор 1.
L MW64	Загрузить слово памяти MW64 в аккумулятор 1.
T DBD12	Передать содержимое из аккумулятора 1 в двойное слово данных DBD12.

3.3 Косвенная адресация через память

Описание

Команда, использующая косвенную адресацию через память, имеет операнд, указывающий местоположение значения, которое команда будет обрабатывать, и состоящий из следующих двух частей:

- идентификатора операнда (например, «IB» для «входного байта»)
- одного из следующих указателей:
  - слова, содержащего номер таймера (T), счетчика (C), блока данных (DB), функции (FC) или функционального блока (FB)
  - двойного слова, содержащего точный адрес значения внутри области памяти, указанной идентификатором операнда

Операнд задает адрес значения или номер косвенно через указатель. Это слово или двойное слово может находиться в одной из следующих областей:

- битовая память (M)
- блок данных (DB)
- экземплярный блок данных (DI)
- локальные данные (L)

Преимуществом косвенной адресации через память является то, что вы можете динамически модифицировать операнды команды во время обработки программы.

Использование правильного синтаксиса

Когда вы работаете с косвенно адресованным через память операндом, который хранится в области памяти блока данных, вам нужно вначале открыть блок данных, используя команду *Открыть блок данных* (OPN). После этого вы можете использовать слово или двойное слово данных в качестве косвенного адреса, как показано в следующем примере:

```
OPN  DB10
L  IB [DBD20]
```

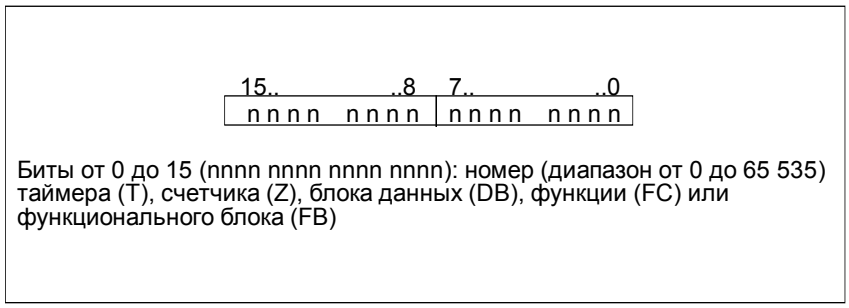
Примеры

Таблица 3–3. Косвенная адресация через память

Пример	Описание
A I [MD2] или A I [anna]	Выполнить логическую операцию И с входным битом, точный адрес которого находится в двойном слове памяти MD2 или по обозначенному через «anna» адресу в таблице символов в качестве ссылки на MD2.
= DIX [DBD2]	Присвоить бит RLO биту данных экземплярного блока данных, точный адрес которого находится в двойном слове данных DBD2.
OPN DB [LW2]	Открыть блок данных, номер которого находится в слове локальных данных LW2.
O Q [LD3] или O Q [boxcar]	Выполнить логическую операцию ИЛИ с выходным битом, который находится по адресу, указанному в двойном слове локальных данных LD3 или в локальной переменной типа TEMP, обозначенной как «boxcar».

Формат указателя

Есть два возможных формата указателя: слово и двойное слово. Сокращенное обозначение для формата в виде слова заканчивается на W (например, DBW). Рис. 3–1 показывает формат указателя для слова. Сокращенное обозначение для формата в виде двойного слова заканчивается на D (например, DBD). Рис. 3–2 показывает формат указателя для двойного слова.

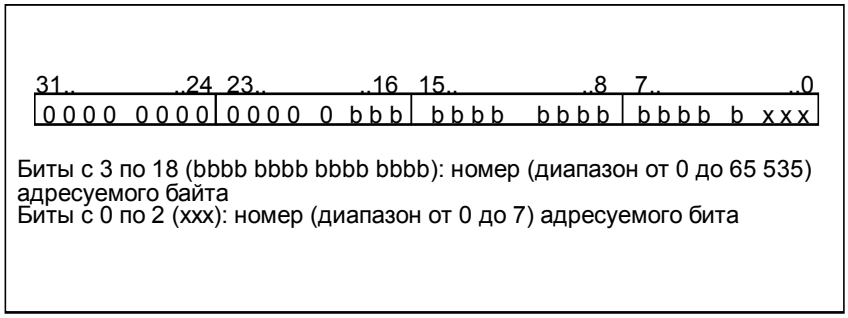


Биты от 0 до 15 (nnnn nnnn nnnn nnnn): номер (диапазон от 0 до 65 535) таймера (T), счетчика (Z), блока данных (DB), функции (FC) или функционального блока (FB)

Рис. 3–1. Формат указателя в виде слова для косвенной адресации через память

AWL		Объяснение
L	+5	Загрузить значение 5 как целое число в аккумулятор 1.
T	MW2	Передать содержимое аккумулятор 1 в слово памяти MW2.
OPN	DB[MW2]	Открыть блок данных 5.

STL		Объяснение
OPN	DB10	Открыть блок данных DB10.
L	+20	Загрузить значение 20 как целое число в аккумулятор 1.
T	DBW10	Передать содержимое аккумулятор 1 в слово данных DBW10.
A	T[DBW10]	Опросить состояние сигнала таймера T 20.



Биты с 3 по 18 (bbbb bbbb bbbb bbbb): номер (диапазон от 0 до 65 535) адресуемого байта
Биты с 0 по 2 (xxx): номер (диапазон от 0 до 7) адресуемого бита

Рис. 3–2. Формат указателя в виде двойного слова для косвенной адресации через память

Замечание

Если вы обращаетесь к байту, слову или двойному слову, то убедитесь, что номер бита указателя равен 0.

Два следующих примера показывают, как работать с форматом указателя в виде двойного слова:

AWL	Объяснение
L P#8.7	Загрузить 2#0000 0000 0000 0000 0000 0000 0100 0111 (двоичное значение) в аккумулятор 1.
T MD2	Запомнить адрес 8.7 в двойном слове памяти MD2.
A I [MD2] = Q [MD2]	Контроллер опрашивает входной бит I 8.7 и присваивает состояние его сигнала выходному биту Q 8.7.

AWL	Объяснение
L P#8.0	Загрузить 2#0000 0000 0000 0000 0000 0000 0100 0000 (двоичное значение) в аккумулятор 1.
T MD2	Запомнить адрес 8 в двойном слове памяти MD2.
L IB [MD2] T MW [MD2]	Контроллер загружает входной байт IB8 и передает содержимое в слово памяти MW8. Точный адрес 8 получается из двойного слова памяти MD2.

3.4 Адресные регистры

Объяснение

Для некоторых видов косвенной адресации при программировании в AWL необходимы определенные регистры CPU. Эти регистры описаны ниже.

Адресные регистры 1 и 2

Адресные регистры 1 и 2 (AR1 и AR2) – это 32-битные регистры, которые принимают указатель на адрес внутри заданной области или указатель, содержащий информацию об области, для команд, использующих косвенную адресацию через регистр (см. разделы 3.5 и 3.6).

Указатели

Указатели используются при косвенной адресации через регистр (см. разделы 3.5 и 3.6). В распоряжении имеются два следующих вида указателей:

- внутренние для области: для доступа внутри заданной области к битам, байтам, словам и двойным словам в областях памяти P, I, Q, M, DBX, DIX и L
- с указанием области: для доступа с указанием области к битам, байтам, словам и двойным словам в областях памяти P, I, Q, M, DBX, DIX и L

3.5 Косвенная адресация внутри области через регистр

Описание

Команда, использующая косвенную адресацию внутри области через регистр, имеет операнд, указывающий местоположение значения, которое команда будет обрабатывать, и состоящий из следующих двух частей:

- идентификатора операнда (например, «LD» для «двойного слова локальных данных», см. табл. 2–6)
- адресного регистра и указателя для задания байта и бита. Байт и бит указывают смещение, которое при добавлении к содержимому регистра определяет адрес в памяти значения, которое команда должна обработать.

Операнд указывает на адрес значения косвенно, через адресный регистр плюс смещение.

Оператор, использующий косвенную адресацию внутри области через регистр, не изменяет значения в адресном регистре.

Вычисление адреса операнда

Операнд команды указывает на значение, которое команда будет обрабатывать. При косвенной адресации внутри области через регистр операнд указывает на адрес значения косвенно через адресный регистр плюс смещение. Рис. 3–3 показывает, как вычислить адрес операнда для команды присваивания (=) в следующем операторе:

```
= Q [AR1, P#1.1]
```

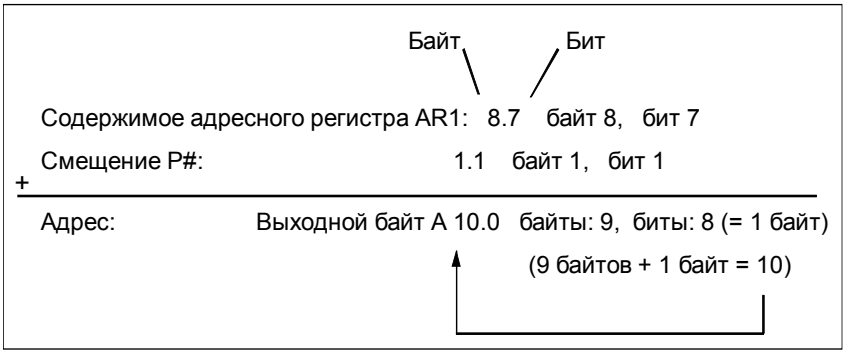


Рис. 3–3. Вычисление адреса выхода Q [AR1, P#1.1]

Вы вычисляете адрес операнда, прибавляя байтовый компонент содержимого адресного регистра к байтовому компоненту указателя смещения и прибавляя битовый компонент содержимого адресного регистра к битовому компоненту указателя смещения. При вычислении байтового компонента адреса используйте десятичную систему счисления, при вычислении битового компонента адреса используйте восьмеричную систему счисления (8 бит = 1 байт). Здесь речь может идти о переносе между битовым и байтовым компонентами.

Примеры

Таблица 3–4. Косвенная адресация внутри области через регистр	
A I [AR1, P#4.3]	Выполнить логическую операцию И с входным битом, адрес которого рассчитывается как содержимое адресного регистра AR 1 плюс 4 байта плюс 3 бита.
= DIX [AR2, P#0.0]	Присвоить бит RLO биту данных экземплярного блока данных, адрес которого находится в адресном регистре AR2.
L IB [AR1, P#100.0]	Загрузить входной байт, адрес которого рассчитывается как содержимое адресного регистра AR 1 плюс 100 байтов, в аккумулятор 1.
T LD [AR2, P#56.0]	Передать содержимое аккумулятора 1 в двойное слово локальных данных LD, адрес которого рассчитывается как содержимое адресного регистра AR 2 плюс 56 байтов. По поводу адресации локальных данных прочитайте, пожалуйста, приведенное ниже предупреждение.

Предупреждение

Возможно переписывание данных, используемых компилятором.

Если вы используете абсолютную адресацию для обращения к временным локальным данным, то нет гарантии, что не возникнет конфликт между локальными данными, используемыми компилятором, и локальными данными, к которым вы пытаетесь обратиться посредством абсолютной адресации. Возможно, что вы перепишете некоторые из данных, используемых компилятором. (Например, компилятор использует локальные данные для передачи формальных параметров). Локальные данные, используемые компилятором, связаны с символическими данными, которые определяются программистом.

При обращении к временным локальным данным рекомендуется выбирать не абсолютную, а символическую адресацию.

Формат указателя

Косвенная адресация внутри области через регистр имеет только один возможный формат указателя: двойное слово. Это двойное слово содержит операнд, закодированный как адрес бита. Сокращенное обозначение для формата в виде двойного слова оканчивается на D (например, DBD). Рис. 3–4 показывает формат указателя для двойного слова.

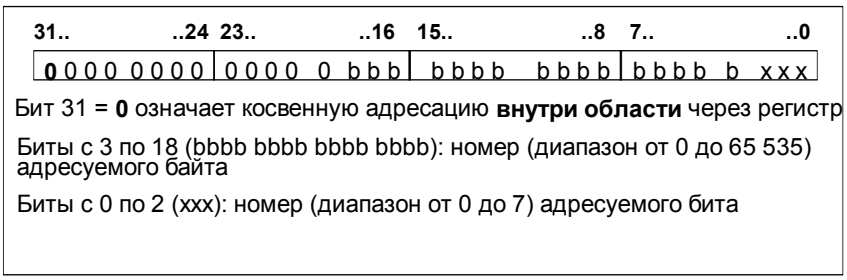


Рис. 3–4. Формат указателя в виде двойного слова для косвенной адресации внутри области через регистр

Замечание

При обращении к байту, слову или двойному слову убедитесь, что номер бита вашего указателя равен 0.

Два следующих примера показывают, как работать с форматом указателя в виде двойного слова:

AWL	Объяснение
L P#8.7	**Загрузить указатель в формате двойного слова на адрес бита 8.7 в аккумулятор 1.**
LAR1	**Сохранить указатель в формате двойного слова на адрес бита 8.7 в адресном регистре AR 1.**
A I [AR1, P#0.0]	**CPU добавляет к содержимому адресного регистра AR 1 (8.7) смещение (P#0.0) и использует этот адрес как операнд битовой логической операции И. Содержимое AR1 остается неизменным.**
= Q [AR1, P#1.1]	**CPU присваивает результат логической операции (RLO) адресу (Q 10.0). CPU вычисляет этот операнд, складывая содержимое адресного регистра AR1 (8.7) и смещение (P#1.1).**

AWL	Объяснение
L P#8.0	**Загрузить указатель в формате двойного слова на адрес бита 8.0 в аккумулятор 1.**
LAR2	**Сохранить указатель в формате двойного слова на адрес бита 8.0 в адресном регистре AR 2.**
L IB [AR2, P#2.0]	**CPU загружает входной байт IB10 в аккумулятор 1.**
T MW [AR2, P#200.0]	**CPU передает содержимое аккумулятора 1 в слово памяти MW208.** **Адрес 208 рассчитывается как 8 (AR 2) плюс 200 (смещение), что равно 208.**

3.6 Косвенная адресация с указанием области через регистр

Описание

Команда, использующая косвенную адресацию с указанием области через регистр, имеет операнд, указывающий местоположение значения, которое команда будет обрабатывать, и состоящий из следующих двух частей:

- идентификатора операнда, указывающего размер объекта данных (например, «B» для «байта», см. табл. 2–8). Область памяти указывается в битах 24, 25 и 26 адресного регистра.
- адресного регистра и указателя, указывающего смещение, которое, будучи добавлено к содержимому адресного регистра, определяет адрес значения, подлежащего обработке командой. Указатель задается в виде P#байт.бит.

Операнд указывает на адрес значения косвенно, а именно, через адресный регистр плюс смещение.

Оператор, использующий косвенную адресацию с указанием области через регистр, не изменяет значения в адресном регистре.

Вычисление адреса операнда

Операнд команды указывает на значение, которое команда будет обрабатывать. При косвенной адресации с указанием области памяти через регистр операнд указывает на адрес значения косвенно, через адресный регистр плюс смещение. Рис. 3–5 показывает, как вычислить адрес операнда для команды присваивания (=) в следующем операторе:

```
= [AR1, P#1.1]
```

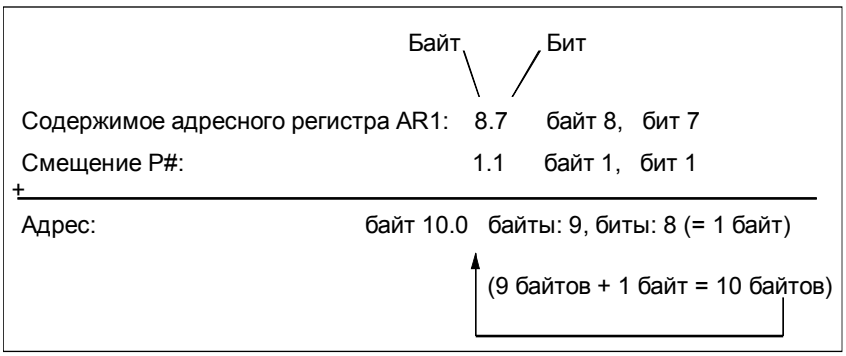


Рис. 3–5. Вычисление адреса [AR1, P#1.1]

Вы вычисляете адрес операнда, прибавляя байтовый компонент содержимого адресного регистра к байтовому компоненту указателя смещения и прибавляя битовый компонент содержимого адресного регистра к битовому компоненту указателя смещения. При вычислении байтового компонента адреса используйте десятичную систему счисления, при вычислении битового компонента адреса используйте восьмеричную систему счисления (8 битов = 1 байт). Здесь речь может идти о переносе между битовым и байтовым компонентами.

Пример

Таблица 3–5 дает примеры косвенной адресации с указанием области памяти через регистр. Операнд должен содержать дополнительный идентификатор области в битах 24, 25 и 26 указателя. Адресуемая информация находится в адресном регистре.

Таблица 3–5. Косвенная адресация с указанием области через регистр

Пример	Описание
A [AR1, P#4.3]	Выполнить логическую операцию И с битом, адрес которого вычисляется как содержимое адресного регистра AR 1 плюс 4 байта плюс 3 бита. Область памяти бита задана в битах 24, 25 и 26 адресного регистра AR1.
= [AR2, P#0.0]	Присвоить бит RLO биту, адрес которого находится в адресном регистре AR 2. Область памяти бита задана в битах 24, 25 и 26 адресного регистра AR2.
L B [AR1, P#100.0]	Загрузить в аккумулятор 1 байт, адрес которого вычисляется как содержимое адресного регистра AR 1 плюс 100 байтов. Область памяти байта задана в битах 24, 25 и 26 адресного регистра AR1.
T D [AR2, P#56.0]	Передать содержимое аккумулятора 1 в двойное слово, адрес которого вычисляется как содержимое адресного регистра AR 2 плюс 56 байтов. Область памяти двойного слова задана в битах 24, 25 и 26 адресного регистра AR2.

Таблица 3–6 перечисляет значения двоичного кода в битах 24, 25 и 26 указателя, которые определяют эту область.

Таблица 3–6. Идентификатор области при косвенной адресации с указанием области через регистр

Идентификатор области (область памяти)	Двоичное содержимое битов 26, 25 и 24
P (периферия, внешние входы и выходы)	000
I (вход образа процесса)	001
Q (выход образа процесса)	010
M (битовая память)	011
DBX (блок данных)	100
DIX (экземплярный блок данных)	101
(предыдущие локальные данные, т.е. локальные данные предыдущего незавершенного блока)	111

Формат указателя

Для косвенной адресации с указанием области памяти через регистр в распоряжении имеется только один возможный формат указателя: двойное слово. Сокращенное обозначение указателя в формате двойного слова заканчивается на D (например, DBD). На рисунке 3–6 вы видите формат указателя для двойного слова.

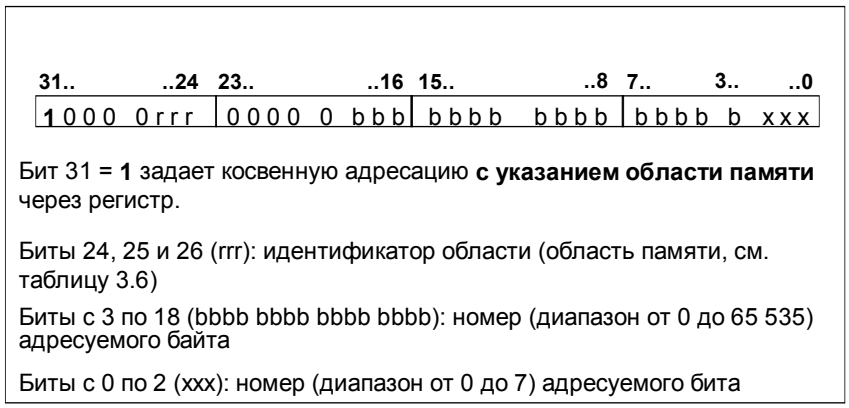


Рис. 3–6. Формат указателя в виде двойного слова для косвенной адресации с указанием области памяти через регистр

Указание

Если вы обращаетесь к байту, слову или двойному слову, то убедитесь, что номер бита вашего указателя равен 0.

К локальным данным нельзя обращаться посредством косвенной адресации с указанием области памяти через регистр!

Два следующих примера показывают, как работать с указателем в формате двойного слова:

AWL	Объяснение
L P#I 8.7	**Загрузить указатель в формате двойного слова на адрес бита I 8.7 в аккумулятор 1.**
LAR1	**Сохранить указатель в формате двойного слова на адрес бита I 8.7 в AR1.**
L P#Q 8.7	**Загрузить указатель в формате двойного слова на адрес бита Q 8.7 в аккумулятор 1.**
LAR2	**Сохранить указатель в формате двойного слова на адрес бита Q 8.7 в AR2.**
A [AR1, P#0.0]	**CPU складывает содержимое адресного регистра AR1 (P# I 8.7) и смещение (P#0.0) и использует операнд, на который указывает результат (I8.7), в качестве операнда битовой логической операции И. Содержимое AR1 остается неизменным.**
= [AR2, P#1.1]	**CPU присваивает результат логической операции (RLO) операнду (Q 10.0). CPU вычисляет этот операнд, складывая содержимое адресного регистра AR2 (P#A 8.7) и смещение (P#1.1), указатель при этом деактивизируется. Содержимое AR2 остается неизменным.**

AWL	Объяснение
L P#I 8.0	**Загрузить указатель в формате двойного слова на адрес бита I 8.0 в аккумулятор 1.**
LAR2	**Сохранить указатель в формате двойного слова на адрес бита I 8.0 в адресном регистре AR 2.**
L P#M 8.0	**Загрузить указатель в формате двойного слова на адрес бита M 8.0 в аккумулятор 1.**
LAR1	**Сохранить указатель в формате двойного слова на адрес бита M 8.0 в адресном регистре AR 1.**
L B [AR2, P#2.0]	**CPU загружает входной байт IB10 в аккумулятор 1.**
T W [AR1, P#200.0]	**CPU передает содержимое аккумулятора 1 в слово памяти MW208.**
	Входной байт 10 вычисляется как 8 (AR 2) плюс 2 (смещение). Слово памяти 208 вычисляется как 8 (AR 1) плюс 200 (смещение), что дает 208.

4 Операции с аккумуляторами и команды, использующие адресные регистры

Обзор главы

4.1 Обзор

В вашем распоряжении имеются следующие команды для манипулирования содержимым одного или обоих аккумуляторов:

Мнемоника	**Команда**	**Объяснение**
TAK	Обменять содержимое аккумулятора 1 с содержимым аккумулятора 2	Эта команда обменивает содержимое аккумулятора 1 с содержимым аккумулятора 2.
PUSH с двумя аккумуляторами	Копировать аккумулятор 1 в аккумулятор 2	Эта команда копирует содержимое аккумулятора 1 в аккумулятор 2.
POP с двумя аккумуляторами	Копировать аккумулятор 2 в аккумулятор 1	Эта команда копирует содержимое аккумулятора 2 в аккумулятор 1.
PUSH с четырьмя аккумуляторами	Копировать аккумулятор 3 в аккумулятор 4, аккумулятор 2 в аккумулятор 3, аккумулятор 1 в аккумулятор 2	Эта команда копирует содержимое аккумулятора 3 в аккумулятор 4, содержимое аккумулятора 2 в аккумулятор 3 и содержимое аккумулятора 1 в аккумулятор 2.
POP с четырьмя аккумуляторами	Копировать аккумулятор 2 в аккумулятор 1, аккумулятор 3 в аккумулятор 2, аккумулятор 4 в аккумулятор 3	Эта команда копирует содержимое аккумулятора 2 в аккумулятор 1, содержимое аккумулятора 3 в аккумулятор 2 и содержимое аккумулятора 4 в аккумулятор 3.
ENT	Войти в стек аккумуляторов	Эта команда копирует содержимое аккумулятора 3 в аккумулятор 4 и содержимое аккумулятора 2 в аккумулятор 3.
LEAVE	Покинуть стек аккумуляторов	Эта команда копирует содержимое аккумулятора 3 в аккумулятор 2 и содержимое аккумулятора 4 в аккумулятор 3.
INC	Инкрементировать аккумулятор 1	Эта команда увеличивает содержимое младшего байта в младшем слове аккумулятора 1 на 8-битовую константу, заданную в операторе команды. Эта константа может находиться в диапазоне от 0 до 255.
DEC	Декрементировать аккумулятор 1	Эта команда уменьшает содержимое младшего байта в младшем слове аккумулятора 1 на 8-битовую константу, заданную в операторе команды. Эта константа может находиться в диапазоне от 0 до 255.
+AR1, +AR2	Прибавить аккумулятор 1 к адресному регистру	Эта команда прибавляет содержимое младшего слова аккумулятора 1 к адресному регистру 1 или 2.
+AR1 P# байт.бит, +AR2 P# байт.бит	Прибавить константу к адресному регистру	Эта команда прибавляет константу к содержимому адресного регистра 1 или 2.

Мнемоника	**Команда**	**Объяснение**
BLD	Команда отображения программы	Эта команда не выполняет никакой функции и не влияет на биты состояния. Она имеет значение только для устройства программирования (PG) при отображении программы. Операнд <число> является идентификатором команды BLD и генерируется устройством программирования.
NOP 0 NOP 1	Пустая команда 0 Пустая команда 1	Эти команды не выполняют никаких действий и не оказывают влияния на содержимое слова состояния. Команды NOP 1 и NOP 0 необходимы для декомпиляции. Код команды содержит битовую комбинацию из 16 нулей или 16 единиц.

За информацией об изменении последовательности байтов в аккумуляторе 1 обратитесь к разделу 12.3.

4.2 ENT и LEAVE

Описание

С помощью команд ENT (Enter Accumulator Stack [Войти в стек аккумуляторов]) и LEAVE (Leave Accumulator Stack [Покинуть стек аккумуляторов]) вы можете выполнять следующие функции:

- Команда ENT копирует содержимое аккумулятора 3 в аккумулятор 4 и содержимое аккумулятора 2 в аккумулятор 3. Если вы программируете команду ENT непосредственно перед командой загрузки, то она СДВИГАЕТ содержимое аккумулятора 2 и аккумулятора 3 глубже в стек.
- Команда LEAVE копирует содержимое аккумулятора 3 в аккумулятор 2 и содержимое аккумулятора 4 в аккумулятор 3. Если вы программируете команду LEAVE непосредственно перед командой сдвига или циклического сдвига, связывающей аккумуляторы, то команда LEAVE будет действовать как арифметическая операция.

ENT

Рис. 4–1 показывает, как работает команда ENT.

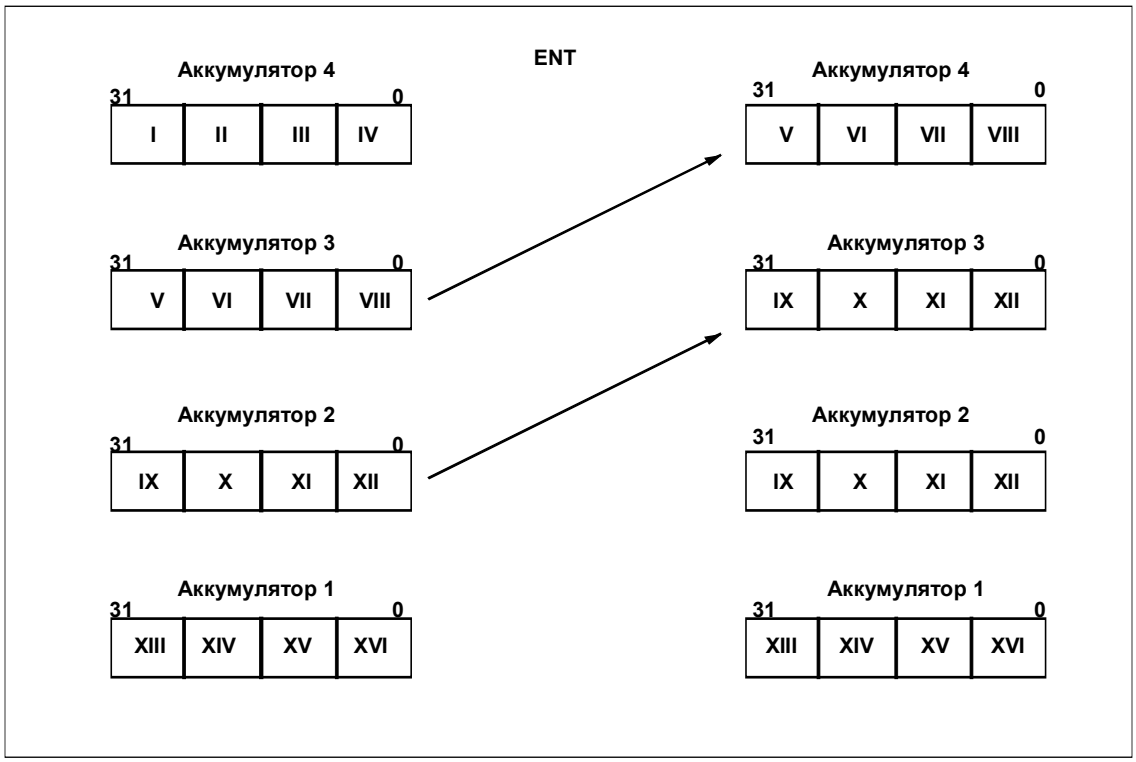


Рис. 4–1. Копирование содержимого аккумулятора 3 в аккумулятор 4 и содержимого аккумулятора 2 в аккумулятор 3 в команде ENT

LEAVE

Рис. 4–2 показывает, как работает команда LEAVE.

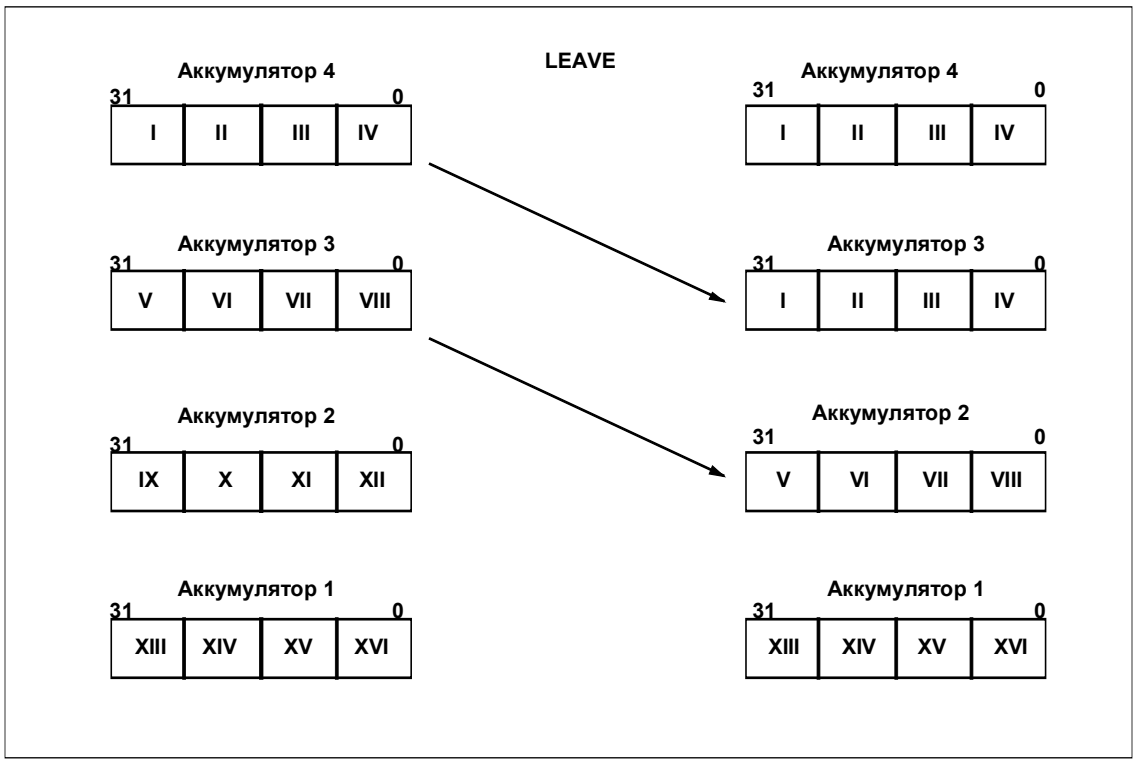


Рис. 4–2. Копирование содержимого аккумулятора 3 в аккумулятор 2 и содержимого аккумулятора 4 в аккумулятор 3 в команде LEAVE

Пример

Следующий фрагмент программы показывает использование команды ENT.

Нужно сложить числа с плавающей точкой, расположенные в двойных словах данных DBD0 и DBD4. Сумму нужно разделить на разность чисел с плавающей точкой, расположенных в двойных словах данных DBD8 и DBD12.

$$DBD16 = \frac{DBD0 + DBD4}{DBD8 - DBD12}$$

Частное от деления должно быть сохранено в DBD16.

В этом примере целью команды ENT является прием промежуточного результата (DBD0+DBD4), который находится в аккумуляторе 2, и сохранение его в аккумуляторе 3. Команда вычитания (–R) копирует этот промежуточный результат обратно в аккумулятор 2 после вычитания.

AWL		Объяснение
L	**DBD0**	**Загрузить значение из двойного слова данных DBD0 в аккумулятор 1 (это значение должно иметь формат числа с плавающей точкой).**
L	**DBD4**	**Копировать это значение из аккумулятора 1 в аккумулятор 2. Загрузить значение из двойного слова данных DBD4 в аккумулятор 1**
+R		**(это значение должно иметь формат числа с плавающей точкой). Сложить содержимое аккумулятора 1 и аккумулятора 2 как числа с плавающей точкой (32 бита, IEEE–FP) и сохранить результат в**
L	**DBD8**	**аккумуляторе 1. Копировать значение из аккумулятора 1 в аккумулятор 2.**
ENT		**Загрузить значение из двойного слова данных DBD8 в аккумулятор 1. Копировать содержимое аккумулятора 3 в аккумулятор 4.**
L	**DBD12**	**Копировать содержимое аккумулятора 2 (промежуточный результат) в аккумулятор 3.**
-R		**Копировать содержимое аккумулятора 1 в аккумулятор 2 Загрузить значение из двойного слова данных DBD12 в аккумулятор 1. Вычесть содержимое аккумулятора 1 из аккумулятора 2. Сохранить**
/R		**результат в аккумуляторе 1. Копировать содержимое аккумулятора 3 в аккумулятор 2.**
T	**DBD16**	**Разделить содержимое аккумулятора 2 на содержимое аккумулятора 1. Сохранить результат в аккумуляторе 1. Передать результат (аккумулятор 1) в двойное слово данных DBD16.**

4.3 Инкрементирование и декрементирование

Описание

Вы можете использовать команды *Инкрементировать аккумулятор 1* (INC) и *Декрементировать аккумулятор 1* (DEC) для выполнения следующих функций:

- INC увеличивает содержимое младшего байта в младшем слове аккумулятора 1 на 8-битовую константу, заданную в операторе команды. Эта константа может лежать в диапазоне от 0 до 255.
- DEC уменьшает содержимое младшего байта в младшем слове аккумулятор 1 на 8-битовую константу, заданную в команде. Эта константа может лежать в диапазоне от 0 до 255.

CPU всегда выполняет команды INC и DEC независимо от результата логической операции. Эти команды не влияют на RLO и не изменяют биты в слове состояния.

Замечание

Эти команды не годятся для 16-битовых и 32-битовых арифметических операций, так как не происходит перенос из младшего байта младшего слова аккумулятора 1 в старший байт младшего слова аккумулятора 1. Для 16-битовых и 32-битовых арифметических операций используйте соответственно команды +I и +D.

Пример

Следующий пример программирования показывает, как работает команда INC внутри программного цикла, который был запущен посредством условного перехода.

AWL			Объяснение
			Тело цикла.
	L	**1**	**Установить счетчик цикла в 1.**
	T	**MB10**	
M1:	**L**	**MB10**	**Загрузить содержимое байта памяти MB10 в аккумулятор 1.**
	INC	**1**	**Увеличить счетчик цикла на 1.**
	T	**MB10**	**Передать содержимое аккумулятора 1 в байт памяти MB10.**
	.		**Раздел команд, который обрабатывается пять раз.**
	.		
	L	**B#16#5**	
	<=	**I**	
	SPB	**M1**	**Если программа не выполнила программный цикл пятикратно, то вернуться к операции цикла.**

4.4 +AR1 и +AR2: прибавление константы к адресному регистру 1 или адресному регистру 2

Описание

Используя команды +AR1 и +AR2, вы можете прибавить константу к содержимому адресных регистров 1 и 2:

Таблица 4–1. Прибавление значения к содержимому адресных регистров

Команда	Операнд	Функция
+AR1	-	Прибавляет содержимое младшего слова аккумулятора 1 к содержимому адресного регистра 1.
+AR2	-	Прибавляет содержимое младшего слова аккумулятора 1 к содержимому адресного регистра 2.
+AR1	P#байт.бит: (диапазон от 0.0 до 4095.7)[1]	Прибавляет константу указателя к содержимому адресного регистра 1.
+AR2	P#байт.бит: (диапазон от 0.0 до 4095.7)[1]	Прибавляет константу указателя к содержимому адресного регистра 2.

[1] Биты 24, 25 и 26 адресного регистра не изменяются. Эти биты указывают область памяти.

Замечание

Адресный регистр 2 используется при обработке мультиэкземпляров. Поэтому перед программированием команды «+AR2» вы должны «сохранить» содержимое AR2 и загрузить его затем снова.

Примеры

Ниже приведены примеры операторов, использующих команды +AR1 и +AR2. Загрузка значения в формате указателя в аккумулятор 1, а затем использование команд +AR1 и +AR2, как показано в первых двух операторах в следующем примере, дает вам возможность выбора из диапазона от 0.0 до 8191.7.

```
AWL                     Объяснение
L  P#250.7              Загрузить константу указателя (250.7) в аккумулятор 1.
+AR1                    Прибавить содержимое аккумулятора (250.7) к содержимому адресного
                        регистра 1.
TAR2 #SAVE_AR2          Из-за мультиэкземпляров, которые используют AR2 как базу

+AR2 P#126.7            Прибавить константу указателя (126.7) к содержимому адресного регистра
.                       2.
.
L AR2 #SAVE_AR2         Восстановить AR2.
```

5 Битовые логические операции

Обзор главы

5.1 Булева битовая логика

Объяснение

Булева битовая логика используется в следующих основных командах:

- И (A) и ее инверсная форма И-НЕ (AN)
- ИЛИ (O) и ее инверсная форма ИЛИ-НЕ (ON)
- Исключающее ИЛИ (X) и ее инверсная форма , Исключающее ИЛИ-НЕ (XN)

Эти команды выполняют следующие основные функции:

- Они опрашивают состояние сигнала операнда, чтобы выяснить активизирован операнд «1» или нет «0».
- Они опрашивают состояние сигнала таймера или счетчика, чтобы выяснить, установлен ли он в «0» (значение = 0) или в «1» (значение > 0).

Бит $\overline{FC}$ определяет результат логической операции (RLO):

- Если $\overline{FC}$ равен 0, то результат опроса состояния остается неизменным и будет сохранен в RLO (начало логической цепи).
- Если $\overline{FC}$ равен 1, то результат опроса состояния будет логически скомбинирован с логической командой (A, O, X) в соответствии с таблицей истинности и будет сохранен в RLO.

Таблица истинности в начале булевой логической цепи

Результат логической операции может быть определен с помощью следующей таблицы истинности:

Мнемоника	Команда	Состояние операнда	Результат в RLO
A	И	0 1	0 1
AN	И-НЕ	0 1	1 0
O	ИЛИ	0 1	0 1
ON	ИЛИ-НЕ	0 1	1 0
X	Исключающее ИЛИ	0 1	0 1
XN	Исключающее ИЛИ-НЕ	0 1	1 0

Таблица истинности внутри булевой логической цепи

После второй булевой битовой операции RLO может быть определен с помощью следующей таблицы:

Мнемоника	Команда	RLO перед командой	Состояние операнда	Результат в RLO
A	И	0 0 1 1	0 1 0 1	0 0 0 1
AN	И-НЕ	0 0 1 1	0 1 0 1	0 0 1 0
O	ИЛИ	0 0 1 1	0 1 0 1	0 1 1 1
ON	ИЛИ-НЕ	0 0 1 1	0 1 0 1	1 0 1 1
X	Исключающее ИЛИ	0 0 1 1	0 1 0 1	0 1 1 0
XN	Исключающее ИЛИ-НЕ	0 0 1 1	0 1 0 1	1 0 0 1

Операнды основных функций (A, AN, O, ON, X, XN)

Операнд команды может быть битом, таймером или счетчиком. Команда обращается к операнду с помощью одного из следующих видов адресации:

- Идентификатор операнда и адрес внутри области памяти, определяемой идентификатором операнда (см. таблицы 5–1 и 5–3).
- Бит, таймер или счетчик, передаваемый в качестве параметра (см. табл. 5–4).
- Условия, выраженные через биты слова состояния (см. табл. 5–8).

Таблица 5–1. Операнды: прямая и косвенная адресация			
Идентификатор операнда	**Максимальный диапазон адресов в соответствии с видом адресации**		
	Прямая	**Косвенная через память**	**Косвенная через регистр, внутри области**
I Q	от 0.0 до 65 535.7	[DBD] от 0 до [DID] 65 532 [LD] [MD]	[AR1, P#байт.бит] от 0.0 до 8 191.7 [AR2, P#байт.бит]
M	от 0.0 до 65 535.7	[DBD] от 0 до [DID] 65 532 [LD] [MD]	[AR1, P#байт.бит] от 0.0 до 8 191.7 [AR2, P#байт.бит]
DBX DIX L	от 0.0 до 65 535.7	[DBD] от 0 до [DID] 65 532 [LD] [MD]	[AR1, P#байт.бит] от 0.0 до 8 191.7 [AR2, P#байт.бит]

Таблица 5–2. Операнды: косвенная адресация с указанием области памяти через регистр	
Идентификатор операнда [1]	**Максимальный диапазон адресов**
I, Q, M, DBX, DIX или L	[AR1, P#байт.бит] от 0.0 до [AR2, P#байт.бит] 8 191.7

[1] Область памяти закодирована в битах указателя 24, 25 и 26 (см. раздел 3.6).

Таблица 5–3. Операнды: таймеры и счетчики		
Идентификатор операнда	**Максимальный диапазон адресов в соответствии с видом адресации**	
	Прямая	**Косвенная через память**
T C	от 0 до 65 535	[DBW] от 0 до 65 534 [DIW] [LW] [MW]

Таблица 5–4. Операнд: бит, таймер или счетчик, передаваемый как параметр	
Операнд	**Формат адресного параметра**
Символическое имя	Бит, таймер или счетчик, передаваемый как параметр

Таблица 5–5. Операнды битовых логических команд: биты слова состояния	
Область памяти или ссылка на адрес	**Биты слова состояния**
>0, <0, <>0, >=0, <=0, ==0	7 и 6: коды условий 1 и 0 (область памяти)
UO	7 и 6: коды условий 1 и 0 (область памяти)
BR	8: двоичный результат (адрес)
OV	5: переполнение (адрес)
OS	4: переполнение, сохраняемое (адрес)

Изменение битов в слове состояния

Команда	OR	STA	RLO	$\overline{FC}$
A	x	x	x	1
AN	x	x	x	1
A(	0	1	-	0
AN(	0	1	-	0
O	0	x	x	1
ON	0	x	x	1
O(	0	1	-	0
ON(	0	1	-	0
X	0	x	x	1
XN	0	x	x	1
X(	0	1	-	0
XN(	0	1	-	0
=	0	x	-	0
CLR	0	0	0	0
FN	0	x	x	1
FP	0	x	x	1
NOT	-	1	x	-
R	0	x	-	0
S	0	x	-	0
SAVE	-	-	-	-
SET	0	1	1	0

5.2 Битовые логические операции и релейно-контактные схемы

Введение

Логические команды, выполняемые над битами, называют также релейными логическими командами, так как они могут выполнять функцию релейной логической схемы. Ниже объясняется, как релейная логическая схема может быть воспроизведена с помощью команд AWL.

Нормально открытый контакт

Рис. 5–1 показывает релейную логическую схему с нормально открытым контактом управляющего реле между токовой шиной и катушкой. В нормальном состоянии этот контакт открыт. Если контакт не активизирован, он остается открытым. Состояние сигнала открытого контакта равно 0 (не активизирован). Если контакт открыт, то ток от токовой шины не может возбудить катушку в конце цепи. Если контакт активизирован (состояние сигнала контакта равно 1), то ток может протекать через катушку.

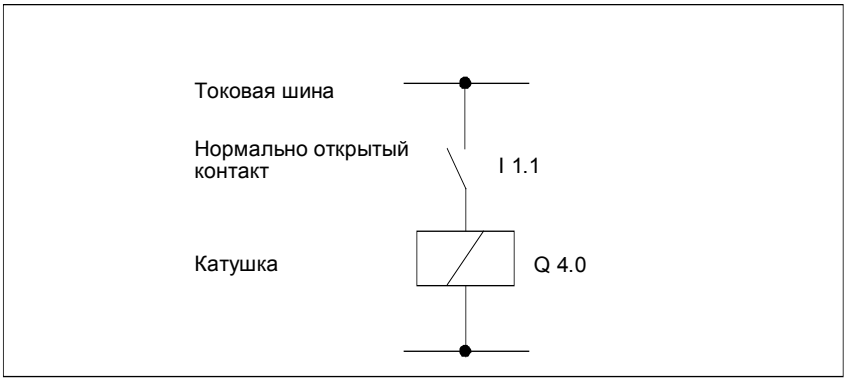


Рис. 5–1. Релейная логическая схема с нормально открытым контактом управляющего реле

Для опроса состояния сигнала нормально открытого контакта управляющего реле вы можете использовать команду И (A) или ИЛИ (O). Если нормально открытый контакт (I1.1 = 0), то результат опроса равен «0», если он замкнут, то результат опроса равен «1».

Нормально замкнутый контакт

Рис. 5–2 представляет релейную логическую схему с нормально замкнутым контактом управляющего реле между токовой шиной и катушкой. В нормальном состоянии этот контакт замкнут. Если контакт не активизирован, то он остается замкнутым. Состояние сигнала замкнутого контакта равно 0 (не активизирован). Если контакт замкнут, то ток от токовой шины может протекать через контакт, возбуждая катушку в конце цепи. Активизация контакта (состояние сигнала контакта равно 1) размыкает контакт, прерывая протекание тока к катушке.

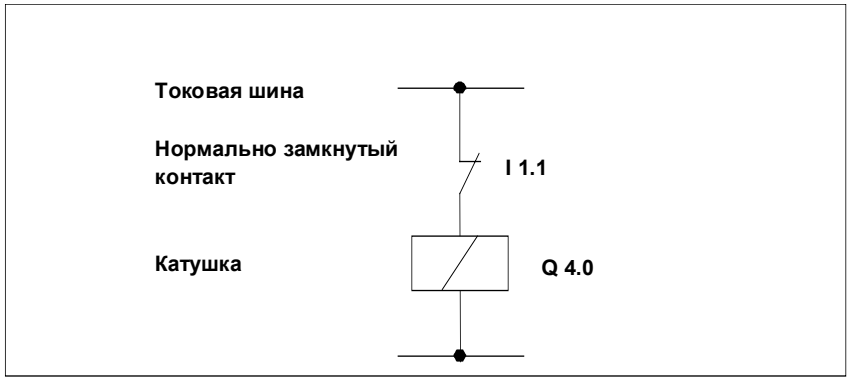


Рис. 5–2. Релейная логическая схема с нормально замкнутым контактом управляющего реле

Для опроса состояния сигнала нормально замкнутого контакта управляющего реле вы можете использовать команду И-НЕ (AN) или ИЛИ-НЕ (ON). Если нормально замкнутый контакт замкнут (I1.1 = 0), то результат опроса равен «1», если он открыт, то результат опроса равен «0».

Поток сигнала в последовательной цепи

Рис. 5–3 показывает пример списка операторов, использующего команду И (A) для программирования двух последовательно включенных нормально открытых контактов. Только в том случае, когда состояние сигнала обоих нормально открытых контактов равно «1», состояние выхода Q4.0 может быть установлено в «1», и катушка проводит ток.

Программа на AWL	Релейная логическая схема
A I 1.0	I 1.0
A I 1.1	I 1.1
= Q 4.0	Q 4.0

Рис. 5–3. Использование команды И для программирования последовательно включенных контактов

Поток сигнала в параллельной цепи

Рис. 5– показывает пример списка операторов, использующего команду ИЛИ (O) для программирования двух нормально открытых контактов, подключенных к катушке параллельно.

Выход Q4.0 может быть установлен в «1», и катушка пропускает ток только тогда, когда состояние сигнала хотя бы одного нормально открытого контакта равно «1».

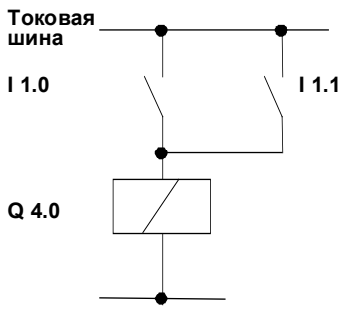
Программа на AWL	Релейная логическая схема
O I 1.0 O I 1.1 = Q 4.0	

Рис. 5–4. Использование команды ИЛИ для программирования параллельно включенных контактов

Исключающее ИЛИ

Команда *Исключающее ИЛИ* (X) в AWL соответствует релейной логической схеме, показанной на рис. 5–5, на которой соединены нормально замкнутый и нормально открытый контакты. Выход Q4.0 равен «1», когда I1.0 и I1.0 имеют разные значения.

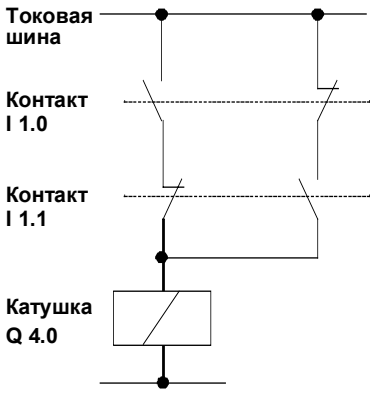
Программа на AWL	Релейная логическая схема
X I 1.0 X I 1.1 = Q 4.0	

Рис. 5–5. Использование команды *Исключающее ИЛИ* для программирования параллельно включенных контактов

AN, ON, XN

Аналогично можно с помощью команды

AN: **реализовать последовательное включение нормально замкнутого контакта.**

Программа на AWL	Релейная логическая схема
	Токовая шина
A I 1.0	I 1.0 Нормально открытый контакт
AN I 1.1	I 1.1 Нормально замкнутый контакт
= Q 4.0	Q 4.0 Катушка

ON: **реализовать параллельное включение нормально замкнутого контакта.**

Программа на AWL	Релейная логическая схема
	Токовая шина
O I 1.0	I 1.0 Нормально открытый контакт
ON I 1.1	I 1.1 Нормально замкнутый контакт
= Q 4.0	Q 4.0 Катушка

XN: **реализовать параллельное соединение последовательно включенных нормально замкнутых и нормально открытых контактов.**

Программа на AWL	Релейная логическая схема
	Токовая шина
X I 1.0	I 1.0 Нормально открытый контакт
XN I 1.1	I 1.1 Нормально замкнутый контакт
= Q 4.0	Q 4.0 Катушка

5.3 Анализ условий с помощью И, ИЛИ и исключающего ИЛИ

Описание

С помощью битовых логических команд можно опрашивать биты слова состояния CC 0, CC 1, BR, OV и OS. На эти биты оказывают влияние следующие команды (табл. 5–6).

Таблица 5–6. Команды, воздействующие на биты слова состояния CC, BR, OV и OS

Тип команды	Команда	Раздел в данном руководстве
Арифметика с фиксированной точкой	)I, *I, /I,<I,)D, *D, /D,<D, MOD	9.1
Операции сравнения (арифметика с фиксированной точкой)	==I, <>I, <I, <=I, >I, >=I, ==D, <>D, <D, <=D,>D, >=D	11.2
Арифметика с плавающей точкой	)R, *R,<R , /R, SQRT, SQR, LN, EXP, SIN, COS, TAN, ASIN, ACOS, ATAN	10.1
Сравнение чисел с плавающей точкой	==R, <>R, <R, <=R, >R, >=R	11.3
Преобразование	BTI, BTD, RND, RND*, RND), TRUNC, NEGI, NEGD	12.1, 12.2, and 12.4
Функции сдвига и циклического сдвига	SLW, SRW, SLD, SRD, SSI, SSD, RLD, RRD, RLDA, RRDA	14.1 and 14.2
Логические операции со словами	AW, OW, XOW, AD, OD, XOD	13.2
Скобочные выражения	)	5.4
Сохранение RLO в регистре BR	SAVE	5.9
Логическое управление	JCB, JNB, SPS	16.1
Управление программой	BEB, BE, BEA, CC, UC	17.6
Передача	T STW	8.3

Отношение результата к 0

Комбинация битов CC 1 и CC 0 в слове состояния может быть легко опрошена с помощью «замещающих операндов» (например, >0, ==0, <0 и т.д.). Табл. 5–7 показывает связь между различными комбинациями битов и упрощенным опросом. Например, вы можете опросить комбинацию CC 1 = 0 и CC 0 = 1 в команде И с помощью A <0.

Таблица 5–7. Комбинация состояний CC 0 и CC 1 и соответствующий вариант опроса

Если имеет место следующая комбинация сигналов в слове состояния,		**то опрос может происходить с помощью**
состояние сигнала CC 1	**состояние сигнала CC 0**	
1	0	>0
0	1	<0
0 или 1	1 или 0	<>0
1 или 0	0 или 0	>=0
0 или 0	1 или 0	<=0
0	0	==0
1	1	UO

AWL	**Объяснение**
L +10 // НИЖНЯЯ ГРАНИЦА	**Загрузить целое число 10 в младшее слово аккумулятора 1 в качестве нижней границы.**
L MW30	**Загрузить значение слова памяти MW30 в младшее слово аккумулятора 1, передав целое значение 10 в младшее слово аккумулятора 2.**
<=I	**10 меньше или равно значению MW30? Если да, то установить RLO в 1; иначе сбросить RLO в 0.**
L +100 // ВЕРХНЯЯ ГРАНИЦА	**Загрузить целое число 100 в младшее слово аккумулятора 1 в качестве верхней границы, передав значение MW30, сохраненное в младшем слове аккумулятора 1, в младшее слово аккумулятора 2.**
-I	**Вычесть 100 из значения MW30. Результат устанавливает CC 1 и CC 0 в соответствии с битовой комбинацией, показывающей отношение результата к 0 (см. табл. 5–7). RLO не меняется.**
A <=0	**В соответствии с битовой комбинацией в CC 1 и CC 0, выполняется ли условие <= 0? Да дает 1; нет дает 0 (см. табл. 5–7). Скомбинировать эту 1 или этот 0 с RLO в соответствии с таблицей истинности для И, сохранить результат в бите RLO.**
= Q 4.0	**Записать значение RLO как состояние сигнала выхода Q 4.0. Через катушку на выходе Q 4.0 протекает ток (она имеет состояние сигнала 1), если значение MW30 больше или равно 10 и меньше или равно 100.**

Булевы битовые логические команды могут также дать вашей программе возможность реагировать на ситуацию, когда результат арифметической операции над числами с плавающей точкой является недопустимым, так как одно из чисел не является допустимым числом с плавающей точкой (недопустимо, UO). Команда опрашивает состояние сигнала битов CC1 и CC0 слова состояния (см. табл. 5–7).

Переполнение и двоичный результат

Некоторые команды, перечисленные в табл. 5–6, могут устанавливать в 1 бит двоичного результата (BR) или биты переполнения (OV и OS) слова состояния. Вы можете использовать битовые логические команды A, AN, O, ON, X и XN вместе со следующими областями памяти, чтобы дать вашей программе возможность реагировать на переполнение или на установленный в 1 бит двоичного результата.

AWL	Объяснение
L MW10	**Загрузить целое число из слова памяти MW10 в младшее слово аккумулятора 1.**
L MW20	**Загрузить целое число из слова памяти MW20 в младшее слово аккумулятора 1, передав значение из MW10 в аккумулятор 2.**
+I	**Сложить два целых числа в аккумуляторах.**
T MW30	**Передать результат из младшего слова аккумулятора 1 в MW30.**
A I 0.0	**Опросить состояние сигнала на входе I 0.0 на равенство 1 или 0.**
A OV	**Опросить бит OV слова состояния на равенство 1 или 0.**
= Q 4.0	**Если состояние сигнала I 0.0 равно 1 и имеется 1 в бите OV слова состояния (т.е. во время последней арифметической операции произошло переполнение), установить состояние сигнала выхода Q 4.0 в 1; в противном случае установить его в 0.**

AWL	Объяснение
A BR	**Опросить бит BR слова состояния на равенство 1 или 0.**
= Q 4.0	**Если в бите BR слова состояния имеется 1, то установить состояние сигнала выхода Q 4.0 в 1; в противном случае установить его в 0.**

Адресация битов слова состояния

Булевы битовые логические команды оценивают условия с помощью операндов, приведенных в табл. 5–8.

Таблица 5–8. Операнды булевых битовых логических команд: биты слова состояния

Таблица 5–8. Операнды булевых битовых логических команд: биты слова состояния

Область памяти или ссылка на адрес	**Биты слова состояния**
>0, <0, <>0, >=0, <=0, ==0	7 и 6: коды условий 1 и 0 (область памяти)
UO	7 и 6: коды условий 1 и 0 (область памяти)
BR	8: двоичный результат (адрес)
OV	5: переполнение (адрес)
OS	4: переполнение, сохраняемое (адрес)

5.4 Скобочные выражения и И перед ИЛИ

Описание

Вы можете использовать команды И (A), ИЛИ (O) исключающее ИЛИ (X) и их инверсные формы AN, ON, XN для выполнения булевых логических операций над фрагментами логической цепи, заключенными в скобки (скобочные выражения). Скобки вокруг фрагмента логической цепи указывают, что ваша программа будет выполнять операции внутри скобок до выполнения логической операции, заданной командой, которая предшествует скобочному выражению.

Вы можете также комбинировать операторы И и ИЛИ в булевой логической цепи без скобок. **В соответствии с соглашением, сначала обрабатываются операторы И, а затем полученные результаты комбинируются в соответствии с таблицей истинности для ИЛИ.**

Результат логической операции

Команда, открывающая скобочное выражение, сохраняет RLO от предшествующей операции в скобочном стеке. Позднее программа свяжет этот сохраненный RLO с результатом, полученным при логических комбинациях внутри скобок.

AWL	Объяснение
A(O I 0.0	Операторы между командами A(и) образуют обычную логическую комбинацию ИЛИ. Результат первичного опроса сохраняется в бите RLO.
O M 10.0)	Результат опроса логически комбинируется с RLO, образованным предыдущим оператором, в соответствии с таблицей истинности для ИЛИ. Эта комбинация образует новый результат, который заменяет значение в бите RLO.
A(	A(копирует текущее значение бита RLO, сохраняет его в скобочном стеке и завершает предыдущую логическую цепь. Поэтому следующий логический оператор начинает новую логическую цепь, выполняя «первичный опрос».
O I 0.2	Операторы между командами A(и) образуют обычную логическую комбинацию ИЛИ. Результат первичного опроса сохраняется в бите RLO.
O M 10.3	Результат опроса логически комбинируется с RLO, образованным предыдущим оператором, в соответствии с таблицей истинности для ИЛИ. Эта комбинация образует новый результат, который заменяет значение в бите RLO.
)	Оператор) комбинирует RLO, сохраненное в скобочном стеке (см. выше команду «A(») с текущим значением RLO в соответствии с таблицей истинности для И. Этот оператор использует таблицу истинности для И, так как команда) завершает скобочное выражение, начатое командой A(. Эта логическая комбинация образует новый RLO.
A M 10.1	Этот обычный оператор И комбинирует новый RLO, образованный в предыдущей команде), с результатом данного опроса в соответствии с таблицей истинности для И.
= Q 4.0	Команда присваивания (=, см. раздел 5.8) присваивает значение RLO выходной катушке.

И перед ИЛИ

Приведенный ниже список команд использует принцип «И перед ИЛИ» для программирования переключательной схемы. По определению, программа сначала обрабатывает операцию И, а затем логически связывает результаты операции И в соответствии с таблицей истинности операции ИЛИ. Скобки не требуются. Принцип, применяемый здесь, называется «И перед ИЛИ».

AWL	Объяснение
A I 0.0	**Результат первичного опроса сохраняется в бите RLO.**
A M 10.0	**Результат опроса логически связывается с RLO, сформированным предыдущим оператором, в соответствии с таблицей истинности операции И. Данное логическое сопряжение создает новый результат, который заменяет имеющееся значение в бите RLO.**
O	**Оператор «O» копирует текущее значение бита RLO, сохраняет его в бите OR и завершает предыдущую логическую цепь. Оператор «O» сохраняет RLO как одно из двух значений, которое он будет использовать для того, чтобы выполнить операцию ИЛИ в соответствии с принципом «И перед ИЛИ».**
A I 0.2	**Результат первичного опроса сохраняется в бите RLO. В каждой операции И, которая следует за операцией ИЛИ, вновь образуемый RLO, логически связывается с битом OR.**
A M 0.3	**Результат опроса логически связывается с RLO, образованным предыдущим оператором, в соответствии с таблицей истинности операции И. Эта комбинация образует новый результат, который заменяет значение, находящееся в бите RLO. В каждой операции И вновь сформированный RLO комбинируется с битом OR.** **Первая операция «O» извлекает сохраненный RLO из скобочного стека и комбинирует его с текущим RLO. Эта операция приводит в результате к новому значению, которое сохраняется в бите RLO как результат операции «И перед ИЛИ». (eration. (Специальной операции для завершения «И перед ИЛИ» не существует. Специальный битовый процессор в программируемом контроллере находит последнюю операцию A в операции «И перед ИЛИ». Операция, следующая за последней операцией A (напр., =, S, R или O) завершает операцию «И перед ИЛИ» автоматически при вычислении RLO.)**
O M 10.1	**Следующая операция «O» логически связывает результат операции «И перед ИЛИ» с результатом опроса второй операции «O».**
= Q 4.0	**Операция присваивания (=, см. раздел 5.8) присваивает значение RLO выходу Q 4.0.**

Выход Q 4.0 становится активным (состояние его сигнала равно “1”), если результат одной **или** другой из пары операций И равен 1, **или** результат обычной операции ИЛИ над M 10.1 равен 1.

5.5 Команды для оценки фронтов: FP, FN

Описание

Вы можете использовать команды *Положительный фронт* (FP) и *Отрицательный фронт* (FN) как чувствительные к изменению сигнала контакты релейно-контактной схемы. Эти команды обнаруживают и реагируют на изменения результата логической операции. Переход с 0 на 1 называется «положительным фронтом». Переход с 1 на 0 называется «отрицательным фронтом» (см. рис. 5–6).

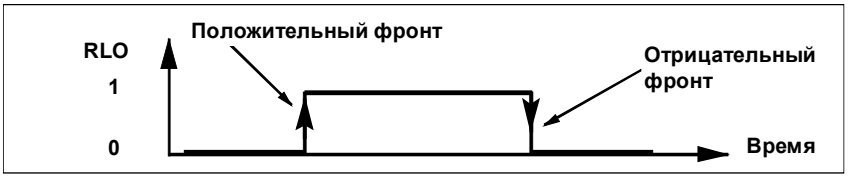


Рис. 5–6. Представление положительного и отрицательного фронтов

Реакция на положительный фронт

Рис. 5–7 показывает список операторов, который дает вашей программе возможность реагировать на положительный фронт. Объяснение следует за рисунком.

Рис. 5–7. Программирование реакции на положительный фронт

Если программируемый логический обнаруживает положительный фронт на контакте I 1.0, он активизирует выход Q 4.0 на один цикл OB1. Программируемый логический контроллер сохраняет результат логической операции, выполненной командой A, в бите памяти (меркере) фронта M 1.0 и сравнивает его с RLO предыдущего цикла сканирования. (В примере на рис. 5–7 RLO оператора «A I 1.0» оказался равным состоянию сигнала входа I 1.0. Однако это не будет иметь места в каждой программе). Если текущий RLO равен 1, а RLO предыдущего цикла сканирования, сохраненный в бите M 1.0, равен 0, то оператор FP устанавливает RLO в 1. Оператор FP обнаруживает положительный фронт на контакте (т.е. изменение состояния RLO с 0 на 1). Если RLO не меняется (текущий RLO и предыдущий RLO, сохраненный в бите памяти фронта, оба равны 0 или 1), то оператор FP сбрасывает RLO в 0.

Таблица 5–9. Опрос на наличие положительного фронта на входе I 1.0				
Номер цикла OB1	**Состояние сигнала на входе в предыдущем цикле**	**Состояние сигнала на входе в текущем цикле**	**Состояние сигнала сменилось с 0 на 1?**	**Катушка на выходе Q 4.0 проводит ток?**
1	0 (значение по умолчанию)	0	Нет	Нет
2	0	1	Да	Да
3	1	1	Нет	Нет
4	1	0	Нет	Нет
5	0	0	Нет	Нет
6	0	1	Да	Да
7	1	0	Нет	Нет
8	0	1	Да	Да
9	1	1	Нет	Нет

Таблица 5–9 относится конкретно к программе на AWL, показанной на рис. 5–7. В общем случае вы должны рассматривать изменения, обнаруживаемые командами FP и FN как изменения, нашедшие отражение в RLO, а не в состояниях сигналов контактов. Например, логическая цепь может образовать RLO, который не связан непосредственно с состоянием сигнала контакта.

Реакция на отрицательный фронт

Рис. 5–8 показывает список операторов, который дает вашей программе возможность реагировать на отрицательный фронт. Объяснение следует за рисунком.

Рис. 5–8. Программирование реакции на отрицательный фронт

Если программируемый логический обнаруживает отрицательный фронт на контакте I 1.0, он активизирует выход Q 4.0 на один цикл OB1. Программируемый логический контроллер сохраняет результат логической операции, выполненной командой A, в бите памяти (меркере) фронта M 1.0 и сравнивает его с RLO предыдущего цикла сканирования (см. табл. 5–10). (В примере на рис. 5–8 RLO оператора «A I 1.0» оказался равным состоянию сигнала входа I 1.0. Однако это не будет иметь места в каждой программе). Если текущий RLO равен 0, а RLO предыдущего цикла сканирования, сохраненный в бите M 1.0, равен 1, то оператор FN устанавливает RLO в 1.

Оператор FN обнаруживает отрицательный фронт на контакте (т.е. изменение состояния RLO с 1 на 0). Если RLO не меняется (текущий RLO и предыдущий RLO, сохраненный в бите памяти фронта, оба равны 0 или 1), то оператор FN сбрасывает RLO в 0.

Таблица 5–10. Опрос на наличие отрицательного фронта на входе I 1.0				
Номер цикла OB1	**Состояние сигнала на входе в предыдущем цикле**	**Состояние сигнала на входе в текущем цикле**	**Состояние сигнала сменилось с 1 на 0?**	**Катушка на выходе Q 4.0 проводит ток?**
1	0 (значение по умолчанию)	0	Нет	Нет
2	0	1	Нет	Нет
3	1	0	Да	Да
4	0	0	Нет	Нет
5	0	1	Нет	Нет
6	1	1	Нет	Нет
7	1	1	Нет	Нет
8	1	0	Да	Да
9	0	0	Нет	Нет

Таблица 5–10 относится конкретно к программе на AWL, показанной на рис. 5–8. В общем случае вы должны рассматривать изменения, обнаруживаемые командами FP и FN как изменения, нашедшие отражение в RLO, а не в состояниях сигналов контактов. Например, логическая цепь может образовать RLO, который не связан непосредственно с состоянием сигнала контакта.

Адресуемый бит

Адрес, к которому обращается команда FP или FN, является битом Команда обращается к выходу посредством одного из следующих видов операндов:

- Идентификатор (ID) операнда и адрес внутри области памяти, заданной идентификатором операнда (см. таблицы 5–11 и 5–12)
- Бит, передаваемый как параметр (см. табл. 5–13)

Таблица 5–11. Операнды FP и FN: прямая и косвенная адресация					
ID операнда [1]	**Максимальный диапазон адресов в зависимости от способа адресации**				
	Прямая	**Косвенная через память**		**Косвенная внутри области через регистр**	
I[2] Q[3] M DBX DIX	от 0.0 до 65 535.7	[DBD] [DID] [LD] [MD]	от 0 до 65 532	[AR1, P#байт.бит] [AR2, P#байт.бит]	от 0.0 до 8 191.7

[1] См. **Предупреждение**, следующее за этой таблицей.

[2] Так как операционная система в начале каждого цикла переписывает таблицу входов образа процесса, то RLO, сохраненный командой FP или FN, который использует входной бит в качестве операнда, искажается. См. **Предупреждение**, следующее за этой таблицей.

[3] Не рекомендуется использовать выходной бит в качестве операнда команды FP или FN. Если вы желаете воздействовать на выход, то используйте для этого команду S, R или =..

Предупреждение

Искажение сохраненного результата логической операции.

Это может иметь вызвать небольшой материальный ущерб.

Если вы используете в своей программе команду FP или FN, то бит памяти, являющийся операндом этой команды, используется FP или FN исключительно для своих собственных целей запоминания. Поэтому вы не должны использовать команды, которые могли бы изменить этот бит. В противном случае вы исказите сохраненный RLO. Это предупреждение относится ко всем областям памяти, задаваемым идентификаторами операнда, перечисленными в табл. 5–11.

Таблица 5–12. Операнды FP и FN: косвенная адресация с указанием области памяти через регистр

Идентификатор операнда[1]	**Диапазон адресов**
I, Q, M, DBX или DIX	[AR1, P#байт.бит] от 0.0 до [AR2, P#байт.бит] 8 191.7

[1] Область памяти закодирована соответственно в AR1 или AR2 (см. раздел 3.6).

Таблица 5–13. Операнды FP и FN: бит, передаваемый как параметр

Операнд	**Формат адресного параметра**
Символическое имя	Бит, передаваемый как параметр

5.6 Выход цепи булевых логических операций

Описание

Вы можете завершить булеву логическую цепь с помощью одной из следующих команд AWL. Каждая из этих команд может воздействовать на бит, представляющий конец этой цепи.

- *Установить* (S): если RLO в предыдущей команде был установлен в 1, то S устанавливает в 1 состояние сигнала контакта или катушки, к которой обращается эта команда;
- *Сбросить* (R): если RLO в предыдущей команде был установлен в 1, то R сбрасывает в 0 состояние сигнала контакта или катушки, к которой обращается эта команда;
- *Присвоить* (=):независимо от состояния RLO его значение присваивается операнду, к которому обращается эта команда.

Завершение логической цепи

Логическая цепь завершается, когда сбрасывается бит первичного опроса (бит $\overline{FC}$). Когда значение бита $\overline{FC}$ равно 0, это указывает, это указывает, что следующая команда в программе является первой командой новой логической цепи (см. раздел 2.2, *Первичный опрос*). Команда *Установить* (S), *Сбросить*) или *Присвоить* (=) завершает логическую цепь сбросом бита первичного опроса (бита $\overline{FC}$) в 0. (Команды условного перехода также сбрасывают бит $\overline{FC}$ в 0, см. разделы 16.3 – 16.5).

Логические цепи, начинающиеся командами A(, AN(, O(и т.д., должны завершаться командой). Так как эти команды могут использоваться также в середине логической цепи, то они образуют разрыв в этой цепи. Это значит, что новая логическая цепь начинается раньше, чем завершена старая. Для продолжения старой логической цепи в правильной последовательности после завершения команд, которые должны быть выполнены в скобках, старый бит $\overline{FC}$ (сохраненный при открытии скобок) снова восстанавливается.

Поэтому вы можете представить разделы программы внутри скобок как своего рода промежуточные вычисления, после завершения которых восстанавливается старая последовательность.

5.7 Команды установки и сброса: S и R

Описание

Вы можете использовать команду *Установить* (S) для установки в 1 состояния сигнала адресуемого бита. (За информацией об использовании команды S для установки на заданное значение адресуемого счетчика обратитесь к разделу 7.2).

Вы можете использовать команду *Сбросить* (R) для сброса в 0 состояния сигнала адресуемого бита. R может также сбрасывать в 0 адресуемый таймер или счетчик (см. разделы 6.3 и 7.2). S и R завершают логическую цепь (см. раздел 5.6).

Установка бита

Команда S устанавливает бит, к которому производится обращение, в 1, если результат логической операции от предыдущего оператора равен 1 и главное управляющее реле (MCR) активизировано (т.е. состояние его сигнала равно 1). Если MCR не активизировано (состояние его сигнала равно 0), то адресуемый бит не меняется. Команда S завершает логическую цепь.

Рис. 5–9 показывает, как команда S удерживает в 1 состояние сигнала адресуемой катушки Q 4.0, пока команда R не изменит состояние сигнала в 0. Тот факт, что состояние сигнала адресуемой катушки остается равным 1, пока команда R не сбросит его в 0, указывает на статический характер команды S.

В релейной схеме, если нормально открытый контакт на входе I 1.0 активизируется (его состояние сигнала становится равным 1), контакт замыкается. Ток течет через контакт на входе I 1.0 и через нормально замкнутый контакт под ним и активизирует катушку на выходе Q 4.0 (состояние сигнала Q 4.0 становится равным 1).

Когда катушка активизируется, нормально открытый контакт Q 4.0 напротив I 1.0 замыкается. После этого, независимо от того открыт или замкнут контакт на входе I 1.0, катушка на выходе Q 4.0 остается активизированной (ее состояние сигнала равно 1). Катушка держит сама себя под напряжением.

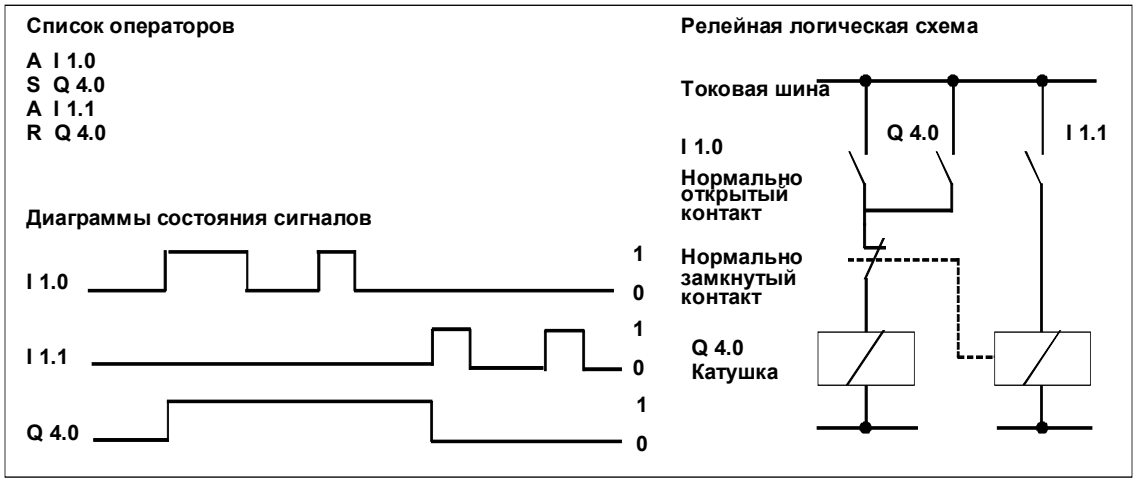


Рис. 5–9. Статическая установка и сброс бита

Сброс бита

Команда R сбрасывает бит, к которому производится обращение, в 0, если результат логической операции от предыдущего оператора равен 1 и главное управляющее реле (MCR) активизировано (т.е. состояние его сигнала равно 1). Если MCR не активизировано (состояние его сигнала равно 0), то адресуемый бит не меняется. Команда R завершает логическую цепь.

Рис. 5–9 показывает, как команда R удерживает в 0 состояние сигнала адресуемой катушки Q 4.0 независимо от изменения состояния сигнала на контакте, который запускает сброс (I 1.1). Тот факт, что состояние сигнала адресуемой катушки остается равным 0, пока команда S не установит его вновь в 1, указывает на статический характер команды R.

В релейной схеме катушка на выходе Q 4.0, которая была активизирована командой S, обесточивается (состояние ее сигнала становится равным 0) при замыкании нормально открытого контакта на входе I 1.1. Замыкание контакта I 1.1 дает возможность протекать току через нижерасположенную катушку. Эта катушка размыкает нормально замкнутый контакт над катушкой на выходе Q 4.0, прерывая протекание тока через эту катушку. Замыкание контакта I 1.1 запускает команду R.

Операнд, к которому происходит обращение

Операнд, к которому обращается команда S, может быть битом. Операнд, к которому обращается команда R, может быть битом, номером таймера или номером счетчика. Эти операнды могут быть заданы с помощью:

- идентификатора (ID) операнда и адреса внутри области памяти, заданной идентификатором операнда (см. таблицы с 5–14 по 5–16)
- бита, таймера или счетчика, передаваемого в качестве параметра (см. табл. 5–17).

Таблица 5–14. Операнды команд S и R: прямая и косвенная адресация

ID операнда	**Максимальный диапазон адресов в зависимости от вида адресации**		
	прямая	**косвенная через память**	**косвенная внутри области через регистр**
I Q	от 0.0 до 65 535.7	[DBD] от 0 до [DID] 65 532 [LD] [MD]	[AR1, P#байт.бит] от 0.0 до 8 191.7 [AR2, P#байт.бит]
M	от 0.0 до 255.7	[DBD] от 0 до [DID] 65 532 [LD] [MD]	[AR1, P#байт.бит] от 0.0 до 8 191.7 [AR2, P#байт.бит]
DBX DIX L	от 0.0 до 65 535.7	[DBD] от 0 до [DID] 65 532 [LD] [MD]	[AR1, P#байт.бит] от 0.0 до 8 191.7 [AR2, P#байт.бит]

Таблица 5–15. Операнды команд S и R: косвенная адресация с указанием области памяти через регистр

Идентификатор операнда[1]	**Максимальный диапазон адресов**
I, Q, M, D, DBX, DIX или L	[AR1, P#байт.бит] от 0.0 до [AR2, P#байт.бит] 8 191.7

[1] Область памяти закодирована в битах указателя 24, 25 и 26 (см. раздел 3.6).

Таблица 5–16. Операнды команды R: таймеры и счетчики		
Идентификатор операнда	**Максимальный диапазон адресов в зависимости от вида адресации**	
	прямая	**косвенная через память**
T[1] C	от 0 до 65 535	[DW] от 0 до 65 534 [DXW] [LW] [MW]

[1] Команда S, устанавливающая адресуемый бит в 1, неприменима к таймерам и счетчикам. Команда S, используемая со счетчиком, устанавливает этот счетчик на заданное значение. Таймеры запускаются командами для определенных типов таймеров (см. разделы 6.2, 6.3 и 7.2).

Таблица 5–17. Операнды команд S и R: бит, таймер или счетчик, передаваемый как параметр	
Операнд	**Формат адресного параметра**
Символическое имя	Бит, таймер [1] или счетчик, передаваемый как параметр

[1] Команда S неприменима к таймерам. Таймеры запускаются командами для определенных типов таймеров (см. разделы 6.2 и 6.3).

5.8 Команда присваивания (=)

Описание

Каждая булева логическая операция дает результат, называемый «результатом логической операции» (RLO). Этот RLO равен 1 или 0. Относительно контактов и катушек 1 означает протекание тока, а 0 означает, что ток не течет.

Вы можете использовать команду *Присвоить* (=) для копирования RLO от предыдущего оператора в логической цепи и присваивания этого RLO в качестве состояния сигнала катушке, к которой обращается команда =. Команда = завершает логическую цепь (см. раздел 5.6).

Установка или сброс бита

Значение, присваиваемое командой = катушке, к которой она обращается, может быть равно 1 или 0, в зависимости от RLO оператора, предшествующего оператору =. В отличие от команд S и R команда = имеет динамический характер. Она присваивает RLO в качестве состояния сигнала катушке, к которой обращается команда =. Рис. 5–10 показывает, как изменяется это значение, когда изменяется RLO оператора «A I 1.0».

На рис. 5–10 команда = дает возможность входному сигналу на контакте I 1. 0 активизировать или деактивизировать катушку на выходе Q 4.0 (т.е. команда = устанавливает или сбрасывает бит, представленный операндом Q 4.0, путем присваивания RLO предыдущего оператора).

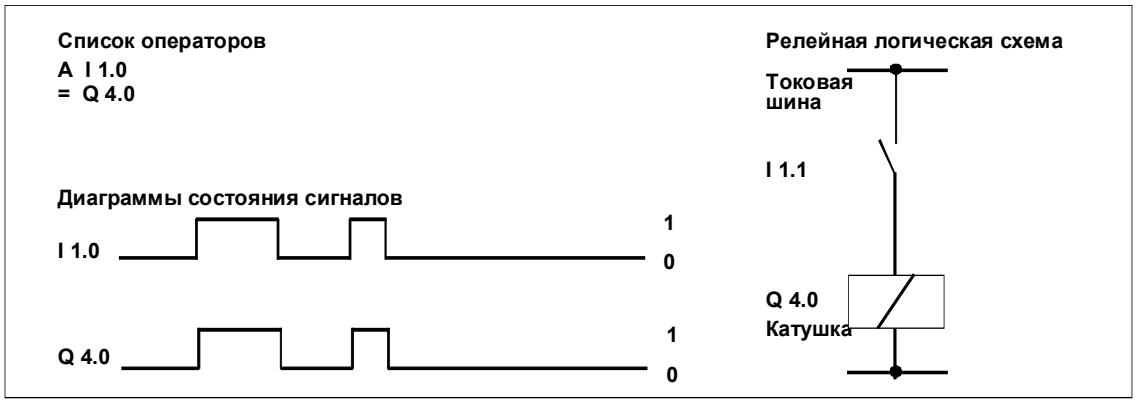


Рис. 5–10. Динамическая установка и сброс бита

Операнды

Операнд, к которому обращается команда =, может быть битом. Команда обращается к катушке через один из следующих операндов:

- идентификатор (ID) операнда и адрес внутри области памяти, указанной идентификатором операнда (см. таблицы 5–18 и 5–19)
- бит, передаваемый как параметр (см. табл. 5–20)

Таблица 5–18. Операнды команды =: прямая и косвенная адресация

ID операнда	**Максимальный диапазон адресов в зависимости от вида адресации**		
	прямая	**косвенная через память**	**косвенная внутри области через регистр**
I Q	от 0.0 до 65 535.7	[DBD] от 0 до [DID] 65 532 [LD] [MD]	[AR1, P#байт.бит] от 0.0 до 8 191.7 [AR2, P#байт.бит]
M	от 0.0 до 65 535	[DBD] от 0 до [DID] 65 532 [LD] [MD]	[AR1, P#байт.бит] от 0.0 до 8 191.7 [AR2, P#байт.бит]
DBX DIX L	от 0.0 до 65 535.7	[DBD] от 0 до [DID] 65 532 [LD] [MD]	[AR1, P#байт.бит] от 0.0 до 8 191.7 [AR2, P#байт.бит]

Таблица 5–19. Операнды команды =: косвенная адресация с указанием области памяти через регистр

Идентификатор операнда[1]	**Максимальный диапазон адресных параметров**
I, Q, M, DBX, DIX или L	[AR1, P#байт.бит] от 0.0 до [AR2, P#байт.бит] 8 191.7

[1] Область памяти закодирована в старших 8 битах AR1 или AR2 соответственно

Таблица 5–20. Операнды команды =: бит, передаваемый как параметр

Операнд	**Формат адресного параметра**
Символическое имя	Бит, передаваемый как параметр

5.9 Отрицание, установка, сброс и сохранение RLO

Описание

Вы можете использовать одну из следующих команд для изменения результата логической операции (RLO), сохраненного в бите RLO слова состояния программируемого логического контроллера (см. раздел 5.8):

Мнемоника	Команда	Значение
NOT	Инвертировать RLO	Отрицание (обращение) текущего RLO
SET	Установить RLO	Установка текущего RLO в 1
CLR	Сбросить RLO	Сброс текущего RLO в 0
SAVE	Сохранить RLO в регистре BR	Сохранение текущего RLO в бите слова состояния

Так как эти команды воздействуют непосредственно на RLO, то они не имеют операндов.

Отрицание RLO

Вы можете использовать команду *Инвертировать RLO* (NOT) в своей программе для отрицание (инвертирования) текущего RLO. Если текущий RLO равен 0, то NOT изменяет его на 1; если текущий RLO равен 1, то NOT изменяет его на 0, если бит OR не установлен. Эта команда полезна для сокращения вашей программы, например, за счет перехода от положительной логики к отрицательной (см. пример таймера в разделе B.3).

Установка RLO в 1

Вы можете использовать команду *Установить RLO* (SET) в своей программе, если вам нужно установить бит RLO в 1 безусловно. Рис. 5–11 показывает, как команда SET работает в программе.

Сброс RLO в 0

Вы можете использовать команду *Сбросить RLO* (CLR) в своей программе, если вам нужно сбросить бит RLO в 0 безусловно. CLR сбрасывает в 0 также биты $\overline{FC}$, OR и STA. В результате логическая цепь завершается.
Рис. 5–11 показывает, как команда CLR работает в программе.

Сохранение RLO

Вы можете использовать команду *Сохранить RLO в регистре BR* (SAVE) в своей программе, если вам нужно сохранить RLO для будущего использования или если вы хотите воздействовать на бит BR слова состояния в программируемом логическом контроллере, например, когда вы программируете функциональные блоки (FB) и функции (FC) для блоков контактного плана.

Команда	Воздействие на биты слова состояния								
	BR	**A1**	**A0**	**OV**	**OS**	**OR**	**STA**	**RLO**	**$\overline{FC}$**
NOT						-	1	x	-
SET						0	1	1	0
CLR						0	0	0	0
SAVE	x								

Применение SET и CLR

Программа, показанная на рис. 5–11, иллюстрирует применение команд SET и CLR, которые устанавливают и сбрасывают бит безусловно.

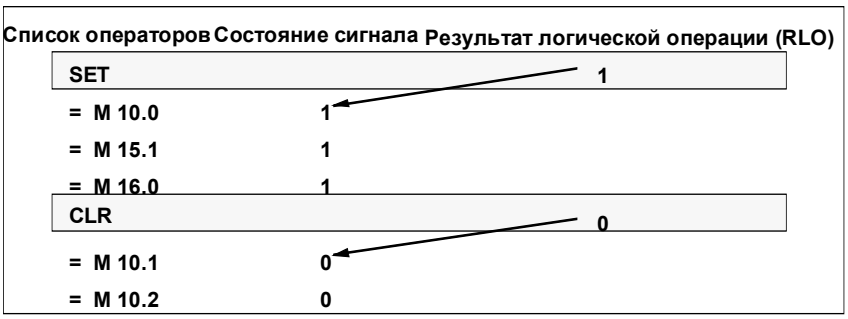


Рис. 5–11. Безусловная установка и сброс бита с помощью SET и CLR

Вы можете использовать операторы программы, показанной на рис. 5–11, в организационном блоке (OB) запуска. После включения вашего программируемого логического контроллера он обрабатывает OB запуска со всеми командами, которые он содержит. После выполнения контроллером всех этих команд следующие биты памяти имеют определенное состояние сигнала независимо от любых условий:

- состояние сигнала битов памяти M 10.0, M 15.1 и M 16.0 равно 1.
- состояние сигнала битов памяти M 10.1 и M 10.2 равно 0.

6 Таймерные команды

Обзор главы

6.1 Обзор

Определение

Таймер – это элемент языка программирования STEP 7, который реализует и контролирует процессы, управляемые временем. Таймерные команды дают вашей программе возможность выполнять следующие функции:

- Обеспечение времен ожидания. Например, согласно технологии литья под давлением форма должна оставаться закрытой в течение двух секунд. Ваша программа обеспечивает, что деталь, отлитая под давлением, будет извлечена из формы по истечении двух секунд.
- Обеспечение времен контроля. Например, программа контролирует скорость двигателя в течение 30 секунд после нажатия пусковой кнопки.
- Генерирование импульсов. Например, ваша программа подает импульсы, вызывающие мигание лампы.
- Измерение времени. Например, ваша программа может определить, сколько времени занимает наполнение резервуара.

Имеющиеся в распоряжении команды

Представление языка программирования STEP 7 в виде списка операторов предоставляет в распоряжение следующие команды:

- Запуск таймера одного из следующих типов:
  - формирователя импульса (SP)
  - формирователя удлиненного импульса (SE)
  - формирователя задержки включения (SD)
  - формирователя задержки включения с запоминанием (SS)
  - формирователя задержки выключения (SF)
- Сброс таймера (R)
- Разрешение на запуск (разблокировка) таймера (FR)
- Загрузка таймера в одном из следующих форматов:
  - в двоичном коде (L)
  - в двоично-десятичном коде (LC)
- Опрос состояния сигнала таймера и логическое сопряжение результата опроса с помощью булевой логической операции (A, AN, O, ON, X, XN) (см. главу 11).

Рис. 6–1 дает обзор команд, использующих слово таймера в качестве операнда.

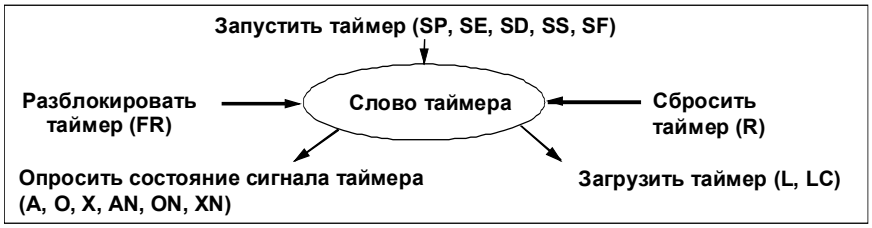


Рис. 6–1. Команды, использующие слово таймера в качестве операнда

6.2 Размещение таймера в памяти и компоненты таймера

Область в памяти

Таймеры имеют собственную зарезервированную область памяти в вашем CPU. Эта область памяти резервирует одно 16-битное слово для каждого адреса таймера. Набор команд списка операторов поддерживает 256 таймеров. Чтобы установить количество доступных таймерных слов, обратитесь к техническому описанию вашего CPU.

К области памяти таймеров имеют доступ следующие функции:

- Таймерные команды
- Обновление таймерных слов с помощью генератора тактовых импульсов. Эта функция уменьшает заданное значение времени на одну единицу через интервалы времени, установленные базой времени, пока значение времени не станет равным нулю.

Значение времени

Биты с 0 по 9 в таймерном слове содержат значение времени в двоичном коде. Значение времени задает количество единиц. Функция обновления таймера уменьшает значение времени на одну единицу через интервал времени, установленный базой времени. Значение времени уменьшается до тех пор, пока оно не станет равным нулю. Вы можете загружать значение времени в младшее слово аккумулятора 1 в двоичном, шестнадцатеричном или двоично-десятичном (BCD) коде. Диапазон времени охватывает значения с 0 по 9 990 секунд.

Вы можете предварительно загрузить значение времени с использованием любого из следующих форматов:

- L W#16#wxyz
  - где w = база времени (то есть интервал времени или разрешающая способность)
  - xyz = значение времени в двоично-десятичном формате
- L S5T# aH_bbM_ccS_dddMS
  - где a = часы, bb = минуты, cc = секунды и ddd = миллисекунды
  - База времени выбирается автоматически и значение округляется до ближайшего меньшего числа с этой базой времени.

Максимальное значение времени, которое вы можете ввести, равно 9 990 секунд или 2H_46M_30S.

База времени

Биты 12 и 13 в таймерном слове содержат базу времени в двоичном коде. База времени определяет интервал, через который значение времени уменьшается на одну единицу. Минимальная база времени равна 10 мс; максимальная - 10 с.

Таблица 6–1. База времени и ее двоичный код	
База времени	**Двоичный код для базы времени**
10 мс	00
100 мс	01
1 с	10
10 с	11

Так как значения времени запоминаются только через один интервал времени, то значения, не являющиеся точными кратными интервала времени, урезаются. Значения, разрешающая способность которых слишком велика для желаемого диапазона, округляются таким образом, что достигается желаемый диапазон, но не желаемая разрешающая способность. Таблица 6–2 показывает возможные разрешающие способности и соответствующие им диапазоны.

Таблица 6–2. Разрешающие способности и диапазоны для базы времени	
Разрешающая способность	**Диапазон**
0,01 секунды	от 10MS до 9S_990MS
0,1 секунды	от 100MS до 1M_39S_900MS
1 секунда	от 1S до 16M_39S
10 секунд	от 10S до 2HR_46M_30S

Конфигурация битов в аккумуляторе 1

Когда таймер запускается, содержимое аккумулятора 1 используется в качестве значения времени. Биты с 0 по 11 в младшем слове аккумулятора 1 содержат значение времени в двоично-десятичном формате (BCD–формат: каждая группа из четырех битов содержит двоичный код одного десятичного разряда). Биты 12 и 13 содержат базу времени в двоичном коде (см. табл. 6–1). Рис. 6–2 показывает содержимое младшего слова аккумулятора 1, загруженного значением таймера 127 с базой времени 1 секунда. (См. также раздел 8.5.)

Рис. 6–2. Содержимое младшего слова аккумулятора 1: значение времени 127, база времени 1 секунда

6.3 Загрузка, запуск, сброс и разблокировка таймера

Описание

Для запуска таймера в своей программе на AWL включите три оператора для реализации следующих операций:

- Опрос состояния сигнала на равенство 0 или 1 (например, A I 2.1)
- Загрузка значения времени и соответствующей базы времени (например, L IW0)
- Запуска таймера одного из следующих типов:
  - формирователя импульса (SP, например, SP T 1)
  - формирователя удлиненного импульса (SE)
  - формирователя задержки включения (SD)
  - формирователя задержки включения с запоминанием (SS)
  - формирователя задержки выключения (SF)

В вашей программе на AWL изменение результата логической операции (RLO) перед командой запуска приводит к запуску таймера. Изменение RLO с 1 на 0 запускает таймер в качестве формирователя задержки выключения (SF); изменение с 0 на 1 запускает любой из остальных таймеров. Запрограммированное время и операторы запуска таймера должны следовать непосредственно за операцией, определяющей условие запуска таймера. Раздел 6.4 дает примеры пяти типов команд запуска для таймеров.

Загрузка

Загрузка значения времени в виде целого числа или числа в формате BCD описана в разделах 8.4 и 8.5.

Стартовое время

Так как таймер работает в обратном направлении от установленного времени к нулю, то вы должны снабдить таймер стартовым временем. Когда вы запускаете таймер в своей программе, CPU ищет стартовое время в аккумуляторе 1. Диапазон времени составляет от 0 до 9 990 секунд.

Пример запуска таймера

Рис. 6–3 дает пример запуска таймера в качестве формирователя импульса. Изменение состояния сигнала с 0 на 1 на входе I 2.1 запускает таймер. Рис. 6–3 относится к следующей программе на языке AWL:

AWL		Объяснение
A	**I 2.1**	**Опросить состояние сигнала на входе I 2.1.**
L	**S5T#00H02M23S00MS**	**Загрузить стартовое время в аккумулятор 1.**
SP	**T 1**	**Запустить таймер T1 как формирователь импульса.**

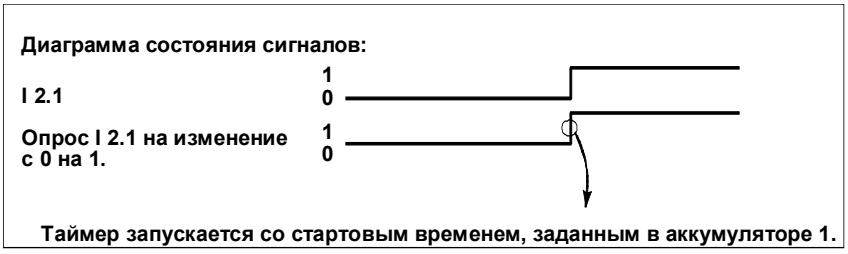


Рис. 6–3. Запуск таймера как формирователя импульса

Сброс таймера

Таймер сбрасывается с помощью команды *Сбросить* (R). CPU сбрасывает таймер, если в вашей программе результат логической операции непосредственно перед командой R равен 1. Пока RLO перед командой R равен 1, команда A, O или X, опрашивающая состояние сигнала таймера, дает результат, равный 0, а команда AN, ON или XN дает результат, равный 1.

Сброс таймера останавливает текущее функционирование таймера и сбрасывает значение времени в 0.

Разблокировка таймера для повторного пуска

Смена результата логической операции с 0 на 1 перед командой *Разблокировать* (FR) разблокирует таймер. Эта смена состояния сигнала всегда необходима для разблокировки таймера. CPU выполняет команду FR только при положительном фронте сигнала.

Для запуска или нормальной работы таймера разблокировка не требуется. Разблокировка используется только для перезапуска работающего таймера. Такой перезапуск возможен только, когда операция запуска продолжает обрабатываться с RLO, равным 1.

6.4 Примеры таймеров

Введение

Чтобы удовлетворить ваши потребности в решении задач автоматизации, программирование в форме списка операторов предоставляет в ваше распоряжение пять видов таймеров. Ниже приведен пример для каждого вида таймеров.

Таймер как формирователь импульса: SP

Рисунки 6–4 и 6–5 дают примеры таймера как формирователя импульса. Числа в прямоугольниках на рисунках указывают на объяснения, следующие за рисунком 6–4. Эти рисунки относятся к следующей программе на AWL:

AWL		Объяснение
A	**I 2.0**	
FR	**T 1**	**Разблокировать таймер T1.**
A	**I 2.1**	
L	**S5T#0H2M23S0MS**	
SP	**T 1**	**Запустить таймер T1 в качестве формирователя импульса.**
A	**I 2.2**	
R	**T 1**	**Сбросить таймер T1.**
A	**T 1**	**Опросить состояние сигнала таймера T1.**
=	**Q 4.0**	
L	**T 1**	
T	**MW10**	
LC	**T 1**	**Загрузить таймер T1 в аккумулятор.**
T	**MW12**	

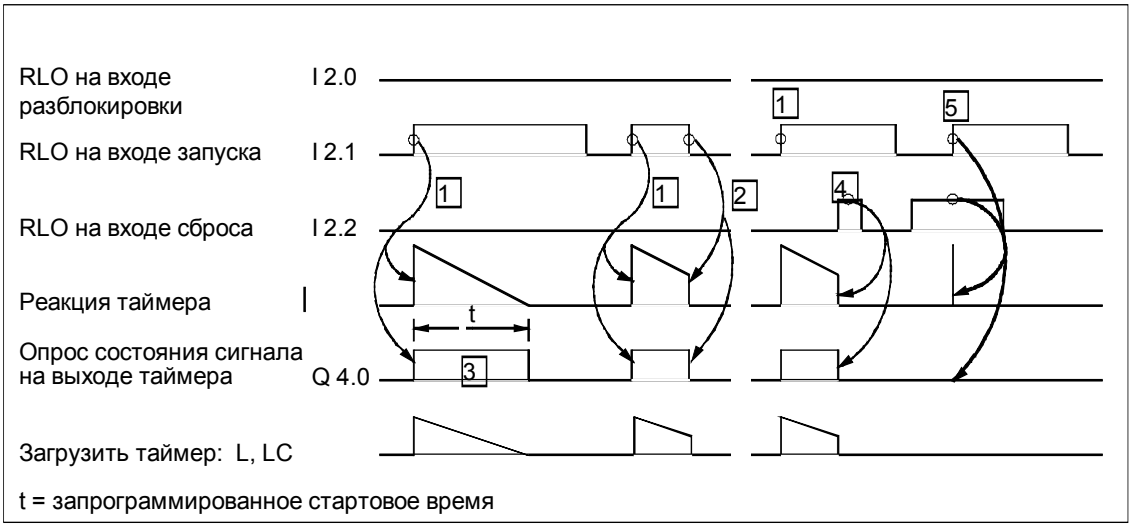


Рис. 6–4. Пример таймера как формирователя импульса, часть 1

Следующий список описывает элементы рисунков 6–4 и 6–5:

1. Изменение RLO с 0 на 1 на входе запуска запускает таймер. С этого момента отсчитывается запрограммированное время t.
2. Если на входе запуска RLO равен 0, то таймер сбрасывается.
3. Опрос состояния сигнала на выходе Q 4.0 таймера дает состояние сигнала, равное 1, в течение всей работы таймера.
4. Если RLO, равный 1, прикладывается к входу сброса, то таймер сбрасывается. Пока на входе запуска состояние сигнала остается равным 1, изменение RLO с 1 на 0 на входе сброса, не оказывает никакого влияния на таймер.
5. Изменение RLO с 0 на 1 на входе запуска при наличии сигнала на входе сброса приводит к кратковременному запуску с немедленным сбросом из-за следующего непосредственно далее в программе оператора сброса (на рис. 6–4 показано в виде импульсной линии на временной диаграмме). Для этого импульса не может быть получен результат опроса, если соблюдается описанная выше последовательность записи операторов.
6. Изменение RLO с 0 на 1 на входе разблокировки во время работы таймера перезапускает таймер. Запрограммированное время используется при перезапуске как текущее время. Изменение RLO на входе разблокировки с 1 на 0 воздействия не оказывает.
7. Если RLO меняется с 0 на 1 на входе разблокировки, когда таймер не работает, а на входе запуска RLO все еще равен 1, то таймер также будет запущен как формирователь импульса с запрограммированным временем.
8. Изменение RLO с 0 на 1 на входе разблокировки, когда RLO на входе запуска еще равен 0, не оказывает воздействия на таймер.

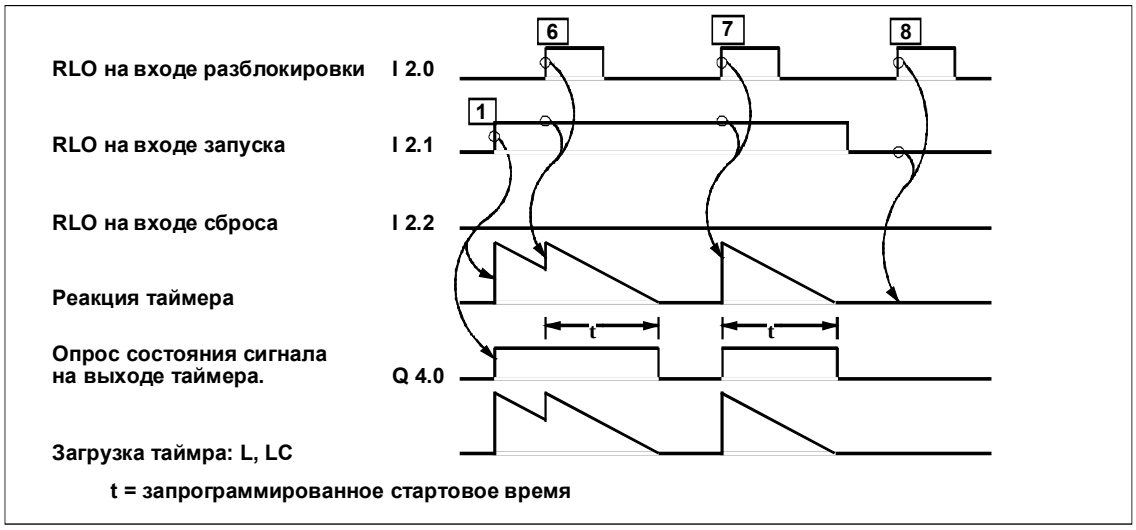


Рис. 6–5. Пример таймера как формирователя импульса, часть 2

Таймер как формирователь удлиненного импульса: SE

Рисунки 6–6 и 6–7 дают примеры таймера как формирователя удлиненного импульса. На рисунках числа в прямоугольниках указывают на объяснения, следующие за рис. 6–6. Рисунки относятся к следующей программе на AWL:

AWL		Объяснение
A	**I 2.0**	
FR	**T 1**	**Разблокировать таймер T1.**
A	**I 2.1**	
L	**S5T#0H2M23S0MS**	
SE	**T 1**	**Запустить таймер T1 в качестве формирователя удлиненного**
A	**I 2.2**	**импульса.**
R	**T 1**	**Сбросить таймер T1.**
A	**T 1**	
=	**Q 4.0**	**Опросить состояние сигнала таймера T1.**
L	**T 1**	**Загрузить таймер T1 (в двоичном коде).**
T	**MW10**	
LC	**T 1**	**Загрузить таймер T 1 (в BCD–коде).**
T	**MW12**	

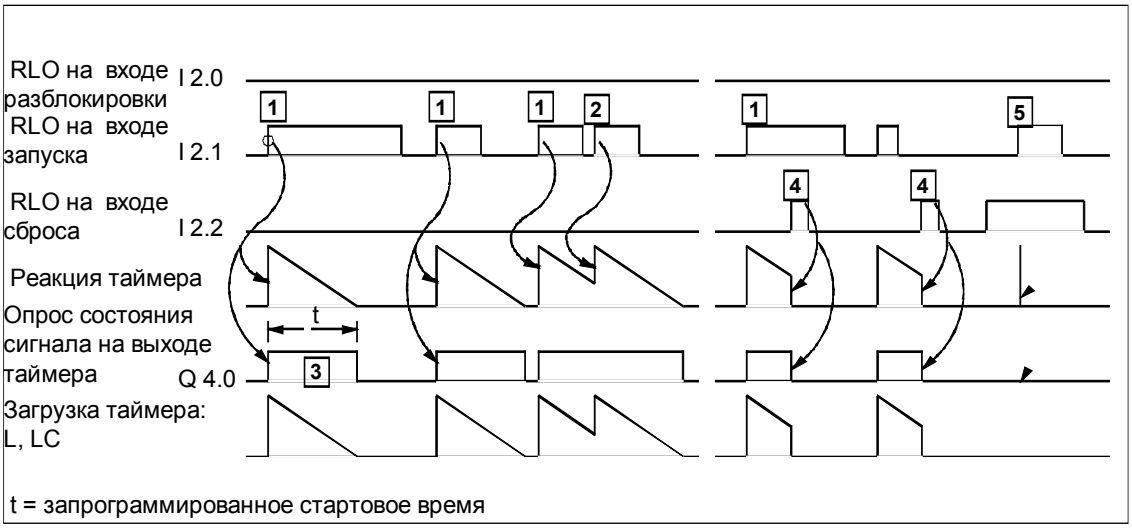


Рис. 6–6. Пример таймера как формирователя удлиненного импульса, часть 1

Следующий список описывает элементы рисунков 6–6 и 6–7:

1. Изменение RLO с 0 на 1 на входе запуска запускает таймер. После этого запрограммированное время t отсчитывается независимо от изменения RLO с 1 на 0 на входе запуска.
2. Если RLO на входе запуска меняется с 0 на 1 до истечения времени, то таймер перезапускается с первоначально запрограммированным временем.
3. Опрос состояния сигнала на выходе таймера дает результат 1 в течение всего времени работы таймера.
4. Если на входе сброса RLO становится равным 1, то таймер сбрасывается. Пока состояние сигнала на входе запуска остается равным 1, смена RLO на входе сброса с 1 на 0 не оказывает влияния на таймер
5. Изменение RLO с 0 на 1 на входе запуска при наличии сигнала на входе сброса приводит к кратковременному запуску с немедленным сбросом из-за следующего непосредственно далее в программе оператора сброса (на рис. 6–6 показано в виде импульсной линии на временной диаграмме). Для этого импульса не может быть получен результат опроса, если соблюдается описанная выше последовательность записи операторов.
6. Изменение RLO с 0 на 1 на входе разблокировки во время работы таймера перезапускает таймер. Запрограммированное время при перезапуске используется как текущее время. Изменение RLO на входе разблокировки с 1 на 0 воздействия не оказывает.
7. Если RLO меняется с 0 на 1 на входе разблокировки, когда таймер не работает, а на входе запуска RLO все еще равен 1, то таймер также будет запущен как формирователь удлиненного импульса с запрограммированным временем.
8. Изменение RLO с 0 на 1 на входе разблокировки, когда RLO на входе запуска еще равен 0, не оказывает воздействия на таймер.

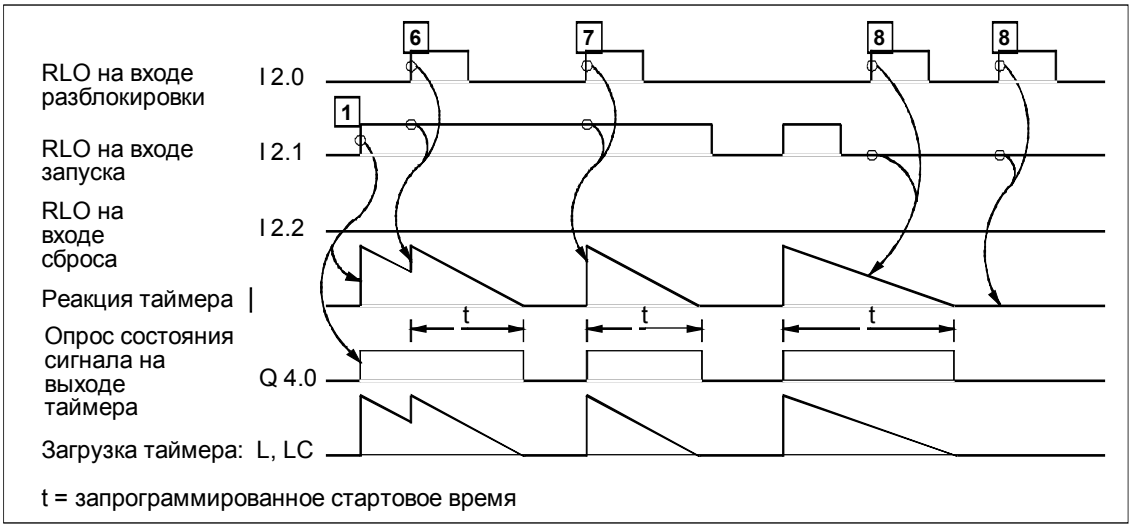


Рис. 6–7. Пример таймера как формирователя удлиненного импульса, часть 2

Таймер как формирователь задержки включения: SD

Рисунки 6–8 и 6–9 дают примеры таймера как формирователя задержки включения. На рисунках числа в прямоугольниках указывают на объяснения, следующие за рис. 6–8. Рисунки относятся к следующей программе на AWL:

AWL		Объяснение
A	**I 2.0**	
FR	**T 1**	**Разблокировать таймер T1.**
A	**I 2.1**	
L	**S5T#0H2M23S0MS**	
SD	**T 1**	**Запустить таймер T1 как формирователь задержки включения.**
A	**I 2.2**	
R	**T 1**	**Сбросить таймер T1.**
A	**T 1**	**Опросить состояние сигнала таймера T1.**
=	**Q 4.0**	
L	**T 1**	
T	**MW10**	
LC	**T 1**	**Загрузить таймер T1 в аккумулятор.**
T	**MW12**	

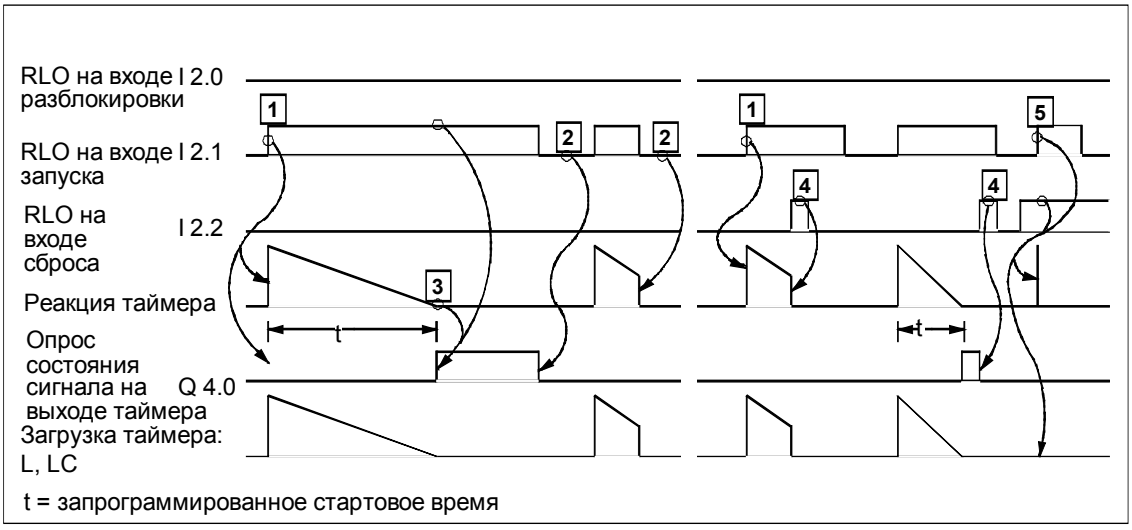


Рис. 6–8. Пример таймера как формирователя задержки включения, часть 1

Следующий список описывает элементы рисунков 6–8 и 6–9:

1. Изменение RLO с 0 на 1 на входе запуска запускает таймер. После этого идет отсчет запрограммированного времени t.
2. Когда на входе запуска RLO становится равным 0, таймер сбрасывается.
3. Опрос состояния сигнала на выходе таймера Q 4.0 дает результат 1, когда время истекло, а сигнал на входе запуска равен 1.
4. Если на входе сброса RLO становится равным 1, то таймер сбрасывается. Пока состояние сигнала на входе запуска остается равным 1, смена RLO на входе сброса с 1 на 0 не оказывает влияния на таймер.
5. Изменение RLO с 0 на 1 на входе запуска при наличии сигнала на входе сброса приводит к кратковременному запуску с немедленным сбросом из-за следующего непосредственно далее в программе оператора сброса (на рис. 6–8 показано в виде импульсной линии на временной диаграмме). Для этого импульса не может быть получен результат опроса, если соблюдается описанная выше последовательность записи операторов.
6. Изменение RLO с 0 на 1 на входе разблокировки во время работы таймера перезапускает таймер. Запрограммированное время при перезапуске используется как текущее время. Изменение RLO на входе разблокировки с 1 на 0 воздействия не оказывает.
7. Если RLO меняется с 0 на 1 на входе разблокировки после завершения нормальной работы таймера, то это не оказывает воздействия на таймер.
8. Изменение RLO с 0 на 1 на входе разблокировки после сброса таймера, когда RLO на входе запуска еще равен 1, запускает таймер. Запрограммированное время используется в качестве текущего времени.

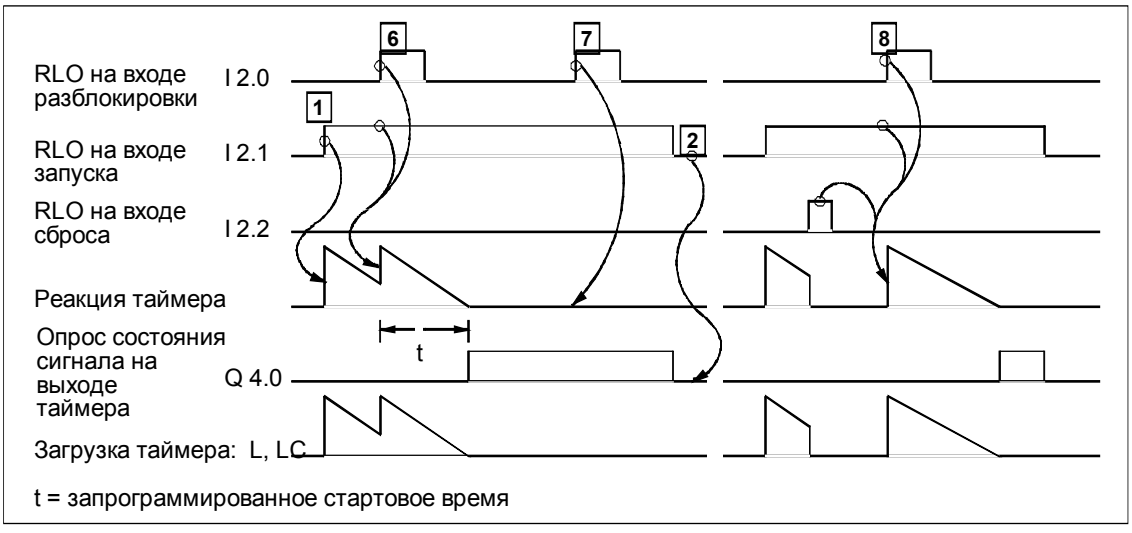


Рис. 6–9. Пример таймера как формирователя задержки включения, часть 2

Таймер как формирователь задержки включения с запоминанием: SS

Рисунки 6–10 и 6–11 дают примеры таймера как формирователя задержки включения с запоминанием. На рисунках числа в прямоугольниках указывают на объяснения, следующие за рис. 6–10. Рисунки относятся к следующей программе на AWL:

AWL		Объяснение
A	I 2.0	
FR	T 1	**Разблокировать таймер T1.**
A	I 2.1	
L	S5T#0H2M23S0MS	
SS	T 1	**Запустить таймер T1 как формирователь задержки включения с**
A	I 2.2	**запоминанием.**
R	T 1	**Сбросить таймер T1.**
A	T 1	**Опросить состояние сигнала таймера T1.**
=	Q 4.0	
L	T 1	
T	MW10	
LC	T 1	**Загрузить таймер T1 в аккумулятор.**
T	MW12	

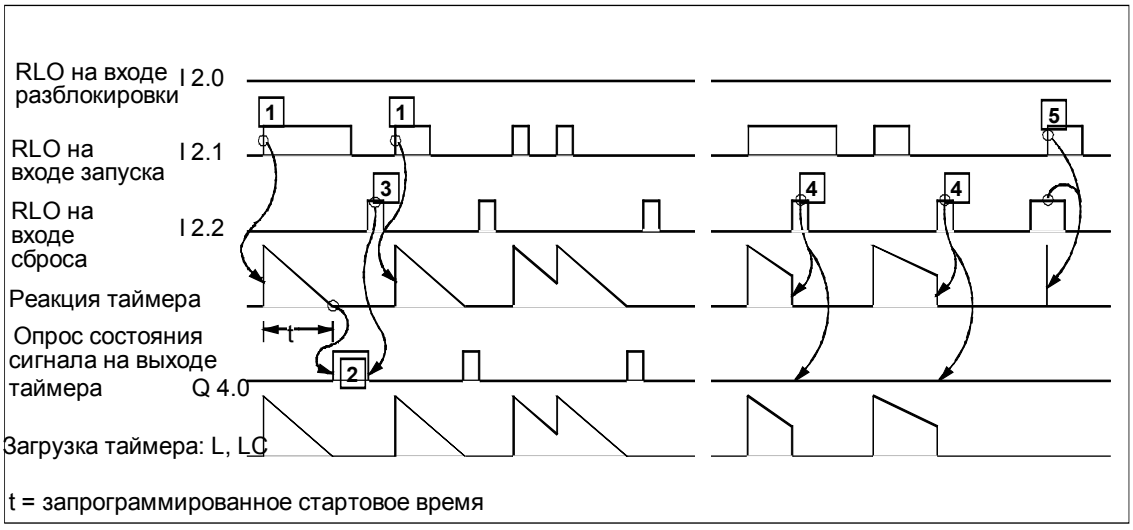


Рис. 6–10. Пример таймера как формирователя задержки включения с запоминанием, часть 1

Следующий список описывает элементы рисунков 6–10 и 6–11:

1. Изменение RLO с 0 на 1 на входе запуска запускает таймер. После этого запрограммированное время t отсчитывается независимо от изменения RLO с 1 на 0 на входе запуска.
2. Опрос состояния сигнала на выходе таймера дает результат 1, когда время истекло.
3. Результат опроса состояния сигнала на выходе Q 4.0 меняется на 0 только тогда, когда RLO на входе сброса равен 1.
4. Если на входе сброса RLO становится равным 1, то таймер сбрасывается. Пока состояние сигнала на входе запуска остается равным 1, смена RLO на входе сброса с 1 на 0 не оказывает влияния на таймер.

5 Изменение RLO с 0 на 1 на входе запуска при наличии сигнала на входе сброса приводит к кратковременному запуску с немедленным сбросом из-за следующего непосредственно далее в программе оператора сброса (на рис. 6–10 показано в виде импульсной линии на временной диаграмме). Для этого импульса не может быть получен результат опроса, если соблюдается описанная выше последовательность записи операторов.

6 Когда RLO на входе разблокировки меняется с 0 на 1 во время работы таймера, а RLO на входе запуска таймера равен 1, таймер перезапускается. Запрограммированное время при перезапуске используется как текущее время. Изменение RLO на входе разблокировки с 1 на 0 воздействия на таймер не оказывает.

7 Если RLO меняется с 0 на 1 на входе разблокировки после завершения нормальной работы таймера, то это не оказывает воздействия на таймер.

8 Если RLO меняется с 0 на 1 на входе разблокировки, когда таймер работает, а RLO на входе запуска таймера равен 0, то это не оказывает воздействия на таймер.

9 Если RLO на входе разблокировки меняется с 0 на 1, когда таймер сброшен, а RLO на входе запуска еще равен 1, то таймер перезапускается. Запрограммированное время используется в качестве текущего времени для перезапуска.

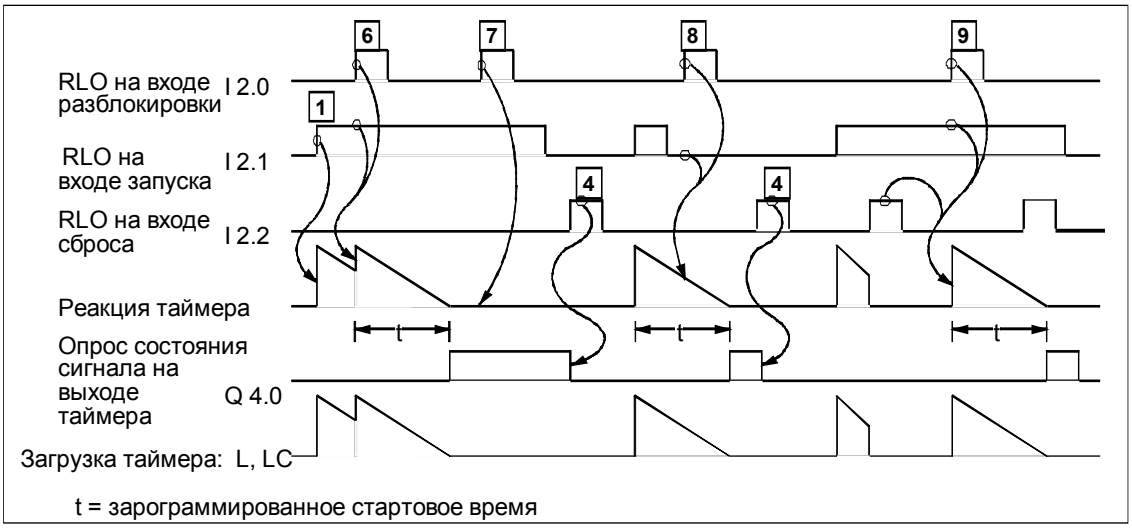


Рис. 6–11. Пример таймера как формирователя задержки включения с запоминанием, часть 2

Таймер как формирователь задержки выключения: SF

Рисунки 6–12 и 6–13 дают примеры таймера как формирователя задержки выключения. На рисунках числа в прямоугольниках указывают на объяснения, следующие за рис. 6–12. Рисунки относятся к следующей программе на AWL:

AWL		Объяснение
A	**I 2.0**	
FR	**T1**	**Разблокировать таймер T1.**
A	**I 2.1**	
L	**S5T#00H02M23S00MS**	
SF	**T1**	**Запустить таймер T1 в качестве формирователя задержки**
A	**I 2.2**	**выключения.**
R	**T1**	**Сбросить таймер T1.**
A	**T1**	**Опросить состояние сигнала таймера T1.**
=	**Q 4.0**	
L	**T1**	
T	**MW10**	
LC	**T1**	**Загрузить таймер T1 в аккумулятор.**
T	**MW12**	

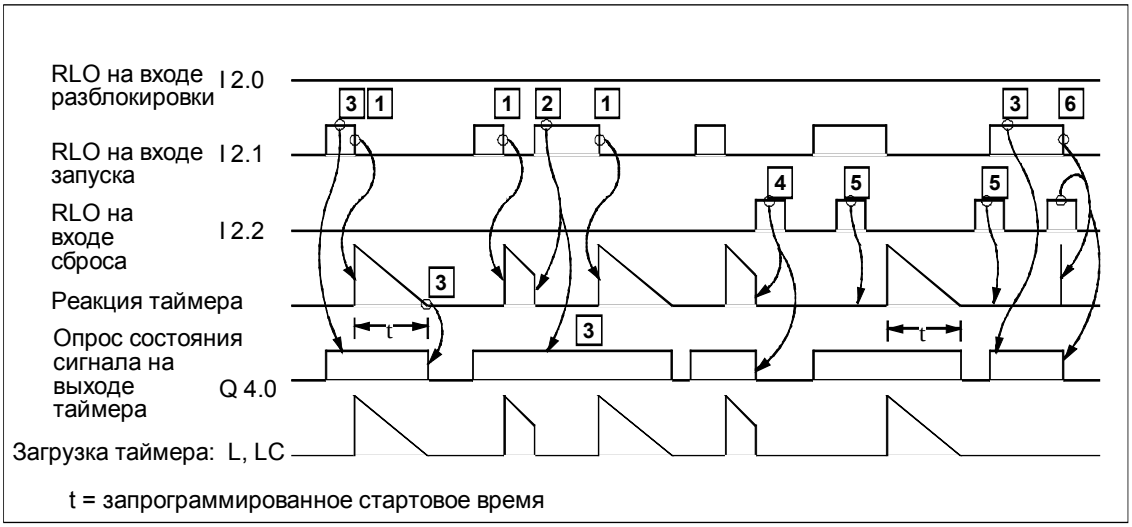


Рис. 6–12. Пример таймера как формирователя задержки выключения, часть 1

Следующий список описывает элементы рисунков 6–12 и 6–13:

1 Изменение RLO с 0 на 1 на входе запуска вызывает изменение с 0 на 1 сигнала на выходе Q 4.0 таймера. Изменение RLO на входе запуска с 1 на 0 запускает таймер. С этого момента ведется отсчет запрограммированного времени t.

2 Если на входе запуска снова появляется RLO, равный 1, то таймер сбрасывается.

3 Опрос состояния сигнала на выходе Q 4.0 таймера дает результат 1, когда RLO на входе запуска равен 1, а время еще не истекло.

4 Если на входе сброса RLO становится равным 1, то таймер сбрасывается. Тогда опрос состояния сигнала таймера дает результат, равный 0. Изменение RLO на входе сброса с 1 на 0 не оказывает воздействия на таймер.

5 Если на входе сброса появляется 1, когда таймер не работает, то это не оказывает никакого воздействия на таймер.

6 Изменение RLO с 1 на 0 на входе запуска при наличии сигнала на входе сброса приводит к кратковременному запуску с немедленным сбросом из-за следующего непосредственно далее в программе оператора сброса (на рис. 6–10 показано в виде импульсной линии на временной диаграмме). Затем опрос состояния сигнала таймера дает результат 0.

7 Если RLO меняется с 0 на 1 на входе разблокировки, когда таймер не работает, то это не оказывает воздействия на таймер. Изменение RLO с 1 на 0 тоже не оказывает влияния на таймер.

8 Изменение RLO с 0 на 1 на входе разблокировки во время работы таймера приводит к перезапуску таймера. Запрограммированное время используется при перезапуске в качестве текущего времени.

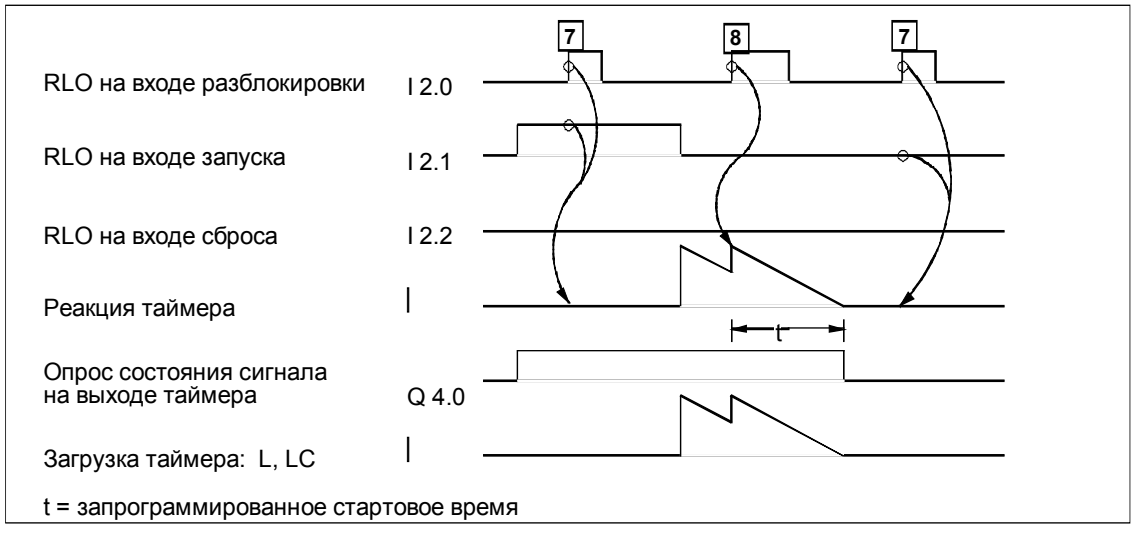


Рис. 6–13. Пример таймера как формирователя задержки выключения, часть 2

6.5 Адреса и диапазоны для таймерных команд

Таблицы 6–3 и 6–4 показывают виды адресации, операнды и диапазоны адресов для таймерных команд.

Таблица 6–3. Операнды, области и виды адресации таймерных команд

Диапазон адресов в зависимости от вида адресации		
прямая	**косвенная через память**	
от 0 до 255	[DBW] [DIW] [LW] [MW]	от 0 до 65 534

Таблица 6–4. Операнд таймера, передаваемый как параметр

Операнд	**Формат адресного параметра**
Символическое имя	Время, передаваемое как параметр.

6.6 Выбор подходящего таймера

Рис. 6–14 дает обзор пяти разных таймеров, описанных в разделе 6.4. Этот обзор должен помочь вам выбрать таймер, адекватный вашим целям.

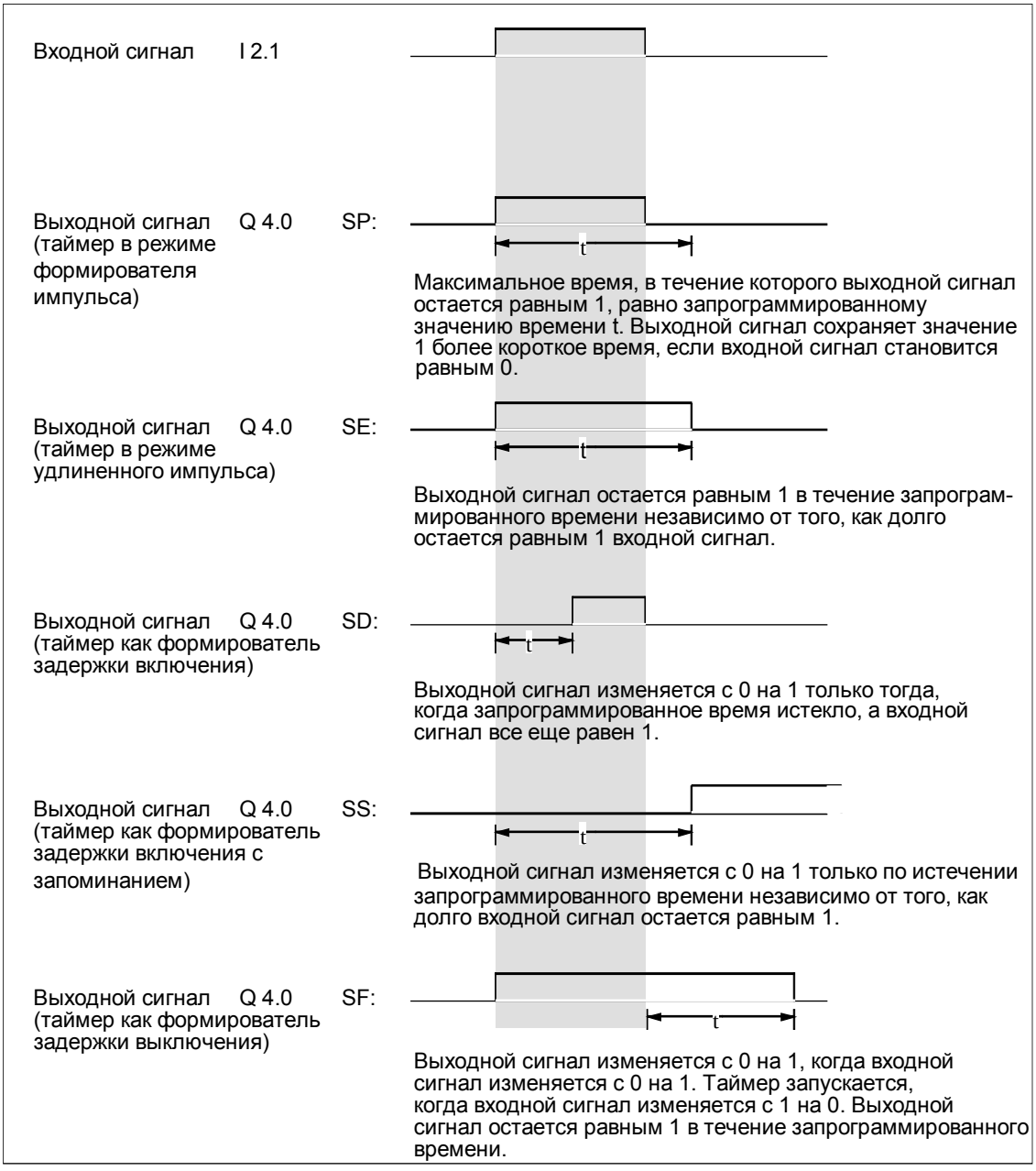


Рис. 6–14. Выбор подходящего таймера

7 Операции со счетчиками

Обзор главы

7.1 Обзор

Определение

Счетчик – это функциональный элемент языка программирования STEP 7, предназначенный для счета.

Счетчики имеют зарезервированную область памяти в вашем CPU. Эта область памяти резервирует одно 16-битовое слово для каждого счетчика. Набор команд списка операторов поддерживает 256 счетчиков. Чтобы выяснить количество счетчиков, доступных в вашем CPU, обратитесь к техническим данным CPU.

Операции со счетчиками являются единственными функциями, имеющими доступ к области памяти, зарезервированной для счетчиков.

Имеющиеся в распоряжении команды

Представление языка программирования STEP 7 в виде списка операторов предоставляет в распоряжение следующие команды:

- Установка (S)
- Сброс (R)
- Прямой счет (CU)
- Обратный счет (CD)
- Разблокировка счетчика (FR)
- Загрузка счетчика в одном из следующих форматов:
  - двоичный (L)
  - двоично-десятичный (LC)
- Опрос состояния сигнала счетчика и логическое сопряжение результата опроса с помощью булевой логической операции (A, AN, O, ON, X, XN). Опрос состояния сигнала с помощью команды A, O или X дает результат, равный «1», когда значение счетчика больше нуля. Опрос состояния сигнала с помощью команды A, O или X дает результат, равный «0», когда значение счетчика равно нулю.

Рис. 7–1 дает обзор команд, использующих слово счетчика в качестве операнда.

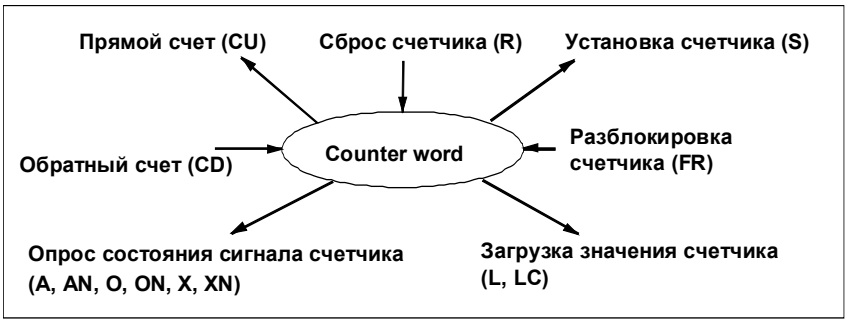


Рис. 7–1. Команды, которые могут использовать слово счетчика в качестве операнда

7.2 Установка, сброс и разблокировка счетчика

Описание

Для установки счетчика в вашей программе на AWL включите в нее три оператора для реализации следующих операций:

- Опрос состояния сигнала на равенство 0 или 1 (например, A I 2.3)
- Загрузка значения счетчика (например, L C# 3) в младшее слово аккумулятора 1
- Установка счетчика со значением, которое вы загрузили (например, S C 1). Эта операция пересылает значение счетчика из аккумулятора 1 в слово счетчика.

В вашей программе на AWL изменение результата логической операции (RLO) с 0 на 1 перед командой *Установить* (S) устанавливает счетчик на запрограммированное счетное значение. Запрограммированное счетное значение и команда установки должны следовать сразу за логической операцией, определяющей условия установки счетчика.

Начальное значение

Вы устанавливаете счетчик на определенное значение, загрузкой этого значения в младшее слово аккумулятора 1 и, немедленно после этого, установкой счетчика. Когда вы устанавливаете в своей программе счетчик, CPU ищет в аккумуляторе 1 счетное значение. Затем CPU передает это счетное значение из аккумулятора 1 в слово счетчика, которое вы указали в операторе установки (например, S C 1). Диапазон счетных значений лежит между 0 и 999.

Пример установки счетчика

Рис. 7–2 дает пример установки счетчика. Изменение состояния сигнала с 0 на 1 на входе I 2.3 устанавливает счетчик. Рис. 7–2 относится к следующей программе:

AWL	Объяснение
A I 2.3	**Опросить состояние сигнала на входе I 2.3.**
L C# 3	**Если состояние сигнала равно 1, загрузить счетное значение 3 в аккумулятор 1.**
S C 1	**Установить счетчик C 1 на счетное значение 3. Эта операция пересылает счетное значение 3 из аккумулятора в слово счетчика 1.**

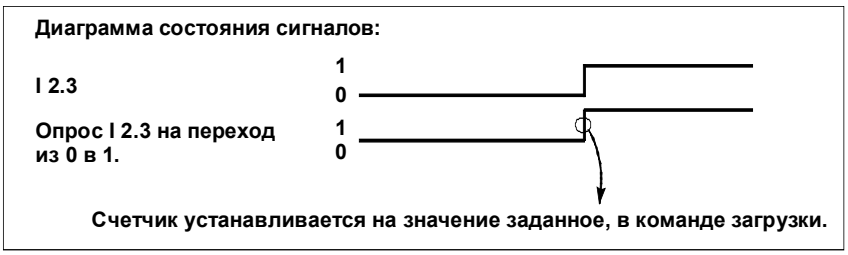


Рис. 7–2. Установка счетчика

Сброс счетчика

Счетчик сбрасывается с помощью команды *Сбросить* (R). CPU сбрасывает счетчик, когда результат логической операции равен 1 непосредственно перед командой R в вашей программе. Пока RLO перед оператором R равен 1, команда A, O или X, опрашивающая состояние сигнала счетчика, дает результат, равный 0, а команда AN, ON или XN – результат, равный 1.

Когда ваша программа сбрасывает счетчик, она очищает его, т.е. сбрасывает его значение в 0.

Если счетчик должен сбрасываться статическим сигналом на входе сброс (R) и независимо от RLO других команд счетчика, вы должны записать оператор сброса непосредственно после оператора установки, прямого счета или обратного счета (см. раздел 7.3) и перед опросом сигнала или операцией загрузки.

При программировании счетчиков следует придерживаться следующей последовательности (см. также пример программирования в разделе 7.6):

1. Прямой счет
2. Обратный счет
3. Установка счетчика
4. Сброс счетчика
5. Опрос состояния сигнала счетчика
6. Загрузка значения счетчика (чтение значения счетчика)

Разблокировка счетчика для повторного пуска

Изменение с 0 на 1 результата логической операции перед командой *Разблокировать* (FR) разблокирует счетчик. CPU выполняет команду FR только при положительном фронте сигнала.

Разблокировка счетчика не требуется ни для установки счетчика, ни для нормального счета. Разблокировка используется только для того, чтобы устанавливать счетчик или производить прямой или обратный счет, если перед соответствующим оператором счета требуется положительный фронт сигнала (переход из 0 в 1) и если бит RLO перед соответствующим оператором имеет состояние сигнала 1.

7.3 Прямой и обратный счет

Описание прямого счета

В вашей программе на AWL изменение результата логической операции с 0 на 1 перед командой прямого счета (CU) увеличивает счетчик. Каждый раз, когда непосредственно перед командой прямого счета появляется положительный фронт RLO, счетчик увеличивается на одну единицу.

Когда счет достигает своего верхнего предела, равного 999, увеличение прекращается, и дальнейшее изменение состояния сигнала на входе прямого счета никакого влияния не оказывает. Меры против переполнения (OV) не предусмотрены.

AWL	Объяснение
A I 0.1 **CU C1**	**Когда на входе I 0.1 появляется положительный фронт, значение счетчика C 1 увеличивается на 1.**

Описание обратного счета

В вашей программе на AWL изменение результата логической операции с 0 на 1 перед командой обратного счета (CD) уменьшает счетчик. Каждый раз, когда непосредственно перед командой обратного счета появляется положительный фронт RLO, счетчик уменьшается на одну единицу.

Когда счет достигает своего нижнего предела, равного 0, уменьшение прекращается, и дальнейшее изменение состояния сигнала на входе обратного счета никакого влияния не оказывает. Счетчик не ведет счета с отрицательными значениями.

AWL	Объяснение
A I 0.2 **CD C1**	**Если на входе I 0.2 появляется положительный фронт, счетчик C 1 уменьшается на 1.**

7.4 Загрузка значения счетчика в виде целого числа

Описание

Значение счетчика хранится в слове счетчика в двоичном коде. Вы можете использовать следующую команду для считывания двоичного значения счетчика из слова счетчика и загрузки его в младшее слово аккумулятора 1:

L <слово счетчика>

Такой способ загрузки называют непосредственной загрузкой значения счетчика.

AWL	Объяснение
L C 1	**Загрузить младшее слово аккумулятора 1 счетным значением счетчика C 1 в двоичном формате.**

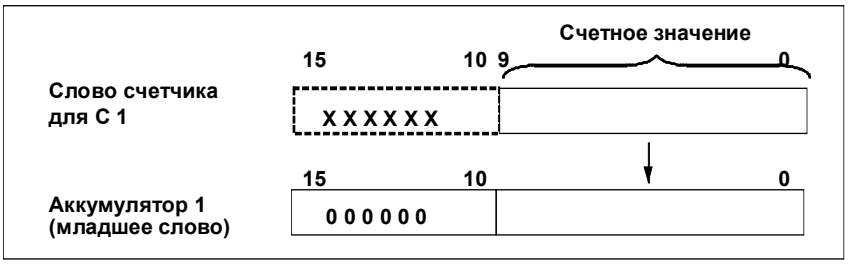


Рис. 7–3. Загрузка счетного значения в аккумулятор 1 с помощью команды L

Вы можете использовать значение, содержащееся в аккумуляторе 1 в качестве результата операции загрузки L для дальнейшей обработки. Но вы не можете передать значение из аккумулятора в слово счетчика. Если вы хотите запустить счетчик с заданным счетным значением, вам нужно использовать соответствующий оператор установки счетчика.

7.5 Загрузка значения счетчика в двоично-десятичном формате

Описание

Значение счетчика хранится в слове счетчика в двоичном коде. Вы можете использовать следующую команду для считывания значения счетчика в двоично-десятичном формате (BCD) и загрузки его в младшее слово аккумулятора 1:

LC <слово счетчика>

Такой способ загрузки называют загрузкой значения счетчика в формате BCD.

Значение, которое содержится в младшем слове аккумулятора 1 в качестве результата операции LC, имеет тот же формат, который необходим для установки счетчика.

AWL	Объяснение
LC C 1	**Загрузить младшее слово аккумулятора 1 счетным значением счетчика C 1 в двоично-десятичном формате.**

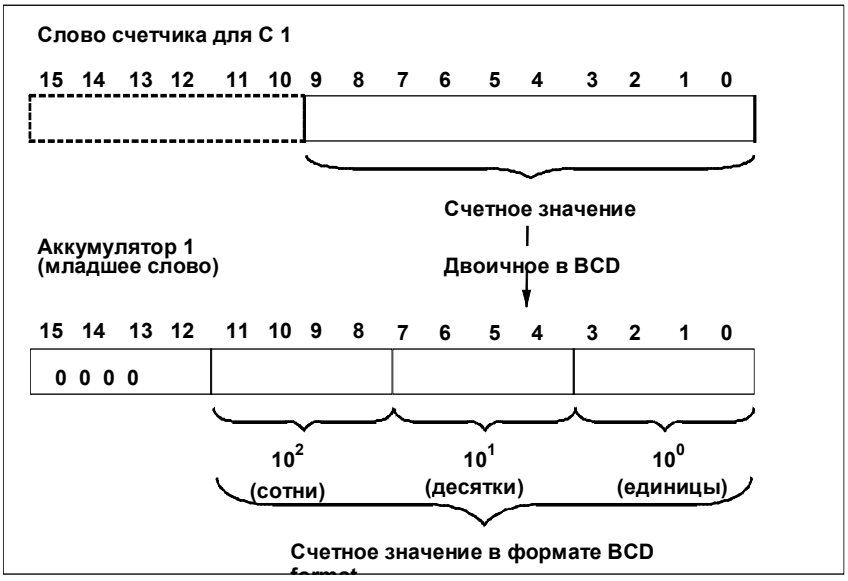


Рис. 7–4. Загрузка счетного значения в аккумулятор 1 с помощью команды LC

Значение, которое содержится в аккумуляторе в качестве результата операции LC, может использоваться для дальнейшей обработки, например, для передачи значения на выходы для отображения.

7.6 Пример счетчика

Рис. 7–5 дает пример прямого счета, обратного счета, установки и сброса счетчика, опроса состояния сигнала счетчика и загрузки значения счетчика. Пример придерживается последовательности программирования, рекомендованной в разделе 7.2. На рисунке числа в прямоугольниках указывают на объяснения, следующие за рисунком. Рисунок 7–5 относится к программе на AWL, следующей за списком с пояснениями.

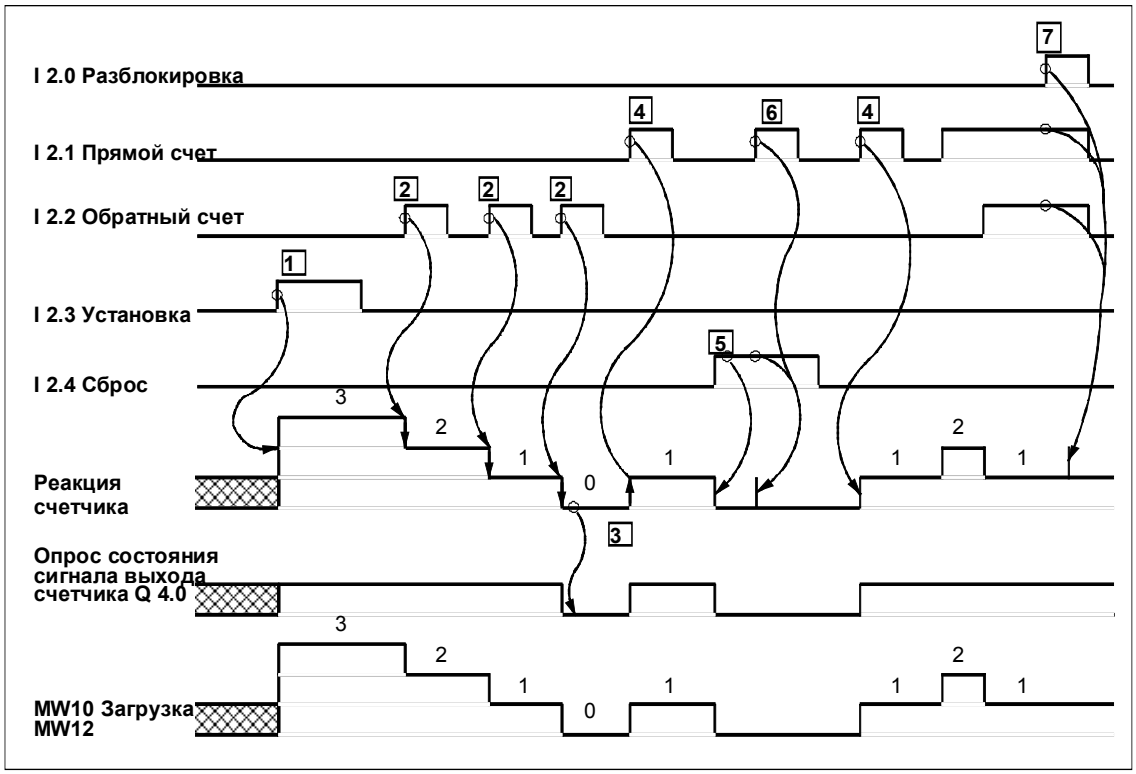


Рис. 7–5. Пример команд счетчика

Следующий список описывает элементы рисунка 7–5:

1 Изменение RLO с 0 на 1 на входе установки устанавливает счетчик на счетное значение 3. Переход с 1 на 0 на входе установки воздействия на счетчик не оказывает.

2 Изменение RLO с 0 на 1 на входе обратного счета уменьшает счетчик на единицу. Переход с 1 на 0 на входе обратного счета воздействия на счетчик не оказывает.

3 Результат опроса состояния сигнала оператором A C 1 равен 0, когда счетное значение равно 0.

4 Изменение RLO с 0 на 1 на входе прямого счета увеличивает счетчик на единицу. Переход с 1 на 0 на входе прямого счета воздействия на счетчик не оказывает.

5 Если RLO на входе сброса равен 1, счетчик сбрасывается. Опрос состояния сигнала дает 0. Изменение RLO с 1 на 0 на входе сброса воздействия на счетчик не оказывает.

6 Изменение RLO с 0 на 1 на входе прямого счета при наличии сигнала сброса вызывает кратковременное увеличение счетчика, после чего он немедленно сбрасывается непосредственно следующим далее в программе оператором сброса. (На рис. 7–5 это кратковременное увеличение показано импульсной линией на временной диаграмме). После этого опрос состояния сигнала дает результат, равный 0.

7 Изменение RLO с 0 на 1 на входе разблокировки при наличии сигналов прямого и обратного счета заставляет счетчик кратковременно увеличиться, а затем немедленно уменьшиться из-за непосредственно следующего в программе оператора обратного счета. (На рис. 7–5 это кратковременное увеличение показано импульсной линией на временной диаграмме). Переход с 1 на 0 на входе разблокировки воздействия на счетчик не оказывает.

AWL		Объяснение
A	I 2.0	
FR	C 1	Разблокировать счетчик C 1.
A	I 2.1	
CU	C 1	Прямой счет (увеличение на 1).
A	I 2.2	
CD	C 1	Обратный счет (уменьшение на 1).
A	I 2.3	
L	C# 3	
S	C 1	Установить счетчик C 1.
A	I 2.4	
R	C 1	Сбросить счетчик C 1.
A	C 1	
=	Q 4.0	Опросить состояние сигнала счетчика C 1.
L	C 1	Загрузить счетчик C 1 (в двоичном коде)
T	MW10	
LC	C 1	
T	MW12	Загрузить счетчик C 1 (в BCD–коде).

7.7 Адреса и диапазоны для операций со счетчиками

Таблица 7–1 показывает виды адресации, адреса и диапазоны адресов для операций со счетчиками.

Таблица 7–1. Адреса, диапазоны и виды адресации для операций со счетчиками

Диапазон адресов в зависимости от вида адресации		
прямая	**косвенная через память**	
от 0 до 65 535	[DBW] [DIW] [LW] [MW]	от 0 до 65 534

8 Команды загрузки и передачи

Обзор главы

8.1 Обзор

Определение

Команды загрузки (L) и передачи (T) позволяют программировать обмен информацией между модулями ввода или вывода и областями памяти или между областями памяти. CPU выполняет эти команды в каждом цикле как безусловные команды, т.е. результат логической операции на них не влияет.

Обмен информацией

Команды L и T позволяют производить обмен информацией между следующими модулями и областями памяти:

- Модули ввода и вывода и следующие области памяти:
  - таблицы входов и выходов образа процесса
  - битовая память (меркеры)
  - таймеры и счетчики
  - области данных
- Таблицы входов и выходов образа процесса и следующие области памяти:
  - битовая память (меркеры)
  - таймеры и счетчики
  - области данных
- Таймеры и счетчики и следующие области памяти:
  - таблицы входов и выходов образа процесса
  - битовая память (меркеры)
  - области данных

Способ обмена

Команды L и T производят обмен информацией через аккумулятор. Команда L записывает (загружает) содержимое своего исходного адреса в аккумулятор 1, сдвигая всю уже содержащуюся там информацию в аккумулятор 2. Старое содержимое аккумулятора 2 заменяется. Команда T копирует содержимое аккумулятора 1 и записывает его в соответствующую целевую память. Так как команда T только копирует информацию, находящуюся в аккумуляторе 1, то эта информация остается доступной для других команд.

Команды L и T могут обрабатывать информацию байтами (8 битов), словами (16 битов) и двойными словами (32 бита).

Аккумулятор содержит 32 бита. Данные длиной менее 32 битов располагаются в аккумуляторе справа. Остальные биты аккумулятора заполняются нулями.

8.2 Загрузка и передача

Описание

Вы можете использовать команду загрузки (L) или передачи (T) для передачи информации в аккумулятор 1 или из него порциями следующих размеров:

- байт (B, 8 битов)
- слово (W, 16 битов)
- двойное слово (D, 32 бита)

Байт загружается в младший байт младшего слова аккумулятора 1. Слово загружается в младшее слово аккумулятора 1. Неиспользуемые байты при загрузке в аккумулятор сбрасываются в ноль.

Непосредственная адресация

Команда L может обращаться к константам в 8, 16 и 32 бита, а также к символам ASCII. Этот вид адресации называется непосредственной адресацией (см. раздел 3.1 и таблицу 8–1).

Таблица 8–1. Операнды команд загрузки: непосредственная адресация

Операнд	Пример	Пояснение
±..	L +5	Загружает 16-битовую целую константу в аккумулятор 1.
B#(..,..)	L B#(1,10)	Загружает константу как 2 байта в аккумулятор 1. (В этом примере 10 поступает в младший байт младшего слова аккумулятора 1; 1 поступает в старший байт младшего слова аккумулятора 1, см. рис. 2–4.)
	L B#(1,10,5,4)	Загружает константу как 4 байта в аккумулятор 1. (В этом примере 4 и 5 поступают соответственно в младший и старший байт младшего слова аккумулятора 1; 10 и 1 поступают соответственно в младший и старший байт старшего слова аккумулятора, см. рис. 2–4.)
L#..	L L#+5	Загружает 32–битовую целую константу в аккумулятор 1.
16#..	L B#16#EF	Загружает 8–битовую шестнадцтеричную константу в аккумулятор 1.
	L W#16#FAFB	Загружает 16–битовую шестнадцтеричную константу в аккумулятор 1.
	L DW#16#1FFE_1ABC	Загружает 32–битовую шестнадцтеричную константу в аккумулятор 1.
2#..	L 2#1111_0000_1111_0000	Загружает 16–битовую двоичную константу в аккумулятор 1.
	L 2#1111_0000_1111_0000_ 1111_0000_1111_0000	Загружает 32–битовую двоичную константу в аккумулятор 1.
'..'	L 'AB' L 'ABCD'	Загружает 2 символа в аккумулятор 1. Загружает 4 символа в аккумулятор 1.
C#..	L C#1000	Загружает 16–битовую константу счетчика в аккумулятор 1.
S5TIME#..	L S5TIME#2S	Загружает 16–битовую константу S5TIME в аккумулятор 1.
..	L 1.0E+5	Загружает 32–битовое число с плавающей точкой в формате IEEE в аккумулятор 1.

Таблица 8–1. Операнды команд загрузки: непосредственная адресация		
Операнд	**Пример**	**Пояснение**
P#..	L P#I1.0 L P##Start L P#ANNA	Загружает 32–битовый указатель в аккумулятор 1. Загружает 32–битовый указатель на локальную переменную (Start) в аккумулятор 1 Загружает указатель на указанный параметр в аккумулятор 1. (Эта команда загружает относительное смещение адреса указанного параметра. Для определения абсолютного смещения в экземплярном блоке данных функционального блока с мультиэкземплярами к этому значению должно быть добавлено содержимое регистра AR2).
D#..	L D#1994–3–15	Загружает 16–битовую дату в аккумулятор 1.
T#..	L T#0D_1H_1M_0S_0MS	Загружает 32–битовое значение времени в аккумулятор 1.
TOD#..	L TOD#1:10:3.3	Загружает 32–битовое значение времени суток в аккумулятор 1.

Прямая и косвенная адресация

Команды L и T могут обращаться к байту (B), слову (W) или двойному слову (D) в следующих областях памяти с помощью прямой и косвенной адресации (см. также разделы 3.2, 3.3 и 3.5):

- Вход и выход образа процесса (идентификаторы операндов IB, IW, I, QB, QW, QD)
- Внешние входы и выходы (идентификаторы операндов PIB, PIW, PID, PQB, PQW, PQD). Внешние входы могут быть операндами только команд L; внешние выходы могут быть операндами только команд T.
- Битовая память (идентификаторы операндов MB, MW, MD)
- Блок данных (идентификаторы операндов DBB, DBW, DBD, DIB, DIW, DID)
- Локальные данные (временные локальные данные, идентификаторы операндов LB, LW, LD)

В таблице 8–2 перечислены операнды команд L и T, использующих прямую и косвенную адресацию.

Таблица 8–2. Операнды команд L и T: прямая и косвенная адресация			
Идентификатор операнда	**Максимальный диапазон адресов в соответствии с видом адресации**		
	прямая	**косвенная через память**	**косвенная внутри области через регистр**
IB IW ID QB QW QD	от 0 до 65 535 от 0 до 65 534 от 0 до 65 532 от 0 до 65 535 от 0 до 65 534 от 0 до 65 532	[DBD] от 0 до [DID] 65 532 [LD] [MD]	[AR1, P#байт.бит] от 0 до 8 191 [AR2, P#байт.бит]
PIB PIW PID (только L)	от 0 до 65 535 от 0 до 65 534 от 0 до 65 532	[DBD] от 0 до [DID] 65 532 [LD] [MD]	[AR1, P#байт.бит] от 0 до 8 191 [AR2, P#байт.бит]

Таблица 8–2. Операнды команд L и T: прямая и косвенная адресация			
Идентификатор операнда	**Максимальный диапазон адресов в соответствии с видом адресации**		
	прямая	**косвенная через память**	**косвенная внутри области через регистр**
PQB PQW PQD (только T)	от 0 до 65 535 от 0 до 65 534 от 0 до 65 532		
MB MW MD	от 0 до 255 от 0 до 254 от 0 до 252	[DBD] от 0 до [DID] 65 532 [LD] [MD]	[AR1, P#байт.бит] от 0 до 8 191 [AR2, P#байт.бит]
DBB DBW DBD DIB DIW DID LB LW LD	от 0 до 65 535 от 0 до 65 534 от 0 до 65 532 от 0 до 65 535 от 0 до 65 534 от 0 до 65 532 от 0 до 65 535 от 0 до 65 534 от 0 до 65 532	[DBD] от 0 до [DID] 65 532 [LD] [MD]	[AR1, P#байт.бит] от 0 до 8 191 [AR2, P#байт.бит]

Косвенная адресация с указанием области памяти

Команды L и T могут обращаться к байту (B), слову (W) или двойному слову (D) с помощью косвенной адресации с указанием области памяти через регистр (см. раздел 3.6).

Таблица 8–3. Операнды команд L и T: косвенная адресация с указанием области памяти через регистр	
Идентификатор операнда[1]	**Диапазон адресов**
B (байт), W (слово), D (двойное слово)	[AR1, P#байт.бит] от 0 до 8 191 [AR2, P#байт.бит]

[1] Эта область памяти закодирована в битах с 24 по 31 адресного регистра AR 1 или AR2 (см. раздел 3.6).

Байт, слово или двойное слово как параметр

Команды L и T могут также использовать в качестве операндов байт, слово или двойное слово, которые передаются в качестве параметра.

Таблица 8–4. Операнд команд L и T: байт, слово или двойное слово, передаваемые в качестве параметра	
Операнд	**Формат адресного параметра**
Символическое имя	Байт, слово или двойное слово, передаваемые в качестве параметра

8.3 Чтение слова состояния или передача в слово состояния

Загрузка слова состояния

Вы можете использовать команду загрузки (L) для загрузки битов с 0 по 8 слова состояния (см. рис. 8–1) в аккумулятор 1. Биты с 9 по 31 аккумулятора 1 сбрасываются в 0. Эта команда показана в примере, следующем за рис. 8–1.

Замечание

В CPU семейства S7–300 оператор L STW не загружает биты слова состояния $\overline{FC}$, STA OR. В соответствующие позиции младшего слова аккумулятора 1 загружаются только биты 1, 4, 5, 6, 7 и 8.

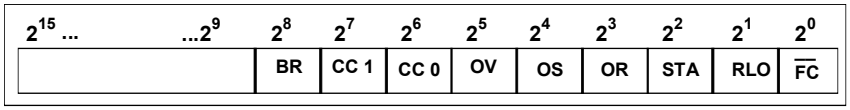


Рис. 8–1. Структура слова состояния

AWL	Объяснение
L STW	Загружает биты с 0 по 8 слова состояния в младшее слово аккумулятора 1.

Передача в слово состояния

Вы можете использовать команду передачи (T) для передачи содержимого аккумулятора 1 в слово состояния (см. рис. 8–1). Эта команда показана в следующем фрагменте программы.

AWL	Объяснение
T STW	Передает содержимое аккумулятора 1 в слово состояния.

8.4 Загрузка значений времени и счетчиков как целых чисел

Загрузка времени

Значение времени хранится в слове таймера в двоичном коде. Вы можете использовать следующую команду загрузки (L) для считывания двоичного значения времени из слова таймера и загрузки его в младшее слово аккумулятора 1:

L <слово таймера>

Этот вид загрузки называют прямой загрузкой значения времени.

Значение времени в слове таймера при обработке программы пользователя в CPU уменьшается с начального значения до 0. Когда вы используете команду L со словом таймера в качестве операнда, вы получаете значение, находящееся между стартовым временем слова таймера и 0. Время, прошедшее с момента запуска таймера, вычисляется как разность между стартовым временем и временем, считанным в данный момент.

AWL	Объяснение
L T 1	**Загрузить младшее слово аккумулятора 1 временем из таймера T 1 в двоичном коде.**

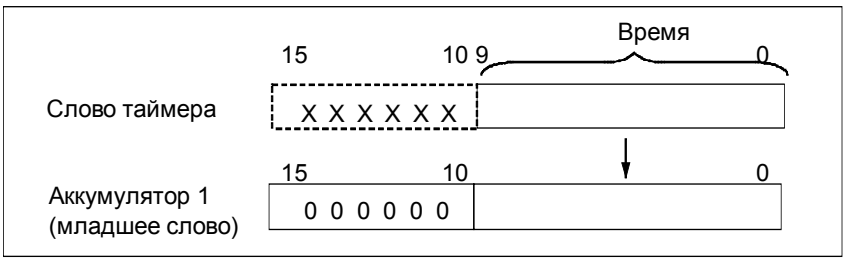


Рис. 8–2. Загрузка значения времени в аккумулятор 1 с помощью команды L

Вы можете использовать значение, содержащееся в аккумуляторе как результат операции загрузки, для дальнейшей обработки. Однако вы не можете передать значение из аккумулятора в слово таймера.

Замечание

Когда вы используете команду L для считывания слова таймера, вы получаете значение между 0 и 999. Вы не получаете базу времени, которая была загружена вместе со значением времени.

Загрузка значения счетчика

Значение счетчика хранится в слове счетчика в двоичном коде. Вы можете использовать следующую команду загрузки (L) для считывания двоичного значения счетчика из слова счетчика и загрузки этого значения в младшее слово аккумулятора 1:

L <слово счетчика>

Этот вид загрузки называют прямой загрузкой значения счетчика.

AWL	Объяснение
L C 1	**Загрузить младшее слово аккумулятора 1 счетным значением счетчика C 1 в двоичном формате.**

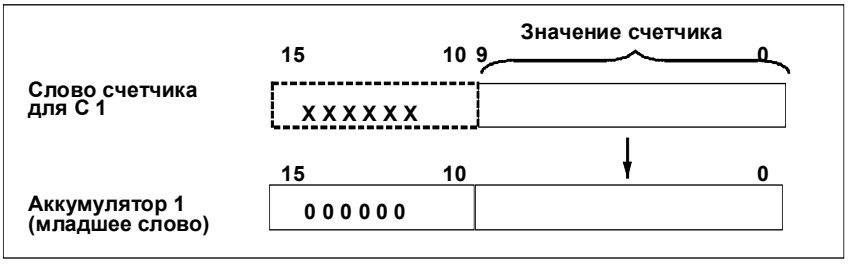


Рис. 8–3. Загрузка значения счетчика в аккумулятор 1 с помощью команды L

Вы можете использовать значение, содержащееся в аккумуляторе как результат операции загрузки L, для дальнейшей обработки. Однако вы не можете передать значение из аккумулятора в слово счетчика. Если вы хотите запустить счетчик с определенным значением, то вы должны использовать соответствующий оператор установки счетчика (см. раздел 7.2).

8.5 Загрузка значений времени и счетчиков в двоично-десятичном формате

Загрузка значения времени в формате BCD

Значение времени хранится в слове таймера в двоичном коде. Вы можете использовать следующую команду загрузки для считывания значения времени из слова таймера в двоично-десятичном формате (BCD) и загрузки его в младшее слово аккумулятора 1:

LC <слово таймера>

Кроме значения времени, загружается также и база времени. Значение, которое появляется в младшем слове аккумулятора 1 как результат команды LC, имеет формат, который необходим для запуска таймера. Этот вид загрузки называется загрузкой времени в формате BCD.

Значение времени в слове таймера уменьшается с начального значения до 0. Когда вы используете команду LC со словом таймера в качестве операнда, вы получаете значение, находящееся между стартовым временем слова таймера и 0. Время, прошедшее с момента запуска таймера, вычисляется как разность между стартовым временем и временем, считанным в данный момент.

AWL	Объяснение
LC T 1	**Загрузить младшее слово аккумулятора 1 временем и базой времени из таймера T 1 в формате BCD.**

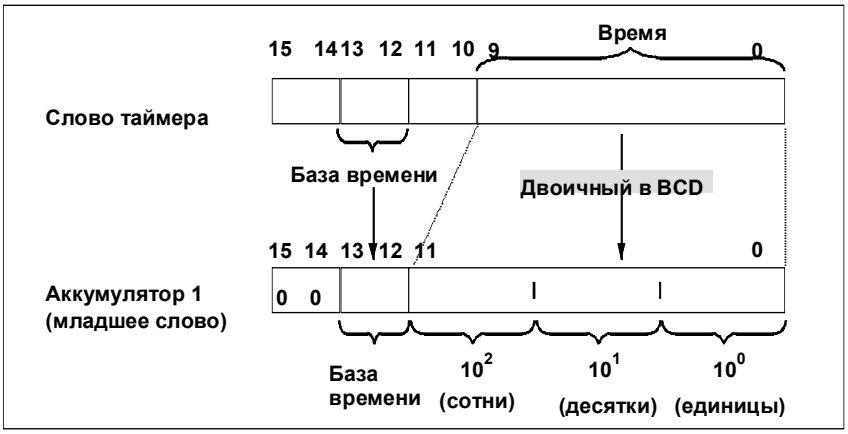


Рис. 8–4. Загрузка значения таймера в аккумулятор 1 с помощью команды LC

Значение, содержащееся в аккумуляторе как результат операции LC, может быть использовано для дальнейшей обработки, например, для передачи значения на выходы для отображения. Однако вы не можете значение из аккумулятора передать в слово таймера.

Загрузка значения счетчика в формате BCD

Значение счетчика хранится в слове счетчика в двоичном коде. Вы можете использовать следующую команду загрузки для считывания значения счетчика из слова счетчика в двоично-десятичном формате (BCD) и загрузки этого значения в младшее слово аккумулятора 1:

LC <слово счетчика>

Этот вид загрузки называют загрузкой значения счетчика в формате BCD. Значение, содержащееся в младшем слове аккумулятора 1 как результат операции LC, имеет как раз тот формат, который необходим для установки счетчика.

Значение счетчика хранится в слове счетчика в двоичном коде. Вы можете загрузить значение счетчика в младшее слово аккумулятора 1 в двоично-десятичном (BCD) коде (в формате BCD, см. рис. 8–5). С помощью команды LC вы можете считать значение счетчика в формате BCD.

AWL	Объяснение
LC C 1	**Загрузить младшее слово аккумулятора 1 счетным значением счетчика C 1 в двоично-десятичном формате.**

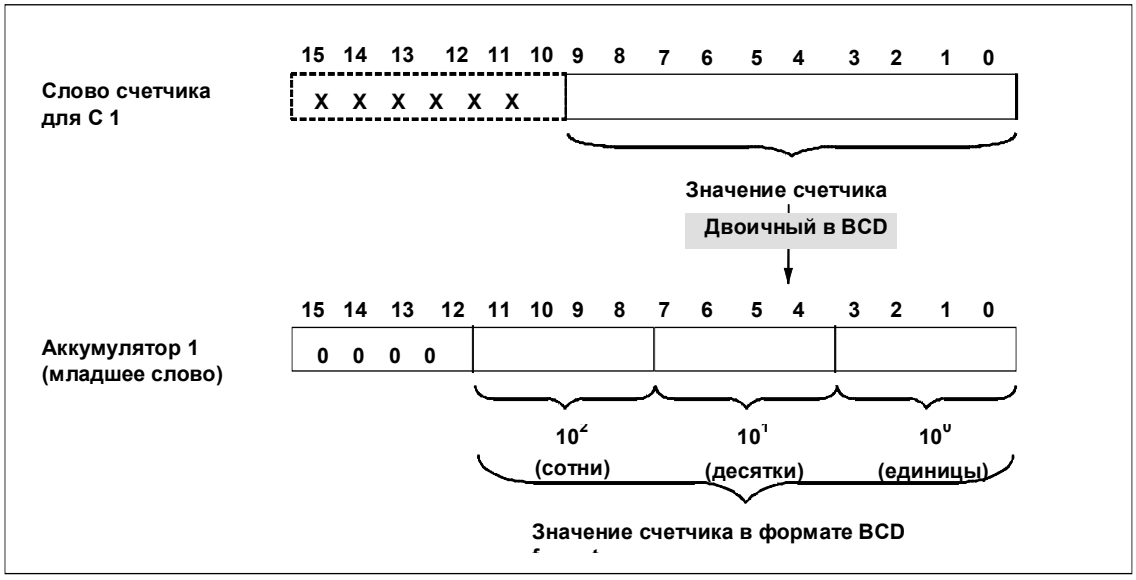


Рис. 8–5. Загрузка значения счетчика в аккумулятор 1 с помощью команды LC

Значение, содержащееся в аккумуляторе как результат операции LC, может быть использовано для дальнейшей обработки, например, для передачи значения на выходы для отображения. Однако вы не можете значение из аккумулятора передать в слово счетчика.

8.6 Загрузка и передача между адресными регистрами

Описание

Ваша программа может использовать следующие команды, чтобы дать CPU возможность обмена информацией между адресными регистрами или обмена содержимым этих двух регистров:

Команда	Объяснение
LAR1	Загружает адресный регистр 1 содержимым области, к которой обращается команда. Если операнд не указан, то LAR1 загружает адресный регистр 1 содержимым аккумулятора 1. LAR1 может использовать в качестве операнда также AR2, т.е. LAR1 может загрузить AR1 содержимым AR2.
LAR2	Загружает адресный регистр 2 содержимым области, к которой обращается команда. Если операнд не указан, то LAR2 загружает адресный регистр 2 содержимым аккумулятора 1.
TAR1	Передает содержимое адресного регистра 1 в приемник, к которому обращается команда. Если операнд не указан, то TAR1 передает содержимое адресного регистра 1 в аккумулятор 1. TAR1 может использовать в качестве операнда также AR2, т.е. TAR1 может передать содержимое AR1 в AR2.
TAR2	Передает содержимое адресного регистра 2 в приемник, к которому обращается команда. Если операнд не указан, то TAR2 передает содержимое адресного регистра 2 в аккумулятор 1.
TAR	Обменивает между собой содержимое AR1 и AR2.

Непосредственная адресация

Команды LAR1 и LAR2 могут обращаться к 32-битовым константам. Этот вид адресации называют непосредственной адресацией (см. раздел 3.1). Непосредственный адрес применяется для загрузки 32-битового указателя непосредственно в адресный регистр (см. табл. 8–5).

LAR1 P#{область,} байт{.бит}

где {область} = {I, Q, M, D, DX, L}
байт = от 0 до 65 535
{.бит} = 0 до 7

Таблица 8–5. LAR1 и LAR2: непосредственная адресация

Пример		Описание
LAR1	P#I0.0	Загружает указатель с информацией об области памяти P#I0.0 в адресный регистр 1
LAR2	P#0.0	Загружает указатель с адресом 0 внутри области в адресный регистр 2
LAR1	P##Start	Загружает указатель на локальную переменную (Start), содержащий информацию об области памяти, в адресный регистр 1

Прямая адресация

Вы можете использовать прямую адресацию с командами LAR1, LAR2, TAR1 и TAR2.

Таблица 8–6. LAR1, LAR2, TAR1, TAR2: прямая адресация

Команда	Регистр как операнд	Идентификатор операнда и диапазон
LAR1	{ПУСТО}[1)] или AR2	DBD DID LD MD от 0 до 65 532
LAR2	{ПУСТО}[1]	
TAR1	{ПУСТО}[2)] или AR2	DBD DID LD MD от 0 до 65 532
TAR2	{ПУСТО}[2)]	

1 {ПУСТО} Если операнд не указан, то LAR1/LAR2 загружает адресный регистр содержимым аккумулятора 1.

2 {ПУСТО} Если операнд не указан, то TAR1/TAR2 передает содержимое адресного регистра в аккумулятор 1.

8.7 Загрузка информации о блоке данных

Вы можете использовать команду загрузки (L) для загрузки длины или номера блока данных в аккумулятор 1. Табл. 8–7 дает обзор операндов для этого вида загрузки. За дополнительной информацией о загрузке длины или номера блока данных в аккумулятор 1 обратитесь к разделу 15.3.

Таблица 8–7. Загрузка длины или номера блока данных в аккумулятор 1

Операнд	Объяснение
DBLG	Загружает длину (в байтах) глобального блока данных в аккумулятор 1
DILG	Загружает длину (в байтах) экземплярного блока данных в аккумулятор 1
DBNO	Загружает номер глобального блока данных аккумулятор 1
DINO	Загружает номер экземплярного блока данных в аккумулятор 1

9 Арифметические операции с целыми числами

Обзор главы

9.1 Основные арифметические операции

Описание

В табл. 9–1 перечислены команды AWL, с помощью которых можно складывать, вычитать, умножать и делить целые числа (16 битов) и двойные целые числа (32 бита).

Таблица 9–1. Основные арифметические операции для целых и двойных целых чисел

Команда	Размер в битах	Функция
+I	16	Складывает содержимое младших слов аккумуляторов 1 и 2 и сохраняет результат в младшем слове аккумулятора 1.
-I	16	Вычитает содержимое младшего слова аккумулятора 1 из содержимого младшего слова аккумулятора 2 и сохраняет результат в младшем слове аккумулятора 1.
*I	16	Перемножает содержимое младших слов аккумуляторов 1 и 2 и сохраняет результат (32 бита) в аккумуляторе 1.
/I	16	Делит содержимое младшего слова аккумулятора 2 на содержимое младшего слова аккумулятора 1. Результат сохраняется в младшем слове аккумулятора 1. Целый остаток от деления сохраняется в старшем слове аккумулятора 1.
+D	32	Складывает содержимое аккумуляторов 1 и 2 и сохраняет результат в аккумуляторе 1.
-D	32	Вычитает аккумулятора 1 из содержимого аккумулятора 2 и сохраняет результат в аккумуляторе 1.
*D	32	Перемножает содержимое аккумуляторов 1 и 2 и сохраняет результат в аккумуляторе 1.
/D	32	Делит содержимое аккумулятора 2 на содержимое аккумулятора 1 и сохраняет частное в аккумуляторе 1.
MOD	32	аккумулятора 1 и сохраняет остаток от деления как результат в аккумуляторе 1.

Связь между арифметическими операциями и аккумуляторами

Описание функций в таблице 9–1 показывает, что арифметические операции комбинируют содержимое аккумуляторов 1 и 2. Результат сохраняется в аккумуляторе 1. Старое содержимое аккумулятора 1 сдвигается в аккумулятор 2.

В CPU с четырьмя аккумуляторами затем содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 в аккумулятор 3. Старое содержимое аккумулятора 4 не меняется.

Соединение двух целых чисел (16 битов) в CPU с двумя аккумуляторами

Команда сложения аккумуляторов 1 и 2 как целых чисел (+I) указывает CPU, что нужно сложить содержимое младшего слова аккумулятора 1 и младшего слова аккумулятора 2 и сохранить результат в младшем слове аккумулятора 1. Эта операция заменяет старое содержимое младшего слова аккумулятора 1. Старое содержимое аккумулятора 2 и старшего слова аккумулятора 1 не меняются (см. рис. 9–1). Пример программы следует за рисунком.

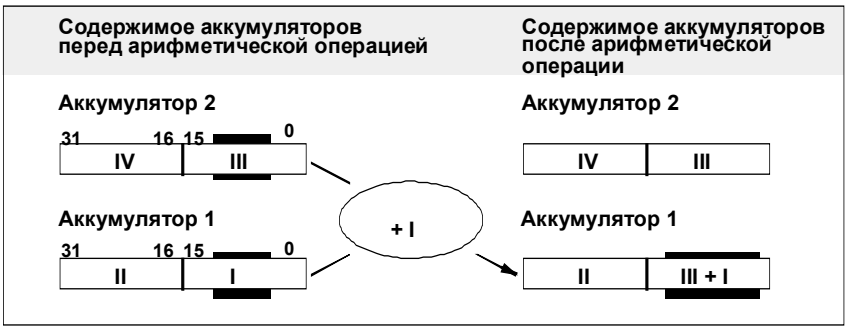


Рис. 9–1. Сложение двух целых чисел

Соединение двух целых чисел (16 битов) в CPU с четырьмя аккумуляторами

Команда сложения аккумуляторов 1 и 2 как целых чисел (+I) указывает CPU, что нужно сложить содержимое младшего слова аккумулятора 1 и младшего слова аккумулятора 2 и сохранить результат в младшем слове аккумулятора 1. Эта операция заменяет старое содержимое младшего слова аккумулятора 1. Затем она копирует содержимое аккумулятора 3 в аккумулятор 2, а содержимое аккумулятора 4 в аккумулятор 3. Аккумулятор 4 и старшее слово аккумулятора 1 не меняются (см. рис. 9–2).

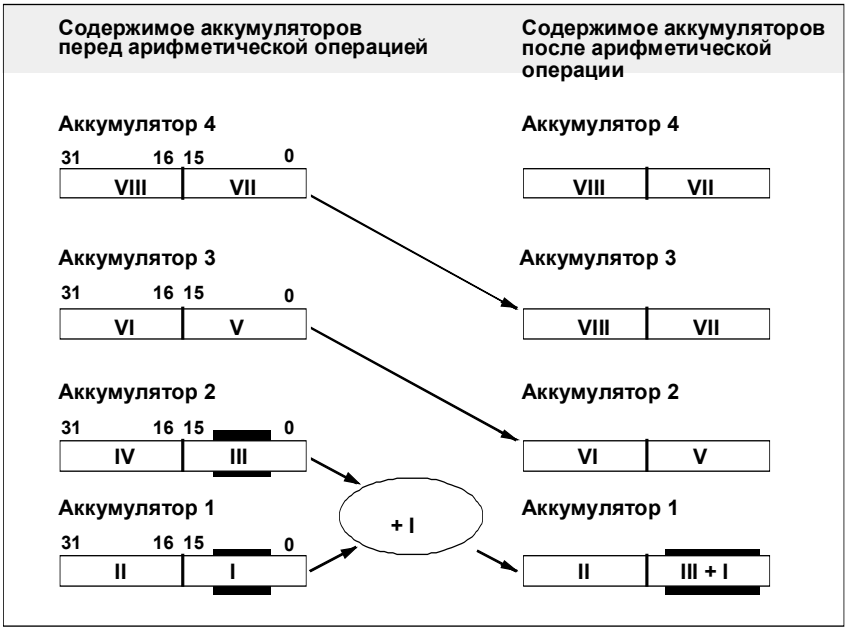


Рис. 9–2. Сложение двух целых чисел в CPU с четырьмя аккумуляторами

AWL	Объяснение
L MW10	**Загрузить значение из слова памяти MW10 в аккумулятор 1.**
L DBW12	**Загрузить значение из слова данных DBW12 в аккумулятор 1. Старое содержимое аккумулятора 1 сдвигается в аккумулятор 2.**
+I	**CPU интерпретирует содержимое младших слов аккумуляторов 1 и 2 как 16-битовые целые числа, складывает их и сохраняет результат в младшем слове аккумулятора 1.**
T DBW14	**Передать содержимое младшего слова аккумулятора 1 (результат) в слово данных DBW14.**

Анализ битов в слове состояния

Арифметические операции влияют на следующие биты слова состояния:

- CC 1 и CC 0
- OV
- OS

Допустимый результат

Прочерк (-) в столбце бита в таблице означает, что на соответствующий бит результат арифметической операции с целыми числами не влияет. Вы можете использовать команды из табл. 9–5 для анализа этих битов слова состояния.

Таблица 9–2. Состояние сигнала битов слова состояния для результата арифметической операции с целыми числами, находящегося в допустимом диапазоне

Допустимый диапазон значений результата для целых (16 битов) и двойных целых (32 бита) чисел	**Биты слова состояния**			
	CC 1	**CC 0**	**OV**	**OS**
0 (ноль)	0	0	0	-
I: –32 768 ≤ результат < 0 (отрицательное число) D: –2 147 483 648 ≤ результат < 0 (отрицательное число)	0	1	0	-
I: 32 767 ≥ результат >0 (положительное число) D: 2 147 483,647 ≥ результат >0 (положительное число)	1	0	0	-

Недопустимый результат

Таблица 9–3. Состояние сигнала битов слова состояния для результата арифметической операции с целыми числами, находящегося в недопустимом диапазоне

Диапазон недопустимых значений результата для целых (16 битов) и двойных целых (32 бита) чисел	**CC 1**	**CC 0**	**OV**	**OS**
I: результат > 32 767 (положительное число) D: результат > 2 147 483 647 (положительное число)	1	0	1	1
I: результат < -32 768 (отрицательное число) D: результат < –2 147 483 648 (отрицательное число)	0	1	1	1

Таблица 9–4. Состояние сигнала битов в слове состояния для арифметических операций с двойными целыми числами +D, /D и MOD

Команда	**Биты слова состояния**			
	CC 1	**CC 0**	**OV**	**OS**
+D: результат = –4 294 967 296	0	0	1	1
/D или MOD: произошло деление на 0.	1	1	1	1

Таблица 9–5. Команды, оценивающие биты CC 1, CC 0, OV и OS

Команда	**Ссылка на бит в слове состояния или метка перехода**	**Анализируемые биты в слове состояния (помечены X)**	**Раздел в этом руководстве**
A,O,X,AN,ON,XN	>0, <0, <>0, >=0, <=0, ==0, UO, OV, OS	CC 1, CC 0, OV, OS	5.3
JO	<метка перехода>	OV	16.4
JOS	<метка перехода>	OS	16.4
JUO	<метка перехода>	CC 1 и CC 0	16.5
JZ	<метка перехода>	CC 1 и CC 0	16.5
JN	<метка перехода>	CC 1 и CC 0	16.5
JP	<метка перехода>	CC 1 и CC 0	16.5
JM	<метка перехода>	CC 1 и CC 0	16.5
JMZ	<метка перехода>	CC 1 и CC 0	16.5
JPZ	<метка перехода>	CC 1 и CC 0	16.5

9.2 Прибавление целого числа к аккумулятору 1

Прибавление целых 16-битовых и 32-битовых констант

Вы можете использовать команду *Прибавить целую константу* для прибавления целой константы к содержимому аккумулятора 1 или его младшего слова. Эти возможности перечислены в табл. 9–6. Эти команды не влияют на биты слова состояния.

Таблица 9–6. Прибавление целого числа к аккумулятору 1

Команда	Операнд	Функция
+	+ целое число	Прибавляет 16-битовую целую константу к содержимому младшего слова аккумулятора 1. Результат сохраняется в аккумуляторе 1. Старое содержимое младшего слова этого аккумулятора заменяется. Аккумулятор 2 и старшее слово аккумулятора 1 не изменяются.
+	+ L#двойное целое число	Прибавляет 32–битовую целую константу к содержимому аккумулятора 1. Результат сохраняется в аккумуляторе 1. Старое содержимое этого аккумулятора заменяется. Аккумулятор 2 не изменяется.

Примеры

Ниже представлены две программы с командами прибавления целой константы.

AWL	Объяснение
L MW10	**Загрузить значение из MW10 в аккумулятор 1.**
L MW20	**Загрузить значение из MW20 в аккумулятор 1.**
+I	**Сложить 16–битовые значения в аккумуляторах 1 и 2.**
+ -5	**Прибавить минус 5 к результату операции +I.**
T MW14	**Передать новый результат в MW14.**

AWL	Объяснение
L MD10	**Загрузить значение из MD10 в аккумулятор 1.**
L MD16	**Загрузить значение из MD16 в аккумулятор 1.**
+D	**Сложить 32–битовые значения в аккумуляторах 1 и 2.**
+ L#-1	**Прибавить минус 1 к результату операции +D.**
T MD24	**Передать новый результат в MD24.**

10 Операции над числами с плавающей точкой

Обзор главы

10.1 Основные арифметические операции

Описание

В таблице 10–1 перечислены команды AWL, которые вы можете использовать для сложения, вычитания умножения и деления 32–битовых чисел с плавающей точкой в формате IEEE. Так как эти числа относятся к типу данных REAL (вещественные), то в качестве мнемонического сокращения для этих команд используется R.

Таблица 10–1. Основные арифметические операции для чисел с плавающей точкой (32 бита, IEEE)

Команда	Функция
+R	Складывает числа с плавающей точкой (32 бита, IEEE) в аккумуляторах 1 и 2 и сохраняет 32-битовый результат в аккумуляторе 1.
–R	Вычитает число с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 из числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 2 и сохраняет 32-битовый результат в аккумуляторе 1.
*R	Умножает число с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 на число с плавающей точкой (32 бита, IEEE) в аккумуляторе 2 и сохраняет 32-битовый результат аккумуляторе 1.
/R	Делит число с плавающей точкой (32 бита, IEEE) в аккумуляторе 2 на число с плавающей точкой (32 бита, IEEE) в аккумуляторе 1. 32-битовый результат сохраняется в аккумуляторе 1.

Связь арифметических операций с аккумуляторами

Описание функций в таблице 10–1 показывает, что арифметические операции комбинируют содержимое аккумуляторов 1 и 2. Результат сохраняется в аккумуляторе 1. Старое содержимое аккумулятора 1 сдвигается в аккумулятор 2. Содержимое аккумулятора 2 не меняется.

В CPU с четырьмя аккумуляторами содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 в аккумулятор 3. Старое содержимое аккумулятора 4 не меняется.

Результат комбинирования двух чисел с плавающей точкой в CPU с двумя аккумуляторами

Команда *Сложить аккумуляторы 1 и 2 как вещественные числа* (+R) говорит CPU о необходимости сложить содержимое аккумуляторов 1 и 2 и сохранить результат в аккумуляторе 1. Эта операция заменяет старое содержимое аккумулятора 1. Старое содержимое аккумулятора 2 не меняется (см. рис. 10–1). Пример программы следует за рис. 10–2.

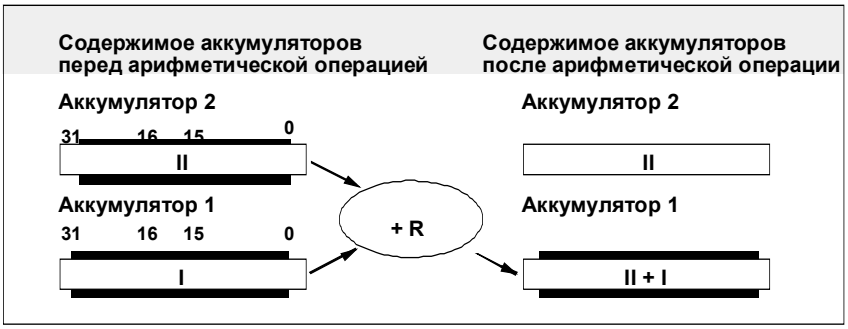


Рис. 10–1. Сложение двух чисел с плавающей точкой в формате IEEE

Такая же схема действует и для остальных команд над числами с плавающей точкой.

Таблица 10–2. Результат выполнения команд для ненормализованных чисел в CPU с двумя аккумуляторами

Команда	Входное значение		Результат	
	Аккумулятор 1	Аккумулятор 2	Аккумулятор 1	Аккумулятор 2
+R	a	b	a	b
–R	a	b	–a	b
*R	a	b	0	b
+R	a	b	FFFF	b

Результат комбинирования двух чисел с плавающей точкой в CPU с четырьмя аккумуляторами

Команда *Сложить аккумуляторы 1 и 2 как вещественные числа* (+R) говорит CPU о необходимости сложить содержимое аккумуляторов 1 и 2 и сохранить результат в аккумуляторе 1. Эта операция заменяет старое содержимое аккумулятора 1. Затем содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 (см. рис. 10–2).

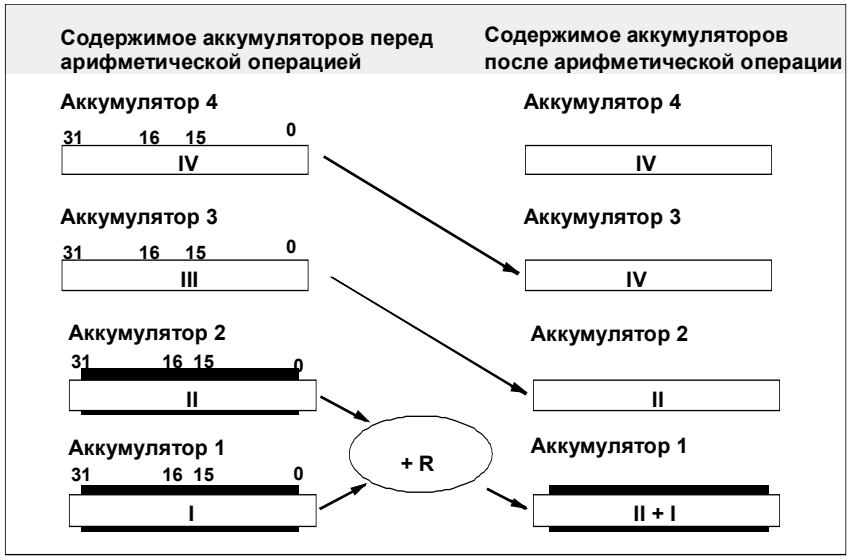


Рис. 10–2. Сложение двух чисел с плавающей точкой в формате в CPU с четырьмя аккумуляторами

Такая же схема действует и для остальных команд над числами с плавающей точкой.

Пример

AWL	Объяснение
L MD100	**Загрузить значение двойного слова памяти MD100 в аккумулятор 1.**
L DBD4	**Загрузить значение двойного слова данных DBD4 в аккумулятор 1. Старое содержимое аккумулятора 1 сдвигается в аккумулятор 2. (Значения в этих двойных словах должны быть в формате чисел с плавающей точкой).**
+R	**CPU интерпретирует содержимое аккумуляторов 1 и 2 как 32–битовые числа с плавающей точкой в формате IEEE, складывает их и сохраняет результат в аккумуляторе 1.**
T DBD16	**Передать содержимое аккумулятора 1 (результат) в двойное слово данных DBD16 (DBD16 = MD100 + DBD4)**

Биты, на которые оказывают воздействие арифметические операции

Арифметические операции влияют на следующие биты слова состояния:

- CC 1 и CC 0
- OV
- OS

Вы можете использовать команды из табл. 10–5 для анализа этих битов слова состояния. Табл. 10–3 показывает сигнальное состояние битов слова состояния для результатов арифметики с плавающей точкой внутри области допустимых значений. Прочерк (-) в столбце, соответствующем некоторому биту, означает, что результат арифметической операции с плавающей точкой на этот бит не влияет.

Таблица 10–3. Сигнальное состояние битов в слове состояния для результата арифметической операции с числами с плавающей точкой внутри области допустимых значений

Допустимый диапазон результата для чисел с плавающей точкой (32 бита)	**Биты слова состояния**			
	CC 1	**CC 0**	**OV**	**OS**
+0, –0 (ноль)	0	0	0	-
–3.402823E+38 < результат< –1.175494E-38 (отрицательное число)	0	1	0	-
+1.175494E-38 < результат< 3.402823E+38 (положительное число)	1	0	0	-

Таблица 10–4. Сигнальное состояние битов в слове состояния для результата арифметической операции с числами с плавающей точкой вне области допустимых значений

Недопустимый диапазон результата для чисел с плавающей точкой (32 бита)	**Bits of Status Word**			
	CC 1	**CC 0**	**OV**	**OS**
–1.175494E-38 < результат< –1.401298E-45 (отрицательное число) потеря значимости	0	0	1	1
+1.401298E-45 < результат< +1.175494E-38 (положительное число) потеря значимости	0	0	1	1
результат < –3.402823E+38 (отрицательное число) переполнение	0	1	1	1
результат > –3.402823E+38 (положительное число) переполнение	1	0	1	1

Таблица 10–5. Команды, анализирующие биты CC 1, CC 0, OV и OS слова состояния

Команда	**Ссылка на бит слова состояния или метка перехода**	**Анализируемые биты в слове состояния (помечены X)**	**Раздел в этом руководстве**
A,O,X,AN,ON,XN	>0, <0, <>0, >=0, <=0, ==0, UO, OV, OS	CC 1, CC 0, OV, OS	5.3
JO	<метка перехода>	OV	16.4
JOS	<метка перехода>	OS	16.4
JUO	<метка перехода>	CC 1 и CC 0	16.4
JZ	<метка перехода>	CC 1 и CC 0	16.5
JN	<метка перехода>	CC 1 и CC 0	16.5
JP	<метка перехода>	CC 1 и CC 0	16.5
JM	<метка перехода>	CC 1 и CC 0	16.5
JMZ	<метка перехода>	CC 1 и CC 0	16.5
JPZ	<метка перехода>	CC 1 и CC 0	16.5

10.2 Образование абсолютной величины числа с плавающей точкой

Описание

Вы можете использовать команду *Абсолютное значение вещественного числа* (ABS) для образования абсолютного значения числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1. Абсолютная величина – это неотрицательное число, числовое значение которого равно данному числу с плавающей точкой. Для абсолютной величины знак числа (+ или -) не имеет значения, так, например, 5 – это абсолютная величина +5 и –5.

Пример

Следующая программа дает пример команды ABS:

AWL	Объяснение
L DBD0	**Загрузить значение из двойного слова данных DBD0 в аккумулятор 1.**
L +12.3E+00	**Загрузить значение +12.3E+00 в аккумулятор 1. Старое содержимое аккумулятора 1 сдвигается в аккумулятор 2.**
/R	**CPU делит содержимое аккумулятора 2 на содержимое аккумулятора 1 и сохраняет результат в аккумуляторе 1.**
T MD20	**Передать содержимое аккумулятора 1 (результат) в двойное слово памяти MD20. (MD20 = DBD0 / 12.3)**
NEGR	**Инвертировать число с плавающей точкой (IEEE) в аккумуляторе 1 (см. раздел 12.4).**
T MD24	**Передать результат из аккумулятора 1 в двойное слово памяти MD24. (MD24 = [-1] * MD20)**
ABS	**CPU образует абсолютную величину числа с плавающей точкой (IEEE) в аккумуляторе 1.**
T MD28	**Передать абсолютную величину из аккумулятора 1 в двойное слово памяти MD28. (MD28 = ABS[MD20])**

10.3 Расширенные арифметические операции

Описание

Табл. 10–6 перечисляет команды AWL, которые могут быть использованы для выполнения расширенного набора арифметических операций над числами с плавающей точкой.

Таблица 10–6. Расширенный набор арифметических команд для чисел с плавающей точкой (32 бита, IEEE)

Команда	**Функция**
SQRT	Вычисляет квадратный корень числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 и сохраняет 32-битовый результат в аккумуляторе 1.
SQR	Вычисляет квадрат числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 и сохраняет 32-битовый результат в аккумуляторе 1.
LN	Вычисляет натуральный логарифм числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 и сохраняет 32-битовый результат в аккумуляторе 1.
EXP	Вычисляет экспоненциальное значение числа с плавающей точкой (32 бита, IEEE) по основанию E и сохраняет 32-битовый результат в аккумуляторе 1.
SIN	Вычисляет синус числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 и сохраняет 32-битовый результат в аккумуляторе 1.
COS	Вычисляет косинус числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 и сохраняет 32-битовый результат в аккумуляторе 1.
TAN	Вычисляет тангенс числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 и сохраняет 32-битовый результат в аккумуляторе 1.
ASIN	Вычисляет арксинус числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 и сохраняет 32-битовый результат в аккумуляторе 1.
ACOS	Вычисляет арккосинус числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 и сохраняет 32-битовый результат в аккумуляторе 1.
ATAN	Вычисляет арктангенс числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 и сохраняет 32-битовый результат в аккумуляторе 1.

Связь расширенных арифметических команд с аккумуляторами

Расширенные арифметические команды работают только с аккумулятором 1. Ожидается, что значение, к которому обращается операция, находится в аккумуляторе 1. Результат сохраняется в аккумуляторе 1; старое содержимое аккумулятора 1 заменяется. Содержимое аккумулятора 2, аккумулятора 3 и аккумулятора 4 не изменяется.

Влияние расширенных арифметических операций на биты слова состояния

CPU выполняет основные арифметические команды, перечисленные в табл. 10–1, не принимая во внимание и не оказывая влияния на результат логической операции. Расширенные арифметические операции влияют на следующие биты слова состояния:

- CC 1 и CC 0
- OV
- OS

Для анализа этих битов вы можете использовать команды, приведенные в таблице 10–7 (раздел 10.1), см. также раздел 5.3.

Таблица 10–7. Расширенные арифметические операции для чисел с плавающей точкой (32 бита, IEEE)

Команда	**Ссылка на биты в слове состояния или метка перехода**	**Анализируемые биты слова состояния**	**Раздел в этом руководстве**
A, O, X, AN, ON, XN	>0, <0, <>0, >=0, <=0, ==0, UO, OV, OS	CC 1, CC 0, OV, OS	5.3
JO	<метка перехода>	OV	16.4
JOS	<метка перехода>	OS	16.4
JUO	<метка перехода>	CC 1 и CC 0	16.4
JZ	<метка перехода>	CC 1 и CC 0	16.5
JN	<метка перехода>	CC 1 и CC 0	16.5
JP	<метка перехода>	CC 1 и CC 0	16.5
JM	<метка перехода>	CC 1 и CC 0	16.5
JMZ	<метка перехода>	CC 1 и CC 0	16.5
JPZ	<метка перехода>	CC 1 и CC 0	16.5

10.4 Образование квадрата / квадратного корня числа с плавающей точкой

Описание

Команда SQR (квадрат) вычисляет квадрат числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 и сохраняет 32-битовый результат в аккумуляторе 1. Команда SQR заменяет старое содержимое аккумулятора 1; содержимое аккумулятора 2, аккумулятора 3 и аккумулятора 4 не меняется.

Команда SQRT (квадратный корень) вычисляет квадратный корень числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 и сохраняет 32-битовый результат в аккумуляторе 1. Входное значение должно быть больше или равно нулю. Команда SQRT заменяет старое содержимое аккумулятора 1; содержимое аккумулятора 2, аккумулятора 3 и аккумулятора 4 не меняется.

Эта команда дает положительный результат, когда все операнды больше нуля («0»). Единственное исключение: квадратный корень из -0 равен -0.

Влияние на биты CC 1, CC 0, OV и OS слова состояния

Таблица 10–8. Влияние команды SQR на биты CC 1, CC 0, OV и OS

Результат в аккумуляторе 1	CC 1	CC 0	OV	OS
+ qNaN	1	1	1	1
+ бесконечность (переполнение)	1	0	1	1
+ нормализован	1	0	0	-
+ не нормализован (потеря значимости)	0	0	1	1
+ нуль	0	0	0	-
– qNaN	1	1	1	1

Пример

Следующий фрагмент программы показывает на примере применение команды SQR.

AWL	Объяснение
OPN DB17	**Открыть блок данных DB17. (Предполагается, что он содержит входную величину и будет хранить результат).**
L DBD0	**Загрузить значение из двойного слова данных DBD0 в аккумулятор 1. (Значение в этом двойном слове должно быть в формате числа с плавающей точкой.)**
SQR	**Вычислить квадрат числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1. Сохранить результат в аккумуляторе 1.**
AN OV	**Опросить бит состояния OV на равенство 0.**
JC OK	**Если при выполнении операции SQR не произошло ошибки, перейти на метку OK**
...	**(Здесь реализуется реакция на возникновение ошибки.)**
OK: T DBD4	**Передать результат из аккумулятора 1 в двойное слово данных DBD4.**

Таблица 10–9. Влияние команды SQRT на биты CC 1, CC 0, OV и OS

Результат в аккумуляторе 1	**CC 1**	**CC 0**	**OV**	**OS**
+ qNaN	1	1	1	1
+ бесконечность (переполнение)	1	0	1	1
+ нормализован	1	0	0	-
+ не нормализован (потеря значимости)	0	0	1	1
+ нуль	0	0	0	-
– нуль	0	0	0	-
– qNaN	1	1	1	1

Пример

Следующий фрагмент программы является примером использования команды SQRT.

AWL	**Объяснение**
L MD10	**Загрузить значение из двойного слова памяти MD10 в аккумулятор 1. (Это значение должно иметь формат числа с плавающей точкой.)**
SQRT	**Образовать квадратный корень из числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1. Сохранить результат в аккумуляторе 1.**
AN OV	**Опросить бит состояния OV на равенство 0.**
JC OK	**Если при выполнении операции SQRT не произошло ошибки, то перейти на метку OK.**
...	**(Здесь реализуется реакция на возникновение ошибки.)**
OK: T MD20	**Передать результат (аккумулятор 1) в двойное слово памяти MD20.**

10.5 Образование натурального логарифма числа с плавающей точкой

Описание

Команда LN (натуральный логарифм) вычисляет натуральный логарифм числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 и сохраняет 32-битовый результат в аккумуляторе 1. Входное значение должно быть больше нуля. Команда LN заменяет старое содержимое аккумулятора 1; содержимое аккумулятора 2, аккумулятора 3 и аккумулятора 4 не меняется.

Влияние на биты CC 1, CC 0, OV и OS слова состояния

Табл. 10–10 показывает влияние, которое команда LN оказывает на состояние битов CC 1, CC 0, OV и OS слова состояния. Прочерк (–) в столбце соответствующего бита означает, что результат арифметической операции над числами с плавающей точкой на этот бит влияния не оказывает.

Таблица 10–10. Воздействие команды LN на биты CC 1, CC 0, OV и OS

Результат в аккумуляторе 1	**CC 1**	**CC 0**	**OV**	**OS**
+ qNaN	1	1	1	1
+ бесконечность (переполнение)	1	0	1	1
+ нормализован	1	0	0	-
+ не нормализован (потеря значимости)	0	0	1	1
+ нуль	0	0	0	-
– нуль	0	0	0	-
– не нормализован (потеря значимости)	0	0	1	1
– нормализован	0	1	0	-
– бесконечность (переполнение)	0	1	1	1
– qNaN	1	1	1	1

Пример

Следующий фрагмент программы является примером использования команды LN.

```
AWL                 Объяснение
L MD10              Загрузить значение из двойного слова памяти MD10 в аккумулятор 1. (Это
                    значение должно иметь формат числа с плавающей точкой.)
LN                  Образовать натуральный логарифм числа с плавающей точкой (32 бита,
                    IEEE) в аккумуляторе 1. Сохранить результат в аккумуляторе 1.
AN OV               Опросить бит состояния OV на равенство 0.
JC OK               Если при выполнении операции LN не произошла ошибка, то перейти на
                    метку OK.
...                 (Здесь реализуется реакция на возникновение ошибки.)
OK: T MD20          Передать результат из аккумулятора 1 в двойное слово памяти MD20.
```

10.6 Образование экспоненциального значения числа с плавающей точкой

Описание

Команда EXP (экспоненциальное значение по основанию E) вычисляет экспоненциальное значение числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 по основанию E (= 2,71828...) и сохраняет 32-битовый результат в аккумуляторе 1. Команда EXP заменяет старое содержимое аккумулятора 1; содержимое аккумулятора 2, аккумулятора 3 и аккумулятора 4 не меняется.

Влияние на биты CC 1, CC 0, OV и OS слова состояния

Табл.10–11 показывает влияние, которое команда LN оказывает на состояние битов CC 1, CC 0, OV и OS слова состояния. Прочерк (–) в столбце соответствующего бита означает, что результат арифметической операции над числами с плавающей точкой на этот бит влияния не оказывает.

Таблица 10–11. Воздействие команды EXP на биты CC 1, CC 0, OV и OS

Результат в аккумуляторе 1	**CC 1**	**CC 0**	**OV**	**OS**
+ qNaN	1	1	1	1
+ бесконечность (переполнение)	1	0	1	1
+ нормализован	1	0	0	-
+ не нормализован (потеря значимости)	0	0	1	1
+ нуль	0	0	0	-
– qNaN	1	1	1	1

Пример

Следующий фрагмент программы является примером использования команды EXP.

```
AWL                 Объяснение
L MD10              Загрузить значение из двойного слова памяти MD10 в аккумулятор 1. (Это
                    значение должно иметь формат числа с плавающей точкой.)
EXP                 Образовать экспоненциальное значение по основанию E числа с
                    плавающей точкой (32 бита, IEEE) в аккумуляторе 1. Сохранить результат в
                    аккумуляторе 1.
AN OV               Опросить бит состояния OV на равенство 0.
JC OK               Если при выполнении операции EXP не произошла ошибка, то перейти на
                    метку OK.
...                 (Здесь реализуется реакция на возникновение ошибки.)
OK: T MD20          Передать результат из аккумулятора 1 в двойное слово памяти MD20.
```

10.7 Образование тригонометрических функций углов как чисел с плавающей точкой

Описание

С помощью следующих команд вы можете образовать тригонометрические функции углов, представленных в виде чисел с плавающей точкой (32 бита, IEEE). 32-битовый результат сохраняется в аккумуляторе 1, содержимое аккумулятора 2, аккумулятора 3 и аккумулятора 4 не меняется.

Команда	Функция
SIN	Образовать синус числа с плавающей точкой, представляющего угол, выраженный в радианах. Сохранить угол в виде числа с плавающей точкой в аккумуляторе 1.
ASIN	Образовать арксинус числа с плавающей точкой в аккумуляторе 1. Результатом является угол, выраженный в радианах. Значение будет находиться в следующем диапазоне: $-\pi / 2 \leq$ арксинус (ACCU 1) $\leq + \pi / 2$, где $\pi = 3.14...$
COS	Образовать косинус числа с плавающей точкой, представляющего угол, выраженный в радианах. Сохранить угол в виде числа с плавающей точкой в аккумуляторе 1.
ACOS	Образовать арккосинус числа с плавающей точкой в аккумуляторе 1. Результатом является угол, выраженный в радианах. Значение будет находиться в следующем диапазоне: $0 \leq$ арккосинус (ACCU 1) $\leq + \pi$, где $\pi = 3.14...$
TAN	Образовать тангенс числа с плавающей точкой, представляющего угол, выраженный в радианах. Сохранить угол в виде числа с плавающей точкой в аккумуляторе 1.
ATAN	Образовать арктангенс числа с плавающей точкой в аккумуляторе 1. Результатом является угол, выраженный в радианах. Значение будет находиться в следующем диапазоне: $-\pi / 2 \leq$ арктангенс (AKKU 1) $\leq + \pi / 2$, где $\pi = 3.14...$

Влияние на биты CC 1, CC 0, OV и OS слова состояния

Табл.10–12 показывает влияние, которое команды SIN, ASIN, COS, ACOS и ATAN оказывают на состояние битов CC 1, CC 0, OV и OS слова состояния. Табл.10–13 показывает, как влияет на эти биты команда TAN. Прочерк (-) означает, что команда не оказывает влияния на этот бит.

Таблица 10–121. Влияние команд SIN, ASIN, COS, ACOS и ATAN

Результат в аккумуляторе 1	**CC 1**	**CC 0**	**OV**	**OS**
+ qNaN	1	1	1	1
+ нормализован	1	0	0	-
+ бесконечность (переполнение)	0	0	1	1
+ нуль	0	0	0	-
– нуль	0	0	0	-
– не нормализован (потеря значимости)	0	0	1	1
– нормализован	0	1	0	-
– qNaN	1	1	1	1

Таблица 10–13. Воздействие команды TAN на биты CC 1, CC 0, OV и OS

Результат в аккумуляторе 1	**CC 1**	**CC 0**	**OV**	**OS**
+ qNaN	1	1	1	1
+ бесконечность (переполнение)	1	0	1	1
+ нормализован	1	0	0	-
+ не нормализован (потеря значимости)	0	0	1	1
+ нуль	0	0	0	-
– нуль	0	0	0	-
– не нормализован (потеря значимости)	0	0	1	1
– нормализован	0	1	0	-
– бесконечность (переполнение)	0	1	1	1
– qNaN	1	1	1	1

Пример

Следующий фрагмент программы является примером использования команды SIN.

AWL	**Объяснение**
L MD10	**Загрузить значение из двойного слова памяти MD10 в аккумулятор 1. (Это значение должно иметь формат числа с плавающей точкой.)**
SIN	**Образовать синус числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1. Сохранить результат в аккумуляторе 1.**
T MD20	**Передать результат из аккумулятора 1 в двойное слово памяти MD20.**

Пример

Следующий фрагмент программы является примером использования команды ASIN.

AWL	Объяснение
L MD10	**Загрузить значение из двойного слова памяти MD10 в аккумулятор 1. (Это значение должно иметь формат числа с плавающей точкой.)**
ASIN	**Образовать арксинус числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1. Сохранить результат в аккумуляторе 1.**
AN OV	**Опросить бит состояния OV на равенство 0.**
JC OK	**Если при выполнении операции ASIN не произошла ошибка, то перейти на метку OK.**
...	**(Здесь реализуется реакция на возникновение ошибки.)**
OK: T MD20	**Передать результат из аккумулятора 1 в двойное слово памяти MD20.**

Анализ битов в слове состояния

Расширенные арифметические операции влияют на следующие биты в слове состояния:

- CC 1 и CC 0
- OV
- OS

Допустимый результат

Прочерк (-) в столбце таблицы означает, что результат арифметической операции над числами с плавающей точкой на соответствующий бит влияния не оказывает.

Таблица 10–14. Обратные тригонометрические функции для чисел с плавающей точкой (32 бита, IEEE) и допустимые диапазоны значений для входной величины

Допустимый диапазон для результата, являющегося числом с плавающей точкой (32 бита)	**Биты слова состояния**			
	CC 1	**CC 0**	**OV**	**OS**
+0, –0 (нуль)	0	0	0	-
–3.402823E+38 < результат< –1.175494E-38 (отрицательное число)	0	1	0	-
+1.175494E-38 < результат< 3.402823E+38 (положительное число)	1	0	0	-

Недопустимый результат

Таблица 10–14. Состояния битов в слове состояния: результат вычислений над числами с плавающей точкой вне допустимого диапазона

Недопустимый диапазон для результата, являющегося числом с плавающей точкой (32 бита)	**Биты слова состояния**			
	CC 1	**CC 0**	**OV**	**OS**
–1.175494E-38 < результат< –1.401298E-45 (отрицательное число) потеря значимости	0	0	1	1
+1.401298E-45 < результат< +1.175494E-38 (положительное число) потеря значимости	0	0	1	1
результат< –3.402823E+38 (отрицательное число) переполнение	0	1	1	1
результат > –3.402823E+38 (положительное число) переполнение	1	0	1	1

Таблица 10–16. Состояния битов в слове состояния: входная величина представляет недопустимое число с плавающей точкой или находится вне допустимого диапазона

Недопустимый диапазон входного значения для чисел с плавающей точкой (32 бита)	**Биты слова состояния**			
	CC 1	**CC 0**	**OV**	**OS**
В аккумуляторе 1 не 32-битовое число с плавающей точкой формата IEEE.	1	1	1	1
Недопустимая команда: входная величина в аккумуляторе 1 находится вне допустимого диапазона.	1	1	1	1

11 Операции сравнения

Обзор главы

11.1 Обзор

Вы можете использовать команды *Сравнить* для сравнения следующих пар числовых значений:

- два целых числа (16 битов)
- два двойных целых числа (32 бита)
- два вещественных числа (числа с плавающей точкой в формате IEEE, 32 бита)

Вы загружаете числовые значения в аккумуляторы 1 и 2. Команда *Сравнить* сравнивает значение в аккумуляторе 1 со значением в аккумуляторе 2 в соответствии с критериями, перечисленными в табл. 11–1.

Результатом сравнения является двоичная цифра, т.е. 1 или 0. 1 указывает, что результат сравнения является истиной; 0 указывает, что результат сравнения является ложью (см. табл. 11–2). Этот результат сохраняется в бите результата логической операции (бит RLO, см. раздел 3.4). Этот результат вы можете использовать в своей программе для дальнейшей обработки.

Когда CPU выполняет команду сравнения, он также устанавливает биты в слове состояний. Другие команды AWL могут анализировать биты слова состояния. CPU выполняет команды сравнения независимо от результата логической операции.

Таблица 11–1. Критерии сравнения

Вид числового значения в аккумуляторе 2	**Критерий сравнения**	**Символ (ы) для команды**	**Вид числового значения в аккумуляторе 1**
Целое число (16 битов)	равно	==I	Целое число (16 битов)
Двойное целое число (32 бита)		==D	Двойное целое число (32 бита)
Вещественное число (с плавающей точкой, 32 бита)		==R	Вещественное число (с плавающей точкой, 32 бита)
	не равно	<>I <>D <>R	
	больше, чем	>I >D >R	
	меньше, чем	<I <D <R	
	больше или равно	>=I >=D >=R	
	меньше или равно	<=I <=D <=R	

11.2 Сравнение двух целых чисел

Описание

Команды *Сравнить целые числа* сравнивают два целых числа (16 битов каждое), а команды *Сравнить двойные целые числа* сравнивают два двойных целых (32 бита каждое) в соответствии с критериями, перечисленными в табл. 11–2. Пример программы следует за таблицей 11–3.

Таблица 11–2. Команды сравнения целых и двойных целых чисел

Команда	Объяснение
==I	Целое число в младшем слове аккумулятора 2 равно целому числу в младшем слове аккумулятора 1.
==D	Двойное целое число в аккумуляторе 2 равно двойному целому числу в аккумуляторе 1.
<>I	Целое число в младшем слове аккумулятора 2 не равно целому числу в младшем слове аккумулятора 1.
<>D	Двойное целое число в аккумуляторе 2 не равно двойному целому числу в аккумуляторе 1.
>I	Целое число в младшем слове аккумулятора 2 больше целого числа в младшем слове аккумулятора 1.
>D	Двойное целое число в аккумуляторе 2 больше двойного целого числа в аккумуляторе 1.
<I	Целое число в младшем слове аккумулятора 2 меньше целого числа в младшем слове аккумулятора 1.
<D	Двойное целое число в аккумуляторе 2 меньше двойного целого числа в аккумуляторе 1.
>=I	Целое число в младшем слове аккумулятора 2 больше или равно целому числу в младшем слове аккумулятора 1.
>=D	Двойное целое число в аккумуляторе 2 больше или равно двойному целому числу в аккумуляторе 1.
<=I	Целое число в младшем слове аккумулятора 2 меньше или равно целому числу в младшем слове аккумулятора 1.
<=D	Двойное целое число в аккумуляторе 2 меньше или равно двойному целому числу в аккумуляторе 1.

Установка битов CC 1 и CC 0 слова состояния

Команда *Сравнить целые числа* или *Сравнить двойные целые числа* устанавливает определенные комбинации битов CC 1 и CC 0, чтобы показать, какое условие было выполнено (см. табл. 11–3).

Таблица 11–3. Состояния битов CC 1 и CC 0 после команды сравнения

Условие	Состояния CC 1 и CC 0:		Возможный опрос с помощью команд
	CC 1	CC 0	A, O, X, AN, ON, XN
ACCU 2 > ACCU 1	1	0	>0
ACCU 2 < ACCU 1	0	1	<0
ACCU 2 = ACCU 1	0	0	==0
ACCU 2 <> ACCU 1	0 или 1	1 или 0	<>0
ACCU 2 >= ACCU 1	1 или 0	0 или 0	>=0
ACCU 2 <= ACCU 1	0 или 0	1 или 0	<=0

Пример

Следующий пример программы показывает, как работают команды сравнения для целых чисел (16 битов).

AWL	Объяснение
L MW10	**Загрузить содержимое слова памяти MW10 в аккумулятор 1.**
L IW0	**Загрузить содержимое входного слова IW0 в аккумулятор 1. Старое содержимое аккумулятора 1 смещается в аккумулятор 2.**
==I	**Сравнить значение в младшем слове аккумулятора 2 со значением в младшем слове аккумулятора 1, чтобы выяснить, равны ли они.**
= Q 4.0	**Выход Q 4.0 будет проводить ток, если MW10 и IW0 равны.**
>I	**Сравнить значение в младшем слове аккумулятора 2 со значением в младшем слове аккумулятора 1, чтобы выяснить, больше ли оно, чем значение в младшем слове аккумулятора 1.**
= Q 4.1	**Выход Q 4.1 будет проводить ток, если MW10 больше, чем IW0.**
<I	**Сравнить значение в младшем слове аккумулятора 2 со значением в младшем слове аккумулятора 1, чтобы выяснить меньше ли оно, чем значение в младшем слове аккумулятора 1.**
= Q 4.2	**Выход Q 4.2 будет проводить ток, если MW10 меньше, чем IW0.**

11.3 Сравнение двух чисел с плавающей точкой

Описание

Команды *Сравнить числа с плавающей точкой* сравнивают два числа с плавающей точкой (32 бита, IEEE) в соответствии с критериями, перечисленными в табл. 11–4. Так как числа с плавающей точкой (32 бита, IEEE) относятся к типу данных REAL (вещественные), то в качестве мнемонического сокращения для этих команд используется R.

Таблица 11–4. Команды сравнения вещественных чисел (32-битовых чисел с плавающей точкой в формате IEEE)

Команда	Объяснение
==R	Число с плавающей точкой (32 бита, IEEE) в аккумуляторе 2 равно числу с плавающей точкой (32 бита, IEEE) в аккумуляторе 1.
<>R	Число с плавающей точкой (32 бита, IEEE) в аккумуляторе 2 не равно числу с плавающей точкой (32 бита, IEEE) в аккумуляторе 1.
>R	Число с плавающей точкой (32 бита, IEEE) в аккумуляторе 2 больше числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1.
<R	Число с плавающей точкой (32 бита, IEEE) в аккумуляторе 2 меньше числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1.
>=R	Число с плавающей точкой (32 бита, IEEE) в аккумуляторе 2 больше или равно числу с плавающей точкой (32 бита, IEEE) в аккумуляторе 1.
<=R	Число с плавающей точкой (32 бита, IEEE) в аккумуляторе 2 меньше или равно числу с плавающей точкой (32 бита, IEEE) в аккумуляторе 1.

Установка битов слова состояния

Команда *Сравнить вещественные числа* устанавливает определенные комбинации битов CC 1, CC 0, OV и OS слова состояния, чтобы показать, какое условие было выполнено.

Таблица 11–5. Состояния битов в слове состояния после команды сравнения вещественных чисел (чисел с плавающей точкой в формате IEEE, 32 бита)

Условие	CC 1	CC 0	OV	OS
==	0	0	0	неприменимо
<>	0 или 1	1 или 0	0	неприменимо
>	1	0	0	неприменимо
<	0	1	0	неприменимо
>=	1 или 0	0 или 0	0	неприменимо
<=	0 или 0	1 или 0	0	неприменимо
UO	1	1	1	1

Анализ битов слова состояния

Другие команды AWL могут анализировать биты в слове состояния (см. раздел 5.3 и табл. 11–6).

Таблица 11–6. Команды, анализирующие биты CC 1, CC 0, OV и OS слова состояния

Команда	Ссылка на биты слова состояния или метка перехода	Раздел в этом руководстве
A,O,X,AN,ON,XN	>0, <0, <>0, >=0, <=0, ==0, UO, OV, OS	5.3
JUO	<метка перехода>	16.4
JZ	<метка перехода>	16.5
JN	<метка перехода>	16.5
JP	<метка перехода>	16.5
JM	<метка перехода>	16.5
JMZ	<метка перехода>	16.5
JPZ	<метка перехода>	16.5

Пример

Следующий пример программы показывает, как работают команды сравнения вещественных чисел.

AWL	Объяснение
L MD24	**Загрузить содержимое двойного слова памяти MD24 в аккумулятор 1.**
L +1.00E+00	**Загрузить значение 1.0 как число с плавающей точкой (32 бита) в аккумулятор 1. Старое содержимое аккумулятора 1 сдвигается в аккумулятор 2.**
>R	**Сравнить значение в аккумуляторе 2 со значением в аккумуляторе 1, чтобы выяснить, больше ли оно, чем значение в аккумуляторе 1.**
= Q 4.1	**Выход Q 4.1 будет проводить ток, если MD24 больше, чем 1.0.**
<R	**Сравнить значение в аккумуляторе 2 со значением в аккумуляторе 1, чтобы выяснить, меньше ли оно, чем значение в аккумуляторе 1.**
= Q 4.2	**будет проводить ток, если MD24 меньше, чем 1.0.**

12 Команды преобразования

Обзор главы

12.1 Преобразование чисел в двоично-десятичном коде и целых чисел

Описание

Вы можете использовать следующие команды для преобразования чисел, представленных в двоично-десятичном коде, и целых чисел в другие виды чисел:

Мнемоника	Команда	Функция
BTI	BCD в целое	Эта команда преобразует находящееся в младшем слове аккумулятора 1 число, представленное в двоично-десятичном коде, в 16-битовое целое число.
BTD	BCD в двойное целое	Эта команда преобразует находящееся в аккумуляторе 1 число, представленное в двоично-десятичном коде, в 32–битовое целое число.
ITB	Целое в BCD	Эта команда преобразует 16–битовое целое число в младшем слове аккумулятора 1 в двоично-десятичное число.
ITD	Целое в двойное целое	Эта команда преобразует 16–битовое целое число в младшем слове аккумулятора 1 в 32–битовое целое число.
DTB	Двойное целое число в BCD	Эта команда преобразует 32–битовое целое число в аккумуляторе 1 в двоично-десятичное число.
DTR	Двойное целое число в вещественное	Эта команда преобразует 32–битовое целое число в аккумуляторе 1 в 32–битовое число с плавающей точкой в формате IEEE (вещественное число).

BCD в целое: BTI

Команда *BCD в целое* (BTI) преобразует трехразрядное число, записанное в двоично-десятичном коде (BCD-число, см. рис. 12–1) в младшем слове аккумулятора 1 в 16–битовое целое число. BCD-число может находиться в диапазоне от –999 до +999. Результат преобразования сохраняется в младшем слове аккумулятора 1.

Рис. 12–1 Структура двоично-десятичного числа, которое должно быть преобразовано в целое число

Если какой-либо разряд BCD–числа находится в недопустимом диапазоне между 10 и 15, то при попытке преобразования возникает ошибка BCDF. Имеет место один из следующих случаев:

- CPU переходит в состояние STOP. В диагностический буфер вносится запись «BCD Conversion Error [Ошибка BCD-преобразования]» с идентификатором события номер 2521.
- Если запрограммирован OB121, то он вызывается.

Следующий пример программы содержит команду BTI. Рис. 12–2 показывает, как работает эта команда.

AWL	Объяснение
L MW10	Загрузить величину в BCD–коде из слова памяти MW10 в аккумулятор 1.
BTI	Преобразовать величину в BCD–коде в 16–битовое целое число и сохранить результат в аккумуляторе 1.
T MW20	Передать результат в слово памяти MW20.

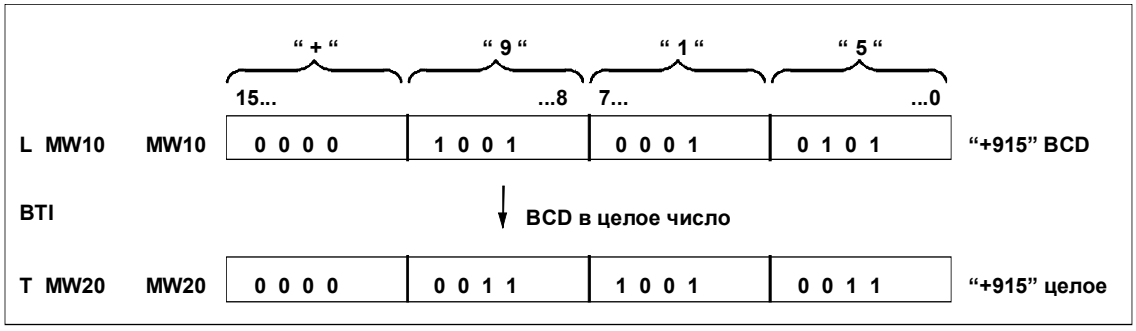


Рис. 12–2. Использование команды BTI для преобразования BCD–числа в 16–битовое целое число

BCD в двойное целое: BTD

Команда *BCD в двойное целое* (BTD) преобразует семиразрядное число, записанное в двоично-десятичном коде (BCD–число, см. рис. 12–3) в аккумуляторе 1 в 32–битовое целое число. BCD–число может находиться в диапазоне от –9 999 999 до +9 999 999. Результат преобразования сохраняется в аккумуляторе 1.

Если какой-либо разряд BCD–числа находится в недопустимом диапазоне между 10 и 15, то при попытке преобразования возникает ошибка BCDF. Имеет место один из следующих случаев:

- CPU переходит в состояние STOP. В диагностический буфер вносится запись «BCD Conversion Error [Ошибка BCD-преобразования]» с идентификатором события номер 2521.
- Если запрограммирован OB121, то он вызывается.

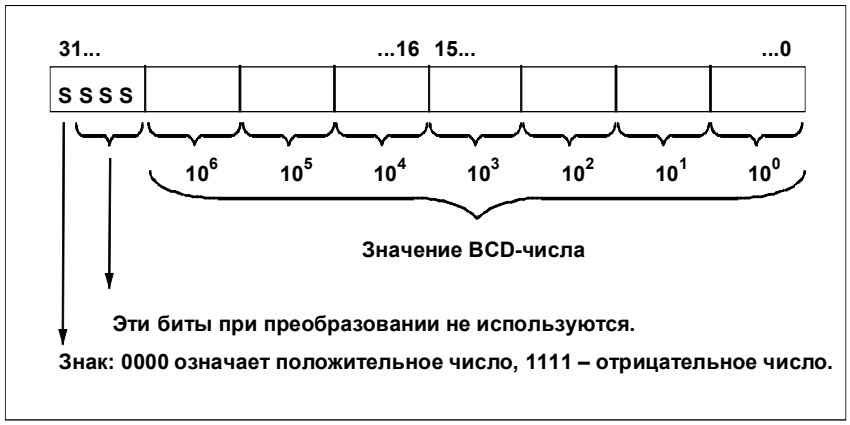


Рис. 12–3. Структура 32–битового BCD–числа, которое должно быть преобразовано в двойное целое число

Следующий пример программы содержит команду BTD. Рис. 12–4 показывает, как работает эта команда.

AWL	Объяснение
L MD10	Загрузить величину в BCD–коде из двойного слова памяти MD10 в аккумулятор 1.
BTD	Преобразовать величину в BCD–коде в 32–битовое целое число и сохранить результат в аккумуляторе 1.
T MD20	Передать результат в двойное слово памяти MD20.

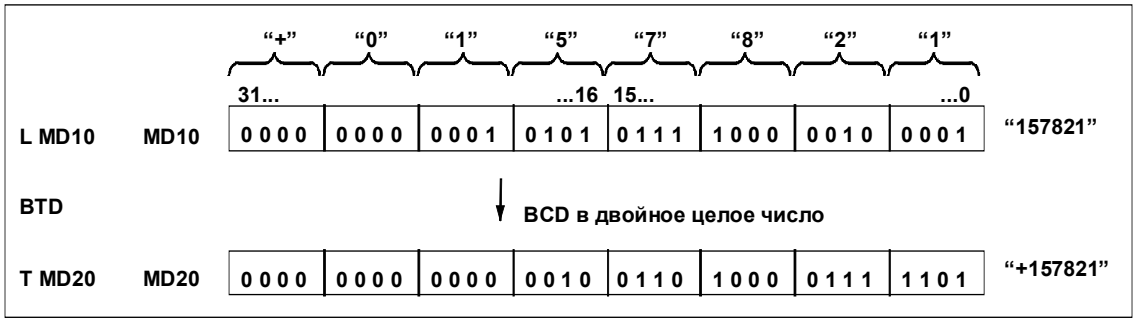


Рис. 12–4. Использование команды BTD для преобразования BCD–числа в 32– битовое целое число

Целое в BCD: ITB

Команда *Целое в BCD* (ITB) преобразует 16–битовое целое число, находящееся в младшем слове аккумулятора 1 в трехразрядное число, записанное в двоично-десятичном коде. BCD–число может находиться в диапазоне от–999 до +999. Результат преобразования сохраняется в младшем слове аккумулятора 1.

Если целое число слишком велико, чтобы быть представленным в формате BCD, то преобразование не производится, биты переполнения (OV) и сохраняемого переполнения (OS) в слове состояния (см. раздел 3.4) устанавливаются в 1.

Следующий пример программы содержит команду ITB. На рис. 12–5 показано, как эта команда работает.

AWL	Объяснение
L MW10	Загрузить значение 16-битового целого числа из слова памяти MW10 в аккумулятор 1.
ITB	Преобразовать 16–битовое целое число в BCD–код и сохранить результат в аккумуляторе 1.
T MW20	Передать результат в слово памяти MW20.

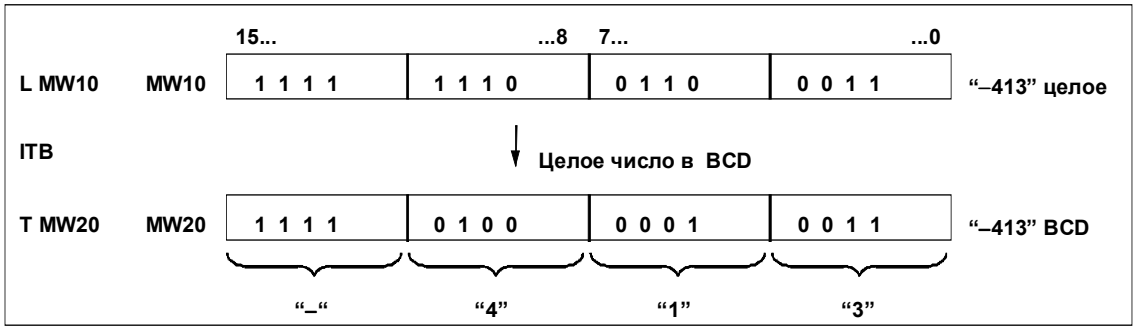


Рис. 2–5. Использование команды ITB для преобразования 16–битового целого числа в BCD–число

Целое в двойное целое: ITD

Команда *Целое в двойное целое* (ITD) преобразует 16–битовое целое число, находящееся в младшем слове аккумулятора 1, в 32–битовое целое число. Результат преобразования сохраняется в аккумуляторе 1. Следующий пример программы содержит команду ITD. На рис. 12–6 показано, как эта команда работает.

AWL	Объяснение
L MW10	Загрузить 16–битовое целое число из слова памяти MW10 в аккумулятор 1.
ITD	Преобразовать 16–битовое целое число в 32–битовое целое число и сохранить результат в аккумуляторе 1.
T MD20	Передать результат в двойное слово памяти MD20.

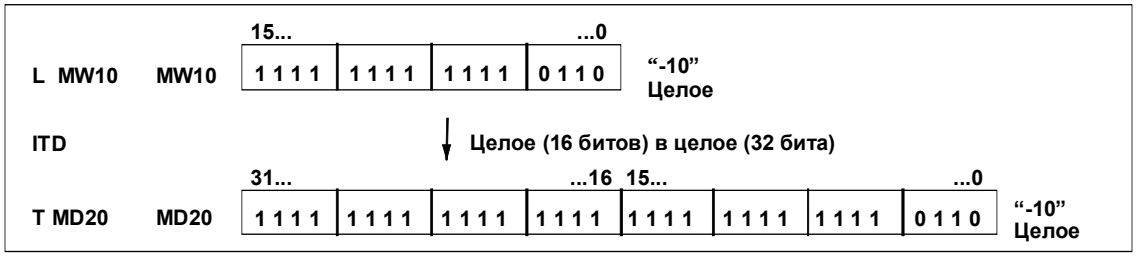


Рис. 12–6. Использование команды ITD для преобразования 16–битового целого числа в 32–битовое целое число

Двойное целое в BCD: DTB

Команда *Двойное целое в BCD* (DTB) преобразует 32–битовое целое число, находящееся в аккумуляторе 1, в семиразрядное число, записанное в двоично-десятичном коде. BCD–число может находиться в диапазоне от –9 999 999 до +9 999 999. Результат преобразования сохраняется в аккумуляторе 1.

Если двойное целое число велико, чтобы быть представленным в формате BCD, то преобразование не выполняется, биты переполнения (OV) и сохраняемого переполнения (OS) в слове состояния (см. раздел 3.4) устанавливаются в 1.

Следующий пример программы содержит команду DTB. На рис. 12–7 показано, как эта команда работает.

AWL	Объяснение
L MD10	Загрузить целое число (32 бита) из двойного слова памяти MD10 в аккумулятор 1.
DTB	Преобразовать целое число (32 бита) в BCD–код и сохранить результат в аккумуляторе 1.
T MD20	Передать результат в двойное слово памяти MD20.

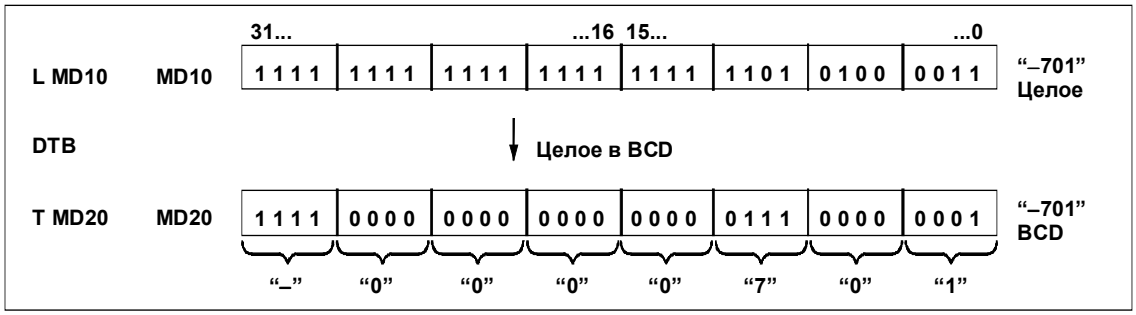


Рис. 12–7. Использование команды DTB для преобразования 32–битового целого числа в BCD–число

Двойное целое в вещественное: DTR

Команда *Двойное целое в вещественное* (DTR) преобразует 32–битовое целое число, находящееся в аккумуляторе 1 в 32–битовое число с плавающей точкой в формате IEEE (вещественное число). Если необходимо, команда округляет результат. Результат преобразования сохраняется в аккумуляторе 1. Следующий пример программы содержит команду DTR. На рис. 12–8 показано, как эта команда работает.

AWL	Объяснение
L MD10	**Загрузить целое число (32 бита) из двойного слова памяти MD10 в аккумулятор 1.**
DTR	**Преобразовать целое число (32 бита) в значение числа с плавающей точкой (32 бита, IEEE) и сохранить результат в аккумуляторе 1.**
T MD20	**Передать результат в двойное слово памяти MD20.**

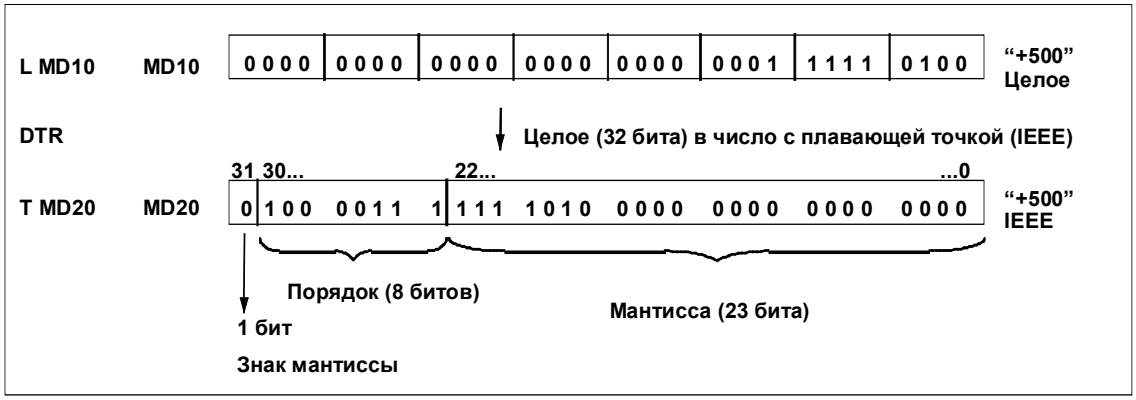


Рис. 12–8. Использование команды DTR для преобразования 32–битового целого числа в 32–битовое число с плавающей точкой в формате IEEE

Обобщение преобразования чисел представлено на рис. 12–13 в конце раздела 12.2.

12.2 Преобразование чисел с плавающей точкой (32 бита) в целые числа (32 бита)

Описание

Вы можете использовать любую из следующих команд для преобразования 32–битового числа с плавающей точкой в формате IEEE, находящегося в аккумуляторе 1, в 32–битовое целое (двойное целое) число. Отдельные команды отличаются друг от друга методом округления.

Мнемоника	Команда	Функция
RND	Округление	Эта команда округляет преобразуемое число до ближайшего целого числа. Если дробная часть преобразуемого числа находится точно между четным и нечетным результатом, команда выбирает четный результат.
RND+	Округление до ближайшего большего двойного целого числа	Эта команда округляет преобразуемое число до наименьшего целого числа, большего или равного преобразуемому числу с плавающей точкой.
RND–	Округление до ближайшего меньшего двойного целого числа	Эта команда округляет преобразуемое число до наибольшего целого числа, меньшего или равного преобразуемому числу с плавающей точкой.
TRUNC	Округление отбрасыванием	Эта команда преобразует целочисленную составляющую числа с плавающей точкой.

Результат преобразования сохраняется в аккумуляторе 1. Если преобразуется число, не являющееся числом с плавающей точкой, или число с плавающей точкой, которое не может быть преобразовано в целое число (32 бита), то преобразование не выполняется и появляется указание о переполнении.

Округление: RND

Команда *Округлить* (RND) преобразует число с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 в 32–битовое целое (двойное целое) число и округляет его до ближайшего целого числа. Если дробная часть преобразуемого числа находится точно между четным и нечетным результатом, то команда выбирает четный результат. Следующий пример программы содержит команду RND. На рис. 12–9 показано, как эта команда работает.

AWL	Объяснение
L MD10	**Загрузить значение числа с плавающей точкой (32 бита, IEEE) из двойного слова памяти MD10 в аккумулятор 1.**
RND	**Преобразовать число с плавающей точкой (32 бита) в целое число (32 бита), округлить до ближайшего целого числа и сохранить результат в аккумуляторе 1.**
T MD20	**Передать результат в двойное слово памяти MD20.**

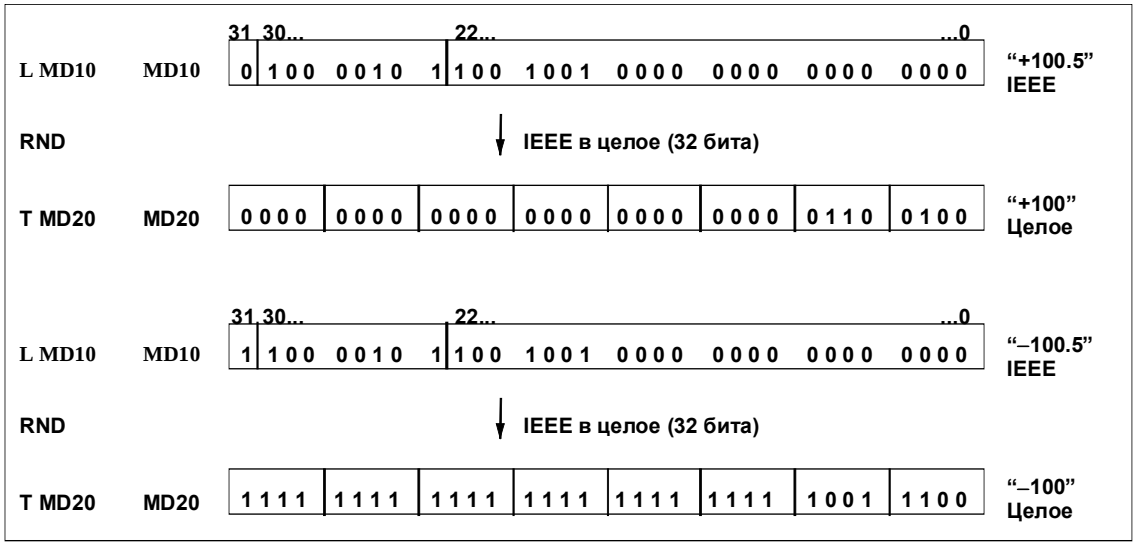


Рис. 12–9. Использование команды RND для преобразования числа с плавающей точкой (32 бита, IEEE) в 32–битовое целое число

Округление до ближайшего большего двойного целого числа: RND+

Команда *Округлить до ближайшего большего двойного целого числа* (RND+) преобразует число с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 в 32–битовое целое (двойное целое) число. Команда округляет преобразуемое число до наименьшего целого числа, большего или равного преобразуемому числу с плавающей точкой. Следующий пример программы содержит команду RND+. На рис. 12–10 показано, как эта команда работает.

AWL	Объяснение
L MD10	**Загрузить значение числа с плавающей точкой (32 бита, IEEE) из двойного слова памяти MD10 в аккумулятор 1.**
RND+	**Преобразовать число с плавающей точкой (32 бита) в целое число (32 бита). Округлить до наименьшего целого числа, большего или равного преобразуемому числу с плавающей точкой, и сохранить результат в аккумуляторе 1.**
T MD20	**Передать результат в двойное слово памяти MD20.**

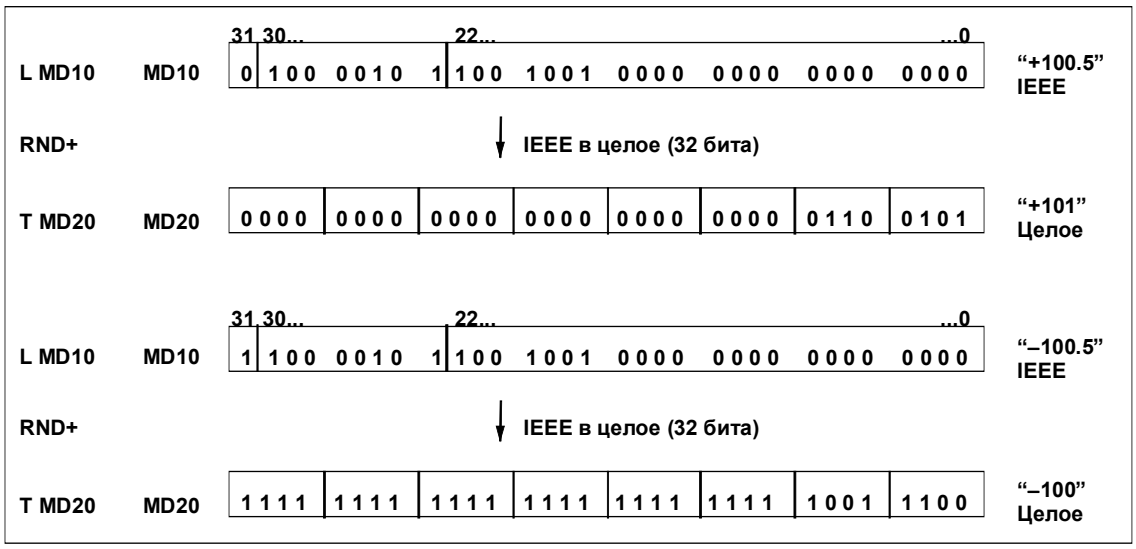


Рис. 12–10. Использование команды RND+ для преобразования числа с плавающей точкой (32 бита, IEEE) в 32–битовое целое число

Округление до ближайшего меньшего двойного целого числа: RND–

Команда *Округлить до ближайшего меньшего двойного целого числа* (RND–) преобразует число с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 в 32–битовое целое (двойное целое) число. Команда округляет преобразуемое число до наибольшего целого числа, меньшего или равного преобразуемому числу с плавающей точкой. Следующий пример программы содержит команду RND–. На рис. 12–11 показано, как эта команда работает.

AWL	Объяснение
L MD10	**Загрузить значение числа с плавающей точкой (32 бита, IEEE) из двойного слова памяти MD10 в аккумулятор 1.**
RND–	**Преобразовать число с плавающей точкой (32 бита) в целое число (32 бита). Округлить до наибольшего целого числа, меньшего или равного преобразуемому числу с плавающей точкой и сохранить результат в аккумуляторе 1.**
T MD20	**Передать результат в двойное слово памяти MD20.**

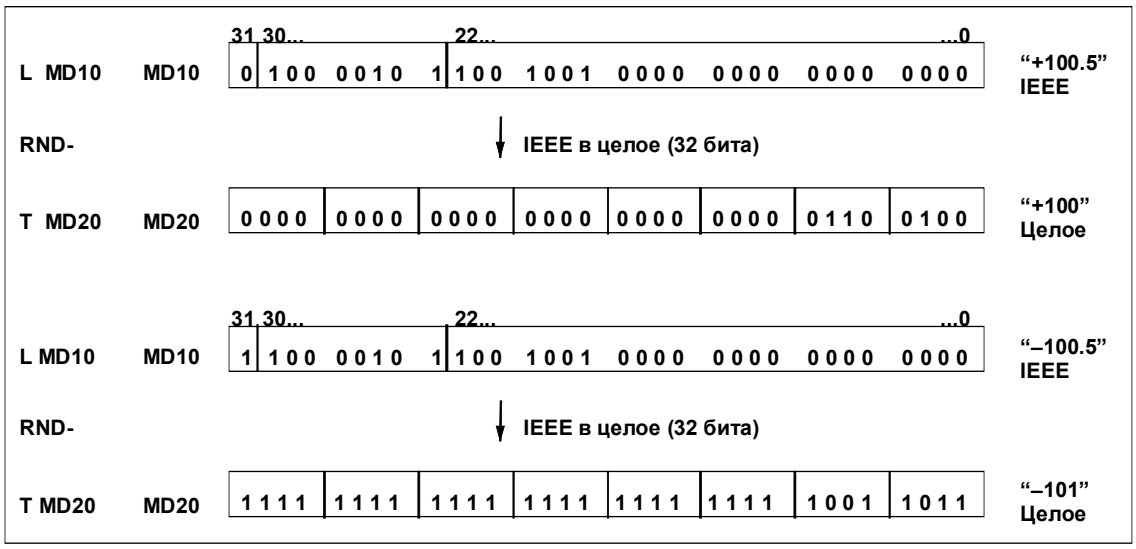


Рис. 12–11. Использование команды RND- для преобразования числа с плавающей точкой (32 бита, IEEE) в 32–битовое целое число

Округление отбрасыванием: TRUNC

Команда *Округлить отбрасыванием* (TRUNC) преобразует число с плавающей точкой (32 бита, IEEE) в аккумуляторе в 32–битовое целое (двойное целое) число. Команда преобразует целочисленную составляющую числа с плавающей точкой. Следующий пример программы содержит команду TRUNC. На рис. 12–12 показано, как эта команда работает.

AWL	Объяснение
L MD10	Загрузить значение числа с плавающей точкой (32 бита, IEEE) из двойного слова памяти MD10 в аккумулятор 1.
TRUNC	Преобразовать число с плавающей точкой (32 бита) в целое число (32 бита). Округлить до целого числа, отбросив дробную часть, следующую за десятичной точкой, и сохранить результат в аккумуляторе 1.
T MD20	Передать результат в двойное слово памяти MD20.

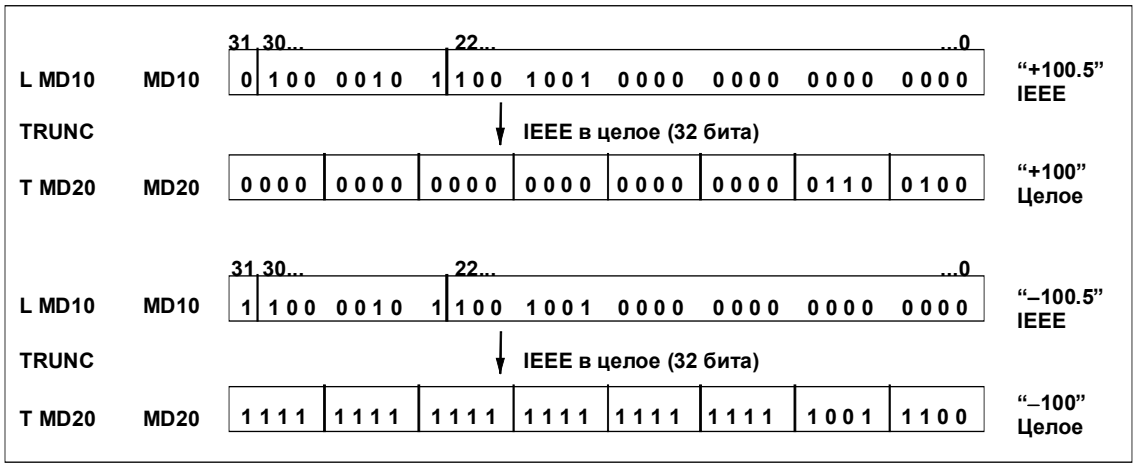


Рис. 12–12. Использование команды TRUNC для преобразования числа с плавающей точкой (32 бита, IEEE) в 32–битовое целое число

Подведение итогов по преобразованию чисел

На рис. 12–13 подводятся итоги по преобразованию и округлению чисел (см. также разделы 12.1 и 12.2).

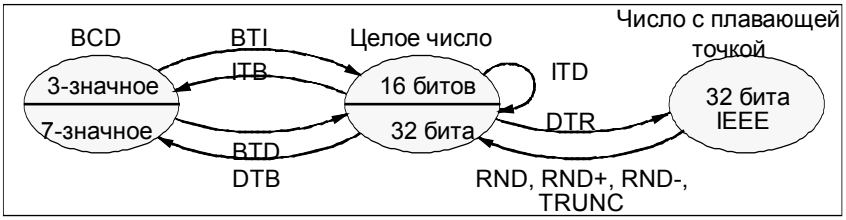


Рис. 12–13. Обзор преобразования и округления чисел

12.3 Изменение последовательности байтов в аккумуляторе 1

Описание

Вы можете использовать следующие команды *Изменить последовательность байтов в аккумуляторе 1* для изменения порядка следования байтов в младшем слове аккумулятора 1 или во всем аккумуляторе:

- Изменить последовательность байтов в аккумуляторе 1, 16 битов (CAW)
- Изменить последовательность байтов в аккумуляторе 1, 32 бита (CAD)

CAW

CAW изменяет последовательность байтов в младшем слове аккумулятора 1 (см. рис. 12–14).

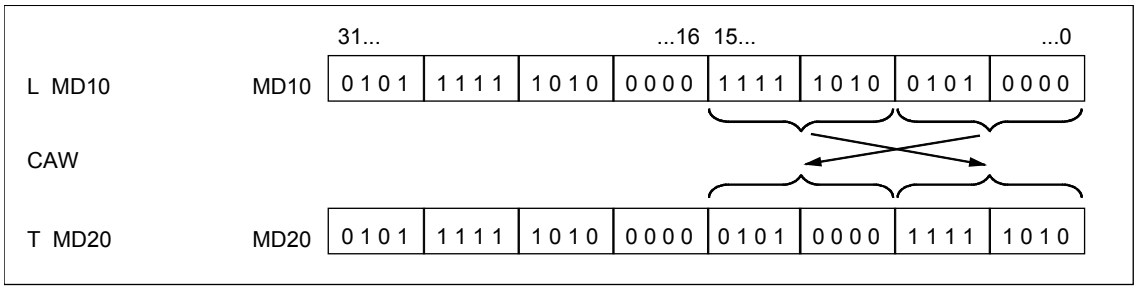


Рис. 12–14. Использование CAW для изменения последовательности байтов в младшем слове аккумулятора 1

CAD

CAD изменяет последовательность байтов во всем аккумуляторе 1 (см. рис. 12–15).

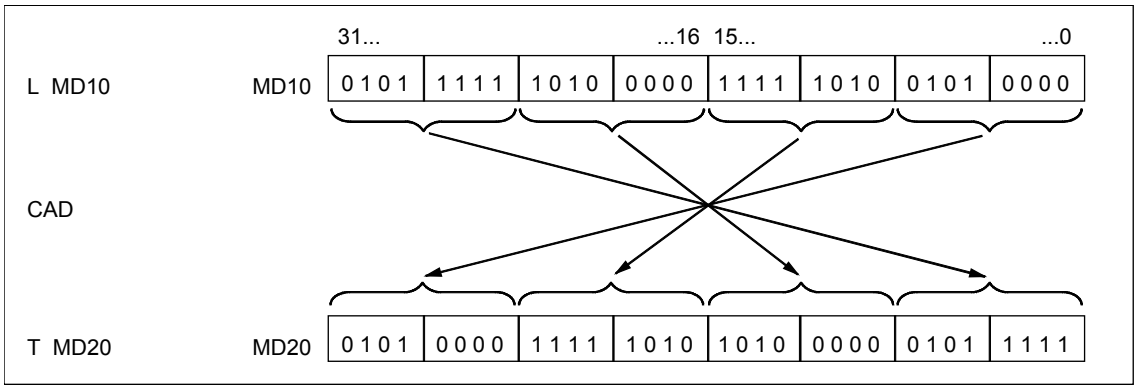


Рис. 12–15. Использование CAD для изменения последовательности байтов в аккумуляторе 1

12.4 Образование дополнений и изменение знака чисел с плавающей точкой

Описание

При образовании дополнения целого числа до единицы в аккумуляторе обращаются отдельные биты, т.е. нули заменяются единицами, а единицы нулями.

При образовании дополнения целого числа до двух в аккумуляторе также обращаются отдельные биты, т.е. нули заменяются единицами, а единицы нулями. Затем к содержимому аккумулятора прибавляется +1. Образование дополнения до двух целого числа соответствует умножению числа на -1. Изменение знака числа с плавающей точкой обращает знаковый бит.

Вы можете использовать одну из следующих команд для образования дополнения целого числа или изменения знака числа с плавающей точкой:

Мнемоника	Команда	Функция
INVI	Дополнение целого числа до единицы	Эта команда образует дополнение до единицы 16–битового числа в младшем слове аккумулятора 1. Результат сохраняется в младшем слове аккумулятора 1 (см. рис. 12–16).
INVD	Дополнение двойного целого числа до единицы	Эта команда образует дополнение до единицы 32–битового числа в аккумуляторе 1. Результат сохраняется в аккумуляторе 1.
NEGI	Дополнение целого числа до двух	Эта команда образует дополнение до двух 16–битового числа в младшем слове аккумулятора 1, т.е. она умножает его на –1. Результат сохраняется в младшем слове аккумулятора 1 (см. рис. 12–17). В слове состояния (см. раздел 3.4) NEGI устанавливает биты CC 1, CC 0, OV и OS.
NEGD	Дополнение двойного целого числа до двух	Эта команда образует дополнение до двух 32–битового числа в аккумуляторе 1, т.е. она умножает его на –1. Результат сохраняется в аккумуляторе 1. В слове состояния (см. раздел 3.4), NEGD устанавливает биты CC 1, CC 0, OV и OS.
NEGR	Изменение знака числа с плавающей точкой	Эта команда изменяет знаковый бит числа с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 (см. рис. 12–18).

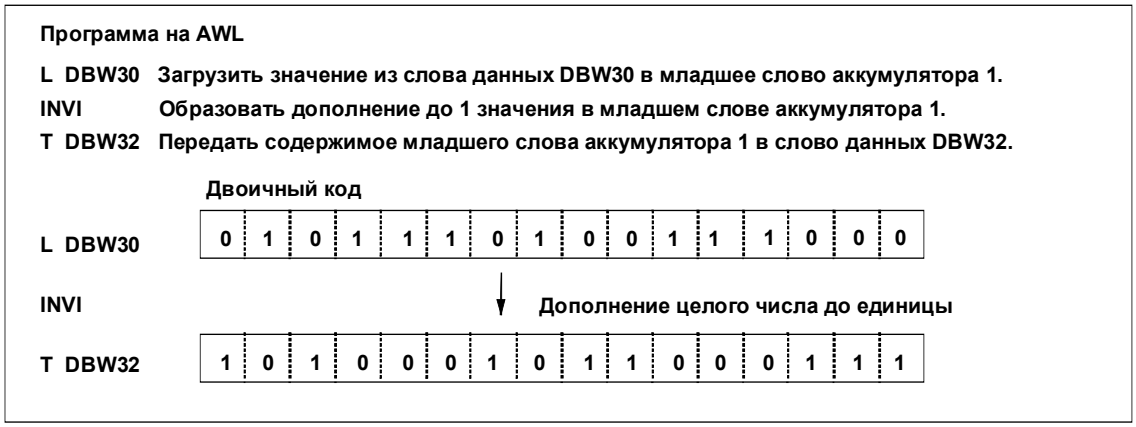


Рис. 12–16. Образование дополнения до единицы 16–битового целого числа

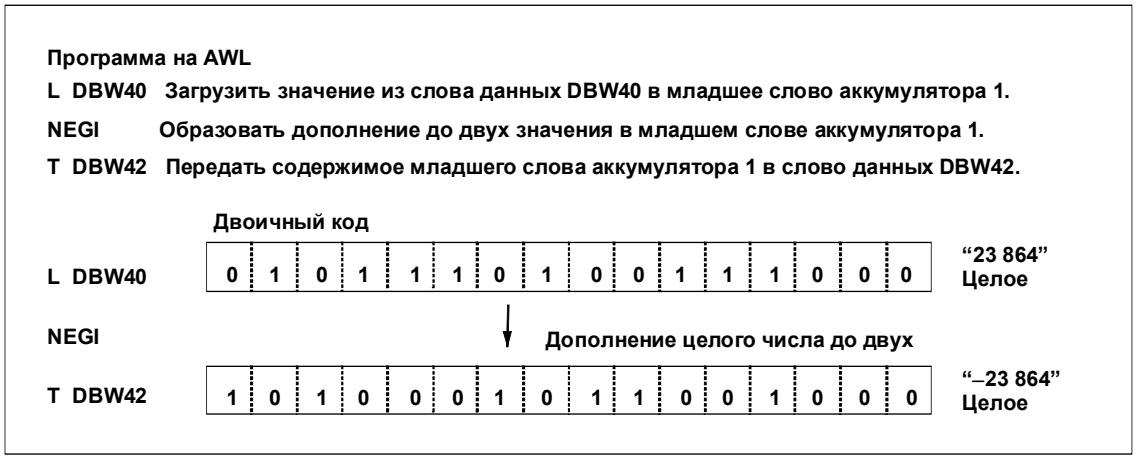


Рис. 12–17. Образование дополнения до двух 16–битового целого числа

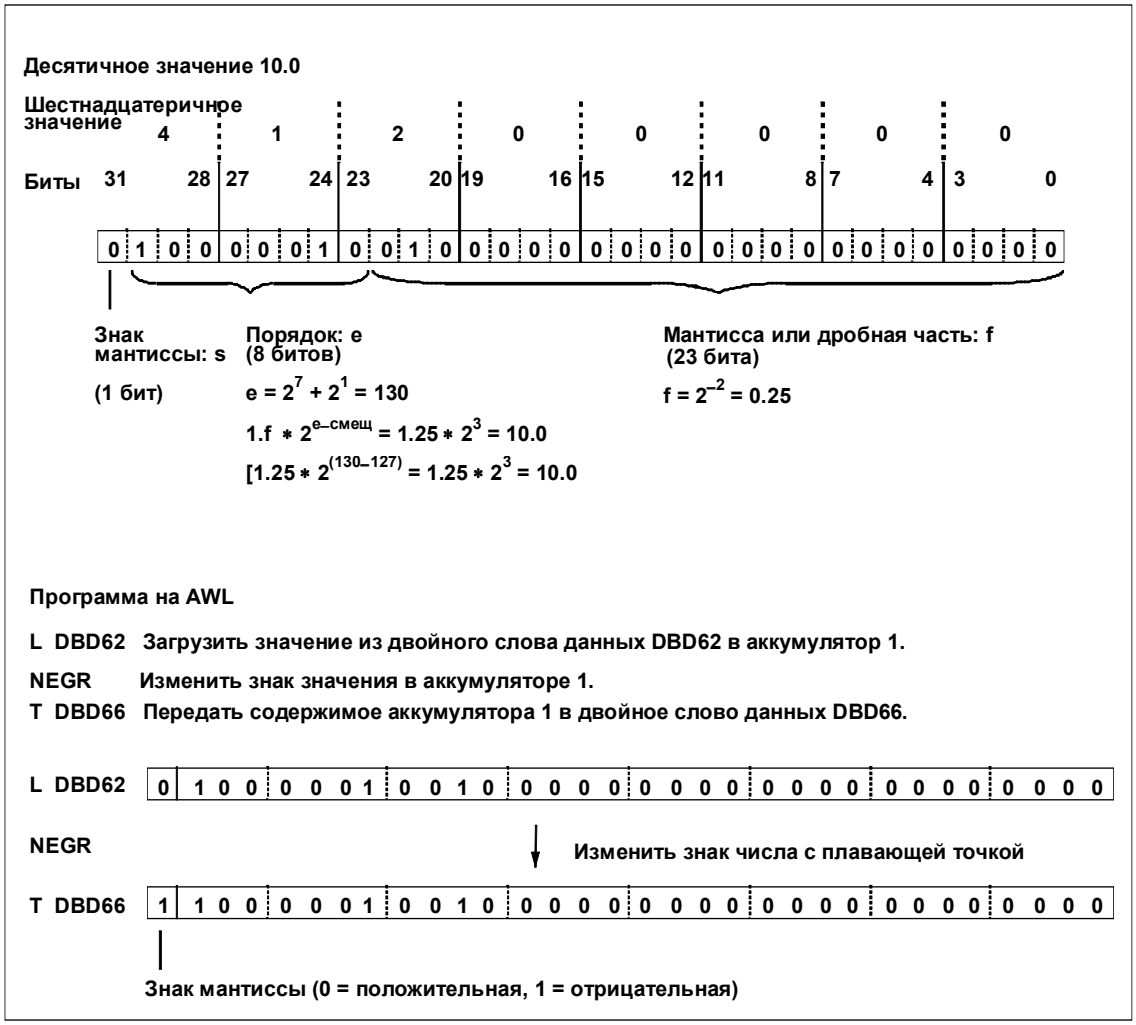


Рис. 12–18. Изменение знака числа с плавающей точкой

13 Логические операции со словами

Обзор главы

13.1 Обзор

Описание

Логические операции со словами комбинируют пары слов (16 битов) или двойных слов (32 бита) бит за битом в соответствии с правилами булевой логики. Каждое слово или двойное слово должно находиться в одном из двух аккумуляторов.

Управление аккумуляторами

Для слов содержимое младшего слова аккумулятора 2 комбинируется с содержимым младшего слова аккумулятора 1. Результат комбинирования сохраняется в младшем слове аккумулятора 1, заменяя его старое содержимое.

Для двойных слов содержимое аккумулятора 2 комбинируется с содержимым аккумулятора 1. Результат комбинирования сохраняется в аккумуляторе 1, заменяя его старое содержимое.

Влияние на биты слова состояния

Если результат логического комбинирования равен 0, то бит CC 1 слова состояния сбрасывается в 0. Если результат логического комбинирования не равен 0, то CC 1 устанавливается в 1. В любом случае биты CC 0 и OV слова состояния сбрасываются в 0.

Имеющиеся команды

Для выполнения логических операций со словами в вашем распоряжении имеются следующие команды:

Мнемоника	Команда	Функция
AW	Поразрядное логическое И со словами	Комбинирует бит за битом два слова в соответствии с таблицей истинности для И
OW	Поразрядное логическое ИЛИ со словами	Комбинирует бит за битом два слова в соответствии с таблицей истинности для ИЛИ
XOW	Поразрядное исключающее ИЛИ со словами	Комбинирует бит за битом два слова в соответствии с таблицей истинности для исключающего ИЛИ
AD	Поразрядное логическое И с двойными словами	Комбинирует бит за битом два двойных слова в соответствии с таблицей истинности для И
OD	Поразрядное логическое ИЛИ с двойными словами	Комбинирует бит за битом два двойных слова в соответствии с таблицей истинности для ИЛИ
XOD	Поразрядное исключающее ИЛИ с двойными словами	Комбинирует бит за битом два двойных слова в соответствии с таблицей истинности для исключающего ИЛИ

Константы в качестве операндов

Команда AW, OW или XOW может использовать в качестве операнда 16–битовую константу. Команда комбинирует содержимое младшего слова аккумулятора 1 с 16–битовой константой.

Команда AD, OD или XOD может использовать в качестве операнда 32–битовую константу.

13.2 Логические операции с 16–битовыми словами

Описание

Команды *Поразрядное логическое И, Поразрядное логическое ИЛИ* и *Поразрядное исключающее ИЛИ со словами* (AW, OW, XOW) поразрядно комбинируют пары слов (16 битов) в соответствии с правилами булевой логики.

Мнемоника	Команда	RLO перед логической операцией	Операнд	Результат в RLO
AW	Поразрядное логическое И со словами	0 0 1 1	0 1 0 1	0 0 0 1
OW	Поразрядное логическое ИЛИ со словами	0 0 1 1	0 1 0 1	0 1 1 1
XOW	Поразрядное исключающее ИЛИ со словами	0 0 1 1	0 1 0 1	0 1 1 0

Связь с аккумуляторами

Для команд, комбинирующих 16–битовые слова, содержимое младшего слова аккумулятора 2 комбинируется с содержимым младшего слова аккумулятора 1. Результат логического комбинирования сохраняется в младшем слове аккумулятора 1, заменяя его старое содержимое. Содержимое старшего слова аккумулятора 1 и обоих слов аккумулятора 2 остается неизменным (см. рис. 13–1).

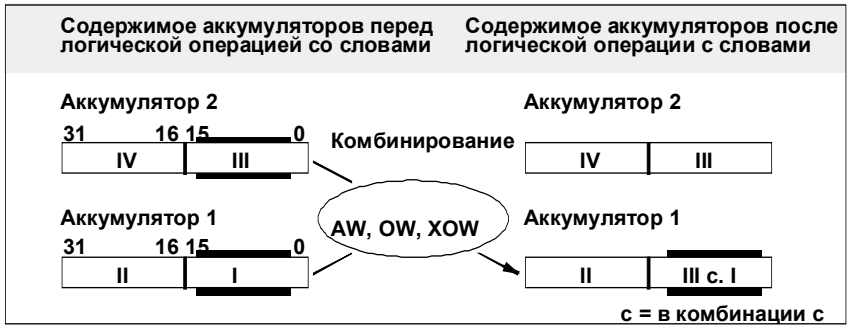


Рис. 13–1. Комбинирование содержимого младших слов аккумуляторов 2 и 1

Пример команды AW

Следующий пример программы содержит команду AW. На рис. 13–2 показано, как эта команда работает.

AWL	Объяснение
L MW10	**Загрузить содержимое слова памяти MW10 в аккумулятор 1.**
L MW20	**Загрузить содержимое слова памяти MW20 в аккумулятор 1. Старое содержимое аккумулятора 1 смещается в аккумулятор 2.**
AW	**Содержимое младшего слова аккумулятора 2 поразрядно комбинируется с содержимым младшего слова аккумулятора 1 в соответствии с таблицей истинности для И. Результат сохраняется в младшем слове аккумулятора 1.**
T MW24	**Передать содержимое аккумулятора 1 в слово памяти MW24.**

Рис. 13–2. Комбинирование двух слов с помощью команды AW

Комбинирование аккумулятора и константы

Команда AW, OW или XOW может использовать в качестве операнда 16–битовую константу. Команда логически комбинирует содержимое младшего слова аккумулятора 1 с 16–битовой константой, указанной в команде. Результат комбинирования сохраняется в младшем слове аккумулятора 1. Аккумулятор 2 и старшее слово аккумулятора 1 остаются неизменными (см. рис. 13–3).

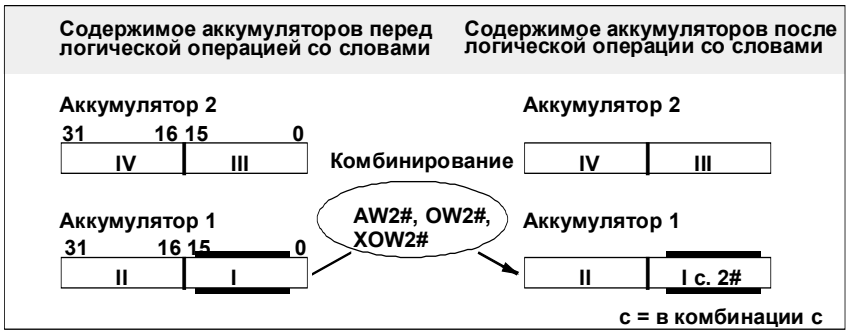


Рис. 13–3. Комбинирование младшего слова аккумулятора 1 с 16–битовой константой

Пример команды AW с константой

Следующий пример программы содержит команду AW, выполняющую логическое комбинирование с 16–битовой константой, указанной в команде. На рис. 13–4 показано, как эта команда работает.

AWL	Объяснение
L MW10	**Загрузить содержимое слова памяти MW10 в аккумулятор 1.**
AW 2#1010_1010_0101_0101	**Содержимое младшего слова аккумулятора 1 поразрядно комбинируется в соответствии с таблицей истинности для И с константой 1010_1010_0101_0101. Результат сохраняется в младшем слове аккумулятора 1.**
T MW24	**Передать содержимое аккумулятора 1 в слово памяти MW24.**

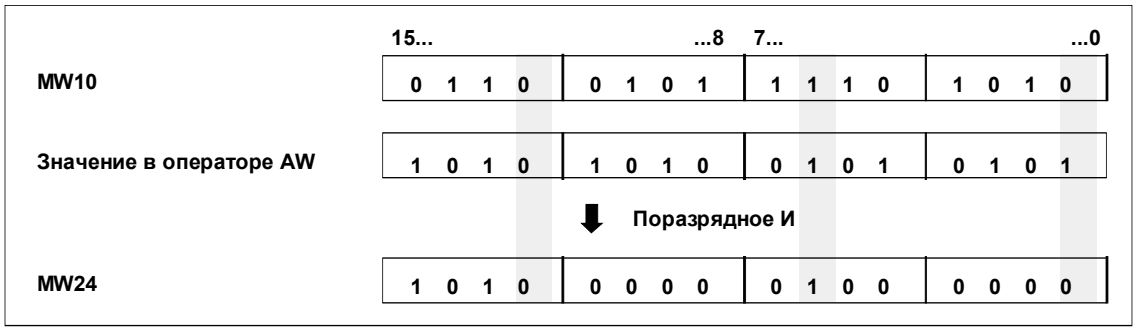


Рис. 13–4. Использование команды AW с 16–битовой константой

13.3 Логические операции с 32–битовыми словами

Описание

Команды *Поразрядное логическое И, Поразрядное логическое ИЛИ* и *Поразрядное исключающее ИЛИ с двойными словами* (AD, OD, XOD) поразрядно комбинируют пары слов (32 бита) в соответствии с правилами булевой логики.

Мнемоника	Команда	RLO перед логической операцией	Операнд	Результат в RLO
AD	Поразрядное логическое И с двойными словами	0 0 1 1	0 1 0 1	0 0 0 1
OD	Поразрядное логическое ИЛИ с двойными словами	0 0 1 1	0 1 0 1	0 1 1 1
XOD	Поразрядное исключающее ИЛИ с двойными словами	0 0 1 1	0 1 0 1	0 1 1 0

Связь с аккумуляторами

Для команд, комбинирующих двойные слова, содержимое аккумулятора 2 комбинируется с содержимым аккумулятора 1. Результат логического комбинирования сохраняется в аккумуляторе 1 , заменяя его старое содержимое. Содержимое аккумулятора 2 остается неизменным (см. рис. 13–5).

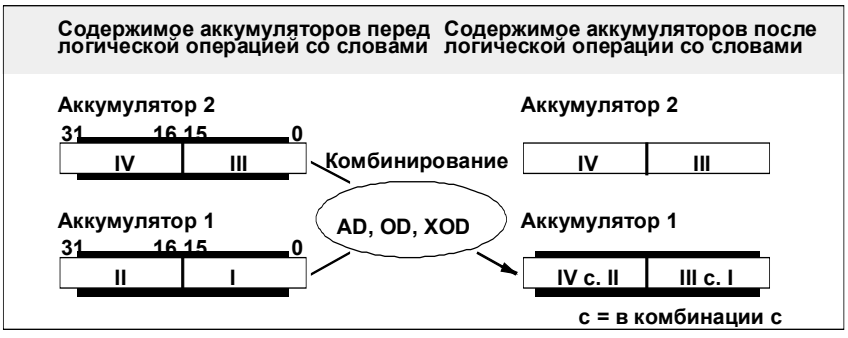


Рис. 13–5. Комбинирование содержимого аккумуляторов 2 и 1

Пример команды AD

Следующий пример программы содержит команду AW. На рис. 13–6 показано, как эта команда работает.

AWL	Объяснение
L MD10	**Загрузить содержимое двойного слова памяти MD10 в аккумулятор 1.**
L MD20	**Загрузить содержимое двойного слова памяти MD20 в аккумулятор 1. Старое содержимое аккумулятора 1 сдвигается в аккумулятор 2.**
AD	**Содержимое аккумулятора 2 поразрядно комбинируется с содержимым аккумулятора 1 в соответствии с таблицей истинности для И. Результат сохраняется в аккумуляторе 1.**
T MD24	**Передать содержимое аккумулятора 1 в двойное слово памяти MD24.**

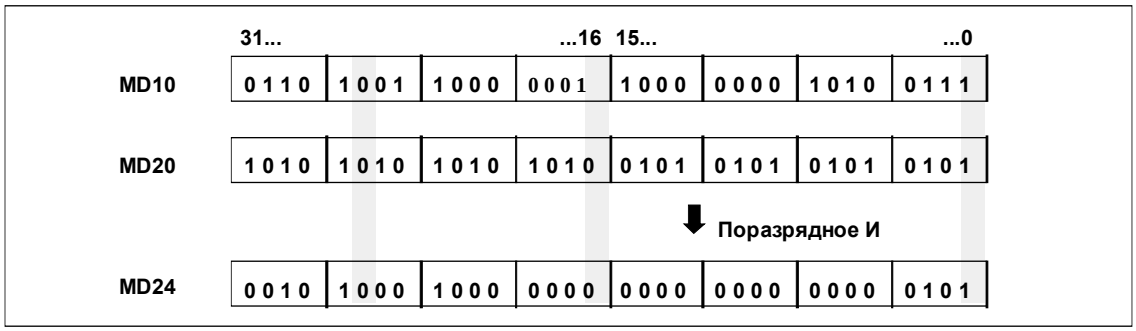


Рис. 13–6. Использование команды AD

Комбинирование аккумулятора и константы

Команда AD, OD или XOD может использовать в качестве операнда 32–битовую константу. Команда логически комбинирует содержимое аккумулятора 1 с 32–битовой константой, указанной в команде. Результат комбинирования сохраняется в аккумуляторе 1. Аккумулятор 2 остается неизменным (см. рис. 13–7).

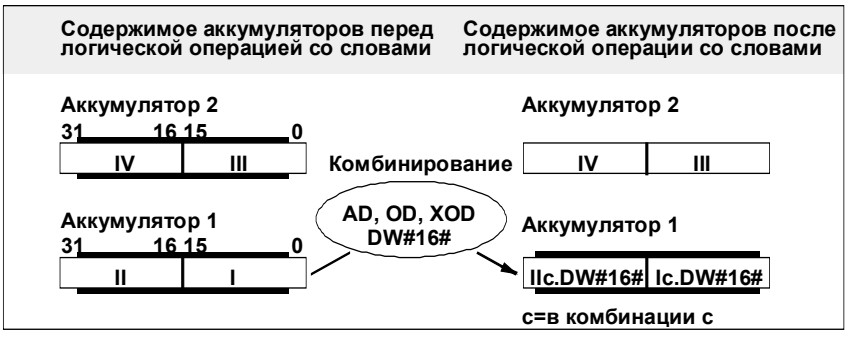


Рис. 13–7. Комбинирование аккумулятора 1 с 32–битовой константой

Пример команды AD с константой

Следующий пример программы содержит команду AD, выполняющую логическое комбинирование с 32–битовой константой, указанной в команде. На рис. 13–8 показано, как эта команда работает.

AWL	Объяснение
L MD10	**Загрузить содержимое двойного слова памяти MD10 в аккумулятор 1.**
AD DW#16#AAAA_5555	**Содержимое аккумулятора 1 поразрядно логически комбинируется в соответствии с таблицей истинности для И с константой DW#16#AAAA_5555. Результат сохраняется в аккумуляторе 1.**
T MD24	**Передать содержимое аккумулятора 1 в двойное слова памяти MD24.**

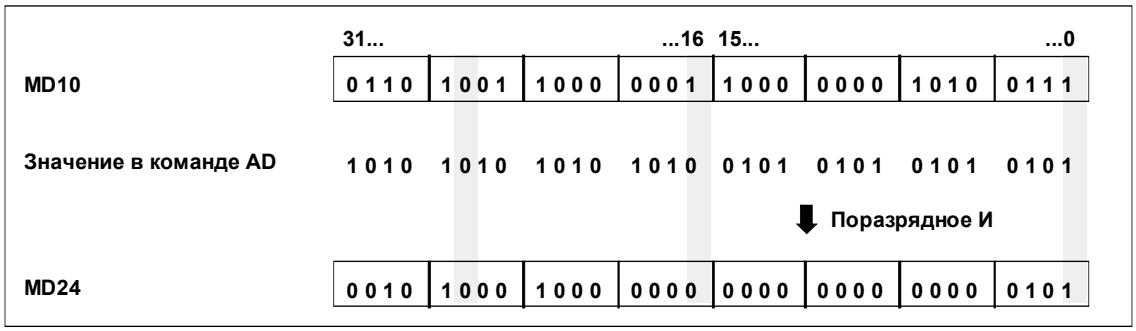


Рис. 13–8. Использование команды AD с 32–битовой константой

14 Команды сдвига и циклического сдвига

Обзор главы

14.1 Команды сдвига

Описание

С помощью команд сдвига вы можете побитно сдвинуть налево или направо содержимое младшего слова аккумулятора 1 или содержимое всего аккумулятора. Сдвиг на n битов влево умножает содержимое аккумулятора на 2^n, сдвиг на n битов вправо делит содержимое аккумулятора на 2^n. Таким образом, если, например, вы сдвигаете двоичный эквивалент десятичного числа 3 на 3 бита влево, то в аккумуляторе получается двоичный эквивалент десятичного числа 24. Если вы сдвигаете двоичный эквивалент десятичного числа 16 на 2 бита вправо, то в аккумуляторе получается двоичный эквивалент десятичного числа 4.

Число, следующее за командой сдвига или записанное в младшем байте младшего слова аккумулятора 2, указывает, на сколько битов должен быть произведен сдвиг. Разряды, освобождаемые командой сдвига, заполняются нулями или сигнальным состоянием знакового бита (0 - для положительных чисел, 1 - для отрицательных). Последний сдвинутый бит загружается в бит CC 1 слова состояния. Биты CC 0 и OV слова состояния сбрасываются в 0. Вы можете анализировать бит CC 1 с помощью команд перехода.

Команды сдвига безусловны, т.е. их выполнение не зависит ни от каких определенных условий. Они не влияют на результат логической операции.

Команды сдвига: числа без знака

Следующие команды сдвигают побитно содержимое младшего слова аккумулятора 1 влево или вправо:

- Сдвинуть слово влево (SLW, 16 битов)
- Сдвинуть слово вправо (SRW, 16 битов)

Следующие команды сдвигают все содержимое аккумулятора 1 побитно влево или вправо:

- Сдвинуть двойное слово влево (SLD, 32 бита)
- Сдвинуть двойное слово вправо (SRD, 32 бита)

Во всех случаях освободившиеся битовые разряды заполняются нулями.

Сдвинуть влево слово: SLW

Следующий пример программы и рис. 14–1 показывают, как работает команда SLW. В табл. 14–1 вы найдете обзор всех команд сдвига.

AWL	Объяснение
L MW10	**Загрузить содержимое слова памяти MW10 в младшее слово аккумулятора 1.**
SLW 6	**Сдвинуть биты в младшем слове аккумулятора 1 на 6 разрядов влево.**
T MW20	**Передать содержимое младшего слова аккумулятора 1 в слово памяти MW20.**

Рис. 14–1. Сдвиг битов младшего слова аккумулятора 1 на 6 разрядов влево

Сдвинуть вправо двойное слово (32 бита): SRD

Следующий пример программы и рис. 14–2 показывают, как работает команда SRD. В таблице 14–1 вы найдете обзор всех команд сдвига.

AWL	Объяснение
L +3	Загрузить значение +3 в аккумулятор 1.
L MD10	Загрузить содержимое двойного слова памяти MD10 в аккумулятор 1. Старое содержимое аккумулятора 1 (+3) сдвигается в аккумулятор 2.
SRD	Сдвинуть биты в аккумуляторе 1 на три разряда вправо.
T MD20	Передать содержимое аккумулятора 1 в двойное слово памяти MD20.

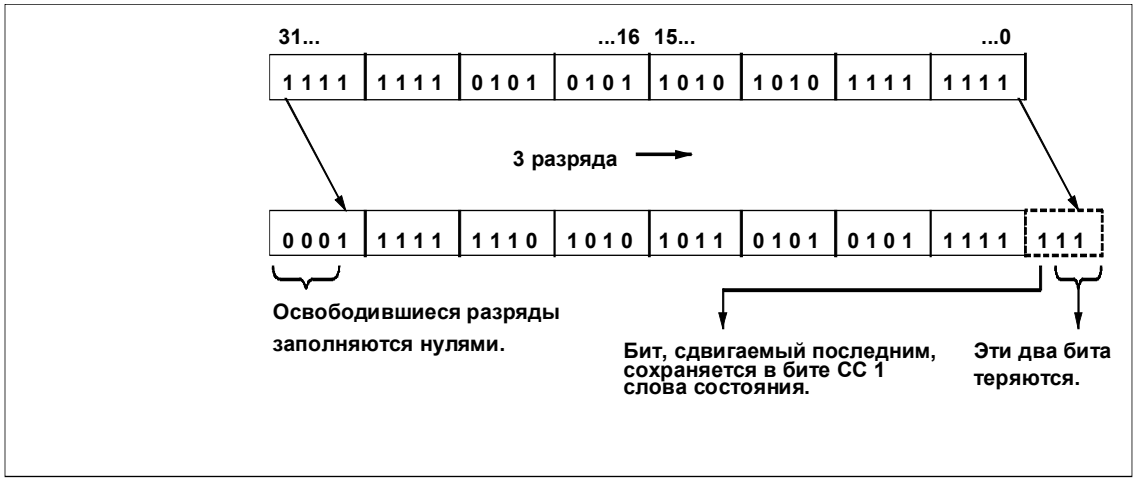


Рис. 14–2. Сдвиг битов аккумулятора 1 на три разряда вправо

Операции сдвига: числа со знаком

Команда *Сдвинуть целое число со знаком* (SSI, 16 битов) сдвигает содержимое младшего слова аккумулятора 1 побитно вправо, включая знак, как описано в обзоре этой главы.

Команда *Сдвинуть двойное целое число со знаком* (SSD, 32 бита) сдвигает все содержимое аккумулятора побитно вправо, включая знак.

Знаковый бит копируется в освободившиеся битовые разряды.

Сдвинуть целое число со знаком: SSI

Следующий пример программы и рис. 14–3 показывают, как работает команда SSI. В таблице 14–1 вы найдете обзор всех команд сдвига.

AWL	Объяснение
L MW10	**Загрузить содержимое слова памяти MW10 в младшее слово аккумулятора 1.**
SSI 4	**Сдвинуть биты в младшем слове аккумулятора 1, включая знак, на четыре разряда вправо.**
T MW20	**Передать содержимое младшего слова аккумулятора 1 в слово памяти MW20.**

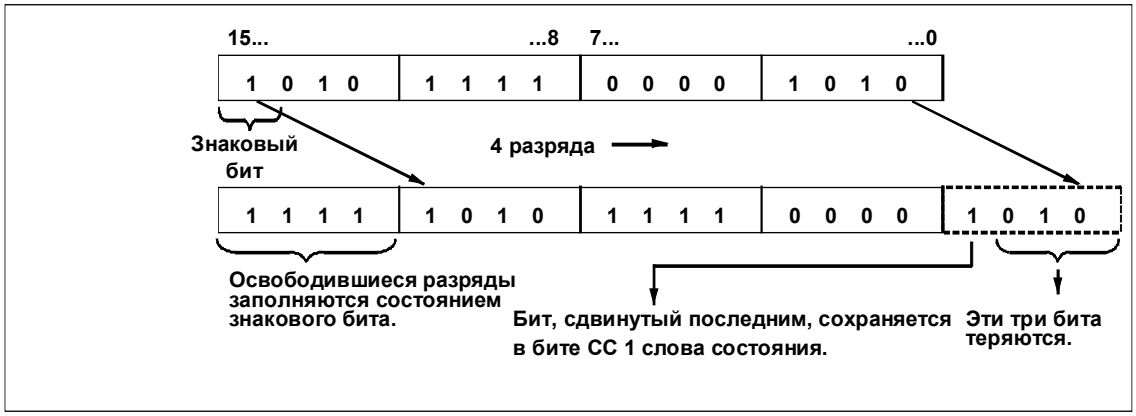


Рис. 14–3. Сдвиг битов младшего слова аккумулятора 1 на четыре разряда вправо со знаком

Таблица 14–1. Обзор команд сдвига

Команда	Задействованная область	Направление	Указание количества разрядов для сдвига	Заполнитель для освободившихся разрядов	Диапазон сдвига
SLW n	Младшее слово аккумулятора 1	влево	В операторе	0	n=0 до 15
SLW	Младшее слово аккумулятора 1	влево	В младшем байте младшего слова аккумулятора 2	0	0 до 255[2)]
SLD n	Аккумулятор 1	влево	В операторе	0	n=0 до 32
SLD	Аккумулятор 1	влево	В младшем байте младшего слова аккумулятора 2	0	0 до 255[3)]
SRW n	Младшее слово аккумулятора 1	вправо	В операторе	0	n=0 до 15
SRW	Младшее слово аккумулятора 1	вправо	В младшем байте младшего слова аккумулятора 2	0	0 до 255[2)]
SRD n	Аккумулятор 1	вправо	В операторе	0	n=0 до 32
SRD	Аккумулятор 1	вправо	В младшем байте младшего слова аккумулятора 2	0	0 до 255[3)]
SSI n	Младшее слово аккумулятора 1	вправо	В операторе	Знаковый бит	n=0 до 15
SSI	Младшее слово аккумулятора 1	вправо	В младшем байте младшего слова аккумулятора 2	Знаковый бит	0 до 255[4)]
SSD n	Аккумулятор 1	вправо	В операторе	Знаковый бит	n=0 до 32
SSD	Аккумулятор 1	вправо	В младшем байте младшего слова аккумулятора 2	Знаковый бит	0 до 255[5)]

[1] Если количество разрядов, на которые необходимо произвести сдвиг или циклический сдвиг, равно 0, то команда выполняется как NOP.

[2] Для величин сдвига, больших 16, результат функции сдвига равен W#16#0000 и CC 1 = 0.

[3] Для величин сдвига, больших 32, результат функции сдвига равен DW#16#0000_0000 и CC 1 = 0.

[4] Для величин сдвига, больших 15, результат функции сдвига равен W#16#0000 и CC 1 = 0 или W#16#FFFF и CC 1 = 1 в зависимости от знака (0 или 1).

[5] Для величин сдвига, больших 31, результат функции сдвига равен DW#16#0000_0000 (CC 1 = 0) или DW#16#FFFF_FFFF (CC 1 = 1) в зависимости от знака числа битов, подлежащих сдвигу.

14.2 Команды циклического сдвига

Описание

С помощью команд циклического сдвига вы можете побитно циклически сдвигать все содержимое аккумулятора 1 направо или налево. Команды циклического сдвига выполняют функции, подобные описанным в разделе 14.1 функциям сдвига. Однако, освободившиеся разряды заполняются сигнальными состояниями битов, выдвигаемых из аккумулятора.

Число, следующее за командой циклического сдвига, или значение в младшем байте младшего слова аккумулятора 2 указывает, на сколько разрядов должен быть произведен циклический сдвиг.

В зависимости от команды циклический сдвиг выполняется через бит CC 1 в слове состояния (см. раздел 2.2). Бит CC 0 в слове состояния сбрасывается в 0.

В вашем распоряжении имеются следующие команды циклического сдвига:

- Сдвинуть циклически влево двойное слово (RLD)
- Сдвинуть циклически вправо двойное слово (RRD)
- Сдвинуть циклически влево аккумулятор 1 через бит CC 1 (RLDA)
- Сдвинуть циклически вправо аккумулятор 1 через бит CC 1 (RRDA)

Если число битов, на которое нужно произвести циклический сдвиг, равно 0, то команда выполняется как пустая операция (NOP).

Таблица 14–2. Обзор команд циклического сдвига

Команда	Сдвигать через CC 1?	Направление	Указание количества разрядов для сдвига	Диапазон сдвига
RLD n	нет	влево	В операторе	n=0 до 32
RLD	нет	влево	В младшем байте младшего слова аккумулятора2	0 до 255
RRD	нет	вправо	В операторе	0 до 32
RRD	нет	вправо	В младшем байте младшего слова аккумулятора2	0 до 255
RLDA	да	влево		1 (фиксировано)
RRDA	да	вправо		1 (фиксировано)

Сдвинуть циклически влево двойное слово: RLD

Табл. 14–2 дает обзор всех команд циклического сдвига. Следующий пример программы и рис. 14–4 показывают, как работает команда RLD.

AWL	Объяснение
L MD10	**Загрузить содержимое двойного слова памяти MD10 в аккумулятор 1.**
RLD 3	**Сдвинуть циклически биты в аккумуляторе 1 на три разряда влево.**
T MD20	**Передать содержимое аккумулятора 1 в двойное слово памяти MD20.**

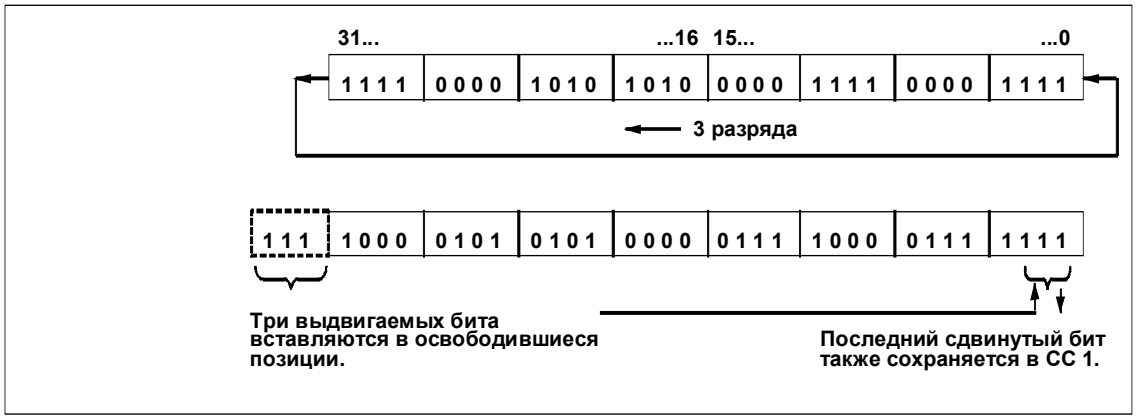


Рис. 14–4. Циклический сдвиг битов аккумулятора 1 на три разряда влево

Сдвинуть циклически вправо двойное слово: RRD

Следующий пример программы и рис. 14–5 показывают, как работает команда RRD.

AWL	Объяснение
L +3	Загрузить значение +3 в аккумулятор 1.
L MD10	Загрузить содержимое двойного слова памяти MD10 в аккумулятор 1. Старое содержимое аккумулятора 1 (+3) сдвигается в аккумулятор 2.
RRD	Сдвинуть циклически биты в аккумуляторе 1 на три разряда вправо.
T MD20	Передать содержимое аккумулятора 1 в двойное слово памяти MD20.

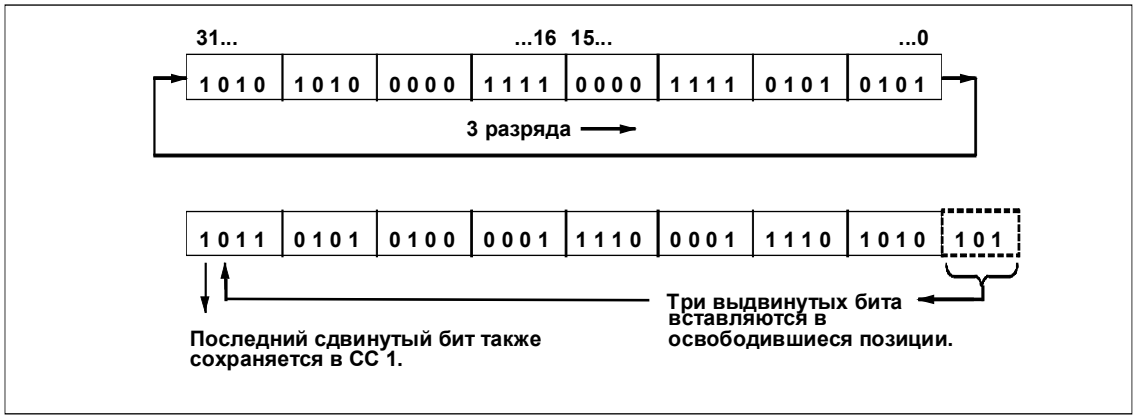


Рис. 14–5. Циклический сдвиг битов аккумулятора 1 на три разряда вправо

Циклический сдвиг аккумулятора 1 влево через CC 1: RLDA

На рис. 14–6 показан пример работы команды RLDA.

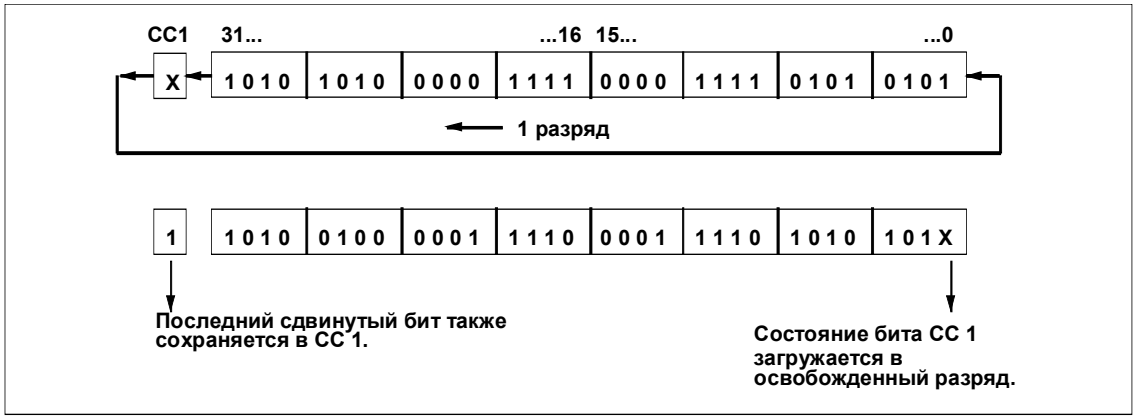


Рис. 14–6. Циклический сдвиг аккумулятора 1 на один разряд влево через бит CC 1 слова состояния

Циклический сдвиг аккумулятора 1 вправо через CC 1: RRDA

Команда RRDA работает аналогично команде RLDA; единственное отличие состоит в том, что сдвиг происходит в другом направлении.

15 Операции с блоками данных

Обзор главы

15.1 Открытие блоков данных

Описание

Вы можете использовать команду *Открыть блок данных* (OPN) для открытия блока данных как совместно используемого (глобального) блока данных или как экземплярного блока данных. Одновременно в программе могут быть открыты один глобальный блок данных и один экземплярный блок данных.

Формат адресации

Таблицы 15–1 и 15–2 перечисляют операнды и их диапазоны для команды OPN.

Таблица 15–1. Операнды команды *Открыть блок данных* OPN

Область блока данных	**Максимальный диапазон адресов в соответствии с видом адресации**	
	прямая	**косвенная через память**
DB DI	от 1 до 65 535	[DBW] от 1 до 65 534 [DIW] [LW] [MW]

Таблица 15–2. Операнды команды *Открыть блок данных* OPN, передаваемые как параметры

Вид открываемого DB	**Формат адресного параметра**
DBpara	BLOCK_DB
DIpara	

15.2 Обмен регистрами блоков данных

Описание

Вы можете использовать команду *Обменять глобальный DB и экземплярный DB* (CDB) для обмена регистрами блоков данных. Глобальный блок данных становится экземплярным блоком данных и наоборот.

15.3 Загрузка длин и номеров блоков данных

Описание

Вы можете использовать следующие команды для загрузки длины (в байтах) или номера глобального или экземплярного блока данных в аккумулятор 1:

- Загрузить длину глобального DB в аккумулятор 1 (L DBLG)
- Загрузить номер глобального DB в аккумулятор 1 (L DBNO)
- Загрузить длину экземплярного DB в аккумулятор 1 (L DILG)
- Загрузить номер экземплярного DB в аккумулятор 1 (L DINO)

Примеры

Следующий пример программы иллюстрирует, как можно использовать команду L DBLG для перехода на метку ERR, если длина блока данных равна 50 байтам или больше. Оператор на метке ERR вызывает FC10, который вы запрограммировали как соответствующую реакцию на равенство или превышение длины блока данных 50 байтов.

AWL			Объяснение
	OPN	**DB40**	**Открыть глобальный блок данных DB40.**
	L	**DBLG**	**Загрузить длину открытого блока данных в аккумулятор 1.**
	L	**+50**	**Загрузить целое число +50 в аккумулятор 1. Старое содержимое аккумулятора 1 (длина открытого блока данных) перемещается в аккумулятор 2.**
	>=I		**Сравнить содержимое аккумулятора 2 (длину открытого блока данных) с содержимым аккумулятора 1 (+50).**
	JC	**ERR**	**Если длина блока данных больше или равна +50, перейти на метку ERR. Если длина блока данных меньше +50, перейти к следующей команде.**
	A **=**	**I 0.0** **M 1.0**	**Программа продолжает работу с команды И.**
	BEU		**Завершить текущий блок независимо от результата логической операции.**
ERR:	**CALL**	**FC10**	**FC10 содержит реакцию для случая, когда длина открытого в данный момент блока (DB40) больше или равна 50 байтам.**

Следующий пример программы показывает, как применить команду L DBNO, чтобы проверить, попадает ли открытый в данный момент в вашей программе блок в определенный диапазон блоков данных, например, от DB190 до DB250.

AWL			Объяснение
	L	**DBNO**	**Загрузить номер открытого в данный момент блока данных в аккумулятор 1.**
	L	**+190**	**Загрузить целое число +190 как нижнюю границу в аккумулятор 1. Старое содержимое аккумулятора 1 (номер открытого блока данных) перемещается в аккумулятор 2.**
	<I		**Сравнить содержимое аккумулятора 2 (номер открытого блока данных) с содержимым аккумулятора 1 (+190).**
	JC	**ERR**	**Если номер блока данных меньше +190, перейти на метку ERR. Если номер блока данных не меньше +190, перейти к следующему оператору.**
	L	**DBNO**	**Загрузить номер открытого блока данных в аккумулятор 1.**
	L	**+250**	**Загрузить целое число +250 как верхнюю границу в аккумулятора 1. Старое содержимое аккумулятора 1 (номер открытого блока данных) перемещается в аккумулятор 2.**
	>I		**Сравнить содержимое аккумулятора 2 (номер открытого блока данных) с содержимым аккумулятора 1 (+250).**
	JC	**ERR**	**Если номер блока данных больше +250, перейти на метку ERR. Если номер блока данных не больше +250, перейти к следующему оператору.**
	SET		**Установить RLO в 1.**
	=	**M 1.0**	**Присвоить RLO биту памяти (меркеру) M 1.0.**
	BEU		**Завершить текущий блок независимо от результата логической операции.**
ERR:	**CALL**	**FC10**	**FC10 содержит реакцию для случая, когда номер открытого в данный момент блока не попадает в диапазон между 190 и 250.**

16 Команды перехода

Обзор главы

16.1 Обзор

Команды

Вы можете использовать следующие команды перехода и организации циклического выполнения для управления ходом выполнения программы, позволяя ей прерывать линейную процедуру выполнения, чтобы возобновить обработку с другого места. Операндом команды перехода и циклического исполнения является метка.

Команда	Объяснение
Команды безусловного перехода	
JU	Перейти безусловно
JL	Перейти по списку
Команды условного перехода с условием, зависящим от RLO	
JC	Перейти, если RLO = 1
JCN	Перейти, если RLO = 0
JCB	Перейти, если RLO = 1 с сохранением RLO в BR
JNB	Перейти, если RLO = 0 с сохранением RLO в BR
Команды условного перехода с условием, зависящим от BR или OV/OS	
JBI	Перейти, если BR = 1
JNBI	Перейти, если BR = 0
JO	Перейти, если OV = 1
JOS	Перейти, если OS = 1
Команды условного перехода с условием, зависящим от результата операции, закодированного в CC 1 и CC 0	
JZ	Перейти, если результат = 0
JN	Перейти, если результат <> 0
JP	Перейти, если результат > 0
JM	Перейти, если результат < 0
JMZ	Перейти, если результат <= 0
JPZ	Перейти, если результат >= 0
JUO	Перейти, если результат недействителен
Циклическое управление	
LOOP	Перейти, если содержимое аккумулятора 1 > 0

Хотя команды Главного управляющего реле (Master Control Relay, MCR) также управляют исполнением программы, они в эту главу не включены. За информацией о командах MCR обратитесь к разделам 17.4 и 17.5.

Метка перехода

Метка перехода может служить операндом команды перехода или служить обозначением цели команды перехода. Она состоит не более чем из 4 символов. Первый символ должен быть буквой, а остальные символы могут быть буквами или числами (напр., SEG3). В качестве обозначения цели метка перехода должна, кроме того, завершаться двоеточием. В этом случае за меткой перехода всегда должна следовать команда (напр., SEG3: NOP 0).

16.2 Команды безусловного перехода

Описание

Вы можете использовать следующие команды перехода для безусловного прерывания нормальной последовательности выполнения своей программы:

- Перейти безусловно (JU)
- Перейти по списку (JL)

Перейти безусловно: JU

Команда *Перейти безусловно* (JU) прерывает в вашей программе нормальную последовательность логического управления и заставляет вашу программу перейти на метку (операнд команды JU). Метка отмечает точку, с которой программа должна продолжить свою работу. Переход выполняется независимо от каких-либо условий.

Перейти по списку: JL

Команда *Перейти по списку* (JL) представляет собой распределитель переходов. За ней следует ряд безусловных переходов на метки (см. рис. 16–1). Доступ к списку происходит через аккумулятор 1.

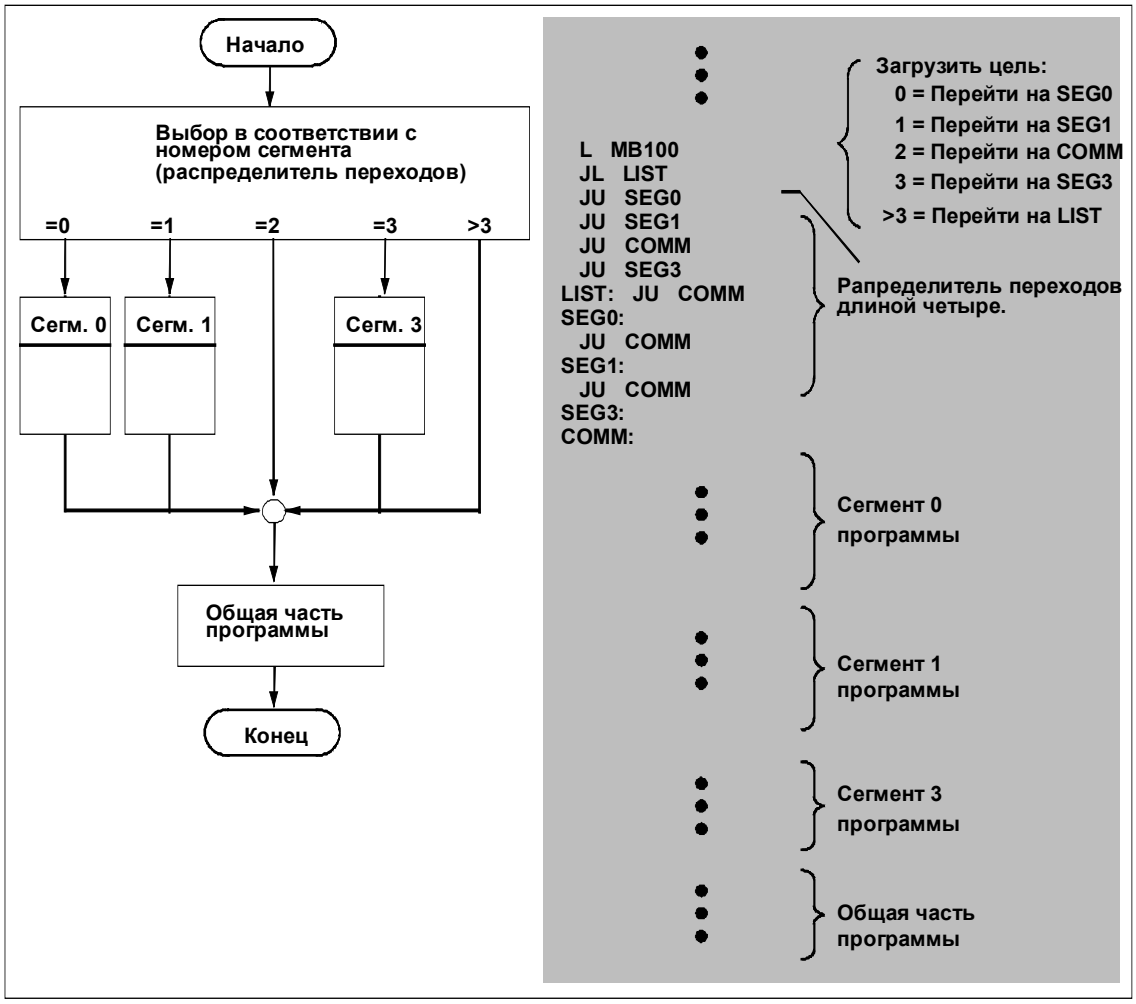


Рис. 16–1. Управление ходом выполнения программы с помощью команды *Перейти по списку* JL

16.3 Команды условного перехода, зависящие от результата логической операции

Описание

Следующие команды перехода прерывают последовательность выполнения программы на основе результата логической операции (RLO), образованного предыдущим оператором:

- Перейти, если RLO = 1 (JC)
- Перейти, если RLO = 0 (JCN)
- Перейти, если RLO = 1 с сохранением RLO в BR (JCB): RLO сохраняется в бите BR слова состояния.
- Перейти, если RLO = 0 с сохранением RLO в BR (JNB): RLO сохраняется в бите BR слова состояния.

Следующие биты в слове состояния записываются независимо от исполнения перехода:

OR :=0
STA :=1
RLO :=1

$\overline{\text{FC}}$:=0

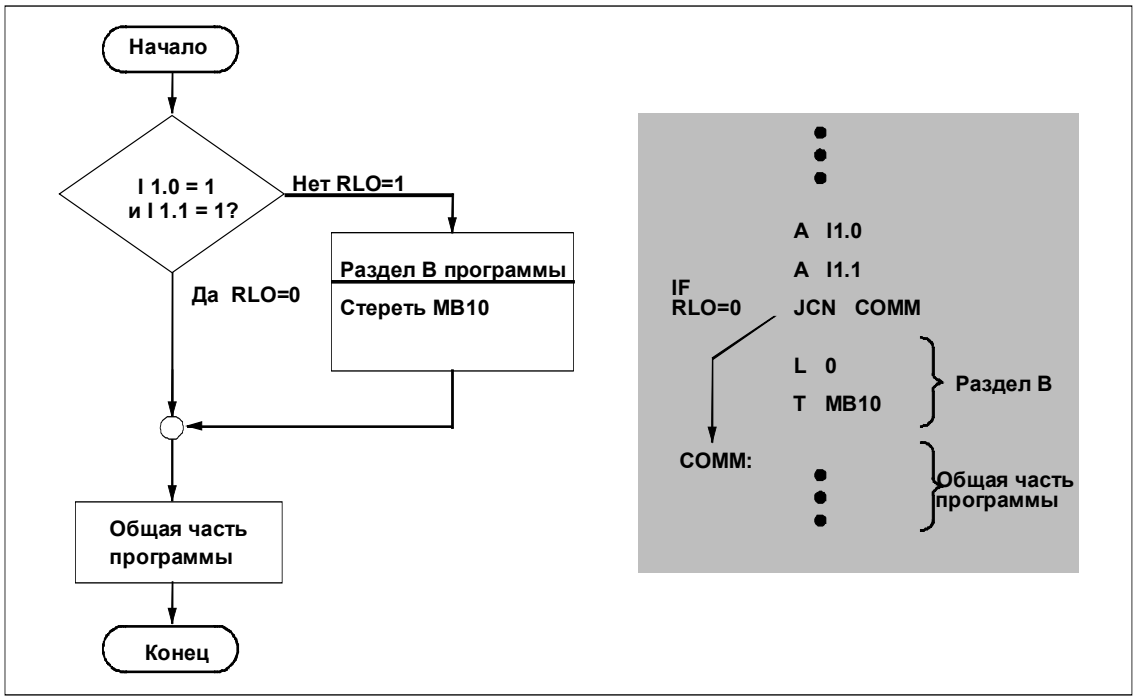


Рис. 16–2. Управление последовательностью выполнения программы с помощью команды JCN *Перейти, если RLO = 0*

16.4 Команды условного перехода, зависящие от битов BR, OV или OS слова состояния

Описание

Следующие команды перехода прерывают последовательность выполнения вашей программы на основе сигнального состояния бита в слове состояния (см. раздел 2.2).

- Перейти, если BR = 1 (JBI) или Перейти, если BR = 0 (JNBI)
- Перейти, если OV = 1 (JO) или Перейти, если OS = 1 (JOS)

Команды JBI и JNBI сбрасывают биты OR и $\overline{FC}$ слова состояния в 0 и устанавливают бит STA в 1 . Команда JOS сбрасывает бит OS в 0.

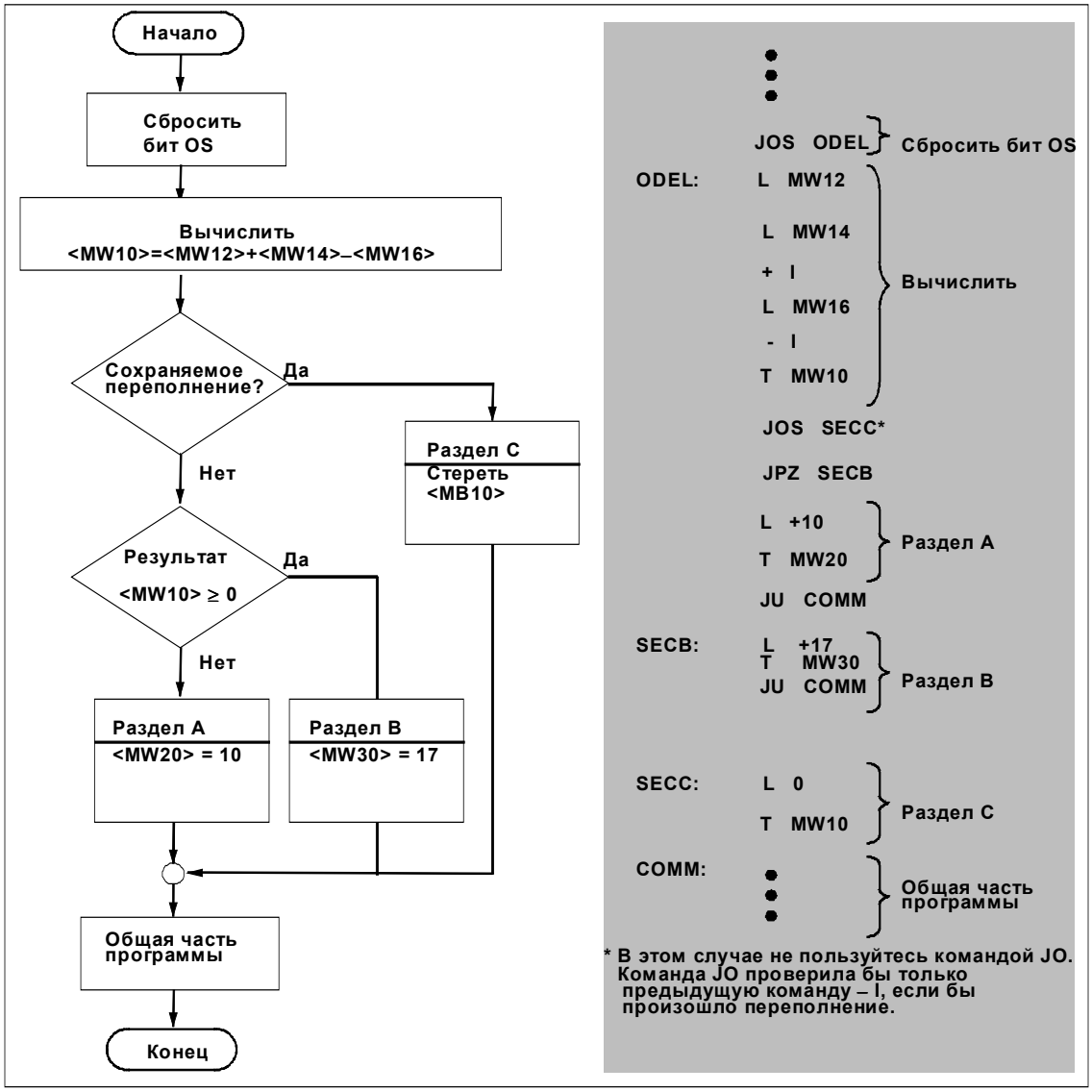


Рис. 16–3. Управление выполнением программы с помощью команды JOS (Перейти, если OS = 1)

16.5 Команды условного перехода, зависящие от значения битов CC 1 и CC 0 слова состояния

Описание

Следующие команды перехода прерывают последовательность выполнения вашей программы на основе результата вычислений:

- Перейти, если результат = 0 (JZ)
- Перейти, если результат < > 0 (JN)
- Перейти, если результат > 0 (JP)
- Перейти, если результат < 0 (JM)
- Перейти, если результат <=0 (JMZ, т.е. меньше или равен нулю)
- Перейти, если результат >= 0 (JPZ, т.е. больше или равен нулю, см. рис. 16–4)
- Перейти, если результат недействителен (JUO, т.е. если одно из чисел в арифметической операции с плавающей точкой не является допустимым числом с плавающей точкой).

CC 1 и CC 0 в слове состояния

Биты слова состояния CC 1 и CC 0 записываются в зависимости от результата предыдущей операции. Сигнальные состояния битов CC 1 и CC 0 слова состояния указывают на условия, представленные в табл. 16–1.

Таблица 16–1. Связь CC 1 и CC 0 с командами условного перехода

Сигнальное состояние		**Результат вычисления**	**Запускаемая команда перехода**
CC 1	**CC 0**		
0	0	=0	JZ
1 или 0	0 или 1	<>0	JN
1	0	>0	JP
0	1	<0	JM
0 или 1	0 или 0	>=0	JPZ
0 или 0	0 или 1	<=0	JMZ
1	1	UO (недействителен)	JUO

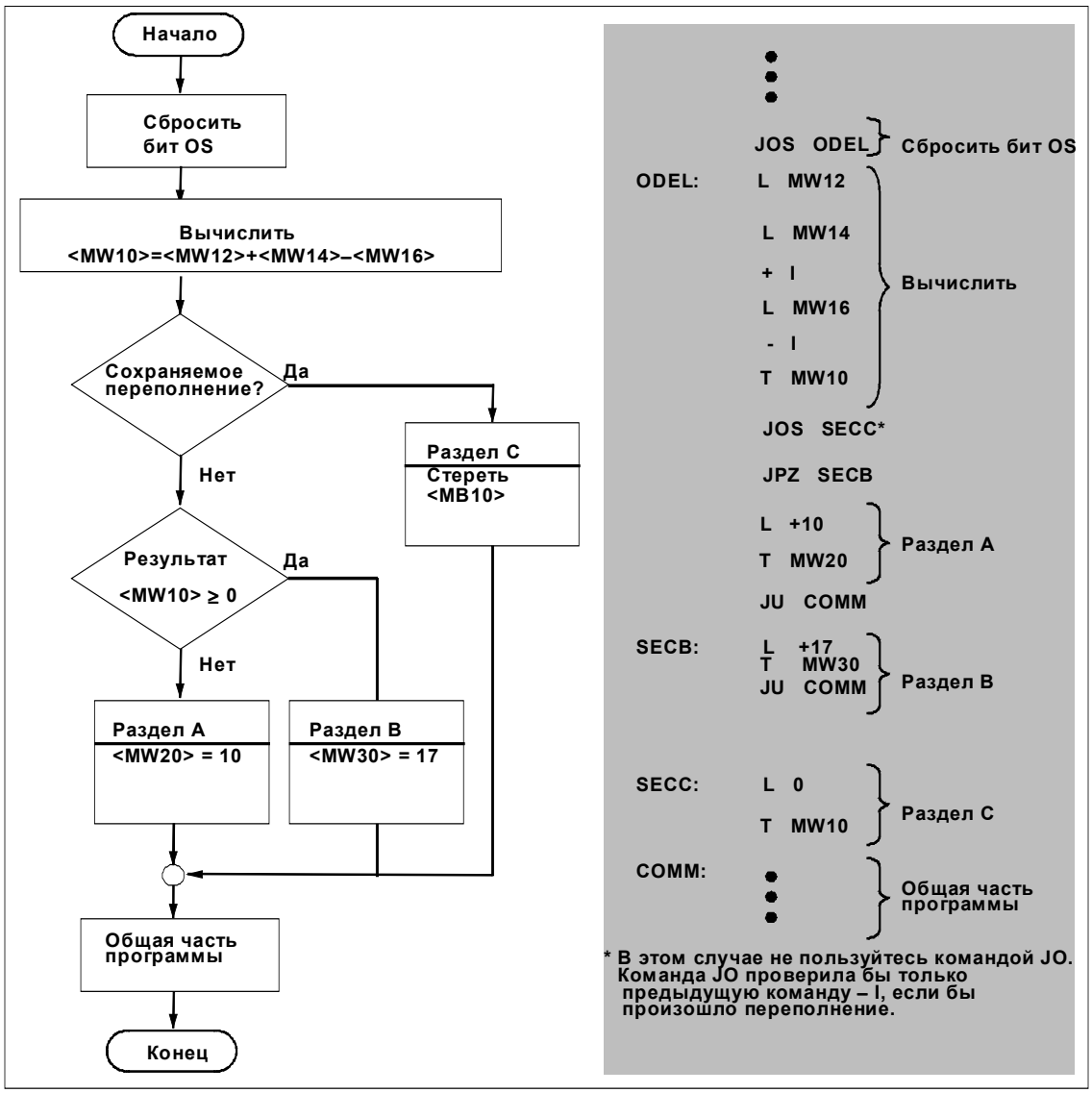


Рис. 16–4. Управление выполнением программы с помощью команды JPZ (Перейти, если результат ≥0)

16.6 Циклическое управление

Описание

Вы можете использовать команду *Цикл* (LOOP) для многократного вызова раздела программы (см. рис. 16–5). Команда *Цикл* уменьшает младшее слово аккумулятора 1 на 1. Затем значение в младшем слове аккумулятора 1 проверяется. Если оно не равно 0, то выполняется переход к метке, указанной в операнде команды LOOP; в противном случае выполняется следующая команда.

Метка перехода в качестве операнда

Вы снабжаете команду LOOP меткой, так что она знает точку, к которой она должна вернуться в программе. Например, команда LOOP в программе, показанной на рис. 16–5, имеет в качестве операнда метку перехода NEXT. Эта метка сообщает команде о необходимости возврата в программе к оператору T MB10. В этой точке программа обрабатывает раздел A. Команда LOOP возвращается к этой метке столько раз, сколько вы ей укажете. Эту информацию вы предоставляете в младшем слове аккумулятора 1. Одним из способов сделать это является установка счетчика цикла и загрузка его в аккумулятор.

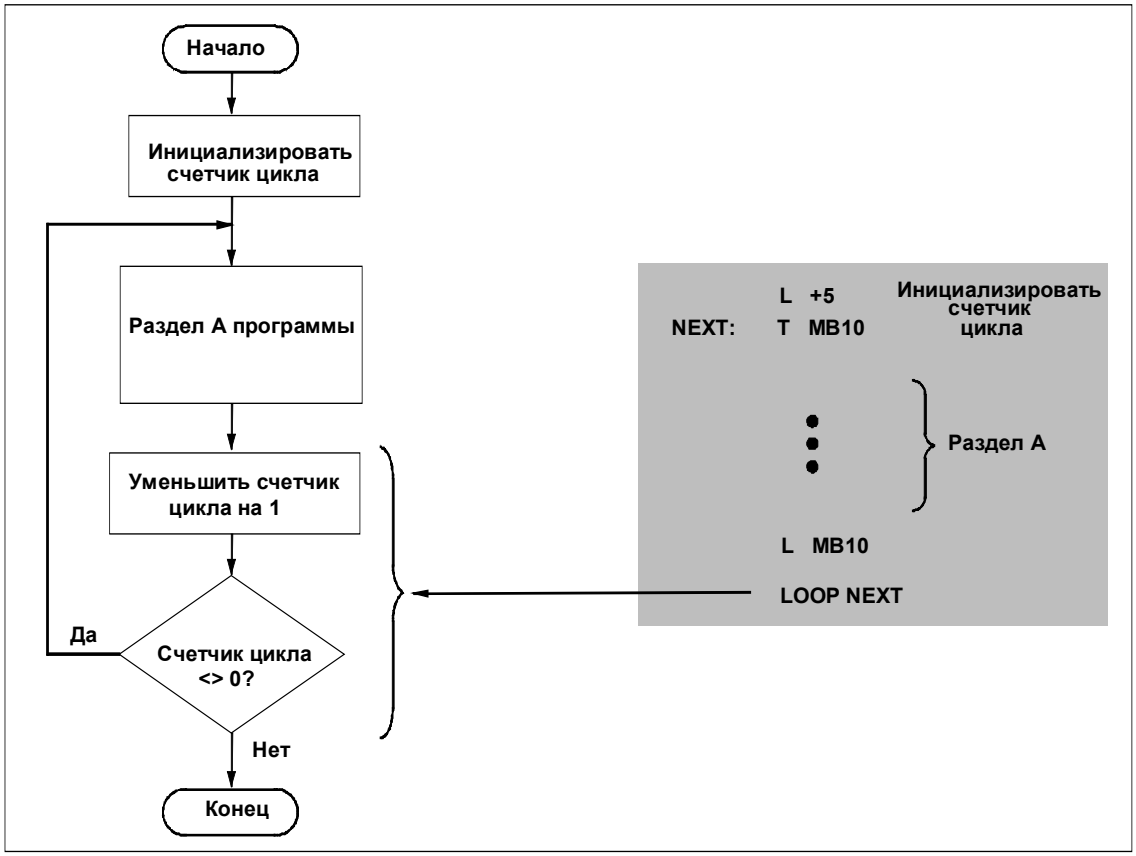


Рис. 16–5. Использование команды LOOP для многократного вызова раздела программы

Установка счетчика цикла

Вы снабжаете команду LOOP значением, указывающим, сколько раз эта команда должна вызвать определенный раздел программы.

Команда LOOP интерпретирует счетчик цикла как тип данных WORD.

В таблице 16–2 вы найдете два возможных формата для счетчика цикла.

Таблица 16–2. Возможный формат для счетчика цикла

Тип величины	Диапазон значений	Тип данных	Область памяти
Целое число	от 1 до 65 535 (только положительное значение)	WORD	I, Q, M, D, L
Слово	от W#16#0001 до W#16#FFFF	WORD	I, Q, M, D, L

Как эффективно использовать команду LOOP

Во избежание исполнения программного цикла большее число раз, чем это необходимо, вам нужно принять во внимание следующие свойства команды LOOP:

- Если вы инициализируете счетчик цикла нулем, то цикл исполняется 65 535 раз.
- Следует избегать инициализации цикла отрицательным числом.

17 Команды управления программой

Обзор главы

17.1 Назначение параметров при вызове FC и FB

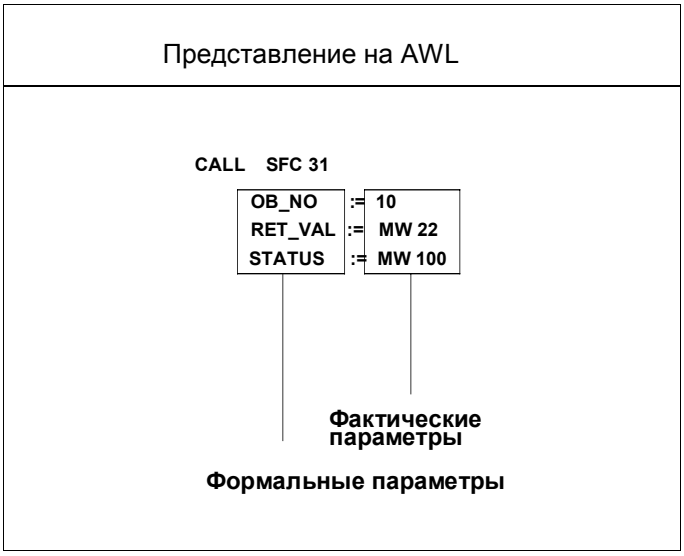
Терминология

При вызове блоков, требующих параметров, важную роль играют термины: **формальные параметры** и **фактические параметры**.

Формальный параметр – это параметр, имя и тип данных которого были определены и описаны (например, как INPUT, OUTPUT) при создании блока. При вызове блока в редакторе пошагового ввода (например, CALL SFC31) STEP 7 automatically displays a list of all the formal par автоматически отображает список формальных параметров.

Следующий шаг состоит в сопоставлении **фактических параметров** формальным параметрам. Фактический параметр – это параметр, который функция или функциональный блок использует во время фактического выполнения программы пользователя.

Следующая диаграмма иллюстрирует вызов SFC31 «QRY_TINT» (опрос прерывания по времени) на AWL.



17.2 Вызов функций и функциональных блоков с помощью CALL

Описание

С помощью команды вызова (CALL) вы можете вызывать функции (FC) и функциональные блоки (FB), которые вы сами создали для своей программы или получили от фирмы Siemens в качестве стандартных функций или функциональных блоков. Команда CALL вызывает FC или FB, которую или который вы указали в качестве операнда, независимо от результата логической операции или любого другого условия.

Когда вы вызываете с помощью команды CALL функциональный блок, вы должны снабдить его экземплярным блоком данных (экземплярным DB) или описать его как локальный экземпляр. В этом экземплярном блоке данных сохраняются все статические переменные и фактические параметры функционального блока.

Информацию о том, как можно запрограммировать функцию или функциональный блок или как работать с их параметрами, вы найдете в *Оперативной помощи STEP 7* в режиме *Online*.

Формальные и фактические параметры

При вызове функции (FC) или функционального блока (FB) вы должны присвоить формальным параметрам, определенным при описании блока, соответствующие фактические параметры.

Фактические параметры, указываемые при вызове функционального блока, должны иметь тот же тип данных, что и соответствующие формальные параметры.

Задание фактических параметров

Фактические параметры, используемые при вызове функции (FC) или функционального блока (FB), задаются, как правило, в виде символических имен. Абсолютная адресация фактических параметров возможна только для операндов, максимальная величина которых составляет двойное слово (например, I 1.0, MB2, QW4, ID0).

При вызове функции все формальные параметры должны быть снабжены фактическими параметрами. При вызове функционального блока необходимо описывать фактические параметры только в том случае, когда они отличаются о параметров предыдущего вызова (после завершения обработки функционального блока его фактические параметры сохраняются в экземплярном блоке данных).

При вызове функционального блока команда CALL копирует в экземплярный блок данных функционального блока один из следующих элементов в зависимости от типа данных фактического параметра и описания формального параметра (IN, OUT, IN_OUT):

- Значение фактического параметра
- Указатель на операнд фактического параметра
- Указатель на L–стек вызывающего блока, в котором было буферизовано значение фактического параметра.

Вызов FB с экземплярным DB и параметрами блока

Вызов может иметь место при вводе следующих элементов:

- имени функционального блока
- имени экземплярного блока данных и
- параметров (если фактическим параметром является блок данных, то всегда должен быть указан его полный абсолютный адрес, например, DB1.DBW2).

Вызов может использовать абсолютный или символический адрес.

Абсолютный вызов:

CALL FBx, DBy (*передача параметров*);
x = номер блока

y = номер блока данных

Символический вызов:

CALL *имя_fb, имя_блока_данных (передача параметров);*
имя_fb = символическое имя блока

имя_блока_данных = символическое имя блока данных

Примеры

В следующем примере программы вызывается функциональный блок FB40 с экземплярным блоком данных DB41. В этом примере формальные параметры имеют следующие типы данных:

IN1: BOOL
IN2: WORD
OUT1: DWORD

AWL		Объяснение
CALL	**FB40,DB41**	**Вызов функционального блока FB40 с экземплярным блоком данных DB41.**
IN1:=	**I1.0**	**Формальный параметр IN1 снабжается фактическим параметром I 1.0.**
IN2:=	**MW2**	**Формальный параметр IN2 снабжается фактическим параметром MW2.**
OUT1:=	**MD20**	**Формальный параметр OUT1 снабжается фактическим параметром MD20.**
L	**MD20**	**С помощью этой команды программа обращается к формальному параметру OUT1.**

Следующий пример показывает вызов функционального блока FB50 с экземплярным данных DB51. В этом примере формальные параметры имеют следующие типы данных:

IN10: BOOL
OUT11: STRUCT
V1: BOOL
V2: INT
END_STRUCT

AWL		Объяснение
CALL	**FB50,DB51**	**Вызов функционального блока FB50 с экземплярным блоком данных DB51.**
IN10:=	**I1.0**	**Формальный параметр IN10 снабжается фактическим параметром I 1.0.**
OUT11:=	**ACTPA11**	**В этом случае невозможно задать абсолютный фактический параметр (например, MW10), так как формальный параметр OUT11 был определен как структура. Вместо этого задан символический фактический параметр ACTPA11. Обратите, пожалуйста, внимание, что ACTPA11 должен иметь ту же структуру, что и формальный параметр OUT11.**

Доступ значениям структуры OUT11 в FB50 мог бы быть организован следующим образом:

AWL		Объяснение
A	**OUT11.V1**	**Выполнить логическую операцию И с битом OUT11.V1.**
L	**OUT11.V2**	**Загрузить слово OUT11.V2 в аккумулятор 1.**

Вызов мультиэкземпляров

Вызов может иметь место при вводе следующих элементов:

- имени экземпляра (= имени статической переменной типа FB z) и
- параметров

Вызов всегда имеет символическое обозначение.

CALL *имя_экземпляра (передача параметров);*

AWL	Объяснение
FUNCTION_BLOCK FB 11	**Исходный файл**
VAR	
loc_inst : FB 10;	**Описание мультиэкземпляра с типом данных FB10**
END_VAR	
BEGIN	
NETWORK	
CALL #loc_inst (	**Вызов мультиэкземпляра в соответствии с синтаксисом**
in_bool := M 0.0);	**Передача параметров (при этом in_bool является переменной, описанной в FB10)**

Вызов FC с параметрами блока

Вызов может иметь место при вводе следующих элементов:

- имени функции и
- параметров.

Вызов может производиться абсолютно или символически.

Абсолютный вызов:

CALL FCx (*передача параметров*);

x = номер блока

Символический вызов:

CALL *имя_fc (передача параметров);*

имя_fc = символическое имя блока

Пример

Следующий пример показывает вызов функции FC80 с параметрами блока. В этом примере формальные параметры имеют следующие типы данных:

INC1: BOOL
INC2: INT
OUT: WORD

AWL		Объяснение
CALL	**FC80**	**Вызов FC80.**
INC1:=	**M 1.0**	**Формальный параметр INC1 снабжается фактическим параметром M 1.0.**
INC2:=	**IW2**	**Формальный параметр INC2 снабжается фактическим параметром IW2.**
OUT:=	**QW4**	**Формальный параметр OUT снабжается фактическим параметром QW4.**

Вызов FC, поставляющей возвращаемое значение

Вы можете создать функцию (FC), которая поставляет возвращаемое значение (RET_VAL). Например, если вы хотите создать арифметическую функцию, работающую с числами с плавающей точкой, то вы можете использовать это возвращаемое значение для вывода результата этой функции. Когда вы вызываете функцию в своей программе, то вы предоставляете в распоряжение выход RET_VAL с адресом двойного слова, так что он может воспринять 32-битовый результат арифметической операции с плавающей точкой.

17.3 Вызов функций и функциональных блоков с помощью CC и UC

Описание

Вы можете использовать следующие команды для вызова функций (FC) и функциональных блоков (FB), которые вы создали для своей программы, таким же образом, как и команду CALL. Однако, используя эти команды, вы не можете передавать параметры.

- Условный вызов (CC): вызывает функцию или функциональный блок, указанную(ый) в качестве операнда, если результат логической операции равен 1.
- Безусловный вызов (UC): вызывает функцию или функциональный блок, указанную(ый) в качестве операнда, независимо от результата логической операции или какого-либо иного условия.

Функциональные блоки, вызываемые с помощью команды CC или UC, не могут иметь соответствующих блоков данных.

Формат адресации

Команда CC или UC может вызывать функцию (FC) или функциональный блок (FB) с помощью прямой или косвенной (через память) адресации или через FC или FB, передаваемый в качестве параметра (см. таблицы 17–1 и 17–2). Операндом является FC или FB плюс номер FC или FB.

Таблица 17–1. Операнды команд CC и UC: прямая и косвенная адресация

FC– или FB–часть операнда		**Максимальный диапазон адресов в соответствии с видом адресации**		
		прямая	**косвенная через память**	
FC	FB	от 0 до 65 535	[DBW] [DIW] [LW] [MW]	от 0 до 65 535

Таблица 17–2. Операнды команд CC и UC: FC передается как параметр

Операнд	**Виды адресных параметров**
Имя формального параметра или символическое имя	BLOCK_FC [1] BLOCK_FB [1]

[1] Параметры типа BLOCK_FC или BLOCK_FB не могут применяться в FC и FB с командой CC

Пример

Для вызова созданной вами FC с присвоенным ей номером 12 вам следует одну из следующих команд в зависимости от того, хотите ли вы, чтобы вызов был условным или нет:

```
CC  FC12 (Вызвать FC12, если RLO равен 1)
UC  FC12 (Вызвать FC12 независимо от значения RLO)
```

Назначение фактических параметров

В зависимости от типа данных у вас есть различные возможности назначения формальным параметрам фактических параметров при вызове функции или функционального блока. Следующая таблица упорядочена по длине типа данных.

Таблица 17–3. Назначение фактических параметров

Тип данных формального параметра	**Пример назначения фактического параметра**		
	Прямой ввод (значение)	**Ввод элемента глобальных данных**	**Символический ввод [1]**
BOOL (бит)	TRUE	M 100.0 I 0.0 Q 0.0 DBX 3.0	#OK_BIT
BYTE (байт)	B#16#1F	MB 100 IB 0 QB 0	#TYP_BYTE
CHAR	’K’	DBB 1	#TYP_CHAR
WORD (слово)	W#16#1F12 2#0001_1111_0001_0010 C#32 B#(5,25)	MW 100 IW 0 QW 0 DBW 2	#TYP_WORD
INT (целое число)	27 -25		#TYP_INT
S5TIME (время в формате S5)	S5T#10MS		#TYP_S5_TIME
DATE (дата в формате IEC)	D#1995-12-24		#TYP_DATE
DWORD (двойное слово)	DW#16#FFFF_0F02 2#0001_1111_0001_0010_0001_1111_0001_0010 B#(5,4,59,8)	MD 100 ID 0 QD 0 DBD 4	#TYP_DWORD
DINT (двойное целое число)	L#170 L#–350		#TYP_DINT
REAL (число с плавающей точкой)	1.23		#TYP_REAL
TIME (время в формате IEC)	T#20MS		#TYP_TIME

Таблица 17–3. Назначение фактических параметров

Тип данных формального параметра	**Пример назначения фактического параметра**		
	Прямой ввод (значение)	**Ввод элемента глобальных данных**	**Символический ввод [1]**
TIME_OF_DAY (время суток в формате IEC)	TOD#23:59:12.3		#TYP_TOD
DATE_AND_TIME (дата и время в формате IEC)	Невозможен (переменная должна быть описана, например, как временная переменная)		#TYP_8_BYTE
ANY (данные любого типа и размера)	**P# M0.0 BYTE20** **20 байтов[2] …[3]** **... от бита памяти 0.0** **Префикс для любого указателя ANY** **P#DB58.DBX16.0 BYTE14** **14 байтов[2] ..** **... в DB58 от бита данных 16.** **Префикс для любого указателя ANY**	I 0.0 MB 5 AW 2 (возможно использование любых глобальных операндов STEP 7)	#TYP_ANYTYP (Описание массивов и структур)

[1] Предпосылка: В случае глобальных данных имя (= символ) должно быть описано в таблице символов до того, как оно сможет быть использовано в качестве фактического параметра. В случае локальных данных имя (= символ) должно быть описано в таблице описаний блока до того, как оно сможет быть использовано в качестве фактического параметра. Символам локальных данных должен предшествовать знак #.

[2] Данные о длине могут включать элементарные типы данных, например, BOOL, BYTE, WORD или DWORD, или составные типы данных, например, DATE_AND_TIME.

[3] Всегда вводите битовый адрес; при задании длины вводите в качестве битового адреса 0 (исключение: BOOL).

Условный вызов в AWL

Чтобы произвести условный вызов SFC, вы можете использовать последовательность команд, аналогичную следующей:

AWL			**Объяснение**
	A	**#OK_BIT_MEMORY**	**Условие для вызова.**
	JCNB	**m001**	**Если условие не выполнено (RLO=0), вызов SFC пропускается, а RLO сохраняется в бите состояния BR.**
	CALL SFC 28		
	OB_NO	**:= 10**	
	SDT	**:= #OUT_TIME_DATE**	
	PERIOD	**:= W#16#1201**	
	RET_VAL	**:= MW 200**	
m001:	**A**	**BR**	**Опрос бита состояния BR**
	=	**M 202.3**	

17.4 Работа с функциями Master Control Relay

Описание

Master Control Relay (главное управляющее реле, MCR) используется в релейно-контактных схемах для активизации и деактивизации потока сигнала (пути тока). Деактивированный путь тока соответствует последовательности команд, которая записывает нулевое значение вместо рассчитанного, или последовательности команд, которая оставляет неизменным существующее значение памяти. От MCR зависят следующие битовые логические операции и команды передачи:

- =
- S
- R
- T (применяется с байтом, словом или двойным словом)

Команда T, используемая с байтом, словом или двойным словом, и команда = записывают в память 0, если MCR равно 0. Команды S и R не изменяют существующее значение.

Таблица 17–4. Операции, зависящие от MCR, и их реакция на сигнальное состояние MCR

Сигнальное состояние MCR	**=**	**S или R**	**T**
0	Записывает 0 (имитирует реле, которое при исчезновении напряжения переходит в состояние покоя)	Не записывает (имитирует реле с защелкой, которое при исчезновении напряжения остается в текущем состоянии)	Записывает 0 (имитирует компонент, который при исчезновении напряжения выдает значение 0)
1	Нормальное исполнение	Нормальное исполнение	Нормальное исполнение

17.5 Команды Master Control Relay

Обзор

Вы можете использовать следующие операторы для реализации главного управляющего реле:

- MCRA Активизировать область MCR
- MCRD Деактивизировать область MCR
- MCR(Сохранить RLO в стеке MCR, начало области MCR
-)MCR Завершить область MCR

Описание: MCRA, MCRD

Следующие операции активизируют и деактивируют область MCR, т.е. они указывают, какие команды в вашей программе зависят от MCR (см. также рис. 17–1):

- Активизировать область MCR: MCRA
- Деактивизировать область MCR: MCRD

Команды, запрограммированные между MCRA и MCRD, зависят от сигнального состояния бита MCR. Команды, запрограммированные вне последовательности MCRA–MCRD, не зависят от сигнального состояния бита MCR. Если команда MCRD отсутствует, то от бита MCR зависят команды, запрограммированные между MCRA и BEU (см. рис. 17–1).

Зависимость функций (FC) и функциональных блоков (FB) от MCR в блоках вы должны программировать сами. Если эта функция или этот функциональный блок вызывается из последовательности MCRA–MCRD, то не все команды внутри этой последовательности автоматически зависят от бита MCR. Чтобы сделать команды в вызванном блоке зависящими от бита MCR, вы должны использовать команду MCRA в вызванном блоке.

Опасность

Никогда не используйте команду MCR в качестве устройства для аварийного отключения или защиты персонала.

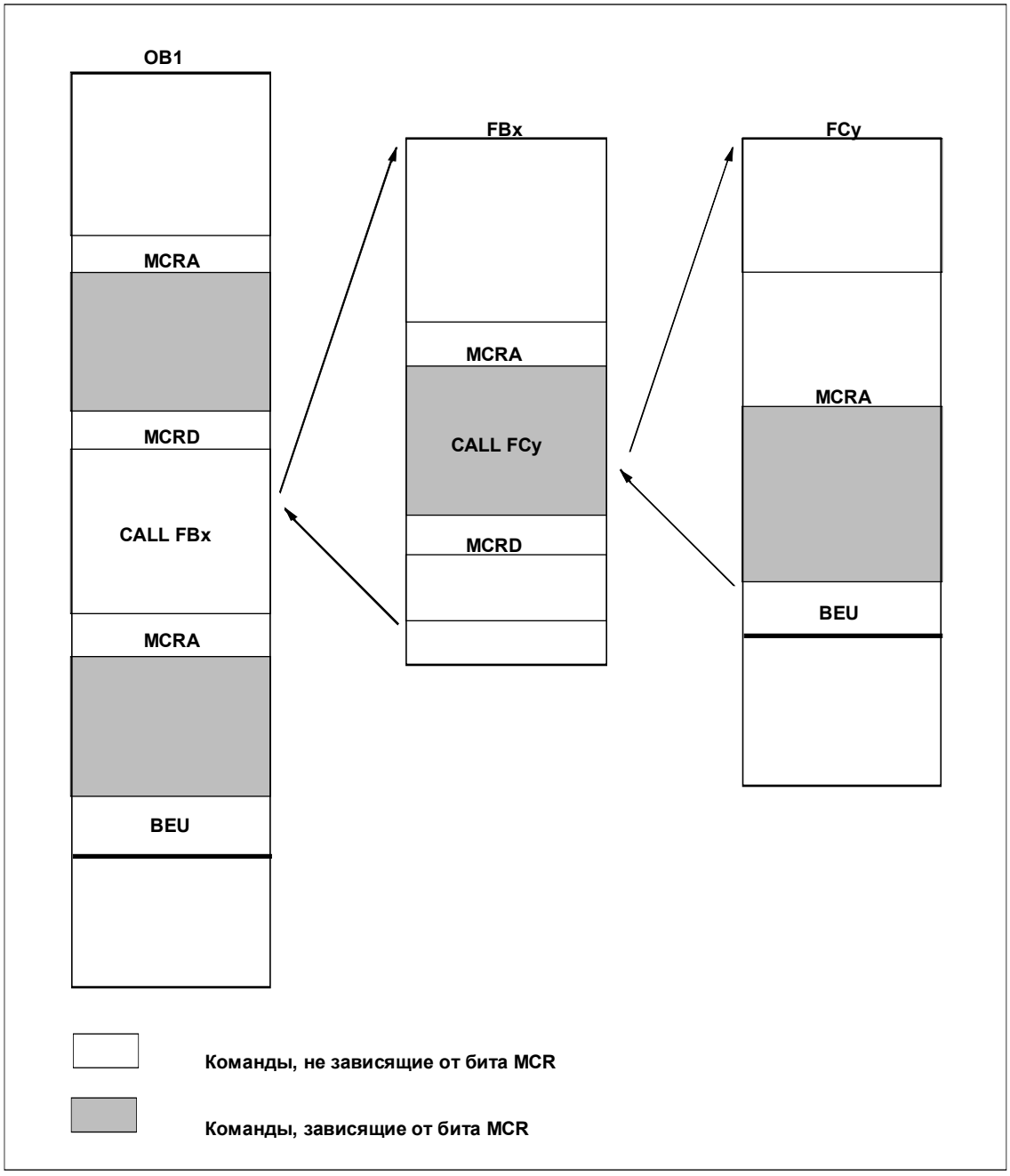


Рис. 17–1. Активизация и деактивизация области Master Control Relay

Описание: MCR(,)MCR

Следующие команды включают и выключают функцию Master Control Relay:

- Сохранить RLO в стеке MCR, начать MCR: MCR(
- Завершить MCR:)MCR

Команды MCR(и)MCR можно вкладывать друг в друга. Максимальная глубина вложения равна восьми, т.е. вы можете записать друг за другом не более восьми команд MCR(, прежде чем вставите команду)MCR. Вы должны запрограммировать одинаковое количество команд MCR(и)MCR (см. рис. 17–2).

Когда команды MCR(вкладываются друг в друга, то формируется бит MCR более глубокого уровня вложенности. Для образования этого бита MCR команда MCR(комбинирует текущее значение RLO с текущим битом MCR в соответствии с таблицей истинности для И.

Команда)MCR завершает уровень вложения, восстанавливая бит MCR из более высокого уровня. Команда)MCR самого высокого уровня устанавливает бит MCR в 1.

Команды MCR(и)MCR в своей программе вы всегда должны применять парами.

Пример

На рис. 17–2 показано, как реализуется Master Control Relay.

Если бит MCR равен 1, то контакт MCR замкнут. Сигнальные состояния выходов Q 4.0 и Q 4.1 рассчитываются в соответствии с сигнальными состояниями входов от I 1.0 до I 1.3 и их логическими комбинациями.

Если бит MCR равен 0, то контакт MCR разомкнут. Выходы Q 4.0 и Q 4.1 сбрасываются в 0 независимо от сигнальных состояний входов от I 1.0 до I 1.3.

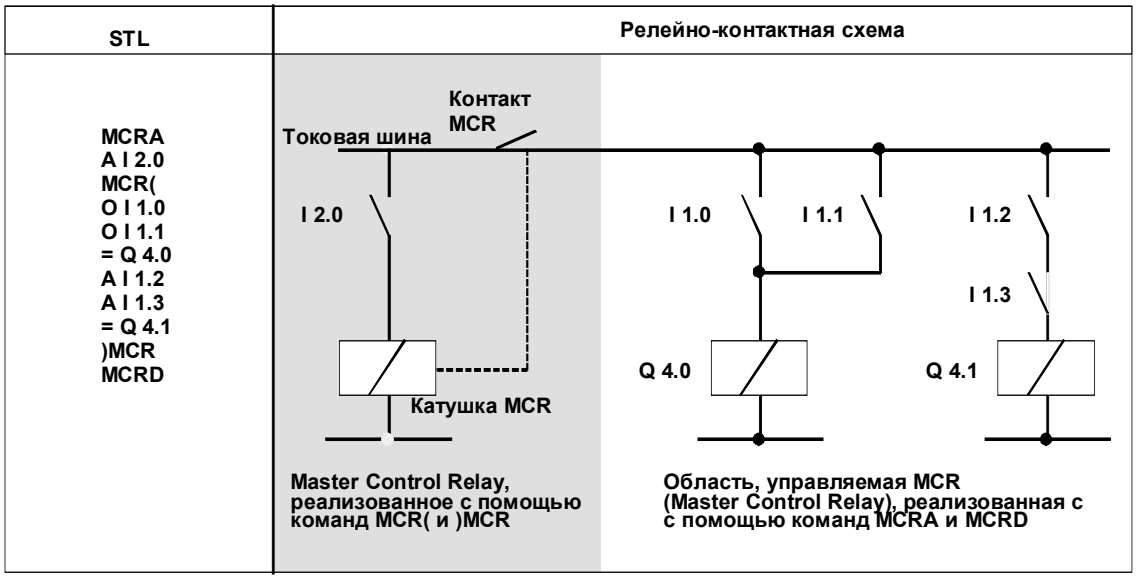


Рис. 17–2. Реализация Master Control Relay

Рис. 17–3 показывает применение вложения команд.

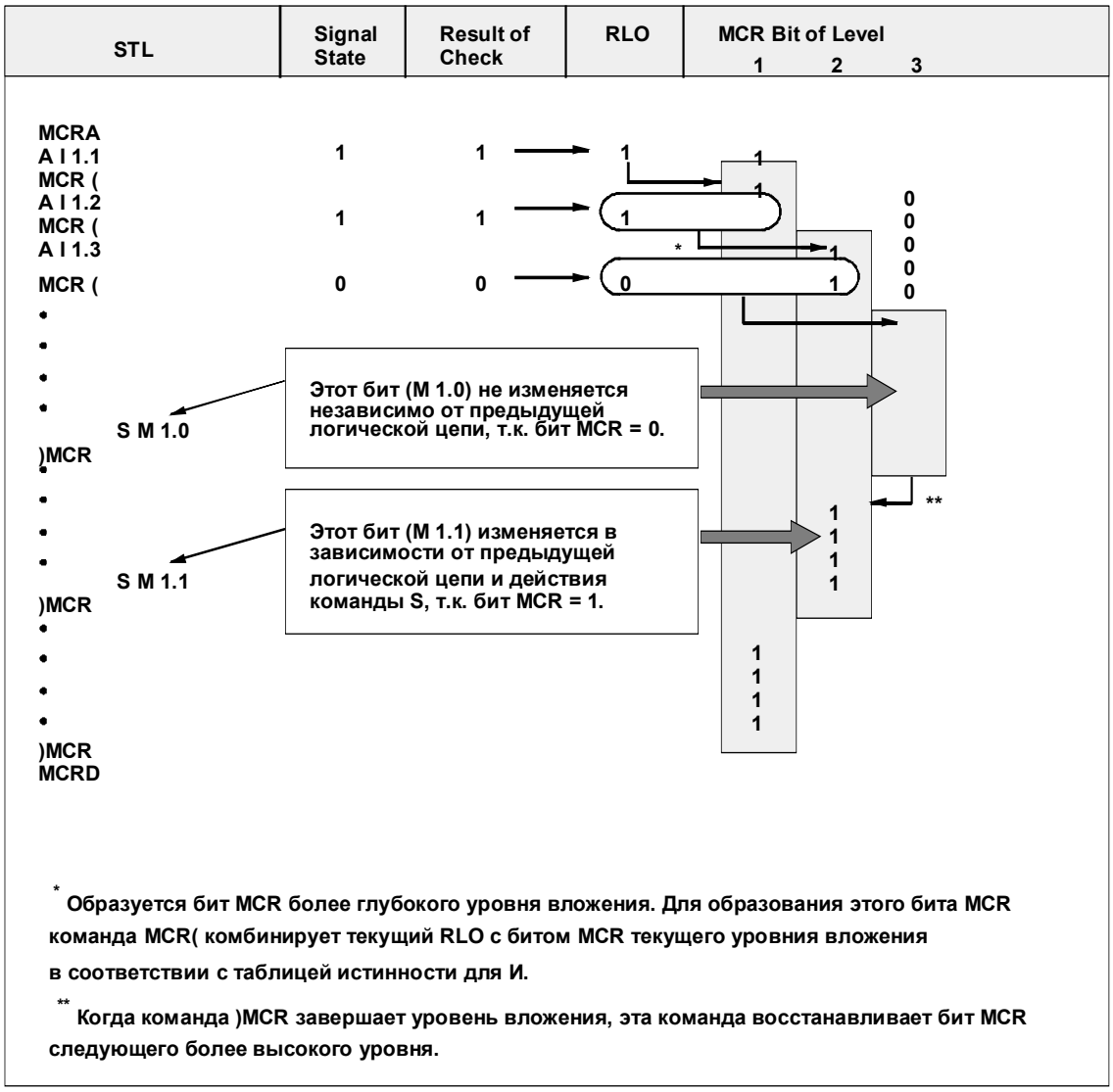


Рис. 17–3. Применение вложения команд MCR

Важные замечания об использовании функций MCR

Будьте осторожны с блоками, в которых с помощью MCRA было активизировано Master Control Relay:

- Если MCR деактивизируется, то всеми командами присваивания в сегментах программы между MCR< и MCR> записывается 0.
- MCR деактивизируется, если RLO был =0 перед командой MCR<.

Опасность

ПЛК переходит в STOP или имеет неопределенные характеристики этапа выполнения программы!

Компилятор также использует доступ на запись к локальным данным после временных переменных, определенных в VAR_TEMP, для вычисления адресов.

Обращение к формальным параметрам

- Обращение к компонентам составных параметров FC типа STRUCT, UDT, ARRAY, STRING
- Обращение к компонентам составных параметров FB типа STRUCT, UDT, ARRAY, STRING из области IN_OUT в блоке версии 2.
- Обращение к параметрам функционального блока версии 2, если его адрес больше 8180.0.
- Обращение в функциональном блоке версии 2 к параметру типа BLOCK_DB открывает DB0. Любое последующее обращение к данным переводит CPU в STOP. T 0, C 0, FC0 или FB0 всегда используются для TIMER, COUNTER, BLOCK_FC и BLOCK_FB.

Передача параметров

- Вызовы, в которых параметры передаются.

KOP/FUP

- T-образные ветви и промежуточные выходы (коннекторы) в KOP или FUP, начинающиеся с RLO=0.

Устранение:

Освободите вышеуказанные команды от их зависимости от MCR:

1. Деактивизируйте Master Control Relay с помощью команды *Деактивизировать Master Control Relay (MCRD)* перед соответствующим оператором или сегментом.
2. Снова активизируйте Master Control Relay с помощью команды *Активизировать Master Control Relay Activate (MCRA)* после соответствующего оператора или сегмента.

17.6 Завершение блоков

Описание

Команда *Конец блока* сама по себе является оператором, завершающим обработку блока. Вы можете использовать для завершения блока любой из следующих типов команд *Конец блока*:

- Безусловный конец блока (BEU): Эта команда завершает обработку текущего блока и возвращает управление блоку, вызвавшему завершенный блок. Когда программа встречает команду BEU, она завершает текущий блок независимо от результата логической операции.
- Условный конец блока (BEC): Эта команда завершает обработку текущего блока и возвращает управление блоку, вызвавшему завершенный блок. Когда программа встречает команду BEC, она завершает текущий блок только в том случае, если результат логической операции равен 1 (RLO = 1). Если RLO равен 0, то программа не выполняет оператор *Условный конец блока* (BEC). RLO устанавливается в 1, и обработка программы продолжается внутри текущего блока.

A Алфавитный список команд

Обзор главы

A.1 Список немецкой (SIMATIC) и международной мнемоники

В таблице A–1 приведен алфавитный список мнемонических сокращений команд языка *Список операторов*. Вслед за каждым немецким сокращением идет эквивалентное международное сокращение, описание команды на русском языке и страница, на которой эта команда объясняется.

Таблица A–1. Алфавитный список немецких (SIMATIC) и международных мнемонических сокращений

Мнемоника SIMATIC	Междуна-родная мнемоника	Описание	Стр.
+	+	Прибавить целую константу (8, 16, 32 бита)	9–6
=	=	Присвоить	5–24
)	)	Вложение закрыто	5–14
+AR1	+AR1	Прибавить аккумулятор 1 к адресному регистру 1	4–7
+AR2	+AR2	Прибавить аккумулятор 1 к адресному регистру 2	4–7
+D	+D	Сложить аккумулятор 1 и аккумулятор 2 как двойные целые числа (32 бита)	9–2
–D	–D	Вычесть аккумулятор 1 из аккумулятора 2 как двойные целые числа (32 бита)	9–2
*D	*D	Умножить аккумулятор 1 на аккумулятор 2 как двойные целые числа (32 бита)	9–2
/D	/D	Разделить аккумулятор 2 на аккумулятор 1 как двойные целые числа (32 бита)	9–2
==D	==D	Сравнить двойные целые числа (32 бита) >, <, >=, <=, ==, <>	11–3
+I	+I	Сложить аккумулятор 1 и аккумулятор 2 как целые числа (16 битов)	9–2
–I	–I	Вычесть аккумулятор 1 из аккумулятора 2 как целые числа (16 битов)	9–2
*I	*I	Умножить аккумулятор 1 на аккумулятор 2 как целые числа (16 битов)	9–2
/I	/I	Разделить аккумулятор 2 на аккумулятор 1 как целые числа (16 битов)	9–2
==I	==I	Сравнить целые числа (16 битов) >, <, >=, <=, ==, <>	11–3
+R	+R	Сложить аккумулятор 1 и аккумулятор 2 как вещественные числа (32 бита, IEEE FP)	10–2
–R	–R	Вычесть аккумулятор 1 из аккумулятора 2 как вещественные числа (32 бита, IEEE FP)	10–2
*R	*R	Умножить аккумулятор 1 на аккумулятор 2 как вещественные числа (32 бита, IEEE FP)	10–2
/R	/R	Разделить аккумулятор 2 на аккумулятор 1 как вещественные числа (32 бита, IEEE FP)	10–2
==R	==R	Сравнить вещественные числа >, <, >=, <=, ==, <>	11–5
ABS	ABS	Абсолютное значение вещественного числа (32 бита, IEEE FP)	10–6
ACOS	ACOS	Арккосинус числа с плавающей точкой (32 бита, IEEE FP)	10–7
ASIN	ASIN	Арксинус числа с плавающей точкой (32 бита, IEEE FP)	10–7
ATAN	ATAN	Арктангенс числа с плавающей точкой (32 бита, IEEE FP)	10–7
AUF	OPN	Открыть блок данных	15–2
BEA	BEU	Конец блока безусловный	17–16
BEB	BEC	Конец блока условный	17–16
BLD	BLD	Команда программирования изображения	4–2
BTD	BTD	Преобразовать BCD в двойное целое число (32 бита)	12–4
BTI	BTI	Преобразовать BCD в целое число (16 битов)	12–2
CALL	CALL	Вызов блока	17–3
CC	CC	Условный вызов блока	17–7
CLR	CLR	Очистить RLO (= 0)	5–26

Таблица A–1. Алфавитный список немецких (SIMATIC) и международных мнемонических сокращений

Мнемоника SIMATIC	**Междуна-родная мнемоника**	**Описание**	**Стр.**
COS	COS	Косинус числа с плавающей точкой (32 бита, IEEE FP)	10–7
DEC	DEC	Уменьшить аккумулятор 1	4–6
DTB	DTB	Преобразовать двойное целое число (32 бита) в BCD	12–6
DTR	DTR	Преобразовать двойное целое число (32 бита) в вещественное число (32 бита, IEEE FP)	12–7
ENT	ENT	Аккумулятор 3 ——> Аккумулятор 4, Аккумулятор 2 ——> Аккумулятор 3	4–3
EXP	EXP	Экспоненциальное значение числа с плавающей точкой (32 бита, IEEE FP) по основанию E	10–12
FN	FN	Отрицательный фронт	5–17
FP	FP	Положительный фронт	5–16
FR	FR	Разблокировать счетчик (Free, FR C 0 до C 255)	6–5
FR	FR	Разблокировать таймер (Free, FR T 0 до T 255)	7–3
INC	INC	Увеличить аккумулятор 1	4–6
INVD	INVD	Дополнение до 1 двойного целого числа (32 бита)	12–14
INVI	INVI	Дополнение до 1 целого числа (16 битов)	12–14
ITB	ITB	Преобразовать целое число (16 битов) в BCD	12–5
ITD	ITD	Преобразовать целое число (16 битов) в двойное целое число (32 бита)	12–6
L	L	Загрузить	8–3
L	L	Загрузить длину глобального блока данных в аккумулятор 1 (L DBLG)	8–12 15–2
L	L	Загрузить номер глобального блока данных в аккумулятор 1 (L DBNO)	8–12
L	L	Загрузить длину экземплярного блока данных в аккумулятор 1 (L DILG)	8–12 15–2
L	L	Загрузить номер экземплярного блока данных в аккумулятор 1 (L DINO)	8–12 15–2
L	L	Загрузить слово состояния в аккумулятор 1 (L STW)	8–6
L	L	Загрузить текущее значение таймера как целое число в аккумулятор 1 (где номер текущего таймера может находиться в диапазоне от 0 до 255, например: L T 32)	8–7
L	L	Загрузить текущее значение счетчика как целое число в аккумулятор 1 (где номер текущего счетчика может находиться в диапазоне от 0 до 255, например: L C 15)	7–6 8–8
LAR1	LAR1	Загрузить адресный регистр 1 из аккумулятора 1 (если не указан адрес)	8–11
LAR1	LAR1	Загрузить адресный регистр 1 из... (указанного адреса)	8–11
LAR1	LAR1	Загрузить адресный регистр 1 из адресного регистра 2 (LAR1 AR2)	8–11
LAR1	LAR1	Загрузить адресный регистр 1 двойным целым числом (32 бита, LAR1 P#область байт.бит)	8–11
LAR2	LAR2	Загрузить адресный регистр 2 из аккумулятора 1 (если не указан адрес)	8–11
LAR2	LAR2	Загрузить адресный регистр 2 из... (указанного адреса)	8–11
LAR2	LAR2	Загрузить адресный регистр 2 двойным целым числом (32 бита, LAR2 P#область байт.бит)	8–11
LC	LC	Загрузить текущее значение счетчика как BCD в аккумулятор 1 (где номер текущего счетчика может быть в диапазоне от 0 до 255, например: LC C 15)	8–9
LC	LC	Загрузить текущее значение таймера как BCD в аккумулятор 1 (где номер текущего таймера может быть в диапазоне от 0 до 255, например: LC T 32)	7–7 8–9
LEAVE	LEAVE	Аккумулятор 3 ——> Аккумулятор 2, Аккумулятор 4 ——> Аккумулятор 3	4–3
LN	LN	Натуральный логарифм числа с плавающей точкой (32 бита, IEEE FP)	10–11
LOOP	LOOP	Программный цикл	16–8

Таблица A–1. Алфавитный список немецких (SIMATIC) и международных мнемонических сокращений

Мнемоника SIMATIC	**Междуна-родная мнемоника**	**Описание**	**Стр.**
MCR(	MCR(	Сохранить RLO в стеке MCR, начать MCR	17–11
)MCR	MCR)	Восстановить RLO, завершить MCR	17–11
MCRA	MCRA	Активизировать область MCR	17–11
MCRD	MCRD	Деактивизировать область MCR	17–11
MOD	MOD	Остаток от деления двойного целого числа (32 бита)	9–5
NEGD	NEGD	Дополнение до 2 двойного целого числа (32 бита)	12–14
NEGI	NEGI	Дополнение до 2 целого числа (16 битов)	12–14
NEGR	NEGR	Изменить знак вещественного числа (32 бит, IEEE FP)	12–14
NOP 0	NOP 0	Пустая операция 0	4–2
NOP 1	NOP 1	Пустая операция 1	4–2
NOT	NOT	Инвертировать RLO	5–26
O	O	ИЛИ	5–10
O(	O(	ИЛИ с открытием вложения	5–14
OD	OD	Поразрядное ИЛИ с двойными словами (32 бита)	13–6
ON	ON	ИЛИ-НЕ	5–8
ON(	ON(	ИЛИ-НЕ с открытием вложения	5–14
OW	OW	Поразрядное ИЛИ со словами (16 битов)	13–3
POP	POP	Аккумулятор 1 <–– Аккумулятор 2, Аккумулятор 2 <–– Аккумулятор 3, Аккумулятор 3 <–– Аккумулятор 4	4–2
PUSH	PUSH	Аккумулятор 3 ––> Аккумулятор 4, Аккумулятор 2 ––> Аккумулятор 3, Аккумулятор 1 ––> Аккумулятор 2	4–2
R	R	Сбросить	5–22
R	R	Сбросить счетчик (где текущий счетчик может иметь номер в диапазоне от 0 до 255, например: R C 15)	6–5
R	R	Сбросить таймер (где текущий таймер может иметь номер в диапазоне от 0 до 255, например: R T 32)	7–4
RLD	RLD	Выполнить циклический сдвиг двойного слова влево (32 бита)	14–8
RLDA	RLDA	Выполнить циклический сдвиг аккумулятора 1 влево через CC 1 (32 бита)	14–6
RND	RND	Округлить	12–9
RND+	RND+	Округлить до ближайшего большего двойного целого числа	12–10
RND–	RND–	Округлить до ближайшего меньшего двойного целого числа	12–11
RRD	RRD	Выполнить циклический сдвиг двойного слова вправо (32 бита)	14–8
RRDA	RRDA	Выполнить циклический сдвиг аккумулятора 1 вправо через CC 1 (32 бита)	14–6
S	S	Установить	5–21
S	S	Установить начальное значение счетчика (где текущий счетчик может иметь номер в диапазоне от 0 до 255, например: S C 15)	7–3
SA	SF	Таймер – формирователь задержки выключения	6–15
SAVE	SAVE	Сохранить RLO в регистре BR	5–26
SE	SD	Таймер – формирователь задержки включения	6–11
SET	SET	Установить RLO (= 1)	5–26
SI	SP	Таймер – формирователь импульса	6–7
SIN	SIN	Синус числа с плавающей точкой (32 бита, IEEE FP)	10–7
SLD	SLD	Сдвинуть влево двойное слово (32 бита)	14–2
SLW	SLW	Сдвинуть влево слово (16 битов)	14–2
SPA	JU	Перейти безусловно	16–3
SPB	JC	Перейти, если RLO = 1	16–4
SPBB	JCB	Перейти, если RLO = 1 с сохранением RLO в BR	16–4
SPBI	JBI	Перейти, если BR = 1	16–4
SPBIN	JNBI	Перейти, если BR = 0	16–4
SPBN	JCN	Перейти, если RLO = 0	16–4
SPBNB	JNB	Перейти, если RLO = 0 с сохранением RLO в BR	16–4
SPL	JL	Перейти по списку	16–3
SPM	JM	Перейти, если результат < 0	16–6

Таблица A–1. Алфавитный список немецких (SIMATIC) и международных мнемонических сокращений

Мнемоника SIMATIC	Междуна-родная мнемоника	Описание	Стр.
SPMZ	JMZ	Перейти, если результат < = 0	16–6
SPN	JN	Перейти, если результат < > 0	16–6
SPO	JO	Перейти, если OV = 1	16–5
SPP	JP	Перейти, если результат > 0	16–6
SPPZ	JPZ	Перейти, если результат > = 0	16–6
SPS	JOS	Перейти, если OS = 1	16–5
SPU	JUO	Перейти, если результат недействителен	16–6
SPZ	JZ	Перейти, если результат = 0	16–6
SQR	SQR	Квадрат числа с плавающей точкой (32 бита, IEEE PF)	10–9
SQRT	SQRT	Квадратный корень числа с плавающей точкой (32 бита IEEE PF)	10–9
SRD	SRD	Сдвинуть вправо двойное слово (32 бита)	14–3
SRW	SRW	Сдвинуть вправо слово (16 битов)	14–2
SS	SS	Таймер – формирователь задержки включения с запоминанием	6–13
SSD	SSD	Сдвинуть двойное целое число со знаком (32 бита)	14–4
SSI	SSI	Сдвинуть целое число со знаком (16 битов)	14–4
SV	SE	Таймер – формирователь удлиненного импульса	6–9
T	T	Передать	8–3
T	T	Передать аккумулятор 1 в слово состояния (T STW)	8–6
TAD	CAD	Изменить последовательность байтов в аккумуляторе 1 (32 бита)	12–13
TAK	TAK	Обменять аккумулятор 1 с аккумулятором 2	4–2
TAN	TAN	Тангенс числа с плавающей точкой (32 бита, IEEE FP)	10–7
TAR	CAR	Обменять адресный регистр 1 с адресным регистром 2	8–11
TAR1	TAR1	Передать адресный регистр 1 в аккумулятор 1 (если не указан адрес)	8–11
TAR1	TAR1	Передать адресный регистр 1 в... (указанный адрес)	8–11
TAR1	TAR1	Передать адресный регистр 1 в адресный регистр 2 (T AR1 AR2)	8–11
TAR2	TAR2	Передать адресный регистр 2 в аккумулятор 1 (если не указан адрес)	8–11
TAR2	TAR2	Передать адресный регистр 2 в... (указанный адрес)	8–11
TAW	CAW	Изменить последовательность байтов в аккумуляторе 1 (16 битов)	12–13
TDB	CDB	Обменять глобальный блок данных и экземплярный блок данных	15–2
TRUNC	TRUNC	Округлить до целого отбрасыванием младших разрядов	12–12
U	A	И	5–10
U(	A(	И с открытием вложения	5–14
UC	UC	Безусловный вызов блока	17–7
UD	AD	Поразрядное И с двойными словами (32 бита)	13–6
UN	AN	И-НЕ	5–8
UN(	AN(	И-НЕ с открытием вложения	5–14
UW	AW	Поразрядное И со словами (16 битов)	13–3
X	X	Исключающее ИЛИ	5–10
X(	X(	Исключающее ИЛИ с открытием вложения	5–14
XN	XN	Исключающее ИЛИ-НЕ	5–8
XN(	XN(	Исключающее ИЛИ-НЕ с открытием вложения	5–14
XOD	XOD	Поразрядное исключающее ИЛИ с двойными словами (32 бита)	13–6
XOW	XOW	Поразрядное исключающее ИЛИ со словами (16 битов)	13–3
ZR	CD	Счетчик обратного счета	7–5
ZV	CU	Счетчик прямого счета	7–5

В таблице A–2 приведен алфавитный список мнемонических сокращений команд языка *Список операторов*. Вслед за каждым международным сокращением идет эквивалентное сокращение SIMATIC (немецкое), описание команды на русском языке и страница, на которой эта команда объясняется.

Таблица A–2. Алфавитный список международных и немецких (SIMATIC) мнемонических сокращений

Междуна-родная мнемоника	Мнемоника SIMATIC	Описание	Стр.
+	+	Прибавить целую константу (8, 16, 32 бита)	9–6
=	=	Присвоить	5–24
)	)	Вложение закрыто	5–14
+AR1	+AR1	Прибавить аккумулятор 1 к адресному регистру 1	4–7
+AR2	+AR2	Прибавить аккумулятор 1 к адресному регистру 2	4–7
+D	+D	Сложить аккумулятор 1 и аккумулятор 2 как двойные целые числа (32 бита)	9–2
–D	–D	Вычесть аккумулятор 1 из аккумулятора 2 как двойные целые числа (32 бита)	9–2
*D	*D	Умножить аккумулятор 1 на аккумулятор 2 как двойные целые числа (32 бита)	9–2
/D	/D	Разделить аккумулятор 2 на аккумулятор 1 как двойные целые числа (32 бита)	9–2
==D	==D	Сравнить двойные целые числа (32 бита) >, <, >=, <=, ==, <>	11–3
+I	+I	Сложить аккумулятор 1 и аккумулятор 2 как целые числа (16 битов)	9–2
–I	–I	Вычесть аккумулятор 1 из аккумулятора 2 как целые числа (16 битов)	9–2
*I	*I	Умножить аккумулятор 1 на аккумулятор 2 как целые числа (16 битов)	9–2
/I	/I	Разделить аккумулятор 2 на аккумулятор 1 как целые числа (16 битов)	9–2
==I	==I	Сравнить целые числа (16 битов) >, <, >=, <=, ==, <>	11–3
+R	+R	Сложить аккумулятор 1 и аккумулятор 2 как вещественные числа (32 бита, IEEE FP)	10–2
–R	–R	Вычесть аккумулятор 1 из аккумулятора 2 как вещественные числа (32 бита, IEEE FP)	10–2
*R	*R	Умножить аккумулятор 1 на аккумулятор 2 как вещественные числа (32 бита, IEEE FP)	10–2
/R	/R	Разделить аккумулятор 2 на аккумулятор 1 как вещественные числа (32 бита, IEEE FP)	10–2
==R	==R	Сравнить вещественные числа >, <, >=, <=, ==, <>	11–5
A	U	И	5–10
A(	U(	И с открытием вложения	5–14
ABS	ABS	Абсолютное значение вещественного числа (32 бита, IEEE FP)	10–6
ACOS	ACOS	Арккосинус числа с плавающей точкой (32 бита, IEEE FP)	10–7
AD	UD	Поразрядное И с двойными словами (32 бита)	13–6
AN	UN	И-НЕ	5–9
AN(	UN(	И-НЕ с открытием вложения	5–14
ASIN	ASIN	Арксинус числа с плавающей точкой (32 бита, IEEE FP)	10–7
ATAN	ATAN	Арктангенс числа с плавающей точкой (32 бита, IEEE FP)	10–7
AW	UW	Поразрядное И со словами (16 битов)	13–3
BEC	BEB	Конец блока условный	17–16
BEU	BEA	Конец блока безусловный	17–16
BLD	BLD	Команда программирования изображения	4–2
BTD	BTD	Преобразовать BCD в двойное целое число (32 бита)	12–4
BTI	BTI	Преобразовать BCD в целое число (16 битов)	12–2

Таблица A–2. Алфавитный список международных и немецких (SIMATIC) мнемонических сокращений

Междуна-родная мнемоника	**Мнемоника SIMATIC**	**Описание**	**Стр.**
CAD	TAD	Изменить последовательность байтов в аккумуляторе 1 (32 бита)	12–13
CALL	CALL	Вызов блока	17–3
CAR	TAR	Обменять адресный регистр 1 с адресным регистром 2	8–11
CAW	TAW	Изменить последовательность байтов в аккумуляторе 1 (16 битов)	12–13
CC	CC	Условный вызов блока	17–7
CD	ZR	Счетчик обратного счета	7–5
CDB	TDB	Обменять глобальный блок данных и экземплярный блок данных	15–2
CLR	CLR	Очистить RLO (= 0)	5–26
COS	COS	Косинус числа с плавающей точкой (32 бита, IEEE FP)	10–7
CU	ZV	Счетчик прямого счета	7–5
DEC	DEC	Уменьшить аккумулятор 1	4–6
DTB	DTB	Преобразовать двойное целое число (32 бита) в BCD	12–6
DTR	DTR	Преобразовать двойное целое число (32 бита) в вещественное число (32 бита, IEEE FP)	12–7
ENT	ENT	Аккумулятор 3 ——> Аккумулятор 4, Аккумулятор 2 ——> Аккумулятор 3	4–3
EXP	EXP	Экспоненциальное значение числа с плавающей точкой (32 бита, IEEE FP) по основанию E	10–12
FN	FN	Отрицательный фронт	5–17
FP	FP	Положительный фронт	5–16
FR	FR	Разблокировать счетчик (Free, FR C 0 до C 255)	6–5
FR	FR	Разблокировать таймер (Free, FR T 0 до T 255)	7–3
INC	INC	Увеличить аккумулятор 1	4–6
INVD	INVD	Дополнение до 1 двойного целого числа (32 бита)	12–14
INVI	INVI	Дополнение до 1 целого числа (16 битов)	12–14
ITB	ITB	Преобразовать целое число (16 битов) в BCD	12–5
ITD	ITD	Преобразовать целое число (16 битов) в двойное целое число (32 бита)	12–6
JBI	SPBI	Перейти, если BR = 1	16–4
JC	SPB	Перейти, если RLO = 1	16–4
JCB	SPBB	Перейти, если RLO = 1 с сохранением RLO в BR	16–4
JCN	SPBN	Перейти, если RLO = 0	16–4
JL	SPL	Перейти по списку	16–3
JM	SPM	Перейти, если результат < 0	16–6
JMZ	SPMZ	Перейти, если результат < = 0	16–6
JN	SPN	Перейти, если результат < > 0	16–6
JNB	SPBNB	Перейти, если RLO = 0 с сохранением RLO в BR	16–4
JNBI	SPBIN	Перейти, если BR = 0	16–4
JO	SPO	Перейти, если OV = 1	16–5
JOS	SPS	Перейти, если OS = 1	16–5
JP	SPP	Перейти, если результат > 0	16–6
JPZ	SPPZ	Перейти, если результат > = 0	16–6
JU	SPA	Перейти безусловно	16–3
JUO	SPU	Перейти, если результат недействителен	16–6
JZ	SPZ	Перейти, если результат = 0	16–6
L	L	Загрузить	8–3
L	L	Загрузить длину глобального блока данных в аккумулятор 1 (L DBLG)	8–12 15–2
L	L	Загрузить номер глобального блока данных в аккумулятор 1 (L DBNO)	8–12
L	L	Загрузить длину экземплярного блока данных в аккумулятор 1 (L DILG)	8–12 15–2
L	L	Загрузить номер экземплярного блока данных в аккумулятор 1 (L DINO)	8–12 15–2
L	L	Загрузить слово состояния в аккумулятор 1 (L STW)	8–6

Таблица A–2. Алфавитный список международных и немецких (SIMATIC) мнемонических сокращений

Междуна-родная мнемоника	Мнемоника SIMATIC	Описание	Стр.
L	L	Загрузить текущее значение таймера как целое число в аккумулятор 1 (где номер текущего таймера может находиться в диапазоне от 0 до 255, например: L T 32)	8–7
L	L	Загрузить текущее значение счетчика как целое число в аккумулятор 1 (где номер текущего счетчика может находиться в диапазоне от 0 до 255, например: L C 15)	7–6 8–8
LAR1	LAR1	Загрузить адресный регистр 1 из аккумулятора 1 (если не указан адрес)	8–11
LAR1	LAR1	Загрузить адресный регистр 1 из... (указанного адреса)	8–11
LAR1	LAR1	Загрузить адресный регистр 1 из адресного регистра 2 (LAR1 AR2)	8–11
LAR1	LAR1	Загрузить адресный регистр 1 двойным целым числом (32 бита, LAR1 P#область байт.бит)	8–11
LAR2	LAR2	Загрузить адресный регистр 2 из аккумулятора 1 (если не указан адрес)	8–11
LAR2	LAR2	Загрузить адресный регистр 2 из... (указанного адреса)	8–11
LAR2	LAR2	Загрузить адресный регистр 2 двойным целым числом (32 бита, LAR2 P#область байт.бит)	8–11
LC	LC	Загрузить текущее значение счетчика как BCD в аккумулятор 1 (где номер текущего счетчика может быть в диапазоне от 0 до 255, например: LC C 15)	8–9
LC	LC	Загрузить текущее значение таймера как BCD в аккумулятор 1 (где номер текущего таймера может быть в диапазоне от 0 до 255, например: LC T 32)	7–7 8–9
LEAVE	LEAVE	Аккумулятор 3 ——> Аккумулятор 2, Аккумулятор 4 ——> Аккумулятор 3	4–3
LN	LN	Натуральный логарифм числа с плавающей точкой (32 бита, IEEE FP)	10–11
LOOP	LOOP	Программный цикл	16–8
MCR(	MCR(	Сохранить RLO в стеке MCR, начать MCR	17–11
MCR)	)MCR	Восстановить RLO, завершить MCR	17–11
MCRA	MCRA	Активизировать область MCR	17–11
MCRD	MCRD	Деактивизировать область MCR	17–11
MOD	MOD	Остаток от деления двойного целого числа (32 бита)	9–5
NEGD	NEGD	Дополнение до 2 двойного целого числа (32 бита)	12–14
NEGI	NEGI	Дополнение до 2 целого числа (16 битов)	12–14
NEGR	NEGR	Изменить знак вещественного числа (32 бит, IEEE FP)	12–14
NOP 0	NOP 0	Пустая операция 0	4–2
NOP 1	NOP 1	Пустая операция 1	4–2
NOT	NOT	Инвертировать RLO	5–26
O	O	ИЛИ	5–10
O(	O(	ИЛИ с открытием вложения	5–14
OD	OD	Поразрядное ИЛИ с двойными словами (32 бита)	13–6
ON	ON	ИЛИ-НЕ	5–8
ON(	ON(	ИЛИ-НЕ с открытием вложения	5–14
OPN	AUF	Открыть блок данных	15–2
OW	OW	Поразрядное ИЛИ со словами (16 битов)	13–3
POP	POP	Аккумулятор 1 <—— Аккумулятор 2, Аккумулятор 2 <—— Аккумулятор 3, Аккумулятор 3 <—— Аккумулятор 4	4–2
PUSH	PUSH	Аккумулятор 3 ——> Аккумулятор 4, Аккумулятор 2 ——> Аккумулятор 3, Аккумулятор 1 ——> Аккумулятор 2	4–2
R	R	Сбросить	5–22
R	R	Сбросить счетчик (где текущий счетчик может иметь номер в диапазоне от 0 до 255, например: R C 15)	6–5
R	R	Сбросить таймер (где текущий таймер может иметь номер в диапазоне от 0 до 255, например: R T 32)	7–4
RLD	RLD	Выполнить циклический сдвиг двойного слова влево (32 бита)	14–8

Таблица A–2. Алфавитный список международных и немецких (SIMATIC) мнемонических сокращений

Междуна-родная мнемоника	**Мнемоника SIMATIC**	**Описание**	**Стр.**
RLDA	RLDA	Выполнить циклический сдвиг аккумулятора 1 влево через CC 1 (32 бита)	14–6
RND	RND	Округлить	12–9
RND+	RND+	Округлить до ближайшего большего двойного целого числа	12–10
RND–	RND–	Округлить до ближайшего меньшего двойного целого числа	12–11
RRD	RRD	Выполнить циклический сдвиг двойного слова вправо (32 бита)	14–8
RRDA	RRDA	Выполнить циклический сдвиг аккумулятора 1 вправо через CC 1 (32 бита)	14–6
S	S	Установить	5–21
S	S	Установить начальное значение счетчика (где текущий счетчик может иметь номер в диапазоне от 0 до 255, например: S C 15)	7–3
SAVE	SAVE	Сохранить RLO в регистре BR	5–26
SD	SE	Таймер – формирователь задержки включения	6–11
SE	SV	Таймер – формирователь удлиненного импульса	6–9
SET	SET	Установить RLO (= 1)	5–26
SF	SA	Таймер – формирователь задержки выключения	6–15
SIN	SIN	Синус числа с плавающей точкой (32 бита, IEEE FP)	10–7
SLD	SLD	Сдвинуть влево двойное слово (32 бита)	14–2
SLW	SLW	Сдвинуть влево слово (16 битов)	14–2
SP	SI	Таймер – формирователь импульса	6–7
SQR	SQR	Квадрат числа с плавающей точкой (32 бита, IEEE PF)	10–9
SQRT	SQRT	Квадратный корень числа с плавающей точкой (32 бита IEEE PF)	10–9
SRD	SRD	Сдвинуть вправо двойное слово (32 бита)	14–3
SRW	SRW	Сдвинуть вправо слово (16 битов)	14–2
SS	SS	Таймер – формирователь задержки включения с запоминанием	6–13
SSD	SSD	Сдвинуть двойное целое число со знаком (32 бита)	14–4
SSI	SSI	Сдвинуть целое число со знаком (16 битов)	14–4
T	T	Передать	8–3
T	T	Передать аккумулятор 1 в слово состояния (T STW)	8–6
TAK	TAK	Обменять аккумулятор 1 с аккумулятором 2	4–2
TAN	TAN	Тангенс числа с плавающей точкой (32 бита, IEEE FP)	10–7
TAR1	TAR1	Передать адресный регистр 1 в аккумулятор 1 (если не указан адрес)	8–11
TAR1	TAR1	Передать адресный регистр 1 в... (указанный адрес)	8–11
TAR1	TAR1	Передать адресный регистр 1 в адресный регистр 2 (T AR1 AR2)	8–11
TAR2	TAR2	Передать адресный регистр 2 в аккумулятор 1 (если не указан адрес)	8–11
TAR2	TAR2	Передать адресный регистр 2 в... (указанный адрес)	8–11
TRUNC	TRUNC	Округлить до целого отбрасыванием младших разрядов	12–12
UC	UC	Безусловный вызов блока	17–7
X	X	Исключающее ИЛИ	5–10
X(	X(	Исключающее ИЛИ с открытием вложения	5–14
XN	XN	Исключающее ИЛИ-НЕ	5–8
XN(	XN(	Исключающее ИЛИ-НЕ с открытием вложения	5–14
XOD	XOD	Поразрядное исключающее ИЛИ с двойными словами (32 бита)	13–6
XOW	XOW	Поразрядное исключающее ИЛИ со словами (16 битов)	13–3

A.2 Алфавитный список международных наименований

В таблице A–3 приведен алфавитный список полных международных наименований команд языка *Список операторов* с переводом (в квадратных скобках) на русский язык. Вслед за каждым наименованием идет его мнемоническое обозначение и страница, на которой эта команда объясняется.

Таблица A–3. Команды *Списка операторов*, упорядоченные по алфавиту в соответствии с их полными международными именами

Наименование команды	Мнемоническое обозначение	Стр.
Absolute Value of a Real (32–Bit IEEE FP) [Абсолютное значение вещественного числа (32 бита, IEEE FP)]	ABS	10–6
Accumulator 1 ——> Accumulator 2	PUSH	4–2
Accumulator 1 <—— Accumulator 2	POP	4–2
Accumulator 1 <—— Accumulator 2, Accumulator 2 <—— Accumulator 3, Accumulator 3 <—— Accumulator 4	POP	4–2
Accumulator 3 ——> Accumulator 2, Accumulator 4 ——> Accumulator 3	LEAVE	4–3
Accumulator 3 ——> Accumulator 4, Accumulator 2 ——> Accumulator 3	ENT	4–3
Accumulator 3 ——> Accumulator 4, Accumulator 2 ——> Accumulator 3, Accumulator 1 ——> Accumulator 2	PUSH	4–2
Activate MCR Area [Активизировать область MCR]	MCRA	17–11
Add Accumulator 1 and Accumulator 2 as Double Integer (32–Bit) [Сложить аккумулятор 1 и аккумулятор 2 как двойные целые числа (32 бита)]	+D	9–2
Add Accumulator 1 and Accumulator 2 as Integer (16–Bit) [Сложить аккумулятор 1 и аккумулятор 2 как целые числа (16 битов)]	+I	9–2
Add Accumulator 1 and Accumulator 2 as Real (32–Bit IEEE FP) [Сложить аккумулятор 1 и аккумулятор 2 как вещественные числа (32 бита, IEEE FP)]	+R	10–2
Add Accumulator 1 to Address Register 1 [Прибавить аккумулятор 1 к адресному регистру 1]	+AR1	4–7
Add Accumulator 1 to Address Register 2 [Прибавить аккумулятор 1 к адресному регистру 2]	+AR2	4–7
Add Integer Constant (8, 16, 32–Bit) [Прибавить целую константу (8, 16, 32 бита)]	+	9–6
And [И]	A	5–10
And Double Word (32–Bit) [Поразрядное И с двойными словами (32 бита)]	AD	13–6
And Not [И-НЕ]	AN	5–9
And Not with Nesting Open [И-НЕ с открытием вложения]	AN(	5–14
And with Nesting Open [И с открытием вложения]	A(	5–14
And Word (16–Bit) [Поразрядное И со словами (16 битов)]	AW	13–3
Arc Cosine of a Floating–Point Number (32–Bit IEEE FP) [Арккосинус числа с плавающей точкой (32 бита, IEEE FP)]	ACOS	10–7
Arc Sine of a Floating–Point Number (32–Bit IEEE FP) [Арксинус числа с плавающей точкой (32 бита, IEEE FP)]	ASIN	10–7
Arc Tangent of a Floating–Point Number (32–Bit IEEE FP) [Арктангенс числа с плавающей точкой (32 бита, IEEE FP)]	ATAN	10–7
Assign [Присвоить]	=	5–24
BCD to Double Integer (32–Bit) [Преобразовать BCD в двойное целое число (32 бита)]	BTD	12–4
BCD to Integer (16–Bit) [Преобразовать BCD в целое число (16 битов)]	BTI	12–2
Block End Conditional [Конец блока условный]	BEC	17–15

Таблица A–3. Команды *Списка операторов*, упорядоченные по алфавиту в соответствии с их полными международными именами

Наименование команды	Мнемони-ческое обозначение	Стр.
Block End Unconditional [Конец блока безусловный]	BEU	17–15
Call [Вызов блока]	CALL	17–3
Change Byte Sequence in Accumulator 1 (16–Bit) [Изменить последовательность байтов в аккумуляторе 1 (16 битов)]	CAW	12–13
Change Byte Sequence in Accumulator 1 (32–Bit) [Изменить последовательность байтов в аккумуляторе 1 (32 бита)]	CAD	12–13
Clear RLO (= 0) [Очистить RLO (= 0)]	CLR	5–26
Compare Double Integer (32–Bit) >, <, >=, <=, ==, <> [Сравнить двойные целые числа (32 бита)]	==D	11–3
Compare Integer (16–Bit) >, <, >=, <=, ==, <> [Сравнить целые числа (16 битов)]	==I	11–3
Compare Real >, <, >=, <=, ==, <> [Сравнить вещественные числа]	==R	11–5
Conditional Call [Условный вызов блока]	CC	17–7
Cosine of a Floating–Point Number (32–Bit IEEE FP) [Косинус числа с плавающей точкой (32 бита, IEEE FP)]	COS	10–7
Counter Down [Счетчик обратного счета]	CD	7–5
Counter Up [Счетчик прямого счета]	CU	7–5
Deactivate MCR Area [Деактивизировать область MCR]	MCRD	17–11
Decrement Accumulator 1 [Уменьшить аккумулятор 1]	DEC	4–6
Divide Accumulator 2 by Accumulator 1 as Double Integer (32–Bit) [Разделить аккумулятор 2 на аккумулятор 1 как двойные целые числа (32 бита)]	/D	9–2
Divide Accumulator 2 by Accumulator 1 as Integer (16–Bit) [Разделить аккумулятор 2 на аккумулятор 1 как целые числа (16 битов)]	/I	9–2
Divide Accumulator 2 by Accumulator 1 as Real (32–Bit IEEE FP) [Разделить аккумулятор 2 на аккумулятор 1 как вещественные числа (32 бита, IEEE FP)]	/R	10–2
Division Remainder Double Integer (32–Bit) [Остаток от деления двойного целого числа (32 бита)]	MOD	9–5
Double Integer (32–Bit) to BCD [Преобразовать двойное целое число (32 бита) в BCD]	DTB	12–6
Double Integer (32–Bit) to Real (32–Bit IEEE FP) [Преобразовать двойное целое число (32 бита) в вещественное число (32 бита, IEEE FP)]	DTR	12–7
Edge Negative [Отрицательный фронт]	FN	5–17
Edge Positive [Положительный фронт]	FP	5–16
Enable Counter (Free, FR C 0 to C 255) [Разблокировать счетчик (Free, FR C 0 до C 255)]	FR	6–5
Enable Timer (Free, FR T 0 to T 255) [Разблокировать таймер (Free, FR T 0 до T 255)]	FR	7–3
Exchange Address Register 1 with Address Register 2 [Обменять адресный регистр 1 с адресным регистром 2]	CAR	8–11
Exchange Shared Data Block and Instance Data Block [Обменять глобальный блок данных и экземплярный блок данных]	CDB	15–2
Exclusive Or [Исключающее ИЛИ]	X	5–10
Exclusive Or Double Word (32–Bit) [Поразрядное исключающее ИЛИ с двойными словами (32 бита)]	XOD	13–6
Exclusive Or Not [Исключающее ИЛИ-НЕ]	XN	5–8
Exclusive Or Not with Nesting Open [Исключающее ИЛИ-НЕ с открытием вложения]	XN(	5–14
Exclusive Or with Nesting Open [Исключающее ИЛИ с открытием вложения]	X(	5–14
Exclusive Or Word (16–Bit) [Поразрядное исключающее ИЛИ со словами (16 битов)]	XOW	13–3

Таблица A–3. Команды *Списка операторов*, упорядоченные по алфавиту в соответствии с их полными международными именами

Наименование команды	**Мнемони-ческое обозначение**	**Стр.**
Exponential Value of a Floating–Point Number (32–Bit IEEE FP) to base E [Экспоненциальное значение числа с плавающей точкой (32 бита, IEEE FP) по основанию E]	EXP	10–12
Extended Pulse Timer [Таймер – формирователь удлиненного импульса]	SE	6–9
Increment Accumulator 1 [Увеличить аккумулятор 1]	INC	4–6
Integer (16–Bit) to BCD [Преобразовать целое число (16 битов) в BCD]	ITB	12–5
Integer (16–Bit) to Double Integer [Преобразовать целое число (16 битов) в двойное целое число (32 бита)]	ITD	12–6
Jump if 0 [Перейти, если результат = 0]	JZ	16–6
Jump if BR = 0 [Перейти, если BR = 0]	JNBI	16–4
Jump if BR = 1 [Перейти, если BR = 1]	JBI	16–4
Jump if Minus [Перейти, если результат < 0]	JM	16–6
Jump if Minus or 0 [Перейти, если результат < = 0]	JMZ	16–6
Jump if Not 0 [Перейти, если результат < > 0]	JN	16–6
Jump if OS = 1 [Перейти, если OS = 1]	JOS	16–5
Jump if OV = 1 [Перейти, если OV = 1]	JO	16–5
Jump if Plus [Перейти, если результат > 0]	JP	16–6
Jump if Plus or 0 [Перейти, если результат > = 0]	JPZ	16–6
Jump if RLO = 0 [Перейти, если RLO = 0]	JCN	16–4
Jump if RLO = 0 with BR [Перейти, если RLO = 0 с сохранением RLO в BR]	JNB	16–4
Jump if RLO = 1 [Перейти, если RLO = 1]	JC	16–4
Jump if RLO = 1 with BR [Перейти, если RLO = 1 с сохранением RLO в BR]	JCB	16–4
Jump if Unordered [Перейти, если результат недействителен]	JUO	16–6
Jump to List [Перейти по списку]	JL	16–3
Jump Unconditional [Перейти безусловно]	JU	16–3
Load [Загрузить]	L	8–3
Load Address Register 1 from ... (from address indicated) [Загрузить адресный регистр 1 из... (указанного адреса)]	LAR1	8–11
Load Address Register 1 from Accumulator 1 (if no address is indicated) [Загрузить адресный регистр 1 из аккумулятора 1 (если не указан адрес)]	LAR1	8–11
Load Address Register 1 from Address Register 2 (LAR1 AR2) [Загрузить адресный регистр 1 из адресного регистра 2]	LAR1	8–11
Load Address Register 1 with Double Integer (32–Bit, LAR1 P#area byte.bit) [Загрузить адресный регистр 1 двойным целым числом (32 бита, LAR1 P#область байт.бит)]	LAR1	8–11
Load Address Register 2 from ... (from address indicated) [Загрузить адресный регистр 2 из... (указанного адреса)]	LAR2	8–11
Load Address Register 2 from Accumulator 1 (if no address is indicated) [Загрузить адресный регистр 2 из аккумулятора 1 (если не указан адрес)]	LAR2	8–11
Load Address Register 2 with Double Integer (32–Bit, LAR2 P#area byte.bit) [Загрузить адресный регистр 2 двойным целым числом (32 бита, LAR2 P#область байт.бит)]	LAR2	8–11
Load Current Counter Value into Accumulator 1 as Integer (where the number of the current counter can be in the range of 0 to 255, for example: L C 15) [Загрузить текущее значение счетчика как целое число в аккумулятор 1 (где номер текущего счетчика может находиться в диапазоне от 0 до 255, например: L C 15)]	L	7–6 8–8
Load Current Counter Value into Accumulator 1 as BCD (where the number	LC	8–9

Таблица A–3. Команды *Списка операторов*, упорядоченные по алфавиту в соответствии с их полными международными именами

Наименование команды	Мнемони-ческое обозначение	Стр.
of the current counter can be in the range of 0 to 255, for example: LC C 15) [Загрузить текущее значение счетчика как BCD в аккумулятор 1 (где номер текущего счетчика может быть в диапазоне от 0 до 255, например: LC C 15)]		
Load Current Timer Value into Accumulator 1 as BCD (where the number of the current timer can be in the range of 0 to 255, for example: LC T 32) [Загрузить текущее значение таймера как BCD в аккумулятор 1 (где номер текущего таймера может быть в диапазоне от 0 до 255, например: LC T 32)]	LC	7–7 8–9
Load Current Timer Value into Accumulator 1 as Integer (where the number of the current timer can be in the range of 0 to 255, for example: L T 32) [Загрузить текущее значение таймера как целое число в аккумулятор 1 (где номер текущего таймера может находиться в диапазоне от 0 до 255, например: L T 32)]	L	8–7
Load Length of Instance Data Block into Accumulator 1 (L DILG) [Загрузить длину экземплярного блока данных в аккумулятор 1]	L	8–12 15–2
Load Length of Shared Data Block into Accumulator 1 (L DBLG) [Загрузить длину глобального блока данных в аккумулятор 1]	L	8–12 15–2
Load Number of Instance Data Block into Accumulator 1 (L DINO) [Загрузить номер экземплярного блока данных в аккумулятор 1]	L	8–12 15–2
Load Number of Shared Data Block into Accumulator 1 (L DBNO) [Загрузить номер глобального блока данных в аккумулятор 1]	L	8–12 15–2
Load Status Word into Accumulator 1 (L STW) [Загрузить слово состояния в аккумулятор 1]	L	8–6
Loop [Программный цикл]	LOOP	16–8
Multiply Accumulator 1 by Accumulator 2 as Double Integer (32–Bit) [Умножить аккумулятор 1 на аккумулятор 2 как двойные целые числа (32 бита)]	*D	9–2
Multiply Accumulator 1 by Accumulator 2 as Integer (16–Bit) [Умножить аккумулятор 1 на аккумулятор 2 как целые числа (16 битов)]	*I	9–2
Multiply Accumulator 1 by Accumulator 2 as Real (32–Bit IEEE FP) [Умножить аккумулятор 1 на аккумулятор 2 как вещественные числа (32 бита, IEEE FP)]	*R	10–2
Natural Logarithm of a Floating–Point Number (32–Bit IEEE FP) [Натуральный логарифм числа с плавающей точкой (32 бита, IEEE FP)]	LN	10–11
Negate Real Number (32–Bit IEEE FP) [Изменить знак вещественного числа (32 бит, IEEE FP)]	NEGR	12–14
Negate RLO [Инвертировать RLO]	NOT	5–26
Nesting Closed [Вложение закрыто]	)	5–14
Null Operation 0 [Пустая операция 0]	NOP 0	4–2
Null Operation 1 [Пустая операция 1]	NOP 1	4–2
Off–Delay Timer [Таймер – формирователь задержки выключения]	SF	6–15
On–Delay Timer [Таймер – формирователь задержки включения]	SD	6–11
Ones Complement Double Integer (32–Bit) [Дополнение до 1 двойного целого числа (32 бита)]	INVD	12–14
Ones Complement Integer (16–Bit) [Дополнение до 1 целого числа (16 битов)]	INVI	12–14
Open a Data Block [Открыть блок данных]	OPN	15–2
Or [ИЛИ]	O	5–10
Or Double Word (32–Bit) [Поразрядное ИЛИ с двойными словами (32 бита)]	OD	13–6
Or Not [ИЛИ-НЕ]	ON	5–8
Or Not with Nesting Open [ИЛИ-НЕ с открытием вложения]	ON(	5–14

Таблица A–3. Команды *Списка операторов*, упорядоченные по алфавиту в соответствии с их полными международными именами

Наименование команды	Мнемони-ческое обозначение	Стр.
Or with Nesting Open [ИЛИ с открытием вложения]	O(	5–14
OR Word (16–Bit) [Поразрядное ИЛИ со словами (16 битов)]	OW	13–3
Program Display Instruction [Команда программирования изображения]	BLD	4–2
Pulse Timer [Таймер – формирователь импульса]	SP	6–7
Reset [Сбросить]	R	5–22
Reset Counter (where the current counter can have a number in the range of 0 to 255, for example: R C 15) [Сбросить счетчик (где текущий счетчик может иметь номер в диапазоне от 0 до 255, например: R C 15)]	R	6–5
Reset Timer (where the current timer can have a number in the range of 0 to 255, for example: R T 32) [Сбросить таймер (где текущий таймер может иметь номер в диапазоне от 0 до 255, например: R T 32)]	R	7–4
Restore RLO, End MCR [Восстановить RLO, завершить MCR]	)MCR	17–11
Retentive On–Delay Timer [Таймер – формирователь задержки включения с запоминанием]	SS	6–13
Rotate Accumulator 1 Left via CC 1 (32–Bit) [Выполнить циклический сдвиг аккумулятора 1 влево через CC 1 (32 бита)]	RLDA	14–6
Rotate Accumulator 1 Right via CC 1 (32–Bit) [Выполнить циклический сдвиг аккумулятора 1 вправо через CC 1 (32 бита)]	RRDA	14–6
Rotate Left Double Word (32–Bit) [Выполнить циклический сдвиг двойного слова влево (32 бита)]	RLD	14–8
Rotate Right Double Word (32–Bit) [Выполнить циклический сдвиг двойного слова вправо (32 бита)]	RRD	14–8
Round [Округлить]	RND	12–9
Round to Lower Double Integer [Округлить до ближайшего меньшего двойного целого числа]	RND–	12–11
Round to Upper Double Integer [Округлить до ближайшего большего двойного целого числа]	RND+	12–10
Save RLO in BR Register [Сохранить RLO в регистре BR]	SAVE	5–26
Save RLO in MCR Stack, Begin MCR [Сохранить RLO в стеке MCR, начать MCR]	MCR(	17–11
Set [Установить]	S	5–21
Set Counter Preset Value (where the current counter can have a number in the range of 0 to 255, for example: S C 15) [Установить начальное значение счетчика (где текущий счетчик может иметь номер в диапазоне от 0 до 255, например: S C 15)]	S	7–3
Set RLO (= 1) [Установить RLO (= 1)]	SET	5–26
Shift Left Double Word (32–Bit) [Сдвинуть влево двойное слово (32 бита)]	SLD	14–2
Shift Left Word (16–Bit) [Сдвинуть влево слово (16 битов)]	SLW	14–2
Shift Right Double Word (32–Bit) [Сдвинуть вправо двойное слово (32 бита)]	SRD	14–3
Shift Right Word (16–Bit) [Сдвинуть вправо слово (16 битов)]	SRW	14–2
Shift Sign Double Integer (32–Bit) [Сдвинуть двойное целое число со знаком (32 бита)]	SSD	14–4
Shift Sign Integer (16–Bit) [Сдвинуть целое число со знаком (16 битов)]	SSI	14–4
Sine of a Floating–Point Number (32–Bit IEEE FP) [Синус числа с плавающей точкой (32 бита, IEEE FP)]	SIN	10–7
Square of a Floating–Point Number (32–Bit IEEE PF) [Квадрат числа с плавающей точкой (32 бита, IEEE PF)]	SQR	10–9
Square Root of a Floating–Point Number (32–Bit IEEE PF) [Квадратный корень числа с плавающей точкой (32 бита IEEE PF)]	SQRT	10–9
Subtract Accumulator 1 from Accumulator 2 as Double Integer (32–Bit)	–D	9–2

Таблица A–3. Команды *Списка операторов*, упорядоченные по алфавиту в соответствии с их полными международными именами

Наименование команды	**Мнемони-ческое обозначение**	**Стр.**
[Вычесть аккумулятор 1 из аккумулятора 2 как двойные целые числа (32 бита)]		
Subtract Accumulator 1 from Accumulator 2 as Integer (16–Bit) [Вычесть аккумулятор 1 из аккумулятора 2 как целые числа (16 битов)]	–I	9–2
Subtract Accumulator 1 from Accumulator 2 as Real (32–Bit IEEE FP) [Вычесть аккумулятор 1 из аккумулятора 2 как вещественные числа (32 бита, IEEE FP)]	–R	10–2
Tangent of a Floating–Point Number (32–Bit IEEE FP) [Тангенс числа с плавающей точкой (32 бита, IEEE FP)]	TAN	10–7
Toggle Accumulator 1 with Accumulator 2 [Обменять аккумулятор 1 с аккумулятором 2]	TAK	4–2
Transfer [Передать]	T	8–3
Transfer Accumulator 1 to Status Word (T STW) [Передать аккумулятор 1 в слово состояния]	T	8–6
Transfer Address Register 1 to ... (to address indicated) [Передать адресный регистр 1 в... (указанный адрес)]	TAR1	8–11
Transfer Address Register 1 to Accumulator 1 (if no address is indicated) [Передать адресный регистр 1 в аккумулятор 1 (если не указан адрес)]	TAR1	8–11
Transfer Address Register 1 to Address Register 2 (T AR1 AR2) [Передать адресный регистр 1 в адресный регистр 2]	TAR1	8–11
Transfer Address Register 2 to ... (to address indicated) [Передать адресный регистр 2 в... (указанный адрес)]	TAR2	8–11
Transfer Address Register 2 to Accumulator 1 (if no address is indicated) [Передать адресный регистр 2 в аккумулятор 1 (если не указан адрес)]	TAR2	8–11
Truncate [Округлить до целого отбрасыванием младших разрядов]	TRUNC	12–12
Twos Complement Double Integer (32–Bit) [Дополнение до 2 двойного целого числа (32 бита)]	NEGD	12–14
Twos Complement Integer (16–Bit) [Дополнение до 2 целого числа (16 битов)]	NEGI	12–14
Unconditional Call [Безусловный вызов блока]	UC	17–7

B Примеры программирования

Обзор главы

B.1 Обзор

Практические применения

Каждая из описанных в данном руководстве команд языка *Список операторов* запускает определенную операцию. Объединяя эти команды в программу, вы можете решать широкий спектр задач автоматизации. Эта глава дает вам следующие примеры практического применения команд *Списка операторов:*

- Управление лентой транспортера с помощью битовых логических операций
- Определение направления движения ленты транспортера с помощью битовых логических операций
- Генерирование тактовых импульсов с помощью таймерных команд
- Контроль зоны хранения с помощью операций счета и сравнения
- Решение задачи с помощью арифметических операций для целых чисел
- Установка длительности времени нагрева печи

При создании ASCII-файла, который будет импортироваться в редактор AWL вы должны следовать соглашениям, описанным в приложении C.

Используемые команды

Примеры в этой главе используют следующие команды:

- Арифметические операции с целыми числами (16 битов)
  - Деление аккумулятора 2 на аккумулятор 1 как целых чисел (/I)
  - Сложение аккумуляторов 1 и 2 как целых чисел (+I)
  - Умножение аккумуляторов 1 и 2 как целых чисел (*I)
- Загрузить (L) и Передать (T)
- И (A) и И-НЕ (AN)
- ИЛИ (O) и ИЛИ-НЕ (ON)
- Инвертировать RLO (NOT)
- Конец блока (BE) и Конец бока условный (BEC)
- Положительный фронт (FP)
- Поразрядные логические операции И и ИЛИ со словами
- Присвоить (=)
- Сравнить целые числа (16 битов, <=, >=)
- Счетчик обратного счета (CD) и Счетчик прямого счета (CU)
- Таймер – формирователь удлиненного импульса (SE)
- Увеличить аккумулятор 1 (INC)
- Установить (S) и Сбросить (R)

B.2 Битовые логические операции

Управление лентой транспортера

На рисунке B -1 представлена лента транспортера, которая может приводиться в движение с помощью электродвигателя. В начале ленты находятся две кнопки: S1 – ПУСК и S2 – СТОП. В конце ленты также находятся две кнопки: S3 – ПУСК и S4 – СТОП. Лента может запускаться или останавливаться с любого конца. Кроме того, датчик S5 останавливает ленту, если предмет на ленте доходит до конца.

Символическое программирование

Вы можете составить программу управления лентой транспортера, показанной на рис. B-1, представив различные компоненты конвейерной системы с помощью символов. Если вы выберете этот метод, вы должны создать таблицу символов, чтобы увязать выбранные вами символы с абсолютными значениями (см. таблицу B–1). Таблица B–3 сравнивает программу на языке *Список операторов*, использующую в качестве адресов символы, с программой, использующей в качестве адресов абсолютные значения. Символы определяются в таблице символов (см. оперативную помощь STEP 7).

Таблица B–1. Элементы символического программирования для конвейерной системы

Компонент системы	Абсолютный адрес	Символ	Таблица символов
Кнопка ПУСК	I 1.1	S1	I 1.1 S1
Кнопка СТОП	I 1.2	S2	I 1.2 S2
Кнопка ПУСК	I 1.3	S3	I 1.3 S3
Кнопка СТОП	I 1.4	S4	I 1.4 S4
Датчик	I 1.5	S5	I 1.5 S5
Двигатель	Q 4.0	MOTOR_ON	Q 4.0 MOTOR_ON

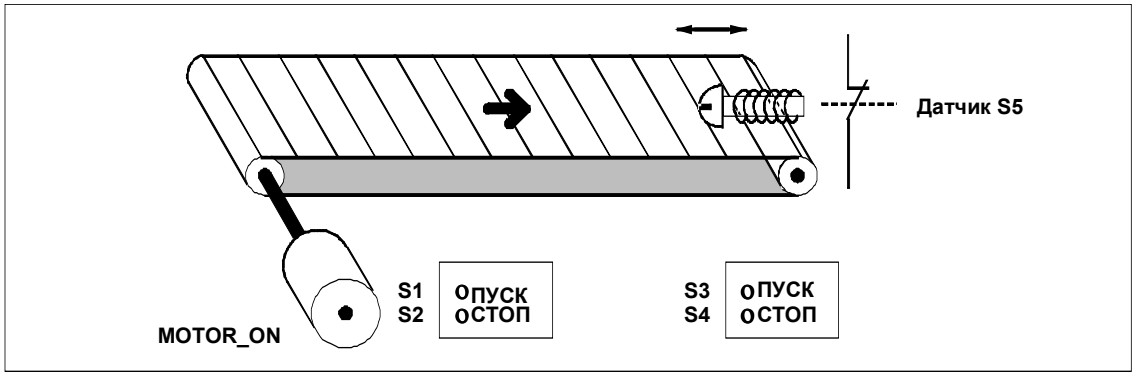


Рис. B–1. Конвейерная система

Абсолютное программирование

Вы можете написать программу для управления лентой транспортера, показанного на рис. B-1, используя абсолютные значения для представления различных компонентов конвейерной системы (см. таблицу B–2). Таблица B–3 сравнивает программу на языке *Список операторов*, использующую в качестве адресов абсолютные значения, с программой, использующей в качестве адресов символы. Вслед за таблицами приведено объяснение программы.

Таблица B–2. Элементы абсолютного программирования для конвейерной системы

Компонент системы	Абсолютный адрес
Кнопка ПУСК	I 1.1
Кнопка СТОП	I 1.2
Кнопка ПУСК	I 1.3
Кнопка СТОП	I 1.4
Датчик	I 1.5
Двигатель	Q 4.0

Таблица B–3. Символическая и абсолютная программы для управления лентой транспортера

Символическая программа	Абсолютная программа
O S1	O I 1.1
O S3	O I 1.3
S MOTOR_ON	S Q 4.0
O S2	O I 1.2
O S4	O I 1.4
ON S5	ON I 1.5
R MOTOR_ON	R Q 4.0

AWL		Объяснение
O	**I 1.1**	**Нажатие любой кнопки ПУСК включает двигатель.**
O	**I 1.3**	
S	**Q 4.0**	
O	**I 1.2**	**Нажатие любой кнопки Стоп или открытие нормально замкнутого контакта в конце ленты выключает двигатель.**
O	**I 1.4**	
ON	**I 1.5**	
R	**Q 4.0**	

Определение направления движения ленты транспортера

На рис. B-2 показана лента транспортера, которая оснащена двумя фотоэлектрическими датчиками (PEB1 и PEB2), спроектированными для определения направления, в котором перемещается пакет на ленте. Каждый из фотодатчиков работает как нормально открытый контакт (см. раздел 5.1).

Символическое программирование

Вы можете написать программу, которая активизирует указатель направления движения конвейерной системы, изображенной на рис. B-2, используя символы, представляющие различные компоненты конвейерной системы, включая фотоэлектрические датчики для обнаружения направления. Если вы выберете этот метод, вы должны создать таблицу символов, связывающую выбранные вами символы с абсолютными значениями (см. таблицу B–4). Таблица B–6 сравнивает программу на языке *Список операторов*, использующую в качестве адресов символы, с программой, использующей в качестве адресов абсолютные значения. Символы определяются в таблице символов (см. оперативную помощь STEP 7).

Таблица B–4. Элементы символического программирования для определения направления

Компонент системы	Абсолютный адрес	Символ	Таблица символов
Фотодатчик 1	I 0.0	PEB1	I 0.0 PEB1
Фотодатчик 2	I 0.1	PEB2	I 0.1 PEB2
Указатель движения вправо	Q 4.0	RIGHT	Q 4.0 RIGHT
Указатель движения влево	Q 4.1	LEFT	Q 4.1 LEFT
Импульсный бит памяти 1	M 0.0	PMB1	M 0.0 PMB1
Импульсный бит памяти 2	M 0.1	PMB2	M 0.1 PMB2

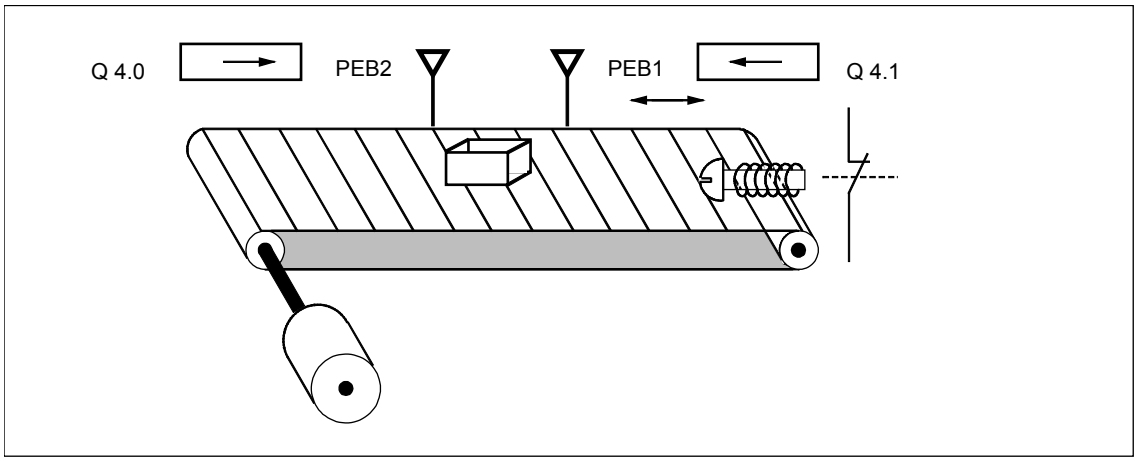


Рис. B–2. Конвейерная система с фотоэлектрическими датчиками для определения направления движения

Абсолютное программирование

Вы можете написать программу для управления указателем направления движения ленты транспортера, показанной на рис. B–2, используя абсолютные значения, представляющие фотодатчики для определения направления движения (см. таблицу B–5). Таблица B–6 сравнивает программу на языке *Список операторов*, использующую в качестве адресов абсолютные значения, с программой, использующей в качестве адресов символы. Объяснение программы следует за рисунком.

Таблица B–5. Элементы абсолютного программирования для определения направления

Компонент системы	Абсолютный адрес
Фотодатчик 1	I 0.0
Фотодатчик 2	I 0.1
Указатель движения вправо	Q 4.0
Указатель движения влево	Q 4.1
Импульсный бит памяти 1	M 0.0
Импульсный бит памяти 2	M 0.1

Таблица B–6. Символическая и абсолютная программы для определения направления

Символическая программа	Абсолютная программа
A I PEB1	A I 0.0
FP PMB1	FP M 0.0
AN PEB2	AN I 0.1
S LEFT	S Q 4.1
A PEB2	A I 0.1
FP PMB2	FP M 0.1
AN PEB1	AN I 0.0
S RIGHT	S Q 4.0
AN PEB1	AN I 0.0
AN PEB2	AN I 0.1
R RIGHT	R Q 4.0
R LEFT	R Q 4.1

AWL		Объяснение
A	**I 0.0**	**Если имеет место переход состояния сигнала с 0 на 1 (положительный**
FP	**M 0.0**	**фронт) на входе I 0.0**
AN	**I 0.1**	**и при этом состояние сигнала на входе I 0.1 равно 0, то**
S	**Q 4.1**	**пакет на ленте движется влево.**
A	**I 0.1**	**Если имеет место переход состояния сигнала с 0 на 1 (положительный**
FP	**M 0.1**	**фронт) на входе I 0.1**
AN	**I 0.0**	**и при этом состояние сигнала на входе I 0.0 равно 0, то**
S	**Q 4.0**	**пакет на ленте движется вправо. Если световой барьер одного из фотодатчиков прерывается, это значит, что пакет находится между датчиками.**
AN	**I 0.0**	**Если ни один из фотобарьеров не нарушен, то пакета между датчиками**
AN	**I 0.1**	**нет.**
R	**Q 4.0**	**Указатель направления выключается.**
R	**Q 4.1**	

B.3 Таймерные команды

Генератор тактовых импульсов

Вы можете использовать генератор тактовых импульсов, или мигающее реле, для создания периодически повторяющегося сигнала. Генератор тактовых импульсов часто используется в системах сигнализации, управляющих миганием индикаторных ламп.

При использовании S7–300 вы можете реализовать функцию генератора тактовых импульсов, применив управляемую временем обработку в специальных организационных блоках. Пример, показанный в следующей программе на языке *Список операторов*, однако, иллюстрирует использование таймерных функций для генерирования тактовых импульсов.

Следующий пример показывает, как реализовать тактовый генератор в режиме свободных колебаний с помощью таймера (относительная длительность импульсов 1:1). Частота принимает значения, приведенные в таблице B–7.

AWL		Объяснение
AN	**T 1**	**Если таймер T 1 закончил работу,**
L	**S5T#250ms**	**загрузить в T 1 значение времени 250 мс и**
SE	**T 1**	**запустить T 1 как формирователь продленного импульса.**
NOT		**Инвертировать результат логической операции.**
BEC		**Если таймер работает, завершить текущий блок. Если таймер завершил**
L	**MB100**	**работу, загрузить содержимое байта памяти MB100,**
INC	**1**	**увеличить это содержимое на 1 и**
T	**MB100**	**передать результат в байт памяти MB100.**

Опрос сигнала таймера T 1 определяет результат логической операции.

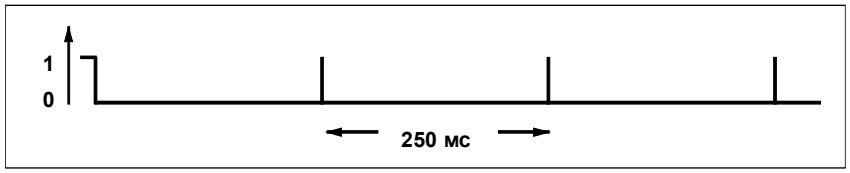


Рис. B–3. RLO для оператора AN T 1 в примере генератора тактовых импульсов

Как только время таймера истекает, таймер запускается вновь. Поэтому опрос сигнала, который выполняется оператором AN T 1, выдает состояние сигнала "1" очень кратковременно.

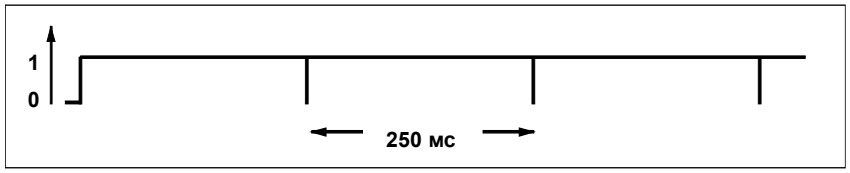


Рис. B–4. Инвертированный бит RLO таймера T 1 в примере генератора тактовых импульсов

Каждые 250 мс бит RLO становится равным 0. Рис. B–4 показывает, как выглядит инвертированный бит RLO. Тогда команда BEC не завершает обработку блока. Вместо этого увеличивается на 1 содержимое байта памяти MB100.

Содержимое байта памяти MB100 меняется каждые 250 мс следующим образом: 0 →1→2→3→ ... →254→255→0→1 ...

Достижение определенной частоты

В таблице B–7 перечислены частоты, которые вы можете получить из отдельных битов байта памяти MB100. Программа на языке *Список операторов*, следующая за таблицей, показывает, как можно использовать сгенерированные частоты.

Таблица B–7. Частоты для примера генератора тактовых импульсов

Биты MB100	Частота в герцах	Длительность
M 100.0	2,0	0,5 с (250 мс вкл./250 мс выкл.)
M 100.1	1,0	1 с (0.5 с вкл./0.5 с выкл.)
M 100.2	0,5	2 с (1 с вкл./1 с выкл.)
M 100.3	0,25	4 с (2 с вкл./2 с выкл.)
M 100.4	0,125	8 с (4 с вкл./4 с выкл.)
M 100.5	0,0625	16 с (8 с вкл./8 с выкл.)
M 100.6	0,03125	32 с (16 с вкл./16 с выкл.)
M 100.7	0,015625	64 с (32 с вкл./32 с выкл.)

AWL		**Объяснение**
A	**M 10.0**	**M 10.0 = 1, когда появляется неисправность.**
A	**M 100.1**	
=	**Q 4.0**	**Лампа, сигнализирующая о неисправности, мигает с частотой 1 Гц при ее появлении.**

В таблице B–8 приведен перечень состояний сигнала битов байта памяти MB100. Рис. B–5 показывает состояние бита памяти M100.1.

Таблица B–8. Состояния битов байта памяти MB100

Цикл	**Состояние сигнала битов байта памяти MB100**								**Значение времени в мс**
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
0	0	0	0	0	0	0	0	0	250
1	0	0	0	0	0	0	0	1	250
2	0	0	0	0	0	0	1	0	250
3	0	0	0	0	0	0	1	1	250
4	0	0	0	0	0	1	0	0	250
5	0	0	0	0	0	1	0	1	250
6	0	0	0	0	0	1	1	0	250
7	0	0	0	0	0	1	1	1	250
8	0	0	0	0	1	0	0	0	250
9	0	0	0	0	1	0	0	1	250
10	0	0	0	0	1	0	1	0	250
11	0	0	0	0	1	0	1	1	250
12	0	0	0	0	1	1	0	0	250

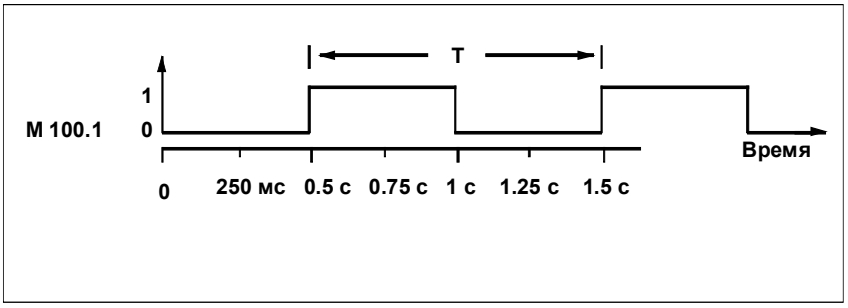


Рис. B–5. Состояние сигнала бита 1 байта MB100 (M 100.1)

B.4 Операции счета и сравнения

Зона хранения со счетчиком и компаратором

Рис. B-6 показывает систему с двумя конвейерами и зоной временного хранения между ними. Конвейер 1 транспортирует пакеты к зоне хранения. Фотодатчик в конце конвейера 1 рядом с зоной хранения определяет, сколько пакетов доставлено в зону хранения. Конвейер 2 транспортирует пакеты из зоны временного хранения к погрузочной площадке, где грузовые автомобили забирают пакеты для доставки их клиентам. Фотодатчик в конце конвейера 2 у зоны временного хранения определяет, сколько пакетов покидает зону хранения для отправки на погрузочную площадку.

Информационное табло с пятью лампочками показывает уровень заполнения зоны временного хранения. Программа, приведенная в качестве примера вслед за рис. B-6, активизирует индикаторные лампы на информационном табло.

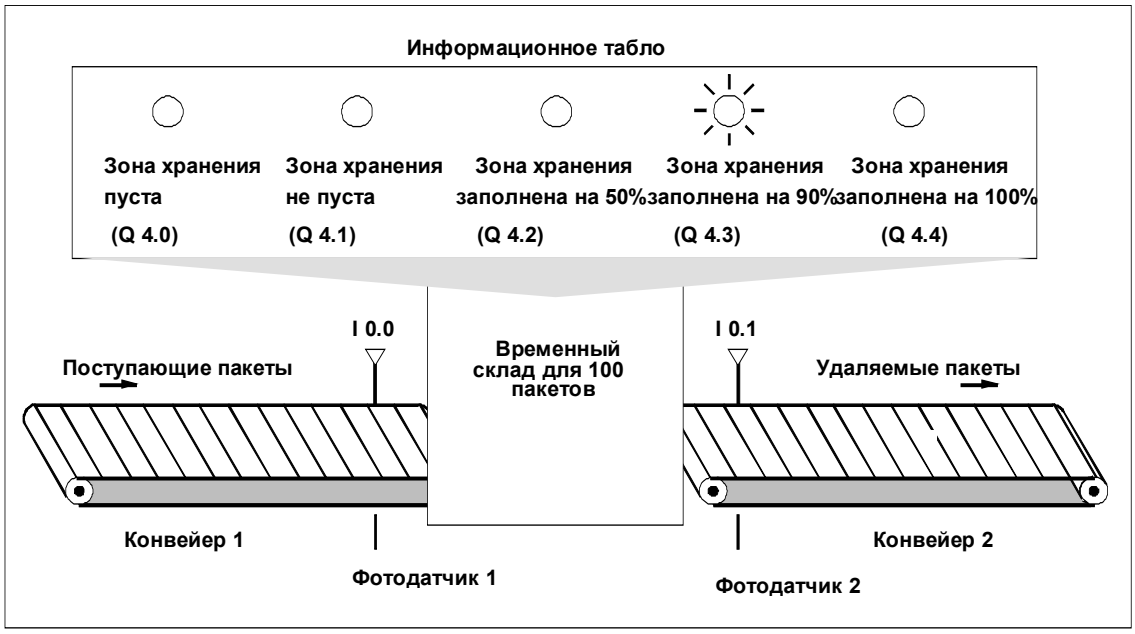


Рис. B–6. Зона хранения со счетчиком и компаратором

AWL	Объяснение
A I 0.0 **CU C 1**	**Каждый импульс, сгенерированный фотодатчиком 1, увеличивает значение счетчика C 1 на единицу, подсчитывая тем самым количество пакетов, поступающих в зону хранения.**
A I 0.1 **CD C 1**	**Каждый импульс, сгенерированный фотодатчиком 2, уменьшает значение счетчика C 1 на единицу, подсчитывая тем самым количество пакетов, покидающих зону хранения.**
AN C 1 **= Q 4.0**	**Если значение счетчика равно 0, загорается индикаторная лампа «Зона хранения пуста».**
A C 1 **= Q 4.1**	**Если значение счетчика не равно 0, загорается индикаторная лампа «Зона хранения не пуста».**
L +50 **L C 1** **<=I** **= Q 4.2**	**Если 50 меньше или равно значению счетчика, то загорается индикаторная лампа «Зона хранения заполнена на 50%».**
L +90 **>=I** **= Q 4.3**	**Если значение счетчика больше или равно 90, то загорается индикаторная лампа «Зона хранения заполнена на 90%».**
L C 1 **L 100** **>=I** **= Q 4.4**	**Если значение счетчика больше или равно 100, то загорается индикаторная лампа «Зона хранения заполнена на 100%». (Вам следует также использовать выход Q 4.4 для блокировки конвейера 1.)**

B.5 Арифметические операции с целыми числами

Решение математической задачи

Следующий пример программы (применимый только для S7–300) показывает, как использовать три команды над целыми числами с командами загрузки и передачи для получения того же результата, который дает следующее уравнение:

$$MD4 = \frac{(IW0 + DBW3) \times 15}{MW2}$$

AWL		Объяснение
L	**IW0**	**Загрузить значение из входного слова IW0 в аккумулятор 1.**
L	**DB5.DBW3**	**Загрузить значение из слова глобальных данных DBW3 блока DB5 в аккумулятор. Старое содержимое аккумулятора 1 перемещается в аккумулятор 2.**
+I		**Сложить содержимое младших слов аккумуляторов 1 и 2. Результат сохраняется в младшем слове аккумулятора 1. Содержимое аккумулятора 2 и старшего слова аккумулятора 1 остается неизменным.**
L	**+15**	**Загрузить постоянное значение +15 в аккумулятор 1. Старое содержимое аккумулятора 1 перемещается в аккумулятор 2.**
***I**		**Умножить содержимое младшего слова аккумулятора 2 на содержимое младшего слова аккумулятора 1. Результат сохраняется в аккумуляторе 1. Содержимое аккумулятора 2 остается неизменным.**
L	**MW2**	**Загрузить значение из слова памяти MW2 в аккумулятор 1. Старое содержимое аккумулятора 1 перемещается в аккумулятор 2.**
/I		**Разделить содержимое младшего слова аккумулятора 2 на содержимое младшего слова аккумулятора 1. Результат сохраняется в аккумуляторе 1. Содержимое аккумулятора 2 остается неизменным.**
T	**MD4**	**Передать окончательный результат в двойное слово памяти MD4. Содержимое обоих аккумуляторов остается неизменным.**

На рис. B–7 показана связь программы с уравнением.

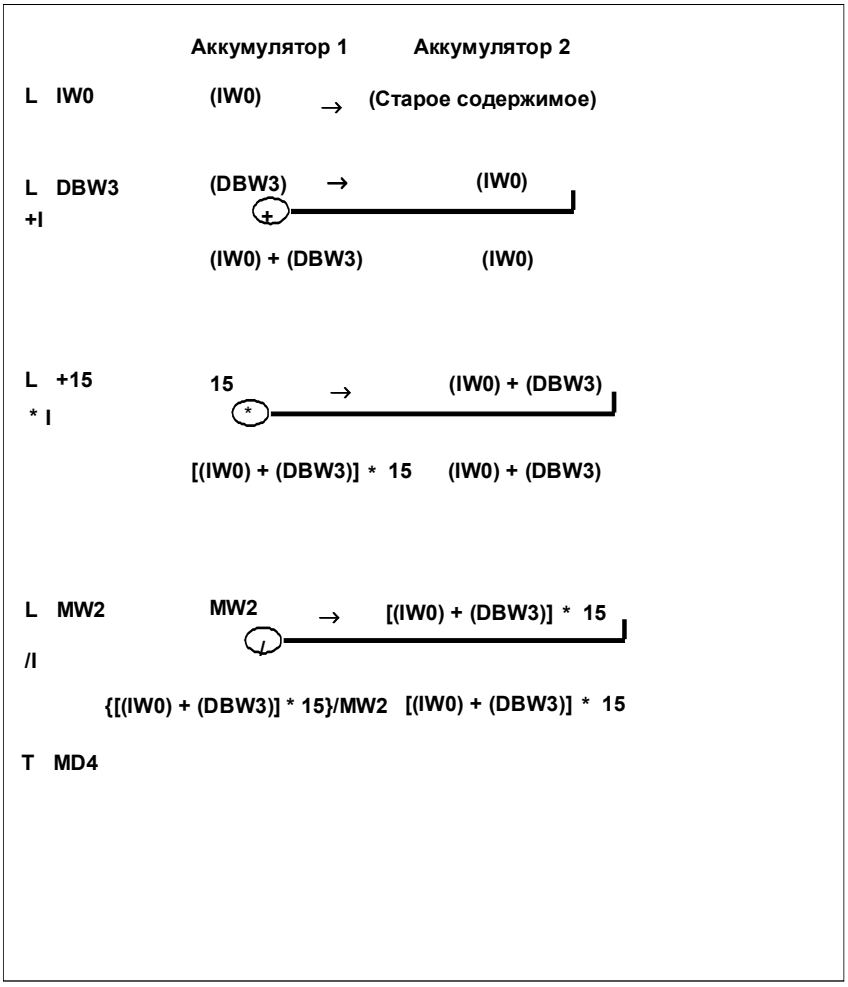


Рис. B–7. Связь операторов арифметики с целыми числами с уравнением (S7–300)

B.6 Логические операции со словами

Нагревание печи

Оператор показанной на рис. B–8 печи запускает нагрев печи, нажимая кнопку ПУСК. Оператор может устанавливать длительность нагревания с помощью переключателей, показанных на рисунке. Значение, устанавливаемое оператором, указывает секунды в двоично-десятичном (BCD) формате. Таблица B–9 перечисляет компоненты системы нагрева и соответствующие им абсолютные адреса, используемые в примере программы, следующем за рис. B–8.

Таблица B–9. Компоненты системы нагрева и соответствующие абсолютные адреса

Компонент системы	Абсолютный адрес в программе AWL
Кнопка ПУСК	I 0.7
Переключатель для единиц	от I 1.0 до I 1.3
Переключатель для десятков	от I 1.4 до I 1.7
Переключатель для сотен	от I 0.0 до I 0.3
Нагрев начат	Q 4.0

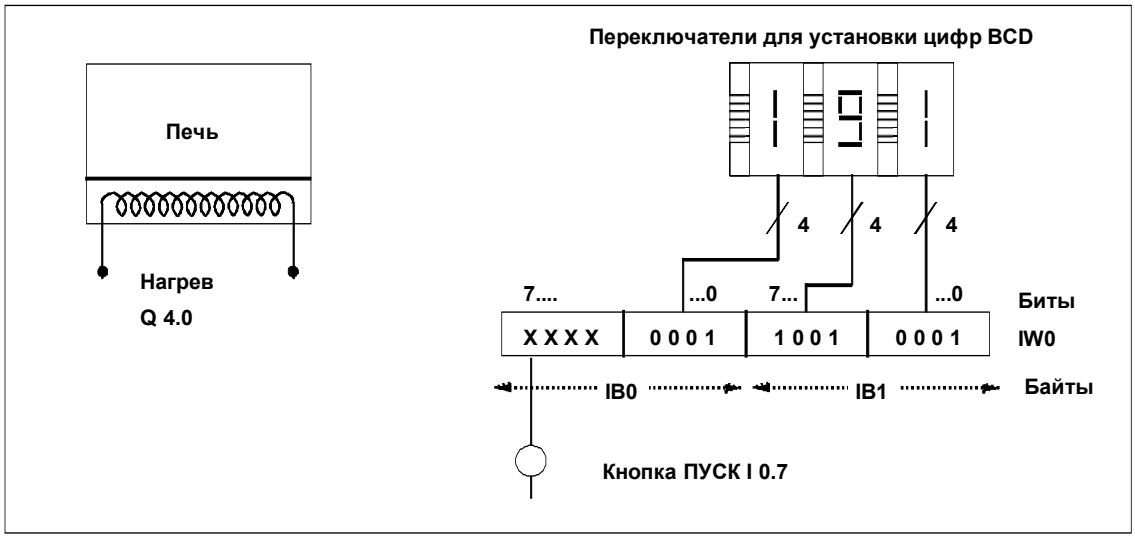


Рис. B–8. Использование входов и выходов для ограниченного временем процесса нагрева

AWL		Объяснение
A	**T 1**	**Если таймер работает, включить нагрев.**
=	**Q 4.0**	
BEC		**Если таймер работает, то закончить обработку здесь. Это предотвращает повторный запуск таймера T 1, если кнопка ПУСК еще нажата.**
L	**IW0**	**Замаскировать входные биты с I 0.4 по I 0.7 (т.е. сбросить их в 0). Значение времени в секундах находится в младшем слове аккумулятора 1 в двоично-десятичном формате.**
AW	**W#16#0FFF**	
OW	**W#16#2000**	**Назначить базу времени в виде секунд в битах 12 и 13 младшего слова аккумулятора 1.**
A	**I 0.7**	**Запустить таймер T 1 как формирователь продленного импульса, если нажата кнопка.**
SE	**T 1**	
BE		**Завершить сегмент программы.**

C Зарезервированные ключевые слова, используемые в исходных файлах

Определение

Ключевое слово – это зарезервированный идентификатор, который не может быть использован в качестве обычного идентификатора.

Для использования ключевого слова в качестве глобального символа оно должно быть помечено в качестве контрольной точки цикла сканирования (SCC).

Для использования ключевого слова в качестве локального символа оно должно быть помечено #.

Обзор

Таблица C -1 показывает ключевые слова, зарезервированные в STEP 7.

Таблица C–1. Ключевые слова

Ключевые слова	
A	B
AB	BEGIN
AD	BIE
ANY	BLOCK_DB
AO	BLOCK_FB
AR1	BLOCK_FC
AR2	BLOCK_SDB
ARRAY	BOOL
AUTHOR	BYTE
AW	
C	DATA_BLOCK
CALL	DATE
CHAR	DATE_AND_TIME
COUNTER	DB
	DBB
	DBD
	DBLG
	DBNO
	DBW
	DBX
	DI

Таблица C–1. Ключевые слова	
	Ключевые слова
	DIB
	DID
	DILG
	DINO
	DINT
	DIW
	DIX
	DT
	DWORD
E	FALSE
EB	FAMILY
ED	FB
END_Data_Block	FC
END_Function	FUNCTION
END_Function_Block	FUNCTION_BLOCK
END_Organization_Block	
END_Struct	I
END_System_Function	IB
END_System_Function_Block	ID
END_Type	INT
END_VAR	IW
EW	
KA	L
KNOW_HOW_PROTECT	LB
KP	LD
	LW
M	NAME
MB	NETWORK
MD	NI
MW	NO
OB	PA
OF	PAB
ORGANIZATION_BLOCK	PAD
OS	PAW
OV	PE
	PEB
	PED
	PEW
	PI
	PIB
	PID
	PIW
	PQ
	PQB
	PQD
	PQW
	POINTER
Q	READ_ONLY
QB	REAL
QD	RET_VAL

Таблица C–1. Ключевые слова	
Ключевые слова	
QW	
S5T	T
S5TIME	TIME
SDB	TIME_OF_DAY
SFB	TIMER
SFC	TITLE
STANDARD	TOD
STRING	TRUE
STRUCT	TYPE
STW	
SYSTEM_FUNCTION	
SYSTEM_FUNCTION_BLOCK	
UDT	VAR
UNLINKED	VAR_IN_OUT
UO	VAR_INPUT
	VAR_OUTPUT
	VAR_TEMP
	VERSION
	VOID
WORD	Z

D Литература

/30/ Getting Started: Working with STEP 7 V5.0
[Введение: Работа со STEP 7 V5.0]

/70/ Руководство: *S7–300 Programmable Controller,*
Hardware and Installation
[*Программируемый контроллер S7-300*.
Аппаратура и установка]

/71/ Справочное руководство: *S7–300, M7–300 Programmable Controllers*
Module Specifications
[*Программируемые контроллеры S7-300, M7-300*
Описания модулей]

/72/ Instruction List: *S7–300 Programmable Controller*
[Список команд: Программируемый контроллер S7-300]

/100/ Руководство: *S7–400/M7–400 Programmable Controllers,*
Hardware and Installation
[*Программируемые контроллеры S7-400, M7-400*.
Аппаратура и установка]

/101/ Справочное руководство: *S7–400/M7–400 Programmable Controllers*
Module Specifications
[*Программируемые контроллеры S7-400, M7-400*
Описания модулей]

/102/ Instruction List: *S7–400 Programmable Controller*
[Список команд: Программируемый контроллер S7-400]

/231/ Руководство:*Configuring Hardware and Communication Connections, STEP 7 V5.0*
[Конфигурирование аппаратуры и проектирование соединений, STEP 7 V5.0]

/233/ Справочное руководство: *Ladder Logic (LAD) for S7–300 and S7–400* Programming
[*Контактный план (КОР) для S7–300 и S7–400. Программирование*]

/234/ Руководство: *Programming with STEP 7 V5.0*
[Программирование с помощью STEP 7 V5.0]

/235/ Справочное руководство: *System Software for S7–300 and S7–400* System and Standard Functions
[*Системное программное обеспечение для S7-300 и S7-400*
Системные и стандартные функции]

/236/ Руководство: FBD *for S7–300 and 400,*
Programming
[*FUP для S7-300 и 400, Программирование*]

/250/ Руководство: *Structured Control Language (SCL) for S7–300/S7–400,* Programming
[*Язык структурного управления (SCL) для S7-300 и S7-400*, Программирование]

/251/ Руководство: *S7–GRAPH for S7–300 and S7–400,*
Programming Sequential Control Systems
[S7-*GRAPH для S7-300 и S7-400*,
Программирование систем последовательного управления]

/252/ Руководство: *S7–HiGraph for S7–300 and S7–400,*
Programming State Graphs
[S7-*HiGraph для S7-300 и S7-400,*
Программирование графов состояния]

/253/ Руководство: *C Programming for S7–300 and S7–400,*
Writing C Programs
[*Программирование на языке C для S7-300 и S7-400,*
Написание программ на языке C]

/254/ Руководство: *Continuous Function Charts (CFC) for S7 and M7,*
Programming Continuous Function Charts
[*Непрерывные функциональные схемы (CFC) для S7 и M7,*
Программирование непрерывных функциональных схем]

//270/ Руководство: S7–PDIAG for S7–300 and S7–400
«Configuring Process Diagnostics for LAD, STL, and FBD»
[*S7-PDIAG для S7-300 и S7-400,* "Проектирование диагностики процесса для контактного плана, списка команд и функционального плана"]

/271/ Руководство: *NETPRO,*
«Configuring Networks»
[*NETPRO, «*Проектирование сетей»]

/800/ *DOCPRO*
Creating Wiring Diagrams (CD only)
[*DOCPRO*
Создание коммутационных схем (только на CD)]

/801/ *TeleService for S7, C7 and M7*
Remote Maintenance for Automation Systems (CD only)
[*TeleService для S7, C7 и M7*
Дистанционное обслуживание систем автоматизации]

/802/ PLC Simulation for S7–300 and S7–400 (CD only)
[Имитация ПЛК для S7-300 и S7-400 (только на CD)]

/803/ Справочное руководство: *Standard Software for S7–300 and S7–400,*
STEP 7 Standard Functions, Part 2
[*Стандартное программное обеспечение для S7-300 и S7-400,*
Стандартные функции STEP 7, часть 2]

Глоссарий

А

Абсолютная адресация

При абсолютной адресации указывается расположение в памяти операнда, подлежащего обработке.

Адрес

Адрес (операнд) – это часть оператора STEP 7, которая указывает, с чем процессор должен выполнить команду. Адреса могут быть абсолютными или символическими.

Адресный регистр

Адресные регистры – это регистры в комуникационной части CPU. Они действуют как указатели для косвенной адресации через регистры (возможны в AWL).

Аккумулятор

Аккумуляторы - это регистры в CPU, которые служат в качестве промежуточной памяти для операций загрузки, передачи, сравнения, а также арифметических операций и операций преобразования.

Б

Бит переполнения

Бит состояния OV означает “переполнение”. Переполнение может возникнуть, например, после выполнения математической операции.

Бит состояния

Бит состояния (STA) хранит значение бита, к которому производится обращение. Состояние логической операции, имеющей доступ к памяти на чтение (A, AN, O, ON, X, XN) всегда равно значению бита, опрашиваемого этой операцией (бита, с которым выполняется логическая операция). Состояние логической операции, имеющей доступ к памяти на запись (S, R, =), равно значению бита, в который осуществляется запись. Если запись не имеет места, то оно равно значению бита, к которому операция обращается. Бит состояния не имеет значения для логических операций, которые не обращаются к памяти. Эти операции устанавливают бит состояния в 1 (STA = 1). Бит состояния не опрашивается командой. Он интерпретируется только во время тестирования программы (статус программы).

Бит сохраняемого переполнения

Бит состояния OS – это «бит сохраняемого переполнения слова состояния». Переполнение может иметь, например, место после выполнения математической операции.

Бит OR

Бит OR необходим при выполнении логической операции И перед операцией ИЛИ. Бит OR показывает этим командам, что ранее выполненная функция И дала значение 1, тем самым предсказывая результат логической операции ИЛИ. Любая другая команда, обрабатывающая биты, сбрасывает бит OR (см. раздел 5.4).

Блок данных (DB)

Блоки данных (DB) - это области данных в программе пользователя, содержащие данные пользователя. Имеются глобальные (совместно используемые) блоки данных, к которым могут обращаться любые логические блоки, и экземплярные блоки данных, связанные с определенным вызовом функционального блока (FB). В отличие других блоков, блоки данных не содержат команд.

В

Ввод, пошаговый

При пошаговом (инкрементном) вводе блока каждая строка или элемент немедленно проверяется на наличие ошибок (например, синтаксических). Если обнаружена ошибка, то она выделяется и должна быть исправлена до завершения программирования. Пошаговый ввод возможен при программировании в виде списка операторов (STL, AWL), контактного плана (LAD, KOP) и функционального плана (FBD, FUP).

Г

Глобальный блок данных (DB)

Глобальный (совместно используемый) блок данных – это DB, адрес которого при открытии загружается в адресный регистр DB. Он предоставляет память и данные для всех исполняющихся логических блоков (FC, FB и OB).

В противоположность этому, экземплярный DB проектируется для использования в качестве места хранения данных для FB, которому он поставлен в соответствие.

Д

Данные, статические

Статические данные - это локальные данные функционального блока, которые хранятся в экземплярном блоке данных и поэтому остаются незатронутыми до следующей обработки функционального блока.

Двоичный результат (BR)

Двоичный результат (binary result, BR) связывает обработку битов и слов. Это эффективный метод, позволяющий давать двоичную интерпретацию результату операции над словами и включать его в последовательность логических операций.

И

Идентификатор адреса

Идентификатор адреса – это часть адреса, содержащая различные данные, которые могут включать такие элементы, как само значение (объект данных) или размер объекта, с которым команда может, например, выполнить логическую операцию. В операторе «L IB10» IB – это идентификатор адреса («I» указывает на область входов памяти, а «B» означает один байт в этой области).

Иерархия вызовов

Прежде чем блоки могут быть обработаны, они должны быть вызваны. Последовательность и вложенность этих вызовов внутри организационного блока называется иерархией вызовов.

Исходный файл

Исходный файл (текстовый файл) – это часть программы, создаваемая с помощью графического или текстового редактора и компилируемая в исполняемую программу пользователя S7 или в машинный код для M7.

Исходный файл S7 хранится в папке «Sources [Исходные тексты]» под папкой «S7 program [Программа S7]».

Исходный файл на AWL

Исходный файл, представляющий собой программу на языке *Список операторов (STL, AWL)*; то же, что и файл-источник или текстовый файл.

К

Ключевое слово

Ключевые слова применяются при программировании с помощью исходных файлов для обозначения начала и конца блока и выделения разделов в описательной части блоков, начала комментариев к блоку и начала заголовков.

Коды условий CC 1 и CC 0

Биты CC 1 и CC 0 (коды условий) дают информацию о следующих результатах или битах:

- результат арифметической операции
- результат сравнения
- результат дискретной операции
- биты, выдвинутые операцией сдвига или циклического сдвига

Команда

Команда – это часть оператора STEP 7; она указывает, что должен делать процессор.

Контактный план (KOP, LAD)

Контактный план (Ladder Logic, LAD, KOP) – это графический язык программирования в STEP 5 и STEP 7. Его представление стандартизовано в соответствии с DIN 19239 (международный стандарт IEC 1131–1). Представление в виде контактного плана соответствует представлению в виде релейно-контактных схем. В отличие от списка команд (STL, AWL), LAD имеет более ограниченный набор команд.

Косвенная адресация через память

Вид адресации, при котором адрес команды указывает расположение в памяти значения, с которым команда должна работать.

Косвенная адресация через регистр

Вид адресации, при котором адрес (операнд) команды указывает косвенно через адресный регистр и смещение расположение в памяти значения, с которым команда должна работать.

Л

Логический блок

Логические блоки – это блоки в SIMATIC S7, которые содержат часть программы пользователя STEP 7. В отличие от них, блоки данных (DB) содержат только данные. Имеются следующие виды кодовых блоков: организационные блоки (OB), функциональные блоки (FB), функции (FC), системные функциональные блоки (SFB) и системные функции (SFC). Блоки хранятся в папке «Blocks [Блоки]» под папкой «S7 Program [Программа S7]».

М

Массив

Массив - это составной тип данных, состоящий из элементов данных одного типа. Эти элементы данных могут быть элементарными или составными.

Мнемоническое представление

Мнемоническое представление – это сокращенная форма отображения имен адресов и команд программирования в программе (например, «I» означает «input [вход]»). STEP 7 поддерживает международное представление (базирующееся на английском языке) и представление SIMATIC (основанное на немецких сокращениях набора команд и соглашениях об адресации, принятых в SIMATIC).

Н

Непосредственная адресация

При непосредственной адресации операнд содержит величину, с которой должна быть выполнена операция.

Пример: L27 означает загрузку в аккумулятор константы 27.

О

Область памяти

В SIMATIC S7 CPU имеет три области памяти:

- загрузочную память
- рабочую память
- системную память

Оператор

Оператор – это наименьшая независимая часть программы пользователя, создаваемой на текстовом языке. Оператор представляет команду для процессора.

Описание

Раздел описаний используется для описания локальных данных логического блока при программировании в текстовом редакторе.

Описание переменной

Описание переменной включает в себя символическое имя, тип данных и, при желании, начальное значение, адрес и комментарий.

П

Папка

Каталог пользовательского интерфейса Администратора SIMATIC (SIMATIC Manager), который может быть открыт и может содержать другие каталоги и объекты.

Первичный опрос

Первый опрос результата логической операции.

Программа пользователя

Программа пользователя содержит все операторы и описания и все данные для обработки сигналов, которые могут быть использованы для управления устройством или процессом. Это часть программируемого модуля (CPU, FM), которая может быть структурирована разбиением на более мелкие единицы (блоки).

Проект

Проект – это папка для всех объектов задачи автоматизации независимо от количества станций, модулей и их соединения в сети.

Прямая адресация

При прямой адресации адрес содержит расположение в памяти величины, которая должна использоваться командой.

Пример:

Ячейка памяти Q4.0 определяет бит 0 в байте 4 таблицы выходов образа процесса.

Р

Результат логической операции (RLO, VKE)

Результат логической операции (RLO, VKE) – это результат цепи логических операций, который используется для обработки других двоичных сигналов. Исполнение определенных команд целиком зависит от предшествующего RLO.

С

Сегмент

Сегменты подразделяют блоки LAD (KOP) и FBD (FUP) на законченные цепи тока, а блоки списка операторов (STL, AWL) на удобочитаемые единицы.

Символ

Символ - это имя, которое может быть определено пользователем при соблюдении определенных синтаксических правил. После определения (например, в качестве переменной, типа данных, метки перехода, блока) оно может применяться для программирования и для функций взаимодействия с оператором.
Пример: адрес: I 5.0, тип данных: BOOL, символ: Кнопка аварийного отключения.

Символическая адресация

При символической адресации адрес, подлежащий обработке, обозначается символом (в противоположность абсолютному адресу).

Системная функция (SFC)

Системная функция – это функция (без памяти), которая встроена в операционную систему CPU S7 и, если необходимо, может быть вызвана из программы пользователя STEP 7 подобно функции (FC).

Системный функциональный блок (SFB)

Системный функциональный блок (SFB) – это функция (с памятью), которая встроена в операционную систему S7 и, если необходимо, может быть вызвана из программы пользователя STEP 7 подобно функциональному блоку (FB).

Скобочный стек

Скобочный стек – это байт памяти, используемый скобочными командами A(, O(, X(, AN(, ON(, XN(. В стек можно поместить всего восемь битовых логических команд.

Слово состояния

Слово состояния – это часть регистра CPU. Оно содержит информацию о состоянии и ошибках, которая отображается при выполнении определенных команд STEP 7. Биты состояния могут считываться и записываться пользователем, биты ошибок можно только считывать.

Список команд (STL, AWL)

Список команд (STL, AWL) – это текстовое представление языка программирования STEP 7, подобное машинному коду. STL (AWL) – это язык ассемблера STEP 5 и STEP 7. Если вы программируете на STL (AWL), то отдельные операторы представляют фактические шаги, которыми CPU исполняет программу.

Справочные данные

Справочные данные служат для контроля программы CPU и включают в себя список перекрестных ссылок, списки занятости, структуру программы, список не использованных адресов и список адресов без символов.

Станция

Станция – это устройство, которое может быть подключено к одной или нескольким подсетям; например, программируемый контроллер, устройство программирования или станция оператора.

Структура программы пользователя

Структура программы пользователя описывает иерархию вызовов блоков внутри программы S7 и обеспечивает обзор применяемых блоков и их зависимостей.

T

Таблица описания переменных

В таблице описания переменных описываются локальные данные логического блока, если программирование производится в редакторе пошагового ввода.

Таблица переменных (VAT)

В таблице переменных собраны переменные, которые вы хотите наблюдать и изменять, устанавливая соответствующие форматы.

Таблица символов

Таблица для сопоставления символов адресам глобальных данных и блоков. Примеры: Аварийное отключение (символ) –I 1.7 (адрес) или Регулятор (символ) – SFB24 (блок).

Тип данных

Тип данных определяет, как значение переменной или константы должно применяться в программе пользователя.

В SIMATIC STEP 7 пользователю предоставляются в распоряжение два вида типов данных (IEC 1131-3):

- элементарные типы данных
- составные типы данных

Тип данных, составной

Составные типы данных создаются пользователем с помощью описания типа. Они не имеют собственного имени и поэтому не могут применяться повторно. Они могут быть массивами или структурами. Сюда же относятся типы данных STRING и DATE_AND_TIME.

Тип данных, элементарный

В соответствии с IEC 1131–3 элементарные типы данных - это предопределенные типы данных.

Примеры:

- тип данных «BOOL» определяет двоичную переменную («бит»)
- тип данных «INT» определяет 16-битовую переменную с фиксированной точкой.

Типы данных пользователя (UDT)

Типы данных пользователя – это специальные структуры данных, которые вы можете создавать сами и, после их определения, использовать их во всей программе. Они могут использоваться подобно элементарным или составным типам данных в описании переменных логических блоков (FC, FB, OB) или в качестве шаблонов для создания блоков данных с такой же структурой данных.

У

Указатель

С помощью указателя вы можете распознать адрес переменной. Указатель содержит идентификатор вместо значения. Если вы параметрическому типу Pointer [Указатель] ставите в соответствие фактический параметр, то вы передаете адрес в памяти. С помощью STEP 7 вы можете ввести указатель в формате указателя или просто как идентификатор (например, M50.0). В следующем примере показан формат указателя, с помощью которого обращаются к данным, начиная с M 50.0:

```
P#M50.0
```

Ф

Фактический параметр

Фактические параметры заменяют формальные параметры при вызове функциональных блоков (FB) или функций (FC).

Пример: Формальный параметр «Start» заменяется фактическим параметром «I3.6».

Формальный параметр

Формальный параметр - это метка-заполнитель для фактического параметра в логических блоках. В функциональных блоках (FB) и функциях (FC) формальные параметры описываются пользователем, а в системных функциональных блоках (SFB) и системных функциях (SFC) они уже имеются. При вызове блока формальным параметрам ставятся в соответствие фактические параметры, так что вызываемый блок работает с этими фактическими значениями.
Формальные параметры относятся к локальным данным блока и делятся на входные, выходные и проходные (in/out) параметры.

Функциональный блок (FB)

В соответствии со стандартом Международной электротехнической комиссии IEC 1131-3, функциональные блоки – это логические блоки с "памятью" (т.е. у них есть статические данные). Функциональный блок позволяет передавать параметры в программе пользователя, т.е. он пригоден для программирования часто повторяющихся сложных функций, например, регулирования по замкнутому контуру или выбора режима работы. Так как у функционального блока есть память (экземплярный блок данных), то вы можете обратиться к его параметрам (например, к выходам) в любое время и в любой точке программы пользователя.

Функциональный план (FBD, FUP)

Функциональный план (Function Block Diagram, FBD, FUP) – это один из языков программирования в STEP 5 и STEP 7. FBD представляет логику в виде блоков, известных из булевой алгебры. Кроме того, сложные функции (например, математические) могут быть представлены в непосредственном соединении с логическими блоками. Программы, созданные с помощью FBD, могут быть также преобразованы в представления на других языках программирования (например, в виде контактного плана).

Функция (FC)

В соответствии со стандартом Международной электротехнической комиссии IEC 1131-3, функции – это логические блоки без "памяти" (т.е. у них нет статических данных). Функции позволяют передавать параметры в программе пользователя, т.е. они пригодны для программирования часто повторяющихся сложных функций, например, расчетов. Важно: Так как у функции нет памяти, то вы должны продолжить обработку вычисленных значений непосредственно после вызова функции.

Ц

Цепь логических операций

Цепь логических операций – это часть программы пользователя, которая начинается битом FC, имеющим сигнальное состояние 0, и которая заканчивается, когда команда или событие сбрасывает бит FC в 0. Когда CPU выполняет первую команду в цепи логических операций, бит FC устанавливается в 1. Некоторые команды, например, команды вывода (установка, сброс, присваивание) сбрасывают бит FC в 0. *См.* Первичный опрос.

Цепь тока

Характерная особенность языка программирования *Контактный план* (KOP, Ladder Logic). Цепи тока содержат контакты и катушки. В цепь могут быть вставлены также сложные элементы (напр., математические функции) в форме «блоков». Цепи тока подключены к токовым шинам.

Э

Экземпляр

«Экземпляром» называется вызов функционального блока. Каждому вызову ставится в соответствие экземплярный блок данных.

Экземплярный блок данных (DB)

Экземплярный блок данных хранит формальные параметры и статические данные функциональных блоков. Экземплярный блок данных может быть поставлен в соответствие одному вызову функционального блока или иерархии вызовов функциональных блоков.

C

CPU

CPU (central processing unit – центральный процессор) – это центральный модуль программируемого контроллера, в котором хранится и обрабатывается программа пользователя. Он состоит из операционной системы, процессора и устройств сопряжения с системой связи.

M

Master Control Relay

Master Control Relay (MCR) – главное управляющее реле – применяется в релейных контактных планах для активизации и деактивизации потока сигнала (цепи тока). Деактивированная цепь тока соответствует последовательности команд, которая записывает нулевое значение вместо рассчитанного, или последовательности команд, которая оставляет неизменным существующее значение памяти.

S

SIMATIC Manager

SIMATIC Manager [Администратор SIMATIC]– это графический пользовательский интерфейс для пользователей SIMATIC в среде Windows 95.

S7 Program (Программа S7)

Папка для блоков, исходных файлов и схем для программируемых контроллеров S7. Программа S7 включает в себя также таблицу символов.