SIEMENS

SIMATIC

STL для S7-300 и S7-400 Программирование

Справочное руководство

Это руководство является частью документации с заказным номером:
6ES7810-4CA07-8BW1

Редакция 01/2004
A5E00261408-01

Указания по технике безопасности

Данное руководство содержит указания, которые Вы должны соблюдать для обеспечения собственной безопасности, а также защиты от повреждений продукта и связанного с ним оборудования. Эти замечания выделены предупреждающим треугольником и представлены, в соответствии с уровнем опасности следующим образом:

Опасность
указывает, что если не будут приняты надлежащие меры предосторожности, то это **приведет** к гибели людей, тяжким телесным повреждениям или существенному имущественному ущербу.

Предупреждение
указывает, что при отсутствии надлежащих мер предосторожности это **может привести** к гибели людей, тяжким телесным повреждениям или к существенному имущественному ущербу.

Осторожно
указывает, что возможны легкие телесные повреждения и нанесение небольшого имущественного ущерба при непринятии надлежащих мер предосторожности.

Осторожно
указывает, что возможно повреждение имущества, если не будут приняты надлежащие меры безопасности.

Примечание
привлекает ваше внимание к особо важной информации о продукте, обращении с ним или к соответствующей части документации.

Квалифицированный персонал

К монтажу и работе на этом оборудовании должен допускаться только **квалифицированный персонал**. Квалифицированный персонал – это люди, которые имеют право вводить в действие, заземлять и маркировать электрические цепи, оборудование и системы в соответствии со стандартами техники безопасности.

Надлежащее использование

Примите во внимание следующее:

Предупреждение
Это устройство и его компоненты могут использоваться только для целей, описанных в каталоге или технической документации, и в соединении только с теми устройствами или компонентами других производителей, которые были одобрены или рекомендованы фирмой Siemens.
Этот продукт может правильно и надежно функционировать только в том случае, если он правильно транспортируется, хранится, устанавливается и монтируется, а также эксплуатируется и обслуживается в соответствии с рекомендациями.

Товарные знаки

SIMATIC®, SIMATIC HMI® и SIMATIC NET® - это зарегистрированные товарные знаки SIEMENS AG.

Некоторые другие обозначения, использованные в этих документах, также являются зарегистрированными товарными знаками; права собственности могут быть нарушены, если они используются третьей стороной для своих собственных целей.

Siemens AG
Департамент автоматизации и приводов
Промышленные системы автоматизации
П\я 4848, D- 90327, Нюрнберг

Отказ от ответственности

Мы проверили содержание этого руководства на соответствие с описанным аппаратным и программным обеспечением. Так как отклонения не могут быть полностью исключены, то мы не можем гарантировать полного соответствия. Однако данные, приведенные в этом руководстве, регулярно пересматриваются, и все необходимые исправления вносятся в последующие издания. Мы будем благодарны за предложения по улучшению содержания.

Siemens Aktiengesellschaft
A5E00261408-01

Excellence in Automation & Drives: Siemens

Предисловие

Цель руководства

Это руководство поможет Вам при разработке пользовательских программ на языке программирования *Список операторов* - STL (от англ. Statement List).

Кроме того, в этом руководстве имеется справочный раздел по элементам языка программирования STL, в котором описываются синтаксис и принцип работы отдельных инструкций языка.

Требования к начальной подготовке

Это руководство рассчитано на программистов - разработчиков S7-программ, пусконаладчиков и обслуживающий персонал.

Для понимания излагаемого материала желательно наличие общих знаний в области техники автоматизации.

Дополнительно желательно иметь опыт работы с компьютером и знание другого подобного PC оборудования (например программаторов) с операционными системами MS Windows 2000 Professional или MS Windows XP Professional.

Область примениенения руководства

Это руководство разработано для программного пакета STEP 7 версии 5.3.

Соответствие стандартам

STL соответствует языку ""Instruction List" (“Список инструкций”), определенному в стандарте Международной электротехнической комиссии IEC 1131–3, хотя имеются некоторые существенные отличия по отдельным операторам. По поводу детальных подробностей обратитесь к таблице стандартов, находящейся в файле NORM_TBL.WRI пакета STEP 7.

Требования

Для эффективного использования этого руководства, Вы должны быть знакомы с теорией программирования на S7 , описанной во встроенной помощи STEP 7. Языковые пакеты также используют программное обеспечение STEP 7, которое Вы должны изучить по соответствующей документации.

Это руководство является частью пакета документации "STEP 7 Reference". В следующей таблице приведен обзор STEP 7 документации:

Документация	**Содержание**	**Заказной номер**
Базовая информация по STEP 7: • Работа со STEP 7 V5.3, Первые шаги • Программирование в STEP 7 V5.3 • Конфигурация аппаратной части и коммуникаций в STEP 7 V5.3 • От S5 к S7, Руководство по конвертации	Базовая информация для технического персонала , описывающая методы выполнения задач управления со STEP 7 на программируемых контроллерах S7-300/400.	6ES7810-4CA07-8BW0
STEP 7 Справочники: • Руководства по Контактному плану (KOP)/Функциональному плану (FBD)/Списку инструкций (STL) для S7-300/400 • Стандартные и системные функции S7-300/400	Справочная информация по описанию языков программирования LAD, FBD и STL, а также стандартных и системных функций STEP 7.	6ES7810-4CA07-8BW1

Встроенная справка	**Содержание**	**Заказной номер**
Help on STEP 7	Базовая информация по программированию и конфигурированию аппаратной части STEP 7 в форме встроенной справки.	Часть стандартного программного обеспечения STEP 7.
Справочная информация по STL/KOP/FBD Справочная информация по SFBs/SFCs Справочная информация по организационным блокам.	Контекстная справочная информация.	Часть стандартного программного обеспечения STEP 7.

Online Помощь

Руководство соответствует встроенной в стандартное программное обеспечение помощи. Эта встроенная помощь предоставляет Вам детальную информацию по использованию программного обеспечения.

Встроенная система помощи доступна пользователю следующими способами:

- Контекстная помощь предоставляет помощь в контексте с текущим объектом, например, по открытому диалоговому окну . Вы можете открыть контекстную помощь через команду меню **Help > Context-Sensitive Help**, нажатием клавиши F1 или с помощью знака вопроса из панели инструментов.
- Вы можете вызвать помощь по STEP 7 с использованием команды меню **Help > Contents** или кнопки "Help on STEP 7" окна контекстно-зависимой помощи.
- Вы можете открыть словарь терминов приложений STEP 7 с помощью кнопки "Glossary".

Это руководство является выборкой из "Help on Statement List ". Руководство, как и встроенная помощь имеют одинаковую структуру, поэтому, можно легко перейти от руководства к встроенной помощи.

Дальнейшая поддержка

Если у Вас возникают технические вопросы, пожалуйста, свяжитесь с Вашим представительством Siemens .

Вы можете найти контактное лицо через:

http://www.siemens.com/automation/partner

Учебные центры

Siemens предлагает широкий спектр учебных курсов по изучению систем автоматизации SIMATIC S7. Пожалуйста свяжитесь с вашим региональным учебным центром или нашим центральным учебным центром в D 90327 Нюрнберге, Германия: Москва,Россия

Телефон: +49 (911) 895-3200. (095) 737-23-88

Internet: http://www.sitrain.com http://www.aud.ru

A&D Техническая поддержка

Всемирная, круглосуточная:

Всемирная (Nuernberg) **техническая поддержка** 24 часа в день, 365 дней в году Телефон: +49 (180) 5050-222 Факс: +49 (180) 5050-223 E-Mail: adsupport@ siemens.com GMT: +1:00		
Европа/Африка (Nuernberg) **техническая поддержка** Местное время:Пн.-Пт. 8:00 - 17:00 Телефон: +49 (180) 5050-222 Fax: +49 (180) 5050-223 E-Mail: adsupport@ siemens.com GMT: +1:00	**United States** (Johnson City) **техническая поддержка** Местное время:Пн.-Пт. 8:00 - 17:00 Телефон: +1 (423) 262 2522 Fax: +1 (423) 262 2289 E-Mail: simatic.hotline@ sea.siemens.com GMT: -5:00	**Asia / Australia** (Beijing) **техническая поддержка** Местное время:Пн.-Пт. 8:00 - 17:00 Телефон: +86 10 64 75 75 75 Fax: +86 10 64 74 74 74 E-Mail: adsupport.asia@ siemens.com GMT: +8:00
Языки , используемые на SIMATIC Hotlines, английский или немецкий.		

Сервис и поддержка в Интернете

В дополнение к нашей документации, мы предлагаем наши Know-how online в Интернете на сайте:

http://www.siemens.com/automation/service&support

где Вы найдете следующее:

- Бюллетень, предоставляющий вам обновляемую информацию по Вашим продуктам.
- Необходимые документы с помощью функции Search в Service & Support.
- Форум, где пользователи и специалисты со всего мира обмениваются опытом.
- Ваши местные представительства департамента автоматизации и приводов.
- Информацию по сервису, ремонту, запчастям и прочему в разделе "Services".

Содержание

1 Битовые логические инструкции

1.1 Обзор битовых логических инструкций

Описание

Битовые логические инструкции работают с двумя числами, 1 и 0. Эти две цифры образуют базис системы счисления, называемой двоичной системой. Цифры 1 и 0 называются двоичными цифрами (**bi**nary digi**t**s) или просто битами. При работе со схемами, использующими контакты и катушки, значение 1 означает активное состояние или протекание тока, а 0 – неактивное состояние или отсутствие протекания тока.

Битовые логические инструкции интерпретируют состояния сигналов 1 и 0 и комбинируют их по правилам булевой логики. Эти комбинации дают результат 1 или 0, называемый « результатом логической операции» (RLO).
Для приложений битовой логики используются следующие битовые логические инструкции:

- A И
- AN И-НЕ
- O ИЛИ
- ON ИЛИ-НЕ
- X ИСКЛЮЧАЮЩЕЕ ИЛИ
- XN ИСКЛЮЧАЮЩЕЕ ИЛИ - НЕ
- O И перед ИЛИ

Вы можете использовать следующие инструкции для выполнения вложенных функций :

- A(И с открытием скобки
- AN(И – НЕ с открытием скобки
- O(ИЛИ с открытием скобки
- ON(ИЛИ - НЕ с открытием скобки
- X(ИСКЛЮЧАЮЩЕЕ ИЛИ с открытием скобки
- XN(ИСКЛЮЧАЮЩЕЕ ИЛИ- НЕ с открытием скобки
-) Закрытие скобки

Вы можете закончить битовое логическое выражение с помощью одной из следующих инструкций:

- = Присвоение
- R Сброс
- S Установка

Для изменения результата логической операции Вы можете использовать следующие:

- NOT Инверсия RLO
- SET Установка RLO в 1
- CLR Сброс RLO в 0
- SAVE Сохранение RLO в BR регистре

Другие инструкции реагируют на появление нарастающего или падающего фронта РЛО:

- FN Выделение падающего фронта РЛО
- FP Выделение нарастающего фронта РЛО

1.2 A: Логическое И

Формат

A <Бит>

Адрес	Тип данных	Области памяти
<Бит>	BOOL	I, Q, M, L, D, T, C

Описание

Инструкция **A** проверяет состояние указанного адреса на "1", и производит сопряжение полученного результата опроса с РЛО предыдущей инструкции по схеме логического И .

Инструкция **И** может также использоваться для прямого контроля слова состояния при использовании следующих адресов : ==0, <>0, >0, <0, >=0, <=0, OV, OS, UO, BR.

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	x	x	x	1

Пример

STL Программа		Релейная схема	
		Шина питания	
A	**I 1.0**	Состояние I 1.0 = 1	Н.О. контакт
A	**I 1.1**	Состояние I 1.1 = 1	Н.О. контакт
=	**Q 4.0**	Состояние Q4.0 = 1	Катушка
Изображение замкнутого контакта			

1.3 AN: Логическое И - НЕ

Формат

N <Бит>

Адрес	Тип данных	Область памяти
<Бит>	BOOL	I, Q, M, L, D, T, C

Описание

Инструкция **AN** (И – НЕ) проверяет состояние указанного адреса на "0", и производит сопряжение полученного результата опроса с РЛО предыдущей инструкции по схеме логического И .

Инструкция **И - НЕ** может также использоваться для прямого контроля слова состояния при использовании следующих адресов : ==0, <>0, >0, <0, >=0, <=0, OV, OS, UO, BR.

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	x	x	x	1

Пример

STL Программа		Релейная схема
		Шина питания
A	**I 1.0**	Состояние I 1.0 =0 — Н.О. контакт
AN	**I 1.1**	Состояниe I 1.1=1 — Н.З. контакт
=	**Q 4.0**	СостояниеQ 4.0=0 — Катушка

1.4 O: Логическое ИЛИ

Формат

O <Бит>

Адрес	Тип данных	Область памяти
<Бит>	BOOL	I, Q, M, L, D, T, C

Описание

Инструкция **O** (ИЛИ) проверяет состояние указанного адреса на "1", и производит сопряжение полученного результата опроса с РЛО предыдущей инструкции по схеме логического ИЛИ .

Инструкция **ИЛИ** может также использоваться для прямого контроля слова состояния при использовании следующих адресов : ==0, <>0, >0, <0, >=0, <=0, OV, OS, UO, BR.

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	x	x	1

Пример

STL Программа		Релейная схема
		Шина питания
O	**I 1.0**	Состояние I 1.0=1 Н.О. контакт; Состояние I 1.1=0 Н.О. контакт
O	**I 1.1**	
=	**Q 4.0**	СтатусQ 4.0 =1 — Катушка
		Изображение замкнутого контакта

1.5 ON: Логическое ИЛИ-НЕ

Формат

ON <Бит>

Адрес	Тип данных	Область памяти
<Бит>	BOOL	I, Q, M, L, D, T, C

Описание

Инструкция **ON** (ИЛИ - НЕ) проверяет состояние указанного адреса на "0", и производит сопряжение полученного результата опроса с РЛО предыдущей инструкции по схеме логического ИЛИ .

Инструкция **ИЛИ - НЕ** может также использоваться для прямого контроля слова состояния при использовании следующих адресов : ==0, <>0, >0, <0, >=0, <=0, OV, OS, UO, BR.

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:									

Пример

STL Программа		Релейная схема
		Шина питания
O	**I 1.0**	Состояние I 1.0=0 Н.О. контакт
ON	**I 1.1**	Состояние I 1.1 =1 Н.З. контакт
=	**Q 4.0**	Состояние Q 4.0=1 Катушка

1.6 X: Исключающее ИЛИ

Формат

X <Бит>

Адрес	Тип данных	Область памяти
<Бит>	BOOL	I, Q, M, L, D, T, C

Описание

Инструкция **X** проверяет состояние указанного адреса на "1", и производит сопряжение полученного результата опроса с РЛО предыдущей инструкции по схеме логическое Исключающее ИЛИ . Вы также можете использовать эту инструкцию несколько раз. При этом окончательный результат логической операции даст “1” при нечетном количестве результатов опроса = “1”.

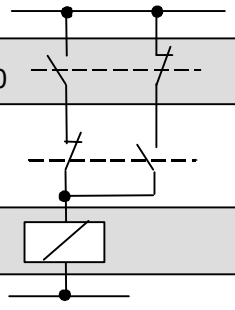
Инструкция Исключающее ИЛИ может также использоваться для прямого контроля слова состояния при использовании следующих адресов : ==0, <>0, >0, <0, >=0, <=0, OV, OS, UO, BR.

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	x	x	1

Пример

STL Программа		Релейная схема
		Шина питания
X	**I 1.0**	Контакт I 1.0
X	**I 1.1**	Контакт I 1.1
=	**Q 4.0**	Q 4.0 Катушка



1.7 XN: Исключающее ИЛИ - НЕ

Формат

XN <Бит>

Адрес	Тип данных	Область памяти
<Бит>	BOOL	I, Q, M, L, D, T, C

Описание

Инструкция **XN** проверяет состояние указанного адреса на "0", и производит сопряжение полученного результата опроса с РЛО предыдущей инструкции по схеме логическое Исключающее ИЛИ .

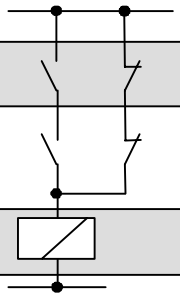
Инструкция Исключающее ИЛИ-НЕ может также использоваться для прямого контроля слова состояния при использовании следующих адресов : ==0, <>0, >0, <0, >=0, <=0, OV, OS, UO, BR.

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	x	x	1

Пример

STL программа		Релейная логика
		Шина питания
X	**I 1.0**	Контакт I 1.0
XN	**I 1.1**	Контакт I 1.1
=	**Q 4.0**	Q 4.0 Катушка



1.8 O: И перед ИЛИ

Формат

O

Описание

Инструкция **O** выполняет функцию логического ИЛИ в сочетании с функциями логического И согласно следующему правилу: И перед ИЛИ.

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	x	1	-	x

Пример

STL Программа		Релейная схема
		Шина питания
A	**I 0.0**	I 0.0, M 10.1
A	**M 10.0**	I 0.2
O		
A	**I 0.2**	M 10.0
A	**M 0.3**	M 0.3
O	**M 10.1**	
=	**Q 4.0**	Q 4.0 Катушка

1.9 A(: И с открывающей скобкой

Формат

A(

Описание

A(: Инструкция “И (“ сохраняет биты RLO и OR , а также код функции в стеке скобок. Допускается максимум 7 таких вложений.

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	1	-	0

Пример

STL Программа		Релейная схема
		Шина питания
A(		
O	**I 0.0**	I 0.0
O	**M 10.0**	M 10.0
)		
A(		
O	**I 0.2**	I 0.2
O	**M 10.3**	M10.3
)		
A	**M 10.1**	M 10.1
=	**Q 4.0**	Q 4.0 Катушка

1.10 AN(: И-НЕ с открывающей скобкой

Формат

AN(

Описание

AN (: Инструкция “**И-НЕ (** “ сохраняет биты RLO и OR , а также код функции в стеке скобок. Допускается максимум 7 таких вложений.

Слово состояния

	BR	**CC 1**	**CC 0**	**OV**	**OS**	**OR**	**STA**	**RLO**	**/FC**
Записывает:	-	-	-	-	-	0	1	-	0

1.11 O(: ИЛИ с открывающей скобкой

Формат

O(

Описание

O(: Инструкция “ИЛИ (“ сохраняет биты RLO и OR , а также код функции в стеке скобок. Допускается максимум 7 таких вложений.

Слово состояния

	BR	**CC 1**	**CC 0**	**OV**	**OS**	**OR**	**STA**	**RLO**	**/FC**
Записывает:	-	-	-	-	-	0	1	-	0

1.12 ON(: ИЛИ-НЕ с открывающей скобкой

Формат

ON(

Описание

ON (: Инструкция “**ИЛИ-НЕ (** “ сохраняет биты RLO и OR , а также код функции в стеке скобок. Допускается максимум 7 таких вложений.

Слово состояния

	BR	**CC 1**	**CC 0**	**OV**	**OS**	**OR**	**STA**	**RLO**	**/FC**
Записывает:	-	-	-	-	-	0	1	-	0

1.13 X(: Исключающее ИЛИ с открывающей скобкой

Формат

X(

Описание

X(: Инструкция “**Исключающее ИЛИ (** “ сохраняет биты RLO и OR , а также код функции в стеке скобок. Допускается максимум 7 таких вложений.

Слово состояния

	BR	**CC 1**	**CC 0**	**OV**	**OS**	**OR**	**STA**	**RLO**	**/FC**
Записывает:	-	-	-	-	-	0	1	-	0

1.14 XN(: Исключающее ИЛИ-НЕ с открывающей скобкой

Формат

XN(

Описание

XN(: Инструкция **“ Исключающее ИЛИ-НЕ (“** сохраняет биты RLO и OR , а также код функции в стэке скобок. Допускается максимум 7 таких вложений.

Слово состояния

	BR	**CC 1**	**CC 0**	**OV**	**OS**	**OR**	**STA**	**RLO**	**/FC**
Записывает:	-	-	-	-	-	0	1	-	0

1.15): Закрывающая скобка

Формат

)

Описание

Инструкция) (закрытие вложения) считывает последний ввод в стек скобок, восстанавливает бит OR, выполняет сопряжение RLO , ранее сохраненного в стеке скобок с текущим RLO в соответствии с логической операцией, код которой был указан при открытии скобки и был сохранен в стеке скобок и присваивает полученный результат RLO. Бит OR также может быть использован с функциональным кодом "AND" или "AND NOT".

Инструкции, которые открывают вложение:

- U(И с открывающей скобкой
- UN(И - НЕ с открывающей скобкой
- O(ИЛИ с открывающей скобкой
- ON(ИЛИ-НЕ с открывающей скобкой
- X(Исключающее ИЛИ с открывающей скобкой
- XN(Исключающее ИЛИ - НЕ с открывающей скобкой

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	x	1	x	1

Пример

STL Программа		Релейная схема
		Шина питания
A(**O** **I 0.0** **O** **M 10.0** **)**		I 0.0 M 10.0
A(**O** **I 0.2** **O** **M 10.3** **)**		I 0.2 M10.3
A	**M 10.1**	M 10.1
=	**Q 4.0**	Q 4.0 Катушка

1.16 = Присвоение

Формат

= <Бит>

Адрес	Тип данных	Область памяти
<Бит>	BOOL	I, Q, M, L, D

Описание

= <Бит>: Записывает RLO в адресуемый бит при включенном главном управляющем реле(MCR = 1). Если MCR = 0, то в адресуемый бит записывается 0 независимо от значения RLO.

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	x	-	0

Пример

1.17 R: Сброс бита

Формат

R <Бит>

Адрес	Тип данных	Область памяти
<Бит>	BOOL	I, Q, M, L, D

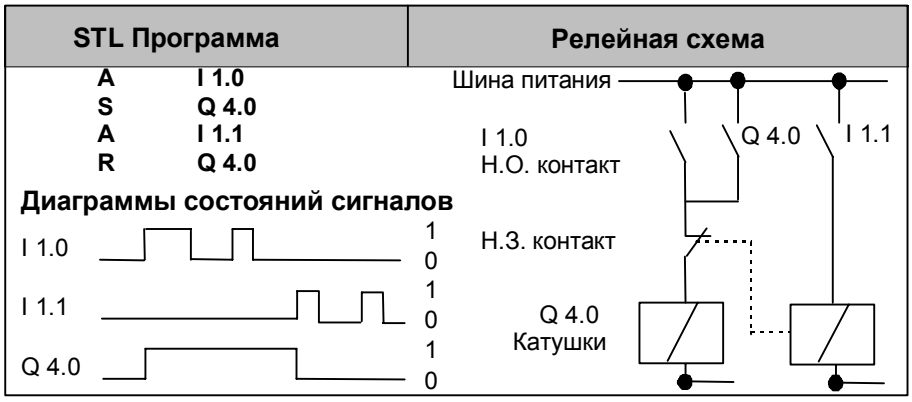
Описание

Инструкция **R** (сброс бита) записывает "0" в адресуемый бит если RLO = 1 и главное управляющее реле MCR = 1. Если MCR = 0, тогда адресуемый в инструкции бит остается баз изменения.

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	x	-	0

Пример



1.18 S: Установка бита

Формат

S <Бит>

Адрес	Тип данных	Область памяти
<Бит>	BOOL	I, Q, M, L, D

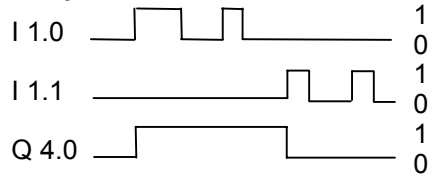
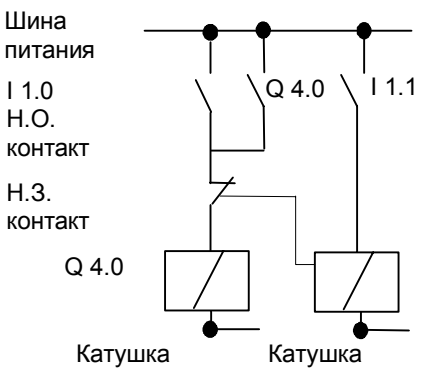
Описание инструкции

Инструкция **S** (установка бита) записывает "1" в адресуемый бит при RLO = 1, включенном главном управляющем реле MCR = 1. Если MCR = 0, адресуемый в инструкции бит не меняет своего состояния.

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	x	-	0

Пример

STL Программа	Релейная схема
A I 1.0 **S Q 4.0** **A I 1.1** **R Q4.0** **Диаграммы состояний сигнала** 	

1.19 NOT: Инверсия RLO

Формат

NOT

Описание

NOT инвертирует значение RLO.

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	1	x	-

1.20 SET: Установка RLO в 1

Формат

SET

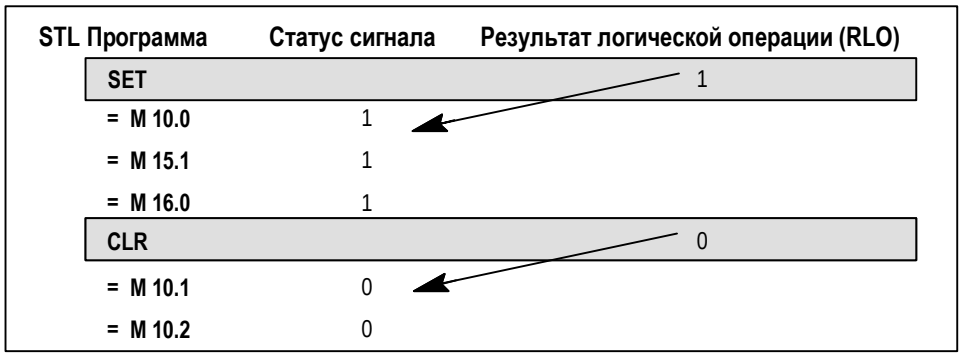
Описание

SET устанавливает текущий RLO в состояние "1".

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	1	1	0

Пример



1.21 CLR: Сброс RLO (=0)

Формат

CLR

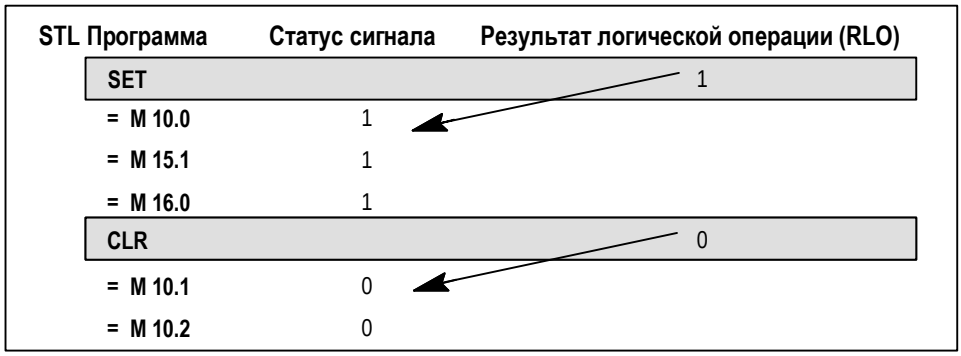
Описание

CLR сбрасывает текущий RLO в "0".

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	0	0	0

Пример



1.22 SAVE: Сохранение бита RLO в регистре BR

Формат

SAVE

Описание инструкции

Инструкция **SAVE** сохраняет бит RLO в бите BR. При этом бит первичного опроса(/FC) не сбрасывается. Поэтому, статус бита BR может участвовать в логическом сопряжении по схеме И в следующем сегменте.

Использование инструкции **SAVE** и последующий опрос бита BR в том же самом блоке или в вызываемом блоке не рекомендуется использовать , так как бит BR может быть изменен некоторыми математическими инструкциями. Необходимо учитывать, что использование инструкции **SAVE** перед выходом из блока, может привести к изменению выхода ENO (= BR) на величину бита RLO , с помощью этого Вы можете организовать обработку возникающих ошибок в блоке.

Слово состояния

	BR	**CC 1**	**CC 0**	**OV**	**OS**	**OR**	**STA**	**RLO**	**/FC**
Записывает:	x	-	-	-	-	-	-	-	-

1.23 FN: Выделение отрицательного фронта RLO

Формат

FN <Бит>

Адрес	Тип данных	Область памяти	Описание
<Бит>	BOOL	I, Q, M, L, D	Бит фронта, сохраняет предыдущее состояние RLO.

Описание

FN <Бит> (Выделение отрицательного фронта RLO) определяет падающий фронт при смене состояния RLO с "1" на "0", что отображается с помощью RLO = 1.

В каждом цикле, состояние бита RLO сравнивается с его значением в предыдущем цикле для определения появления отрицательного фронта RLO. Состояние RLO в предыдущем цикле для этого сохраняется в бите памяти (**<Бит>**) для дальнейшего сравнения. При появлении отличия между текущим и предыдущим состоянием RLO,на выходе выдается состояние "1" (выделение отрицательного фронта), т.е. бит RLO после выполнения этой инструкции принимает значение "1".

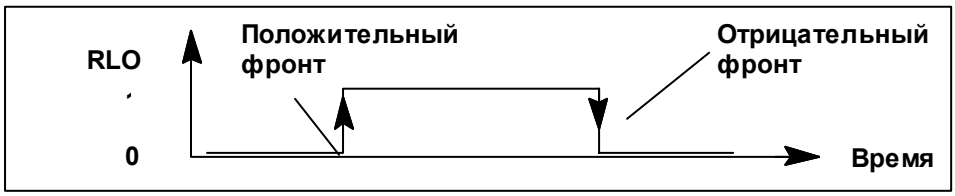
Примечание

Нельзя использовать в качестве бита памяти локальные данные, так как они актуальны только на момент выполнения текущего блока .

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	x	x	1

Определение



Пример

При определении отрицательного фронта на входе I 1.0, активируется выход Q 4.0 на один цикл обработки OB1.

STL программа		Диаграммы состояний сигналов
A	**I 1.0**	I 1.0
FN	**M 1.0**	M 1.0
=	**Q 4.0**	Q 4.0
Цикл OB1:		1 2 3 4 5 6 7 8 9

1.24 FP: Выделение положительного фронта RLO

Формат

FP <Бит>

Адрес	Тип данных	Область памяти	Описание
<Бит>	BOOL	I, Q, M, L, D	Бит фронта, сохраняет предыдущее состояние RLO.

Описание

FP <Бит> (Выделение положительного фронта RLO) определяет нарастающий фронт при смене состояния RLO с "0" на "1", что отображается с помощью RLO = 1.

В каждом цикле, состояние бита RLO сравнивается с его значением в предыдущем цикле для определения появления положительного фронта RLO. Состояние RLO в предыдущем цикле для этого сохраняется в бите памяти (**<Бит>**) для дальнейшего сравнения. При появлении отличия между текущим(“1”) и предыдущим состоянием RLO(“0”),на выходе выдается состояние "1" (выделение положительного фронта), т.е. бит RLO после выполнения этой инструкции принимает значение "1".

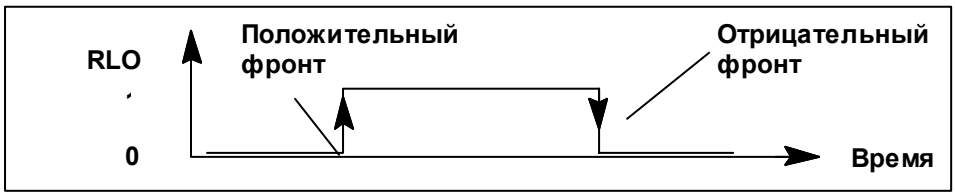
Примечание

Нельзя использовать в качестве бита памяти локальные данные, так как они актуальны только на момент выполнения текущего блока .

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	x	x	1

Определение



Пример

При определении контроллером положительного фронта на входе I 1.0, будет активирован выход Q 4.0 в течение одного цикла выполнения OB1.

STL программа		Диаграммы состояний сигналов
A	**I 1.0**	I 1.0
FP	**M 1.0**	M 1.0
=	**Q 4.0**	Q 4.0
Цикл OB1:		1 2 3 4 5 6 7 8 9

2 Инструкции сравнения

2.1 Обзор инструкций сравнения

Описание

Аккумуляторы 1 (ACCU1) и 2 (ACCU2) сравниваются в соответствии с выбранным Вами типом сравнения :

== ACCU1 равен ACCU2
<> ACCU1 не равен ACCU2
> ACCU1 больше ACCU2
< ACCU1 меньше ACCU2
>= ACCU1 больше или равен ACCU2
<= ACCU1 меньше или равен ACCU2

Если условие сравнения выполняется, то RLO получает значение "1". Биты слова состояния CC1 и CC0 изменяются в соответствии с выполняемыми инструкциями сравнения на “меньше”, ”равно” или ”больше”.

Вы можете использовать следующие типы сравнения:

- ? I : Сравнение чисел типа Integer (16-битовых),
- ? D : Сравнение чисел типа Double Integer (32-битовых),
- ? R : Сравнение чисел с плавающей точкой (32-битовых).

2.2 ? I: Сравнение целых чисел (16-битовых)

Формат

==I, <>I, >I, <I, >=I, <=I

Описание инструкции

Инструкции сравнения целых(16-битовых) чисел сравнивают содержимое младших слов аккумуляторов (ACCU2-L и ACCU 1-L). Их содержимое интерпретируется как 16-битовые целые числа. Если условие сравнения выполняется, то RLO получает значение "1". Если условие сравнения не выполняется, то RLO получает значение "0". Биты слова состояния CC1 и CC0 изменяются в соответствии с выполняемыми инструкциями сравнения на “меньше”, ”равно” или ”больше”.

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	0	-	0	x	x	1

Значение RLO

Инструкция сравнения	Значение RLO при ACCU 2 > ACCU 1	Значение RLO при ACCU 2 = ACCU 1	Значение RLO при ACCU 2 < ACCU 1
==I	0	1	0
<>I	1	0	1
>I	1	0	0
<I	0	0	1
>=I	1	1	0
<=I	0	1	1

Пример

```
STL                 Комментарий
L     MW10          //Загрузка содержимого MW10 (16-битовое целое).
L     IW24          // Загрузка содержимого IW24 (16-битовое целое).
>I                  //Сравнение:  ACCU 2-L (MW10) больше (>) чем ACCU 1- L (IW24).
=     M 2.0         //RLO = 1 если  MW10 > IW24.
```

2.3 ? D: Сравнение двойных целых чисел (32-битовых)

Формат

==D, <>D, >D, <D, >=D, <=D

Описание инструкций

Инструкции сравнения двойных целых (32-битовых) чисел сравнивают содержимое аккумуляторов (ACCU2 и ACCU 1). Их содержимое интерпретируется как 32-битовые целые числа. Выполнение инструкции сравнения приводит к изменению бита RLO , а также отдельных битов слова состояния. Если условие сравнения выполняется, то RLO получает значение "1". Если условие сравнения не выполняется, то RLO получает значение "0". Биты слова состояния CC1 и CC0 изменяются в соответствии с выполняемыми инструкциями сравнения на “меньше”, ”равно” или ”больше”.

Слово состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	0	-	0	x	x	1

Значение RLO

Инструкция сравнения	Значение RLO при ACCU 2 > ACCU 1	Значение RLO при ACCU 2 = ACCU 1	Значение RLO при ACCU 2 < ACCU 1
==D	0	1	0
<>D	1	0	1
>D	1	0	0
<D	0	0	1
>=D	1	1	0
<=D	0	1	1

Пример

```
STL             Комментарий
L     MD10      // Загрузка содержимого MD10 (двойное целое, 32 - битовое).
L     ID24      // Загрузка содержимого ID24 (двойное целое, 32 - битовое).
>D              // Сравнение:  ACCU 2 (MD10) больше (>) чем ACCU 1 (ID24).
=     M 2.0     //RLO = 1 если  MD10 > ID24
```

2.4 ? R: Сравнение чисел с плавающей точкой (32-битовых)

Формат

==R, <>R, >R, <R, >=R, <=R

Описание инструкций

Инструкции сравнения чисел с плавающей точкой (32-битовых) сравнивают содержимое аккумуляторов (ACCU2 и ACCU 1). Их содержимое интерпретируется как 32-битовые числа с плавающей точкой (32-бита , по IEEE-FP). Выполнение инструкции сравнения приводит к изменению бита RLO , а также отдельных битов слова состояния. Если условие сравнения выполняется, то RLO получает значение "1". Если условие сравнения не выполняется, то RLO получает значение "0". Биты слова состояния CC1 и CC0 изменяются в соответствии с выполняемыми инструкциями сравнения на “меньше”, ”равно” или ”больше”.

Слово состояния

	BR	**CC 1**	**CC 0**	**OV**	**OS**	**OR**	**STA**	**RLO**	**/FC**
Записывает:	-	x	x	x	x	0	x	x	1

Значение RLO

Инструкция сравнения	**Значение RLO при ACCU 2 > ACCU 1**	**Значение RLO при ACCU 2 = ACCU 1**	**Значение RLO при ACCU 2 < ACCU 1**
==R	0	1	0
<>R	1	0	1
>R	1	0	0
<R	0	0	1
>=R	1	1	0
<=R	0	1	1

Пример

```
STL                 Комментарий
L     MD10          // Загрузка содержимого MD10 (число с плавающей точкой).
L     1.359E+02     //Загрузка константы 1.359E+02.
>R                  //Сравнение: ACCU 2 (MD10) больше (>) чем ACCU 1 (1.359-E+02).
=     M 2.0         //RLO = 1 если  MD10 > 1.359E+02.
```

3 Инструкции преобразования

3.1 Обзор инструкций преобразования

Описание

Вы можете использовать следующие инструкции для преобразования двоично-десятичного кода в двоичный код и другие типы данных:

- BTI Преобразование числа в BCD- коде в целое число (16-бит)
- ITB Преобразование целого числа (16-бит) в BCD- код
- BTD Преобразование числа в BCD- коде в двойное целое число (32-бита)
- ITD Преобразование целого числа (16-бит) в двойное целое число (32-бита)
- DTB Преобразование двойного целого числа (32-бита) в BCD- код
- DTR Преобразование двойного целого числа (32-бита) в число с плавающей точкой (32-бита по IEEE-FP)

Вы можете использовать следующие инструкции для формирования дополнений целых чисел или изменения знака чисел с плавающей точкой:

- INVI Инверсия числа типа Integer (16-бит)
- INVD Инверсия числа типа Double Integer (32-бита)
- NEGI Инверсия знака числа типа Integer (16-bit)
- NEGD Инверсия знака числа типа Double Integer (32-bit)
- NEGR Инверсия знака числа с плавающей точкой (32-bit, IEEE-FP)

Вы можете использовать следующие инструкции для изменения расположения байт в Аккумуляторе 1 или в его младшем слове:

- CAW Изменение порядка байт в младшем слове ACCU 1-L (16-бит)
- CAD Изменение порядка байт в ACCU 1 (32-бита)

Вы можете использовать следующие инструкции для преобразования 32-битового числа с плавающей точкой в Аккумуляторе 1 в 32-битовое двойное целое число . Эти инструкции отличаются методами округления:

- RND Округление до двойного целого
- TRUNC Выделение целой части
- RND+ Округление до ближайшего большего
- RND- Округление до ближайшего меньшего

3.2 BTI: Преобразование BCD в Integer (16-бит)

Формат

BTI

Описание

Инструкция преобразования BCD в целое число (BTI) преобразует трехразрядное число, записанное в двоично-десятичном коде (BCD-число) в младшем слове аккумулятора 1 в 16–битовое целое число. Результат преобразования сохраняется в младшем слове аккумулятора 1.Старшее слово Аккумулятора 1 и Аккумулятор 2 остаются неизменными.

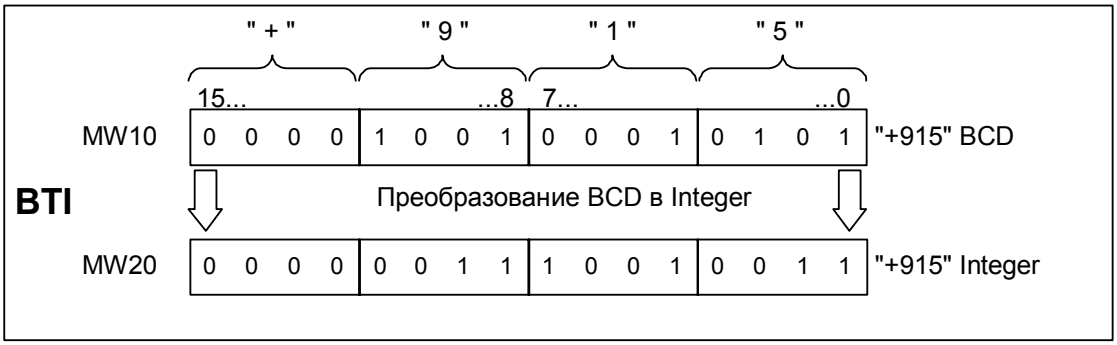
BCD число в ACCU 1-L: BCD-число может находиться в диапазоне от "-999" до "+999". Биты с 0 по 11 интерпретируются как значение числа и бит 15 как знак BCD числа (0 = положительное, 1= отрицательное). Биты с 12 по 14 не используются в преобразовании. Если тетрада (4 бита) BCD числа содержит недопустимое значение - в диапазоне от 10 до 15, при конвертации происходит BCDF ошибка. При этом CPU переходит в режим STOP. Однако, Вы можете этого избежать программированием организационного блока OB121 для обработки синхронной ошибки программирования.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL          Комментарий
L     MW10   //Загрузка BCD числа в аккумулятор ACCU 1-L.
BTI          //Преобразование из BCD в  integer; результат - в ACCU 1-L.
T     MW20   //Сохранение результата в MW20.
```



3.3 ITB: Преобразование Integer (16-бит) в BCD

Формат

ITB

Описание

Инструкция преобразования целого числа в BCD (ITB) интерпретирует ACCU 1-L как 16–битовое целое число и преобразует его в трехразрядное число, записанное в двоично-десятичном коде (BCD-число). Результат преобразования сохраняется в младшем слове аккумулятора 1.
Биты с 0 по 11 содержат значение BCD числа , а биты с 12 по 15 устанавливаются в соответствии со знаком BCD числа :
(0000= положительное, 1111= отрицательное). Старшее слово аккумулятора 1 и аккумулятор 2 остаются неизменными. BCD-число может находиться в диапазоне от "-999" до "+999". Если число находится за пределами этого диапазона, биты OV и OS слова состояния устанавливаются в “1”.

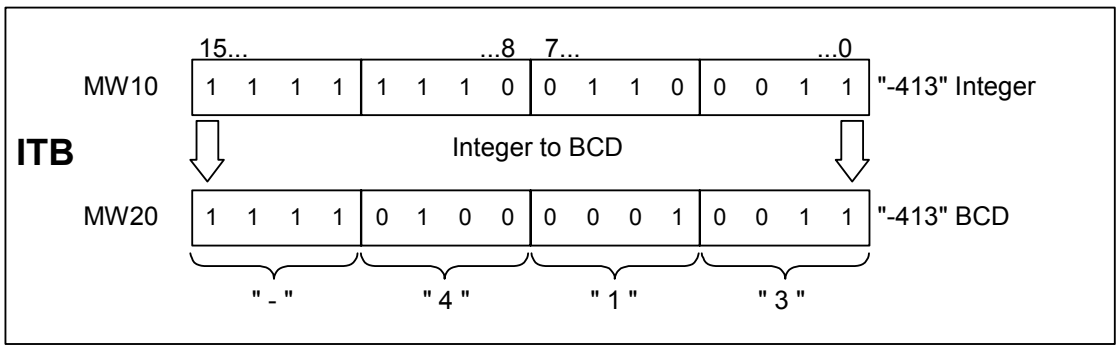
Инструкция выполняется независимо от RLO и не изменяет его.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	x	x	-	-	-	-

Пример

```
STL          Комментарий
L     MW10   //Загрузка целого числа в ACCU 1-L.
ITB          //Преобразование из integer в BCD (16-бит); результат - в ACCU 1-L.
T     MW20   //Передача результата (BCD число) в MW20.
```



3.4 BTD: Преобразование BCD в Double Integer (32-бита)

Формат

BTD

Описание

Инструкция преобразования BCD числа в двойное целое (BTD) интерпретирует содержимое Аккумулятора 1 как 7-значное число, записанное в двоично-десятичном коде (BCD-число)и преобразует его в 32–битовое двойное целое число. Результат преобразования сохраняется в аккумуляторе 1. Аккумулятор 2 остается неизменным.

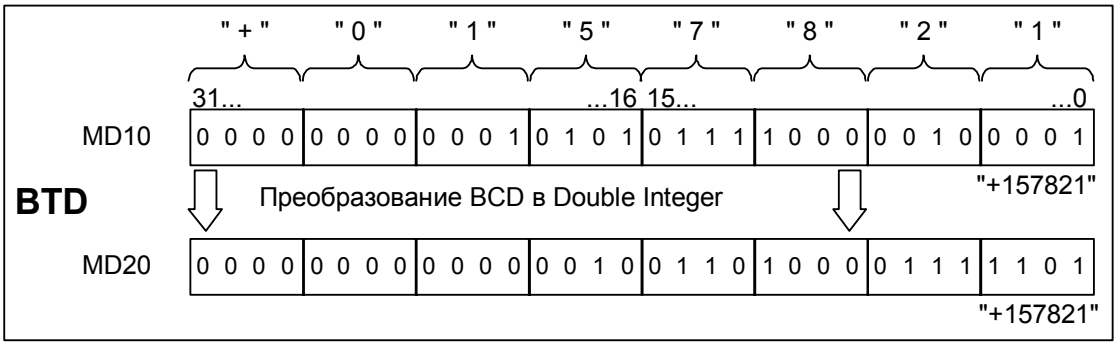
BCD число в ACCU 1: BCD-число может находиться в диапазоне от "-9,999,999" до "+9,999,999". Биты с 0 по 27 интерпретируются как значение числа и бит 31 как знак BCD числа (0 = положительное, 1= отрицательное). Биты с 28 по 30 не используются в преобразовании. Если одна из тетрад (4 бита) BCD числа содержит недопустимое значение - в диапазоне от 10 до 15, при конвертации происходит BCDF ошибка. При этом CPU переходит в режим STOP. Однако, Вы можете этого избежать программированием организационного блока OB121 для обработки синхронной ошибки программирования.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL          Комментарий
L     MD10   //Загрузка BCD числа в ACCU 1.
BTD          //Преобразование из BCD в Integer; результат в  ACCU 1.
T     MD20   //передача результата в MD20.
```



3.5 ITD Integer (16 Bit) to Double Integer (32-bit)

Формат

ITD

Описание

Инструкция преобразования целого числа в двойное целое (ITD) интерпретирует содержимое младшего слова аккумулятора 1 (ACCU 1-L) как 16–битовое целое число и преобразует его в 32-битовое двойное целое число. Результат преобразования сохраняется в аккумуляторе 1. Аккумулятор 2 остается неизменным.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL          Комментарий
L     MW12   //Загрузка числа integer в ACCU 1.
ITD          //Преобразование integer (16-бит) в double integer (32-бит); результат - в
             ACCU 1.
T     MD20   //Передача результата (double integer) в MD20.
```

Пример: MW12 = "-10" (Integer, 16-бит)

Содержание	ACCU1-H				ACCU1-L			
№ бита	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
Перед инструкцией **ITD**	XXXX	XXXX	XXXX	XXXX	1111	1111	1111	0110
После инструкции **ITD**	1111	1111	1111	1111	1111	1111	1111	0110
	(X = 0 или 1, биты не используются в преобразовании)							

3.6 DTB: Преобразование Double Integer (32-бита) в BCD

Формат

DTB

Описание

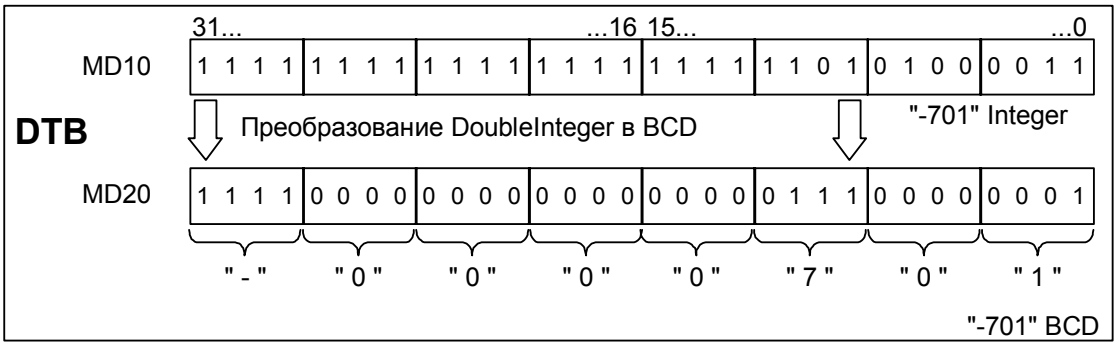
Инструкция преобразования двойного целого числа в BCD (DTB) интерпретирует содержимое аккумулятора 1 как 32–битовое двойное целое число и преобразует его в семизначное число в двоично-десятичном коде (BCD-число). Результат преобразования сохраняется в аккумуляторе 1. Биты с 0 по 27 содержат значение BCD числа , а биты с 28 по 31 устанавливаются в соответствии со знаком BCD числа :
(0000= положительное, 1111= отрицательное). Аккумулятор 2 остается неизменным.
BCD-число может находиться в диапазоне от "-9,999,999" до "+9,999,999" . Если число находится за пределами этого диапазона, биты OV и OS слова состояния устанавливаются в “1”.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	x	x	-	-	-	-

Пример

```
STL          Комментарий
L     MD10   //Загрузка 32-битового Double Integer в ACCU 1.
DTB          //Преобразование из  Double Integer (32-бита) в BCD, результат - в ACCU 1.
T     MD20   //Передача результата (BCD число) в MD20.
```



3.7 DTR: Преобразование Double Integer (32-бита) в число с плавающей точкой (32-бита, IEEE-FP)

Формат

DTR

Описание

DTR (преобразование 32-битового целого числа в 32-битовое IEEE число с плавающей точкой) интерпретирует содержимое аккумулятора 1 как 32–битовое двойное целое число и преобразует его в 32-битовое IEEE число с плавающей точкой. При необходимости, инструкция округляет результат. (Т.к. 32-битовое целое имеет больее высокую разрешающую способность, чем число с плавающей точкой). Результат сохраняется в аккумуляторе 1.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL          Комментарий
L     MD10   //Загрузка 32-битового целого в  ACCU 1.
DTR          //Преобразование из Double Integer в число с плавающей точкой (32-бита,
             IEEE FP); результат - в ACCU 1.
T     MD20   //Передача результата в MD20.
```

3.8 INVI: Инверсия числа типа Integer (16-бит)

Формат

INVI

Описание

INVI (инверсия числа типа integer) производит инверсию всех битов 16-битовой величины в ACCU 1-L. При этом каждый 0 заменяется на 1 , а 1, соответственно, на 0. Результат располагается в младшем слове аккумулятора 1.

Биты слова состояния

	BR	**CC 1**	**CC 0**	**OV**	**OS**	**OR**	**STA**	**RLO**	**/FC**
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL          Комментарий
L     IW8    //Загрузка значения в ACCU 1-L.
INVI         //Инверсия каждого из 16-бит аккумулятора.
T     MW10   //Передача результата в MW10.
```

Содержание	**ACCU1-L**			
№ бита	15 . . .	. .	. .	. . . 0
Перед выполнением **INVI**	0110	0011	1010	1110
После выполнения **INVI**	1001	1100	0101	0001

3.9 INVD : Инверсия числа Double Integer (32-бита)

Формат

INVD

Описание

INVD (Инверсия числа double integer) производит инверсию всех 32 бит значения, находящегося в ACCU 1. При этом каждый 0 заменяется на 1 , а 1, соответственно, на 0. Результат располагается в аккумуляторе 1.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL          Комментарий
L     ID8    //Загрузка значения в ACCU 1.
INVD         //Инверсия всех битов аккумулятора (32-бита).
T     MD10   //Передача результата в MD10.
```

Содержание	ACCU1-H				ACCU1-L			
№ бита	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
Перед выполнением **INVD**	0110	1111	1000	1100	0110	0011	1010	1110
После выполнения **INVD**	1001	0000	0111	0011	1001	1100	0101	0001

3.10 NEGI: Инверсия знака числа Integer (16-бит)

Формат

NEGI

Описание

NEGI (дополнение до двух числа integer) инвертирует знак числа типа Integer в ACCU 1-L. При этом каждый 0 заменяется на 1 , а 1, соответственно, на 0, после чего к полученному результату прибавляется "1" . Результат располагается в младшем слове аккумулятора 1. Эта инструкция эквивалентна умножению целого числа на "-1." Биты слова состояния CC 1, CC 0, OS и OV устанавливаются . в зависимости от результата выполнения операции.

Биты слова состояния

	BR	**CC 1**	**CC 0**	**OV**	**OS**	**OR**	**STA**	**RLO**	**/FC**
Записывает:	-	x	x	x	x	-	-	-	-

Биты слова состояния	**CC 1**	**CC 0**	**OV**	**OS**
Результат = 0	0	0	0	-
-32768 <= Результат <= -1	0	1	0	-
32767 >= Результат >= 1	1	0	0	-
Результат = 2768	0	1	1	1

Пример

```
STL          Комментарий
L     IW8    //Загрузка значения в ACCU 1-L.
NEGI         //Формирование дополнения до двух в 16 битах.
T     MW10   //Передача результата в MW10.
```

Содержание	**ACCU1-L**			
№ бита	15 . . .	. .	. .	. . . 0
До выполнения **NEGI**	0101	1101	0011	1000
После выполнения **NEGI**	1010	0010	1100	1000

3.11 NEGD: Инверсия знака числа Double Integer (32-бита)

Формат

NEGD

Описание

NEGD (дополнение до двух числа Double Integer) инвертирует знак числа типа Double Integer в ACCU 1. При этом каждый 0 заменяется на 1 , а 1, соответственно, на 0, после чего к полученному результату прибавляется "1" . Результат располагается в аккумуляторе 1. Эта инструкция эквивалентна умножению на "-1." Инструкция выполняется независимо от RLO и не изменяет его. Биты слова состояния CC 1, CC 0, OS и OV устанавливаются в зависимости от результата выполнения операции.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	x	-	-	-	-

Биты слова состояния	CC 1	CC 0	OV	OS
Результат = 0	0	0	0	-
-2.147.483.648 <= Результат <= -1	0	1	0	-
2.147.483.647 >= Результат >= 1	1	0	0	-
Результат = 2 147 483 648	0	1	1	1

Пример

```
STL          Комментарий
L     ID8    //Загрузка значения в ACCU 1.
NEGD         // Формирование дополнения до двух в 32 битах
T     MD10   //передача результата в MD10.
```

Содержание	ACCU1-H				ACCU1-L			
№ бита	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
До выполнения **NEGD**	0101	1111	0110	0100	0101	1101	0011	1000
После выполнения **NEGD**	1010	0000	1001	1011	1010	0010	1100	1000
	(X = 0 или 1, биты не участвуют в преобразовании)							

3.12 NEGR Инверсия знака числа с плавающей точкой (32-бита, IEEE-FP)

Формат

NEGR

Описание инструкции

NEGR (инверсия знака 32-битового IEEE числа с плавающей точкой) инвертирует знак числа с плавающей точкой в ACCU 1. Инструкция инвертирует состояние бита 31 в ACCU 1 (знак мантиссы). Результат сохраняется в аккумуляторе 1.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL          Комментарий
L     ID8    //Загрузка значения в ACCU 1 (Пример: ID 8 = 1.5E+02).
NEGR         //Инверсия знака числа с плавающей точкой и сохранение результата в ACCU 1
T     MD10   //Передача результата в MD10 (Пример: результат = -1.5E+02).
```

3.13 CAW: Изменение последовательности байтов в ACCU 1-L (16-бит)

Формат

CAW

Описание

Инструкция CAW меняет местами байты в младшем слове аккумулятора 1 (ACCU 1-L). Результат помещается в младшее слово аккумулятора 1. Старшее слово аккумулятора 1 и аккумулятор 2 остаются неизменными.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL          Комментарий
L     MW10   //Загрузка значения MW10 в ACCU 1.
CAW          //Изменение последовательности  ACCU 1-L.
T     MW20   //Передача результата в MW20.
```

Содержание	ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
До выполнения CAW	значение A	значение B	значение C	значение D
После выполнения CAW	значение A	значение B	значение D	значение C

3.14 CAD: Изменение последовательности байтов в ACCU 1 (32-бита)

Формат

CAD

Описание

Инструкция CAD измененяет последовательность байтов в ACCU 1. Результат помещается в аккумулятор 1. Аккумулятор 2 остается неизменным.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL          Комментарий
L     MD10   //Загрузка значения из MD10 в ACCU 1.
CAD          // Изменение последовательности байтов в ACCU 1.
T     MD20   //Передача результата в MD20.
```

Содержание	ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
До выполнения CAD	значение A	значение B	значение C	значение D
После выполнения CAD	значение D	значение C	значение B	значение A

3.15 RND: Округление до ближайшего целого

Формат

RND

Описание

Инструкция RND (Округление) преобразует число с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 в 32–битовое 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. Если число находится вне допустимого диапазона, то биты слова состояния OV и OS устанавливаются в 1. Результат сохраняется в аккумуляторе 1.

Преобразование не выполняется если преобразуемое число не является числом с плавающей точкой или в случае числа с плавающей точкой, которое не может быть преобразовано в двойное целое число (32 бита). В этом случае взводятся биты переполнения.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	x	x	-	-	-	-

Пример

```
STL          Комментарий
L    MD10    //Загрузка числа с плавающей точкой в ACCU 1.
RND          //Преобразование числа с плавающей точкой в двойное целое и округление
             результата.
T    MD20    //Передача результата в MD20.
```

Значение до преобразования		Значение после преобразования
MD10 = "100.5"	=> RND =>	MD20 = "+100"
MD10 = "-100.5"	=> RND =>	MD20 = "-100"

3.16 TRUNC: Выделение целой части числа

Формат

TRUNC

Описание

Инструкция TRUNC (Выделение целой части числа) преобразует число с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 в 32–битовое двойное целое число. Команда выделяет целую часть числа с плавающей точкой, отбрасывая дробную.

Если число находится вне допустимого диапазона, то биты слова состояния OV и OS устанавливаются в1. Результат сохраняется в аккумуляторе 1.

Преобразование не выполняется если преобразуемое число не является числом с плавающей точкой или в случае числа с плавающей точкой, которое не может быть преобразовано в двойное целое число (32 бита). В этом случае взводятся биты переполнения.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	x	x	-	-	-	-

Пример

```
STL          Комментарий
L     MD10   // Загрузка числа с плавающей точкой в ACCU 1.
TRUN         // Преобразование числа с плавающей точкой в двойное целое и округление
C            результата. Результат в ACCU 1.
T     MD20   // Передача результата в MD20.
```

Значение до преобразования		Значение после преобразования
MD10 = "100.5"	=> TRUNC =>	MD20 = "+100"
MD10 = "-100.5"	=> TRUNC =>	MD20 = "-100"

3.17 RND+: Округление до большего целого

Формат

RND+

Описание

Инструкция RND+ (Округлить до ближайшего большего двойного целого числа) преобразует число с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 в 32–битовое двойное целое число. Команда округляет преобразуемое число до наименьшего целого числа, большего или равного преобразуемому числу с плавающей точкой. Если число находится вне допустимого диапазона, то биты слова состояния OV и OS устанавливаются в 1. Результат сохраняется в аккумуляторе 1.

Преобразование не выполняется если преобразуемое число не является числом с плавающей точкой или в случае числа с плавающей точкой, которое не может быть преобразовано в двойное целое число (32 бита). В этом случае взводятся биты переполнения.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	x	x	-	-	-	-

Пример

```
STL          Комментарий
L     MD10   // Загрузка числа с плавающей точкой в ACCU 1.
RND+         // Преобразование числа с плавающей точкой в двойное целое и округление
             результата. Результат в ACCU 1.
T     MD20   // Передача результата в MD20
```

Значение до преобразования		Значение после преобразования
MD10 = "100.5"	=> RND+ =>	MD20 = "+101"
MD10 = "-100.5"	=> RND+ =>	MD20 = "-100"

3.18 RND-: Округление до меньшего целого

Формат

RND-

Описание

Инструкция RND- (Округлить до ближайшего меньшего двойного целого числа) преобразует число с плавающей точкой (32 бита, IEEE) в аккумуляторе 1 в 32–битовое двойное целое число. Команда округляет преобразуемое число до наибольшего целого числа, меньшего или равного преобразуемому числу с плавающей точкой. Если число находится вне допустимого диапазона, то биты слова состояния OV и OS устанавливаются в 1. Результат сохраняется в аккумуляторе 1.

Преобразование не выполняется если преобразуемое число не является числом с плавающей точкой или в случае числа с плавающей точкой, которое не может быть преобразовано в двойное целое число (32 бита). В этом случае взводятся биты переполнения.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	x	x	-	-	-	-

Пример

```
STL          Комментарий
L     MD10   // Загрузка числа с плавающей точкой в ACCU 1.
RND-         // Преобразование числа с плавающей точкой в двойное целое и округление
             результата. Результат в ACCU 1.
T     MD20   // Передача результата в MD20.
```

Значение до преобразования		Значение после преобразования
MD10 = "100.5"	=> RND- =>	MD20 = "+100"
MD10 = "-100.5"	=> RND- =>	MD20 = "-101"

4 Операции со счетчиками

4.1 Обзор операций со счетчиками

Описание

Счетчики являются функциональным элементом языка программирования STEP7 функций счета. Счетчики имеют область, зарезервированную для них в памяти CPU. Эта область памяти резервирует по одному 16-битному слову для каждого адреса счетчика. При программировании в STL Вы можете адресоваться к 256 счетчикам. Чтобы определить сколько счетчиков может использоваться в Вашем CPU, обратитесь к техническому описанию CPU.

Инструкции счета являются единственными функциями, которые имеют доступ к области памяти счетчиков.

Вы можете изменять значение счетчика, используя следующие инструкции:

- FR Деблокировка счетчика
- L Загрузка текущего значения счетчика в ACCU 1
- LC Загрузка текущего значения счетчика в BCD-коде в ACCU 1
- R Сброс счетчика
- S Установка счетчика на заданное значение
- CU Прямой счет
- CD Обратный счет

4.2 FR: Деблокировка счетчика

Формат

FR <counter>

Адрес	Тип данных	Область памяти	Описание
<counter>	COUNTER	C	Счетчик, диапазон номеров зависит от CPU.

Описание

Изменение с "0" на "1", результата логической операции перед командой деблокировки FR <counter> разблокирует счетчик, сбрасывая биты выделения фронта для установки счетчика и счета на увеличение и уменьшение.

Деблокировка 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 (для чего перед соответствующим оператором счета требуется положительный фронт сигнала) , а бит RLO перед соответствующим оператором имеет состояние сигнала 1.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	-	-	0

Пример

```
STL          Комментарий
A    I 2.0   //Проверка состояния сигнала на входе  I 2.0.
FR   C3      //Деблокировка счетчика C3 при переходе  RLO из 0 в 1.
```

4.3 L: Загрузка значения счетчика в ACCU 1

Формат

L <counter>

Адрес	Тип данных	Область памяти	Описание
<counter>	COUNTER	C	Счетчик, диапазон номеров зависит от CPU.

Описание

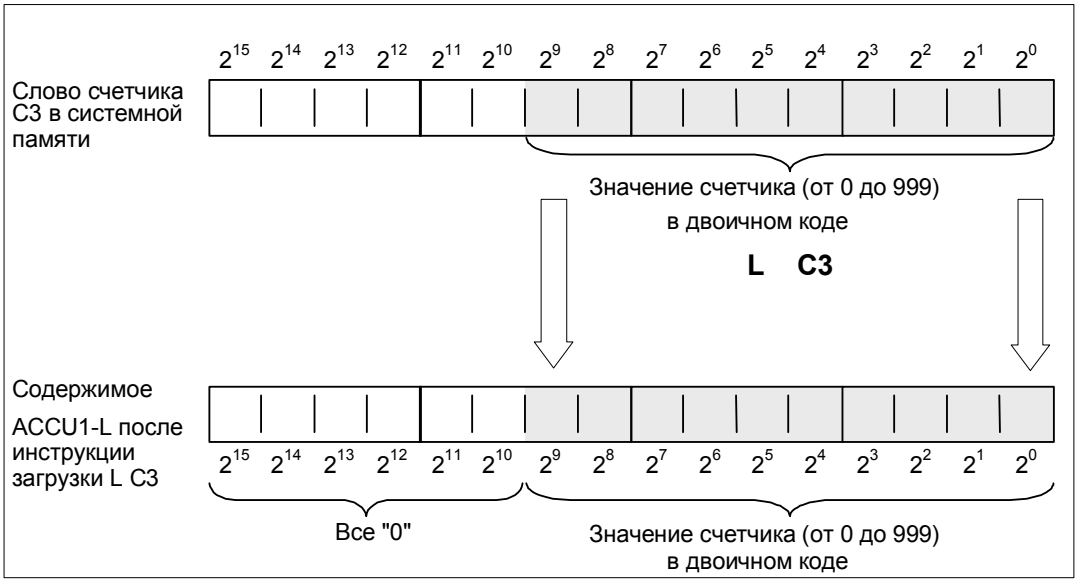
Инструкция L <counter> загружает текущее значение адресуемого счетчика в двоичном коде (Integer) в младшее слово аккумулятора 1(ACCU 1-L) после сохранения содержимого ACCU 1 в аккумуляторе 2 (ACCU 2).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

STL		Комментарий
L	**C3**	**//Загрузка в ACCU 1-L значения C3 в двоичном коде.**



4.4 LC Загрузка значения счетчика в ACCU 1 в BCD коде

Формат

LC <counter>

Адрес	Тип данных	Область памяти	Описание
<counter>	COUNTER	C	Счетчик, диапазон номеров зависит от CPU.

Описание

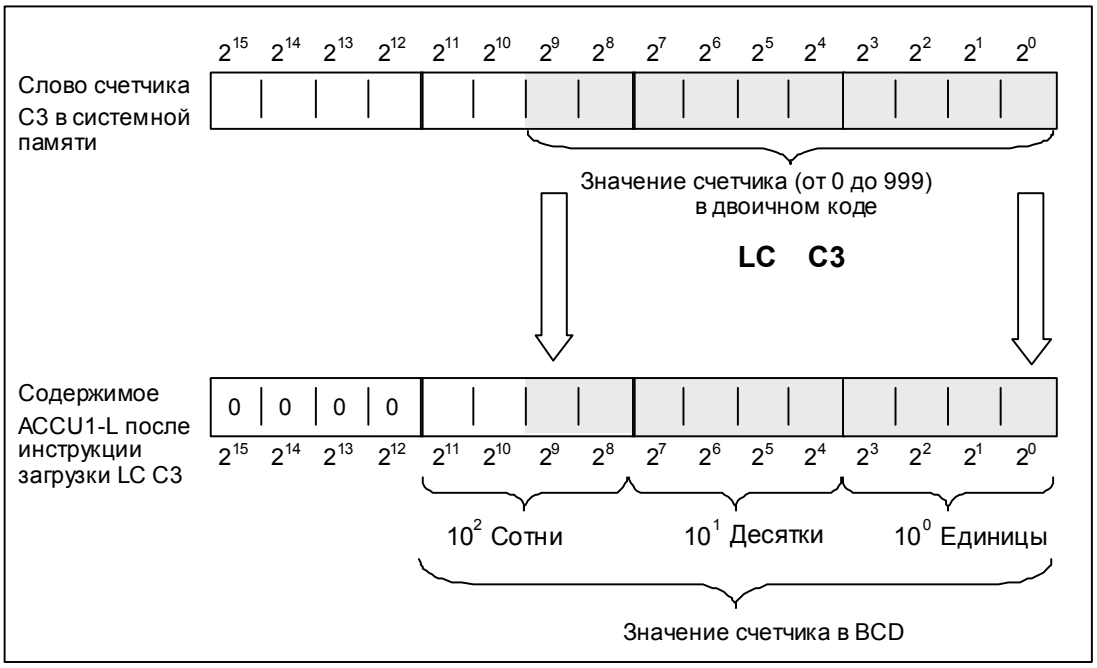
Инструкция LC <counter> загружает текущее значение адресуемого счетчика в BCD коде в младшее слово аккумулятора 1(ACCU 1-L) после сохранения содержимого ACCU 1 в аккумуляторе 2 (ACCU 2).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

STL		Комментарий
LC	**C3**	**// Загрузка в ACCU 1-L значения C3 в BCD коде..**



4.5 R: Сброс счетчика

Формат

R <counter>

Адрес	Тип данных	Область памяти	Описание
<Counter>	COUNTER	C	Счетчик, диапазон номеров зависит от CPU.

Описание

Инструкция R <counter> записывает значение "0" в указанный счетчик (<counter>) при RLO=1.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	-	-	0

Пример

STL		Комментарий
A	**I 2.3**	**//Опрос состояния входа I 2.3.**
R	**C3**	**//Сброс счетчика C3 в значение 0 при значении RLO = 1.**

4.6 S: Установка счетчика на заданное значение

Формат

S <counter>

Адрес	Тип данных	Область памяти	Описание
<Counter>	COUNTER	C	Счетчик, диапазон номеров зависит от CPU.

Описание

Инструкция S <counter> загружает содержимое младшего слова аккумулятора ACCU 1-L в адресуемую область счетчиков при переходе RLO из "0" в "1". Значение в ACCU 1 должно быть в BCD коде и находиться между "0" и "999".

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	-	-	0

Пример

```
STL          Комментарий
A    I 2.3   //Опрос состояния входа I 2.3.
L    C#3     //Загрузка константы 3 в ACCU 1-L.
S    C1      //Установка счетчика C1 на заданное значение при переходе RLO из  0 в 1.
```

4.7 CU: Прямой счет

Формат

CU <counter>

Адрес	Тип данных	Область памяти	Описание
<counter>	COUNTER	C	Счетчик, диапазон номеров зависит от CPU.

Описание

Инструкция прямого счета (CU) увеличивает содержимое указанного счетчика на 1 при изменении результата логической операции с 0 на 1 и значении счетчика меньше "999". Когда счетчик достигает своего верхнего предела ("999"), увеличение прекращается. В дальнейшем фронт RLO на входе прямого счета никакого влияния на значение счетчика не оказывает и бит переполнения (OV) не устанавливается.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	-	-	0

Пример

```
STL          Комментарий
A    I 2.1   // Опрос  входа I 2.1.
CU   C3      //Счетчик C3 увеличивается на 1 при появлении положительного фронта RLO
```

4.8 CD: Обратный счет

Формат

CD <counter>

Адрес	Тип данных	Область памяти	Описание
<counter>	COUNTER	C	Счетчик, диапазон номеров зависит от CPU.

Описание

Инструкция обратного счета (CD) уменьшает содержимое указанного счетчика на 1 при изменении результата логической операции с 0 на 1 и значении счетчика больше "0". Когда счетчик достигает своего нижнего предела, увеличение прекращается. В дальнейшем фронт RLO на входе обратного счета никакого влияния на значение счетчика не оказывает и бит переполнения (OV) не устанавливается.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	-	-	0

Пример

STL		Комментарий
L	**C#14**	**//Загрузка константы.**
A	**I 0.1**	**//При появлении положительного фронта RLO после опроса входа I 0.1**
S	**C1**	**//Счетчик 1 устанавливается на значение из ACCU1-L по фронту RLO.**
A	**I 0.0**	**// При появлении положительного фронта RLO после опроса входа I 0.0.**
CD	**C1**	**//Содержимое счетчика C1 уменьшается на 1 по фронту RLO.**
AN	**C1**	**//Опрос счетчика C1 на нулевое значение.**
=	**Q 0.0**	**//Выход Q 0.0 = 1 если счетчик 1 имеет значение 1.**

5 Инструкции с блоками данных

5.1 Обзор инструкций с блоками данных

Описание

Вы можете использовать инструкцию “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.

Для работы с блоками данных Вы можете использовать инструкции:

- OPN Открыть блок данных
- CDB Обмен регистрами блоков данных DB и DI
- L DBLG Загрузка длины глобального блока данных в ACCU 1
- L DBNO Загрузка номера глобального блока данных в ACCU 1
- L DILG Загрузка длины экземплярного блока данных в ACCU 1
- L DINO Загрузка номера экземплярного блока данных в ACCU 1

5.2 OPN : Открыть блок данных

Формат

OPN <data block>

Адрес	Тип блока данных	Номер блока
<data block>	DB, DI	От 1 до 65535

Описание инструкции

Инструкция **OPN <data block>** открывает блок данных как глобальный блок данных или как экземплярный блок данных. Одновременно могут быть открыты один глобальный блок данных и один экземплярный блок данных.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

STL		Комментарий
OPN	**DB10**	**//Открыть глобальный блок данных DB10.**
L	**DBW35**	**//Загрузить слово данных 35 открытого блока данных в ACCU 1-L.**
T	**MW22**	**//Передача содержимого ACCU 1-L в MW22.**
OPN	**DI20**	**// Открыть экземплярный блок данных DB20.**
L	**DIB12**	**// Загрузить байт данных 12 открытого экземплярного блока данных в ACCU 1-L.**
T	**DBB37**	**// Передача содержимого ACCU 1-L в байт данных 37 открытого глобального блока данных.**

5.3 CDB: Обмен регистрами блоков данных

Формат

CDB

Описание инструкций

Инструкция **CDB** используется для обмена содержимым регистров блоков данных DB и DI. Глобальный блок данных становится открытым как экземплярный и наоборот.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

5.4 L DBLG: Загрузка длины глобального блока данных в ACCU 1

Формат

L DBLG

Описание инструкции

Инструкция **L DBLG** загружает длину глобального блока данных в аккумулятор 1 (ACCU 1) после перемещения содержимого ACCU 1 в ACCU 2.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

STL		Комментарий
OPN	**DB10**	**// Открыть глобальный блок данных DB10.**
L	**DBLG**	**//Загрузить длину глобального блока данных (длина DB10).**
L	**MD10**	**//Загрузка значения для определения достаточности длины блока данных .**
<D		**// Сравнение**
JC	**ERRO**	**//Переход на метку ERRO если длина блока меньше значения MD10.**

5.5 L DBNO: Загрузка номера глобального блока данных в ACCU 1

Формат

L DBNO

Описание инструкции

Инструкция **L DBNO** загружает номер открытого глобального блока данных в ACCU 1-L после перемещения содержимого ACCU 1 в ACCU 2.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

5.6 L DILG: Загрузка длины экземплярного блока данных в ACCU 1

Формат

L DILG

Описание инструкции

Инструкция **L DILG** загружает длину экземплярного блока данных в аккумулятор 1 (ACCU 1) после перемещения содержимого ACCU 1 в ACCU 2.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL          Комментарий
OPN  D120    // Открыть экземплярный блок данных DB20.
L    DILG    // Загрузить длину  экземплярного блока данных (длина DB20).
L    MW10    // Загрузить значение для определения достаточности длины блока данных.
<1           // Сравнение
JC   ERRO    // Переход на метку  ERRO если длина блока меньше значения  MW10.
```

5.7 L DINO: Загрузка номера экземплярного блока данных в ACCU 1

Формат

L DINO

Описание инструкции

Инструкция **L DINO** загружает номер открытого глобального блока данных в ACCU 1-L после перемещения содержимого ACCU 1 в ACCU 2.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

6 Инструкции перехода

6.1 Обзор инструкций перехода

Описание

Вы можете использовать инструкции перехода для управления ходом выполнения программы, позволяя ей прерывать последовательную процедуру выполнения и возобновить обработку с другого места. Вы можете использовать инструкцию циклического выполнения LOOP для обработки участка программы несколько раз подряд.

Операндом команды перехода и циклического выполнения является метка. Метка может состоять из четырех символов, первый из которых должен быть буквой. Метка перехода должна заканчиваться двоеточием "**:**" и ставиться в строке, содержащей инструкцию.

Примечание

Для программ в CPU S7-300 в случае выполнения инструкции перехода, место перехода (кроме CPU 318-2) формирует **начало** двоичной логической цепочки. Место перехода не должно быть в середине логической цепочки.

Вы можете использовать следующие инструкции перехода для безусловного прерывания линейного выполнения программы :

- JU Безусловный переход
- JL Распределенный переход

Вы можете использовать следующие инструкции условного перехода в зависимости от результата логической операции (RLO) предыдущей логической инструкции:

- JC Переход при RLO = 1
- JCN Переход при RLO = 0
- JCB Переход при RLO = 1 с сохранением в BR
- JNB Переход при RLO = 0 с сохранением в BR

Следующие инструкции выполняют условный переход в зависимости от состояния битов слова состояния:

- JBI Переход при BR = 1
- JNBI Переход при BR = 0
- JO Переход при OV = 1
- JOS Переход при OS = 1

Следующие инструкции выполняют условный переход в зависимости от результатов вычислений:

- JZ Переход при нулевом результате
- JN Переход при ненулевом результате
- JP Переход при положительном результате
- JM Переход при отрицательном результате
- JPZ Переход при неотрицательном результате
- JMZ Переход при отрицательном или нулевом результате
- JUO Переход при недействительном результате

6.2 JU: Безусловный переход

Формат

JU <метка>

Адрес	Описание
<метка >	Символьное имя метки перехода.

Описание

Инструкция **JU <метка>** прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с другого места независимо от статуса битов слова состояния. Линейное выполнение программы продолжается с метки перехода, определяющей место перехода. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL                         Комментарий
      A     I 1.0
      A     I 1.2
      JC    DELE    //Переход при  RLO=1 на метку DELE.
      L     MB10
      INC   1
      T     MB10
      JU    FORW    //Безусловный переход  на метку  FORW.
DELE: L     0
      T     MB10
FORW: A     I 2.1   //Выполнение программы продолжается после перехода на метку  FORW.
```

6.3 JL: Распределенный переход

Формат

JL <метка>

Адрес	Описание
<метка >	Символьное имя метки перехода.

Описание

Инструкция **JL <метка>** (переход по списку) представляет собой распределитель переходов. За ней следует ряд безусловных переходов на метки (максимум 255) и заканчивается этот список перед меткой , указанной в самой инструкции **JL<метка>**. Все инструкции списка : **JU** инструкции. Номер выполняемого перехода (от 0 до 255) берется из ACCU 1-L-L.

Инструкция **JL** передает управление одной из **JU** инструкций, если значение в аккумуляторе меньше общего количества безусловных переходов между инструкцией **JL** и меткой этой инструкции. Первая в списке инструкция JU будет выполнена при ACCU 1-L-L=0. Вторая в списке инструкция **JU** будет выполнена при ACCU 1-L-L=1, и так далее. Инструкция **JL** передает управление на метку за последней инструкцией **JU** в списке если число в аккумуляторе, определяющее номер перехода, больше или равно общему числу переходов в списке .

Список инструкций распределенного перехода не должен содержать других инструкций кроме **JU**.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL                     Комментарий
      L     MB0         //Загрузка номера перехода в  ACCU 1-L-L.
      JL    LSTX        //Переход выполняется при ACCU 1-L-L > 3.
      JU    SEG0        // Переход выполняется при ACCU 1-L-L = 0.
      JU    SEG1        // Переход выполняется при ACCU 1-L-L = 1.
      JU    COMM        // Переход выполняется при ACCU 1-L-L = 2.
      JU    SEG3        // Переход выполняется при ACCU 1-L-L = 3.
LSTX: JU    COMM
SEG0: *                 //Исполняемая инструкция
      *
      JU    COMM
SEG1: *                 // Исполняемая инструкция
      *
      JU    COMM
SEG3: *                 // Исполняемая инструкция
      *
      JU    COMM
COMM: *
      *
```

6.4 JC: Переход при RLO = 1

Формат

JC <метка>

Адрес	Описание
<метка >	Символьное имя метки перехода.

Описание

Если результат логической операции равен “1” , то инструкция **JC <метка>** прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Линейное выполнение программы продолжается с места перехода, определенного меткой. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Если результат логической операции равен “0”, переход не будет выполнен. RLO устанавливается в “1” и выполнение программы продолжается со следующей инструкции.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	1	1	0

Пример

STL			Комментарий
	A	I 1.0	
	A	I 1.2	
	JC	JOVR	//Переход при RLO=1 на метку JOVR.
	L	IW8	//Выполнение программы продолжается здесь, если переход не выполнен .
	T	MW22	
JOVR:	A	I 2.1	// Выполнение программы продолжается здесь, при переходе на метку JOVR.

6.5 JCN: Переход при RLO = 0

Формат

JCN <метка>

Адрес	Описание
<метка >	Символьное имя метки.

Описание

Если результат логической операции равен “0” , то инструкция **JCN <метка>** прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Линейное выполнение программы продолжается с места перехода, определенного меткой. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Если результат логической операции равен “1”, переход не будет выполнен. Выполнение программы продолжается со следующей инструкции.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	1	1	0

Пример

```
STL                     Комментарий
      A    I 1.0
      A    I 1.2
      JC   JOVR    //Переход при RLO = 0  на метку JOVR.
      N
      L    IW8     // Выполнение программы продолжается здесь, если переход не выполнен.
      T    MW22
JOVR: A    I 2.1   // Выполнение программы продолжается здесь, при переходе на метку JOVR.
```

6.6 JCB: Переход при RLO = 1 с сохранением его в BR

Формат

JCB <метка>

Адрес	Описание
<метка >	Символьное имя метки перехода.

Описание

Если результат логической операции равен “1” , то инструкция **JCB <метка>** прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Линейное выполнение программы продолжается с места перехода, определенного меткой. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Если результат логической операции равен “0”, переход не будет выполнен. RLO устанавливается в “1” и выполнение программы продолжается со следующей инструкции.

Независимо от значения RLO, он копируется инструкцией **JCB <метка>** в бит BR.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	x	-	-	-	-	0	1	1	0

Пример

```
STL                          Комментарий
      A     I 1.0
      A     I 1.2
      JCB   JOVR   //Переход при RLO = 1 на метку JOVR. Значение RLO копируется в бит BR .
      L     IW8    // Выполнение программы продолжается здесь, если переход не выполнен.
      T     MW22
JOVR: A     I 2.1  // Выполнение программы продолжается здесь, при переходе на метку JOVR.
```

6.7 JNB: Переход при RLO = 0 с сохранением его в BR

Формат

JNB <метка>

Адрес	Описание
<метка >	Символьное имя метки.

Описание

Если результат логической операции равен “0” , то инструкция **JNB <метка>** прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Линейное выполнение программы продолжается с места перехода, определенного меткой. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Если результат логической операции равен “1”, переход не будет выполнен. RLO устанавливается в “1” и выполнение программы продолжается со следующей инструкции.

Независимо от значения RLO, он копируется инструкцией **JNB <метка>** в бит BR.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	x	-	-	-	-	0	1	1	0

Пример

```
STL                     Комментарий
      A     I 1.0
      A     I 1.2
      JNB   JOVR    // Переход при RLO = 0 на метку JOVR. Значение RLO копируется в бит BR.
      L     IW8     // Выполнение программы продолжается здесь, если переход не выполнен.
      T     MW22
JOVR: A     I 2.1   // Выполнение программы продолжается здесь, при переходе на метку JOVR.
```

6.8 JBI: Переход при BR = 1

Формат

JBI <метка>

Address	Описание
<метка >	Символьное имя метки.

Описание

Если статус бита BR равен “1” , то инструкция **JBI <метка>** прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Метка может состоять из четырех символов, первый из которых должен быть буквой. Метка перехода должна заканчиваться двоеточием "**:**" и должна ставиться в строке, содержащей инструкцию. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	1	-	0

6.9 JNBI: Переход при BR = 0

Формат

JNBI <метка>

Адрес	Описание
<метка >	Символьное имя метки.

Описание

Если статус бита BR равен “0” , то инструкция **JNBI <метка>** прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	1	-	0

6.10 JO: Переход OV = 1

Формат

JO <метка>

Адрес	Описание
<метка >	Символьное имя метки.

Описание

Если статус бита OV равен “1” , то инструкция **JO <метка>** прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

При работе с несколькими математическтми инструкциями следует проверять корректность их работы и отсутствие переполнения после каждной отдельной инструкции или пролизводить оценку вцелом с использованием инструкции **JOS**.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL                          Комментарий
      L     MW10
      L     3
      *I                     //Умножение содержимого  MW10 на"3".
      JO    OVER             //Переход на метку  при переполнении (OV=1).
      T     MW10             // Выполнение программы продолжается здесь, если переход не выполнен.
      A     M 4.0
      R     M 4.0
      JU    NEXT
OVER: AN    M 4.0            // Выполнение программы продолжается здесь, при переходе на метку OVER.
      S     M 4.0
NEXT: NOP 0
```

6.11 JOS: Переход при OS = 1

Формат

JOS <метка>

Адрес	Описание
<метка >	Символическое имя метки.

Описание

Если статус бита OS равен “1” , то инструкция **JOS <метка>** прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	0	-	-	-	-

Пример

```
STL                          Комментарий
      L     IW10
      L     MW12
      *I
      L     DBW25
      +I
      L     MW14
      -I
      JOS   OVER   //Переход по переполнению в одной из трех инструкций умножения при
                   OS=1. (См.Примечание).
      T     MW16   // Выполнение программы продолжается, если переход не выполнен.
      A     M 4.0
      R     M 4.0
      JU    NEXT
OVER: AN    M 4.0  // Выполнение программы продолжается при переходе на метку OVER.
      S     M 4.0
NEXT: NOP 0        // Выполнение программы продолжается при переходе на метку NEXT.
```

Примечание

В таком случае не следует использовать инструкцию **JO**. Инструкция **JO** проверяет на переполнение только последнюю математическую инструкцию.

6.12 JZ: Переход при нулевом результате

Формат

JZ <метка>

Адрес	Описание
<метка >	Символьное имя метки.

Описание

Если состояния битов CC 1 и CC 0 равны “0” , то инструкция **JZ <метка>** (переход при нулевом результате) прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL                     Комментарий
      L     MW1
            0
      SRW   1
      JZ    ZERO  // Переход на метку ZERO при последнем сдвинутом бите “ 0”.
      L     MW2   // Выполнение программы продолжается, если переход не выполнен.
      INC   1
      T     MW2
      JU    NEXT
ZERO: L     MW4   // Выполнение программы продолжается при переходе на метку ZERO.
      INC   1
      T     MW4
NEXT: NOP 0       // Выполнение программы продолжается при переходе на метку NEXT
```

6.13 JN: Переход при ненулевом результате

Формат

JN <метка>

Адрес	Описание
<метка >	Символьное имя метки.

Описание

Если состояния битов CC 1 и CC 0 не равны (CC 1=0/CC 0=1 или CC 1=1/CC 0=0), это означает неравенство нулю полученного результата . В этом случае инструкция **JN <метка> (**переход при ненулевом результате) прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL                          Комментарий
      L     IW8
      L     MW12
      XOW
      JN    NOZE    // Переход на метку NOZE если содержимое ACCU 1-L не равно " 0 ".
      AN    M 4.0   // Выполнение программы продолжается, если переход не выполнен.
      S     M 4.0
      JU    NEXT
NOZE: AN    M 4.1   // Выполнение программы продолжается здесь при переходе на метку  NOZE.
      S     M 4.1
NEXT: NOP 0         // Выполнение программы продолжается здесь при переходе на метку NEXT.
```

6.14 JP: Переход при положительном результате

Формат

JP <метка>

Адрес	Описание
<метка >	Символьное имя метки.

Описание

Если состояния битов: CC 1=1 и CC 0=0, это означает что полученный результат больше нуля. В этом случае инструкция **JP <метка> (**переход при положительном результате) прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL                          Комментарий
      L     IW8
      L     MW12
      -I                     //Вычитание содержимого MW12 из  IW8.
      JP    POS              // Переход  при результате >0 (т.е., ACCU 1 > 0).
      AN    M 4.0            //Выполнение программы продолжается, если переход не выполнен.
      S     M 4.0
      JU    NEXT
POS:  AN    M 4.1            // Выполнение программы продолжается здесь при переходе на метку POS.
      S     M 4.1
NEXT: NOP 0                  // Выполнение программы продолжается здесь при переходе на метку
                             NEXT.
```

6.15 JM: Переход при отрицательном результате

Формат

JM <метка>

Адрес	Описание
<метка >	Символьное имя метки.

Описание

Если состояния битов: CC 1=0 и CC 0=1, это означает что полученный результат меньше нуля. В этом случае инструкция **JM <метка> (**переход при отрицательном результате) прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL                     Комментарий
      L     IW8
      L     MW12
      -I                // Вычитание содержимого MW12 из  IW8.
      JM    NEG         // Переход  при результате <0 (т.е.,  ACCU 1 < 0).
      AN    M 4.0       // Выполнение программы продолжается, если переход не выполнен
      S     M 4.0
      JU    NEXT
NEG:  AN    M 4.1       // Выполнение программы продолжается здесь при переходе на метку NEG.
      S     M 4.1
NEXT: NOP 0               // Выполнение программы продолжается здесь при переходе на метку NEXT.
```

6.16 JPZ: Переход при неотрицательном результате

Формат

JPZ <метка>

Адрес	Описание
<метка >	Символьное имя метки.

Описание

Если состояния битов CC 1=0/CC 0=0 или CC 1=1/CC 0=0, это означает что полученный результат не меньше нуля. В этом случае инструкция **JPZ <метка> (**переход при неотрицательном результате) прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	

Пример

```
STL                     Комментарий
      L     IW8
      L     MW12
      -I                // Вычитание содержимого MW12 из  IW8.
      JPZ   REG0        // Переход  при результате >=0 (т.е.,  ACCU 1 >= 0).
      AN    M 4.0       // Выполнение программы продолжается, если переход не выполнен.
      S     M 4.0
      JU    NEXT
REG0: AN    M 4.1       // Выполнение программы продолжается здесь при переходе на метку REG0.
      S     M 4.1
NEXT: NOP 0               // Выполнение программы продолжается здесь при переходе на метку NEXT.
```

6.17 JMZ: Переход при отрицательном или нулевом результате

Формат

JMZ <метка>

Адрес	Описание
<метка >	Символьное имя метки.

Описание

Если состояния битов CC 1=0/CC 0=0 или CC 1=0/CC 0=1, то это означает что полученный результат не больше нуля. В этом случае инструкция **JMZ <метка>** (переход при отрицательном или нулевом результате) прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL                          Комментарий
      L     IW8
      L     MW12
      -I                     // Вычитание содержимого MW12 из  IW8..
      JMZ   RGE0             // Переход  при результате <=0 (т.е.,  ACCU 1 <= 0).
      AN    M 4.0            // Выполнение программы продолжается, если переход не выполнен.
      S     M 4.0
      JU    NEXT
RGE0: AN    M 4.1            // Выполнение программы продолжается здесь при переходе на метку RGE0.
      S     M 4.1
NEXT: NOP 0                  // Выполнение программы продолжается здесь при переходе на метку NEXT.
```

6.18 JUO: Переход при недействительном результате

Формат

JUO <метка>

Адрес	Описание
<метка >	Символьное имя метки.

Описание

Если состояния битов CC 1=1 и CC 0=1, в этом случае инструкция **JUO <метка>** (переход при недействительном результате) прерывает линейную процедуру выполнения программы, чтобы возобновить обработку с метки перехода. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Состояния битов CC 1=1 и CC 0=1 в следующих случаях:

- Выполнение деления на 0
- Выполнение недопустимой инструкции
- Результат сравнения чисел с плавающей точкой "неопределен," например, при использовании недопустимого формата числа типа REAL.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL                          Комментарий
      L     MD10
      L     ID2
      /D                     //Деление содержимого  MD10 на содержимое ID2.
      JUO   ERRO             //Переход при делении на “0” (т.е. , ID2 = 0).
      T     MD14             //Продолжение программы при невыполнении перехода.
      A     M 4.0
      R     M 4.0
      JU    NEXT
ERRO: AN    M 4.0            //Выполнение программы при переходе на метку ERRO.
      S     M 4.0
NEXT: NOP 0                  // Выполнение программы при переходе на метку NEXT.
```

6.19 LOOP: Циклическое управление

Формат

LOOP <метка>

Адрес	Описание
<метка >	Символьное имя метки.

Описание

Инструкция **LOOP <метка>** уменьшает содержимое ACCU 1-L на 1. Затем производится проверка младшего слова аккумулятора 1.Если оно не равно 0, то выполняется переход к метке, указанной в операнде инструкции LOOP; в противном случае выполняется следующая за ней инструкция. Переход выполняется до тех пор, пока содержимое ACCU 1-L не станет равным нулю.

Последовательное выполнение программы, продолжается с метки перехода. Возможны переходы как вперед, так и назад. Переходы могут выполняться внутри одного блока, т.е. инструкция перехода и метка перехода должны находиться внутри одного блока. Метка не должна повторяться внутри блока. Максимальная длина перехода -32768 или +32767 слов программного кода. Фактическое максимальное число инструкций, которые можно пропустить с помощью инструкции перехода зависит от используемых в программе инструкций (одно-, двух-, или трехсловные инструкции).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример работы цикла

```
STL                    Комментарий
      L     L#1        //Загрузка константы (32 бита) в ACCU 1.
      T     MD20       //Передача содержимого ACCU 1 в  MD20 (инициализация).
      L     5          //Загрузка счетчика цикла в ACCU 1-L.
NEXT: T     MW10       //Метка перехода = начало цикла / сохранение ACCU 1-L в счетчике циклов.
      L     MD20
      *     D          //Умножение текущего содержимого MD20 на текущее значение  MB10.
      T     MD20       //Сохранение результата в MD20.
      L     MW10       //Закрузка  значения счетчика в  ACCU 1.
      LOOP  NEXT       //Уменьшение содержимого ACCU 1 и переход на метку NEXT если ACCU 1-L > 0.
      L     MW24       //Выполнение программы продолжается после окончания выполнения цикла.
      L     200
      >I
```

7 Математические инструкции с целыми числами

7.1 Обзор математических инструкций с целыми числами

Описание

Математические инструкции производят обработку содержимого аккумуляторов 1 и 2. Старое содержимое аккумулятора 1 при выполнении инструкции загрузки сдвигается в аккумулятор 2. При выполнении инструкции, результат сохраняется в аккумуляторе 1, содержимое аккумулятора 2 остается неизменным.

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

С помощью математических инструкций, Вы можете выполнять следующие операции с **двумя целыми числами** (16 и 32 бита):

- +I Сложение ACCU 1 и ACCU 2 в формате Integer (16-бит)
- -I Вычитание ACCU 1 из ACCU 2 в формате Integer (16-бит)
- *I Умножение ACCU 1 на ACCU 2 в формате Integer (16-бит)
- /I Деление ACCU 2 на ACCU 1 в формате Integer (16-бит)
- + Сложение констант типа Integer (16, 32 Бит)
- +D Сложение ACCU 1 и ACCU 2 в формате Double Integer (32-бит)
- -D Вычитание ACCU 1 из ACCU 2 в формате Double Integer (32-бит)
- *D Умножение ACCU 1 и ACCU 2 в формате Double Integer (32-бит)
- /D Деление ACCU 2 на ACCU 1 в формате Double Integer (32-бит)
- MOD Получение остатка от деления в формате Double Integer (32-бит)

Смотрите также оценку битов слова состояния при выполнении математических инструкций над целыми числами.

7.2 Оценка битов слова состояния при выполнении математических инструкций с целыми числами

Описание

Математические инструкции с целыми числами приводят к изменениям следующих бит слова состояния: CC1 и CC0, OV и OS.

В следующих таблицах показано изменение битов слова состояния при выполнении инструкций с числами типа Integers (16 и 32 бит):

Допустимый диапазон значения	**CC 1**	**CC 0**	**OV**	**OS**
0 (ноль)	0	0	0	*
16 бит: -32 768 <= результат< 0 (отрицательное число) 32 бит: -2 147 483 648 <= результат< 0 (отрицательное число)	0	1	0	*
16 бит: 32 767 >= результат > 0 (положительное число) 32 бит: 2 147 483 647 >= результат > 0 (положительное число)	1	0	0	*

* Бит OS не изменяется при выполнении этих инструкций.

Недопустимый диапазон значения	**CC1**	**CC0**	**OV**	**OS**
Недопустимо малое значение (сложение) 16 бит: результат = -65536 32 бит: результат = -4 294 967 296	0	0	1	1
Недопустимо малое значение (умножение) 16 бит: результат < -32 768 (отрицательное число) 32 битs: результат < -2 147 483 648 (отрицательное число)	0	1	1	1
Переполнение (сложение , вычитание) 16 бит: результат > 32 767 (положительное число) 32 бит: результат > 2 147 483 647 (положительное число)	0	1	1	1
Переполнение (умножение, деление) 16 бит: результат > 32 767 (положительное число) 32 бит: результат > 2 147 483 647 (положительное число)	1	0	1	1
Переполнение (сложение , вычитание) 16 бит: результат < -32. 768 (отрицательное число) 32 бит: результат < -2 147 483 648 (отрицательное число)	1	0	1	1
Деление на 0	1	1	1	1

Инструкция	**CC1**	**CC0**	**OV**	**OS**
+D: результат = -4 294 967 296	0	0	1	1
/D или MOD: деление на 0	1	1	1	1

7.3 +I: Сложение целых чисел(16-бит) в ACCU 1 и ACCU 2

Формат

+I

Описание

Инструкция **+I :** (сложение двух целых чисел (16 бит)) выполняет сложение содержимого младших слов аккумуляторов (ACCU 1-L и ACCU 2-L) и сохраняет результат в ACCU 1-L. Содержимое ACCU 1-L и ACCU 2-L интерпретируется как 16-битовые целые числа. Инструкция выполняется независимо от RLO и не изменяет его. Биты слова состояния CC 1, CC 0, OS и OV устанавливаются в зависимости от полученного результата. В случае выхода за нижний или верхний пределы при выполнении инструкции, результат представляется в виде 32-битового целого числа вместо 16-битового целого числа, с установкой битов переполнения.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами.

Аккумулятор 4 остается неизменным.

Смотрите также раздел “Оценка битов слова состояния при выполнении математических инструкций над целыми числами”.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	x	-	-	-	-

Диапазон значений результата сложения	CC 1	CC 0	OV	OS
Сумма = 0	0	0	0	-
-32768 <= Сумма < 0	0	1	0	-
32767 >= Сумма > 0	1	0	0	-
Сумма = -65536	0	0	1	1
65534 >= Сумма > 32767	0	1	1	1
-65535 <= Сумма < -32768	1	0	1	1

Пример

```
STL                 Комментарий
L     IW10          //Загрузка значения IW10 в ACCU 1-L.
L     MW14          //Перемещение содержимого ACCU 1-L в ACCU 2-L. Загрузка значения MW14 в ACCU 1-L.
+I                  //Сложение ACCU 2-L и ACCU 1-L; сохранение результата в ACCU 1-L.
T     DB1.DBW25     //Содержимое ACCU 1-L (результат) передается в DBW25 блока данных DB1.
```

7.4 –I: Вычитание целых чисел (16-бит) в ACCU 1 из ACCU 2

Формат

-I

Описание

Инструкция **-I: (**Вычитание двух целых чисел (16 бит))выполняет вычитание содержимого ACCU 1-L из ACCU 2-L и сохраняет результат в ACCU 1-L. Содержимое ACCU 1-L и ACCU 2-L интерпретируется как 16-битовые целые числа. Инструкция выполняется независимо от RLO и не изменяет его. Биты слова состояния CC 1, CC 0, OS и OV устанавливаются в зависимости от полученного результата. В случае выхода за нижний или верхний пределы при выполнении инструкции, результат представляется в виде 32-битового целого числа вместо 16-битового целого числа, с установкой битов переполнения.Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами.

Аккумулятор 4 остается неизменным.

Смотрите также раздел “Оценка битов слова состояния при выполнении математических инструкций над целыми числами”.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	x	-	-	-	-

Диапазон значений результата вычитания	CC 1	CC 0	OV	OS
Результат = 0	0	0	0	-
-32768 <= Результат < 0	0	1	0	-
32767 >= Результат > 0	1	0	0	-
65534 >= Результат > 32767	0	1	1	1
-65535 <= Результат < -32768	1	0	1	1

Пример

STL		Комментарий
L	**IW10**	**//Загрузка значения IW10 в ACCU 1-L.**
L	**MW14**	**//Передача содержимого ACCU 1-L в ACCU 2-L и загрузка значения MW14 в ACCU 1-L.**
-I		**//Вычитание ACCU 1-L из ACCU 2-L; сохранение результата в ACCU 1- L.**
T	**DB1.DBW25**	**//Содержимое ACCU 1-L (результат) передается в DBW25 блока данных DB1.**

7.5 *I: Умножение целых чисел (16-бит) в ACCU 1 и ACCU 2

Формат

***I**

Описание

Инструкция ***I: (**Умножение двух целых чисел (16 бит))выполняет умножение содержимого ACCU 1-L и ACCU 2-L и сохраняет результат в ACCU 1-L. Содержимое ACCU 1-L и ACCU 2-L интерпретируется как 16-битовые целые числа. Результат сохраняется в аккумуляторе 1 как 32- битовое целое число. Если биты слова состояния OV1 = 1 и OS = 1, это означает что результат умножения находится за пределами 16-битового целого числа. Инструкция выполняется независимо от RLO и не изменяет его. Биты слова состояния CC 1, CC 0, OS и OV устанавливаются в зависимости от полученного результата.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами. Аккумулятор 4 остается неизменным.

Смотрите также раздел “Оценка битов слова состояния при выполнении математических инструкций над целыми числами”.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	x	-	-	-	-

Диапазон значений результата умножения	CC 1	CC 0	OV	OS
Результат = 0	0	0	0	-
-32768 <= Результат < 0	0	1	0	-
32767 >= Результат > 0	1	0	0	-
1073741824 >= Результат > 32767	0	1	1	1
-1073709056 <= Результат < -32768	1	0	1	1

Пример

```
STL                 Комментарий
L     IW10          //Загрузка значения IW10 в ACCU 1-L.
L     MW14          //Передача содержимого ACCU 1-L в ACCU 2-L и загрузка содержимого MW14 в
                    ACCU 1-L.
*I                  //Умножение ACCU 2-L и ACCU 1-L, сохранение результата в ACCU 1.
T     DB1.DBW25     //Содержимое ACCU 1 (результат) передается в DBW25 блока данных DB1.
```

7.6 /I: Деление целого числа(16-бит) в ACCU 2 на ACCU 1

Формат

/I

Описание

Инструкция **/I :** (Деление двух целых чисел (16 бит))выполняет деление содержимого ACCU 2-L на ACCU 1-L и сохраняет результат в ACCU 1-L. Содержимое ACCU 1-L и ACCU 2-L интерпретируется как 16-битовые целые числа. Результат сохраняется в аккумуляторе 1 и состоит из двух 16-битовых целых чисел. Целая часть частного от деления находится в ACCU 1-L и остаток располагается в ACCU 1-H. Инструкция выполняется независимо от RLO и не изменяет его. Биты слова состояния CC 1, CC 0, OS и OV устанавливаются в зависимости от полученного результата.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами. Аккумулятор 4 остается неизменным.

Смотрите также раздел “Оценка битов слова состояния при выполнении математических инструкций над целыми числами”.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	x	-	-	-	-

Диапазон значений результата деления	CC 1	CC 0	OV	OS
Результат = 0	0	0	0	-
-32768 <= Результат < 0	0	1	0	-
32767 >= Результат > 0	1	0	0	-
Результат = 32768	1	0	1	1
Деление на ноль	1	1	1	1

Пример

STL		Комментарий
L	**IW10**	**//Загрузка значения IW10 в ACCU 1-L.**
L	**MW14**	**//Передача содержимого ACCU 1-L в ACCU 2-L и загрузка значения MW14 в ACCU 1-L.**
/I		**//Деление ACCU 2-L на ACCU 1-L; сохранение результата в ACCU 1: ACCU 1-L: частное, ACCU 1-H: остаток**
T	**MD20**	**//Содержимое ACCU 1 передается в MD20.**

Пример: Деление 13 на 4

Содержимое ACCU 2-L перед выполнением инструкции (IW10): "13"
Содержимое ACCU 1-L перед выполнением инструкции (MW14):"4"
Инструкция /I (ACCU 2-L / ACCU 1-L): "13/4"
Содержимое ACCU 1-L после выполнения инструкции (частное):"3"
Содержимое ACCU 1-H после выполнения инструкции (остаток):"1"

7.7 +: Сложение ACCU1 с целыми константами

Формат

+ <целая константа>

Адрес	Тип данных	Описание
<целая константа>	(16 или 32-битовое целое)	Константа для сложения

Описание

Инструкция **+ <целая константа>:** выполняет сложение целой константы с содержимым ACCU 1 и сохраняет результат в ACCU 1. Инструкция выполняется независимо от битов слова состояния и не изменяет их.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами. Аккумулятор 4 остается неизменным.

Инструкция **+ <16-битовая целая константа>:** выполняет сложение 16-битовой целой константы (в диапазоне от -32768 до +32767) с содержимым ACCU 1-L и сохраняет результат в ACCU 1-L.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами. Аккумулятор 4 остается неизменным.

Инструкция **+ <32-битовая целая константа>:** выполняет сложение 32-битовых целых чисел (в диапазоне от – 2 147 483 648 до 2 147 483 647) с содержимым ACCU 1 и сохраняет результат в ACCU 1.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами. Аккумулятор 4 остается неизменным.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример 1

STL		Комментарий
L	IW10	//Загрузка значения IW10 в ACCU 1-L.
L	MW14	//Перемещение содержимого ACCU 1-L в ACCU 2-L. Загрузка значения MW14 в ACCU 1-L.
+I		//Сложение ACCU 2-L и ACCU 1-L; сохранение результата в ACCU 1-L.
+	25	//Сложение ACCU 1-L и 25; сохранение результата в ACCU 1-L.
T	DB1.DBW25	//Передача содержимого ACCU 1-L (результат) в DBW25 блока данных DB1.

Пример 2

STL		Комментарий
L	IW12	
L	IW14	
+	100	//Сложение ACCU 1-L с константой 100; сохранение результата в ACCU 1-L.
>I		//При ACCU 2 > ACCU 1 или IW12 > (IW14 + 100)
JC	NEXT	//Выполнить условный переход на метку NEXT.

Пример 3

STL		Комментарий
L	MD20	
L	MD24	
+D		//Сложение ACCU 1 и ACCU 2; сохранение результата в ACCU 1.
+	L#-200	//Сложение ACCU 1 и -200; сохранение результата в ACCU 1.
T	MD28	

7.8 +D: Сложение двойных целых чисел(32-бит) в ACCU 1 и ACCU 2

Формат

+D

Описание

Инструкция **+ D :** (сложение двух двойных целых чисел (32- бита)) выполняет сложение содержимого аккумуляторов (ACCU 1 и ACCU 2) и сохраняет результат в ACCU 1. Содержимое ACCU 1и ACCU 2 интерпретируется как 32-битовые целые числа. Инструкция выполняется независимо от RLO и не изменяет его. Биты слова состояния CC 1, CC 0, OS и OV устанавливаются в зависимости от полученного результата.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами.

Аккумулятор 4 остается неизменным.

Смотрите также раздел “Оценка битов слова состояния при выполнении математических инструкций над целыми числами”.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	x	-	-	-	-

Диапазон значений результата сложения	CC 1	CC 0	OV	OS
Сумма = 0	0	0	0	-
-2147483648 <= Сумма < 0	0	1	0	-
2147483647 >= Сумма > 0	1	0	0	-
Сумма = -4294967296	0	0	1	1
4294967294 >= Сумма > 2147483647	0	1	1	1
-4294967295 <= Сумма < -2147483648	1	0	1	1

Пример

STL		Комментарий
L	**ID10**	**//Загрузка значения ID10 в ACCU 1.**
L	**MD14**	**//Передача содержимого ACCU 1 в ACCU 2. Загрузка значения MD14 в ACCU 1.**
+D		**//Сложение ACCU 2 и ACCU 1; сохранение результата ACCU 1.**
T	**DB1.DBD25**	**//Содержимое ACCU 1 (результат) передается в DBD25 блока данных DB1.**

7.9 –D: Вычитание двойных целых чисел (32-бита) ACCU 1 из ACCU 2

Формат

-D

Описание

Инструкция **- D:** (Вычитание двойных целых чисел (32 бита))выполняет вычитание содержимого ACCU 1 из ACCU 2 и сохраняет результат в ACCU 1. Содержимое ACCU 1 и ACCU 2 интерпретируется как 32-битовые двойные целые числа. Инструкция выполняется независимо от RLO и не изменяет его. Биты слова состояния CC 1, CC 0, OS и OV устанавливаются в зависимости от полученного результата.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами.

Аккумулятор 4 остается неизменным.

Смотрите также раздел “Оценка битов слова состояния при выполнении математических инструкций над целыми числами”.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	x	-	-	-	-

Диапазон значений результата вычитания	CC 1	CC 0	OV	OS
Результат = 0	0	0	0	-
-2147483648 <= Результат < 0	0	1	0	-
2147483647 >= Результат > 0	1	0	0	-
4294967295 >= Результат > 32767	0	1	1	1
-4294967295 <= Результат < -2147483648	1	0	1	1

Пример

```
STL                 Комментарий
L     ID10          //Загрузка значения ID10 в ACCU 1.
L     MD14          //Перемещение содержимого ACCU 1 в ACCU 2. Загрузка значения MD14 в ACCU 1.
-D                  //Вычитание ACCU 1 из ACCU 2; сохранение результата в ACCU 1.
T     DB1.DBD25     //Содержимое ACCU 1 (результат) передается в DBD25 блока данных DB1.
```

7.10 *D Умножение двойных целых чисел (32-бита) в ACCU 1 и ACCU 2

Формат

***D**

Описание

Инструкция *** D : (**Умножение двойных целых чисел (32 бита))выполняет умножение содержимого ACCU 2 на ACCU 1 и сохраняет результат в ACCU 1. Содержимое ACCU 1 и ACCU 2 интерпретируется как 32-битовые двойные целые числа. Результат сохраняется в аккумуляторе 1 как 32-битовое целое число. Если биты слова состояния OV1 = 1 и OS = 1, это означает что результат умножения находится за пределами 32-битового целого числа. Инструкция выполняется независимо от RLO и не изменяет его. Биты слова состояния CC 1, CC 0, OS и OV устанавливаются в зависимости от полученного результата.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами. Аккумулятор 4 остается неизменным.

Смотрите также раздел “Оценка битов слова состояния при выполнении математических инструкций над целыми числами”.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	x	-	-	-	-

Диапазон значений результата умножения	CC 1	CC 0	OV	OS
Результат = 0	0	0	0	-
-2147483648 <= Результат < 0	0	1	0	-
2147483647 >= Результат > 0	1	0	0	-
Результат > 2147483647	1	0	1	1
Результат < -2147483648	0	1	1	1

Пример

```
STL             Комментарий
L     ID10      //Загрузка значения ID10 в ACCU 1.
L     MD14      //Перемещение содержимого ACCU 1 в ACCU 2. Загрузка содержимого MD14 в
                ACCU 1.
*D              //Умножение ACCU 2 на ACCU 1; сохранение результата в ACCU 1.
T     DB1.DBD25 //Содержимое ACCU 1 передается в DBD25 блока данных DB1.
```

7.11 /D: Деление двойных целых чисел (32-бита) ACCU 2 на ACCU 1

Формат

/D

Описание

Инструкция **/D : (**Деление двойных целых чисел (32 бита))выполняет деление содержимого ACCU 2 на ACCU 1 и сохраняет результат в ACCU 1. Содержимое ACCU 1 и ACCU 2 интерпретируется как 32-битовые целые числа. Результат сохраняется в аккумуляторе 1 и содержит только целую часть частного от деления. Остаток от деления можно получить с помощью инструкции **MOD**. Инструкция выполняется независимо от RLO и не изменяет его. Биты слова состояния CC 1, CC 0, OS и OV устанавливаются в зависимости от полученного результата.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами. Аккумулятор 4 остается неизменным.

Смотрите также раздел “Оценка битов слова состояния при выполнении математических инструкций над целыми числами”.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	x	-	-	-	-

Диапазон значений результата деления	CC 1	CC 0	OV	OS
Результат = 0	0	0	0	-
-2147483648 <= Результат < 0	0	1	0	-
2147483647 >= Результат > 0	1	0	0	-
Результат = 2147483648	1	0	1	1
Деление на ноль	1	1	1	1

Пример

```
STL          Комментарий
L    ID10    //Загрузка значения ID10 в ACCU 1.
L    MD14    //Передача содержимого ACCU 1 в ACCU 2. Загрузка значения MD14 в ACCU 1.
/D           //Деление ACCU 2 на ACCU 1; сохранение результата в ACCU 1.
 T   MD20    //Содержимое ACCU 1 передается в MD20.
```

Пример: Деление 13 на 4

Содержимое ACCU 2 перед инструкцией (ID10): "13"
Содержимое ACCU 1 перед инструкцией (MD14): "4"
Инструкция /D (ACCU 2 / ACCU 1): "13/4"
Содержимое ACCU 1 после инструкции (частное): "3"

7.12 MOD : Получение остатка от деления двойных целых чисел (32-бит)

Формат

MOD

Описание

Инструкция **MOD:** вычисляет остаток от деления 32-битовых двойных целых чисел после деления содержимого ACCU 2 на ACCU 1. Содержимое ACCU 1 и ACCU 2 интерпретируется как 32-битовое целое число. Результат дает только остаток от деления целых чисел и располагает его в аккумуляторе 1. (Для получения целой части частного должна использоваться инструкция /D).

Инструкция выполняется независимо от RLO и не изменяет его. Биты слова состояния CC 1, CC 0, OS и OV устанавливаются в зависимости от полученного результата.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами. Аккумулятор 4 остается неизменным.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	x	-	-	-	-

Диапазон значений результата деления	CC 1	CC 0	OV	OS
Остаток = 0	0	0	0	-
-2147483648 <= Остаток < 0	0	1	0	-
2147483647 >= Остаток > 0	1	0	0	-
Деление на ноль	1	1	1	1

Пример

STL		Комментарий
L	**ID10**	**//Загрузка значения ID10 в ACCU 1.**
L	**MD14**	**//Перемещение содержимого ACCU 1 в ACCU 2. Загрузка значения MD14 в ACCU 1.**
MOD		**//Вычисление остатка от деления ACCU 2 на ACCU 1, сохранение результата в ACCU 1.**
T	**MD20**	**//Содержимое ACCU 1 передается в MD20.**

Пример: Деление 13 на 4

Содержимое ACCU 2 перед инструкцией (ID10): "13"
Содержимое ACCU 1 перед инструкцией (MD14): "4"
Инструкция MOD (ACCU 2 / ACCU 1): "13/4"
Содержимое ACCU 1 после инструкции (остаток): "1"

8 Математические инструкции над числами с плавающей точкой

8.1 Обзор математических инструкций над числами с плавающей точкой

Описание

Математические инструкции производят обработку содержимого аккумуляторов 1 и 2 . Старое содержимое аккумулятора 1 при инструкции загрузки сдвигается в аккумулятор 2. При выполнении инструкции, результат сохраняется в аккумуляторе 1, содержимое аккумулятора 2 остается неизменным.
В CPU с четырьмя аккумуляторами после выполнения математической инструкции, содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 в аккумулятор 3. Старое содержимое аккумулятора 4 не меняется.

IEEE 32-битовые числа с плавающей точкой соответствуют типу данных REAL. Вы можете использовать следующие инструкции с плавающей точкой для обработки **двух 32-битовых IEEE чисел с плавающей точкой**:

- +R Сложение ACCU 1 и ACCU
- -R Вычитание ACCU 1 из ACCU 2
- *R Умножение ACCU 1 и ACCU 2
- /R Деление ACCU 2 на ACCU 1

Вы можете использовать следующие инструкции с плавающей точкой для обработки **одного 32-битового IEEE числа с плавающей точкой**:

- ABS Модуль числа
- SQR Вычисление квадрата
- SQRT Вычисление квадратного корня
- EXP Вычисление экспоненты
- LN Вычисление натурального логарифма
- SB Вычисление синуса угла
- COS Вычисление косинуса угла
- TAN Вычисление тангенса угла
- ASB Вычисление арксинуса
- ACOS Вычисление арккосинуса
- ATAN Вычисление арктангенса

Смотрите также оценку битов слова состояния .

8.2 Оценка битов слова состояния при выполнении математических инструкций с плавающей точкой

Описание

Математические инструкции с плавающей точкой приводят к изменениям следующих битов слова состояния: CC1 и CC0, OV и OS.

В следующих таблицах показано изменение битов слова состояния при выполнении инструкций над числами с плавающей точкой (32 бита):

Допустимый диапазон значения	**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 < 3.402824E+38 (положительное число)	1	0	0	*

* Бит OS не изменяется при выполнении этих инструкций.

Недопустимый диапазон значений	**CC 1**	**CC 0**	**OV**	**OS**
Недопустимо малое значение: -1.175494E-38 < результат < - 1.401298E-45 (отрицательное число)	0	0	1	1
Недопустимо малое значение: +1.401298E-45 < результат < +1.175494E-38 (положительное число)	0	0	1	1
Переполнение: Результат < -3.402823E+38 (отрицательное число)	0	1	1	1
Переполнение: Результат > 3.402823E+38 (положительное число)	1	0	1	1
Недействительное число или недопустимая инструкция (входное значение за пределами допустимого диапазона)	1	1	1	1

8.3 Основные инструкции

8.3.1 +R: Сложение значений в ACCU 1 и ACCU 2 как чисел с плавающей точкой (32-бита, IEEE-FP)

Формат

+R

Описание инструкции

Инструкция **+R :** (сложение двух чисел с плавающей точкой 32-битовых IEEE) складывает содержимое аккумуляторов 1 и 2 и сохраняет результат в аккумуляторе 1. Содержимое аккумулятора 1 и аккумулятора 2 интерпретируется как 32-битовое IEEE число с плавающей точкой. Инструкция выполняется независимо от RLO и не изменяет его. Биты слова состояния CC 1, CC 0, OS и OV устанавливаются в зависимости от полученного результата.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами. Аккумулятор 4 остается неизменным.

Результат

Результат в ACCU 1	CC 1	CC 0	OV	OS	Примечание
+ qNaN	1	1	1	1	
+ бесконечность	1	0	1	1	Переполнение
+ нормализован	1	0	0	-	
+ не нормализован	0	0	1	1	Недопустимо малое значение
+ ноль	0	0	0	-	
- ноль	0	0	0	-	
- не нормализован	0	0	1	1	Недопустимо малое значение
- нормализован	0	1	0	-	
- бесконечность	0	1	1	1	Переполнение
- qNaN	1	1	1	1	

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	x	-	-	-	-

Пример

STL		Комментарий
OPN	**DB10**	
L	**ID10**	**//Загрузка значения ID10 в ACCU 1.**
L	**MD14**	**//Сдвиг значения ACCU 1 в ACCU 2. Загрузка значения MD14 в ACCU 1.**
+R		**//Сложение ACCU 2 и ACCU 1; замена результата в ACCU 1.**
T	**DBD25**	**//Содержимое ACCU 1 (результат) передается в DBD25 блока данных DB10.**

8.3.2 –R: Вычитание значения ACCU 1 из ACCU 2 как чисел с плавающей точкой (32-бита IEEE-FP)

Формат

-R

Описание

Инструкция **-R :** (вычитание 32-битовых чисел с плавающей точкой) вычитает содержимое аккумулятора 1 из аккумулятора 2 и сохраняет результат в аккумуляторе 1. Содержимое аккумулятора 1 и аккумулятора 2 интерпретируется как 32-битовое IEEE число с плавающей точкой. Инструкция выполняется независимо от RLO и не изменяет его. Биты слова состояния CC 1, CC 0, OS и OV устанавливаются в зависимости от полученного результата.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами. Аккумулятор 4 остается неизменным.

Результат

Результат в ACCU 1	CC 1	CC 0	OV	OS	Примечание
+ qNaN	1	1	1	1	
+ бесконечность	1	0	1	1	Переполнение
+ нормализован	1	0	0	-	
+ не нормализован	0	0	1	1	Недопустимо малое значение
+ ноль	0	0	0	-	
- ноль	0	0	0	-	
- не нормализован	0	0	1	1	Недопустимо малое значение
- нормализован	0	1	0	-	
- бесконечность	0	1	1	1	Переполнение
- qNaN	1	1	1	1	

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	x	-	-	-	-

Пример

```
STL             Комментарий
OPN   DB10
L     ID10      //Загрузка значения ID10 в ACCU 1.
L     MD14      //Сдвиг значения ACCU 1 в ACCU 2. Загрузка значения MD14 в ACCU 1.
-R              //Вычитание ACCU 1 из ACCU 2; результат в ACCU 1.
T     DBD25     //Содержимое ACCU 1 (результат)  передается в DBD25 блока данных DB10.
```

8.3.3 *R: Умножение значений в ACCU 1 и ACCU 2 как чисел с плавающей точкой (32-bit IEEE-FP)

Формат

***R**

Описание инструкции

Инструкция ***R :** (умножение 32-битовых чисел с плавающей точкой) умножает содержимое аккумулятора 1 и аккумулятора 2 и сохраняет результат в аккумуляторе 1. Содержимое аккумулятора 1 и аккумулятора 2 интерпретируется как 32-битовое IEEE число с плавающей точкой. Инструкция выполняется независимо от RLO и не изменяет его. Биты слова состояния CC 1, CC 0, OS и OV устанавливаются в зависимости от полученного результата.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами. Аккумулятор 4 остается неизменным.

Результат

Результат в ACCU 1	CC 1	CC 0	OV	OS	Примечание
+ qNaN	1	1	1	1	
+ бесконечность	1	0	1	1	Переполнение
+ нормализован	1	0	0	-	
+ не нормализован	0	0	1	1	Недопустимо малое значение
+ ноль	0	0	0	-	
- ноль	0	0	0	-	
- не нормализован	0	0	1	1	Недопустимо малое значение
- нормализован	0	1	0	-	
- бесконечность	0	1	1	1	Переполнение
- qNaN	1	1	1	1	

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	x	-	-	-	-

Пример

```
STL         Комментарий
OPN   DB10
L     ID10    //Загрузка значения  ID10 в ACCU 1.
L     MD14    //Сдвиг  значения ACCU 1 в ACCU 2. Загрузка значения MD14 в ACCU 1.
*R            //Умножение ACCU 2 и ACCU 1; передача результата в ACCU 1.
T     DBD25   //Содержимое ACCU 1 (результат) передается в DBD25 блока данных DB10.
```

8.3.4 /R: Деление значений в ACCU 1 и ACCU 2 как чисел с плавающей точкой (32-bit IEEE-FP)

Формат

/R

Описание инструкции

Инструкция **/R :** (деление 32-битовых чисел с плавающей точкой) делит содержимое аккумулятора 2 на аккумулятор 1 и сохраняет результат в аккумуляторе 1. Содержимое аккумулятора 1 и аккумулятора 2 интерпретируется как 32-битовое IEEE число с плавающей точкой. Инструкция выполняется независимо от RLO и не изменяет его. Биты слова состояния CC 1, CC 0, OS и OV устанавливаются в зависимости от полученного результата.

Содержимое аккумулятора 2 остается неизменным в CPU с двумя аккумуляторами.

Содержимое аккумулятора 3 копируется в аккумулятор 2, а содержимое аккумулятора 4 копируется в аккумулятор 3 в CPU с четырьмя аккумуляторами. Аккумулятор 4 остается неизменным.

Результат

Результат в ACCU 1	CC 1	CC 0	OV	OS	Примечание
+ qNaN	1	1	1	1	
+ бесконечность	1	0	1	1	Переполнение
+ нормализован	1	0	0	-	
+ не нормализован	0	0	1	1	Недопустимо малое значение
+ ноль	0	0	0	-	
- ноль	0	0	0	-	
- не нормализован	0	0	1	1	Недопустимо малое значение
- нормализован	0	1	0	-	
- бесконечность	0	1	1	1	Переполнение
- qNaN	1	1	1	1	

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	x	-	-	-	-

Пример

```
STL     Комментарий
OPN     DB10
L       ID10    //Загрузка значения ID10 в ACCU 1.
L       MD14    //Сдвиг содержимого  ACCU 1 в ACCU 2. Загрузка значения MD14 в ACCU 1.
/R              //Деление ACCU 2 на ACCU 1; сохранение результата в ACCU 1.
T       DBD20   //Содержимое ACCU 1 (результат) передается в DBD20 блока данных DB10.
```

8.3.5 ABS : Вычисление модуля числа с плавающей точкой

Формат

ABS

Описание

Инструкция **ABS:** (модуль числа с плавающей точкой) вычисляет модуль 32-битового числа с плавающей точкой в ACCU 1. Результат сохраняется в аккумуляторе 1. Инструкция выполняется независимо от бит слова состояния и не меняет их.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL     Комментарий
L       ID8     //Загрузка значения в ACCU 1 (Пример: ID8 = -1.5E+02).
ABS             //Вычисление модуля; сохранение результата  в ACCU 1.
T       MD10    //Передача результата в MD10 (Пример: результат = 1.5E+02).
```

8.4 Дополнительные инструкции для чисел с плавающей точкой

8.4.1 SQR: Вычисление квадрата числа с плавающей точкой

Формат

SQR

Описание инструкции

SQR (вычисление квадрата числа с плавающей точкой IEEE-FP) производит вычисление квадрата числа в ACCU 1. Результат сохраняется в аккумуляторе 1. Инструкция изменяет биты CC 1, CC 0, OV, и OS слова состояния.

Содержимое аккумулятора 2 (как и содержимое аккумуляторов 3 и 4 для CPUs с четырьмя ACCUs) остается неизменным.

Результат

Результат в ACCU 1	CC 1	CC 0	OV	OS	Примечание
+ qNaN	1	1	1	1	
+ бесконечность	1	0	1	1	Переполнение
+ нормализован	1	0	0	-	
+ не нормализован	0	0	1	1	Недопустимо малое значение
+ ноль	0	0	0	-	
- qNaN	1	1	1	1	

Пример

```
STL                          Комментарий
      OPN   DB17   //Открыть блок данных DB17.
      L     DBD0   // Загрузить в ACCU 1 значения из двойного слова данных DBD0 . (Эти данные
                   должны быть в формате числа с плавающей точкой)
      SQR          //  Вычисление квадрата числа с плавающей точкой (32-бита, IEEE-FP) в
                   ACCU 1. Результат в ACCU 1.
      AN    OV     //Опрос бита OV на состояние "0."
      JC    OK     //Если при вычислении квадрата не было ошибки, переход на метку OK.
BEU                //Безусловное окончание блока, если произошла ошибка при SQR инструкции.
OK:   T     DBD4   //Сохранение  результата из ACCU 1 в двойное слово данных DBD4.
```

8.4.2 SQRT : Вычисление квадратного корня числа с плавающей точкой

Формат

SQRT

Описание инструкции

SQRT: (вычисление квадратного корня 32-битного числа с плавающей точкой, IEEE-FP) производит вычисление корня квадратного числа с плавающей точкой в ACCU 1. Результат сохраняется в аккумуляторе 1. Обрабатываемое значение не должно быть отрицательным числом. Результат выдается также положительным. За исключением -0 , при котором результат: -0. Инструкция изменяет биты CC 1, CC 0, OV, и OS слова состояния.

Содержимое аккумулятора 2 (как и содержимое аккумулятора 3 и аккумулятора 4 для CPU с четырьмя аккумуляторами) остается неизменным.

Результат

Результат в ACCU 1	CC 1	CC 0	OV	OS	Примечание
+ qNaN	1	1	1	1	
+ бесконечность	1	0	1	1	Переполнение
+ нормализован	1	0	0	-	
+ не нормализован	0	0	1	1	Недопустимо малое значение
+ ноль	0	0	0	-	
- qNaN	1	1	1	1	

Пример

```
STL                     Комментарий
      L     MD10        //Значение из меркерного двойного слова MD10 загружается в ACCU 1. (Это
                        значение должно быть в формате числа с плавающей точкой).
      SQRT              //Вычисление квадратного корня значения  в ACCU 1 и запись результата в
                        ACCU 1.
      AN    OV          // Опрос бита OV на состояние "0."
      JC    OK          // Если при вычислении квадратного корня не было ошибки, переход на метку.
BEU                     // Безусловное окончание блока, если произошла ошибка при SQRT
                        инструкции.
OK:   T     MD20        // Сохранение  результата из ACCU 1 в двойном слове MD20.
```

8.4.3 EXP: Вычисление экспоненты числа с плавающей точкой

Формат

EXP

Описание инструкции

Инструкция **EXP:** (вычисление экспоненты числа с плавающей точкой, 32-bit, IEEE-FP) вычисляет экспоненциальное значение по основанию *"e"* числа с плавающей точкой в ACCU 1. Результат сохраняется в аккумуляторе 1. Инструкция изменяет биты CC 1, CC 0, OV, и OS слова состояния.

Содержимое аккумулятора 2 (как и содержимое аккумулятора 3 и аккумулятора 4 для CPU с четырьмя аккумуляторами) остается неизменным.

Результат

Результат в ACCU 1	CC 1	CC 0	OV	OS	Примечание
+ qNaN	1	1	1	1	
+ бесконечность	1	0	1	1	Переполнение
+ нормализован	1	0	0	-	
+ не нормализован	0	0	1	1	Недопустимо малое значение
+ ноль	0	0	0	-	
- qNaN	1	1	1	1	

Пример

```
STL                     Комментарий
      L     MD10        // Значение из меркерного двойного слова MD10 загружается в ACCU 1. (Это
                        значение должно быть в формате числа с плавающей точкой).
      EXP               // Вычисление экспоненты числа с плавающей точкой значения  в ACCU 1 и
                        запись результата в ACCU 1.
      AN    OV          // Опрос бита OV на состояние "0."
      JC    OK          // Если при вычислении экспоненты не было ошибки, переход на метку.
BEU                     // Безусловное окончание блока, если произошла ошибка при EXP инструкции.
OK:   T     MD20        // Сохранение  результата из ACCU 1 в двойном слове MD20.
```

8.4.4 LN: Вычисление натурального логарифма числа с плавающей точкой

Формат

LN

Описание инструкции

Инструкция **LN:** выполняет вычисление натурального логарифма 32-битового числа с плавающей точкой IEEE-FP в ACCU 1. Результат сохраняется в аккумуляторе 1. Инструкция изменяет биты CC 1, CC 0, OV, и OS слова состояния.

Содержимое аккумулятора 2 (как и содержимое аккумулятора 3 и аккумулятора 4 для CPU с четырьмя аккумуляторами) остается неизменным.

Результат

Результат в ACCU 1	CC 1	CC 0	OV	OS	Примечание
+ qNaN	1	1	1	1	
+ бесконечность	1	0	1	1	Переполнение
+ нормализован	1	0	0	-	
+ не нормализован	0	0	1	1	Недопустимо малое значение
+ ноль	0	0	0	-	
-ноль	0	0	0	-	
- не нормализован	0	0	1	1	Недопустимо малое значение
- нормализован	0	1	0	-	
- бесконечность	0	1	1	1	Переполнение
- qNaN	1	1	1	1	

Пример

```
STL                     Комментарий
      L     MD10        // Значение из меркерного двойного слова MD10 загружается в ACCU 1. (Это
                        значение должно быть в формате числа с плавающей точкой).
      LN                // Вычисление натурального логарифма числа с плавающей точкой в ACCU 1 и
                        запись результата в ACCU 1.
      AN    OV          // Опрос бита OV на состояние "0."
      JC    OK          // Если при вычислении логарифма не было ошибки, переход на метку.
BEU                     // Безусловное окончание блока, если произошла ошибка при LN инструкции.
OK:   T     MD20        // Сохранение результата из ACCU 1 в двойном слове MD20.
```

8.4.5 SIN: Вычисление синуса угла

Формат

SIN

Описание инструкции

Инструкция **SIN:** выполняет вычисление синуса угла в радианной мере, представленного в виде 32-битового числа с плавающей точкой , IEEE-FP в ACCU 1. Результат сохраняется в аккумуляторе 1. Инструкция изменяет биты CC 1, CC 0, OV, и OS слова состояния.

Содержимое аккумулятора 2 (как и содержимое аккумулятора 3 и аккумулятора 4 для CPU с четырьмя аккумуляторами) остается неизменным.

Результат

Результат в ACCU 1	CC 1	CC 0	OV	OS	Примечание
+ qNaN	1	1	1	1	
+ нормализован	1	0	0	-	
+ не нормализован	0	0	1	1	Недопустимо малое значение
+ ноль	0	0	0	-	
-ноль	0	0	0	-	
- не нормализован	0	0	1	1	Недопустимо малое значение
- нормализован	0	1	0	-	
- qNaN	1	1	1	1	

Пример

STL		Комментарий
L	**MD10**	**// Значение из меркерного двойного слова MD10 загружается в ACCU 1. (Это значение должно быть в формате числа с плавающей точкой).**
SB		**// Вычисление синуса угла числа с плавающей точкой в ACCU 1 и запись результата в ACCU 1.**
T	**MD20**	**// Сохранение результата из ACCU 1 в двойном слово MD20.**

8.4.6 COS : Вычисление косинуса угла

Формат

COS

Описание инструкции

Инструкция **COS:** выполняет вычисление косинуса угла в радианной мере, представленного в виде 32-битового числа с плавающей точкой IEEE-FP в ACCU 1. Результат сохраняется в аккумуляторе 1. Инструкция изменяет биты CC 1, CC 0, OV, и OS слова состояния.

Содержимое аккумулятора 2 (как и содержимое аккумулятора 3 и аккумулятора 4 для CPU с четырьмя аккумуляторами) остается неизменным.

Результат

Результат в ACCU 1	CC 1	CC 0	OV	OS	Примечание
+ qNaN	1	1	1	1	
+ нормализован	1	0	0	-	
+ не нормализован	0	0	1	1	Недопустимо малое значение
+ ноль	0	0	0	-	
-ноль	0	0	0	-	
- не нормализован	0	0	1	1	Недопустимо малое значение
- нормализован	0	1	0	-	
- qNaN	1	1	1	1	

Пример

STL		Комментарий
L	**MD10**	**// Значение из меркерного двойного слова MD10 загружается в ACCU 1. (Это значение должно быть в формате числа с плавающей точкой).**
COS		**// Вычисление косинуса угла числа с плавающей точкой в ACCU 1 и запись результата в ACCU 1.**
T	**MD20**	**// Сохранение результата из ACCU 1 в двойном слово MD20.**

8.4.7 TAN: Вычисление тангенса угла

Формат

TAN

Описание инструкции

Инструкция **TAN:** выполняет вычисление тангенса угла в радианной мере, представленного в виде 32-битового числа с плавающей точкой IEEE-FP в ACCU 1. Результат сохраняется в аккумуляторе 1. Инструкция изменяет биты CC 1, CC 0, OV, и OS слова состояния.

Содержимое аккумулятора 2 (как и содержимое аккумулятора 3 и аккумулятора 4 для CPU с четырьмя аккумуляторами) остается неизменным.

Результат

Результат в ACCU 1	CC 1	CC 0	OV	OS	Примечание
+ qNaN	1	1	1	1	
+бесконечность	1	0	1	1	Переполнение
+ нормализован	1	0	0	-	
+ не нормализован	0	0	1	1	Недопустимо малое значение
+ ноль	0	0	0	-	
-ноль	0	0	0	-	
- не нормализован	0	0	1	1	Недопустимо малое значение
- нормализован	0	1	0	-	
- бесконечность	0	1	1	1	Переполнение
- qNaN	1	1	1	1	

Пример

```
STL                          Комментарий
      L     MD10    // Значение из меркерного двойного слова MD10 загружается в ACCU 1. (Это
                    значение должно быть в формате числа с плавающей точкой).
      TAN           // Вычисление тангенса угла  числа с плавающей точкой в ACCU 1 и запись
                    результата в ACCU 1.
      AN    OV      // Опрос бита OV на состояние "0."
      JC    OK      // Если при вычислении тангенса не было ошибки, переход на метку.
BEU                 // Безусловное окончание блока, если произошла ошибка при TAN инструкции.
OK:   T     MD20    // Сохранение  результата из ACCU 1 в двойном слове MD20.
```

8.4.8 ASIN: Вычисление арксинуса

Формат

ASIN

Описание инструкции

Инструкция **ASIN:** выполняет вычисление арксинуса числа, представленного в виде 32-битового числа с плавающей точкой IEEE-FP в ACCU 1. Результат сохраняется в аккумуляторе 1.

Допустимые значения входной величины:
-1 <= входное значение <= +1

Результат выдается в виде угла в радианной мере. Значения могут находиться в следующем диапазоне:

$-\pi / 2 <= \text{Arcsin (ACCU1)} <= + \pi / 2$, где $\pi = 3.14159...$

Инструкция изменяет биты CC 1, CC 0, OV, и OS слова состояния.

Содержимое аккумулятора 2 (как и содержимое аккумулятора 3 и аккумулятора 4 для CPU с четырьмя аккумуляторами) остается неизменным.

Результат

Результат в ACCU 1	CC 1	CC 0	OV	OS	Примечание
+ qNaN	1	1	1	1	
+ нормализован	1	0	0	-	
+ не нормализован	0	0	1	1	Недопустимо малое значение
+ ноль	0	0	0	-	
-ноль	0	0	0	-	
- не нормализован	0	0	1	1	Недопустимо малое значение
- нормализован	0	1	0	-	
- qNaN	1	1	1	1	

Пример

```
STL                    Комментарий
      L     MD10       // Значение из меркерного двойного слова MD10 загружается в ACCU 1. (Это
                       значение должно быть в формате числа с плавающей точкой).
      ASIN             // Вычисление арксинуса числа с плавающей точкой в ACCU 1 и запись
                       результата в ACCU 1.
      AN    OV         // Опрос бита OV на состояние "0."
      JC    OK         // Если при вычислении арксинуса  не было ошибки, переход на метку.
BEU                    // Безусловное окончание блока, если произошла ошибка в инструкции.
OK:   T     MD20       // Сохранение  результата из ACCU 1 в двойном слове  MD20.
```

8.4.9 ACOS: Вычисление арккосинуса

Формат

ACOS

Описание инструкции

Инструкция **ACOS:** выполняет вычисление арккосинуса числа, представленного в виде 32-битового числа с плавающей точкой IEEE-FP в ACCU 1. Результат сохраняется в аккумуляторе 1.

Допустимые значения входной величины:
-1 <= входное значение <= +1

Результат выдается в виде угла в радианной мере. Значения могут находиться в следующем диапазоне:

0 <= арккосинус (ACCU1) <= π, где π = 3.14159...

Инструкция изменяет биты CC 1, CC 0, OV, и OS слова состояния.

Содержимое аккумулятора 2 (как и содержимое аккумулятора 3 и аккумулятора 4 для CPU с четырьмя аккумуляторами) остается неизменным.

Результат

Результат в ACCU 1	CC 1	CC 0	OV	OS	Примечание
+ qNaN	1	1	1	1	
+ нормализован	1	0	0	-	
+ не нормализован	0	0	1	1	Недопустимо малое значение
+ ноль	0	0	0	-	
-ноль	0	0	0	-	
- не нормализован	0	0	1	1	Недопустимо малое значение
- нормализован	0	1	0	-	
- qNaN	1	1	1	1	

Пример

```
STL                     Комментарий
      L     MD10    // Значение из меркерного двойного слова MD10 загружается в ACCU 1. (Это
                    значение должно быть в формате числа с плавающей точкой).
      ACOS          // Вычисление арккосинуса числа с плавающей точкой в ACCU 1 и запись
                    результата в ACCU 1.
      AN    OV      // Опрос бита OV на состояние "0."
      JC    OK      // Если при вычислении арккосинуса не было ошибки, переход на метку.
BEU                 // Безусловное окончание блока, если произошла ошибка в инструкции.
OK:   T     MD20    // Сохранение результата из ACCU 1 в двойном слове MD20.
```

8.4.10 ATAN : Вычисление арктангенса

Формат

ATAN

Описание инструкции

Инструкция **ATAN:** выполняет вычисление арктангенса числа, представленного в виде 32-битового числа с плавающей точкой IEEE-FP в ACCU 1. Результат сохраняется в аккумуляторе 1.

Результат выдается в виде угла в радианной мере. Значения могут находиться в следующем диапазоне:

$-\pi / 2 <=$ арктангенс (ACCU1) $<= +\pi / 2$, где $\pi = 3.14159...$

Инструкция изменяет биты CC 1, CC 0, OV, и OS слова состояния.

Содержимое аккумулятора 2 (как и содержимое аккумулятора 3 и аккумулятора 4 для CPU с четырьмя аккумуляторами) остается неизменным.

Результат

Результат в ACCU 1	CC 1	CC 0	OV	OS	Примечание
+ qNaN	1	1	1	1	
+ нормализован	1	0	0	-	
+ не нормализован	0	0	1	1	Недопустимо малое значение
+ ноль	0	0	0	-	
-ноль	0	0	0	-	
- не нормализован	0	0	1	1	Недопустимо малое значение
- нормализован	0	1	0	-	
- qNaN	1	1	1	1	

Пример

STL			Комментарий
	L	**MD10**	**// Значение из меркерного двойного слова MD10 загружается в ACCU 1. (Это значение должно быть в формате числа с плавающей точкой).**
	ATAN		**// Вычисление арктангенса числа с плавающей точкой в ACCU 1 и запись результата в ACCU 1.**
	AN	**OV**	**// Опрос бита OV на состояние "0."**
	JC	**OK**	**// Если при вычислении арктангенса не было ошибки, переход на метку.**
BEU			**// Безусловное окончание блока, если произошла ошибка в инструкции.**
OK:	**T**	**MD20**	**// Сохранение результата из ACCU 1 в двойном слове MD20.**

9 Инструкции загрузки и передачи

9.1 Обзор инструкций загрузки и передачи

Описание

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

Следующие инструкции загрузки и передачи могут использоваться:

- L Загрузка
- L STW Загрузка битов слова состояния в ACCU 1
- LAR1 AR2 Загрузка в адресный регистр1 (AR1) значения из AR2
- LAR1 <D> Загрузка в адресный регистр 1 константы (32 -битовый указатель)
- LAR1 Загрузка в адресный регистр 1 значения из ACCU 1
- LAR2 <D> Загрузка в адресный регистр константы (32 -битовый указатель)
- LAR2 Загрузка в адресный регистр 2 значения из ACCU 1

- T Передача
- T STW Передача ACCU 1 в слово состояния
- TAR1 AR2 Передача адресного регистра 1 в адресный регистр 2
- TAR1 <D> Передача адресного регистра 1 в целевую область (32-битовый указатель)
- TAR2 <D> Передача адресного регистра 2 в целевую область (32-битовый указатель)
- TAR1 Передача адресного регистра 1 в ACCU 1
- TAR2 Передача адресного регистра 1 в ACCU 1
- CAR Обмен содержимым адресных регистров 1 и 2

9.2 L Загрузка

Формат

L <адрес>

Адрес	Тип данных	Область	Параметр
<адрес>	BYTE WORD DWORD	E, A, PE, M, L, D, указатель, параметр	0...65535 0...65534 0...65532

Описание

L <адрес> выполняет загрузку указанных в операнде байта, слова или двойного слова в ACCU 1 после предварительного сохранения старого содержимого ACCU 1 в ACCU 2 и обнуления ACCU 1 .

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Примеры

STL		Комментарий
L	**IB10**	**//Загрузка входного байтаIB10 в ACCU 1-L-L.**
L	**MB120**	**//Загрузка меркерного байта MB120 в ACCU 1-L-L.**
L	**DBB12**	**//Загрузка байта данных DBB12 в ACCU 1-L-L.**
L	**DIW15**	**//Загрузка слова данных блока экземпляра DIW15 в ACCU 1-L.**
L	**LD252**	**//Загрузка локального двойного слова данных LD252 в ACCU 1.**
L	**P# I 8.7**	**//Загрузка указателя в ACCU 1.**
L	**OTTO**	**//Загрузка параметра "OTTO" в ACCU 1.**
L	**P# ANNA**	**//Загрузка указателя на параметр в ACCU 1. (Это позволяет загрузить относительный адресный указатель на параметр. Для вычисления абсолютного смещения в блоке мультиэкземпляре FB, к полученному относительному указателю должно быть прибавлено содержимое адресного регистра AR2.)**

Содержимое ACCU 1

Содержимое ACCU 1	ACCU1-H-H	ACCU1-H-L	ACCU1-L-H	ACCU1-L-L
Передвыполнением инструкции загрузки	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX
После выполнения **L MB10** (загрузка байта)	00000000	00000000	00000000	<MB10>
После выполнения **L MW10** (загрузка слова)	00000000	00000000	<MB10>	<MB11>
После выполнения **L MD10** (загрузка двойного слова)	<MB10>	<MB11>	<MB12>	<MB13>
После выполнения **L P# ANNA** (в FB)	<86>	<Битовый указатель на ANNA относительно начала FB >. Для вычисления абсолютного указателя в блоке мультиэкземпляре FB, содержимое регистра AR2 должно быть прибавлено к полученной величине.		
После выполнения **L P# ANNA** (в FC)	<межзонный указатель на данные для передачи через параметр ANNA>			
	X = "1" или "0"			

9.3 L STW Загрузка слова состояния в ACCU 1

Формат

L STW

Описание

L STW (инструкция загрузки слова состояния) заносит в ACCU 1 содержимое слова состояния. Инструкция выполняется независимо от слова состояния и не изменяет его.

Примечание

В контроллерах S7-300, инструкция **L STW** не загружает биты FC, STA, и OR слова состояния. Только биты 1, 4, 5, 6, 7, и 8 загружаются в соответствующие биты младшего слова аккумулятора 1.

Биты слова состояния

	BR	**CC 1**	**CC 0**	**OV**	**OS**	**OR**	**STA**	**RLO**	**/FC**
Записывает:	-	-	-	-	-	-	-	-	-

Пример

STL		**Комментарий**
L	**STW**	**//Загрузка содержимого слова состояния в ACCU 1.**

Содержимое ACCU 1 после выполнения инструкции **L STW** :

Бит	**31-9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Содержимое:	0	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC

9.4 LAR1 Загрузка в адресный регистр 1 значения из ACCU 1

Формат

LAR1

Описание

Инструкция **LAR1** загружает в адресный регистр AR1 содержимое ACCU 1 (32-битовый указатель). ACCU 1 и ACCU 2 не изменяются. Инструкция выполняется независимо от слова состояния и не изменяет его.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

9.5 LAR1 <D> Загрузка в адресный регистр AR 1 константы (32-битовый указатель)

Формат

LAR1 <D>

Адрес	Тип данных	Область	Адрес
<D>	DWORD Константа-указатель	D, M, L	0...65532

Описание

Инструкция **LAR1 <D>** загружает в адресный регистр AR1 содержимое указанного в ней адреса в формате двойного слова <D> или константу-указатель. ACCU 1 и ACCU 2 не изменяются. Инструкция выполняется независимо от слова состояния и не изменяет его.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример: прямая адресация

```
STL              Комментарий
LAR1  DBD20      //Загрузка в AR1 указателя из двойного слова данных DBD20.
LAR1  DID30      //Загрузка в AR1 указателя из двойного слова данных DID30.
LAR1  LD180      //Загрузка в AR1 указателя из двойного словалокальных данных  LD180.
LAR1  MD24       //Загрузка в AR1 указателя из двойного меркерного слова MD24.
```

Пример: константа указатель

```
STL              Комментарий
LAR1  P#M100.0   //Загрузка в AR1 константы 32-битового указателя.
```

9.6 LAR1 AR2 Загрузка в адресный регистр AR 1 значения из AR2

Формат

LAR1 AR2

Описание

Инструкция **LAR1 AR2** загружает в адресный регистр AR1 содержимое адресного регистра AR2. ACCU 1 и ACCU 2 не изменяются. Инструкция выполняется независимо от слова состояния и не изменяет его.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

9.7 LAR2 Загрузка в адресный регистр 2 значения из ACCU 1

Формат

LAR2

Описание

Инструкция **LAR2** загружает в адресный регистр AR2 содержимое ACCU 1 (32-битовый указатель). ACCU 1 и ACCU 2 не изменяются. Инструкция выполняется независимо от слова состояния и не изменяет его.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

9.8 LAR2 <D> Загрузка в адресный регистр AR 1 константы (32-битовый указатель)

Формат

LAR2 <D>

Адрес	Тип данных	Область	Адрес
<D>	DWORD Константа-указатель	D, M, L	0...65532

Описание

Инструкция **LAR2 <D>** загружает в адресный регистр AR2 содержимое указанного в ней адреса в формате двойного слова <D> или константу-указатель. ACCU 1 и ACCU 2 не изменяются. Инструкция выполняется независимо от слова состояния и не изменяет его.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример: прямая адресация

```
STL             Комментарий
LAR2  DBD 20    //Загрузка в AR2 указателя из двойного слова данных DBD20.
LAR2  DID 30    //Загрузка в AR2 указателя из двойного слова данных DID30.
LAR2  LD 180    //Загрузка в AR2 указателя из двойного словалокальных данных  LD180.
LAR2  MD 24     //Загрузка в AR2 указателя из двойного меркерного слова MD24.
```

Пример: константа-указатель

```
STL             Комментарий
LAR2  P#M100.0  //Загрузка в AR2 32-битового указателя.
```

9.9 T Передача

Формат

T <адрес>

Адрес	Тип данных	Область	Адрес
<адрес>	BYTE	I, Q, PQ, M, L, D	0...65535
	WORD		0...65534
	DWORD		0...65532

Описание

Инструкция **T <адрес>** копирует содержимое ACCU 1 в целевую область, если главное управляющее реле включено (MCR = 1). При MCR = 0, в целевую область записывается 0. Количество байт, копируемых из ACCU 1, зависит от ширины доступа , указанной в целевом адресе инструкции. ACCU 1 не изменяется после выполнения инструкции. При передаче прямым доступом к периферии (PQ-область) также передает содержимое ACCU 1 или "0" (при MCR=0) в указанный адрес области отображения выходов (Q-область). Инструкция выполняется независимо от слова состояния и не изменяет его.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

STL		Комментарий
T	**QB10**	**//Передает содержимое ACCU 1-L-L в выходной байт QB10.**
T	**MW14**	**// Передает содержимое ACCU 1-L в меркерное слово MW14.**
T	**DBD2**	**// Передает содержимое ACCU 1 в двойное слово данных DBD2.**

9.10 T STW Передача ACCU 1 в слово состояния

Формат

T STW

Описание

Инструкция **T STW** передает биты с 0 по 8 из ACCU 1 в слово состояния.

Инструкция выполняется независимо от слова состояния и не изменяет его.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	x	x	x	x	x	x	x	x	x

Пример

STL		Комментарий
T	STW	//Передача бит с 0 по 8 из ACCU 1 в слово состояния.

Следующие биты слова состояния изменяются после передачи значения ACCU 1:

Бит	31-9	8	7	6	5	4	3	2	1	0
Содержимое:	*)	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC

*) биты не передаются.

9.11 CAR Обмен содержимым адресных регистров 1 и 2

Формат

CAR

Описание

Инструкция **CAR** меняет местами содержимое адресных регистров AR1 и AR2. Инструкция выполняется независимо от слова состояния и не изменяет его.

Содержимое адресного регистра AR1 передается в адресный регистр AR2 и содержимое адресного регистра AR2 передается в адресный регистр AR1.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

9.12 TAR1 Передача адресного регистра AR1 в ACCU 1

Формат

TAR1

Описание

Инструкция **TAR1** передает содержимое адресного регистра AR1 в ACCU 1 (32-битовый указатель). Предварительно содержимое ACCU 1 сохраняется в ACCU 2. Инструкция выполняется независимо от слова состояния и не изменяет его.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

9.13 TAR1 <D> Передача адресного регистра AR1 в целевую область (32-битовый указатель)

Формат

TAR1 <D>

Адрес	Тип данных	Область	Адрес
<D>	DWORD	D, M, L	0...65532

Описание

TAR1 <D> передает содержимое адресного регистра AR1 в указанную целевую область <D>. Возможной целевой областью могут быть : меркерные двойные слова (MD), локальные данные (LD), двойные слова блоков данных (DBD) и экземплярных блоков данных (DID).

ACCU 1 и ACCU 2 не изменяются. Инструкция выполняется независимо от слова состояния и не изменяет его.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

STL		Комментарий
TAR1	**DBD20**	**//Передача содержимого AR1 в двойное слово данных DBD20.**
TAR1	**DID30**	**//Передача содержимого AR1 в двойное слово данных DID30.**
TAR1	**LD18**	**//Передача содержимого AR1 в локальное двойное слово LD18.**
TAR1	**MD24**	**//Передача содержимого AR1 в меркерное двойное слово MD24.**

9.14 TAR1 AR2 Передача адресного регистра AR 1 в AR 2

Формат

TAR1 AR2

Описание

Инструкция **TAR1 AR2** передает содержимое адресного регистра AR1 в адресный регистр AR2.

ACCU 1 и ACCU 2 не изменяются. Инструкция выполняется независимо от слова состояния и не изменяет его.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

9.15 TAR2 Передача адресного регистра 2 в ACCU 1

Формат

TAR2

Описание

Инструкция **TAR2** передает содержимое адресного регистра AR2 в ACCU 1 (32-битовый указатель). Предварительно содержимое ACCU 1 сохраняется в ACCU 2. Инструкция выполняется независимо от слова состояния и не изменяет его.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

9.16 TAR2 <D> Передача адресного регистра AR 2 в целевую область (32-битовый указатель)

Формат

TAR2 <D>

Адрес	Тип данных	Область	Адрес
<D>	DWORD	D, M, L	0...65532

Описание

TAR2 <D> передает содержимое адресного регистра AR2 в указанную целевую область <D>. Возможной целевой областью могут быть : меркерные двойные слова (MD), локальные данные (LD), двойные слова блоков данных (DBD) и экземплярных блоков данных (DID).

ACCU 1 и ACCU 2 не изменяются. Инструкция выполняется независимо от слова состояния и не изменяет его.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

STL		Комментарий
TAR2	**DBD20**	**//Передача содержимого AR1 в двойное слово данных DBD20.**
TAR2	**DID30**	**//Передача содержимого AR1 в двойное слово данных DID30.**
TAR2	**LD18**	**//Передача содержимого AR1 в локальное двойное слово LD18.**
TAR2	**MD24**	**//Передача содержимого AR1 в меркерное двойное слово MD24.**

10 Инструкции управления программой

10.1 Обзор инструкций управления программой

Описание

Следующие инструкции используются для управления программой:

- BE Конец блока
- BEC Условный конец блока
- BEU Безусловный конец блока
- CALL Вызов блока
- CC Условный вызов блока без параметров
- UC Безусловный вызов блока без параметров

- Вызов FB
- Вызов FC
- Вызов SFB
- Вызов SFC
- Вызов мультиэкземпляра
- Вызов библиотечного блока

- MCR (Главное управляющее реле)
- Важные замечания по работе с MCR функциями
- MCR(Сохранение RLO в стеке MCR, начало MCR зоны
-)MCR Конец MCR зоны
- MCRA Активация MCR области
- MCRD Деактивация MCR области

10.2 BE Конец блока

Формат

BE

Описание

Инструкция **BE** (конец блока) завершает обработку текущего блока и передает выполнение программы в вызывающий блок. Обработка программы продолжается с первой инструкции , которая идет за инструкцией вызова в вызывающем блоке. Вновь становятся актуальными локальные данные вызывающего блока. Также открываются блоки данных, которые были открыты на момент вызова. В дополнение, восстанавливается MCR – зона в вызывающем блоке и RLO переносится из текущего в вызывающий блок. BE не зависит ни от каких условий. В то же время , если инструкция BE пропускается инструкцией перехода, выполнение программы продолжается с метки перехода.

Инструкция BE не соответствует аналогу в S5 программах. В S7 она равнозначна использованию инструкции BEU.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	0	0	1	-	0

Пример

```
STL                         Комментарий
      A     I 1.0
      JC    NEXT    //Переход на метку NEXT при RLO = 1 (I 1.0 = 1).
      L     IW4     //Продолжение, если переход не выполнен.
      T     IW10
      A     I 6.0
      A     I 6.1
      S     M 12.0
      BE            //Конец блока
NEXT: NOP 0         //Продолжение здесь при выполненном переходе.
```

10.3 BEC Условный конец блока

Формат

BEC

Описание

При RLO = 1 инструкция **BEC** (конец блока) завершает обработку текущего блока и передает выполнение программы в вызывающий блок. Обработка программы продолжается с первой инструкции , которая идет за инструкцией вызова в вызывающем блоке. Вновь становятся актуальными локальные данные вызывающего блока. Также открываются блоки данных, которые были открыты на момент вызова. В дополнение, восстанавливается MCR – зона в вызывающем блоке .

RLO (= 1) передается из текущего в вызывающий блок. При RLO = 0 инструкция BEC не выполняется. RLO взводится в 1 и выполнение программы продолжается со следующей за инструкцией BEC команды.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	x	0	1	1	0

Пример

```
STL             Комментарий
A   I 1.0       //Формирование RLO.
BE              //Конец блока при RLO = 1.
C
L   IW4         //Продолжение здесь, если инструкция BEC не выполнилась, RLO = 0.
T   MW10
```

10.4 BEU Безусловный конец блока

Формат

BEU

Описание

Инструкция **BEU** (безусловный конец блока) завершает обработку текущего блока и передает выполнение программы в вызывающий блок. Обработка программы продолжается с первой инструкции , которая идет за инструкцией вызова в вызывающем блоке. Вновь становятся актуальными локальные данные вызывающего блока. Также открываются блоки данных, которые были открыты на момент вызова. В дополнение, восстанавливается MCR – зона в вызывающем блоке и RLO переносится из текущего в вызывающий блок. BEU не зависит ни от каких условий. В то же время , если инструкция BEU пропускается инструкцией перехода, выполнение программы продолжается с метки перехода.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	0	0	1	-	0

Пример

```
STL                     Комментарий
      A     I 1.0
      JC    NEXT      //Переход на метку NEXT при RLO = 1 (I 1.0 = 1).
      L     IW4       //Продолжение здесь, если переход не выполнен.
      T     IW10
      A     I 6.0
      A     I 6.1
      S     M 12.0
      BEU             //Безусловный конец блока.
NEXT: NOP 0           //Продолжение здесь после выполненного перехода.
```

10.5 CALL Вызов блока

Формат

CALL <Номер логического блока>

Описание

Инструкция **CALL < Номер логического блока >** используется для вызова функций (FC) или функциональных блоков (FB), системных функций (SFC) или системных функциональных блоков (SFB) или стандартных блоков, поставляемых фирмой Siemens. Инструкция CALL вызывает FC и SFC или FB и SFB , указанные в качестве адреса , независимо от RLO или любых других условий. Если Вы вызываете FB или SFB с помощью инструкции CALL, Вы должны указать соответствующий номер экземплярного блока DB. Вызывающий блок продолжает обработку после окончания выполнения вызываемого блока. Вызываемый блок может быть задан абсолютно или символьно. Содержимое регистров восстанавливается после вызова SFB/SFC.

Пример: CALL FB1, DB1 или CALL FILLVAT1, RECIPE1

Логический блок	**Тип блока**	**Вызов с абсолютным адресом**
FC	Функция	CALL FCn
SFC	Системная функция	CALL SFCn
FB	Функциональный блок	CALL FBn1,DBn2
SFB	Системный функциональный блок	CALL SFBn1,DBn2

Примечание

При использовании STL редактора, соответствующие номера блоков (n, n1, и n2) , указанные в таблице выше должны быть созданы заранее. Также, как и символьные имена должны быть заранее описаны в символьной таблице.

Передача параметров (в инкрементном редакторе)

Вызываемый блок может обмениваться информацией с вызывающим блоком через формальные параметры, определенные в списке описаний (деклараций) переменных вызываемого блока. Список переменных будет запрашиваться автоматически в Вашей STL программе после вызова блока с помощью инструкции CALL.

Если Вы вызываете FB, SFB, FC или SFC и блок содержит список деклараций переменных типа IN, OUT и IN_OUT, эти переменные будут выведены в качестве формальных параметров для присвоения им фактических значений или адресов при вызове блока.

При вызове FC и SFC, Вы обязательно должны задать актуальные параметры для всех формальных параметров.

При вызове FB и SFB, Вы можете задавать только фактические параметры, которые должны отличаться от предыдущего вызова этого блока . В процессе обработки FB, полученные значения сохраняются в экземплярном DB. Если в качестве фактических параметров используется компонент блока данных, то должен указываться полный абсолютный адрес. Пример: DB1.DBW2.

Входной параметр типа IN может быть задан в виде константы, или области с абсолютным или символьным адресом. Параметры OUT и IN_OUT в виде области с абсолютным или символьным адресом. Вы должны задавать в качестве фактических параметров только области памяти или константы с типом, соответствующим типу данных формальных параметров.

Инструкция CALL сохраняет точку возврата (регистры и относительные адреса), регистры блоков данных , а также бит MA в стеке блоков B . В дополнение, инструкция CALL деактивирует MCR (главное управляющее реле) и резервирует область данных в локальном стеке для временных переменных вызываемого блока.

Биты слова состояния

	BR	**CC 1**	**CC 0**	**OV**	**OS**	**OR**	**STA**	**RLO**	**/FC**
Записывает:	-	-	-	-	0	0	1	-	0

Пример 1: Назначение параметров для вызова FC6

```
CALL    FC6
        Формальный параметр     Фактический параметр
        NO OF TOOL              := MW100
        TIME OUT                := MW110
        FOUND                   := Q 0.1
        ERROR                   := Q 100.0
```

Пример 2: Вызов SFC без параметров

```
STL             Комментарий
CALL    SFC43   //Вызов SFC43 для перезапуска контроля времени цикла (без параметров).
```

Пример 3: Вызов FB99 с блоком экземпляром DB1

```
CALL    FB99,DB1
        Формальный параметр     Фактический параметр
        MAX_RPM                 := #RPM1_MAX
        MIN_RPM                 := #RPM1
        MAX_POWER               := #POWER1
        MAX_TEMP                := #TEMP1
```

Пример 4: Вызов FB99 с блоком экземпляром DB2

```
CALL    FB99,DB2
        Формальный параметр     Фактический параметр
        MAX_RPM                 := #RPM2_MAX
        MIN_RPM                 := #RPM2
        MAX_POWER               := #POWER2
        MAX_TEMP                := #TEMP2
```

Примечание

Каждый вызов FB или SFB должен содержать ссылку на экземплярный блок данных. В примере выше, блоки данных DB1 и DB2 уже должны быть созданы до обращения к ним.

10.6 Вызов FB

Формат

CALL FB n1, DB n1

Описание

Эта инструкция используется для вызова пользовательских функциональных блоков (FB). Инструкция CALL вызывает FB, указанный в качестве адреса независимо от RLO или любых других условий. Если Вы вызываете FB с помощью инструкции CALL, Вы должны указать соответствующий номер экземплярного блока DB. Вызывающий блок продолжает обработку после окончания выполнения вызываемого блока. Вызываемый блок может быть задан в инструкции абсолютно или символьно.

Передача параметров (в инкрементном редакторе)

Вызываемый блок может обмениваться информацией с вызывающим блоком через формальные параметры, определенные в списке описаний (деклараций) переменных вызываемого блока. Список переменных будет запрашиваться автоматически в Вашей STL программе после вызова блока с помощью инструкции CALL.

Если Вы вызываете FB и он имеет список деклараций переменных типа IN, OUT и IN_OUT, эти переменные будут выведены в качестве формальных параметров для присвоения им фактических значений или адресов при вызове блока.

При вызове FB , Вы можете задавать только фактические параметры, которые должны отличаться от предыдущего вызова этого блока . В процессе обработки FB, полученные значения сохраняются в экземплярном DB. Если в качестве фактических параметров используется компонент блока данных, то должен указываться полный абсолютный адрес. Пример: DB1.DBW2.

Входной параметр типа IN может быть задан в виде константы, или области с абсолютным или символьным адресом. Параметры OUT и IN_OUT в виде области с абсолютным или символьным адресом. Вы должны задавать в качестве фактических параметров только области памяти или константы с типом, соответствующим типу данных формальных параметров.

Инструкция CALL сохраняет точку возврата (регистры и относительные адреса), регистры блоков данных , а также бит MA в стеке блоков B . В дополнение, инструкция CALL деактивирует MCR (главное управляющее реле) и резервирует область данных в локальном стеке для временных переменных вызываемого блока.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	0	0	1	-	0

Пример 1: Вызов блока FB99 с экземплярным блоком DB1

```
CALL    FB99,DB1
        Формальный параметр     Фактический параметр
        MAX_RPM                 := #RPM1_MAX
        MIN_RPM                 := #RPM1
        MAX_POWER               := #POWER1
        MAX_TEMP                := #TEMP1
```

Пример 2: Вызов блока FB99 с экземплярным блокомDB2

```
CALL    FB99,DB2
        Формальный параметр     Фактический параметр
        MAX_RPM                 := #RPM2_MAX
        MIN_RPM                 := #RPM2
        MAX_POWER               := #POWER2
        MAX_TEMP                := #TEMP2
```

Примечание

Каждый вызов FB должен содержать ссылку на экземплярный блок данных. В примере выше, блоки данных DB1 и DB2 уже должны быть созданы до обращения к ним.

10.7 Вызов FC

Формат

CALL FC n

Примечание

При работе в STL редакторе, каждый вызов блока должен производиться к уже существующему блоку. Для использования символьного имени, это имя должно быть предварительно описано в таблице символов.

Описание

Эта инструкция используется для вызова функций (FC). Инструкция CALL вызывает FC, указанную в качестве адреса независимо от RLO или любых других условий. Вызывающий блок продолжает обработку после окончания выполнения вызываемого блока. Вызываемый блок может быть задан в инструкции абсолютно или символьно.

Передача параметров (в инкрементном редакторе)

Вызываемый блок может обмениваться информацией с вызывающим блоком через формальные параметры, определенные в списке описаний (деклараций) переменных вызываемого блока. Список переменных будет запрашиваться автоматически в Вашей STL программе после вызова блока с помощью инструкции CALL.

Если Вы вызываете FC и она имеет список описаний переменных типа IN, OUT и IN_OUT, эти переменные будут выведены в качестве формальных параметров для присвоения им фактических значений или адресов при вызове блока.

При вызове FC , Вы должны задавать все фактические параметры для каждого формального параметра.

Входной параметр типа IN может быть задан в виде константы, или области с абсолютным или символьным адресом. Параметры OUT и IN_OUT в виде области с абсолютным или символьным адресом. Вы должны задавать в качестве фактических параметров только области памяти или константы с типом, соответствующим типу данных формальных параметров.

Инструкция CALL сохраняет точку возврата (регистры и относительные адреса), регистры блоков данных , а также бит MA в стеке блоков B . B дополнение, инструкция CALL деактивирует MCR (главное управляющее реле) и резервирует область данных в локальном стеке для временных переменных вызываемого блока.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	0	0	1	-	0

Пример: Назначение параметров при вызове FC6

CALL	FC6	
	Формальный параметр	**Фактический параметр**
	NO OF TOOL	**:= MW100**
	TIME OUT	**:= MW110**
	FOUND	**:= Q0.1**
	ERROR	**:= Q100.0**

10.8 Вызов SFB

Формат

CALL SFB n1, DB n2

Описание

Эта инструкция используется для вызова стандартных функциональных блоков (SFB). Инструкция CALL вызывает SFB, указанный в качестве адреса независимо от RLO или любых других условий. Если Вы вызываете SFB с помощью инструкции CALL, Вы должны указать соответствующий номер экземплярного блока DB. Вызывающий блок продолжает обработку после окончания выполнения вызываемого блока. Вызываемый блок может быть задан в инструкции абсолютно или символьно.

Передача параметров (в инкрементном редакторе)

Вызываемый блок может обмениваться информацией с вызывающим блоком через формальные параметры, определенные в списке описаний (деклараций) переменных вызываемого блока. Список переменных будет запрашиваться автоматически в Вашей STL программе после вызова блока с помощью инструкции CALL.

Если Вы вызываете SFB и он имеет список деклараций переменных типа IN, OUT и IN_OUT, эти переменные будут выведены в качестве формальных параметров для присвоения им фактических значений или адресов при вызове блока.

При вызове SFB, Вы можете задавать только фактические параметры, которые должны отличаться от предыдущего вызова этого блока . В процессе обработки SFB, полученные значения сохраняются в экземплярном DB. Если в качестве фактических параметров используется компонент блока данных, то должен указываться полный абсолютный адрес. Пример: DB1.DBW2.

Входной параметр типа IN может быть задан в виде константы, или области с абсолютным или символьным адресом. Параметры OUT и IN_OUT в виде области с абсолютным или символьным адресом. Вы должны задавать в качестве фактических параметров только области памяти или константы с типом, соответствующим типу данных формальных параметров.

Инструкция CALL сохраняет точку возврата (регистры и относительные адреса), регистры блоков данных , а также бит MA в стеке блоков B . В дополнение, инструкция CALL деактивирует MCR (главное управляющее реле) и резервирует область данных в локальном стеке для временных переменных вызываемого блока

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	0	0	1	-	0

Пример

CALL	SFB4,DB4	
	Формальный параметр	**Фактический параметр**
	IN:	**I0.1**
	PT:	**T#20s**
	Q:	**M0.0**
	ET:	**MW10**

Примечание

Каждый вызов SFB должен содержать ссылку на экземплярный блок данных. В примере выше, блок данных DB4 уже должен быть создан до обращения к нему.

10.9 Вызов SFC

Формат

CALL SFC n

Примечание

При работе в STL редакторе, каждый вызов блока должен производиться к уже ранее созданному блоку. Для использования символьного имени, это имя должно быть предварительно описано в таблице символов.

Описание

Эта инструкция используется для вызова стандартных функций (SFC), разработанных для контроллеров Siemens. Инструкция CALL вызывает SFC, указанную в качестве адреса независимо от RLO или любых других условий. Вызывающий блок продолжает обработку после окончания выполнения вызываемого блока. Вызываемый блок может быть задан в инструкции абсолютно или символьно.

Передача параметров (в инкрементном редакторе)

Вызываемый блок может обмениваться информацией с вызывающим блоком через формальные параметры, определенные в списке описаний (деклараций) переменных вызываемого блока. Список переменных будет запрашиваться автоматически в Вашей STL программе после вызова блока с помощью инструкции CALL. Если Вы вызываете SFC и она имеет список описаний переменных типа IN, OUT и IN_OUT, эти переменные будут выведены в качестве формальных параметров для присвоения им фактических значений или адресов при вызове блока. При вызове SFC , Вы должны задавать все фактические параметры для каждого формального параметра.

Входной параметр типа IN может быть задан в виде константы, или области с абсолютным или символьным адресом. Параметры OUT и IN_OUT в виде области с абсолютным или символьным адресом. Вы должны задавать в качестве фактических параметров только области памяти или константы с типом, соответствующим типу данных формальных параметров.

Инструкция CALL сохраняет точку возврата (регистры и относительные адреса), регистры блоков данных , а также бит MA в стеке блоков B . B дополнение, инструкция CALL деактивирует MCR (главное управляющее реле) и резервирует область данных в локальном стеке для временных переменных вызываемого блока.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	0	0	1	-	0

Пример: Вызов SFC без параметров

```
STL             Комментарий
CALL   SFC43    //Вызов SFC43 для перезапуска котроля времени цикла (без параметров).
```

10.10 Вызов мультиэкземпляра

Формат

CALL # variable name

Описание

Мультиэкземпляр создается с помощью задания статической переменной с типом данных “FB”. В каталоге программных элементов, в папке мультиэкземпляров Вы можете видеть только ранее созданные мультиэкземпляры.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	0	0	x	x	x

10.11 Вызов библиотечного блока

Библиотеки могут использоваться в SIMATIC Manager для вызова блока:

- Встроенного в операционную систему Вашего CPU ("Standard Library")
- Сохраненного в Вашей библиотеке для повторного использования.

10.12 CC Условный вызов блока без параметров

Формат

CC <logic block identifier>

Описание

Инструкция **CC <logic block identifier>:** (условный вызов блока) вызывает блок при RLO=1. Инструкция CC для вызова логических блоков типа FC или FB без параметров. Эта инструкция используется таким же образом, как и инструкция **CALL** за исключением возможности передавать параметры в вызываемый блок. Инструкция сохраняет точку возврата (регистры и относительные адреса), регистры блоков данных , а также бит MA в стеке блоков . В дополнение, инструкция CC деактивирует MCR (главное управляющее реле), резервирует область данных в локальном стеке для временных переменных вызываемого блока, после чего начинается выполнение вызванного блока. Адрес вызываемого блока может задаваться абсолютно или символьно.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	0	0	1	1	0

Пример

STL		Комментарий
A	**I 2.0**	**//Опрос статуса сигнала на входе I 2.0.**
CC	**FC6**	**//Вызов функции при I 2.0 = "1".**
A	**M 3.0**	**//Выполнение продолжается после окончания обработки функции FC6 (I 2.0 = 1) или сразу после опроса входа A I 2.0 при статусе I 2.0 = 0.**

Примечание

Каждый вызов FB или SFB должен содержать в инструкции ссылку на экземплярный блок данных. Для вызова с помощью инструкции CC, Вы не можете в ней задавать блок-экземпляр.

В зависимости от созданного сегмента, редактор может при переходе с языка контактного плана на язык программирования STL использовать как инструкцию **UC** , так и **CC**. Вы должны , по-возможности, использовать в своих программах инструкцию **CALL** для предотвращения возникновения ошибок.

10.13 UC Безусловный вызов блока без параметров

Формат

UC <logic block identifier>

Описание

Инструкция **UC <logic block identifier>:** (безусловный вызов блока) вызывает блок типа FC или SFC без параметров. Эта инструкция используется таким же образом, как и инструкция **CALL** за исключением возможности передавать параметры в вызываемый блок. Инструкция сохраняет точку возврата (регистры и относительные адреса), регистры блоков данных , а также бит MA в стеке блоков . В дополнение, инструкция CC деактивирует MCR (главное управляющее реле), резервирует область данных в локальном стеке для временных переменных вызываемого блока, после чего начинается выполнение вызванного блока.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	0	0	1	-	0

Пример 1

```
STL          Комментарий
UC   FC6     //Вызов функции FC6 (без параметров).
```

Пример 2

```
STL          Комментарий
UC   SFC43   //Вызов системной функции SFC43 (без параметров).
```

Примечание

Каждый вызов с помощью инструкции CALL функционального блока FB или SFB должен содержать в инструкции ссылку на экземплярный блок данных. Для вызова с помощью инструкции UC, Вы не можете в ней задавать блок-экземпляр.

В зависимости от созданного сегмента, редактор может при переходе с языка контактного плана на язык программирования STL использовать как инструкцию UC , так и CC. Вы должны , по-возможности, использовать в своих программах инструкцию CALL для предотвращения возникновения ошибок

10.14 MCR Главное управляющее реле

Важные замечания по использованию MCR функций

Предупреждение

Для предотвращения повреждения оборудования, не используйте MCR функции для замены аппаратного механического управляющего реле в устройствах аварийной остановки.

Описание

Master Control Relay (MCR) - главное управляющее реле – это управляющий переключатель для активации или отключения протекания логического потока через релейно-контактную логическую схему. Выполнение инструкций битовой логики и передачи зависит от MCR:

- **=** <bit>
- **S** <bit>
- **R** <bit>
- **T** <byte>, **T** <word>, **T** <double word>

Инструкция **T,** используемая с байтом, словом или двойным словом, записывает 0 в целевую область при MCR = 0. Инструкции **S** и **R** оставляют указанные в инструкциях операнды неизменными. Инструкция **“=”** записывает "0" в адресуемый бит.

Инструкции , зависящие от MCR и их реакция на состояние сигнала в MCR

Статус MCR	**= <bit>**	**S <bit>, R <bit>**	**T <byte>, T <word> T <double word>**
0 ("OFF")	Записывает 0. (Подобно реле, отключившемуся после снятия напряжения)	Не работают. (Подобно реле, оставшемуся в своем прежнем состоянии после снятия напряжения.)	Записывает 0. (Подобно элементу, выдающему значение “0” после снятия напряжения)
1 ("ON")	Нормальная работа	Нормальная работа	Нормальная работа

MCR(- Начало MCR зоны,)MCR - Конец MCR зоны

Функция MCR управляет стеком шириной в 1 бит и глубиной 8. MCR активирована когда все восемь вводов в стек равны 1. Инструкция **MCR(** копирует бит RLO в MCR стек. Инструкция **)MCR** удаляет последний ввод из стека и устанавливает последнюю позицию в 1. Инструкции **MCR(** и **)MCR** всегда работают в паре. Ошибка происходит в случае , если используется больше восьми незакрытых инструкций **MCR(** или при использовании инструкции **)MCR** при пустом MCR стеке, это приводит к возникновению MCRF ошибки.

MCRA - Активация MCR области, MCRD - Деактивация MCR области

Инструкции MCRA и MCRD всегда работают в паре. Инструкции запрограммированные между MCRA и MCRD зависят от статуса MCR бита. Инструкции , запрограммированные за пределами MCRA-MCRD области не зависят от статуса бита MCR.

Вы должны программировать зависимость функций (FC) и функциональных блоков (FB) от MCR в самих блоках с использованием MCRA инструкции в вызванном блоке.

10.15 Важные замечания по использованию MCR функций

Будьте внимательны с блоками, в которых функция Master Control Relay активируется с помощью MCRA

- При деактивации MCR , значение 0 записывается всеми инструкциями присвоения и передачи в сегментах программы между **MCR(** и **)MCR**.
- Функция MCR деактивируется при RLO = 0 перед инструкцией **MCR(** .

Внимание: PLC в режиме STOP или в неопределенном состоянии!

Компилятор использует доступ на чтение к локальным данным , описанным как временные переменные VAR_TEMP для вычисления адреса. Это означает что следующие команды могут перевести PLC в STOP или привести к неопределенному рабочему состоянию:

Доступ к формальным параметрам

- Доступ к параметрам FC сложного типа STRUCT, UDT, ARRAY, STRING
- Доступ к параметрам FB сложного типа : STRUCT, UDT, ARRAY, STRING из области IN_OUT в блоке версии 2.
- Доступ к формальным параметрам функционального блока версии 2 если его адрес больше чем 8180.0.
- Доступ в функциональном блоке версии 2 к формальным параметрам типа BLOCK_DB, открывающим DB0. Некоторые обращения на доступ к данным , также могут перевести CPU в STOP, например: T 0, C 0, FC0, или FB0 для параметров типа TIMER, COUNTER, BLOCK_FC, и BLOCK_FB.

Передача параметров

- Вызов с передачей актуальных параметров.

LAD/FBD

- T образные ветвления и коннекторы в KOP или FBD языках начинаются с RLO = 0.

Способ

Устранение зависимости вышеописанных инструкций от MCR:

1. **Деактивируйте** Master Control Relay с использованием функции Master Control Relay Deactivate (деактивации главного управляющего реле) **до** выполнения критичной инструкции в сегменте.
2. **Активируйте** Master Control Relay **снова** , используя инструкцию Master Control Relay Activate (активации главного управляющего реле) после выполнения критичных инструкций.

10.16 MCR(Сохранение RLO в MCR стеке, начало MCR

Важные замечания по использованию MCR функций

Формат

MCR(

Описание

MCR((открыть MCR зону) запускает операцию, которая сохраняет RLO в стеке MCR и открывает зону действия MCR. В зону MCR включены все инструкции между **MCR(** и соответствующей ей инструкцией **)MCR**. Инструкция **MCR(** всегда должна использоваться в комбинации с инструкцией **)MCR**.

При RLO=1, функция MCR “включена." Все MCR-зависимые инструкции внутри этой MCR зоны функционируют нормально.

При RLO=0, функция MCR “выключена."

Все MCR-зависимые инструкции внутри этой MCR зоны функционируют в соответствии с нижеприведенной таблицей:

Инструкции, зависимые от состояния MCR бита

Статус бита MCR	**= <bit>**	**S <bit>, R <bit>**	**T <byte>, T <word> T <double word>**
0 ("OFF")	Записывает 0. (Подобно реле, отключившемуся после снятия напряжения)	Не работают. (Подобно реле, оставшемуся в своем прежнем состоянии после снятия напряжения.)	Записывает 0. (Подобно элементу, выдающему значение “0” после снятия напряжения)
1 ("ON")	Нормальная работа	Нормальная работа	Нормальная работа

Инструкции **MCR(** и **)MCR** могут иметь вложения. Максимальная глубина вложения - до 8 инструкций. Максимальная глубина стека MCR- восемь. Выполнение инструкции **MCR(** при заполненном стеке вызывает ошибку MCR Stack Fault (MCRF).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	1	-	0

Пример

```
STL                 Комментарий
MCRA                //Активация  MCR области.
A       I 1.0
MCR(                //Сохранение RLO в стеке  MCR, открытие MCR зоны. MCR = "on" при RLO=1 (I 1.0
                    ="1"); MCR = "off" при RLO=0 (I 1.0 ="0")
A       I 4.0
=       Q 8.0       //Если MCR = "off", тогда бит Q 8.0 имеет статус "0" независимо от I 4.0.
L       MW20
T       QW10        //Если MCR = "off", тогда "0" передается в QW10.
)MCR                //Закрытие  MCR зоны.
MCRD                //Деактивация MCR области.
A       I 1.1
=       Q 8.1       //Эти инструкции за пределами MCR зоны  и не зависят от MCR бита.
```

10.17)MCR Конец MCR зоны

Важные замечания по использованию MCR функций

Формат

)MCR

Описание

Инструкция **)MCR** (закрыть MCR зону) удаляет последний ввод в MCR стек и заканчивает MCR зону. Последнее значение MCR стека при выводе устанавливает стек на 1. Инструкция **MCR(** всегда используется в комбинации с инструкцией **)MCR**. Выполнение инструкции **)MCR** при пустом стеке приводит к возникновению ошибки MCR Stack Fault (MCRF).

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	1	-	0

Пример

```
STL             Комментарий
MCRA            //Активация  MCR области
A       I 1.0
MCR(            //Сохранение RLO в MCR стеке; открытие  MCR зоны. MCR = "on" при RLO=1
                (I 1.0 ="1"); MCR = "off"  при RLO=0 (I 1.0 ="0").
A       I 4.0
=       Q 8.0   //Если MCR = "off", то выход Q 8.0 сбрасывается в "0" независимо от I 4.0.
L       MW20
T       QW10    //Если MCR = "off", тогда "0" передается в QW10.
)MCR            //Закрытие MCR зоны.
MCRD            //Деактивация  MCR зоны.
A       I 1.1
=       Q 8.1   //Эти инструкции находятся за пределами  MCR зоны и не зависят от бита MCR .
```

10.18 MCRA Активация MCR области

Важные замечания по использованию MCR функций

Формат

MCRA

Описание

MCRA (Активация главного управляющего реле) активирует MCR зависимость для следующих за ней инструкций. Инструкция MCRA должна всегда использоваться в комбинации с инструкцией MCRD (Деактивация главного управляющего реле). Инструкции записанные между MCRA и MCRD будут выполняться в зависимости от статуса MCR бита.

Инструкции выполняются независимо от слова состояния и не меняют его.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL             Комментарий
MCRA            //Активация  MCR области
A       I 1.0
MCR(            //Сохранение RLO в MCR стеке; открытие  MCR зоны. MCR = "on" при RLO=1
                (I 1.0 ="1"); MCR = "off"  при RLO=0 (I 1.0 ="0").
A       I 4.0
=       Q 8.0   //Если MCR = "off", то выход Q 8.0 сбрасывается в "0" независимо от I 4.0.
L       MW20
T       QW10    //Если MCR = "off", тогда "0" передается в QW10.
)MCR            //Закрытие MCR зоны.
MCRD            //Деактивация  MCR зоны.
A       I 1.1
=       Q 8.1   //Эти инструкции находятся за пределами  MCR зоны и не зависят от бита MCR .
```

10.19 MCRD Деактивация MCR области

Важные замечания по использованию MCR функций

Формат

MCRD

Описание

MCRD (Деактивация главного управляющего реле) деактивирует MCR зависимость для следующих за ней инструкций. Инструкция MCRA должна всегда использоваться в комбинации с инструкцией MCRD (Деактивация главного управляющего реле). Инструкции записанные между MCRA и MCRD будут выполняться в зависимости от статуса MCR бита.

Инструкции выполняются независимо от слова состояния и не меняют его

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL             Комментарий
MCRA            //Активация  MCR области
A       I 1.0
MCR(            //Сохранение RLO в MCR стеке; открытие  MCR зоны. MCR = "on" при RLO=1
                (I 1.0 ="1"); MCR = "off"  при RLO=0 (I 1.0 ="0").
A       I 4.0
=       Q 8.0   //Если MCR = "off", то выход Q 8.0 сбрасывается в "0" независимо от I 4.0.
L       MW20
T       QW10    //Если MCR = "off", тогда "0" передается в QW10.
)MCR            //Закрытие MCR зоны.
MCRD            //Деактивация  MCR зоны.
A       I 1.1
=       Q 8.1   //Эти инструкции находятся за пределами  MCR зоны и не зависят от бита MCR .
```

11 Инструкции сдвига и циклического сдвига

11.1 Инструкции сдвига

11.1.1 Обзор инструкций сдвига

Описание

С помощью инструкций сдвига Вы можете побитно сдвигать содержимое младшего слова аккумулятора или весь аккумулятор (см. Регистры CPU) влево или вправо. Сдвиг на n битов влево умножает содержимое аккумулятора на 2^n; сдвиг на n битов вправо делит содержимое аккумулятора на 2^n. Например, если Вы сдвигаете значение 3 в двоичном коде на 3 бита влево, то получается десятичное значение 24. Если Вы сдвигаете десятичное значение 16 на 2 бита вправо, то в аккумуляторе получается двоичный эквивалент десятичного значения 4.

Число, задаваемое в виде параметра инструкции сдвига или, при его отсутствии, содержимое младшего байта аккумулятора 2, показывает, на сколько битов должен производиться сдвиг. Разряды, освобождающиеся вследствие операции сдвига, заполняются нулями **или** состоянием знакового бита (”0” в случае положительного числа, ”1” в случае отрицательного числа). Бит, сдвигаемый последним, загружается в бит CC1 слова состояния . Биты CC0 и OV сбрасываются в “0”. Вы можете оценить бит CC1 слова состояния с помощью операций перехода. Инструкции сдвига являются безусловными. Это значит, что они выполняются независимо от каких-либо условий. Они не влияют на бит RLO.

Вы можете использовать следующие инструкции сдвига:

- SSI Сдвиг вправо целого числа со знаком
- SSD Сдвиг вправо двойного целого числа со знаком
- SLW Сдвиг слова влево
- SRW Сдвиг слова вправо
- SLD Сдвиг двойного слова влево
- SRD Сдвиг двойного слова вправо

11.1.2 SSI Сдвиг вправо целого числа со знаком (16-бит)

Формат

SSI
SSI <number>

Параметр	Тип данных	Описание
<number>	целое, беззнаковое	Число битовых позиций сдвига от 0 до 15

Описание

SSI (Сдвиг вправо целого числа со знаком) выполняет побитовый сдвиг содержимого младшего слова аккумулятора (ACCU 1- L) вправо. Битовые позиции, освободившиеся после выполнения сдвига, заполняются в соответствии со статусом знакового бита (бит 15). Последний сдвигаемый бит загружается в бит слова состояния CC 1. Число бит , на которые должен выполняться сдвиг, задается в параметре инструкции <number> или с помощью ACCU 2-L-L.

Инструкция **SSI <number>:** побитно сдвигает вправо младшее слово аккумулятора на число бит заданное в параметре <number>. Допустимое значение параметра: от 0 до 15. Биты слова состояния CC 0 и OV сбрасываются в 0 если <number> больше нуля. Если <number> равен 0, инструкция сдвига соответствует нулевой инструкции **NOP**.

Инструкция **SSI:** В случае отсутствия в инструкции параметра <number> число бит для сдвига берется из младшего байта аккумулятора 2 (ACCU 2-L-L). Возможный диапазон значений : от 0 до 255. При числе сдвигаемых бит >16 всегда получаем одинаковый результат (ACCU 1 = 16#0000, CC 1 = 0, или ACCU 1 = 16#FFFF, CC 1 = 1). Биты слова состояния CC 0 и OV сбрасываются в 0 если число бит для сдвига больше нуля. Если это число равно 0, инструкция сдвига соответствует нулевой инструкции **NOP**.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	-	-	-	-	-

Примеры

Содержимое	ACCU1-H				ACCU1-L			
№ бита	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
Перед выполнением инструкции **SSI 6**	0101	1111	0110	0100	1001	1101	0011	1011
После выполнения инструкции **SSI 6**	0101	1111	0110	0100	1111	1110	0111	0100

Пример 1

```
STL          Комментарий
L    MW4     //Загрузка значения ACCU 1.
SRW  6       //Сдвиг бит со знаком ACCU 1 на шесть позиций вправо.
T    MW8     //Сохранение результата в MW8.
```

Пример 2

```
STL         Комментарий
L     +3    //Загрузка значения +3 в ACCU 1.
L     MW20  //Сохранение содержимого ACCU 1 в ACCU 2 и  загрузка значения  MW20 в ACCU 1.
SRW         //Число позиций сдвига в  ACCU 2- L- L =>Сдвиг на три позиции вправо числа со знаком
            в  ACCU 1-L; заполнение освободившихся позиций статусом знакового бита.
JP    NEXT  //Переход на метку  NEXT по последнему сдвинутому биту  (CC 1) = 1.
```

11.1.3 SSD Сдвиг вправо двойного целого числа со знаком (32-бита)

Формате

SSD
SSD <number>

Параметр	Тип данных	Описание
<number>	целое, беззнаковое	Число битовых позиций сдвига от 0 до 32

Описание

SSD (Сдвиг вправо двойного целого числа со знаком) выполняет побитовый сдвиг содержимого аккумулятора 1(ACCU 1) вправо. Битовые позиции, освободившиеся после выполнения сдвига, заполняются в соответствии со статусом знакового бита (бит 31). Последний сдвигаемый бит загружается в бит слова состояния CC 1. Число бит , на которые должен выполняться сдвиг, задается в параметре инструкции <number> или с помощью ACCU 2-L-L..

Инструкция **SSD<number>:** побитно сдвигает вправо содержимое аккумулятора1 на число бит заданное в параметре <number>. Допустимое значение параметра: от 0 до 32. Биты слова состояния CC 0 и OV сбрасываются в 0, если <number> больше нуля. Если <number> равен 0, инструкция сдвига соответствует нулевой инструкции **NOP**.

Инструкция **SSD:** В случае отсутствия в инструкции параметра <number> число бит для сдвига берется из младшего байта аккумулятора 2 (ACCU 2-L-L). Возможный диапазон значений : от 0 до 255. При числе сдвигаемых бит >32 всегда получаем одинаковый результат (ACCU 1 = 16#00000000, CC 1 = 0, или ACCU 1 = 16# FFFFFFFF, CC 1 = 1). Биты слова состояния CC 0 и OV сбрасываются в 0 если число бит для сдвига больше нуля. Если это число равно 0, инструкция сдвига соответствует нулевой инструкции **NOP**

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	-	-	-	-	-

Примеры

Содержимое	**ACCU1-H**				**ACCU1-L**			
№ бита	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
Перед выполнением инструкции **SSD 7**	1000	1111	0110	0100	0101	1101	0011	1011
После выполнения инструкции **SSD 7**	1111	1111	0001	1110	1100	1000	1011	1010

Пример 1

```
STL       Комментарий
L   MD4   //Загрузка значения в ACCU 1.
SSD 7     //Сдвиг содержимого ACCU 1 на семь позиций вправо со знаком.
T   MD8   //Передача  результата в  MD8.
```

Пример 2

```
STL       Комментарий
L   +3    //Загрузка значения +3 в ACCU 1.
L   MD20  //Сохранение содержимого  ACCU 1 в ACCU 2. Загрузка значения  MD20 в ACCU 1.
SSD       //Число позиций сдвига в  ACCU 2- L- L =>Сдвиг на три позиции вправо числа со знаком в
          ACCU 1; заполнение освободившихся позиций статусом знакового бита.
JP  NEXT  //Переход на метку  NEXT по последнему сдвинутому биту  (CC 1) = 1.
```

11.1.4 SLW Сдвиг слова влево (16-бит)

Формат

SLW
SLW <number>

Параметр	Тип данных	Описание
<number>	целое, беззнаковое	Число битовых позиций сдвига от 0 до 15

Описание

SLW (Сдвиг слова влево) выполняет поразрядный сдвиг содержимого младшего слова аккумулятора (ACCU 1- L) влево. Битовые позиции, освободившиеся после выполнения сдвига, заполняются нулями. Последний сдвигаемый бит загружается в бит слова состояния CC 1. Число бит , на которые должен выполняться сдвиг, задается в параметре инструкции <number> или с помощью ACCU 2-L-L.

Инструкция **SLW <number>:** поразрядно сдвигает влево младшее слово аккумулятора на число бит заданное в параметре <number>. Допустимое значение параметра: от 0 до 15. Биты слова состояния CC 0 и OV сбрасываются в 0 если <number> больше нуля. Если <number> равен 0, инструкция сдвига соответствует нулевой инструкции **NOP**.

Инструкция **SLW:** В случае отсутствия в инструкции параметра <number> число бит для сдвига влево берется из младшего байта аккумулятора 2 (ACCU 2-L-L). Возможный диапазон значений : от 0 до 255. При числе сдвигаемых бит >16 всегда получаем одинаковый результат (ACCU 1 = 16#0000, CC 1 = 0, OV=0, CC 1 = 0). Биты слова состояния CC 0 и OV сбрасываются в 0, если число бит для сдвига больше нуля. Если это число равно 0, инструкция сдвига соответствует нулевой инструкции **NOP.**

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	-	-	-	-	-

Примеры

Содержимое	ACCU1-H				ACCU1-L			
№ бита	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
Перед выполнением инструкции **SLW 5**	0101	1111	0110	0100	0101	1101	0011	1011
После выполнения инструкции **SLW 5**	0101	1111	0110	0100	1010	0111	0110	0000

Пример 1

```
STL       Комментарий
L   MW4   //Загрузка значения в ACCU 1.
SLW 5     //Сдвиг содержимого ACCU 1 на 5 разрядов влево.
T   MW8   //Передача результата в MW8.
```

Пример 2

STL		Комментарий
L	**+3**	**//Загрузка значения +3 в ACCU 1.**
L	**MW20**	**//Передача содержимого ACCU 1 в ACCU 2 и загрузка значения MW20 в ACCU 1.**
SLW		**//Число разрядов сдвига – в ACCU 2- L- L =>Сдвигсодержимого ACCU 1-L на 3 разряда влево.**
JP	**NEXT**	**//Переход на метку NEXT если последний сдвигаемый бит (CC 1) = 1.**

11.1.5 SRW Сдвиг вправо слова (16-бит)

Формат

SRW
SRW <number>

Параметр	Тип данных	Описание
<number>	целое, беззнаковое	Число битовых позиций сдвига от 0 до 15

Описание

SRW (Сдвиг слова вправо) выполняет поразрядный сдвиг содержимого младшего слова аккумулятора (ACCU 1- L) вправо. Битовые позиции, освободившиеся после выполнения сдвига, заполняются нулями. Последний сдвигаемый бит загружается в бит слова состояния CC 1. Число бит , на которые должен выполняться сдвиг, задается в параметре инструкции <number> или с помощью ACCU 2-L-L.
Инструкция **SRW <number>:** поразрядно сдвигает вправо младшее слово аккумулятора на число бит заданное в параметре <number>. Допустимое значение параметра: от 0 до 15. Биты слова состояния CC 0 и OV сбрасываются в 0 если <number> больше нуля. Если <number> равен 0, инструкция сдвига соответствует нулевой инструкции **NOP**.

Инструкция **SRW:** В случае отсутствия в инструкции параметра <number> число бит для сдвига влево берется из младшего байта аккумулятора 2 (ACCU 2-L-L). Возможный диапазон значений : от 0 до 255. При числе сдвигаемых бит >16 всегда получаем одинаковый результат (ACCU 1- L = 0, CC 1 = 0, OV=0, CC 1 = 0). Биты слова состояния CC 0 и OV сбрасываются в 0, если число бит для сдвига больше нуля. Если это число равно 0, инструкция сдвига соответствует нулевой инструкции **NOP.**

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	-	-	-	-	-

Примеры

Содержимое	ACCU1-H				ACCU1-L			
№ бита	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
Перед выполнением инструкции **SLW 5**	0101	1111	0110	0100	0101	1101	0011	1011
После выполнения инструкции **SLW 5**	0101	1111	0110	0100	0000	0001	0111	0100

Пример 1

```
STL     Комментарий
L   MW4 //Загрузка value в ACCU 1.
SRW 6   // Сдвиг содержимого ACCU 1 на 6 разрядов вправо.
T   MW8 //Передача результата в MW8.
```

Пример 2

```
STL     Комментарий
L   +3    //Загрузка vчисла +3 в ACCU 1.
L   MW20  //Загрузка содержимого ACCU 1 в ACCU 2. Загрузка значения MW20 в ACCU 1.
SRW       //Число разрядов сдвига – в  ACCU 2- L- L =>Сдвигсодержимого ACCU 1-L  на 3 разряда
          вправо.
SPP NEXT  //Переход на метку NEXT если последний сдвигаемый бит (CC 1) = 1.
```

11.1.6 SLD Сдвиг двойного слова влево (32-бита)

Формате

SLD
SLD <number>

Параметр	Тип данных	Описание
<number>	целое, беззнаковое	Число битовых позиций сдвига от 0 до 32

Описание

SLD (Сдвиг двойного слова влево) выполняет поразрядный сдвиг содержимого аккумулятора1 (ACCU 1) влево. Битовые позиции, освободившиеся после выполнения сдвига, заполняются нулями. Последний сдвигаемый бит загружается в бит слова состояния CC 1. Число бит , на которые должен выполняться сдвиг, задается в параметре инструкции <number> или с помощью ACCU 2-L-L.

Инструкция **SLD <number>:** поразрядно сдвигает влево аккумулятор1 на число бит заданное в параметре <number>. Допустимое значение параметра: от 0 до 32. Биты слова состояния CC 0 и OV сбрасываются в 0 если <number> больше нуля. Если <number> равен 0, инструкция сдвига соответствует нулевой инструкции **NOP**.

Инструкция **SLD :** В случае отсутствия в инструкции параметра <number> число бит для сдвига влево берется из младшего байта аккумулятора 2 (ACCU 2-L-L). Возможный диапазон значений : от 0 до 255. При числе сдвигаемых бит >32 всегда получаем одинаковый результат (ACCU 1 = 0, CC 1 = 0, OV=0, CC 1 = 0). Биты слова состояния CC 0 и OV сбрасываются в 0, если число бит для сдвига больше нуля. Если это число равно 0, инструкция сдвига соответствует нулевой инструкции **NOP.**

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	-	-	-	-	-

Примеры

Содержимое	ACCU1-H				ACCU1-L			
№ бита	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
Перед выполнением инструкции **SLD 5**	0101	1111	0110	0100	0101	1101	0011	1011
После выполнения инструкции **SLD 5**	1110	1100	1000	1011	1010	0111	0110	0000

Пример 1

```
STL       Комментарий
L   MD4   //Загрузка значения в ACCU 1.
SLD 5     //Сдвиг содержимого ACCU 1 на пять разрядов влево.
T   MD8   //Передача результата MD8.
```

Пример 2

```
STL       Комментарий
L   +3    //Загрузка числа +3 в ACCU 1.
L   MD20  //Сдвиг содержимого ACCU 1 в ACCU 2. Загрузка значения MD20 в ACCU 1.
SLD       //Число разрядов сдвига в ACCU 2- L- L =>Сдвиг битов в ACCU 1 на 3 разряда влево.
JP  NEXT  //Переход на метку NEXT при последнем сдвигаемом бите (CC 1) = 1.
```

11.1.7 SRD Сдвиг вправо двойного слова (32-бита)

Формат

SRD
SRD <number>

Параметр	Тип данных	Описание
<number>	целое, беззнаковое	Число битовых позиций сдвига от 0 до 32

Описание

SRD (Сдвиг двойного слова вправо) выполняет поразрядный сдвиг содержимого аккумулятора1 (ACCU 1) вправо. Битовые позиции, освободившиеся после выполнения сдвига, заполняются нулями. Последний сдвигаемый бит загружается в бит слова состояния CC 1. Число бит , на которые должен выполняться сдвиг, задается в параметре инструкции <number> или с помощью ACCU 2-L-L.

Инструкция **SRD <number>:** поразрядно сдвигает вправо аккумулятор1 на число бит заданное в параметре <number>. Допустимое значение параметра: от 0 до 32. Биты слова состояния CC 0 и OV сбрасываются в 0 если <number> больше нуля. Если <number> равен 0, инструкция сдвига соответствует нулевой инструкции **NOP**

Инструкция **SRD :** В случае отсутствия в инструкции параметра <number> число бит для сдвига вправо берется из младшего байта аккумулятора 2 (ACCU 2-L-L). Возможный диапазон значений : от 0 до 255. При числе сдвигаемых бит >32 всегда получаем одинаковый результат (ACCU 1 = 0, CC 1 = 0, OV=0, CC 1 = 0). Биты слова состояния CC 0 и OV сбрасываются в 0, если число бит для сдвига больше нуля. Если это число равно 0, инструкция сдвига соответствует нулевой инструкции **NOP**.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	-	-	-	-	-

Примеры

Содержимое	ACCU1-H				ACCU1-L			
№ бита	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
Перед выполнением инструкции **SRD 7**	0101	1111	0110	0100	0101	1101	0011	1011
После выполнения инструкции **SRD 7**	0000	0000	1011	1110	1100	1000	1011	1010

Пример 1

```
STL       Комментарий
L   MD4   //Загрузка значения в ACCU 1.
SRD 7     //Сдвиг содержимого ACCU 1 на 7 разрядов вправо.
T   MD8   //Передача результата  в  MD8.
```

Пример 2

```
STL       Комментарий
L   +3    //Загрузка числа +3 в ACCU 1.
L   MD20  //Передача содержимого ACCU 1 в ACCU 2. Загрузка значения  MD20 в ACCU 1.
SRD       //Число разрядов сдвига в ACCU 2- L- L =>Сдвиг битов в ACCU 1 на 3 разряда вправо.
JP  NEXT  //Переход на метку NEXT при последнем сдвинутом бите (CC 1) = 1.
```

11.2 Инструкции циклического сдвига

11.2.1 Обзор инструкций циклического сдвига

Описание

С помощью инструкций циклического сдвига, Вы можете поразрядно циклически сдвигать вправо или влево все содержимое аккумулятора 1 (см. также Регистры CPU). Инструкции циклического сдвига выполняют функции подобные инструкциям сдвига, описанным в главе 14.1. Однако, освобождающиеся разряды заполняются состояниями сигналов тех битов, которые выталкиваются из аккумулятора.

Число, указанное в инструкции циклического сдвига или младший байт аккумулятора 2 указывает число сдвигаемых разрядов. В зависимости от выполняемой инструкции, циклический сдвиг выполняется через бит CC 1 слова состояния. Бит CC 0 слова состояния сбрасывается в 0.

Возможны следующие инструкции циклического сдвига:

- RLD Циклический сдвиг двойного слова влево (32-бита)
- RRD Циклический сдвиг двойного слова вправо (32-бита)
- RLDA Циклический сдвиг ACCU1 влево через бит CC1 (32-бита)
- RRDA Циклический сдвиг ACCU1 вправо через бит CC1 (32-бита)

11.2.2 RLD Циклический сдвиг двойного слова влево (32-бита)

Формат

RLD
RLD <number>

Параметр	Тип данных	Описание
<number>	целое, беззнаковое	Число битовых позиций сдвига от 0 до 32

Описание

RLD (Циклический сдвиг двойного слова влево) выполняет поразрядный циклический сдвиг содержимого аккумулятора 1 (ACCU 1) влево. Битовые позиции, освободившиеся после выполнения сдвига, заполняются состояниями сигналов циклически сдвигаемых из ACCU1 битов. Последний сдвигаемый бит загружается в бит слова состояния CC 1. Число бит , на которые должен выполняться сдвиг, задается в параметре инструкции <number> или с помощью ACCU 2-L-L.

Инструкция **RLD <number>:** поразрядно сдвигает влево аккумулятор 1 на число бит заданное в параметре <number>. Допустимое значение параметра: от 0 до 32. Биты слова состояния CC 0 и OV сбрасываются в 0 если <number> больше нуля. Если <number> равен 0, инструкция циклического сдвига соответствует нулевой инструкции **NOP.**

Инструкция **RLD:** В случае отсутствия в инструкции параметра <number> число бит для сдвига вправо берется из младшего байта аккумулятора 2 (ACCU 2-L-L). Возможный диапазон значений : от 0 до 255. При числе сдвигаемых бит >32 всегда получаем одинаковый результат (ACCU 1 = 0, CC 1 = 0, OV=0, CC 1 = 0). Биты слова состояния CC 0 и OV сбрасываются в 0, если число бит для сдвига больше нуля. Если это число равно 0, инструкция циклического сдвига соответствует нулевой инструкции.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	-	-	-	-	-

Примеры

Содержимое	ACCU1-H				ACCU1-L			
№ бита	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
Перед выполнением инструкции **RLD 4**	0101	1111	0110	0100	0101	1101	0011	1011
После выполнения инструкции **RLD 4**	1111	0110	0100	0101	1101	0011	1011	0101

Пример 1

```
STL       Комментарий
L   MD2   //Загрузка числа в ACCU 1.
RLD 4     //Циклический сдвиг ACCU 1 на 4 позиции влево.
T   MD8   //Передать результат в  MD8.
```

Пример 2

```
STL       Комментарий
L   +3    //Загрузка числа  +3 в ACCU 1.
L   MD20  //Перенос содержимого ACCU 1 в ACCU 2. Загрузка содержимого  MD20 в ACCU 1.
RLD       //Число позиций сдвига в ACCU 2- L- L => Циклический сдвиг ACCU 1 на 3 позиции влево.
JP  NEXT  //Переход на метку  NEXT при последнем сдвигаемом бите (CC 1) = 1.
```

11.2.3 RRD Циклический сдвиг двойного слова вправо (32-бита)

Формат

RRD
RRD <number>

Параметр	Тип данных	Описание
<number>	целое, беззнаковое	Число битовых позиций сдвига от 0 до 32

Описание

RRD (Циклический сдвиг двойного слова вправо) выполняет поразрядный циклический сдвиг содержимого аккумулятора 1 (ACCU 1) вправо. Битовые позиции, освободившиеся после выполнения сдвига, заполняются состояниями сигналов циклически сдвигаемых из ACCU1 битов. Последний сдвигаемый бит загружается в бит слова состояния CC 1. Число бит , на которые должен выполняться циклический сдвиг, задается в параметре инструкции <number> или с помощью ACCU 2-L-L.

Инструкция **RRD <number>:** поразрядно сдвигает вправо аккумулятор 1 на число бит заданное в параметре <number>. Допустимое значение параметра: от 0 до 32. Биты слова состояния CC 0 и OV сбрасываются в 0 если <number> больше нуля. Если <number> равен 0, инструкция циклического сдвига соответствует нулевой инструкции **NOP.**

Инструкция **RLD:** В случае отсутствия в инструкции параметра <number> число бит для сдвига вправо берется из младшего байта аккумулятора 2 (ACCU 2-L-L). Возможный диапазон значений : от 0 до 255. Биты слова состояния сбрасываются в 0, если число бит для сдвига больше нуля.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	x	x	-	-	-	-	-

Примеры

Содержимое	ACCU1-H				ACCU1-L			
№ бита	31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
Перед выполнением инструкции **RRD 4**	0101	1111	0110	0100	0101	1101	0011	1011
После выполнения инструкции **RRD 4**	1011	0101	1111	0110	0100	0101	1101	0011

Пример 1

```
STL       Комментарий
L   MD2   //Загрузка числа в ACCU 1.
RRD 4     //Циклический сдвиг ACCU 1 на 4 позиции вправо.
T   MD8   //Передать результат в  MD8.
```

Пример 2

STL		Комментарий
L	**+3**	**//Загрузка числа +3 в ACCU 1.**
L	**MD20**	**//Перенос содержимого ACCU 1 в ACCU 2. Загрузка содержимого MD20 в ACCU 1.**
RRD		**//Число позиций сдвига в ACCU 2- L- L => Циклический сдвиг ACCU 1 на 3 позиции вправо.**
JP	**NEXT**	**//Переход на метку NEXT при последнем сдвигаемом бите (CC 1) = 1.**

11.2.4 RLDA Циклический сдвиг ACCU1(32-бита) влево через бит CC1

Формат

RLDA

Описание

RLDA (Циклический сдвиг двойного слова влево через CC1) выполняет поразрядный циклический сдвиг содержимого аккумулятора1 (ACCU 1) влево на одну позицию через CC1. Биты слова состояния CC 0 и OV обнуляются.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	0	0	-	-	-	-	-

Примеры

Содержимое	CC 1	ACCU1-H				ACCU1-L			
№ бита		31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
Перед выполнением инструкции **RLDA**	**X**	**0**101	1111	0110	0100	0101	1101	0011	1011
После выполнения инструкции **RLDA**	**0**	1011	1110	1100	1000	1011	1010	0111	011**X**
	(X = 0 или 1, предыдущее состояние CC 1)								

STL		Комментарий
L	**MD2**	**//Загрузка содержимого MD2 в ACCU 1.**
RLDA		**//Циклический сдвиг ACCU 1 на 1 позицию влево через CC1 .**
JP	**NEXT**	**//Перейти на метку NEXT, если последний выдвинутый в CC1 бит =1.**

11.2.5 RRDA Циклический сдвиг ACCU1(32-бита) вправо через бит CC1

Формат

RRDA

Описание

RRDA (Циклический сдвиг двойного слова вправо через CC1) выполняет поразрядный циклический сдвиг содержимого аккумулятора 1 (ACCU 1) вправо на одну позицию через CC1. Биты слова состояния CC 0 и OV сбрасываются в 0.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	0	0	-	-	-	-	-

Пример

Содержимое	CC 1	ACCU1-H				ACCU1-L			
№ бита		31 . . .	. .	. .	. . . 16	15 . . .	. .	. .	. . . 0
Перед выполнением инструкции **RRDA**	**X**	0101	1111	0110	0100	0101	1101	0011	101**1**
После выполнения инструкции **RRDA**	**1**	**X**010	1111	1011	0010	0010	1110	1001	1101
	(X = 0 или 1, предыдущее состояние CC 1)								

```
STL          Комментарий
L    MD2     //Загрузка содержимого MD2 в ACCU 1.
RRDA         //Циклический сдвиг ACCU 1 на 1 позицию вправо через CC1 .
JP   NEXT    //Перейти на метку NEXT, если последний выдвинутый  в CC1 бит =1.
```

12 Инструкции с таймерами

12.1 Обзор инструкций с таймерами

Описание

Вы найдете подробную информацию по установке и выбору корректного времени в разделе «Области таймерной памяти и компоненты таймера».

Возможны следующие инструкции с таймерами:

- FR Деблокировка таймера
- L Загрузка текущего значения таймера в ACCU 1 в формате Integer
- LC Загрузка текущего значения таймера в ACCU 1 в BCD - коде
- R Сброс таймера
- SD Таймер задержки включения
- SE Удлиненный импульс
- SF Таймер задержки выключения
- SP Импульс
- SS Таймер задержки включения с памятью

12.2 Области таймерной памяти и компоненты таймера

Область памяти

Таймеры имеют область, зарезервированную для них в памяти Вашего CPU. Эта область памяти резервирует одно 16-битное слово для каждого таймерного адреса. При программировании может адресоваться до 256 таймеров. Для определения точного количества таймеров, Вам необходимо обратиться к руководству на Ваш контроллер. К области памяти таймеров имеют доступ следующие функции:

- Таймерные инструкции
- Актуализация таймерных слов генератором тактовых импульсов. В режиме RUN эта функция CPU уменьшает заданное значение времени на одну единицу с интервалом, установленным 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 временных отрезков. Когда таймер актуализируется, значение времени уменьшается на одну единицу через интервалы, установленные базой времени. Значение времени уменьшается до тех пор, пока оно не станет равным нулю. Вы можете задавать значение времени в двоичном, шестнадцатиричном или двоично-десятичном коде(BCD).

Вы можете загрузить значение времени с использованием следующего синтаксиса:

- W#16#txyz , где

  -t - база времени (временной интервал или разрешение)

  -xyz – значение времени в BCD коде

- S5T#a**H**_b**M**_c**S**_d**MS**

  - где: a = часы, b = минуты, c = секунды и d = миллисекунды
  - База времени выбирается автоматически и значение округляется до ближайшего меньшего числа с этой базой времени.

Максимальное время, которое Вы можете ввести, составляет 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 времени, через который значение времени уменьшается на одну единицу . Минимальная база времени равна 10 мс; максимальная - 10 с.

База времени	**Двоичный код для базы времени**
10 ms	00
100 ms	01
1 s	10
10 s	11

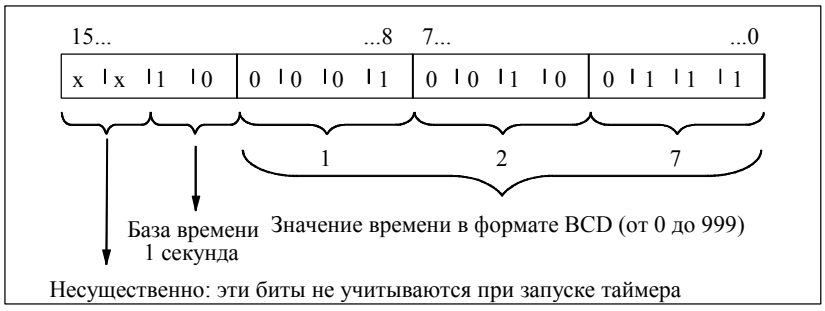
Значения больше 2h46m30s не допускаются. Значения, разрешающая способность которых слишком велика для требуемого диапазона, например 2h10ms, округляются таким образом, что достигается требуемый диапазон, но не желаемая разрешающая способность. Следующая таблица показывает возможные разрешающие способности и соответствующие диапазоны:

Разрешающая способность	**База времени**
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 ACCU 1

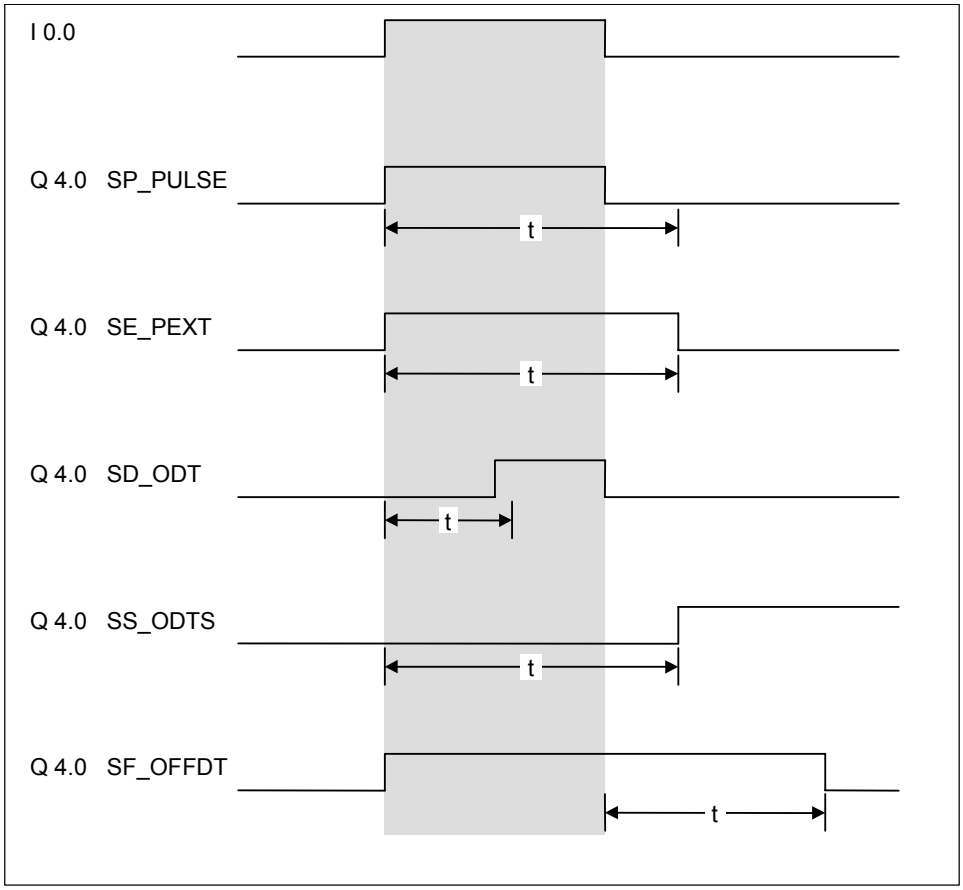
При запуске таймера, содержимое ACCU1 используется в качестве значения времени. Биты с 0 по 11 в ACCU1-L содержат значение времени в двоично-десятичном формате (BCD-формат: каждая группа из четырех битов содержит двоичный код десятичного разряда). Биты 12 и 13 содержат базу времени в двоичном коде .

Следующий рисунок показывает содержимое таймерной ячейки, загруженной значением времени 127 с базой времени 1 секунда:



Выбор подходящего таймера

Этот рисунок поможет Вам выбрать нужный тип таймера для обработки временных событий:



Таймер	Описание
SP_PULSE Таймер «Импульс»	Максимальное время в течение которого выходной сигнал остается равным 1, совпадает с заданным временем t. Выход сбрасывается раньше, если входной сигнал меняется на 0.
SE_PEXT Таймер «Импульс с памятью»	Выходной сигнал остается равным 1 в течение заданного времени независимо от того, как долго остается равным 1 входной сигнал.
SD_ODT Таймер «Задержка включения»	Выходной сигнал устанавливается в 1 только по истечении заданного времени, при этом входной сигнал все еще должен быть равен 1.
SS_ODTS Таймер «Задержка включения с памятью»	Выходной сигнал устанавливается в 1 только по истечении заданного времени независимо от того, как долго остается равным 1 входной сигнал.
SF_OFFDT Таймер «Задержка выключения»	Выходной сигнал устанавливается в 1, когда устанавливается в 1 входной сигнал, и остается равным 1, пока таймер работает. Отсчет времени начинается, когда входной сигнал меняется с 1 на 0.

12.3 FR Деблокировка таймера

Формат

FR <timer>

Параметр	Тип данных	Область памяти	Описание
<timer>	TIMER	T	Номер таймера, диапазон зависит от CPU

Описание инструкции

При переходе RLO из "0" в "1", **FR <timer>** обнуляет флаг выделения фронта, используемый для запуска адресуемого таймера. Изменение RLO с 0 на 1 перед инструкцией деблокировки (FR) запускает таймер.

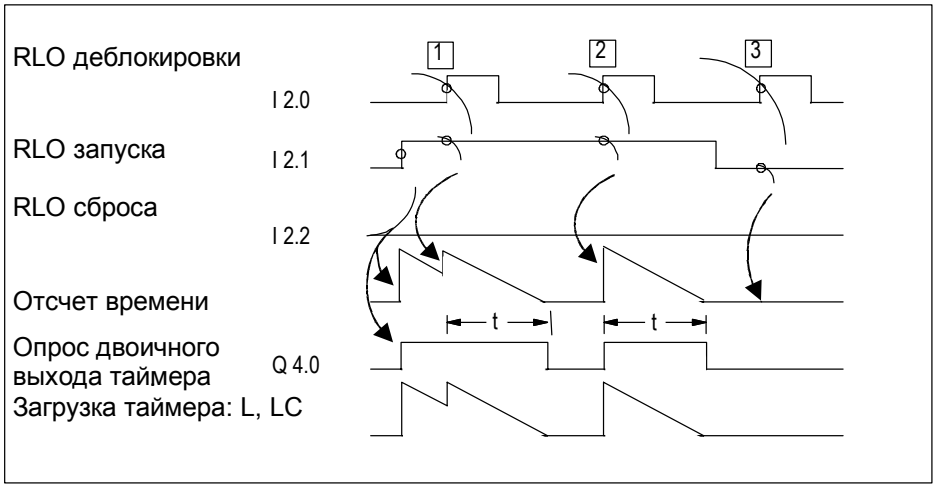
Эта деблокировка не обязательна для нормального запуска таймера с помощью соответствующих инструкций. Деблокировка может быть необходима только для перезапуска работающего таймера. Однако перезапуск возможен только при высоком уровне на входе запуска, т.е. при RLO = 1.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	0	-	-	0

Пример

```
STL          Комментарий
A   I 2.0
FR  T1       //Деблокировка таймера T1.
A   I 2.1
L   S5T#10s  //Задание времени  10 секунд в ACCU 1.
SI  T1       //Запуск T1 как импульс.
A   I 2.2
R   T1       //Сброс  T1.
A   T1       //Оценка состояния таймера T1.
=   Q 4.0
L   T1       //Загрузка текущего значения таймера T1 в двоичном коде.
T   MW10
```



t = заданный интервал времени

(1) Изменение RLO с 0 на 1 на входе деблокировки во время работы таймера производит его перезапуск. При перезапуске происходит переустановка на заданный интервал времени. Изменение RLO с 1 на 0 на входе деблокировки не производит эффекта.

(2) Изменение RLO с 0 на 1 на входе деблокировки при неработающем таймере и RLO = **1** на входе запуска, также производит его перезапуск на заданный интервал времени.

(3) Изменение RLO с 0 на 1 на входе деблокировки при неработающем таймере и RLO = **0** на входе запуска не влияет на состояние таймера.

12.4 L Загрузка текущего значения ACCU 1 в двоичном коде

Формат

L <timer>

Параметр	Тип данных	Область памяти	Описание
<timer>	TIMER	T	Номер таймера, диапазон зависит от CPU

Описание инструкции

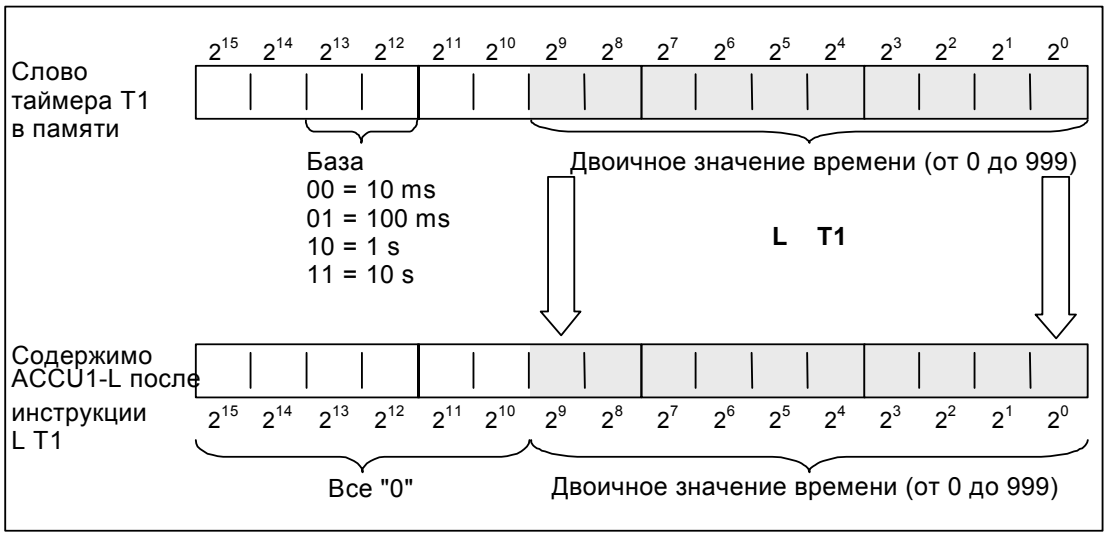
L <timer> загружает текущее значение таймера из указанного таймерного слова без базы времени в формате Integer в ACCU 1-L после предварительного сохранения содержимого ACCU 1 в ACCU 2.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

STL		Комментарий
L	**T1**	**//Загрузка в ACCU 1-L текущего значения времени T1 в двоичном коде.**



Примечание

L <timer> загружает текущее содержимое таймерной ячейки памяти в двоичном коде в ACCU1-L, без базы времени. Загружаемое время равно заданному значению минус отсчитанное время с момента старта таймера.

12.5 LC Загрузка текущего значения таймера в ACCU 1 в BCD-коде

Формат

LC <timer>

Параметр	Тип данных	Область памяти	Описание
<timer>	TIMER	T	Номер таймера, диапазон зависит от CPU

Описание инструкции

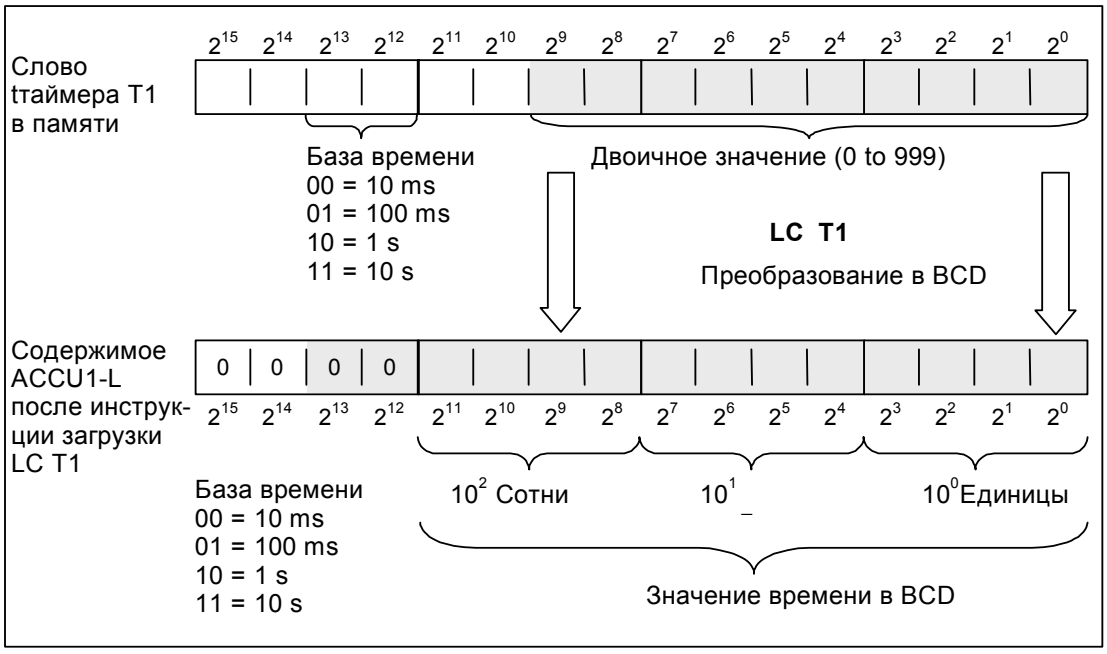
LC <timer> загружает текущее значение времени и базу времени с адресованной таймерной ячейки в двоично-десятичном коде (BCD) в ACCU 1 после предварительного сохранения ACCU 1 в ACCU 2.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

STL		Комментарий
LC	**T1**	**//Загрузка ACCU 1-L значением базы времени и текущего значения таймера T1 в двоично-десятичном коде (BCD).**



12.6 R Сброс таймера

Формат

R <timer>

Параметр	Тип данных	Область памяти	Описание
<timer>	TIMER	T	Номер таймера, диапазон зависит от CPU

Описание инструкции

R <timer> останавливает текущую таймерную функцию и обнуляет значение времени и базу времени в системной памяти при переходе RLO из 0 в 1.

Биты слова состояния

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
Записывает:	-	-	-	-	-	0	-	-	0

Пример

STL		Комментарий
A	**I 2.1**	
R	**T1**	**//Если опрос входа I 2.1 дает переход RLO из 0 в 1, происходит сброс таймера T1.**

12.7 SP Таймер “Импульс”

Формат

SP <timer>

Параметр	Тип данных	Область памяти	Описание
<timer>	TIMER	T	Номер таймера, диапазон зависит от CPU

Описание инструкции

SP <timer> запускает заданный таймер по нарастающему фронту RLO (переход из "0" в "1"). Таймер продолжает работать в течение заданного времени пока RLO = 1. Таймер останавливается, если RLO меняется с 1 на 0 до истечения заданного времени. Таймер запускается в соответствии с временем и временной базой, заданными в младшем слове аккумулятора.

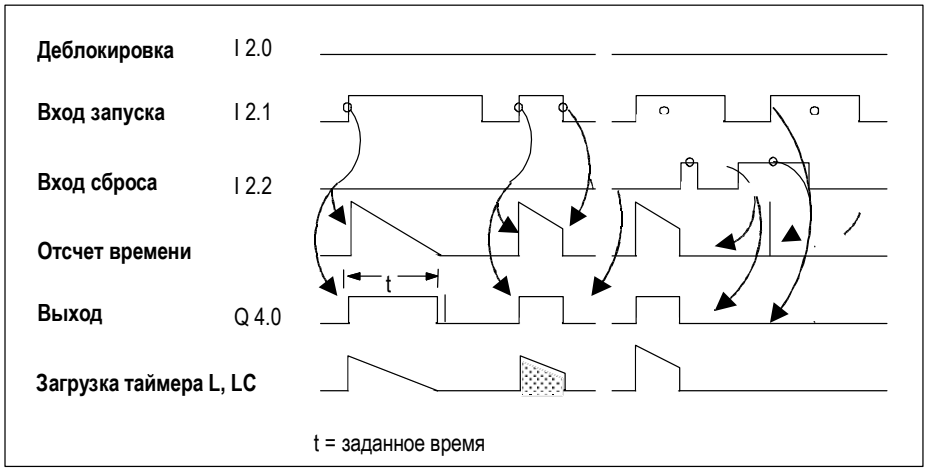
Дополнительную информацию Вы найдете в главе “Расположение таймеров в памяти и компоненты таймера“.

Биты слова состояния

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
Записывает:	-	-	-	-	-	0	-	-	0

Пример

```
STL             Комментарий
A    I 2.0
FR   T1         //Деблокировка таймера T1.
A    I 2.1
L    S5T#10s    //Установка времени 10 секунд в ACCU 1.
SP   T1         //Запуск таймера T1 как “Импульс”.
A    I 2.2
R    T1         //Сброс таймера T1.
A    T1         //Опрос состояния таймера T1.
=    Q 4.0
L    T1         //Загрузка текущего двоичного значения таймера T1.
T    MW10
LC   T1         //Загрузка текущего значения таймера T1 в BCD-коде.
T    MW12
```



12.8 SE Таймер “Удлиненный импульс”

Формат

SE <timer>

Параметр	Тип данных	Область памяти	Описание
<timer>	TIMER	T	Номер таймера, диапазон зависит от CPU

Описание инструкции

SE <timer> запускает заданный таймер по нарастающему фронту RLO (переход из "0" в "1"). Таймер продолжает работать в течение заданного времени, даже если RLO при этом переходит в 0. Отсчет заданного отрезка времени начинается вновь при переходе RLO из 0 в 1 до окончания отсчета времени.Таймер запускается в соответствии с временем и временной базой, заданными в младшем слове аккумулятора.

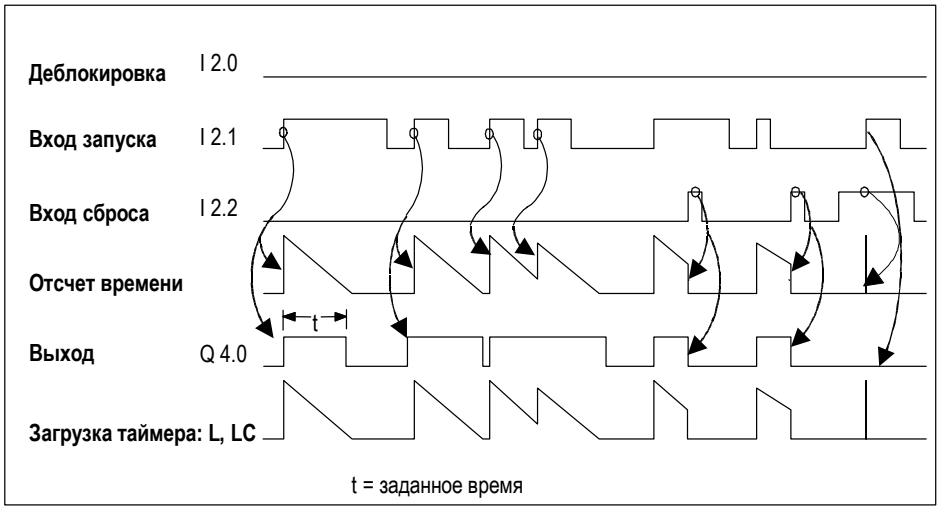
Дополнительную информацию Вы найдете в главе “Расположение таймеров в памяти и компоненты таймера“.

Биты слова состояния

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
Записывает:	-	-	-	-	-	0	-	-	0

Пример

STL		Комментарий
A	**I 2.0**	
FR	**T1**	**//Деблокировка таймера T1.**
A	**I 2.1**	
L	**S5T#10s**	**//Установка времени 10 секунд в ACCU 1.**
SE	**T1**	**//Запуск таймера T1 как “Удлиненный импульс”.**
A	**I 2.2**	
R	**T1**	**//Сброс таймера T1.**
A	**T1**	**//Опрос состояния таймера T1.**
=	**Q 4.0**	
L	**T1**	**//Загрузка текущего двоичного значения таймера T1.**
T	**MW10**	
LC	**T1**	**//Загрузка текущего значения таймера T1 в BCD-коде.**
T	**MW12**	



12.9 SD Таймер “Задержка включения”

Формат

SD <timer>

Параметр	Тип данных	Область памяти	Описание
<timer>	TIMER	T	Номер таймера, диапазон зависит от CPU

Описание инструкции

SD <timer> запускает заданный таймер по нарастающему фронту RLO (переход из "0" в "1"). Таймер продолжает работать в течение заданного времени пока RLO = 1. Таймер останавливается, если RLO меняется с 1 на 0 до истечения заданного времени. Таймер запускается в соответствии с временем и временной базой, заданными в младшем слове аккумулятора.

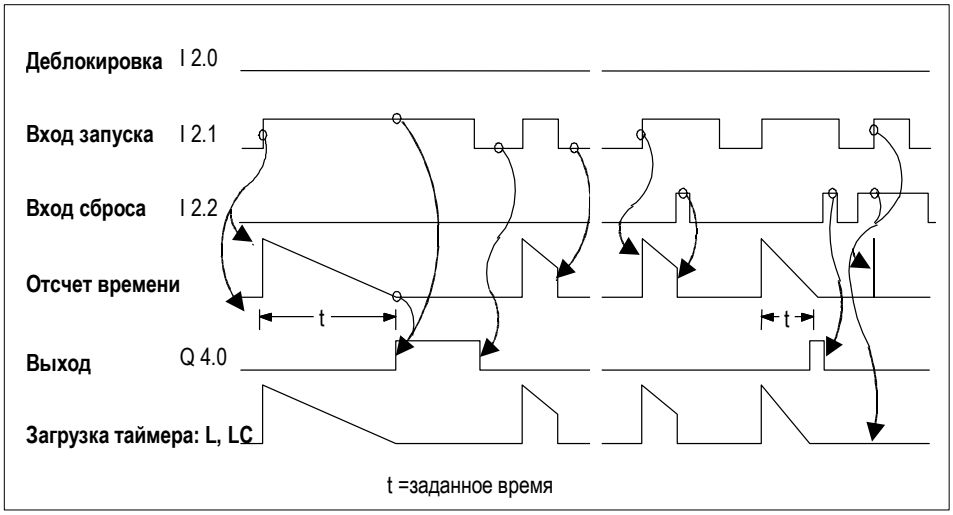
Дополнительную информацию Вы найдете в главе “Расположение таймеров в памяти и компоненты таймера“.

Биты слова состояния

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
Записывает:	-	-	-	-	-	0	-	-	0

Пример

```
STL                 Комментарий
A    I 2.0
FR   T1             //Деблокировка таймера T1.
A    I 2.1
L    S5T#10s        //Установка времени 10 секунд в ACCU 1.
SD   T1             //Запуск таймера T1 как “Задержка включения”.
A    I 2.2
R    T1             //Сброс таймера T1.
A    T1             //Опрос состояния таймера T1.
=    Q 4.0
L    T1             //Загрузка текущего двоичного значения таймера T1.
T    MW10
LC   T1             //Загрузка текущего значения таймера T1 в BCD-коде.
T    MW12
```



12.10 SS Таймер ”Задержка включения с памятью”

Формат

SS <timer>

Параметр	Тип данных	Область памяти	Описание
<timer>	TIMER	T	Номер таймера, диапазон зависит от CPU

Описание инструкции

SS <timer> (задержка включения с памятью) запускает заданный таймер по нарастающему фронту RLO (переход из "0" в "1"). Таймер продолжает работать в течение заданного времени, даже если RLO при этом переходит в 0. Отсчет заданного отрезка времени начинается вновь при переходе RLO из 0 в 1 до окончания отсчета времени.Таймер запускается в соответствии с временем и временной базой, заданными в младшем слове аккумулятора.

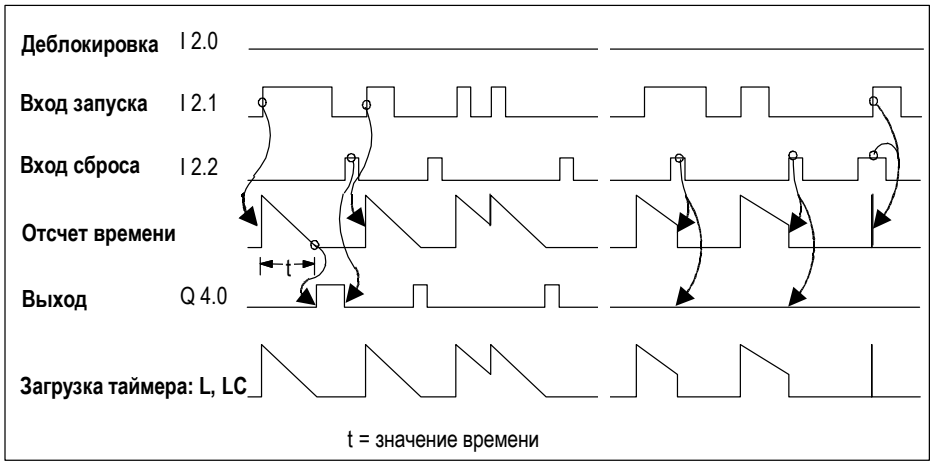
Дополнительную информацию Вы найдете в главе “Расположение таймеров в памяти и компоненты таймера“.

Биты слова состояния

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
Записывает:	-	-	-	-	-	0	-	-	0

Пример

STL		Комментарий
A	**I 2.0**	
FR	**T1**	**//Деблокировка таймера T1.**
A	**I 2.1**	
L	**S5T#10s**	**//Установка времени 10 секунд в ACCU 1.**
SS	**T1**	**//Запуск таймера T1 как “Задержка включения с памятью ”.**
A	**I 2.2**	
R	**T1**	**//Сброс таймера T1.**
A	**T1**	**//Опрос состояния таймера T1.**
=	**Q 4.0**	
L	**T1**	**//Загрузка текущего двоичного значения таймера T1.**
T	**MW10**	
LC	**T1**	**//Загрузка текущего значения таймера T1 в BCD-коде.**
T	**MW12**	



12.11 SF Таймер “Задержка выключения”

Формат

SF <timer>

Параметр	Тип данных	Область памяти	Описание
<timer>	TIMER	T	Номер таймера, диапазон зависит от CPU

Описание инструкции

SF <timer> (задержка выключения) запускает заданный таймер по отрицательному фронту RLO (переход из "1" в "0"). Таймер продолжает работать в течение заданного времени, пока RLO остается в 0. Отсчет заданного отрезка времени прекращается при переходе RLO из 0 в 1 до окончания отсчета времени.Таймер запускается в соответствии с временем и временной базой, заданными в младшем слове аккумулятора.

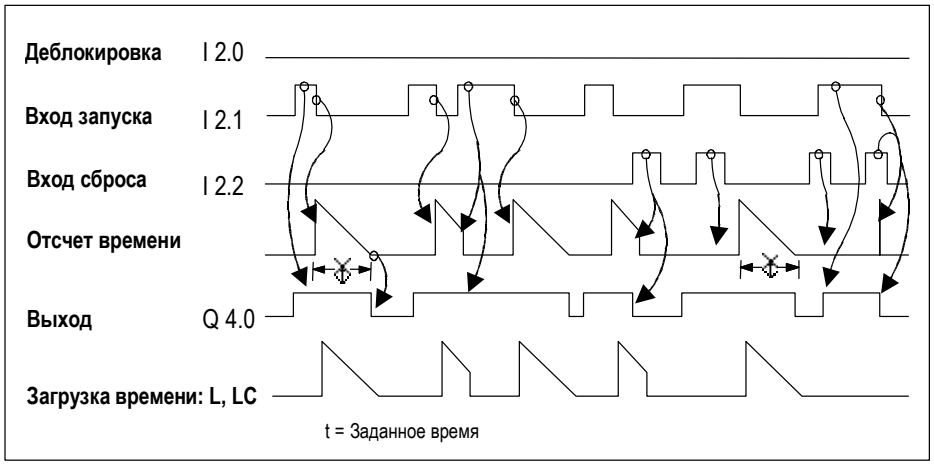
Дополнительную информацию Вы найдете в главе “Расположение таймеров в памяти и компоненты таймера“.

Биты слова состояния

	BIE	A1	A0	OV	OS	OR	STA	VKE	/ER
Записывает:	-	-	-	-	-	0	-	-	0

Пример

```
STL             Комментарий
A    I 2.0
FR   T1         //Деблокировка таймера T1.
A    I 2.1
L    S5T#10s    //Установка времени 10 секунд в ACCU 1.
SF   T1         //Запуск таймера T1 как “Задержка выключения ”.
A    I 2.2
R    T1         //Сброс таймера T1.
A    T1         //Опрос состояния таймера T1.
=    Q 4.0
L    T1         //Загрузка текущего двоичного значения таймера T1.
T    MW10
LC   T1         //Загрузка текущего значения таймера T1 в BCD-коде.
T    MW12
```


Деблокировка
I 2.0
Вход запуска
I 2.1
Вход сброса
I 2.2
Отсчет времени
Выход
Q 4.0
Загрузка времени: L, LC
t = Заданное время

13 Поразрядные логические инструкции со словами

13.1 Обзор поразрядных логических инструкций со словами

Описание

Поразрядные логические инструкции со словами выполняют попарное сопряжение бит в словах (16 бит) и двойных словах (32 бита) в соответствии с Булевой логикой. Оба операнда должны быть предварительно загружены в аккумуляторы процессора.

Для слов содержимое младшего слова аккумулятора 2 сопрягается с содержимым младшего слова аккумулятора 1 по заданной логической функции. Результат выполнения инструкции сохраняется в младшем слове аккумулятора 1, переписывая старое содержимое.

Для двойных слов содержимое аккумулятора 2 сопрягается с содержимым аккумулятора 1 по заданной логической функции. Результат выполнения инструкции сохраняется в аккумуляторе 1, переписывая старое содержимое.

Если результат не равен 0, бит СС 1 слова состояния устанавливается в "1". Если результат равен 0, бит СС 1 слова состояния сбрасывается в"0".

Вы можете использовать следующие инструкции со словами:

- AW Поразрядное И над словами (16-бит)
- OW Поразрядное ИЛИ над словами (16-бит)
- XOW Поразрядное исключающее ИЛИ над словами (16-бит)
- AD Поразрядное И над двойными словами (32-бит)
- OD Поразрядное ИЛИ над двойными словами (32-бит)
- XOD Поразрядное исключающее ИЛИ над двойными словами (32-бита)

13.2 AW Поразрядное И над словами (16-бит)

Формат

AW
AW <constant>

Параметр	Тип данных	Описание
<constant>	WORD, 16-битовая константа	Битовая комбинация для сопряжения с ACCU 1-L по схеме логического И

Описание инструкции

AW (И над словами) выполняет попарное “умножение” битов в младшем слове ACCU 1-L с младшим словом ACCU 2-L или с 16-битовой константой в соответствии с таблицей истинности для операции И. В результате бит получает "1" только если оба соответствующих разряда в сопрягаемых словах имеют статус "1". Результат сохраняется в ACCU 1-L. ACCU 1-H и ACCU 2 (и ACCU 3 и ACCU 4 для CPU с четырьмя аккумуляторами) остаются неизменными. Бит слова состояния CC 1 изменяется при выполнении данной операции (CC 1 = 1 если результат не равен нулю). Биты слова состояния CC 0 и OV сбрасываются в 0.

AW: Выполняет сопряжение ACCU 1-L с ACCU 2-L.

AW <constant>: Выполняет сопряжение ACCU 1 с 16-битовой константой.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	0	0	-	-	-	-	-

Примеры

№ бита	15 . . .	. .	. .	. . . 0
ACCU 1-L до инструкции **AW**	0101	1001	0011	1011
ACCU 2-L или 16-битовая константа:	1111	0110	1011	0101
Результат (ACCU 1-L) после выполнения **AW**	0101	0000	0011	0001

Пример 1

```
STL     Комментарий
L   IW20  //Загрузка содержимого IW20 в ACCU 1-L.
L   IW22  //Сохранение содержимого ACCU 1 в ACCU 2. Загрузка содержимого IW22 в ACCU 1-L.
AW        //Поразрядное сопряжение ACCU 1-L с ACCU 2-L по схеме И; сохранение результата в
          ACCU 1-L.
T   MW 8  //Передача результата в MW8.
```

Пример 2

```
STL             Комментарий
L     IW20      //Загрузка содержимого IW20 в ACCU 1-L.
AW    W#16#0FFF // Поразрядное сопряжение ACCU 1-L с 16-битовой константой
                (0000_1111_1111_1111) по  И; сохранение результата в  ACCU 1-L.
JP    NEXT      //Переход на метку NEXT если результат не равен нулю, (CC 1 = 1).
```

13.3 OW Поразрядное ИЛИ со словами (16-бит)

Формат

OW
OW <constant>

Параметр	Тип данных	Описание
<constant>	WORD, 16-битовая константа	Битовая комбинация для сопряжения с ACCU 1-L по схеме логического ИЛИ

Описание инструкции

OW (ИЛИ над словами) выполняет попарное “сложение” битов в младшем слове ACCU 1-L с младшим словом ACCU 2-L или с 16-битовой константой в соответствии с таблицей истинности для операции ИЛИ. В результате бит получает значение"1" если хотя бы один из соответствующих разрядов в сопрягаемой паре имеет статус "1". Результат сохраняется в ACCU 1-L. ACCU 1-H и ACCU 2 (и ACCU 3 и ACCU 4 для CPU с четырьмя аккумуляторами) остаются неизменными. Бит слова состояния CC 1 изменяется при выполнении данной операции (CC 1 = 1 если результат не равен нулю). Биты слова состояния CC 0 и OV сбрасываются в 0.

OW: Выполняет сопряжение ACCU 1-L с ACCU 2-L.

OW <constant>: Выполняет сопряжение ACCU 1 с 16-битовой константой.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	0	0	-	-	-	-	-

Примеры

№ бита	15 . . .	. .	. .	. . . 0
ACCU 1-L до инструкции **OW**	0101	0101	0011	1011
ACCU 2-L или 16-битовая константа:	1111	0110	1011	0101
Результат (ACCU 1-L) после выполнения **OW**	1111	0111	1011	1111

Пример 1

STL		Комментарий
L	**IW20**	**//Загрузка содержимого IW20 в ACCU 1-L.**
L	**IW22**	**//Передача содержимого ACCU 1 в ACCU 2. Загрузка содержимого IW22 в ACCU 1-L.**
OW		**//Поразрядное сопряжение ACCU 1-L с ACCU 2-L по схеме ИЛИ; сохранение результата в ACCU 1-L.**
T	**MW8**	**//Передача результата в MW8.**

Пример 2

STL		Комментарий
L	**IW20**	**//Загрузка содержимого IW20 в ACCU 1-L.**
OW	**W#16#0FFF**	**// Поразрядное сопряжение ACCU 1-L с 16-битовой константой (0000_1111_1111_1111) по ИЛИ; сохранение результата в ACCU 1-L.**
JP	**NEXT**	**//Переход на метку NEXT если результат не равен нулю, (CC 1 = 1).**

13.4 XOW Поразрядное Исключающее ИЛИ со словами (16-бит)

Формат

XOW
XOW <constant>

Параметр	Тип данных	Описание
<constant>	WORD, 16-битовая константа	Битовая комбинация для сопряжения с ACCU 1-L по схеме Исключающее ИЛИ

Описание инструкции

XOW (Исключающее ИЛИ над словами) выполняет попарное сопряжение битов в младшем слове ACCU 1-L с младшим словом ACCU 2-L или с 16-битовой константой в соответствии с таблицей истинности для операции Исключающее ИЛИ. В результате бит получает значение"1" если только один из соответствующих разрядов в сопрягаемых словах имел статус "1". Результат сохраняется в ACCU 1-L. ACCU 1-H и ACCU 2 (и ACCU 3 и ACCU 4 для CPU с четырьмя аккумуляторами) остаются неизменными. Бит слова состояния CC 1 изменяется при выполнении данной операции (CC 1 = 1 если результат не равен нулю). Биты слова состояния CC 0 и OV сбрасываются в 0. Вы можете использовать эту инструкцию несколько раз подряд. При этом результат получает значение"1" если нечетное число опрашиваемых адресов имеет статус “1”.

XOW: Выполняет сопряжение ACCU 1-L с ACCU 2-L.

XOW <constant>: Выполняет сопряжение ACCU 1 с 16-битовой константой**.**

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	0	0	-	-	-	-	-

Примеры

№ бита	15 . . .	. .	. .	. . . 0
ACCU 1-L до инструкции **XOW**	0101	0101	0011	1011
ACCU 2-L или 16-битовая константа:	1111	0110	1011	0101
Результат (ACCU 1-L) после выполнения **XOW**	1010	0011	1000	1110

Пример 1

```
STL          Комментарий
L    IW20    //Загрузка содержимого  IW20 в ACCU 1-L.
L    IW22    //Передача содержимого ACCU 1 в ACCU 2. Загрузка содержимого IW22 в ACCU 1-L.
XOW          //Поразрядное сопряжение ACCU 1-L с ACCU 2-L по функции Исключающее ИЛИ;
                сохранение результата в ACCU 1-L.
T    MW8     //Передача результата в MW8.
```

Пример 2

```
STL          Комментарий
L    IW20    //Загрузка содержимого IW20 в ACCU 1-L.
XO   16#0FFF // Поразрядное сопряжение ACCU 1-L с 16-битовой константой (0000_1111_1111_1111)
W            по  функции Исключающее ИЛИ; сохранение результата в  ACCU 1-L.
JP   NEXT    //Переход на метку NEXT если результат не равен нулю, (CC 1 = 1).
```

13.5 AD Поразрядное И с двойными словами (32-бита)

Формат

AD
AD <constant>

Параметр	Тип данных	Описание
<constant>	WORD, 32-битовая константа	Битовая комбинация для сопряжения с ACCU 1по схеме И

Описание инструкции

AD (И над двойными словами) выполняет попарное “умножение” битов в аккумуляторе ACCU 1 с ACCU 2 или с 32-битовой константой в соответствии с таблицей истинности для операции И. В результате бит получает "1" только если оба соответствующих разряда в сопрягаемых двойных словах имеют статус "1". Результат сохраняется в ACCU 1.
ACCU 2 (ACCU 3 и ACCU 4 для CPU с четырьмя аккумуляторами) остаются неизменными. Бит слова состояния CC 1 изменяется при выполнении данной операции (CC 1 = 1 если результат не равен нулю). Биты слова состояния CC 0 и OV сбрасываются в 0.

AD: Выполняет сопряжение ACCU 1 с ACCU 2.

AD <constant>: Выполняет сопряжение ACCU 1 с 32-битовой константой.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	0	0	-	-	-	-	-

Примеры

№ бита	31 . .	. .	. .	. .	. .	. .	. .	. . . 0
ACCU 1 до инструкции **AD**	0101	0000	1111	1100	1000	1001	0011	1011
ACCU 2 или 32-битовая константа:	1111	0011	1000	0101	0111	0110	1011	0101
Результат (ACCU 1) после выполнения **AD**	0101	0000	1000	0100	0000	0000	0011	0001

Пример 1

```
STL     Комментарий
L   ID20    //Загрузка содержимого ID20 в ACCU 1.
L   ID24    //Передача содержимого ACCU 1 в ACCU 2. Загрузка содержимого  ID24 в ACCU 1.
AD          //Поразрядное сопряжение ACCU 1 с ACCU 2 по схеме И; сохранение результата в ACCU 1.
T   MD8     //Передача результата в MD8.
```

Пример 2

```
STL     Комментарий
L   ID 20           //Загрузка содержимого ID20 в ACCU 1.
AD  DW#16#0FFF_EF21 // Поразрядное сопряжение ACCU 1 с 32-битовой константой
                    (0000_1111_1111_1111_1110_1111_0010_0001) по схеме И; сохранение
                    результата в ACCU 1.
JP  NEXT            //Переход на метку NEXT , если  результат не равен нулю,
                    (CC 1 = 1).
```

13.6 OD Поразрядное ИЛИ с двойными словами (32-бита)

Формат

OD
OD <constant>

Параметр	Тип данных	Описание
<constant>	WORD, 32-битовая константа	Битовая комбинация для сопряжения с ACCU 1по схеме И

Описание инструкции

OD (ИЛИ над двойными словами) выполняет попарное “сложение” битов в аккумуляторе ACCU 1 с ACCU 2 или с 32-битовой константой в соответствии с таблицей истинности для операции ИЛИ. В результате бит получает значение"1" если хотя бы один из соответствующих разрядов в сопрягаемой паре имеет статус "1". Результат сохраняется в ACCU 1.
ACCU 2 (ACCU 3 и ACCU 4 для CPU с четырьмя аккумуляторами) остаются неизменными. Бит слова состояния CC 1 изменяется при выполнении данной операции (CC 1 = 1 если результат не равен нулю). Биты слова состояния CC 0 и OV сбрасываются в 0.

OD: Выполняет сопряжение ACCU 1 с ACCU 2.

OD <constant>: Выполняет сопряжение ACCU 1 с 32-битовой константой.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	0	0	-	-	-	-	-

Примеры

№ бита	31 . .	. .	. .	. .	. .	. .	. .	. . . 0
ACCU 1 до инструкции **OD**	0101	0000	1111	1100	1000	0101	0011	1011
ACCU 2 или 32-битовая константа:	1111	0011	1000	0101	0111	0110	1011	0101
Результат (ACCU 1) после выполнения **OD**	1111	0011	1111	1101	1111	0111	1011	1111

Пример 1

```
STL                 Комментарий
L     ID20          //Загрузка содержимого ID20 в ACCU 1.
L     ID24          //Передача содержимого ACCU 1 в ACCU 2. Загрузка содержимого ID24 в
                    ACCU 1.
OD                  //Поразрядное сопряжение ACCU 1 с ACCU 2 по схеме ИЛИ; сохранение
                    результата в ACCU 1.
T     MD8           //Передача результата в MD8.
```

Пример 2

STL		Комментарий
L	**ID20**	**//Загрузка содержимого ID20 в ACCU 1.**
OD	**DW#16#0FFF_EF21**	**// Поразрядное сопряжение ACCU 1 с 32-битовой константой (0000_1111_1111_1111_1110_1111_0010_0001) по схеме ИЛИ; сохранение результата в ACCU 1.**
JP	**NEXT**	**//Переход на метку NEXT , если результат не равен нулю, (CC 1 = 1).**

13.7 XOD Поразрядное Исключающее ИЛИ с двойными словами (32-бита)

Формат

XOD
XOD <constant>

Параметр	Тип данных	Описание
<constant>	WORD, 32-битовая константа	Битовая комбинация для сопряжения с ACCU 1по схеме Исключающее ИЛИ

Описание инструкции

XOD (Исключающее ИЛИ над двойными словами) выполняет попарное сопряжение битов в аккумуляторе ACCU 1 с ACCU 2 или с 32-битовой константой в соответствии с таблицей истинности для операции Исключающее ИЛИ. В результате бит получает значение"1" если только один из соответствующих разрядов в сопрягаемой паре имеет статус "1". Результат сохраняется в ACCU 1.
ACCU 2 (ACCU 3 и ACCU 4 для CPU с четырьмя аккумуляторами) остаются неизменными. Бит слова состояния CC 1 изменяется при выполнении данной операции (CC 1 = 1 если результат не равен нулю). Биты слова состояния CC 0 и OV сбрасываются в 0. Вы можете использовать эту инструкцию несколько раз подряд. При этом результат получает значение"1" если нечетное число опрашиваемых адресов имеет статус “1”.

XOD: Выполняет сопряжение ACCU 1 с ACCU 2.

XOD <constant>: Выполняет сопряжение ACCU 1 с 32-битовой константой

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	x	0	0	-	-	-	-	-

Примеры

№ бита	**31 . .**	**. .**	**. .**	**. .**	**. .**	**. .**	**. .**	**. . . 0**
ACCU 1 до инструкции **XOD**	0101	0000	1111	1100	1000	0101	0011	1011
ACCU 2 или 32-битовая константа:	1111	0011	1000	0101	0111	0110	1011	0101
Результат (ACCU 1) после выполнения **XOD**	1010	0011	0111	1001	1111	0011	1000	1110

Пример 1

STL		**Комментарий**
L	**ID20**	**//Загрузка содержимого ID20 в ACCU 1.**
L	**ID24**	**//Передача содержимого ACCU 1 в ACCU 2. Загрузка содержимого ID24 в ACCU 1.**
XOD		**//Поразрядное сопряжение ACCU 1 с ACCU 2 по схеме Исключающее ИЛИ; сохранение результата в ACCU 1.**
T	**MD8**	**//Передача результата в MD8.**

Пример 2

STL		**Комментарий**
L	**ID20**	**//Загрузка содержимого ID20 в ACCU 1.**
XOD	**DW#16#0FFF_EF21**	**// Поразрядное сопряжение ACCU 1 с 32-битовой константой (0000_1111_1111_1111_1110_1111_0010_0001) по схеме Исключающее ИЛИ; сохранение результата в ACCU 1.**
JP	**NEXT**	**//Переход на метку NEXT , если результат не равен нулю, (CC 1 = 1).**

14 Инструкции с аккумуляторами

14.1 Обзор инструкций с аккумуляторами и адресными регистрами

Описание

В Вашем распоряжении имеются следующие инструкции для обработки содержимого одного или обоих аккумуляторов:

- TAK Обмен содержимым аккумуляторов ACCU 1 и ACCU 2
- PUSH Для CPU с двумя аккумуляторами
- PUSH Для CPU с четырьмя аккумуляторами
- POP Для CPU с двумя аккумуляторами
- POP Для CPU с четырьмя аккумуляторами

- ENT Ввод в стек аккумуляторов
- LEAVE Вывод в стек аккумуляторов
- INC Инкремент ACCU 1-L-L
- DEC Декремент ACCU 1-L-L

- +AR1 Сложение ACCU 1 с адресным регистром AR 1
- +AR2 Сложение ACCU 1 с адресным регистром AR 2

- BLD Инструкция отображения программы
- NOP 0 Нулевая инструкция
- NOP 1 Нулевая инструкция

14.2 TAK Обмен содержимым ACCU 1 и ACCU 2

Формат

TAK

Описание

Инструкция **TAK** (Обмен содержимым ACCU 1 и ACCU 2) обменивает содержимое аккумулятора 1 с содержимым аккумулятора 2. Инструкция выполняется независимо от битов слова состояния и не изменяет их. В CPU с четырьмя аккумуляторами, ACCU 3 и ACCU 4 остаются без изменений.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример: Вычитание меньшего числа из большего

```
STL                     Комментарий
      L     MW10        //Загрузка содержимого  MW10 в ACCU 1-L.
      L     MW12        //Перенос содержимого ACCU 1-L в ACCU 2-L. Загрузка содержимого  MW12 в
                        ACCU 1-L.
      >I                //Проверка: ACCU 2-L (MW10) больше чем ACCU 1-L (MW12) ?
      SPB   NEXT        //Переход на метку NEXT если ACCU 2 (MW10) больше, чем ACCU 1 (MW12).
      TAK               //Обмен содержимым ACCU 1 и ACCU 2
NEXT: -I                //Вычитание содержимого  ACCU 2-L  из содержимого ACCU 1-L.

      T     MW14        //Сохранение  результата (= большее – меньшее) в MW14.
```

Содержимое	ACCU 1	ACCU 2
Перед выполнением инструкции **TAK**	<MW12>	<MW10>
После выполнения инструкции **TAK**	<MW**10**>	<MW**12**>

14.3 POP CPU с двумя аккумуляторами

Формат

POP

Описание

POP (CPU с двумя аккумуляторами) Копирует содержимое аккумулятора 1 в аккумулятор 2. При этом ACCU 2 остается неизменным. Инструкция выполняется независимо от битов слова состояния и не изменяет их.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL          Комментарий
T    MD10    //Передать  содержимое ACCU 1 (= число A) в  MD10
POP          //Копия  содержимого  ACCU 2 в ACCU 1
T    MD14    // Передать  содержимое ACCU 1 (= число B) в  MD14
```

Содержимое	ACCU 1	ACCU 2
Перед выполнением инструкции **POP**	число A	число B
После выполнения инструкции **POP**	число B	число B

14.4 POP CPU с четырьмя аккумуляторами

Формат

POP

Описание

POP (CPU с четырьмя аккумуляторами) копирует содержимое ACCU 2 в ACCU 1, содержимое ACCU 3 в ACCU 2, и содержимое ACCU 4 в ACCU 3. ACCU 4 остается неизменным Инструкция выполняется независимо от битов слова состояния и не изменяет их.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

STL		Комментарий
T	MD10	//Передает содержимое ACCU 1 (= число A) в MD10
POP		//Копирует содержимое ACCU 2 в ACCU 1
T	MD14	//Передает содержимое ACCU 1 (= число B) в MD14

Содержимое	ACCU 1	ACCU 2	ACCU 3	ACCU 4
Перед выполнением инструкции **POP**	число A	число B	число C	число D
После выполнения инструкции **POP**	число B	число C	число D	число D

14.5 PUSH CPU с двумя аккумуляторами

Формат

PUSH

Описание

PUSH (ACCU 1 в ACCU 2) копирует содержимое ACCU 1 в ACCU 2.
ACCU 1 остается неизменным Инструкция выполняется независимо от битов слова состояния и не изменяет их.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL             Комментарий
L       MW10    //Загрузка содержимого  MW10 в ACCU 1.
PUSH            //Копирует содержимое  ACCU 1 в ACCU 2.
```

Содержимое	ACCU 1	ACCU 2
Перед выполнением инструкции **PUSH**	<MW10>	<X>
После выполнения инструкции **PUSH**	<MW10>	<MW10>

14.6 PUSH CPU с четырьмя аккумуляторами

Формат

PUSH

Описание

PUSH (CPU с четырьмя аккумуляторами) копирует содержимое ACCU 3 в ACCU 4, содержимое ACCU 2 в ACCU 3, и содержимое ACCU 1 в ACCU 2. ACCU 1 остается неизменным. Инструкция выполняется независимо от битов слова состояния и не изменяет их.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

STL		Комментарий
L	MW10	//Загрузка содержимого MW10 в ACCU 1.
PUSH		//Копия содержимого ACCU 1 в ACCU 2, содержимого ACCU 2- в ACCU 3, и содержимого ACCU 3- в ACCU 4.

Содержимое	ACCU 1	ACCU 2	ACCU 3	ACCU 4
Перед выполнением инструкции **PUSH**	число A	число B	число C	число D
После выполнения инструкции **PUSH**	число A	число A	число B	число C

14.7 ENT Ввод в стек аккумуляторов

Формат

ENT

Описание

ENT (Ввод в стек аккумуляторов) копирует содержимое ACCU 3 в ACCU 4 и содержимое ACCU 2 в ACCU 3. При программировании инструкции ENT непосредственно перед инструкцией загрузки, Вы можете сохранить промежуточный результат в ACCU 3.

Пример

STL		Комментарий
L	DBD0	//Загрузка числа из двойного слова данных DBD0 в ACCU 1. (Это число типа REAL).
L	DBD4	//Копирование числа из ACCU 1 в ACCU 2. Загрузка числа из двойного слова данных DBD4 в ACCU 1. (Это число типа REAL).
+R		//Сложение содержимого ACCU 1 и ACCU 2 как чисел типа REAL (32 бита, IEEE-FP) и сохранение результата в ACCU 1.
L	DBD8	// Копирование числа из ACCU 1 в ACCU 2 и загрузка числа из двойного слова данных DBD8 в ACCU 1.
ENT		// Копирование содержимого ACCU 3 в ACCU 4. Копирование содержимого ACCU 2 (промежуточный результат) в ACCU 3.
L	DBD12	//Загрузка числа из двойного слова данных DBD12 в ACCU 1.
-R		//Вычитание содержимого ACCU 1 из содержимого ACCU 2 .Результат – в ACCU 1. Копирование содержимого ACCU 3 в ACCU 2. Копирование содержимого ACCU 4 в ACCU 3.
/R		//Деление содержимого ACCU 2 (DBD0 + DBD4) на содержимое ACCU 1 (DBD8 - DBD12). Сохранение результата в ACCU 1.
T	DBD16	//Передача результата (ACCU 1) в двойное слово данных DBD16.

14.8 LEAVE Вывод из стека аккумуляторов

Формат

LEAVE

Описание

LEAVE (Вывод из стека аккумуляторов) копирует содержимое ACCU 3 в ACCU 2 и содержимое ACCU 4 в ACCU 3. При программировании инструкции LEAVE непосредственно перед инструкциями сдвига и циклического сдвига и поразрядными инструкциями с аккумуляторами, инструкция вывода из стека аккумуляторов ведет себя как математические инструкции. Содержимое ACCU 1 и ACCU 4 остается неизменным.

14.9 INC Инкремент ACCU 1-L-L

Формат

INC <8-bit integer>

Параметр	Тип данных	Описание
<8-bit integer>	8-битовая целая константа	Константа прибавляется к ACCU 1-L-L; диапазон :от 0 до 255

Описание

INC <8-bit integer> (инкремент ACCU 1-L-L) Складывает 8-битовое целое число и содержимым ACCU 1-L-L и сохраняет результат в ACCU 1-L-L. ACCU 1-L-H, ACCU 1-H, и ACCU 2 остаются неизменными. Инструкция выполняется независимо от битов слова состояния и не изменяет их.

Примечание

Эти инструкции не могут использоваться в 16-битной или 32-битной математике, потому что не выполняют переноса при переполнении младшего байта в аккумуляторе 1 в старший байт младшего слова аккумулятора 1. Для 16-битовой или 32-битовой математики, используйте +I или +D. инструкции, соответственно.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL     Комментарий
L   MB22  //Загрузка числа из MB22
INC 1     //Инструкция "Инкремент ACCU 1 (MB22) на 1"; сохранение результата в ACCU 1-L-L
T   MB22  //Передает содержимое ACCU 1-L-L (результат) назад в MB22
```

14.10 DEC Декремент ACCU 1-L-L

Формат

DEC <8-bit integer>

Address	Тип данных	Описание
<8-bit integer>	8-битовая целая константа	Константа вычитается из ACCU 1-L-L; диапазон :от 0 до 255

Описание

DEC <8-bit integer> (декремент ACCU 1-L-L) Вычитает 8-битовое целое число из содержимого ACCU 1-L-L и сохраняет результат в ACCU 1-L-L. ACCU 1-L-H, ACCU 1-H, и ACCU 2 остаются неизменными. Инструкция выполняется независимо от битов слова состояния и не изменяет их.

Примечание

Эти инструкции не могут использоваться в 16-битной или 32-битной математике, потому что не выполняют переноса при переполнении младшего байта в аккумуляторе 1 в старший байт младшего слова аккумулятора 1. Для 16-битовой или 32-битовой математики, используйте +I или +D. инструкции, соответственно.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример

```
STL             Комментарий
L     MB250     //Загрузка  числа из MB250
DEC   1         //Инструкция "Декремент ACCU 1 (MB250) на 1";сохранение результата в  ACCU 1-L-L
T     MB250     //Передает  содержимое  ACCU 1-L-L (результат) назад в MB250
```

14.11 +AR1 Сложение с адресным регистром AR1

Формат

+AR1
+AR1 <P#Byte.Bit>

Параметр	Тип данных	Описание
<P#Byte.Bit>	Константа - указатель	Адрес, прибавляемый к AR1

Описание

+AR1 (Сложение с AR1) прибавляет значение, заданное в параметре инструкции или в ACCU 1-L к содержимому AR1. Целое (16 битовое) сначала преобразуется в 24-битовое со знаком и затем складывается с младшими 24 битами AR1 (частью относительного адреса в AR1). Часть идентифицирующая область в AR1 (биты 24, 25, и 26) остается неизменной. Инструкция выполняется независимо от битов слова состояния и не изменяет их.

+AR1: Целое (16 битовое) , заданное в ACCU 1-L прибавляется к содержимому AR1. Возможны числа от -32768 до +32767 .

+AR1 <P#Byte.Bit>: Сложение с заданным в параметре смещением <P#Byte.Bit>.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример 1

```
STL          Комментарий
L     +300   //Загрузка числа в ACCU 1-L
+AR1         //Сложение ACCU 1-L (целое, 16 бит) с содержимым AR1.
```

Пример 2

```
STL          Комментарий
+AR1  P#300.0 //Сложение AR1 с постоянным смещением 300.0
```

14.12 +AR2 Сложение с адресным регистром AR2

Формат

+AR2
+AR2 <P#Byte.Bit>

Параметр	Тип данных	Описание
<P#Byte.Bit>	Константа-указатель	Адрес, добавляемый к AR2

Описание

+AR2 (Сложение с AR2) прибавляет значение, заданное в параметре инструкции или в ACCU 1-L к содержимому AR2. Целое (16 битовое) сначала преобразуется в 24-битовое со знаком и затем складывается с младшими 24 битами AR2 (частью относительного адреса в AR2). Часть идентифицирующая область в AR2 (биты 24, 25, и 26) остается неизменной. Инструкция выполняется независимо от битов слова состояния и не изменяет их.

+AR2: Целое (16 битовое) , заданное в ACCU 1-L прибавляется к содержимому AR2. Возможны числа от -32768 до +32767 .

+AR2 <P#Byte.Bit>: Сложение с заданным в параметре смещением <P#Byte.Bit>.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

Пример 1

```
STL          Комментарий
L     +300   //Загрузка числа в ACCU 1-L.
+AR1         // Сложение ACCU 1-L (целое, 16 бит) с содержимым AR2.
```

Пример 2

```
STL          Комментарий
+AR1  P#300.0  // Сложение AR1 с постоянным смещением 300.0
```

14.13 BLD Инструкция отображения программы

Формат

BLD <number>

Параметр	Описание
<number>	Число, определяющее BLD инструкцию, диапазон от 0 до 255

Описание

BLD <number> (Инструкция отображения программы; нулевая операция) не выполняет каких-либо функций и не изменяет слова состояния. Инструкция используется программатором (PG) для графического отображения объектов. Создается автоматически при программировании на языках LAD или FBD и отображается эта инструкция при переключении на STL. Параметр <number> , определяющий BLD инструкцию вычисляется программатором.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

14.14 NOP 0 Нулевая инструкция

Формат

NOP 0

Описание

NOP 0 (Инструкция NOP с параметром "0") не выполняет какой-либо функции и не влияет на биты слова состояния. Код инструкции представляет собой последовательность из 16-ти нулей. Инструкция необходима только для программатора при отображении программы в графическом языке.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

14.15 NOP 1 Нулевая инструкция

Формат

NOP 1

Описание

NOP 1 (Инструкция NOP с параметром "1") не выполняет какой-либо функции и не влияет на биты слова состояния. Код инструкции представляет собой последовательность из 16-ти единиц. Инструкция необходима только для программатора при отображении программы в графическом языке.

Биты слова состояния

	BR	CC 1	CC 0	OV	OS	OR	STA	RLO	/FC
Записывает:	-	-	-	-	-	-	-	-	-

A Обзор всех STL инструкций

A.1 Алфавитный список STL инструкций в немецкой мнемонике (SIMATIC)

Немецкая мнемоника	Английская мнемоника	Каталог программных элементов	Описание
+	+	Integer math Instruction	Сложение констант целого типа (16, 32бит)
=	=	Bit logic Instruction	Присвоение
)	)	Bit logic Instruction	Закрывающая скобка
+AR1	+AR1	Accumulator	Сложение адресного регистра 1 (AR1) с ACCU 1
+AR2	+AR2	Accumulator	Сложение адресного регистра 2 (AR2) с ACCU 1
+D	+D	Integer math Instruction	Сложение ACCU 1 и ACCU 2 как двойных целых чисел (32-бита)
–D	–D	Integer math Instruction	Вычитание ACCU 1 из ACCU 2 в формате двойных целых чисел (32-бита)
*D	*D	Integer math Instruction	Умножение ACCU 1 и ACCU 2 как двойных целых чисел(32-бита)
/D	/D	Integer math Instruction	Деление ACCU 2 на ACCU 1 как двойных целых чисел (32-бита)
? D	? D	Compare	Сравнение двойных целых чисел (32-бита)==, <>, >, <, >=, <=
+I	+I	Integer math Instruction	Сложение ACCU 1 и ACCU 2 как целых чисел (16-бит)
–I	–I	Integer math Instruction	Вычитание ACCU 1 из ACCU 2 в формате целых чисел (16-бит)
*I	*I	Integer math Instruction	Умножение ACCU 1 и ACCU 2 как целых чисел (16-бит)
/I	/I	Integer math Instruction	Деление ACCU 2 на ACCU 1 в формате целых чисел (16-бит)
? I	? I	Compare	Сравнение в формате целых чисел (16-bit) ==, <>, >, <, >=, <=
+R	+R	Floating point Instruction	Сложение ACCU 1 и ACCU 2 как чисел с плавающей точкой (32-бита IEEE-FP)
–R	–R	Floating point Instruction	Вычитание ACCU 1 из ACCU 2 в формате чисел с плавающей точкой (32-бита IEEE-FP)
*R	*R	Floating point Instruction	Умножение ACCU 1 и ACCU 2 как чисел с плавающей точкой (32-бита IEEE-FP)

Немецкая мнемоника	**Английская мнемоника**	**Каталог программных элементов**	**Описание**
/R	/R	Floating point Instruction	Умножение ACCU 2 на ACCU 1 в формате чисел с плавающей точкой (32-бита IEEE-FP)
? R	? R	Compare	Сравнение чисел с плавающей точкой (32-бита) ==, <>, >, <, >=, <=
ABS	ABS	Floating point Instruction	Модуль числа с плавающей точкой (32-бита IEEE-FP)
ACOS	ACOS	Floating point Instruction	Вычисление арккосинуса чисел с плавающей точкой (32-бита)
ASIN	ASIN	Floating point Instruction	Вычисление арксинуса чисел с плавающей точкой (32-бита)
ATAN	ATAN	Floating point Instruction	Вычисление арктангенса чисел с плавающей точкой (32-бита)
AUF	OPN	DB call	Открыть блок данных
BE	BE	Program control	Конец блока
BEA	BEU	Program control	Безусловный конец блока
BEB	BEC	Program control	Условный конец блока
BLD	BLD	Program control	Инструкция отображения программы
BTD	BTD	Convert	Преобразование BCD в Double Integer (32-бита)
BTI	BTI	Convert	Преобразование BCD в Integer (16-бит)
CALL	CALL	Program control	Вызов блока
CALL	CALL	Program control	Вызов мультиэкземпляра
CALL	CALL	Program control	Вызов блока из библиотеки
CC	CC	Program control	Условный вызов
CLR	CLR	Bit logic Instruction	Сброс RLO (=0)
COS	COS	Floating point Instruction	Вычисление косинуса чисел с плавающей точкой (32-бита)
DEC	DEC	Accumulator	Декремент ACCU 1-L-L
DTB	DTB	Convert	Преобразование Double Integer (32-бита) в BCD
DTR	DTR	Convert	Преобразование Double Integer (32-бита) в число с плавающей точкой (32-бита IEEE-FP)
ENT	ENT	Accumulator	Ввод в стек ACCU
EXP	EXP	Floating point Instruction	Вычисление экспоненты чисел с плавающей точкой (32-бита)
FN	FN	Bit logic Instruction	Выделение отрицательного фронта
FP	FP	Bit logic Instruction	Выделение положительного фронта
FR	FR	Counters	Деблокировка счетчика (от C 0 до C 255)
FR	FR	Timers	Деблокировка таймера
INC	INC	Accumulator	Инкремент ACCU 1-L-L
INVD	INVD	Convert	Инверсия двойного целого числа (32-бита)
INVI	INVI	Convert	Инверсия целого числа (16-бит)
ITB	ITB	Convert	Преобразование Integer (16-бит) в BCD
ITD	ITD	Convert	Преобразование Integer (16-бит) в Double Integer (32-бита)

Немецкая мнемоника	**Английская мнемоника**	**Каталог программных элементов**	**Описание**
L	L	Load/Transfer	Загрузка
L DBLG	L DBLG	Load/Transfer	Загрузка длины глобального DB в ACCU 1
L DBNO	L DBNO	Load/Transfer	Загрузка номера глобального DB в ACCU 1
L DILG	L DILG	Load/Transfer	Загрузка длины экземплярного DB в ACCU 1
L DINO	L DINO	Load/Transfer	Загрузка номера экземплярного DB в ACCU 1
L STW	L STW	Load/Transfer	Загрузка слова состояния в ACCU 1
L	L	Load/Transfer	Загрузка текущего значения таймера в ACCU 1 в двоичном коде (номер таймера должен быть в пределах от 0 до 255, например, L T 32)
L	L	Load/Transfer	Загрузка текущего значения счетчика в ACCU 1 в двоичном коде (номер счетчика должен быть в пределах от 0 до 255, например, L C 15)
LAR1	LAR1	Load/Transfer	Загрузка в адресный регистр1 значения из ACCU 1
LAR1 <D>	LAR1 <D>	Load/Transfer	Загрузка в адресный регистр1 двойного целого (32-битовый указатель)
LAR1 AR2	LAR1	Load/Transfer	Загрузка в адресный регистр1 значения из адресного регистра 2
LAR2	LAR2	Load/Transfer	Загрузка в адресный регистр2 значения из ACCU 1
LAR2 <D>	LAR2 <D>	Load/Transfer	Загрузка в адресный регистр 2 двойного целого (32-битовый указатель)
LC	LC	Counters	Загрузка текущего значения счетчика в ACCU 1 в BCD (номер счетчика должен быть в пределах от 0 до 255, например, LC C 15)
LC	LC	Timers	Загрузка текущего значения таймера в ACCU 1 в двоичном коде (номер таймера должен быть в пределах от 0 до 255, например, LC T 32)
LEAVE	LEAVE	Accumulator	Вывод из стека аккумуляторов
LN	LN	Floating point Instruction	Вычисление натурального логарифма чисел с плавающей точкой (32-бита)
LOOP	LOOP	Jumps	Инструкция цикла
MCR(	MCR(	Program control	Сохранение RLO в MCR стеке, начало MCR зоны
)MCR	)MCR	Program control	Конец MCR зоны
MCRA	MCRA	Program control	Активация MCR области
MCRD	MCRD	Program control	Деактивация MCR области
MOD	MOD	Integer math Instruction	Выделение остатка от деления двойных целых чисел (32-бита)
NEGD	NEGD	Convert	Инверсия знака двойных целых чисел (32-бит)
NEGI	NEGI	Convert	Инверсия знака целых чисел (16-бит)
NEGR	NEGR	Convert	Инверсия знака чисел с плавающей точкой (32-бита, IEEE-FP)
NOP 0	NOP 0	Accumulator	Нулевая инструкция
NOP 1	NOP 1	Accumulator	Нулевая инструкция

Немецкая мнемоника	Английская мнемоника	Каталог программных элементов	Описание
NOT	NOT	Bit logic Instruction	Инверсия RLO
O	O	Bit logic Instruction	ИЛИ
O(	O(	Bit logic Instruction	ИЛИ с открытием скобки
OD	OD	Word logic Instruction	Побитовое ИЛИ над двойными словами (32-бита)
ON	ON	Bit logic Instruction	ИЛИ-НЕ
ON(	ON(	Bit logic Instruction	ИЛИ-НЕ_с открытием скобки
OW	OW	Word logic Instruction	Побитовое ИЛИ над словами (16-бит)
POP	POP	Accumulator	Аккумулятор 1 <– Аккумулятор 2, Аккумулятор 2 <– Аккумулятор 3, Аккумулятор 3 <– Аккумулятор 4 (CPU с четырьмя аккумуляторами)
POP	POP	Accumulator	Аккумулятор 1 <– Аккумулятор 2 (CPU с двумя аккумуляторами)
PUSH	PUSH	Accumulator	Аккумулятор 3 –> Аккумулятор 4, Аккумулятор 2 –> Аккумулятор 3, Аккумулятор 1 –> Аккумулятор 2 (CPU с четырьмя аккумуляторами)
PUSH	PUSH	Accumulator	Аккумулятор 1 –> Аккумулятор 2 (CPU с двумя аккумуляторами)
R	R	Bit logic Instruction	Сброс
R	R	Counters	Сброс счетчика (номер счетчика должен быть в пределах от 0 до 255, например, R C 15)
R	R	Timers	Сброс таймера (номер таймера должен быть в пределах от 0 до 255, например, R T 32)
RLD	RLD	Shift/Rotate	Циклический сдвиг двойного слова влево (32-бита)
RLDA	RLDA	Shift/Rotate	Циклический сдвиг двойного слова влево (32-бита) через CC 1
RND	RND	Convert	Округление
RND+	RND+	Convert	Округление до ближайшего большего двойного целого числа
RND–	RND–	Convert	Округление до ближайшего меньшего двойного целого числа
RRD	RRD	Shift/Rotate	Циклический сдвиг двойного слова вправо (32-бита)
RRDA	RRDA	Shift/Rotate	Циклический сдвиг двойного слова вправо (32-бита) через C 1
S	S	Bit logic Instruction	Установка
S	S	Counters	Установить начальное значение счетчика (где текущий счетчик может иметь номер в диапазоне от 0 до 255, например: S C 15)
SA	SF	Timers	Таймер – формирователь задержки выключения
SAVE	SAVE	Bit logic Instruction	Сохранить RLO в регистре BR
SE	SD	Timers	Таймер – формирователь задержки включения
SET	SET	Bit logic Instruction	Установить RLO в 1
SI	SP	Timers	Таймер – формирователь импульса

Немецкая мнемоника	**Английская мнемоника**	**Каталог программных элементов**	**Описание**
SIN	SIN	Floating point Instruction	Синус числа с плавающей точкой (32 бита, IEEE FP)
SLD	SLD	Shift/Rotate	Сдвинуть влево двойное слово (32 бита)
SLW	SLW	Shift/Rotate	Сдвинуть влево слово (16 битов)
SPA	JU	Jumps	Перейти безусловно
SPB	JC	Jumps	Перейти, при RLO = 1
SPBB	JCB	Jumps	Перейти при RLO = 1 с сохранением в BR бите
SPBI	JBI	Jumps	Перейти при BR = 1
SPBIN	JNBI	Jumps	Перейти при BR = 0
SPBN	JCN	Jumps	Перейти при RLO = 0
SPBNB	JNB	Jumps	Перейти при RLO = 0 с сохрангением в BR бите
SPL	JL	Jumps	Распределенный переход
SPM	JM	Jumps	Перейти, если результат < 0
SPMZ	JMZ	Jumps	Перейти, если результат < =0
SPN	JN	Jumps	Перейти, если результат >< 0
SPO	JO	Jumps	Перейти при OV = 1
SPP	JP	Jumps	Перейти, если результат >0
SPPZ	JPZ	Jumps	Перейти, если результат >=0
SPS	JOS	Jumps	Перейти при OS = 1
SPU	JUO	Jumps	Перейти, если результат недействителен
SPZ	JZ	Jumps	Перейти, если результат =0
SQR	SQR	Floating point Instruction	Вычисление квадрата числа с плавающей точкой (32-бита)
SQRT	SQRT	Floating point Instruction	Вычисление квадратного корня числа с плавающей точкой (32-бита)
SRD	SRD	Shift/Rotate	Сдвинуть вправо двойное слово (32 бита)
SRW	SRW	Shift/Rotate	Сдвинуть вправо слово (16 бит)
SS	SS	Timers	Таймер задержки включения с запоминанием
SSD	SSD	Shift/Rotate	Сдвинуть двойное целое число со знаком (32 бита)
SSI	SSI	Shift/Rotate	Сдвинуть целое число со знаком (16 бит)
SV	SE	Timers	Таймер удлиненного импульса
T	T	Load/Transfer	Передать
T STW	T STW	Load/Transfer	Передача содержимого ACCU 1 в слово состояния
TAD	CAD	Convert	Изменить последовательность байтов в аккумуляторе 1 (32 бита)
TAK	TAK	Accumulator	Обменять аккумулятор 1 с аккумулятором 2
TAN	TAN	Floating point Instruction	Вычисление тангенса числа с плавающей точкой (32 бита, IEEE FP)
TAR	CAR	Load/Transfer	Обменять адресный регистр 1 с адресным регистром 2
TAR1	TAR1	Load/Transfer	Передать адресный регистр 1 в ACCU 1

Немецкая мнемоника	Английская мнемоника	Каталог программных элементов	Описание
TAR1	TAR1	Load/Transfer	Передать адресный регистр1 в целевую область (32-битовый указатель)
TAR1	TAR1	Load/Transfer	Передать адресный регистр1 в адресный регистр2
TAR2	TAR2	Load/Transfer	Передать адресный регистр 2 в ACCU 1
TAR2	TAR2	Load/Transfer	Передать адресный регистр 2 в целевую область (32-битовый указатель)
TAW	CAW	Convert	Изменить последовательность байтов в ACCU 1-L (16-бит)
TDB	CDB	Convert	Поменять местами регистры глобального блока данных и экземплярного блока данных
TRUNC	TRUNC	Convert	Округлить до целого отбрасыванием дробной части
U	A	Bit logic Instruction	И
U(	A(	Bit logic Instruction	И с открытием скобки
UC	UC	Program control	Безусловный вызов
UD	AD	Word logic Instruction	Поразрядное И с двойными словами (32 бита)
UN	AN	Bit logic Instruction	И-НЕ
UN(	AN(	Bit logic Instruction	И-НЕ с открытием скобки
UW	AW	Word logic Instruction	Поразрядное И со словами (16 бит)
X	X	Bit logic Instruction	Исключающее ИЛИ
X(	X(	Bit logic Instruction	Исключающее ИЛИ с открытием скобки
XN	XN	Bit logic Instruction	Исключающее ИЛИ-НЕ
XN(	XN(	Bit logic Instruction	Исключающее ИЛИ-НЕ с открытием скобки
XOD	XOD	Word logic Instruction	Поразрядное Исключающее ИЛИ с двойными словами (32 бита)
XOW	XOW	Word logic Instruction	Поразрядное Исключающее ИЛИ со словами (16 бит)
ZR	CD	Counters	Счетчик обратного счета
ZV	CU	Counters	Счетчик прямого счета

A.2 STL инструкции в алфавитном порядке английской мнемоники (International)

Английская мнемоника	Немецкая мнемоника	Каталог программных элементов	Описание
+	+	Integer math Instruction	Сложение констант целого типа (16, 32бит)
=	=	Bit logic Instruction	Присвоение
)	)	Bit logic Instruction	Закрывающая скобка

Английская мнемоника	**Немецкая мнемоника**	**Каталог программных элементов**	**Описание**
+AR1	+AR1	Accumulator	Сложение адресного регистра 1 (AR1) с ACCU 1
+AR2	+AR2	Accumulator	Сложение адресного регистра2 (AR2) с ACCU 1
+D	+D	Integer math Instruction	Сложение ACCU 1 и ACCU 2 как двойных целых чисел (32-бита)
–D	–D	Integer math Instruction	Вычитание ACCU 1 из ACCU 2 в формате двойных целых чисел (32-бита)
*D	*D	Integer math Instruction	Умножение ACCU 1 и ACCU 2 как двойных целых чисел(32-бита)
/D	/D	Integer math Instruction	Деление ACCU 2 на ACCU 1 как двойных целых чисел (32-бита)
? D	? D	Compare	Сравнение двойных целых чисел (32-бита)==, <>, >, <, >=, <=
+I	+I	Integer math Instruction	Сложение ACCU 1 и ACCU 2 как целых чисел (16-бит)
–I	–I	Integer math Instruction	Вычитание ACCU 1 из ACCU 2 в формате целых чисел (16-бит)
*I	*I	Integer math Instruction	Умножение ACCU 1 и ACCU 2 как целых чисел (16-бит)
/I	/I	Integer math Instruction	Деление ACCU 2 на ACCU 1 в формате целых чисел (16-бит)
? I	? I	Compare	Сравнение в формате целых чисел (16-bit) ==, <>, >, <, >=, <=
+R	+R	Floating point Instruction	Сложение ACCU 1 и ACCU 2 как чисел с плавающей точкой (32-бита IEEE-FP)
–R	–R	Floating point Instruction	Вычитание ACCU 1 из ACCU 2 в формате чисел с плавающей точкой (32-бита IEEE-FP)
*R	*R	Floating point Instruction	Умножение ACCU 1 и ACCU 2 как чисел с плавающей точкой (32-бита IEEE-FP)
/R	/R	Floating point Instruction	Умножение ACCU 2 на ACCU 1 в формате чисел с плавающей точкой (32-бита IEEE-FP)
? R	? R	Compare	Сравнение чисел с плавающей точкой (32-бита) ==, <>, >, <, >=, <=
A	U	Bit logic Instruction	И
A(	U(	Bit logic Instruction	И с открывающей скобкой
ABS	ABS	Floating point Instruction	Модуль числа с плавающей точкой (32-бита IEEE-FP)
ACOS	ACOS	Floating point Instruction	Вычисление арккосинуса чисел с плавающей точкой (32-бита)
AD	UD	Word logic Instruction	Поразрядное И над двойным словом (32-бита)
AN	UN	Bit logic Instruction	И-НЕ
AN(	UN(	Bit logic Instruction	И-НЕ с открывающей скобкой
ASIN	ASIN	Floating point Instruction	Вычисление арксинуса чисел с плавающей точкой (32-бита)
ATAN	ATAN	Floating point Instruction	Вычисление арктангенса чисел с плавающей точкой (32-бита)

Английская мнемоника	**Немецкая мнемоника**	**Каталог программных элементов**	**Описание**
AW	UW	Word logic Instruction	Поразрядное И над словом (16-бит)
BE	BE	Program control	Конец блока
BEC	BEB	Program control	Условный конец блока
BEU	BEA	Program control	Безусловный конец блока
BLD	BLD	Program control	Инструкция отображения программы
BTD	BTD	Convert	Преобразование BCD в Double Integer (32-бита)
BTI	BTI	Convert	Преобразование BCD в Integer (16-бит)
CAD	TAD	Convert	Изменение последовательности байтов в аккумуляторе 1 (32 бита)
CALL	CALL	Program control	Вызов блока
CALL	CALL	Program control	Вызов мультиэкземпляра
CALL	CALL	Program control	Вызов блока из библиотеки
CAR	TAR	Load/Transfer	Обменять адресный регистр 1 с адресным регистром 2
CAW	TAW	Convert	Изменить последовательность байтов в ACCU1-L (16-бит)
CC	CC	Program control	Условный вызов
CD	ZR	Counters	Уменьшение счетчика
CDB	TDB	Convert	Поменять местами регистры глобального блока данных и экземплярного блока данных
CLR	CLR	Bit logic Instruction	Сброс RLO (=0)
COS	COS	Floating point Instruction	Вычисление косинуса чисел с плавающей точкой (32-бита)
CU	ZV	Counters	Увеличение счетчика
DEC	DEC	Accumulator	Декремент ACCU 1-L-L
DTB	DTB	Convert	Преобразование Double Integer (32-бита) в BCD
DTR	DTR	Convert	Преобразование Double Integer (32-бита) в число с плавающей точкой (32-бита IEEE-FP)
ENT	ENT	Accumulator	Ввод в стек ACCU
EXP	EXP	Floating point Instruction	Вычисление экспоненты чисел с плавающей точкой (32-бита)
FN	FN	Bit logic Instruction	Выделение отрицательного фронта
FP	FP	Bit logic Instruction	Выделение положительного фронта
FR	FR	Counters	Деблокировка счетчика (от C 0 до C 255)
FR	FR	Timers	Деблокировка таймера
INC	INC	Accumulator	Инкремент ACCU 1-L-L
INVD	INVD	Convert	Инверсия двойного целого числа (32-бита)
INVI	INVI	Convert	Инверсия целого числа (16-бит)
ITB	ITB	Convert	Преобразование Integer (16-бит) в BCD
ITD	ITD	Convert	Преобразование Integer (16-бит) в Double Integer (32-бита)

Английская мнемоника	**Немецкая мнемоника**	**Каталог программных элементов**	**Описание**
JBI	SPBI	Jumps	Перейти при BR = 1
JC	SPB	Jumps	Перейти, при RLO = 1
JCB	SPBB	Jumps	Перейти при RLO = 1 с сохранением в BR бите
JCN	SPBN	Jumps	Перейти при RLO = 0
JL	SPL	Jumps	Распределенный переход
JM	SPM	Jumps	Перейти, если результат < 0
JMZ	SPMZ	Jumps	Перейти, если результат < =0
JN	SPN	Jumps	Перейти, если результат >< 0
JNB	SPBNB	Jumps	Перейти при RLO = 0 с сохрангением в BR бите
JNBI	SPBIN	Jumps	Перейти при BR = 0
JO	SPO	Jumps	Перейти при OV = 1
JOS	SPS	Jumps	Перейти при OS = 1
JP	SPP	Jumps	Перейти, если результат >0
JPZ	SPPZ	Jumps	Перейти, если результат >=0
JU	SPA	Jumps	Безусловный переход
JUO	SPU	Jumps	Перейти, если результат недействителен
JZ	SPZ	Jumps	Переход при нуле
L	L	Load/Transfer	Загрузка
L DBLG	L DBLG	Load/Transfer	Загрузка длины глобального DB в ACCU 1
L DBNO	L DBNO	Load/Transfer	Загрузка номера глобального DB в ACCU 1
L DILG	L DILG	Load/Transfer	Загрузка длины экземплярногоDB в ACCU 1
L DINO	L DINO	Load/Transfer	Загрузка номера экземплярногоDB в ACCU 1
L STW	L STW	Load/Transfer	Загрузка слова состояния в ACCU 1
L	L	Timers	Загрузка текущего значения таймера в ACCU 1 в двоичном коде (номер таймера должен быть в пределах от 0 до 255, например, L T 32)
L	L	Counters	Загрузка текущего значения счетчика в ACCU 1 в двоичном коде (номер счетчика должен быть в пределах от 0 до 255, например, L C 15)
LAR1	LAR1	Load/Transfer	Загрузка в адресный регистр1 значения из ACCU 1
LAR1 <D>	LAR1<D>	Load/Transfer	Загрузка в адресный регистр1 двойного целого (32-битовый указатель)
LAR1 AR2	LAR1 AR2	Load/Transfer	Загрузка в адресный регистр1 значения из адресного регистра 2
LAR2	LAR2	Load/Transfer	Загрузка в адресный регистр 2 значения из ACCU 1
LAR2 <D>	LAR2 <D>	Load/Transfer	Загрузка в адресный регистр2 двойного целого (32-битовый указатель)

Английская мнемоника	**Немецкая мнемоника**	**Каталог программных элементов**	**Описание**
LC	LC	Counters	Загрузка текущего значения счетчика в ACCU 1 в BCD (номер счетчика должен быть в пределах от 0 до 255, например, LC C 15)
LC	LC	Timers	Загрузка текущего значения таймера в ACCU 1 в двоичном коде (номер таймера должен быть в пределах от 0 до 255, например, LC T 32)
LEAVE	LEAVE	Accumulator	Вывод из стека аккумуляторов
LN	LN	Floating point Instruction	Вычисление натурального логарифма чисел с плавающей точкой (32-бита)
LOOP	LOOP	Jumps	Инструкция цикла
MCR(	MCR(	Program control	Сохранение RLO в MCR стеке, Начало MCR зоны
)MCR	)MCR	Program control	Конец MCR зоны
MCRA	MCRA	Program control	Активация MCR области
MCRD	MCRD	Program control	Деактивация MCR области
MOD	MOD	Integer math Instruction	Выделение остатка от деления двойных целых чисел (32-бита)
NEGD	NEGD	Convert	Инверсия знака двойных целых чисел (32-бит)
NEGI	NEGI	Convert	Инверсия знака целых чисел (16-бит)
NEGR	NEGR	Convert	Инверсия знака чисел с плавающей точкой (32-бита, IEEE-FP)
NOP 0	NOP 0	Accumulator	Нулевая инструкция
NOP 1	NOP 1	Accumulator	Нулевая инструкция
NOT	NOT	Bit logic Instruction	Инверсия RLO
O	O	Bit logic Instruction	ИЛИ
O(	O(	Bit logic Instruction	ИЛИ с открытием скобки
OD	OD	Word logic Instruction	Побитовое ИЛИ над двойными словами (32-бита)
ON	ON	Bit logic Instruction	ИЛИ-НЕ
ON(	ON(	Bit logic Instruction	ИЛИ-НЕ с открытием скобки
OPN	AUF	DB call	Открыть блок данных
OW	OW	Word logic Instruction	Побитовое ИЛИ над словами (16-бит)
POP	POP	Accumulator	Аккумулятор 1 <– Аккумулятор 2, Аккумулятор 2 <– Аккумулятор 3, Аккумулятор 3 <– Аккумулятор 4 (CPU с четырьмя аккумуляторами)
POP	POP	Accumulator	Аккумулятор 1 <– Аккумулятор 2 (CPU с двумя аккумуляторами)
PUSH	PUSH	Accumulator	Аккумулятор 3 –> Аккумулятор 4, Аккумулятор 2 –> Аккумулятор 3, Аккумулятор 1 –> Аккумулятор 2 (CPU с четырьмя аккумуляторами)

Английская мнемоника	**Немецкая мнемоника**	**Каталог программных элементов**	**Описание**
PUSH	PUSH	Accumulator	Аккумулятор 1 –> Аккумулятор 2 (CPU с двумя аккумуляторами)
R	R	Bit logic Instruction	Сброс
R	R	Counters	Сброс счетчика (номер счетчика должен быть в пределах от 0 до 255, например, R C 15)
R	R	Timers	Сброс таймера (номер таймера должен быть в пределах от 0 до 255, например, R T 32)
RLD	RLD	Shift/Rotate	Циклический сдвиг двойного слова влево (32-бита)
RLDA	RLDA	Shift/Rotate	Циклический сдвиг двойного слова влево (32-бита) через CC 1
RND	RND	Convert	Округление
RND–	RND–	Convert	Округление до ближайшего меньшего двойного целого числа
RND+	RND+	Convert	Округление до ближайшего большего двойного целого числа
RRD	RRD	Shift/Rotate	Циклический сдвиг двойного слова вправо (32-бита)
RRDA	RRDA	Shift/Rotate	Циклический сдвиг двойного слова вправо (32-бита) через C 1
S	S	Bit logic Instruction	Установка
S	S	Counters	Установить начальное значение счетчика (где текущий счетчик может иметь номер в диапазоне от 0 до 255, например: S C 15)
SAVE	SAVE	Bit logic Instruction	Сохранить RLO в регистре BR
SD	SE	Timers	Таймер – формирователь задержки включения
SE	SV	Timers	Таймер удлиненного импульса
SET	SET	Bit logic Instruction	Установка RLO в 1
SF	SA	Timers	Таймер – формирователь задержки выключения
SIN	SIN	Floating point Instruction	Синус числа с плавающей точкой (32 бита, IEEE FP)
SLD	SLD	Shift/Rotate	Сдвинуть влево двойное слово (32 бита)
SLW	SLW	Shift/Rotate	Сдвинуть влево слово (16 битов)
SP	SI	Timers	Таймер – формирователь импульса
SQR	SQR	Floating point Instruction	Вычисление квадрата числа с плавающей точкой (32-бита)
SQRT	SQRT	Floating point Instruction	Вычисление квадратного корня числа с плавающей точкой (32-бита)
SRD	SRD	Shift/Rotate	Сдвинуть вправо двойное слово (32 бита)
SRW	SRW	Shift/Rotate	Сдвинуть вправо слово (16 бит)
SS	SS	Timers	Таймер задержки включения с запоминанием

Английская мнемоника	**Немецкая мнемоника**	**Каталог программных элементов**	**Описание**
SSD	SSD	Shift/Rotate	Сдвинуть двойное целое число со знаком (32 бита)
SSI	SSI	Shift/Rotate	Сдвинуть целое число со знаком (16 бит)
T	T	Load/Transfer	Передача
T STW	T STW	Load/Transfer	Передача ACCU 1 в слово состояния
TAK	TAK	Accumulator	Обмен ссодержимым ACCU 1 и ACCU 2
TAN	TAN	Floating point Instruction	Вычисление тангенса числа с плавающей точкой (32 бита, IEEE FP)
TAR1	TAR1	Load/Transfer	Передать адресный регистр 1 в ACCU 1
TAR1	TAR1	Load/Transfer	Передать адресный регистр1 в целевую область (32-битовый указатель)
TAR1	TAR1	Load/Transfer	Передать адресный регистр 1 в адресный регистр 2
TAR2	TAR2	Load/Transfer	Передать адресный регистр 2 в ACCU 1
TAR2	TAR2	Load/Transfer	Передать адресный регистр 2 в целевую область (32-битовый указатель)
TRUNC	TRUNC	Convert	Округлить до целого отбрасыванием дробной части
UC	UC	Program control	Безусловный вызов
X	X	Bit logic Instruction	Исключающее ИЛИ
X(	X(	Bit logic Instruction	Исключающее ИЛИ с открытием скобки
XN	XN	Bit logic Instruction	Исключающее ИЛИ-НЕ
XN(	XN(	Bit logic Instruction	Исключающее ИЛИ-НЕ с открытием скобки
XOD	XOD	Word logic Instruction	Поразрядное Исключающее ИЛИ с двойными словами (32 бита)
XOW	XOW	Word logic Instruction	Поразрядное Исключающее ИЛИ со словами (16 бит)

B Примеры программирования

B.1 Обзор примеров программирования

Практические применения

Каждая STL инструкция выполняет специфическую операцию. С помощью комбинирования различных инструкций в программе , Вы можете создавать широкий спектр программ для различных задач автоматизации. В этой главе приводятся примеры практического применения программ написанных с помощью списка операторов:

- Управление транспортером с использованием битовых логических операций
- Определение направления движения ленты транспортера с использованием битовых логических операций
- Генерация тактовых импульсов с помощью таймерных команд
- Контроль склада с помощью операций счета и сравнения
- Решение задачи с использованием математических инструкций с целыми числами
- Установка интервала времени для нагревания печи

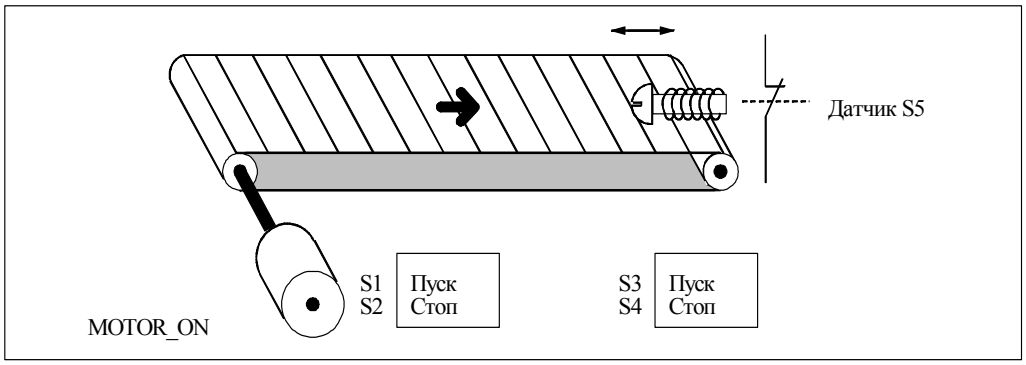
Используемые инструкции

Мнемоника	Каталог программных элементов	Описание
AW	Word logic instruction	Поразрядное И со словами
OW	Word logic instruction	Поразрядное ИЛИ со словами
CD, CU	Counters	Обратный счет ,Прямой счет
S, R	Bit logic instruction	Установка, сброс
NOT	Bit logic instruction	Инверсия RLO
FP	Bit logic instruction	Выделение положительного фронта
+I, /I, *I	Integer instruction	Математические инструкции в формате Integer
>=I, <=I	Compare	Сравнение в формате Integer
A, AN	Bit logic instruction	И, И-НЕ
O, ON	Bit logic instruction	ИЛИ, ИЛИ-НЕ
=	Bit logic instruction	Присвоение
INC	Accumulator	Инкремент аккумулятора 1
BE, BEC	Program Control	Конец блока, условный конец блока
L, T	Load / Transfer	Загрузка и передача
SE	Timers	Таймер удлиненного импульса

B.2 Пример: Битовые логические инструкции

Пример 1: Управление лентой транспортера

На следующем рисунке показана лента транспортера, которая может приводиться в движение с помощью электродвигателя. В начале транспортера имеются две кнопки: S1 для запуска и S2 для останова. В конце транспортера тоже имеются две кнопки: S3 для запуска и S4 для останова. Транспортер можно запускать или останавливать с любого конца. Также датчик S5 останавливает транспортер, когда предмет, находящийся на ленте, достигает конца.



Абсолютное и символьное программирование

Вы можете написать программу для управления лентой транспортера, показанного на рисунке, используя **абсолютные значения** или их **символьные имена**, представляющие различные компоненты конвейера . Вы должны создать таблицу символов для того, чтобы поставить в соответствие выбранным символьным именам абсолютные адреса (см. STEP 7 Online Help).

Компонент системы	Абсолютный адрес	Символ	Таблица символов	
Push Button Start Switch	I 1.1	S1	I 1.1	S1
Push Button Stop Switch	I 1.2	S2	I 1.2	S2
Push Button Start Switch	I 1.3	S3	I 1.3	S3
Push Button Stop Switch	I 1.4	S4	I 1.4	S4
Sensor	I 1.5	S5	I 1.5	S5
Motor	Q 4.0	MOTOR_ON	Q 4.0	MOTOR_ON

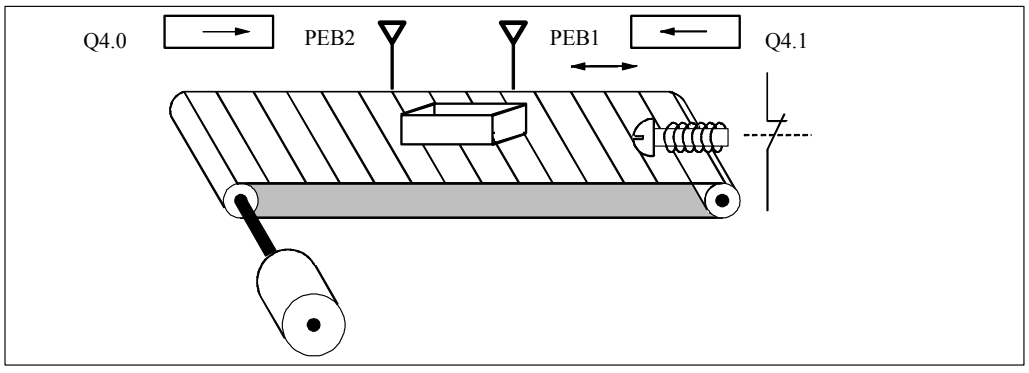
Программа с абсолютными адресами		Программа с символьными именами	
O	I 1.1	O	S1
O	I 1.3	O	S3
S	Q 4.0	S	MOTOR_ON
O	I 1.2	O	S2
O	I 1.4	O	S4
ON	I 1.5	ON	S5
R	Q 4.0	R	MOTOR_ON

Список инструкций для управления конвейером

STL		Комментарий
O	**I 1.1**	**//Нажатие любой кнопки включает конвейер.**
O	**I 1.3**	
S	**Q 4.0**	
O	**I 1.2**	**//Нажатие любой стоповой кнопки или срабатывание нормально закрытого концевика останавливает конвейер.**
O	**I 1.4**	
ON	**I 1.5**	
R	**Q 4.0**	

Пример 2 : Определение направления движения ленты транспортера

На рисунке показана лента транспортера, с двумя фотоэлектрическими датчиками (PEB1 и PEB2), которые служат для определения направления в котором движется пакет, расположенный на ленте.Оба фотобарьера работают как нормальнооткрытые контакты.



Абсолютное и символьное программирование

Вы можете написать программу для определения направления движения транспортера, показанного на рисунке, используя **абсолютные адреса** или их **символьные имена**, представляющие различные компоненты конвейера . Вы должны создать таблицу символов для того, чтобы поставить в соответствие выбранным символьным именам абсолютные адреса (см. STEP 7 Online Help).

Компонент системы	Абсолютный адрес	Символ	Таблица символов
Фотоэлектрический датчик 1	I 0.0	PEB1	I 0.0 PEB1
Фотоэлектрический датчик 2	I 0.1	PEB2	I 0.1 PEB2
Индикатор движения направо	Q 4.0	RIGHT	Q 4.0 RIGHT
Индикатор движения налево	Q 4.1	LEFT	Q 4.1 LEFT
Тактовый меркер 1	M 0.0	PMB1	M 0.0 PMB1
Тактовый меркер 2	M 0.1	PMB2	M 0.1 PMB2

Absolute Program		Symbolic Program	
A	I 0.0	A	PEB1
FP	M 0.0	FP	PMB1
AN	I 0.1	AN	PEB 2
S	Q 4.1	S	LEFT
A	I 0.1	A	PEB 2
FP	M 0.1	FP	PMB 2
AN	I 0.0	AN	PEB 1
S	Q 4.0	S	RIGHT
AN	I 0.0	AN	PEB 1
AN	I 0.1	AN	PEB 2
R	Q 4.0	R	RIGHT
R	Q 4.1	R	LEFT

Statement List

STL		Explanation
A	**I 0.0**	**//При появлении положительного фронта на входе I 0.0 и состоянии сигнала на входе I 0.1 = 0, при наличии упаковки на конвейере, определяется движение влево.**
FP	**M 0.0**	
AN	**I 0.1**	
S	**Q 4.1**	
A	**I 0.1**	**// При появлении положительного фронта на входе I 0.1 и состоянии сигнала на входе I 0.0 = 0, при наличии упаковки на конвейере, определяется движение вправо. Пересечение светового барьера упаковкой дает низкий уровень сигнала на соответствующий вход.**
FP	**M 0.1**	
AN	**I 0.0**	
S	**Q 4.0**	
AN	**I 0.0**	**//Если ни один из световых барьеров не пересечен, это означает отсутствие между ними упаковки и отключает индикацию направления .**
AN	**I 0.1**	
R	**Q 4.0**	
R	**Q 4.1**	

B.3 Пример: Таймерные инструкции

Генератор тактовых импульсов

Для создания периодически повторяющегося сигнала Вы можете использовать генератор тактовых импульсов или импульсное реле. Генераторы тактовых импульсов обычно используются в системах сигнализации, управляющих миганием индикаторных ламп.

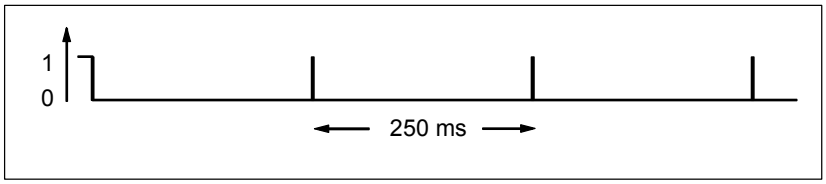
Если Вы используете S7-300, то Вы можете реализовать функцию генератора тактовых импульсов используя вызов программы из специальных организационных блоков, управляемых временем. Пример, показанный в следующей программе STL иллюстрирует использование таймерных функций для генерации тактовых импульсов.

STL программа синхронного генератора импульсов

STL	Комментарий
AN M99.0	**//Если таймер T 1 закончил отсчет,**
L S5T#250ms	**//загрузка значения 250 ms в таймер T 1 и**
SV T1	**//запуск таймера как T 1 удлиненный импульс.**
A T1	
=M99.0	**//Инверсия RLO.**
A M99.0	
BEC	**//Если таймер не закончил отсчет, конец блока.**
L MB100	**//Если таймер закончил отсчет, загрузка содержимого MB100,**
INC 1	**//инкремент на 1,**
T MB100	**//сохранение результата в MB100.**

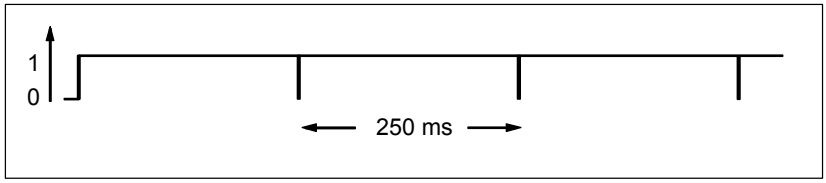
Опрос статуса

Опрос статуса таймера T1 определяет результат логической операции (RLO)



Как только время истекает, таймер перезапускается. Вследствие этого опрос статуса меркера, выполняемый командой AN M99.0, дает RLO "1" лишь кратковременно.

Инвертированный бит РЛО:



Каждые 250 мс бит RLO равен 0. Переход игнорируется, и содержимое меркерного байта MB100 увеличивается на 1.

Содержимое меркерного байта MB100 меняется каждые 250 ms следующим образом:

0 -> 1 -> 2 -> 3 -> ... -> 254 -> 255 -> 0 -> 1 ...

Получение определенной частоты

Из отдельных битов меркерного байта MB100 Вы можете получить следующие частоты:

Биты в MB100	Частота в Гц	Период	
M 100.0	2.0	0.5 s	(250 ms вкл / 250 ms выкл)
M 100.1	1.0	1 s	(0.5 s вкл / 0.5 s выкл)
M 100.2	0.5	2 s	(1 s вкл / 1 s выкл)
M 100.3	0.25	4 s	(2 s вкл / 2 s выкл)
M 100.4	0.125	8 s	(4 s вкл / 4 s выкл)
M 100.5	0.0625	16 s	(8 s вкл / 8 s выкл)
M 100.6	0.03125	32 s	(16 s вкл / 16 s выкл)
M 100.7	0.015625	64 s	(32 s вкл / 32 s выкл)

Программа в STL

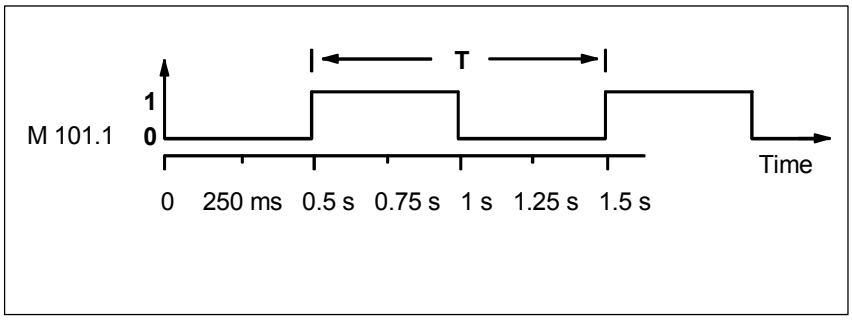
STL		Комментарий
A	**M10.0**	**//M 10.0 = 1 при возникновении ошибки. При этом лампа ошибки Q4.0 мигает с частотой 1 Hz.**
A	**M100.1**	
=	**Q 4.0**	

Состояние отдельных битов меркерного байта MB 100

Цикл	Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	Время в ms
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

Состояние сигнала бита 1 в MB 100 (M 100.1)

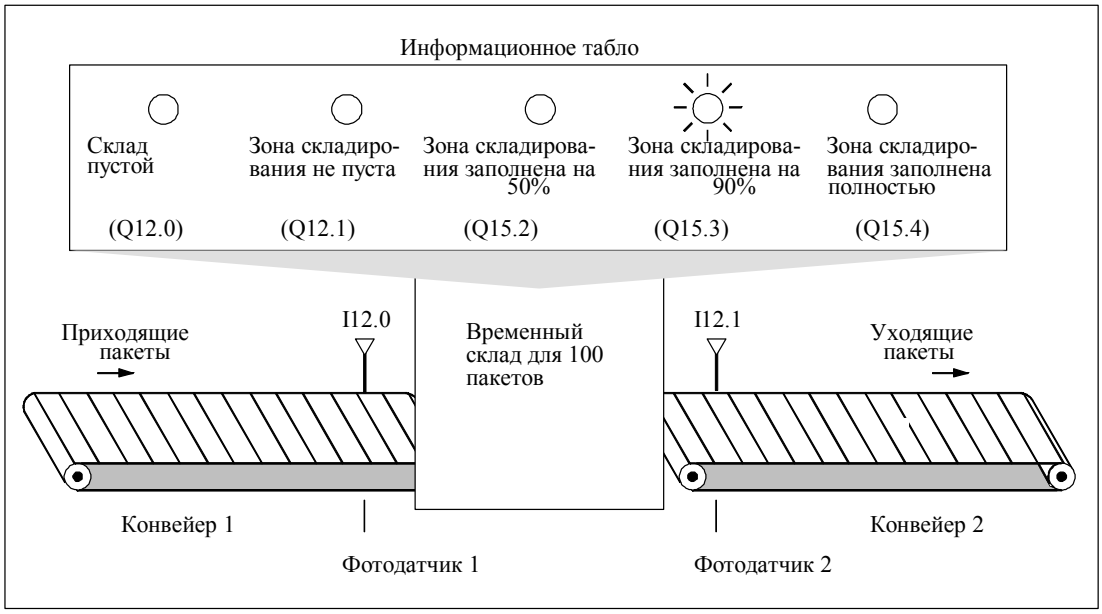
Частота = 1/T = 1/1 s = 1 Hz



B.4 Пример: Инструкции счета и сравнения

Промежуточный склад со счетчиком и компаратором

На следующем рисунке показана система с двумя конвейерами и промежуточным складом между ними. Конвейер 1 доставляет пакеты в зону складирования. Фотоэлектрический датчик в конце конвейера 1 вблизи зоны складирования определяет сколько пакетов доставлено в эту зону. Конвейер 2 транспортирует пакеты из зоны временного складирования к погрузочной площадке, где пакеты грузятся на грузовые автомобили для доставки клиентам. Фотоэлектрический датчик в конце конвейера 2 вблизи зоны складирования определяет, сколько пакетов покидают зону складирования для отправки на погрузочную площадку. Информационное табло с пятью лампами отображает уровень заполнения склада временного хранения.



Программа индикации степени заполнения склада на STL

STL		Комментарий
A	I 0.0	//каждый импульс на световом барьере 1
CU	C1	// увеличивает значение счетчика C 1 на 1, выполняя подсчет числа пакетов, //поступающих на склад временного хранения.
A	I 0.1	// каждый импульс на световом барьере 2
CD	C1	// уменьшает значение счетчика C 1 на 1, выполняя подсчет числа упаковок, //покидающих склад временного хранения.
AN	C1	//При значении счетчика = 0,
=	Q 4.0	//Включается лампа "Склад пустой" .
		//
A	C1	//Если значение счетчика не равно 0,
=	A 4.1	// Включается лампа "Склад не пустой".
		//
L	50	
L	C1	
<=I		//При значении счетчика >= 50,
=	Q 4.2	// Включается лампа "Склад заполнен на 50 %".
		//
L	90	
>=I		// При значении счетчика >= 90,
=	Q 4.3	// Включается лампа "Склад заполнен на 90 %".
		//
L	C1	
L	100	
>=I		// При значении счетчика >= 100
=	Q 4.4	// Включается лампа "Склад заполнен полностью". (Вы можете также использовать выход Q 4.4 для блокировки мотора конвейера 1.)

B.5 Пример: Математические инструкции с целыми числами

Решение математической задачи

Следующий пример программы показывает, как использовать арифметические операции с целыми числами и команды L и T для вычисления результата следующего уравнения:

MD4 = ((IW0 + DBW3) x 15) / MW2

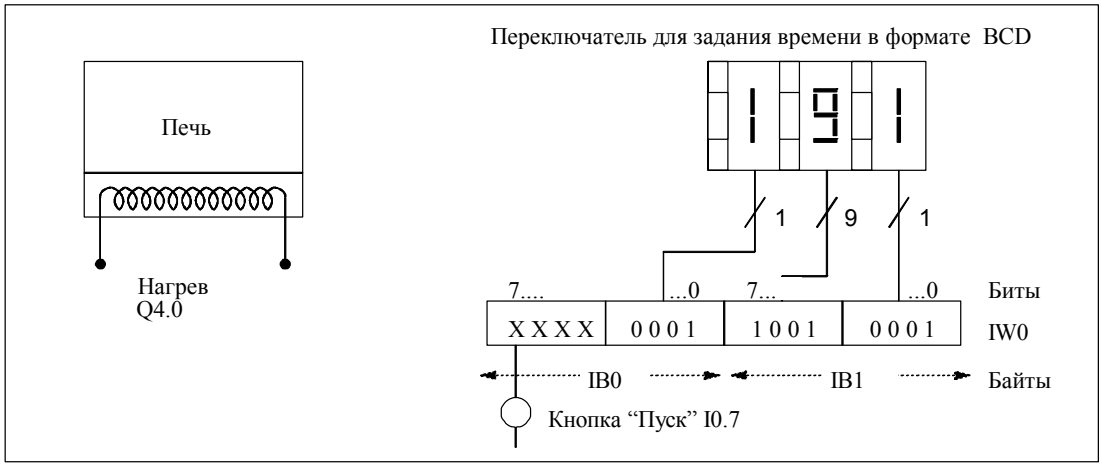
Программа в STL

STL		Комментарий
L	**IW0**	**//Загрузка значения из IW0 в аккумулятор 1.**
L	**DB5.DBW3**	**//Загрузка слова данных DBW3 из глобального блока данных DB5 в //аккумулятор1. Старое содержимое аккумулятора 1 предварительно сдвигается //в аккумулятор 2.**
+I	**I 0.1**	**//Сложение содержимого аккумуляторов 1 и 2. Результат сохраняется в //младшем слове аккумулятора 1. Содержимое аккумулятора 2 и старшее слово //аккумулятора 1 остаются неизменными.**
L	**+15**	**//Загрузка константы +15 в аккумулятор 1. Содержимое аккумулятора 1 //предварительно сдвигается в аккумулятор 2.**
***I**		**// Перемножение содержимого аккумуляторов 1 и 2. Результат сохраняется в //младшем слове аккумулятора 1. Содержимое аккумулятора 2 и старшее слово //аккумулятора 1 остаются неизменными.**
L	**MW2**	**// Загрузка значения из MW2 в аккумулятор 1. Старое содержимое аккумулятора //1 предварительно сдвигается в аккумулятор 2.**
/I		**//Деление содержимого аккумулятора 2 на содержимое аккумулятора 1. //Результат сохраняется в младшем слове аккумулятора 1. Содержимое //аккумулятора 2 и старшее слово аккумулятора 1 остаются неизменными.**
T	**MD4**	**//Передача окончательного результата в двойное слово MD4. Содержимое //аккумуляторов не изменяется.**

B.6 Пример: Поразрядные логические операции со словами

Нагрев печи

Оператор печи начинает нагревание печи нажатием кнопки “Пуск”. Оператор может устанавливать длительность нагрева с помощью цифрового переключателя, показанного на рисунке. Значение, устанавливаемое оператором, задает количество секунд в двоично-десятичном формате (BCD).



Компонент системы	Абсолютный адрес в программе STL
Кнопка “Пуск”	I0.7
Переключатель для единиц	от I1.0 до I1.3
Переключатель для десятков	от I1.4 до I1.7
Переключатель для сотен	от I0.0 до I0.3
Запуск нагревания	Q4.0

Программа в STL

```
STL                 Комментарий
A    T1             //Пока таймер работает,
=    Q 4.0          //работает нагреватель.
BEC                 // Если таймер работает, обработка программы заканчивается. Это
                    //предотвращает повторный запуск таймера T1 во время его работы .
L    IW0
AW   W#16#0FFF      //Маскирование битов с I 0.4 по I 0.7 (сброс их в 0). Значение времени в секундах
                    //находится в младшем слове аккумулятора 1 в BCD коде.
OW   W#16#2000      //Задание базы времени в битах 12 и 13 в младшем слове аккумулятора 1.
A    I 0.7
SE   T1             //Запуск таймера T1 как удлиненного импульса при нажатии пусковой кнопки .
```

C Передача параметров

Формальные параметры блока должны получить для обработки актуальные значения. Функциональные блоки копируют эти значения в экземплярный блок данных, указанный при вызове функционального блока. При работе функций указатель на область фактического значения располагается в локальном стеке вызывающего блока. Перед вызовом функционального блока значения входных параметров копируются в экземплярный блок DB или в L -стек. После окончания обработки блока выходные величины передаются в выходные переменные. Внутри вызываемого блока Вы работаете только с копией фактических параметров. Для этого в вызывающем блоке создаются необходимые STL инструкции, которые скрыты от пользователя.

Примечание

Если меркеры, входы, выходы или периферия используются в качестве фактических адресов функции , они обрабатываются несколько отлично от других адресов. При этом обновление производится напрямую, без L-стека , за исключением входного параметра с типом данных BOOL , где фактическое значение, передается в блок через L -стек.

Предупреждение:

При программировании вызываемого блока, убедитесь, что в выходные параметры, производится запись получаемых результатов. В противном случае на выход может попасть случайная величина! В случае функциональных блоков: значение из экземплярного блока данных, указанного при последнем вызове.При работе с функцией: значение , оказавшееся в локальном стеке.

Учтите следующие моменты:

- Задайте начальные значения для всех выходных параметров, если это возможно.
- Старайтесь не использовать RLO –зависимые инструкции сброса и установки битов. Если RLO имеет значение 0, в результате на выходе может быть получена случайная величина.
- Если Вы программируете переходы внутри блока, убедитесь, что не будут исключены из обработки необходимые Вам инструкции записи выходных параметров. На забудьте об этом при использовании инструкций BEC и MCR.

Предметный указатель

Д

З

И

К

Л

М

Н

О

П

Р

С

Т

У

Ц

A

B

C

D

E

F

I

J

L

M

N

O

P

R

S

T

U

X