

М.П. Кульчавеня

Основы программирования ПЛК

Учебное пособие является вводным курсом по основам программирования ПЛК. В качестве базового контроллера принят Simatic S7-300 фирмы SIEMENS. рассмотрены синтаксис, семантика и принципы программирования на STEP7 - языке программирования промышленных контроллеров SIMATIC S7 фирмы SIEMENS. Описание языка снабжено многочисленными примерами, иллюстрирующими особенности и возможности STEP7. При описании языка и примеров программирования большое внимание уделено основным трудностям и ошибкам, с которыми может столкнуться программист. Учебное пособие предназначено для студентов электротехнических специальностей вузов, изучающих программируемые логические контроллеры. Может быть полезно для студентов и инженеров, осваивающих основы контроллерной техники.

Кульчавеня Михаил Петрович

Основы программирования ПЛК

НОБЕЛЬ
издательство научной литературы **ПРЕСС**

Мы ищем авторов! Nobelpress.ru
Телефон: +7 (495) 221-89-80
E-mail: author@nobelpress.ru

ISBN 9785458543187



9 785458 543187

НОБЕЛЬ
издательство научной литературы **ПРЕСС**



М.П. Кульчавеня

Основы программирования ПЛК

**Москва
Издательство Нобель Пресс**

М11 **М.П. Кульчавеня**
Основы программирования ПЛК / М.П. Кульчавеня – М.: Lennex Corp, — Подготовка макета: Издательство Нобель Пресс, 2023. – 158 с.

ISBN 978-5-458-54318-7

Учебное пособие является вводным курсом по основам программирования ПЛК. В качестве базового контроллера принят Simatic S7-300 фирмы SIEMENS. рассмотрены синтаксис, семантика и принципы программирования на STEP7 - языке программирования промышленных контроллеров SIMATIC S7 фирмы SIEMENS. Описание языка снабжено многочисленными примерами, иллюстрирующими особенности и возможности STEP7. При описании языка и примеров программирования большое внимание уделено основным трудностям и ошибкам, с которыми может столкнуться программист. Учебное пособие предназначено для студентов электротехнических специальностей вузов, изучающих программируемые логические контроллеры. Может быть полезно для студентов и инженеров, осваивающих основы контроллерной техники.

Предисловие

Данное учебное пособие предназначено для студентов ВУЗов, изучающих курс «Программирование средств автоматизации» и инженеров, занимающихся программированием ПЛК. При этом полагается, что читатели знакомы с физическими основами электроники, полупроводниковыми приборами и цифровыми элементами. Автор попытался изложить материал таким образом, чтобы при его последовательном изучении у читателей сложилось представление о работе ПЛК, их функционировании и способах их программирования.

В качестве базового ПЛК для изучения принят ПЛК Simatic S7-300 фирмы SIEMENS, широко распространенный в наше время. На базе данного ПЛК строятся достаточно сложные автоматизированные контроллерные системы.

В данном учебном пособии подробно рассмотрены синтаксис, семантика и принципы программирования на STEP7 - языке программирования промышленных контроллеров SIMATIC S7 фирмы SIEMENS. Описание языка снабжено многочисленными примерами, иллюстрирующими особенности и возможности STEP7. При описании языка и примеров программирования большое внимание уделено основным трудностям и ошибкам, с которыми может столкнуться программист.

Список принятых сокращений

ПЛК/PLC – Программируемый логический контроллер.
ЦПУ/CPU – Центральное процессорное устройство.
«0» – Логический ноль.
«1» – Логическая единица.
Н.О. Контакт – Нормально разомкнутый (открытый) контакт.
Н.З. Контакт – Нормально замкнутый контакт.
Q – Катушка, реле.
ВК (SQ) – Выключатель концевой.
ЯП – Ячейка памяти.
Ассu1 – Аккумулятор 1.
Ассu1L – Младшее слово первого аккумулятора.
ПКМ – Нажатие правой кнопкой мыши.

1. ОБЩИЕ СВЕДЕНИЯ О ПЛК SIEMENS

1.1 Обзор аппаратной части ПЛК

На сегодняшний день среди ПЛК фирмы Siemens большой популярностью пользуются PLC S7-200, S7-300 и S7-400.

S7-200 - модульный программируемый контроллер, предназначенный для решения задач наиболее низкого уровня производительности. Программируется программным средством Step7 MicroWin, в данной книге не рассматривается.

S7-300 - модульный программируемый контроллер (до 32 модулей), предназначенный для построения систем автоматизации низкой и средней степени сложности.

S7-400 - модульный программируемый контроллер (до 32 узлов по MPI-сети), предназначенный для построения систем автоматизации среднего и верхнего уровня сложности

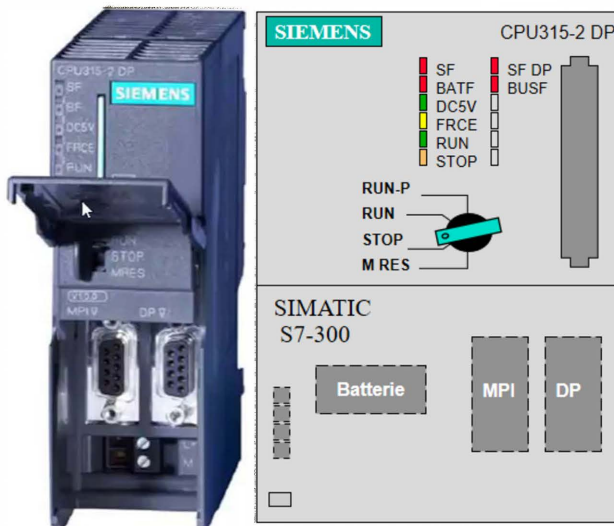
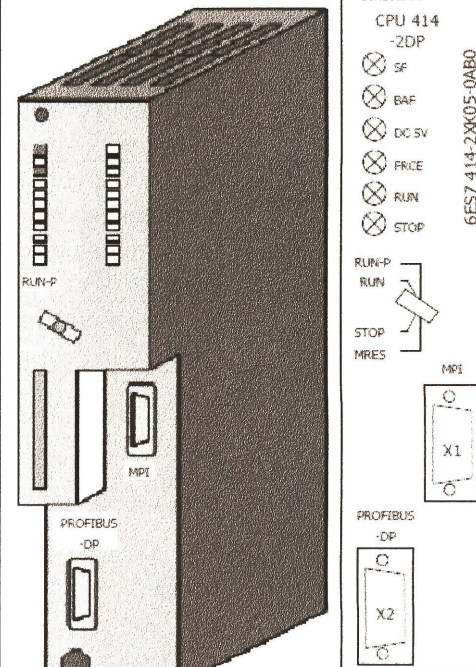
Внешний вид и графическое изображение CPU315-2DP	Внешний вид и графическое изображение CPU 414
	

Таблица 1.1: Внешний вид и графические изображения CPU S7-300 и S7-400

На передней панели CPU могут находиться: переключатель режимов работы, индикаторы состояния, карта памяти, отсек батареи, соединитель MPI, интерфейс DP, другие интерфейсы.

Переключатель режимов	
MRES	Сброс CPU (Module RESet)
STOP	Режим останова; программа не выполняется.
RUN	Программа выполняется, возможно только чтение информации из CPU в программатор
RUN-P	Программа выполняется, доступны чтение и запись информации из программатора
Индикаторы	
SF	Summary Fault, групповая ошибка; внутренняя неисправность CPU или неисправность в модулях с возможностями диагностики
BATF	Ошибка батареи; батарея разряжена или отсутствует
DC5V	Индикация внутреннего постоянного напряжения 5 V
FRCE	Показывает, что один или несколько входов или выходов находятся в режиме FORCE
RUN	Мигает при запуске CPU, светится постоянно в рабочем режиме
STOP	Светится постоянно в режиме останова. Медленно мигает, когда требуется сброс памяти Быстро мигает, когда выполняется сброс памяти Медленно мигает, когда сброс памяти необходим, при включении карты памяти
Карта памяти	Слот для установки карты памяти. Карта памяти сохраняет программу при отключении от сети при отсутствии батареи
Отсек батареи	Место для установки литиевой батареи под крышкой. Батарея поддерживает состояние ОЗУ при отсутствии напряжения
Соединитель MPI	Соединение с устройством программирования или другим устройством с интерфейсом MPI
Интерфейс DP	Интерфейс для прямого подключения распределенной периферии к CPU

Таблица 1.2: Элементы передней панели CPU S7-300

Так как контроллеры S7-300 и S7-400 являются модульными – к ним можно подсоединять дополнительные модули. Подсоединение модулей к CPU S7-300 осуществляется посредством шинного соединителя, к S7-400 через стойку (rack).

Всего существует 3 вида стоек S7-400:

- UR – Universal Rack – Универсальная задняя шина обмена данными, допускается использовать с любыми моделями S7-400.
- CR – Central Rack – Центральная задняя шина обмена данными, не допускается использование принимающих IM
- ER – Expansion Rack – Задняя шина обмена данными, необходима для расширения базовой структуры, возможно использование только SM модулей

PS	Power Supply – блок питания
CPU	Central processing unit – центральный процессор, ЦПУ
IM	Interface Module – интерфейсный модуль. Предназначены для соединения шины одного ряда с шиной другого ряда при многорядной конфигурации.
SM DI	Signal Module Digital Input – сигнальный модуль цифровых входов
SM DO	Signal Module Digital Output – сигнальный модуль цифровых выходов
SM AI	Signal Module Analog Input – сигнальный модуль аналоговых входов
SM AO	Signal Module Analog Output – сигнальный модуль аналоговых выходов
FM	Functional Module – функциональный модуль (счетчики, управление и т.д.)
CP	Communication Processor – коммуникационный процессор. Предназначен для обеспечения работы сети (MPI/PPI/Industrial Ethernet)

Таблица 1.3: Модули CPU S7-300/400

Модули в стойке располагаются в следующей последовательности:

	1	2	3	4	5	6	7	8	9	10
S7-300	PS	CPU	IM	SM DI	SM DO	SM AI	SM AO	FM	CP	
S7-400	PS	CPU	SM DI	SM DO	SM AI	SM AO	CP	FM	SM	IM

Таблица 1.4: Последовательность расположения модулей S7-300 и S7-400

Модель CPU, как правило, выбирается на основании выполняемой им задачи и технико-экономического обоснования.

Технические параметры нескольких моделей CPU приведены в таблице. Более подробную техническую информацию можно найти на сайте производителя и в каталогах.

CPU	314	315-2DP	318-2DP	413-3	416-2	417-4
Время выполнения, мкс						
Двоичная инструкция	0.3-0.6	0.3-0.6	0.1	0.1	0.08	0.1
Word-инструкция	1.2	1.0	0.1	0.1	0.08	0.1
Integer (+/-)	2.0	2.0	0.1	0.1	0.08	0.1
Real (+/-)	50.0	50.0	0.6	0.6	0.48	0.6
Память пользователя						
Рабочая память	24 кб	64 кб	512 кб	2x384кб	2x0.8Mb	2x2Mb
Внутр.загр.память	40 кб	96 кб	64 кб	256 кб	256 кб	256 кб
Внешн. загр.память	4 Мб	4 Мб	4 Мб	64 Мб	64 Мб	64 Мб
Адресное пространство						
Память меркеров, бит	2048	2048	8192	8кб	16кб	16кб
Тактовые меркеры	8	8	8	8	8	8
Таймеры	128	128	512	256	512	512
Счетчики	64	64	512	256	512	512
Число блоков						
FB	128	192	1024	1024	2048	6144
FC	128	192	1024	1024	2048	6144
DB	127	255	2047	1023	4096	8191
Область отображения, каждая, байт						
Входы/выходы	128	128	256	8кб	16кб	16кб
Макс.размер области I/O в байтах	768	1024	8192	8кб	16кб	16кб
Встроенные интерфейсы	MPI	MPI, DP	MPI, DP	2 DP	MPI/DP, DP	MPI/DP, 3DP

Таблица 1.5: Технические параметры CPU

1.1.1 Виды перезапусков CPU

В общем случае CPU имеет три вида перезапуска:

1. Warm restart – теплый перезапуск. Поддерживается абсолютно всеми моделями CPU S7-300 и S7-400. Как правило, выбран по умолчанию. При теплом запуске значения выбранных ячеек памяти остаются без изменений после нового запуска. Однако, данные, отнесенные к неперманентной (неохраняемой) области памяти стираются. При данном виде перезапуска CPU однократно вызывает организационный блок OB100 перед началом обработки основной программы. При его отсутствии продолжает выполнять сканирование программы с точки прерывания ее выполнения в последний раз.

2. Cold restart – холодный перезапуск. Может не поддерживаться моделью CPU. При повторном перезапуске и перманентные и неперманентные данные стираются. При холодном перезапуске CPU вызывает организационный блок OB102. При отсутствии блока 102 CPU немедленно начинает выполнять цикл сканирования основной программы.

3. Hot restart – горячий перезапуск, может не поддерживаться моделью CPU. При повторном запуске все данные остаются без изменения. При данном виде перезапуска CPU вызывает организационный блок OB101. При отсутствии блока 101 CPU немедленно начинает выполнять цикл сканирования основной программы.

У ряда моделей CPU S7-400 вид запуска выставляется на аппаратном уровне – переключателем на лицевой панели.

Настройка видов перезапуска осуществляется в следующей последовательности:

- 1) Запустить утилиту HW-Config
- 2) Выделить CPU, нажать правую кнопку мыши и выбрать пункт object properties
- 3) Открыть вкладку Startup
- 4) В поле startup after power on будет находиться выбор перезапуска.

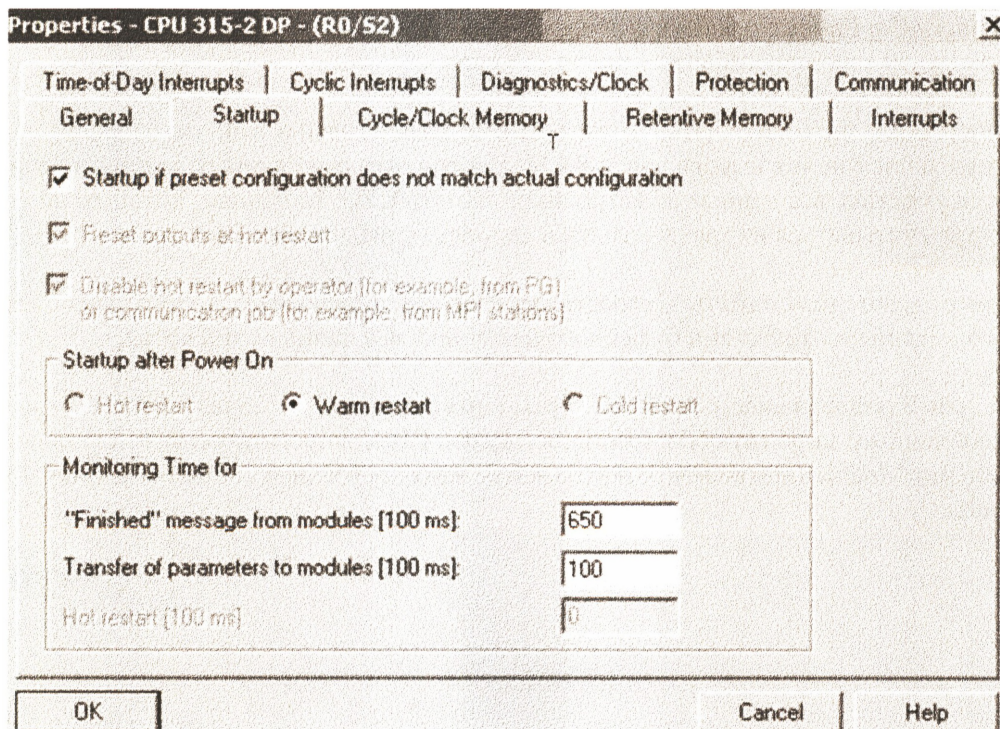


Рисунок 1: Вкладка Startup окна свойств CPU

1.2 Карты памяти

На сегодняшний день фирмой Siemens выпускается 2 типа карт для контроллеров S7-300/400: MC и MMC.

MC (Memory Card) – карты длинного исполнения. Ранее использовались в CPU S7-300, до сих пор используются в CPU S7-400.

Несмотря на одинаковое название и обозначение карт в S7-400 они длиннее. Карту памяти, предназначенную для S7-400 можно использовать в S7-300, но она будет сильно выпирать наружу.

На карте указывается серия, объем и тип (flash или RAM).



Рисунок 1.2: Карты памяти: MC длинного исполнения, MC короткого исполнения, MMC

Также производится еще один вид карт памяти:

MMC – Micro Memory Card и MMC – Multi Media Card. Внешне они выглядят одинаково. Multi media card используется в ОП-панелях и могут быть заменены на обычную SD-карту. Micro Memory Card имеет тип flash и используется только в S7-300. Стоимость MMC-карты на 64кб около 40 евро, на 2Мбайта – около 258 евро.

Flash – энергонезависимая область памяти. Минус flash памяти в том, что скорость считывания/записи не так высока, как в RAM. И в том, что если данных очень много и не получается уместить все данные во встроенной памяти CPU, то данные, которым не хватило места, будут считываться из карты памяти и скорость считывания будет ниже.

RAM – энергозависимая память. Скорость считывания/записи выше, но при выключении основного питания и отсутствии буферного источника все данные стираются.

Максимальный поддерживаемый объем карты памяти зависит от модели CPU. Чтобы его узнать необходимо: запустить HW-Config, выделить CPU, запустить меню PLC строки меню, выбрать пункт Module Information, открыть в нем вкладку Memory и нажать кнопку details memory area

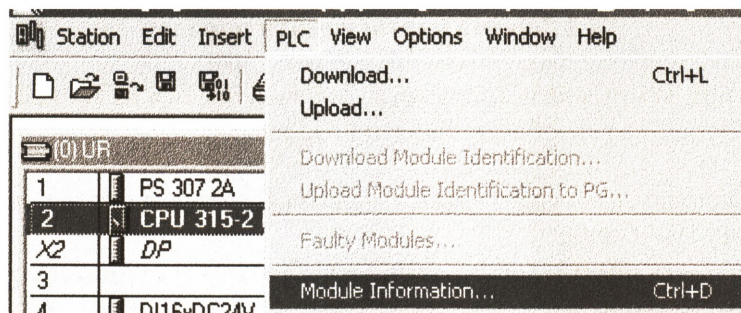


Рисунок 1.2.1: Вкладка Module Information

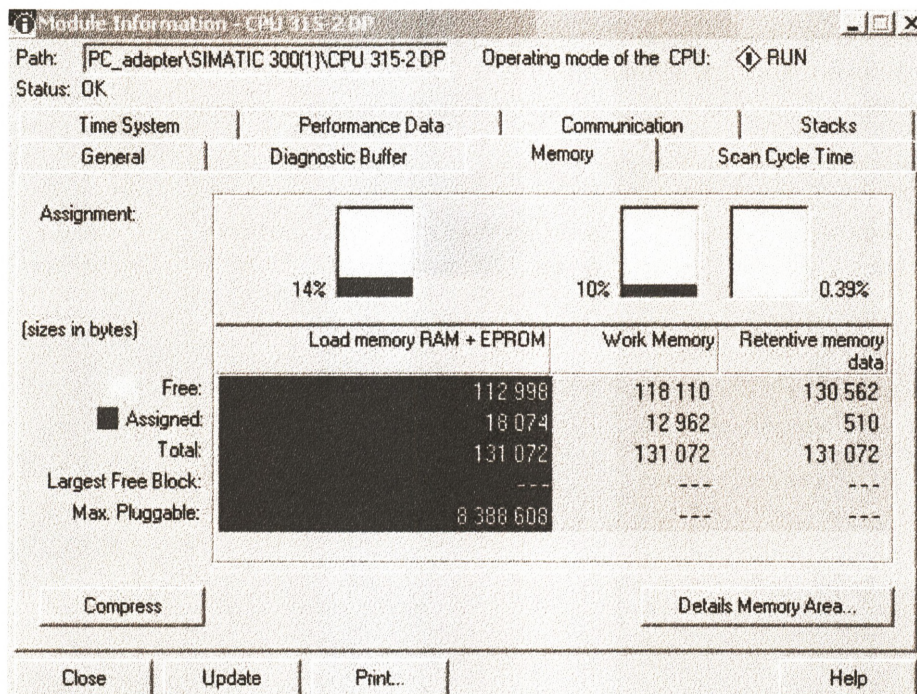


Рисунок 1.2.2: Вкладка Memory

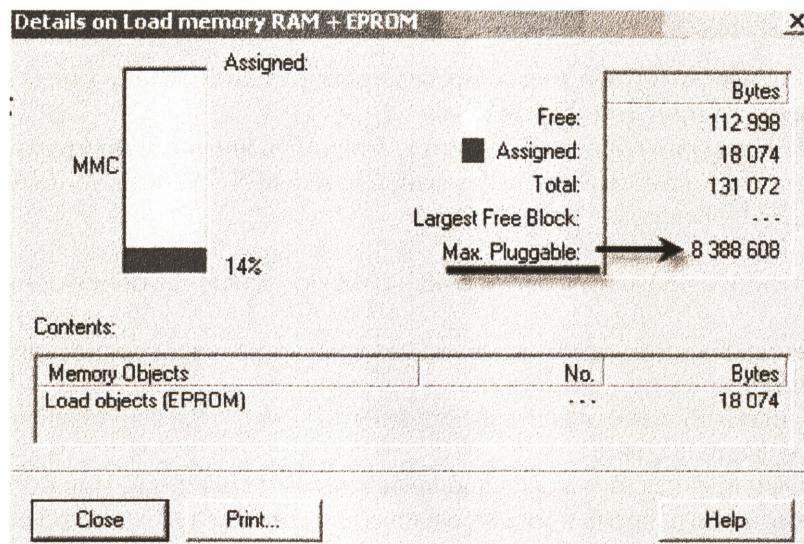


Рисунок 1.2.3: Окно Details Memory Area

Теоретически, CPU может работать без карты памяти, но в таком случае, при сбое питания сотрется вся пользовательская программа.

1.2.1 Защита карты памяти паролем

В Simatic Manager предусмотрена возможность установки пароля на данные, содержащиеся на карте памяти. Точнее, на всю область памяти Load memory.

Настройка видов перезапуска осуществляется в следующей последовательности:

1. Запустить утилиту HW-Config
2. Выделить CPU, нажать правую кнопку мыши и выбрать пункт object properties
3. Открыть вкладку Protection
4. Во вкладке Protection Level выбрать уровень защиты паролем:
 - No Protection – нет защиты паролем
 - Write Protection – установить пароль на запись данных. То есть без пароля будет невозможно записать данные на PLC.
 - Write/Read Protection – защита паролем от считывания и записи.

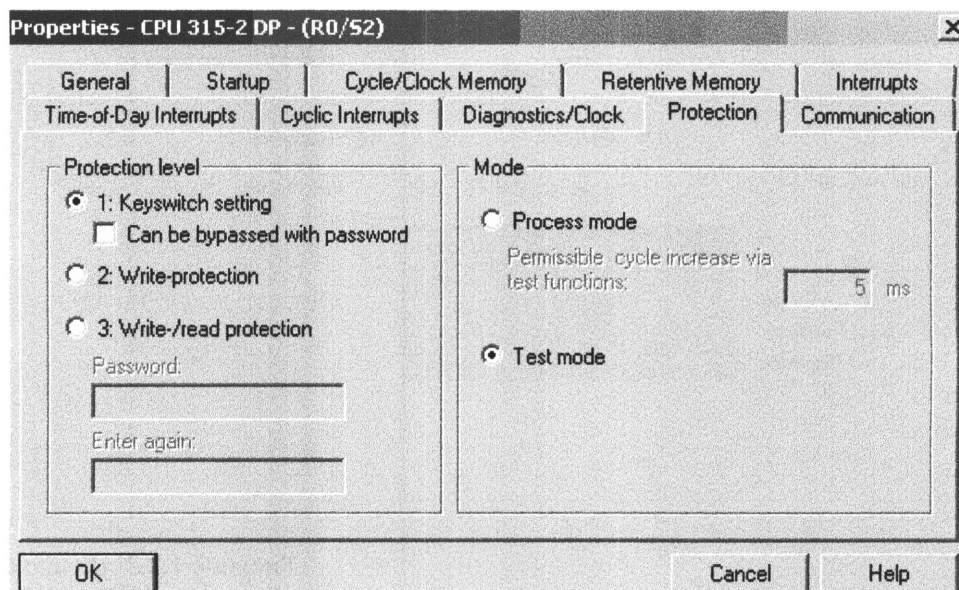



Рисунок 1.2.1.1: Вкладка Protection окна свойств CPU

Сброс забытого пароля возможен только посредством форматирования карты памяти, но и все данные с нее будут при этом уничтожены.

Форматирование карты памяти осуществляется, например, внешним программатором EPROM модулей и карт памяти Siemens (заказной номер 6ES7792-0AA00-0XA0, цена на август 2012 года 943.40 евро).

При его подключении к программатору в Simatic Manager станет активной кнопка  Memory Card.

Вкладка Mode к паролю отношения не имеет, так как настраивает режим отладки программы и содержит следующие пункты:

Test Mode – обычная, базовая отладка с инструментом «очки». Расходует мало ресурсов CPU, не сильно функциональная.

Process Mode – расширенная отладка с дополнительными функциями. При выборе данной опции необходимо указать время в миллисекундах, на которое необходимо увеличить время исполнения программного цикла. Этот интервал времени входит в общее время цикла, но заданная доля будет расходоваться на отладку.

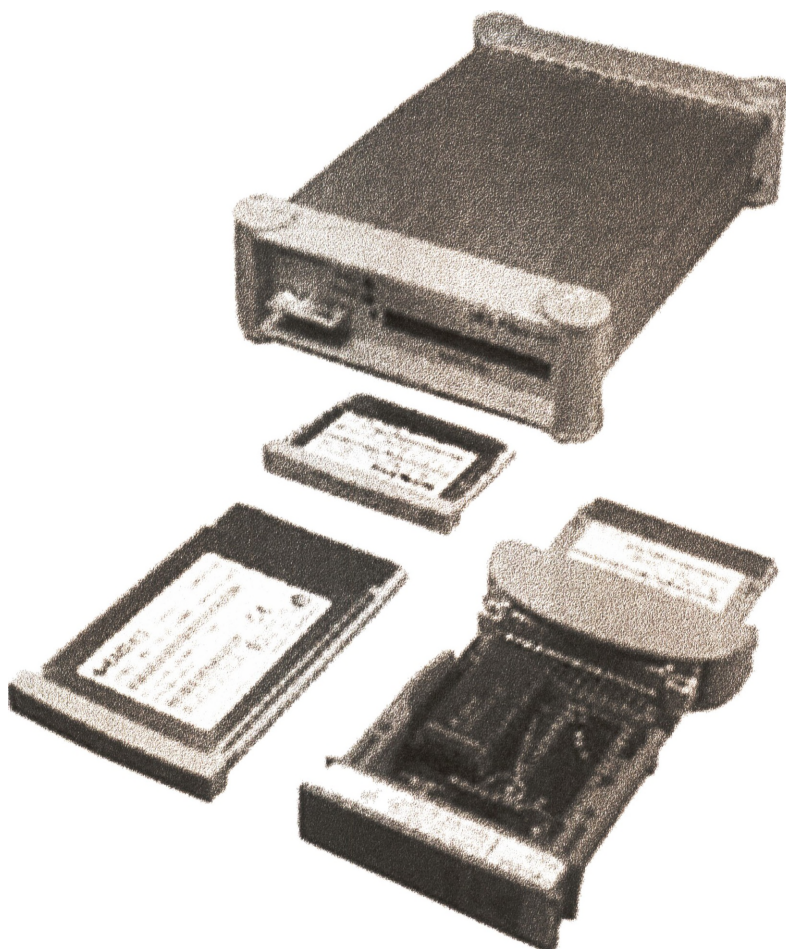


Рисунок 1.2.1.2: Внешний программатор EPROM модулей и карт памяти

1.3 Концепция памяти ПЛК

Память в PLC делится на три основные области: область загрузки (Load Memory), рабочая область памяти (Work Memory) и системную область памяти (System Memory).

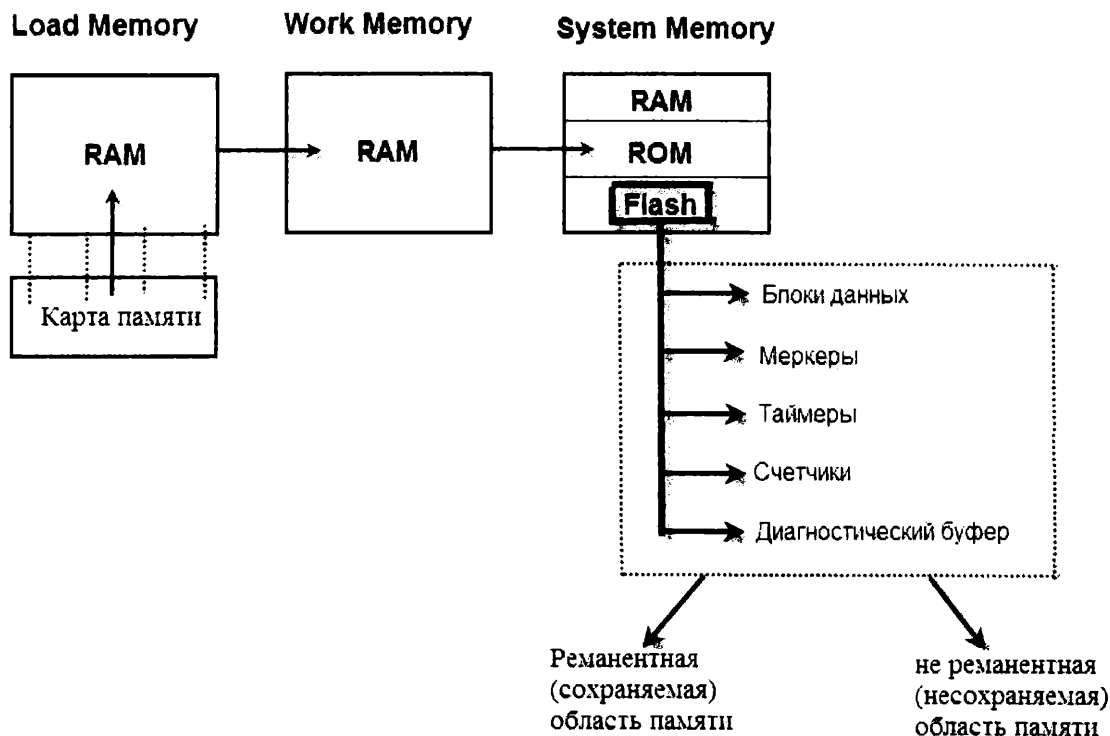


Рисунок 1.3.1: Организация памяти ПЛК

1. Load Memory – область загрузки, имеет тип RAM (встроенная, энергонезависимая, быстрая).

В ней располагаются все данные, которые пользователь загружает в CPU, то есть данные об аппаратной части и все программные блоки.

Расширением области памяти загрузки является внешняя карта памяти.

2. Work Memory – рабочая память, имеет тип RAM, является аналогом ОЗУ компьютера. В рабочей области памяти происходит исполнение программы.

3. System Memory – системная память. Данная область памяти делится на 3 части: RAM, ROM и FLASH

3.1 RAM-область системной памяти содержит локальные данные, точнее – стековые области памяти (скобочный стек - nesting stack, локальный стек – local stack, стек прерываний – interrupts stack и стек вызова – B-stack).

3.2 ROM-область системной памяти содержит операционную систему CPU. Является энергонезависимой, однократнозаписываемой областью памяти.

3.3 Flash-область системной памяти содержит энергозависимые разделы области памяти CPU: область памяти блоков данных, меркеров, аппаратных таймеров, аппаратных счетчиков и диагностический буфер.

Ячейки этих областей памяти можно разделить на 2 области: реманентную (сохраняемую) и нереманентную (неохраняемую) при перезапуске CPU

1.3.1 Настройка ретанентной (сохраняемой) памяти:

Настройка сохраняемой области памяти осуществляется в следующей последовательности:

- 1) запустить утилиту HW-Config
- 2) Выделить CPU, нажать правую кнопку мыши и выбрать пункт object properties
- 3) Открыть вкладку Retentive Memory
- 4) В поле Retentive можно указать количество меркерных байт, аппаратных таймеров и аппаратных счетчиков, которые необходимо отнести к ретанентной (сохраняемой) памяти. В поле Areas можно указать диапазоны блоков данных, которые необходимо отнести к сохраняемой области памяти, если данная функция поддерживается текущей моделью CPU.

Properties - CPU 315-2 DP - (R0/S2)

Time-of-Day Interrupts | Cyclic Interrupts | Diagnostics/Clock | Protection | Communication
General | Startup | Cycle/Clock Memory | Retentive Memory | Interrupts

Retentivity

Number of memory bytes starting with MBO: 16

Number of S7 timers starting with T0: 0

Number of S7 counters starting with C0: 8

Areas

	DB No.	Byte Address	Number of Bytes
Retentive Area 1:	1	0	0
Retentive Area 2:	1	0	0
Retentive Area 3:	1	0	0
Retentive Area 4:	1	0	0
Retentive Area 5:	1	0	0
Retentive Area 6:	1	0	0
Retentive Area 7:	1	0	0
Retentive Area 8:	1	0	0

OK Cancel Help

Рисунок 1.3.1.1: Вкладка Retentive Memory окна свойств CPU

1.4 Влияние внешних воздействий на работу CPU

Изменение положения переключателей на лицевой панели любых CPU S7-300 и S7-400, отключение питания и извлечение карты памяти, однозначно влияют на работу CPU.

1. Выключение напряжения источника:

1.1 CPU с картой памяти RAM – при отсутствии буферного источника питания – сотрутся все данные из RAM-областей памяти. Останется только ROM и сохраняемая часть flash-памяти.

1.2 CPU с картой памяти flash – стираются все данные из RAM-областей памяти, на карте данные остаются.

Скорость повторного включения после отключения питания выше у CPU с буферным источником питания и RAM-картой памяти по сравнению со скоростью повторного включения CPU с flash-картой памяти, так как в первом случае от буферного источника питания питаются все RAM-области памяти, а во втором случае необходимо осуществить первоначальное копирование.

2. Увод CPU в режим STOP: данные всех областей памяти сохраняются, но останавливается исполнение логики

3. Увод CPU в MRES – общий сброс памяти, то есть сброс всех энергозависимых областей памяти.

3.1 CPU с картой памяти RAM – все данные сотрутся

3.2 CPU с картой памяти flash – данные останутся на карте памяти.

Извлечение карты памяти из CPU рекомендуется осуществлять, при отсутствии напряжения на стойке. Если напряжение со стойки снять невозможно – рекомендуется сначала увести CPU в режим STOP и затем извлечь карту памяти. При этом важно, чтобы в момент извлечения карты памяти не осуществлялся обмен данными между CPU и программатором.

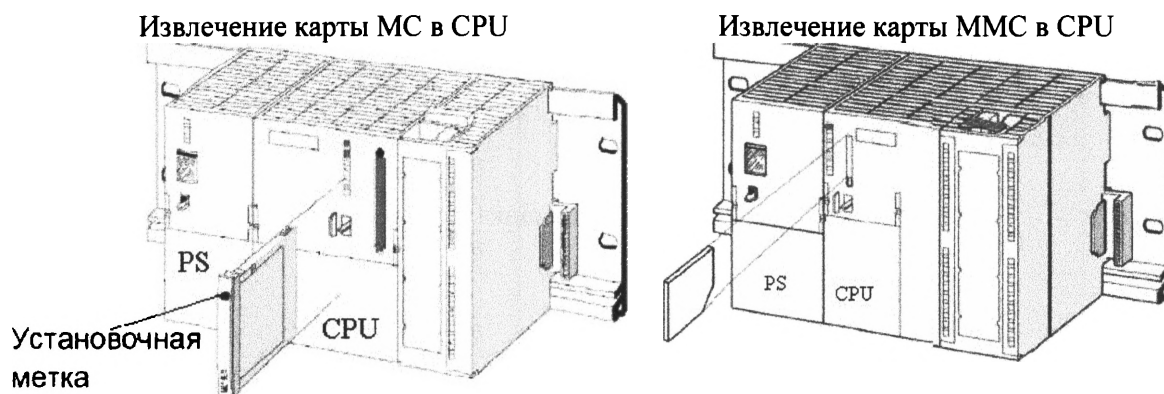


Рисунок 1.4.1: Замена карт памяти PLC S7-300

1.5 Адаптер для связи ПЛК с ПК

Для программирования логического контроллера необходимо установить связь между ним и программатором (компьютером).

Для этого используется или специальная плата CP56-11 или USB-Adapter 6ES7972-0CB20-0XA0

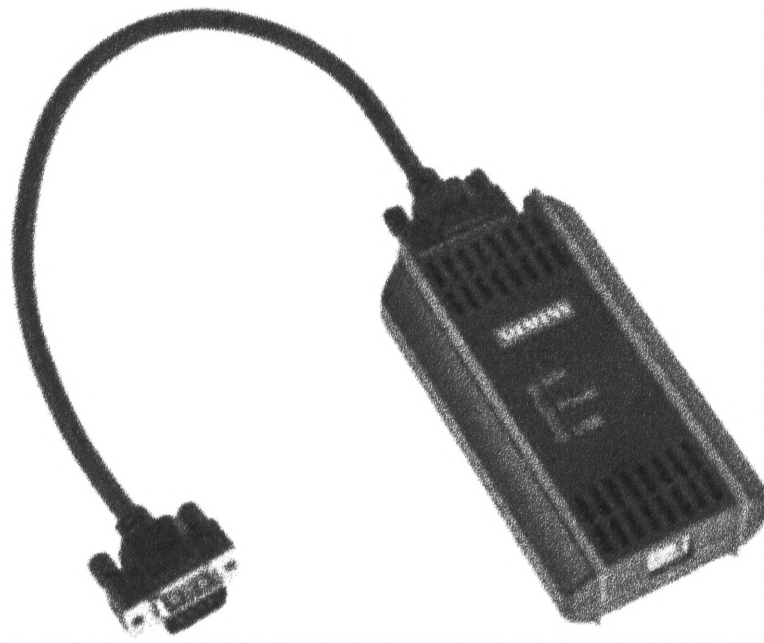


Рисунок 1.5.1: Адаптер 6ES7972-0CB20-0XA0

Данный адаптер, как и все программное обеспечение можно заказать через интернет на сайте <http://iadt.siemens.ru> или у других поставщиков.

Стоимость адаптера 6ES7972-0CB20-0XA0 в августе 2012 года составляла приблизительно 330 Евро.

Также существуют китайские аналоги данного изделия стоимостью порядка 3000 рублей, использование которых возможно только на свой страх и риск.

При отсутствии необходимости подключать реальный контроллер к компьютеру можно обойтись программой имитации контроллера (например-PLCSim).

2. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ SIMATIC MANAGER

2.1 Описание Simatic Manager

Основной программой для работы с ПЛК S7-300/400 фирмы Siemens является Simatic Manager. Дистрибутив данной программы с пробной лицензией на 14 дней встречается на многих интернет-сайтах в свободном доступе и иногда идет в комплекте с учебными пособиями. Также, его можно официально заказать у сименса через интернет по заказному номеру 6ES7810-5CC1-0YA7. Срок поставки около трех недель, цена – около 27 Евро. Полноценная лицензионная версия Simatic S7, Step7 prof, плавающая лицензия на 1 пользователя, ПО разработки, ПО и документация на DVD (6ES7810-5CC11-0YA5) стоит порядка 3000 Евро.

Также можно приобрести весь пакет программ PCS7, но, это будет намного дороже.

Существует несколько версий программы:

- Step 7 Lite (ранее назывался Step 7 mini) — дешёвая и ограниченная по возможностям версия Step 7. Работает с контроллерами Simatic S7-300, Simatic C7, ET 200S (IM 151/CPU и IM 151/CPU FO) и ET 200X (BM 147/CPU) не позволяет реализовать сетевые задачи.
- Step 7 Professional — пакет Step 7 дополненный опциональными пакетами: языками SCL и GRAPH 7, программой имитации контроллера PLCSim. В таком комплекте программное обеспечение наиболее соответствует стандарту IEC (МЭК) 61131 для программируемых логических контроллеров
- В составе PCS 7 с множеством опциональных программ и библиотек

Что касается системных требований к компьютеру, то они следующие:

Минимальные:

- 32-ух разрядная операционная система (Windows XP/Seven/Server 2003/Server2008)
- процессор Pentium 4, 1.7 ГГц;
- оперативная память емкостью 1Гбайт;
- графика 1024x 768 точек.

Рекомендуемые

- процессор Core Duo, 2 ГГц или более мощный;
- оперативная память емкостью 2 Гбайт или выше;
- графика 1280x 1024 точки или выше.

2.2 Установка Simatic Manager

Процесс установки Simatic Manager схож с процессом установки большинства приложений Windows. Однако при установке целесообразно следовать следующим рекомендациям:

1. Установку лучше производить не на системный диск
2. Необходимо помнить, что при удалении Simatic Manager в системе и в реестре может остаться много лишних файлов и записей.
3. Необходимо помнить, что при установке всего пакета PCS-7, в состав которого входит Simatic Manager установится множество дополнительных программ, в результате чего система будет работать медленнее.
4. Лучше осуществлять полную установку Simatic Manager со всеми языковыми пакетами и утилитами.
5. Русского языка программы нет, и русификации программы тоже нет. На просторах Интернета существует лишь русифицированная справка к программе.

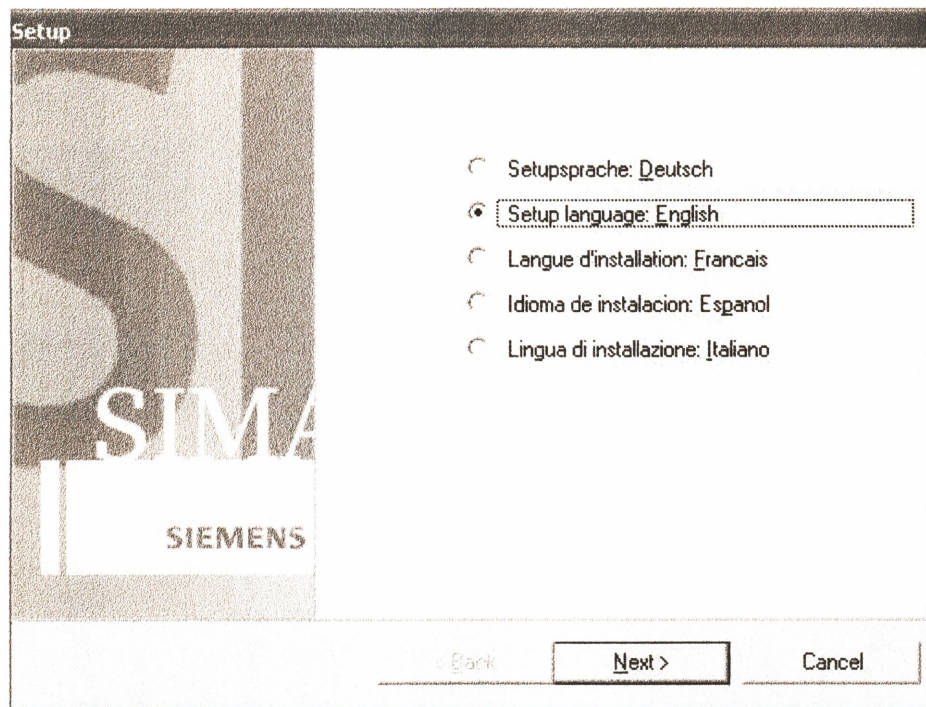


Рисунок 2.2.1: Процесс установки

2.3 Запуск Simatic Manager

Главной утилитой Step7 является Simatic Manager. После установки Step7 на рабочем столе создается ярлык Simatic Manager. Также, он создается в папке Simatic, подменю программы, меню пуск.

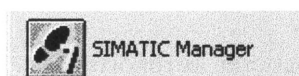


Рисунок 2.3.1: Значок утилиты Simatic Manager

Для запуска приложения необходимо дважды нажать левой кнопкой мыши по иконке. При первом запуске приложения помимо окна программы запустится мастер создания проектов Step7 Wizard New Project. Его можно закрыть.

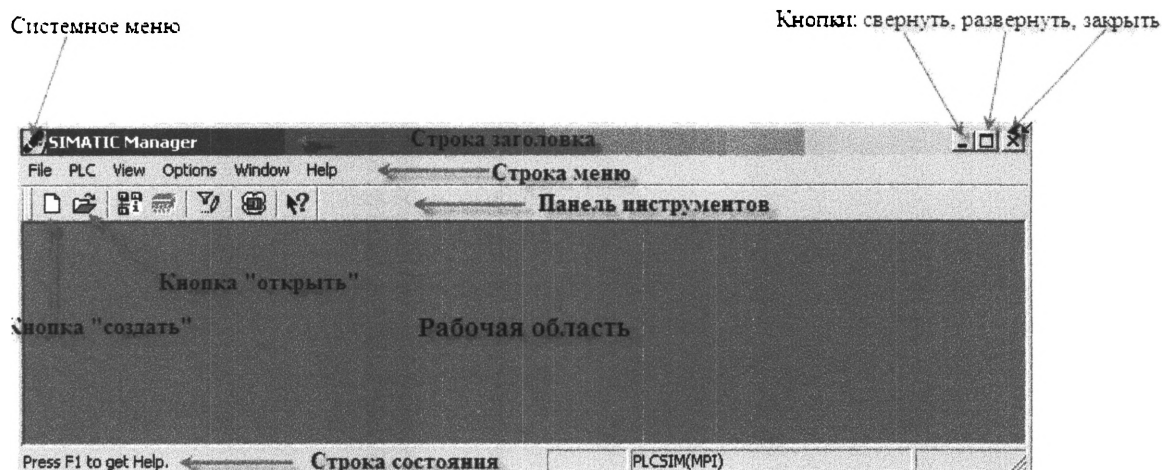


Рисунок 2.3.2: Меню и панель инструментов Simatic Manager

Окно программы, как и большинство Windows-приложений состоит из строки заголовка, строки меню, панели инструментов и рабочей области.

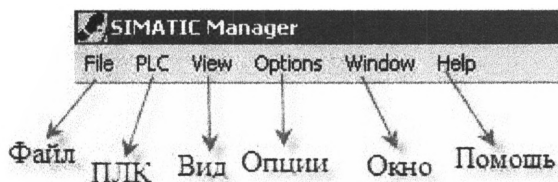


Рисунок 2.3.3: Строка меню

К сожалению, а может быть, и к счастью, приложение не русифицировано. И полноценного русификатора для Simatic Manager на август 2012 года не существовало. В большинстве случаев, русскоязычные пользователи используют английский интерфейс программы.

Рассмотрим отдельные элементы из строки меню приложения:

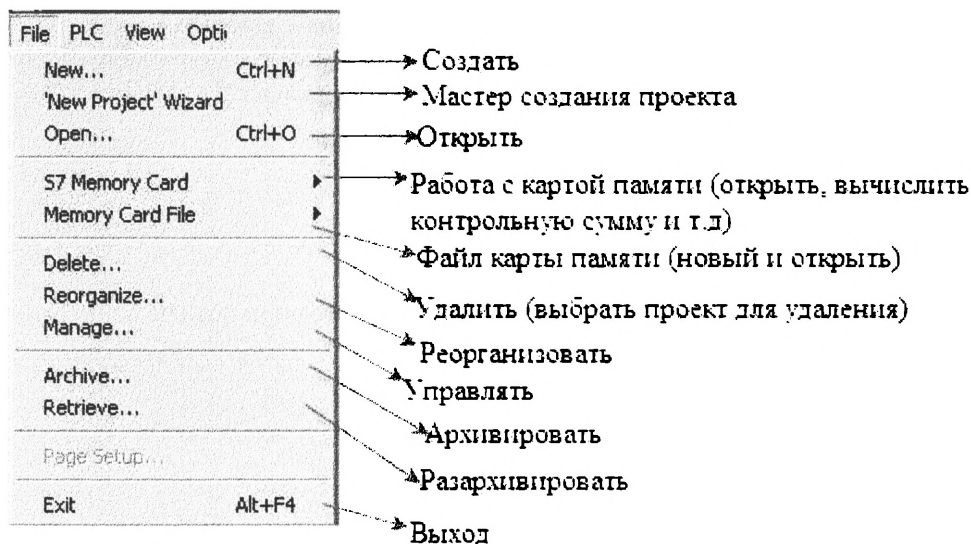


Рисунок 2.3.4: Меню File строки меню приложения

Меню File содержит опции для работы с проектами и картой памяти.

Меню Options содержит опции для настройки приложения (общие настройки, запуск симулятора, настройки интерфейса связи ПЛК-ПК)

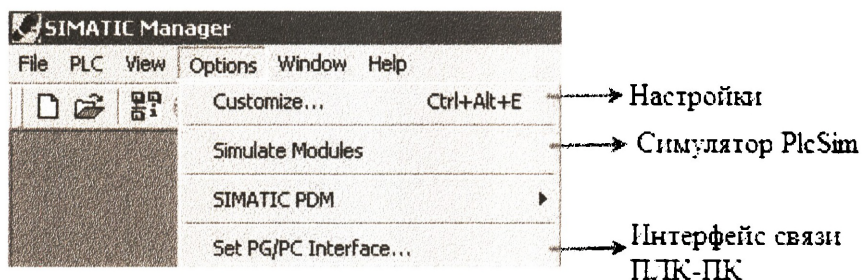


Рисунок 2.3.5: Меню Options строки меню приложения

2.4 Общие настройки проекта (Options-Customize)

При выборе пункта Options-Customize откроется окно, содержащее 8 вкладок:

- General – общие настройки
- Language –языковые настройки
- Date and time of Day – настройки даты и времени
- View –настройки отображения (вида)
- Columns –настройки элементов меню
- Message numbers –настройка сообщений. Используется при обмене данными со станцией верхнего уровня
- Archiving –настройка архивирования

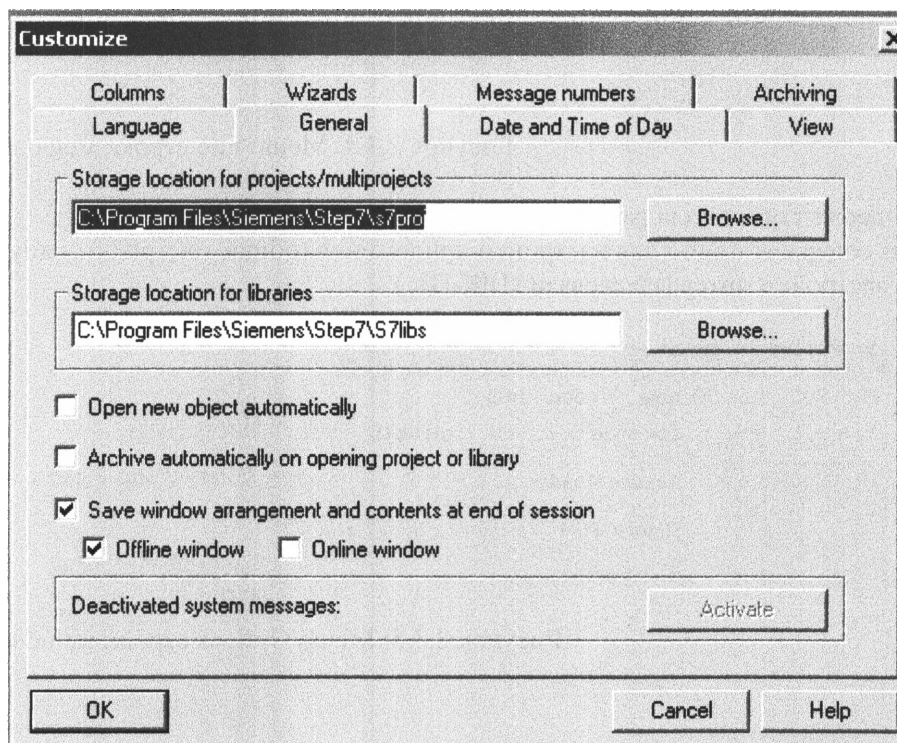


Рисунок 2.4.1: Вкладка General окна настройки приложения

- Storage location for projects/Multiprojects – путь хранения проектов/мультипроектов на жестком диске
- Storage location for libraries – путь хранения библиотек на жестком диске
- Open new object automatically –автоматически открывать новый объект
- Archive automatically on opening project or library –автоматически архивировать открытый проект или библиотеку
- Save window arrangement and contents at end of session –сохранять расположение окон и содержимого в конце сессии
- Offline window – окно offline
- Online window – окно online
- Activate (напротив Deactivated system messages) – активировать деактивированные сообщения. То есть отмена опции do not display this message (не показывать это сообщение) системных сообщений.

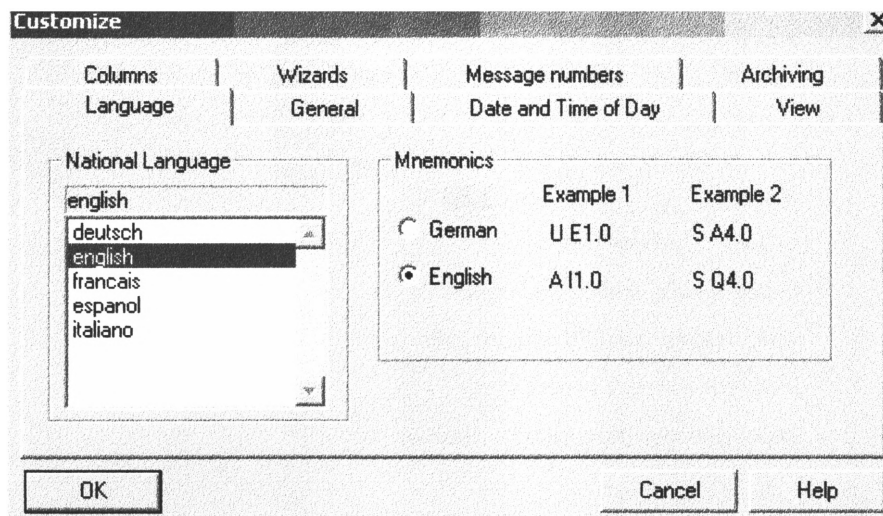


Рисунок 2.4.2: Вкладка Language окна настройки приложения

- National language – выбор языка интерфейса. При изменении языка интерфейса необходим перезапуск приложения.
- Mnemonics – настройка мнемоник то есть буквенных обозначений логических элементов. В большинстве случаев, русскоязычные пользователи используют английскую мнемонику.

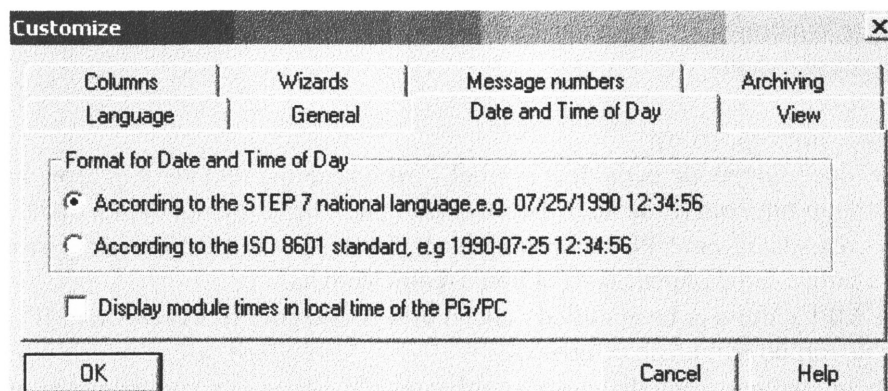


Рисунок 2.4.3: вкладка Date and Time of Day окна настройки приложения

- Format for Date and Time of Day – формат отображения даты и времени.
- According to STEP 7 national language – в соответствии с выбранным языком интерфейса.
- According to ISO 8601 standard – в соответствии со стандартом ISO 8601.
- Display Module Times in the Local Time for Your PG/PC – отображать время модулей по местному времени (установленному в настройках времени программатора)

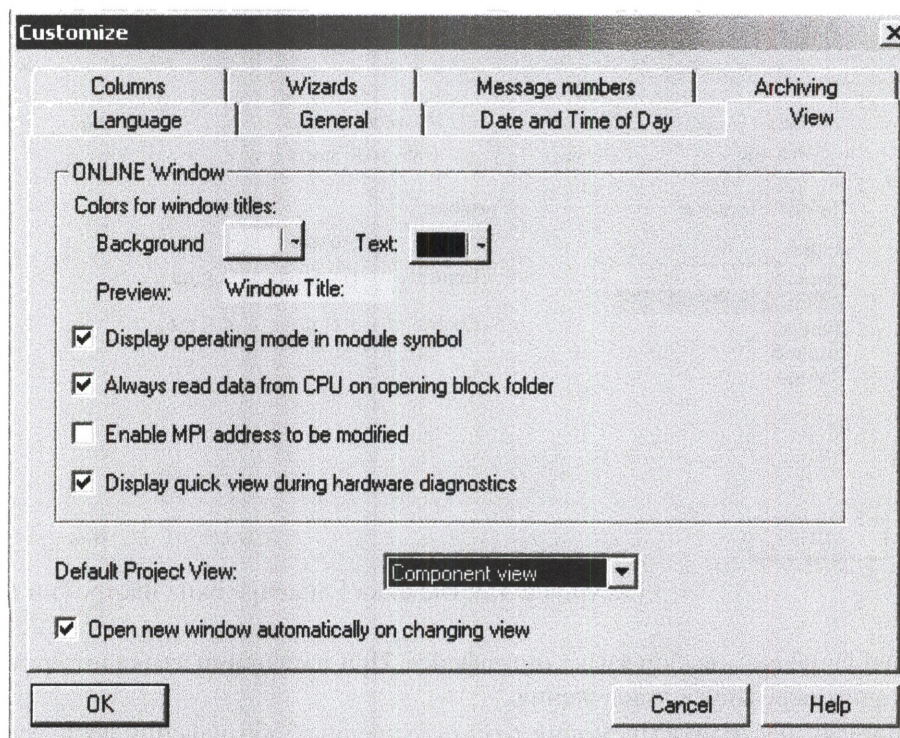


Рисунок 2.4.5: Вкладка View

- ONLINE Window – окно онлайн-программы (при установке связи с PLC).
- Color for Window titles – цвет заголовков окон.
- Background – фон.
- Text – текст
- Preview – предпросмотр
- Display operating mode in module symbol – отображение текущего состояния модуля (stop/startup/run/hold и так далее) символом при просмотре проекта в онлайн.
- Always read data from CPU on opening block folder – всякий раз при открытии папки block в online-окне определяются при чтении модуля.
- Enable MPI address to be modified – включить возможность изменения MPI-адреса. Данную опцию лучше не включать.
- Display quick view when diagnosing hardware – Включения возможности выбора отображения детальной или обычной аппаратной диагностики в утилите HW Config.
- Default Project View – вид проекта по умолчанию. Возможны два варианта: component view и plant view. По умолчанию установлен первый. Второй бывает необходим при использовании дополнительных программных компонентов.
- Open new window automatically on changing view – автоматически открывать новое окно при изменении формата отображения.

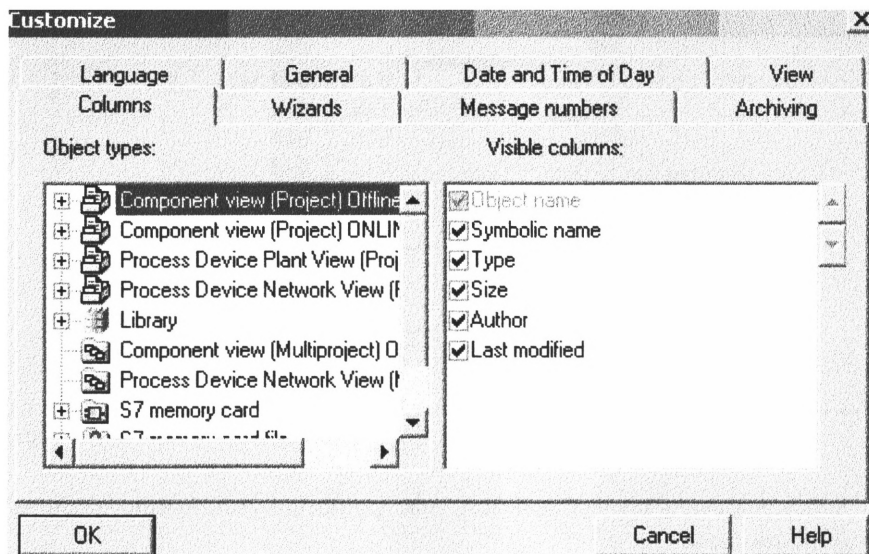


Рисунок 2.4.6: Вкладка Columns

- Object types – тип объекта.
- Visible Columns –видимые столбцы.
- Background – фон.

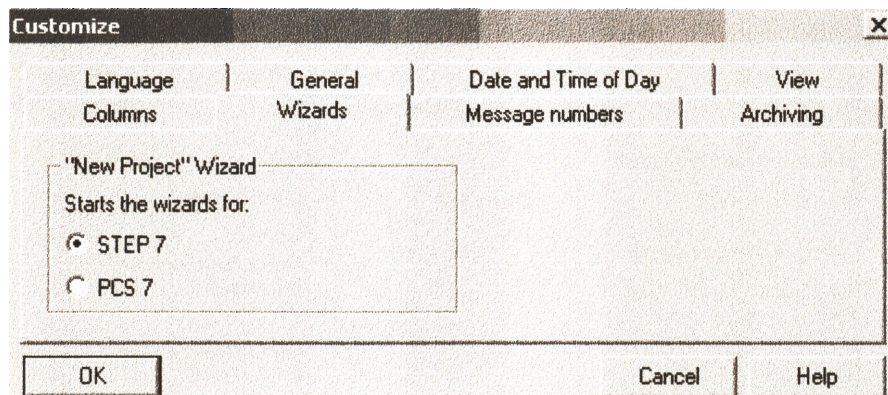


Рисунок 2.4.7: Вкладка Wizards

- «New Project Wizard» – мастер создания нового проекта.
 - STEP 7 – выбрать тип проекта STEP 7, создаваемого мастером по умолчанию.
 - PCS 7 – выбрать тип проекта PCS 7, создаваемого мастером по умолчанию.
- Лучше – оставить первый вариант

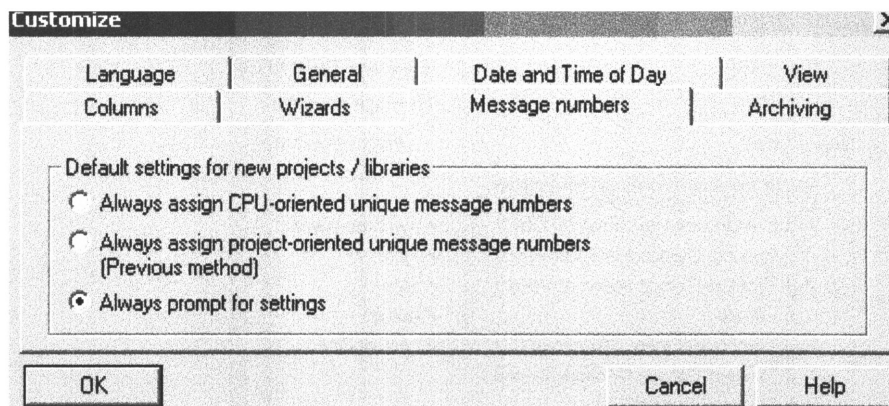


Рисунок 2.4.8: Вкладка Message numbers

- Default settings for new project/libraries – настройки по умолчанию для нового проекта/библиотеки.
- Always assign CPU-oriented unique message numbers – всегда присваивать сообщениям уникальный номер, исходя из параметров CPU.
- Always assign project-oriented unique message numbers – всегда присваивать сообщениям уникальный номер, исходя из параметров проекта.
- Always prompt for settings – всегда подсказывать настройки (по умолчанию).

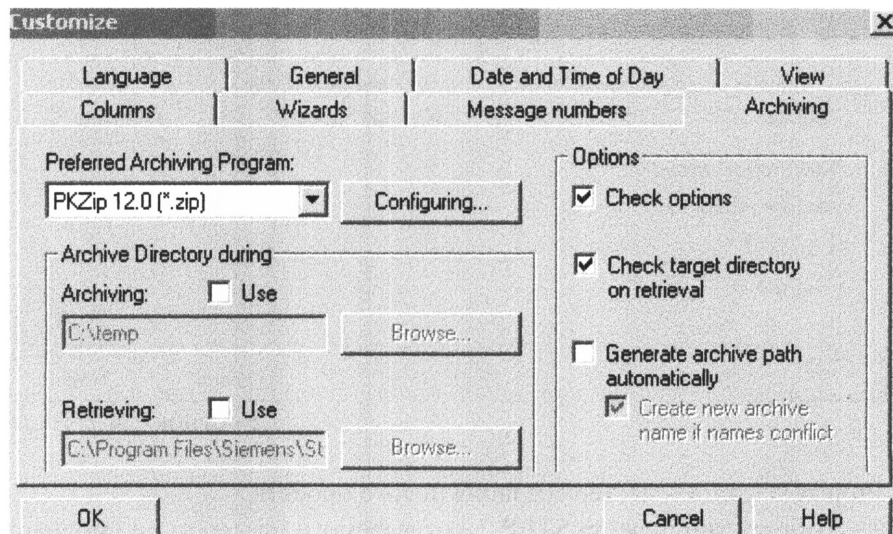


Рисунок 2.4.9: Вкладка Archiving, окно Configuring

- Preferred Archiving Program– предпочитаемая программа для архивирования
- Archive Directory during (Archiving /Retrieving) – папка, в которую переместится готовый архив/в которой находятся файлы, подлежащие архивированию.
- Options –опции
- Check Options – проверить опции.
- Check target directory on retrieval – проверять данные конечной папки.
- Generate archive path automatically –автоматически генерировать пути архива. За основу имени файла архива будет взято имя архивируемого проекта
- Configuring--Archive that goes across Diskettes – настройка разбиения архива на части.

2.5 Создание проекта в Simatic Manager

Для создания нового проекта необходимо запустить Simatic manager, нажать левой кнопкой мышки на меню File и выбрать пункт New, после чего в диалоговом окне ввести имя проекта, определить его тип и место размещения на жестком диске.

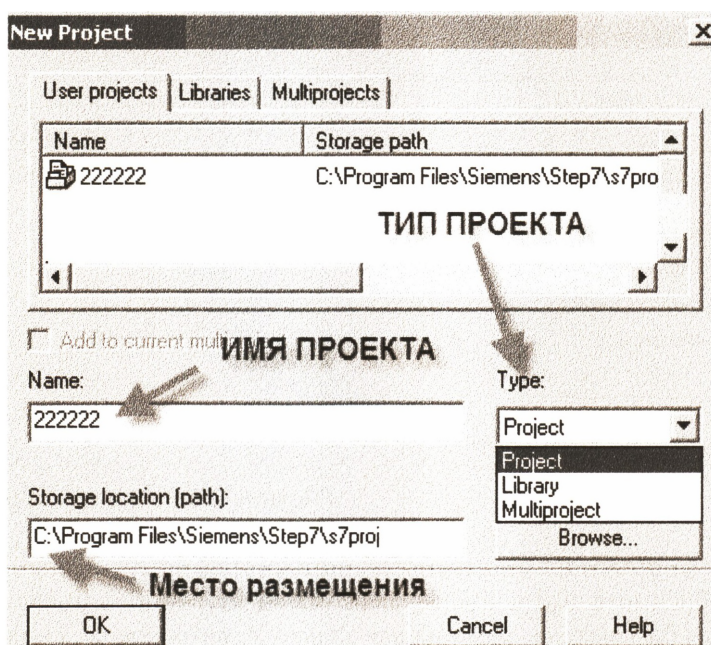


Рисунок 2.5.1: Окно создания проекта

User Project – Пользовательские проекты

Libraries – библиотеки

Multiprojects – многопользовательские проекты

Storage Location (Patch) – размещение на жестком диске (путь)

Type – тип проекта

Кнопка Cancel - отмена

Кнопка Help – вызов справки

Проекты предназначены для хранения данных об аппаратной и программной части решения задачи автоматизации, то есть данных о конфигурации оборудования и сетевых коммуникациях, параметрах модулей и исходных программ.

Структура проектов Simatic Manager и библиотек – иерархическая

Библиотеки (libraries) – подпрограммы, предназначенные для хранения многократно используемых компонентов программы.

В комплект поставки Step7 обязательно включается стандартная библиотека Standard library. Для перехода в библиотеку необходимо в окне создания проекта выбрать вкладку Libraries.

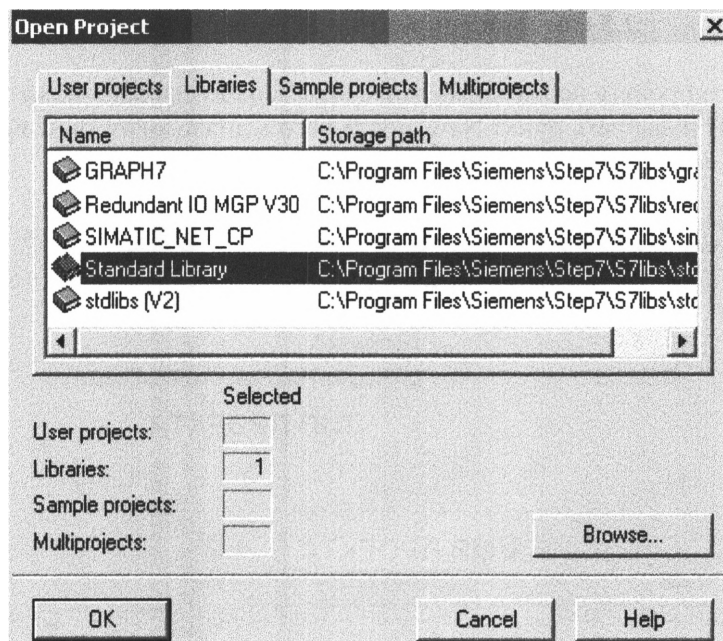


Рисунок 2.5.2: Окно отображения библиотек

Simatic Manager также позволяет создавать библиотеки (File—new—library)

Если при создании библиотеки установлен флажок F library – библиотека идентифицируется как пользовательская библиотека, работающая только в контроллерах F-системы. Иконка такой библиотеки будет желтой.

библиотека Standard library содержит следующие компоненты:

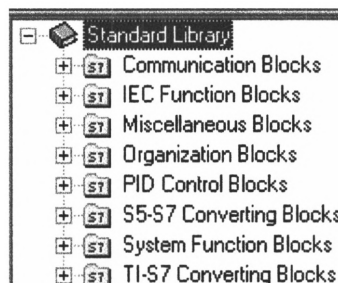


Рисунок 2.5.3: Содержимое библиотеки Standard library

Название английское	Название русское	Содержимое
Communication Blocks	коммуникационные блоки	функции для управления модулями CP
IEC Function Blocks	функциональные блоки IEC	функции для редактирования комплексных переменных и аналоги аппаратных таймеров/счетчиков
Miscellaneous Blocks	Прочие блоки	Блоки синхронизации времени
Organization Blocks	организационные блоки	шаблоны для организационных блоков т.е. раздел объявления переменных для стартовой информации
PID Control Blocks	блоки ПИД управления (пропорционально-интегрально-дифференциальное регулирование)	функциональные блоки для систем автоматического управления
S5-S7 Converting Blocks	Блоки преобразования Step5 в Step7	функции для преобразования программ Step5 в программы Step7
System Function Blocks	Системные функциональные блоки	интерфейсы вызовов системных блоков для создания программ в автономном режиме
TI-S7 Converting Blocks	Блоки преобразования TI-S7	функции и функциональные блоки для преобразования TI-S7. Различные стандартные функции. Например - масштабирование

Таблица 2.5.1: Содержимое библиотеки Standard library

После задания начальных параметров проекта необходимо нажать левой кнопкой мыши на кнопку Ok.

Теперь необходимо добавить в программу информацию об используемом аппаратном обеспечении, то есть создать и сконфигурировать станцию.

Для этого необходимо навести указатель курсора на проект, нажать правую кнопку мыши, выбрать пункт меню Insert New object (вставить новый объект) и выбрать тип станции в зависимости от используемого аппаратного обеспечения. В данном примере показано добавление станции Simatic S300.

После добавления станции структура проекта станет древовидной, слева от имени проекта появится квадрат со знаком «+» внутри, на который необходимо нажать. По нажатию на квадрат со знаком «+» откроется древовидная структура проекта. В ней необходимо однократно нажать левой кнопкой мыши по имени станции (скорее всего, она будет

называться SIMATIC 300 (1)). После этого в правой части окна появится иконка с надписью hardware (оборудование), по которой необходимо дважды кликнуть левой кнопкой мыши.

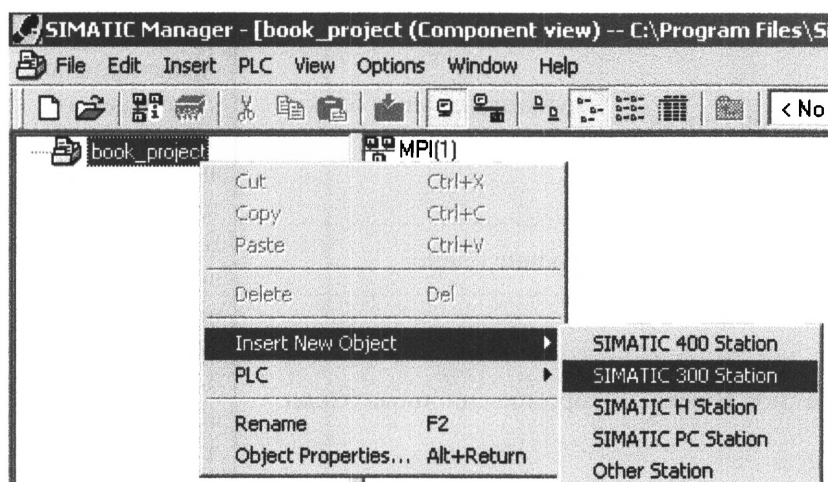


Рисунок 2.5.4: Добавление новой станции S7-300 в проект

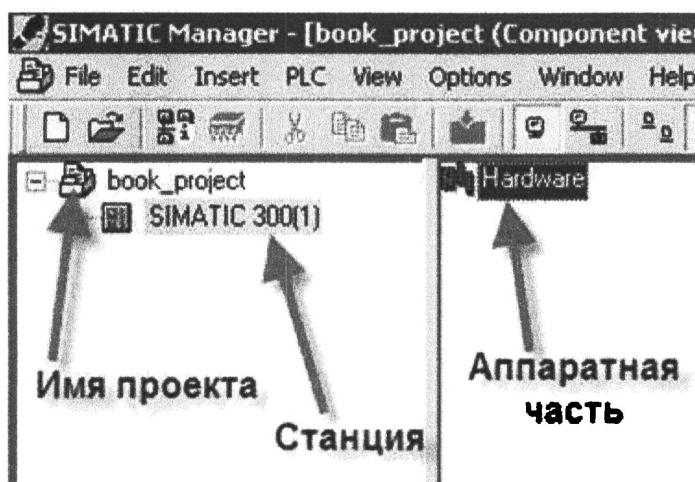


Рисунок 2.5.5: Запуск конфигурирования аппаратной части

Данное действие приведет к открытию утилиты HW-Config, предназначенной для создания и редактирования конфигураций аппаратного обеспечения.

При запуске HW-Config в правой части будет находиться окно станции и каталог оборудования. Если каталог оборудования не отображается – необходимо выбрать пункт меню View и поставить галочку напротив вкладки Catalog.

В каталоге отображены все доступные монтажные стойки (rack), модули, интерфейсные модули совместимые со Step7.

Также предусмотрена возможность скомпилировать свой собственный каталог из наиболее часто используемых модулей. Это осуществляется посредством выбора пункта Edit Catalog Profile в меню Options

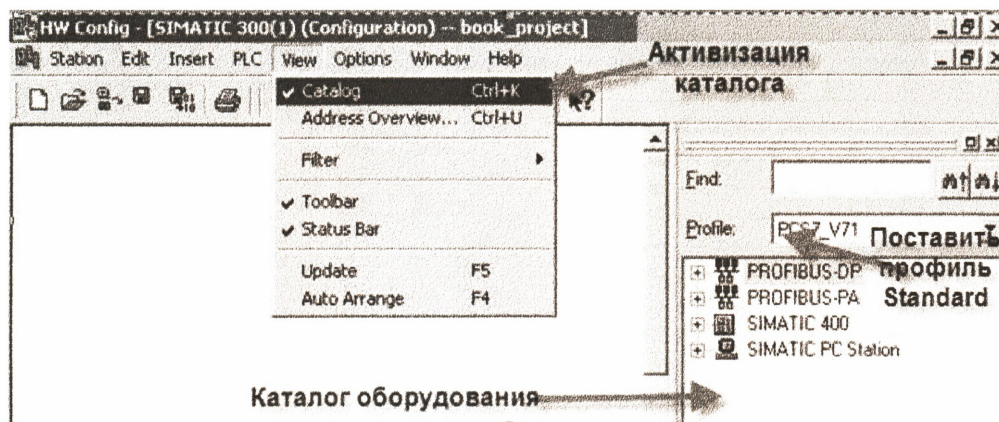


Рисунок 2.5.6: Внешний вид утилиты HW-Config

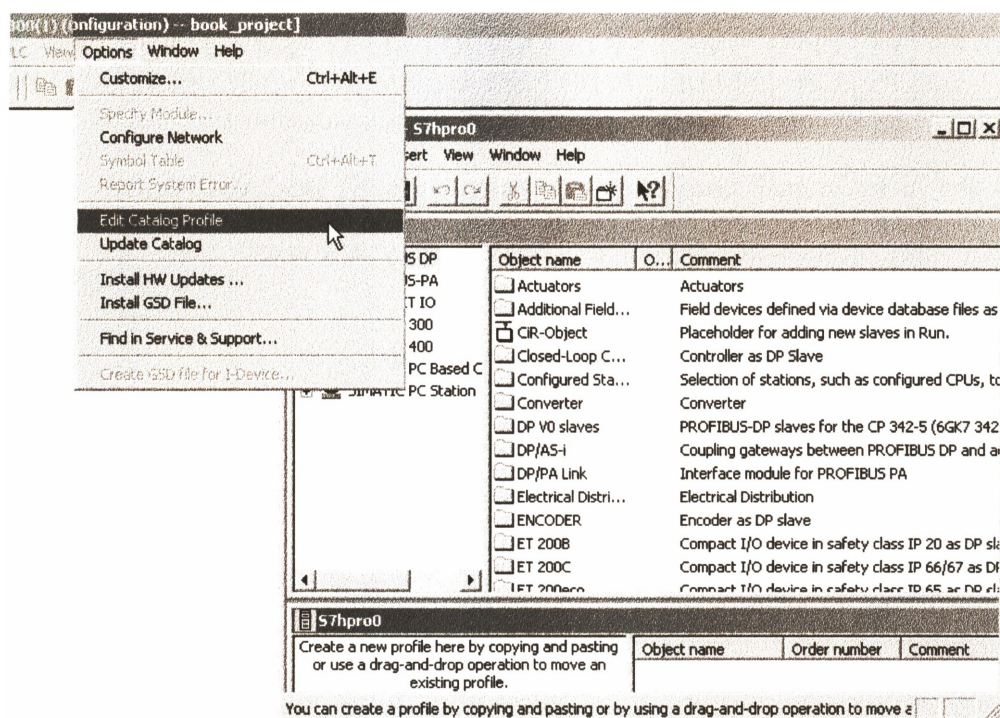


Рисунок 2.5.7: Редактор каталогов

Конфигурирование аппаратной части начинается с несущей (задней) шины (RACK). У CPU S7-300 она активным элементом не является и заказного номера не имеет, называется Rail. У CPU S7-400 задняя шина является активным элементом, имеет заводской номер и множество моделей.

Если в каталоге отсутствует Simatic 300 – необходимо выбрать каталог Standard в окне Profile каталога.

Для добавления несущей шины необходимо перетащить её из каталога в окно станции. Остальные модули, согласно фактической конфигурации модулей перетаскиваются на Rail. При этом во избежание ошибок важно соблюсти соответствие реальной и конфигурируемой аппаратной части.

При перетаскивании модулей на Rail, места, в которые разрешается устанавливать оборудование, будут подсвечиваться зеленым.

Также важно помнить рекомендуемую последовательность установки модулей.

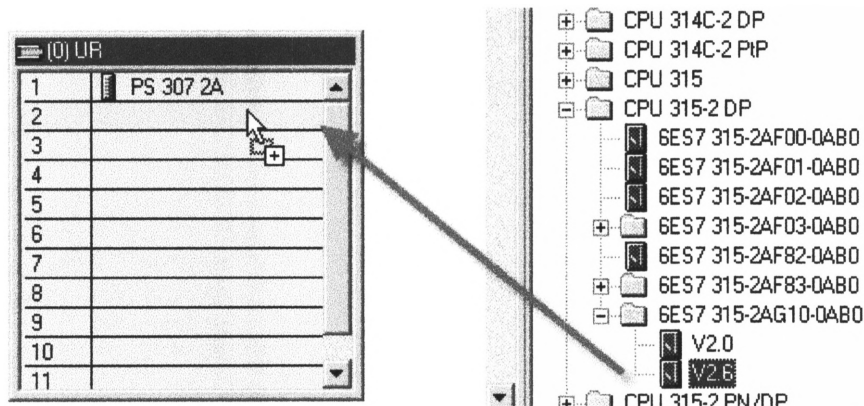


Рисунок 2.5.8: Конфигурирование аппаратной части

	1	2	3	4	5	6	7	8	9	10
S7-300	PS	CPU	IM	SM DI	SM DO	SM AI	SM AO	FM	CP	
S7-400	PS	CPU	SM DI	SM DO	SM AI	SM AO	CP	FM	SM	IM

Таблица 2.5.2: Последовательность расположения модулей S7-300 и S7-400

Каждый модуль имеет свой заказной номер, например 6ES7 315-2AG10-0AB0.

В общем виде заказной номер контроллерного оборудования фирмы Siemens серии S7-300 и S7-400 выглядит следующим образом: 6ES7-ABC-YYYY-WWWW

6ES7 – первые символы для всех элементов стойки.

ABC – Для CPU – номер серии (например - 315)

ABC – Для PS: Для S7-300 – 307. для S7-400 – 405 или 407

Для сигнальных модулей:

A – Для сигнальных модулей S7-300 =3

A – Для сигнальных модулей S7-400 =4

B – Для аналоговых модулей =3

B – Для цифровых модулей =2

C – Для входных модулей =1

C – Для выходных модулей =2

C – Для модулей AI/AO S7-300 = (4 или 5)

C – Для модулей DI/DO S7-300 = (3 или 7)

Для конфигурирования структуры проекта S7-300 достаточно установить модуль CPU. В случае с S7-400, помимо CPU необходимо добавить источник питания PS. Остальные модули можно добавить позже.

В однорядных станциях S7-300, слот №3 оставляют пустым. Он резервируется для интерфейсного модуля связи со стойкой расширения.

Начальные адреса модулей назначаются автоматически.

После конфигурирования аппаратной части необходимо ее сохранить и скомпилировать нажатием на кнопку Save and Compile.

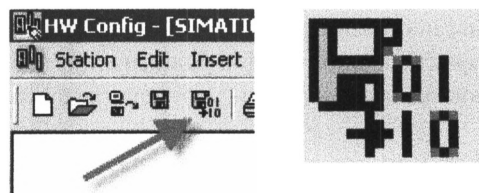


Рисунок 2.5.9: Кнопка Save and Compile

После сохранения и компилирования аппаратной части можно закрыть утилиту HW Config и вернуться к работе в Simatic Manager.

Теперь создание структуры проекта завершено и ее древовидность становится еще более очевидной: в папку проекта добавляется папка CPU, в которой содержится S7-программа (S7-program) со всеми объектами.

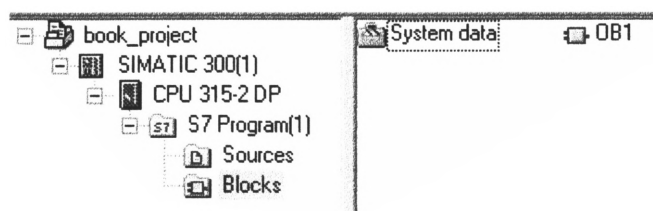


Рисунок 2.5.10: Древовидная структура проекта

2.6 Адресация слотов и установка начальных адресов модулей

Все слоты S7 станции получают конкретный, фиксированный адрес, который формируется из номера монтажной стойки и номера слота.

Распределительные станции ввода-вывода также имеют адрес, сформированный из номера системы ведущего DP-устройства и номера станции.

Следовательно, все сигналы (дискретные/аналоговые/входные/выходные) и последовательные соединения системы имеют уникальный адрес.

Также, все модули имеют начальный адрес, определяющий позицию модуля в области отображения логических адресов. Адресация входов/выходов начинается с 0 и заканчивается значением, определяемым типом оборудования.

Начальный адрес определяет, адресацию входных/выходных сигналов в программе ПЛК. В дискретных модулях содержится, как правило, 8 или 16 дискретных входов/выходов. Каждому входу/выходу соответствует 1 бит из области отображения входов/выходов. Биты собираются в байты (в одном байте 8 бит), имеющие адреса. Например, дискретный модуль с 16 входами/выходами (2 байта) с начальным адресом модуля «0», отдельные байты будут иметь адреса 0 и 1 соответственно. А отдельные биты будут иметь адресацию, содержащую область памяти, номер байта и номер бита.

Пример:

I0.3 –область отображения входов, байт 0, бит 3

I0.7–область отображения входов, байт 0, бит 7.

I1.7 –область отображения входов, байт 1, бит 7

Запись I1.8 – недопустима, т.к. в байте содержится 8 бит с нулевого по седьмой.

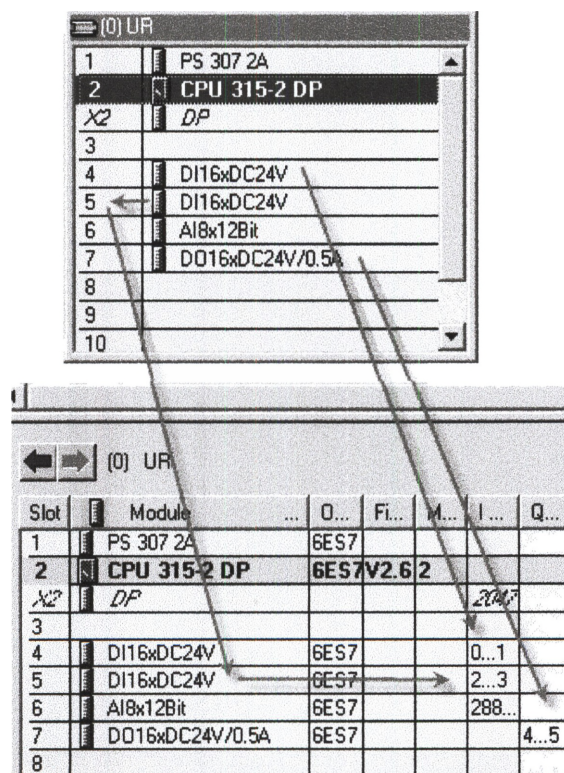


Рисунок 2.6.1: Добавление SM-блоков, начальная адресация

В HW-Config предусмотрена возможность изменения начального адреса модулей. Для изменения начального адреса необходимо: навести указатель мыши на модуль, нажать правой кнопкой мыши, выбрать пункт меню Object Properties (свойства объекта), выбрать вкладку Address.

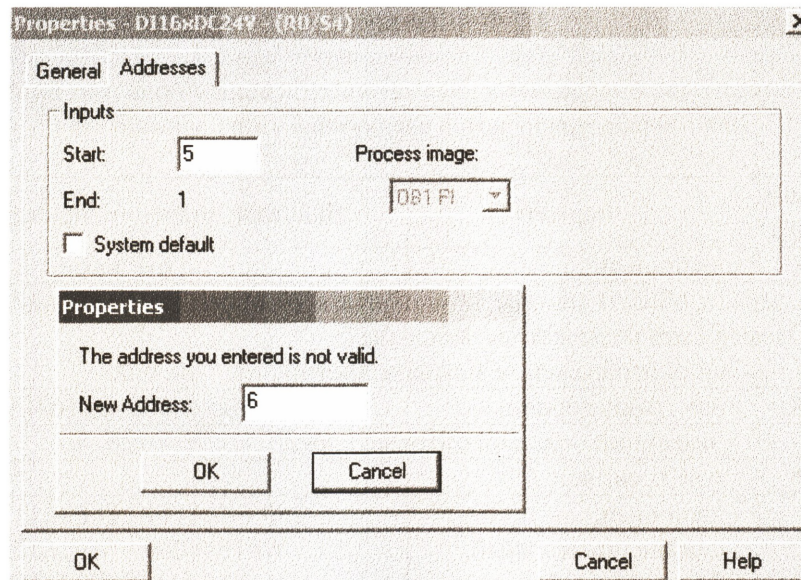


Рисунок 2.6.2: Настройка начального адреса сигнального модуля в области отображения

Если выбрана опция System default (по умолчанию) – то начальный адрес модуля автоматически назначается программой и поле редактирования начального адреса Start заблокировано. Для редактирования необходимо отключить параметр System default. В случае ввода уже занятого другим модулем адреса в качестве нового – автоматически появится окно, предлагающее поменять начальный адрес на незанятый. При использовании аналоговых модулей ввода/вывода (входным/выходным сигналом служит аналоговая величина напряжения или тока) каждый из аналоговых сигналов подается на отдельный канал (channel). Каждый канал имеет свой адрес, занимающий в памяти контроллера 1 машинное слово (16 бит или 2 байта). В зависимости от конструкции аналоговые модули могут иметь от двух до 16-ти каналов и занимать и занимать, соответственно от двух до 16 машинных слов. Удаленные (периферийные) модули и станции также резервируют часть памяти в области отображения входов/выходов. Адреса централизованных и периферийных модулей не должны совпадать. Установка начального адреса каждого модуля производится центральным процессором при включении питания. Начальный адрес каждого модуля зависит от типа модуля, слота и стойки.

Диагностический адрес – еще один адрес устройства, предназначенный для удаленной аппаратной диагностики устройств с помощью специальных функций (Например, SFC13). Диагностику можно выполнять, если между устройствами есть сеть Profibus DP. С другими видами сетей (MPI, Ethernet) – нельзя. Диагностическим адресом обладают все устройства Siemens, которые имеют встроенные Profibus DP интерфейс.

Slot	Module	...	O...	Fi...	M...	I...	Q...	Comment
1	PS 307 2A		6ES7					
2	CPU 315-2 DP		6ES7V2.6 2					
X2	DP					204		
3								
4	DI16xDC24V		6ES7			0...1		
5	DI16xDC24V		6ES7			2...3		
6	AI8x12Bit		6ES7			288...		
7	DO16xDC24V/0.5A		6ES7				4...5	
8								

Рисунок 2.6.3: Отображение диагностического адреса

2.7 Форматы данных

Формат данных служит для определения параметров отдельным областям памяти CPU. Контроллерами S7-300/400 поддерживаются следующие типы данных:

Тип данных	Длина, байт	Формат	Диапазон значений, пример записи
bool	1	двоичный	True, False
Адрес бита содержит область памяти, номер бита, к которому принадлежит байт и номер байта. Номер бита не может быть больше 7. Пример: Q3.7 – область отображения выходов, третий бит, 7-ой байт			
byte	8	шестнадцатеричный	B#16#00 – B#16#FF
старший бит – бит, имеющий больший битовый адрес.			
Char	8	ASCII символ	A
word	16	двоичный, шестнадцатеричный, беззнаковый байтовый	2#0 – 2#1111 1111 1111 1111 W#16#0000 – W#16#FFFF B#(0,0) – B#(255,255)
int	16	Целый, знаковый	-23768.....+32767
старший бит – знаковый. Старший байт – байт с меньшим адресом			
date	16	год-месяц-день	D#1990-01-01....D#2168-12-31
Дата представляется беззнаковым целым числом дней, прошедших с 1 января 1990 года. (1 соответствует 1 января 1990г.)			
s5time	16	час.мин.сек или мин.сек.мсек	S5T#10ms...S5T#2h46m30s
Определяется произведением трехразрядного BCD-числа и базового множителя. Для преобразования формата S5#Time в Time существует функцией FC 33 "S5TI_TIM» Для обратного преобразования – FC40 "TIM_S5TI" Доступ к указанным функциям осуществляется через: Simatic Manager--- File/Open/Library/Standard Library/IEC/<AP-OFF>/			
Dword	32	двоичный шестнадцатеричный беззнаковый байтовый	2#0 – 2#11111111111111111111111111111111 DW#16#0... DW#16#FFFFFFFF B#(0,0,0,0)...B#(255,255,255,255)
старший байт (старшее слово) – байт/слово с меньшим адресом			
dint	32	целый знаковый	L#-2147483648 – L#+2147483647
Старший бит – знаковый			
real	32	вещественный с плавающей точкой	+1.175494e-38....+3.402823e+32 -1.175494e-38...-3.402823e+38
Округляется до шестого знака после запятой			
time	32	интервал времени со знаком; +или –дней, ч, м, с, мс	T#-24d20h31m23s647ms – T#+24d20h31m23s647ms допускается запись T#5h10s и т.д.
time of day или TOD	32	время дня в ч:м:с:мс	TOD#00:00:00.000 – TOD#23:59:59.999 беззнаковое целое число миллисекунд с момента полуночи. Ноль = 0ч 0м 0с

Таблица 2.7.1: Форматы данных

2.8 Адресное пространство CPU

В формате данных Word старший байт – байт с меньшим адресом

В формате данных DWord старший байт (старшее слово) – байт/слово с меньшим адресом

При обращении к биту необходимо указать **[область памяти] [№байта]. [№бита]**

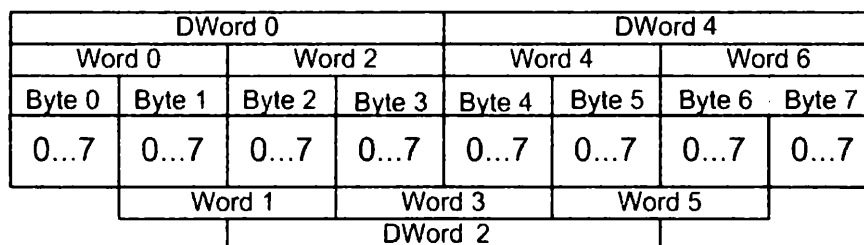


Рисунок 2.8.1: Иллюстрация области памяти

Обращение к переменным: **[Область памяти] [формат данных] [адрес]**

Области памяти:

Области памяти:		Формат данных	
Символ	Область памяти	Символ	Формат данных
I	Область отображения входов		Бит
Q	Область отображения выходов	B	Байт
M	Меркерная область памяти	W	Слово
L	Область памяти локального стека	D	Двойное слово

Таблица 2.8.1: Обозначение областей памяти и форматов данных

Пример:

Формат данных	Пример обращения	Комментарий
Бит	I3.7	Область отображения входов, третий байт, 7-й бит
Байт	QB4	Область отображения выходов, четвертый байт
Слово	MW2	Меркерная область памяти, второе слово
DWord	QD6	Область отображения выходов, 6-ое двойное слово.

Таблица 2.8.2: Пример обозначения областей памяти и форматов данных

В случае обращения к блоку данных запись выглядит иначе:

DB(№блока данных).DBØ [адрес байта].[адрес бита],

Где Ø - формат обращения: X – бит, B-байт, W-слово, D- двойное слово

DB3.DBX10.5 –третий блок данных, десятый байт, пятый бит

DB2.DBD4 – второй блок данных, четвертое двойное слово

Обращение к таймерам и счетчикам осуществляется по их номеру

Пример:

Область памяти	Пример обращения	Комментарий
Таймеров	T 8	Область памяти таймеров, таймер №8
Счетчиков	C 25	Область памяти счетчиков, счетчик №25

Таблица 2.8.3: Пример обозначения таймеров и счетчиков

2.9 Simatic S7 проект

Всякий проект может содержать в себе как одну, так и несколько станций, связанных между собой сетью передачи данных (Profibus, MPI-DP, Industrial Ethernet и т.д.). В каждой станции имеется CPU, который содержит S7-программу, а S7-программа, в свою очередь содержит следующие объекты:

Блоки (Blocks) – содержат скомпилированную часть программы

Источники (Sources) – содержат исходные тексты программ

Символы (Symbols) – содержит символьную информацию проекта

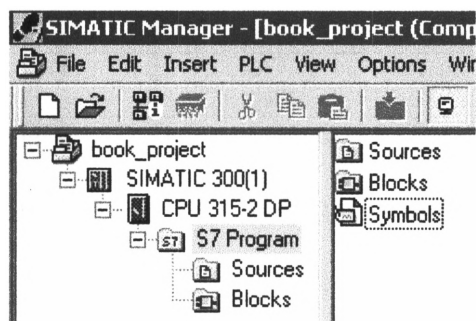


Рисунок 2.9.1: Структура папки S7Programm

В общем случае программа S7 состоит из программы, написанной пользователем (user program) и операционной системы CPU (operating system).

Операционная система (operating system) – системный компонент CPU представляющий совокупность инструкций, предназначенных для управления системными ресурсами и процессами, использующими эти ресурсы.

Операционная система содержит следующие функции:

- Резервирование данных при сбое электропитания
- Активация приоритетных классов
- Решение прочих задач взаимодействия

Пользовательская программа – совокупность инструкций для обработки сигналов, с помощью которых осуществляется управление объектом автоматизации в соответствии с требуемой логикой, программируемой в логических блоках.



Рисунок 2.9.2: Структура пользовательской программы

2.10 Программные блоки

Обозначение	Описание
DB - Data Block (Блок данных)	Предназначены для хранения данных программы пользователя. Форма хранения данных (номер блока, тип данных, порядок хранения) определяется пользователем. Блоки данных делятся на блоки глобальных данных (global data blocks), которые в программе пользователя имеют статус «свободные блоки» (free blocks), не назначаемые кодовому блоку и экземплярные блоки данных (instance data blocks) – блоки данных со статусом «зависимые» - назначаемые функциональному блоку и сохраняющие его локальные данные. Максимальное число DB и их размер определяются типом CPU.
DI – Data block instance Блок данных функционального блока	Предназначен для хранения значений переменных FB. Отличается от DB наличием жесткой структуры, определяемой связанной с ним функциональным блоком.
OB – Organization Block (Организационный блок)	представляют собой интерфейс между операционной системой и программой пользователя. Операционная система вызывает OB при возникновении событий, например, аппаратного прерывания или прерывания по времени (см. рисунок)
FC - Function (Функция)	Используется для программирования часто повторяющихся или сложных функций автоматики. Может быть вызвана из любого блока. Допускается передача параметров в функцию и обратно. Может иметь локальные переменные, которые теряются при выходе из блока.
SFC- System Function (Системная функция)	функция, уже имеющаяся в ОС CPU. Предназначены для выполнения определенных стандартных действий
FB – Function Block (Функциональный блок)	программные части, вызов которых может быть запрограммирован с помощью параметров блока. Имеют область памяти для хранения переменных, располагаемую в блоке данных, который привязан к вызову FB. Допускается назначение нескольких блоков данных (с одинаковой структурой, но разными значениями) одному FB. Такой DB называется экземплярным блоком данных(instance data block). FB также могут хранить свои переменные в экземплярном блоке данных вызывающего функционального блока - такой экземплярный DB называется локальным экземпляром (local instance).
SFB –System Function Block Системный функциональный блок	Аналогичен FB, но, как и SFC уже имеется в составе контроллера.

Таблица 2.10.1: Типы программных блоков

Главная программа ВСЕГДА расположена в организационном блоке ОВ1. Остальные организационные блоки имеют фиксированные номера и предназначены для обработки событий осуществляющих их вызов. Число ОВ и их номера фиксированы и зависят от модели CPU. Организационный блок ОВ1 является основным организационным блоком, а остальные блоки исполняются, прерывая ОВ1. После исполнения управление программой заново отдается в ОВ1. Таким образом осуществляется циклическое исполнение программы.

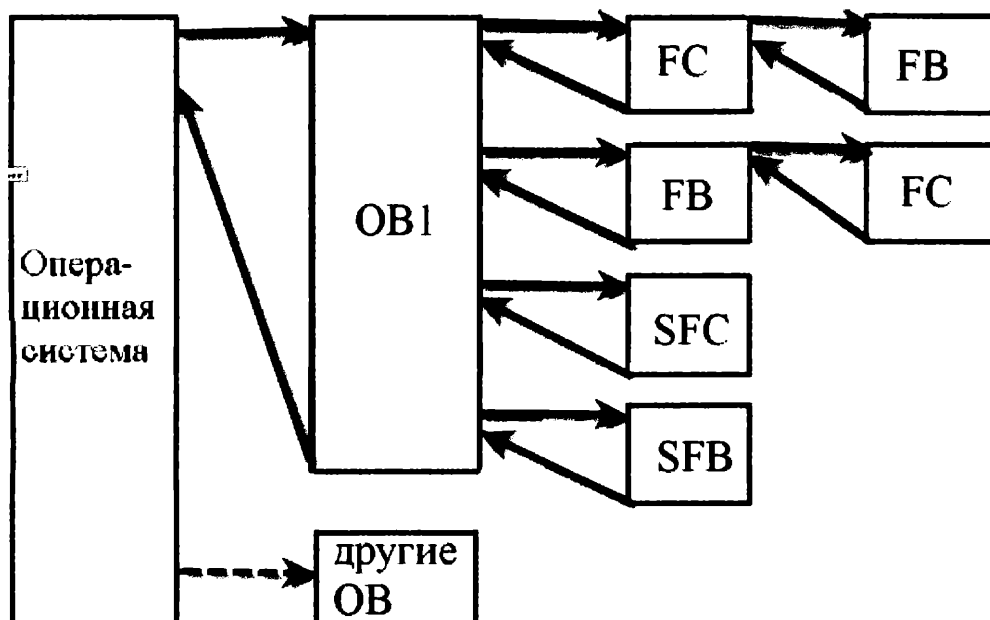


Рисунок 2.10.1: Структура программы контроллера

2.11 Назначение организационных блоков

Назначение	ОВ№	Условие вызова		Приоритет	
				по умолча нию	возмо жные измен ения
свободный цикл	1	Автоматически		1	нет
TOD прерывания	10-17	Определенное время суток или через равные промежутки времени		2	2-24
с задержкой времени	20-23	По истечению запрограммированного времени. Управление из пользовательской программы		3-6	2-24
Циклические прерывания	30	Регулярный вызов через запрограммированные интервалы времени. Например, каждые 100 мс. Интервал времени показан в столбце справа. Фазовое смещение по умолчанию для данной группы блоков 0.	5с	7	2-24
	31		2с	8	
	32		1с	9	
	33		500мс	10	
	34		200мс	11	
	35		100мс	12	
	36		50мс	13	
	37		20мс	14	
	38		10мс	15	
Прерывания процесса (от аппаратуры)	40-47	сигнал прерываний от I/O модулей		16-23	2-24
мультипроцессорное прерывание	60	приход события в мультипроцессорном режиме, управление из пользовательской программы		25	нет
Ошибки резервирования	70	ошибка резервирования периферии		25	2-26
	72	ошибка резервирования CPU		28	2-28
	73	ошибка резервирования коммуникаций		25	24-26
асинхронные ошибки (не связанные с выполнением программы)	80	ошибка времени (timing error)		26	26
	81	ошибка в блоке питания		26	2-26
	82	диагностическое прерывание		26	2-26
	83	прерывание установки/удаления модуля		26	2-26
	84	отказ аппаратной части CPU		26	2-26
	85	ошибка выполнения программы		26	24-26
	86	отказ стойки (rack failure)		26	2-26
	87	коммуникационная ошибка		26	2-26
фоновая обработка	90	Продолжительность минимального цикла еще не достигнута		29	нет
Программа запуска	100	Полный перезапуск		27	нет
	101	Теплый перезапуск		27	нет
	102	Холодный перезапуск		27	нет
Синхронные ошибки	121	Ошибка, связанная с выполнением программы. Например, ошибка I/O доступа		приоритет ОВ, вызвавшего ошибку	
	122				

Таблица 2.11.1: Назначение организационных блоков

2.12 Выбор адаптера

Когда программа уже создана необходимо выбрать адаптер для загрузки программы в ПЛК. Для этого необходимо:

В меню Options строки меню выбрать пункт Set PG/PC Interface

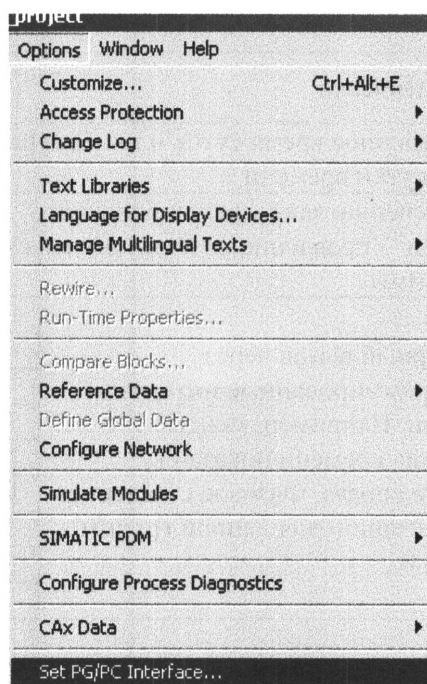


Рисунок 2.12.1: Вкладка Options строки меню

И в появившемся окне выбрать используемый адаптер для связи ПК с ПЛК. Если необходимо использовать симулятор – выбрать пункт PLCSIM (MPI).

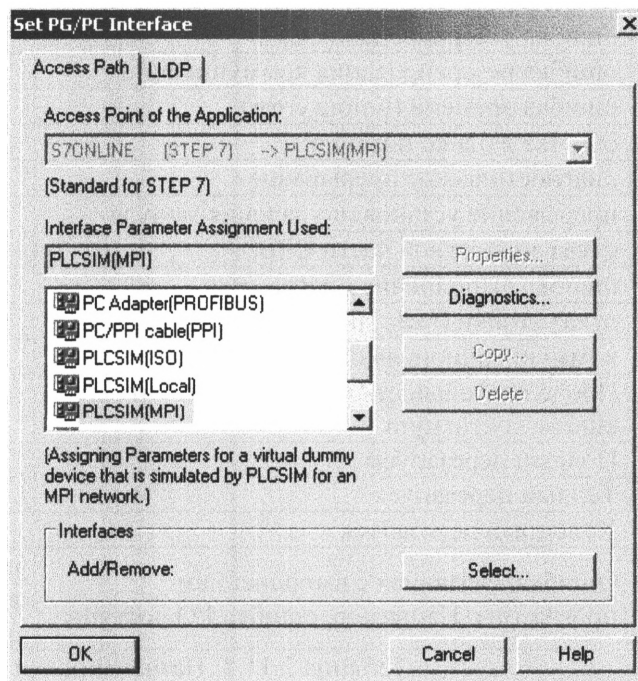



Рисунок 2.12.2: Установка интерфейса связи

После данных манипуляций необходимо перевести ПЛК в режим RUN-P для первоначальной загрузки в него данных. В случае с виртуальным ПЛК необходимо вначале запустить программу-симулятор и ее настроить. Для этого необходимо нажать кнопку  на панели инструментов или запустить симулятор из папки пуск—программы—Simatic—Step7 - S7-PLCSIM Simulating Modules.

После запуска симулятора появится окно открытия проекта симулятора, в котором необходимо выбрать пункт Select CPU access node и нажать кнопку ОК.

В появившемся после этого окне необходимо выбрать ту сеть, при которой кнопка ОК станет активной, и нажать ее.

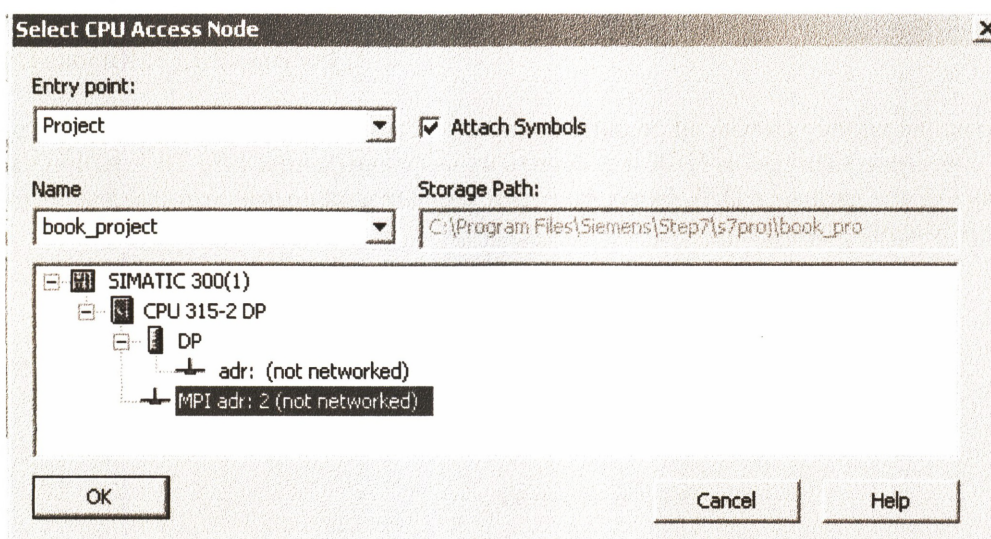



Рисунок 2.12.3: Выбор сети виртуального CPU

Затем, нажатиями на кнопки insert input variable и insert output variable добавить входной и выходной байты. Номера байтов вводятся с клавиатуры. Вводить те же номера входных и выходных байтов, что и в главе «Адресация слотов и установка начальных адресов модулей».



Рисунок 2.13.4: Утилита PLCSIM

Теперь можно произвести первоначальную загрузку данных в ПЛК. Для этого необходимо:

В программе Simatic Manager выделить мышью станцию и нажать кнопку .

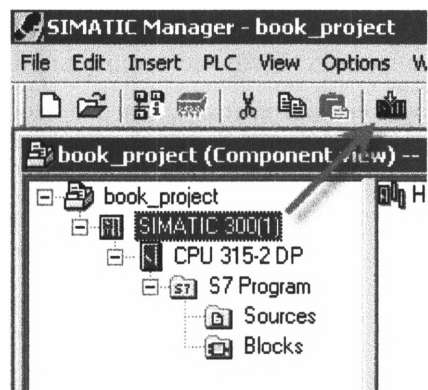


Рисунок 2.13.5: Кнопка Download

По нажатию на кнопку Download появится окно с текстом «вы точно хотите полностью удалить системные данные из ПЛК и заменить их оффлайн-данными?» То есть при нажатии на кнопку ОК все данные с ПЛК будут стерты и заменены данными, которые были созданы в программе Simatic Manager.

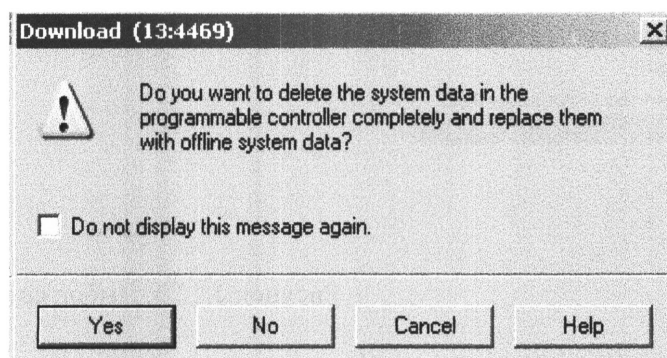


Рисунок 2.13.6: Окно загрузки

2.13 Создание первой программы

Как уже было упомянуто выше, основная программа контроллера пишется в организационном блоке OB1, который находится в папке blocks.

Для написания программы его необходимо открыть двойным щелчком левой кнопки мыши. Или выделить его правой кнопкой мыши и выбрать пункт open object всплывающего меню.

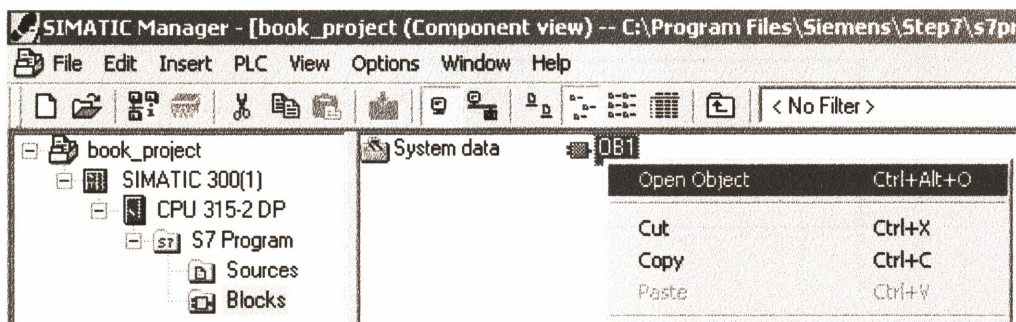


Рисунок 2.13.1: Открытие OB1

Все организационные блоки открываются утилитой «LAD, STL, FBD - Programming S7 Blocks»

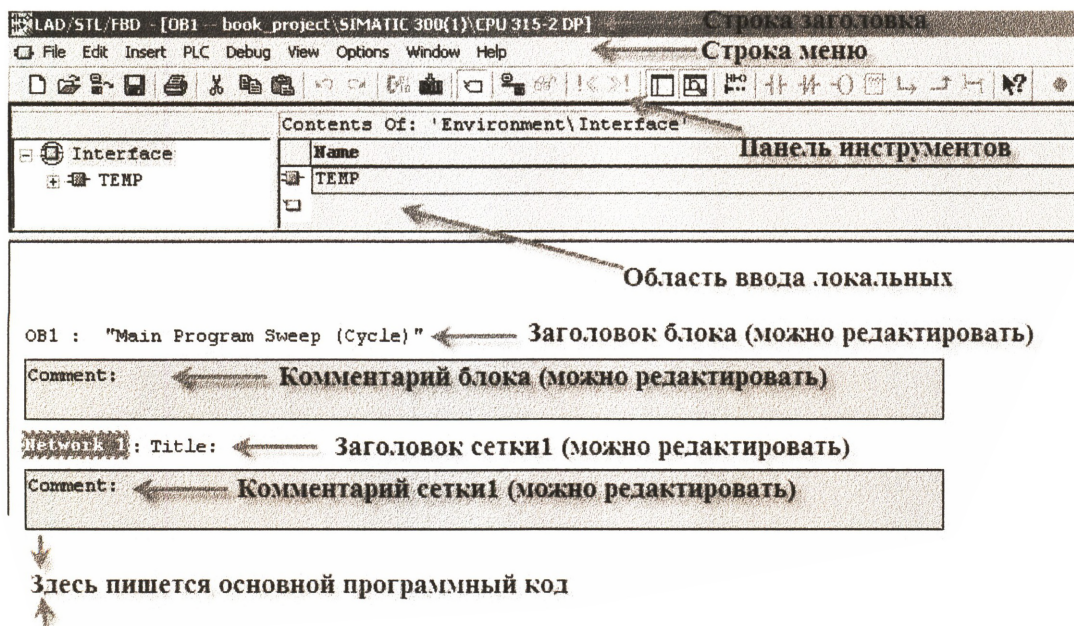


Рисунок 2.13.2: Открытый OB1

В STEP7 предусмотрены три основных способа отображения программы:

- Список команд (инструкций) – **STL** (Statement List). Представляет собой список команд подобно обычному языку Ассемблера.
- Контактный план – **LAD** (Ladder Logic). Управляющая программа записывается при помощи изображений элементов релейных контактных схем.
- Функциональный план – **FBD** (Function Block Diagram). Для отображения программы используются схемы логических элементов

Перед написанием программы необходимо выбрать способ отображения, в зависимости от которого будет меняться содержимое вкладки program elements и панели инструментов. Если вкладка program elements не отображается – выбрать пункт View строки меню и поставить галочку напротив пункта Overviews. Или воспользоваться сочетанием клавиш Ctrl+K. В данном пособии основное внимание уделено способу отображения программ STL.

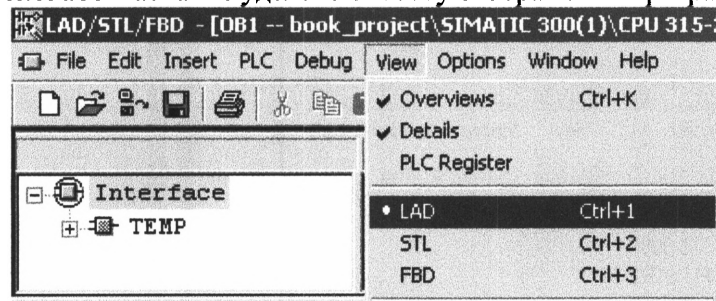


Рисунок 2.13.3: Выбор способа отображения

2.14 Пример программы на LAD

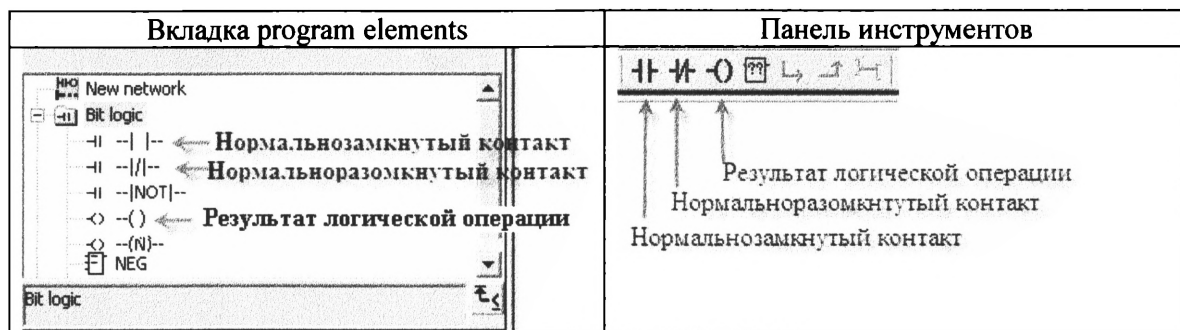


Таблица 2.14.1: Вкладка program elements и панель инструментов LAD

Для составления программы необходимо выделить область ввода кода, перетащить на нее мышью элементы с вкладки program elements и соединить их. Для создания параллельных ветвей потребуется воспользоваться элементами панели инструментов.

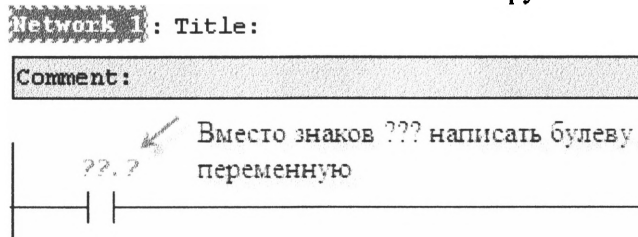


Рисунок 2.14.1: Результат перетаскивания элемента

Network 1: Title:

Comment:

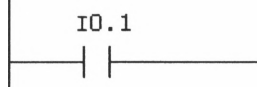


Рисунок 2.14.2: Отображение булевого входа IO.1

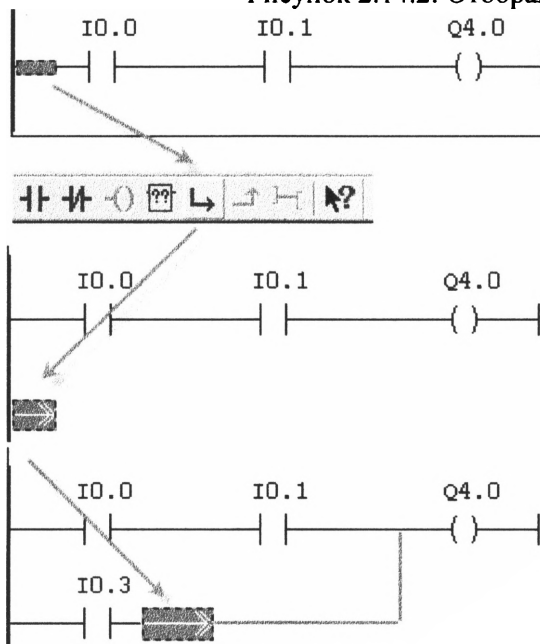


Рисунок 2.14.3: Пример составления программы на LAD

2.15 Пример программы на FBD

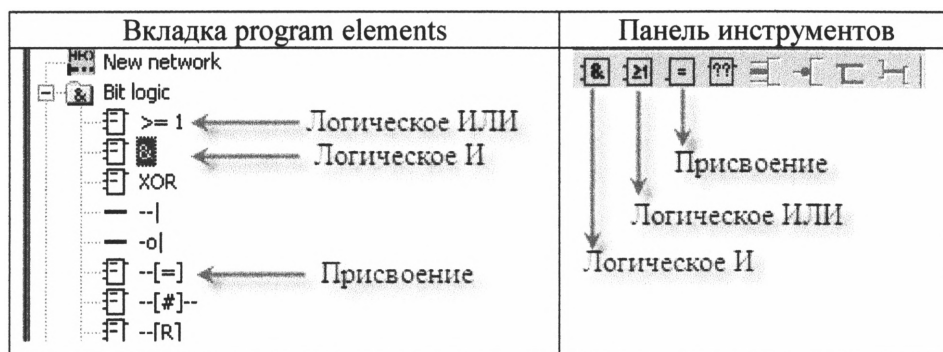


Таблица 2.15.1: Вкладка program elements и панель инструментов FBD

Для составления программы необходимо выделить область ввода кода, перетащить на нее мышью элементы с вкладки program elements и соединить их.

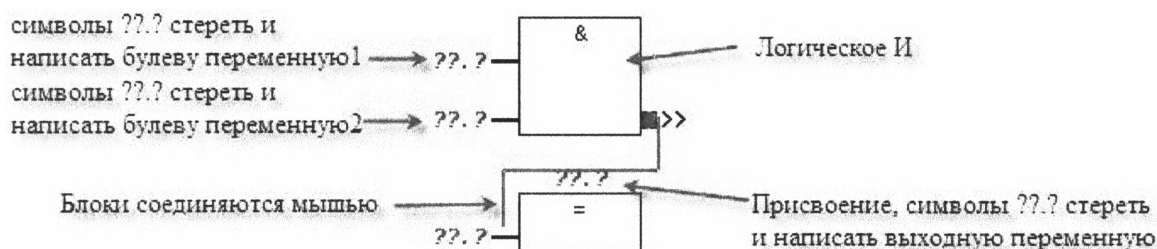


Рисунок 2.15.1: Пример составления программы на FBD

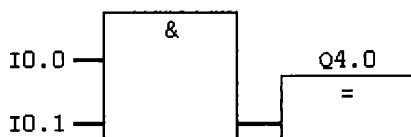


Рисунок 2.15.2: Пример составления программы на FBD

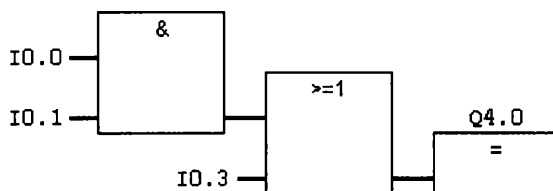


Рисунок 2.15. 3: Пример составления программы на FBD

2.16 Пример программы на STL

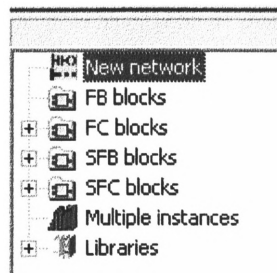


Рисунок 2.16.1: Вкладка program elements на STL



Панель инструментов STL отсутствует.


Для составления программы необходимо написать программный код в области ввода кода.

Network 1: Title:		
Comment:		
A	I	0.0
A	I	0.1
O	I	0.3
=	Q	4.0

Рисунок 2.16.2: Пример составления программы на STL

2.17 Симуляция работы программы

В рассмотренных примерах в конечном результате получалась программа, которая устанавливает выход Q4.0 в логическую единицу при замыкании или входов I0.0 и I0.1 или входа I0.3. Теперь можно загрузить программу в контроллер (если он есть) или воспользоваться симулятором PLCSim (см. раздел «выбор адаптера»). Перед нажатием на кнопку Download  необходимо сохранить написанный код. Сделать это можно либо нажатием на кнопку  панели инструментов либо сочетанием клавиш Ctrl+S.

Теперь можно нажимать кнопку  (или Ctrl+F7), переводить CPU в режим RUN и замыкая входы I0.0, I0.1 и I0.3 (в PLCSim – устанавливая галочки в чек-боксы) любоваться результатом.

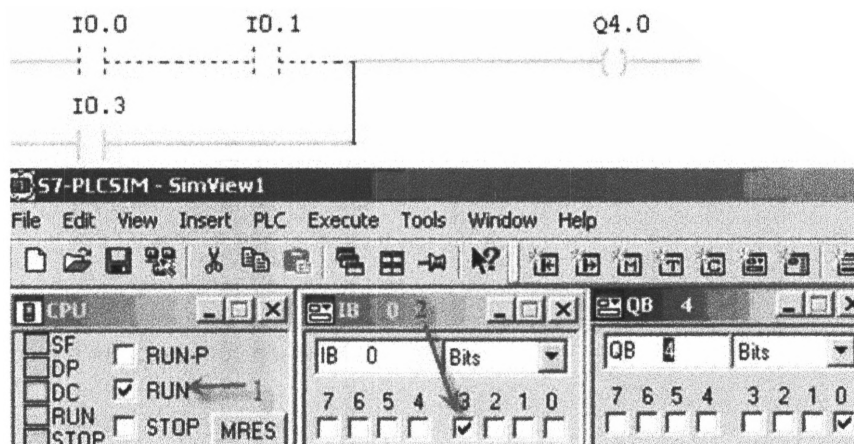


Рисунок 2.17.1: Симуляция программы в LAD

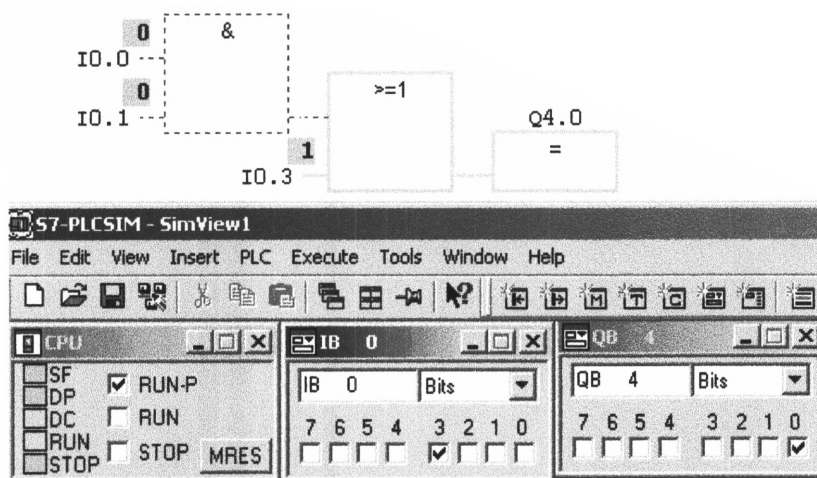


Рисунок 2.17.2: Симуляция программы в FBD

			RLO	STA	STANDARD	ACCU 2
A	I	0.0	0	0	0	0
A	I	0.1	0	0	0	0
O	I	0.3	1	1	0	0
=	Q	4.0	1	1	0	0

Рисунок 2.17.3: Симуляция программы в STL

2.18 Загрузка данных на карту памяти

Существует два варианта загрузки данных на PLC:

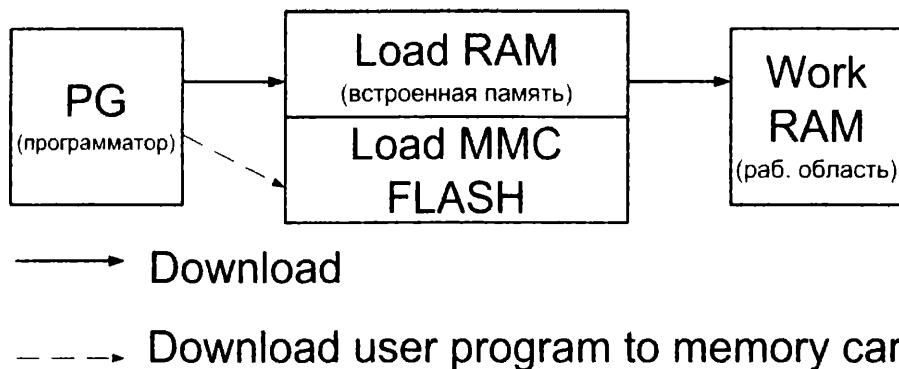



Рисунок 2.18.1: Варианты загрузки программы

1) Копирование данных в RAM-область памяти через карту памяти: данные загружаются на карту памяти и из нее автоматически копируются в область загрузки. Загрузка данным методом занимает много времени, однако в случае сбоя питания данные останутся на карте памяти.

Для передачи данных указанным методом необходимо нажать кнопку  или в меню PLC выбрать опцию Download User Program to Memory Card.

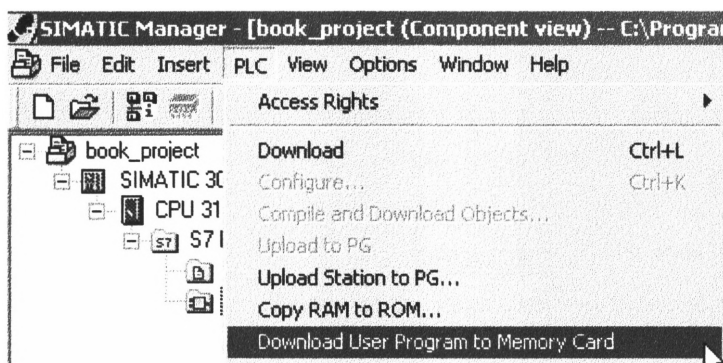


Рисунок 2.18.2: Копирование данных в RAM-область PLC через карту памяти

2) Копирование данных напрямую в RAM-область памяти. Применяется, как правило, при отладке с целью ускорения передачи данных.

Для передачи данных указанным методом необходимо в программе Simatic Manager выделить мышью папку Blocks, вызвать меню PLC и, выбрать пункт Download.

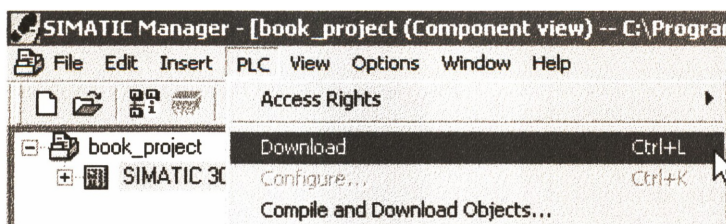


Рисунок 2.18.3: Копирование данных напрямую в RAM-область памяти

Если данное меню неактивно – необходимо изменить профиль проекта на Step-7.
Для этого необходимо в Simatic Manager навести курсор мыши на проект, и нажать правую кнопку мыши, в появившемся меню выбрать пункт Object Properties

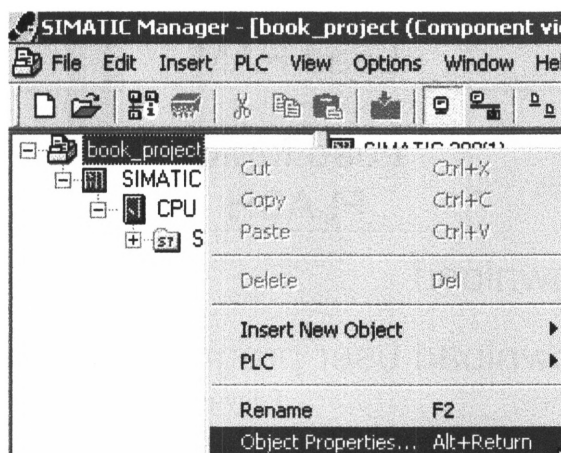


Рисунок 2.18.4:Вызов меню проекта

Если в качестве типа проекта указан PCS7 – данный метод работать не будет, так как для PCS7 ставится отдельная лицензия на объем данных, записываемых в Load Memory.

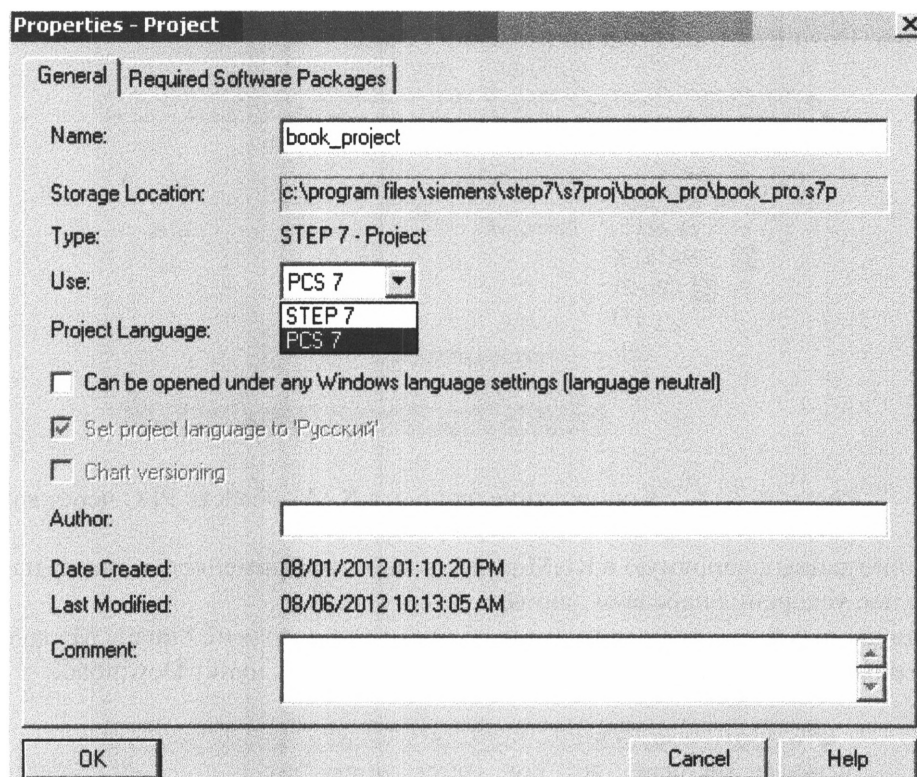


Рисунок 2.18.5: Окно редактирования свойств проекта

Опция Copy RAM to ROM меню PLC записывает все данные из встроенной области памяти CPU на карту памяти.

Опция Upload Station to PG выгружает данные станции низкого уровня на программатор.

Если осталось место на карте памяти – можно сохранить архив программы на карту памяти.

Опция Save to Memory Card – автоматическая архивация проекта и сохранение его на карте памяти. В результате на карте памяти будет полноценный архив проекта, содержащий всю программу, включающую комментарии и все прочее.

После выбора опции Save to Memory Card может появиться окно с попыткой доступа к диску 3,5A (дискета) – на нем необходимо нажать кнопку «Отмена».

После этого появится окно с заголовком «Save to memory Card in PLC»

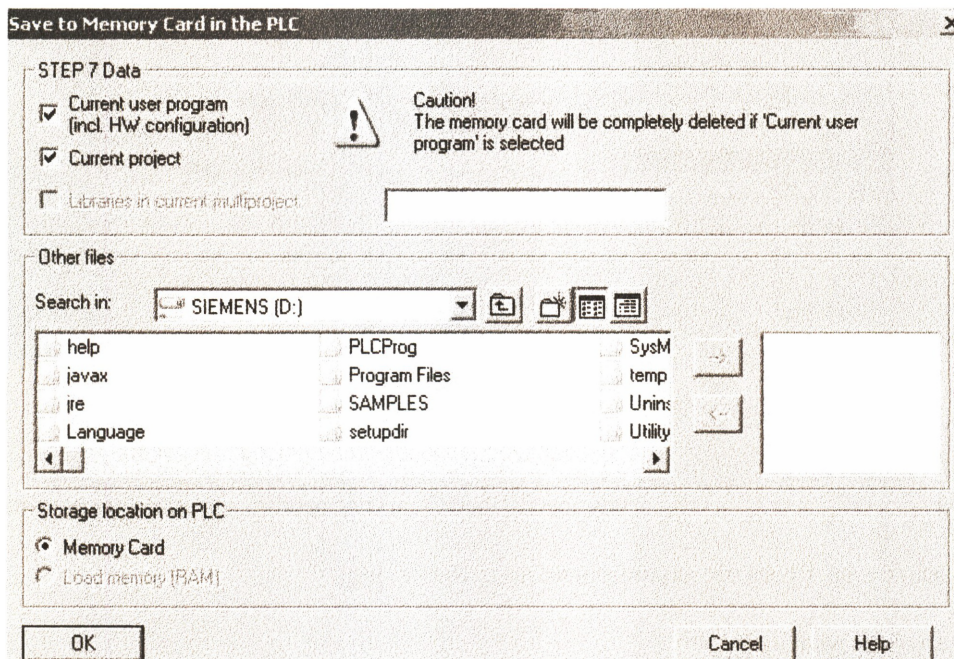


Рисунок 2.18.6: Окно настройки параметров сохранения

В нем необходимо выбрать, что именно загружать:

Current User program: - текущая программа пользователя (включая аппаратную конфигурацию)

Current Project – текущий проект

Также, можно загружать на карту памяти другие файлы, для этого их необходимо явно выбрать в окне настройки параметров сохранения. Например, можно загрузить уже имеющийся архив.

2.19 Настройка параметров архивирования

В меню Options выбрать опцию Customize и в появившемся окне выбрать вкладку Archiving

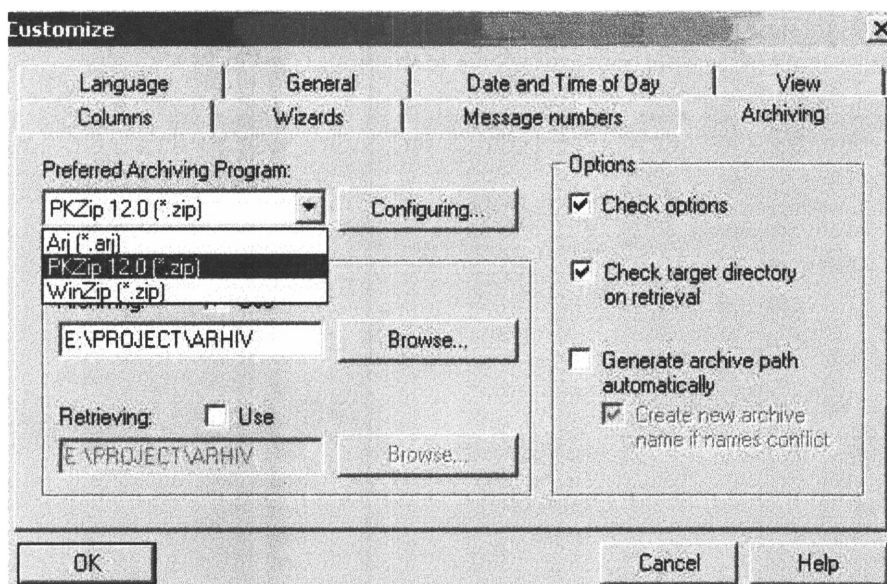


Рисунок 2.19.1: Окно настройки параметров архивирования

Можно указать еще 1 backup, активировав поле Retrieving.

Создание архива руками:

Закреть архивируемый проект

В Simatic Manager выбрать опцию Archive меню File, в появившемся окне выбрать проект, который необходимо заархивировать и нажать кнопку OK

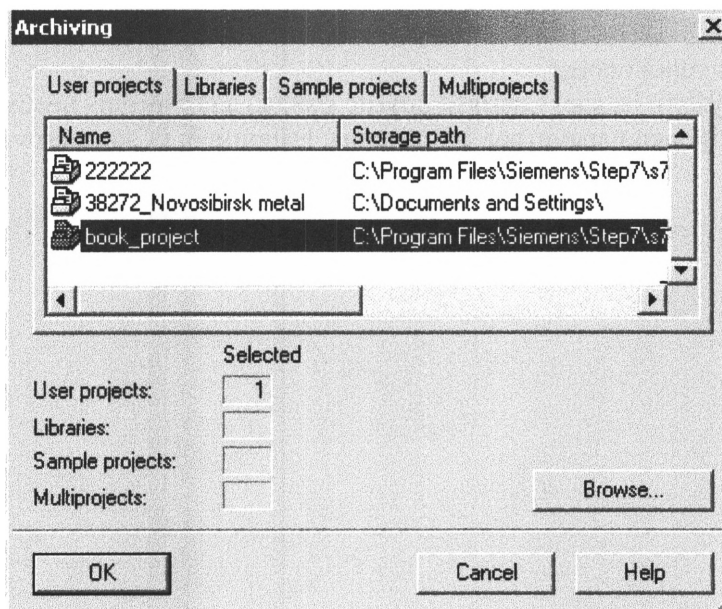


Рисунок 2.19.2: Окно создание архива

В следующем окне необходимо указать конечный путь сохранения архива, то есть место на жестком диске программатора, в котором будет находиться конечный архив

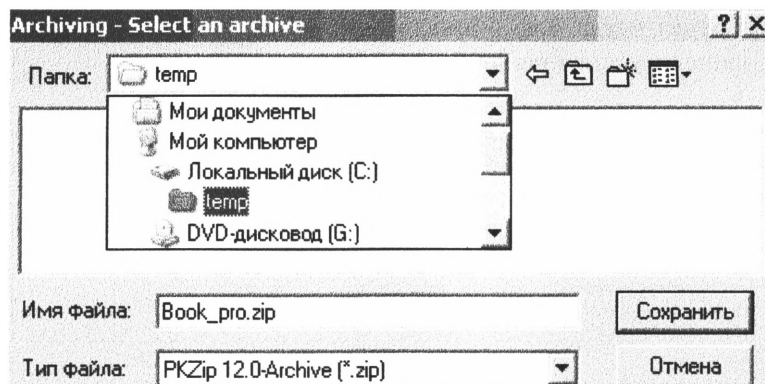


Рисунок 2.19.3: Выбор архива

Далее будет предложено разбить архив на части. Если в этом нет необходимости – выбрать пункт No.

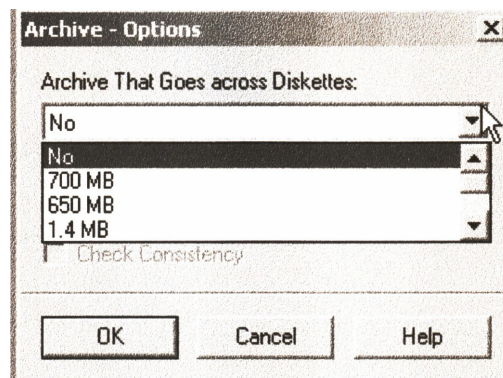


Рисунок 2.19.4: Настройка архива

Для того чтобы записать архив на карту памяти необходимо открыть меню PLC, выбрать пункт Save to Memory Card, в появившемся окне выбрать архив, который необходимо записать, нажать кнопку --> , после чего нажать кнопку Ok.

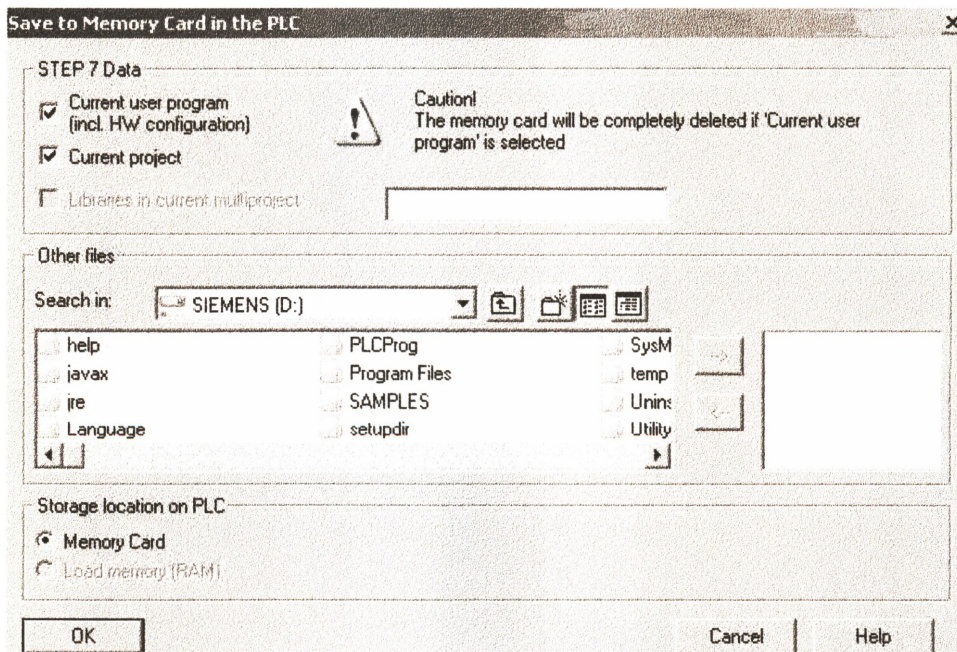


Рисунок 2.19.5: Окно Save to memory card

Если вместо архива будет указан какой-либо файл – на карту памяти будет записан указанный файл. Данная функция работает как обычная запись.

Чтобы извлечь архив (другой файл) из карты памяти на программатор, необходимо выделить папку Blocks, выбрать меню PLC и выбрать опцию Retrieve from memory Card. После этого появятся 2 окна, в которых необходимо указать, что копировать с карты памяти и куда. В данном примере с карты памяти копируются ранее записанные на нее ярлыки.

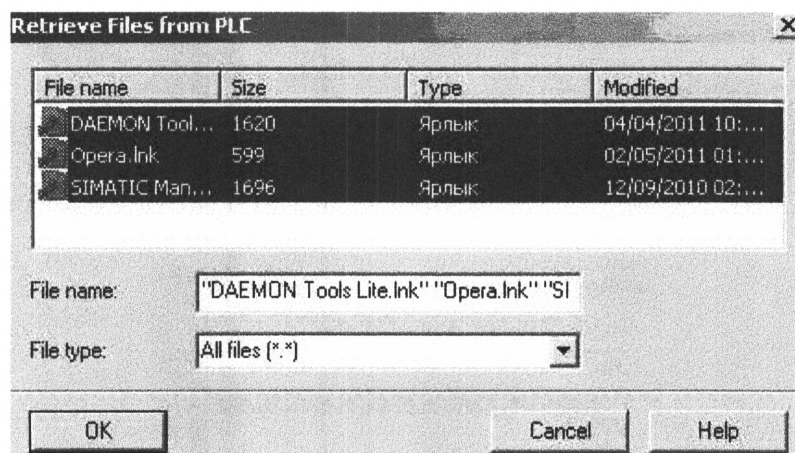




Рисунок 2.19.6: Извлечение данных с карты памяти

2.20 On-line программа

Все программные блоки, находящиеся в памяти компьютера относятся к off-line программе, а загруженные в CPU контроллера – к on-line программе. Для переключения режима

отображения с off-line на on-line необходимо нажать на кнопку  панели инструментов программы Simatic Manager. Для обратного переключения режима отображения –

необходимо нажать кнопку  панели инструментов программы Simatic Manager. Online-программа содержит не только программные блоки, созданные программистом, но и блоки, указывающие контроллеру на его аппаратную часть (папку System Data и SFB-блоки). Если в памяти CPU находится старая, ненужная программа – ее можно удалить. Для этого необходимо переключить режим отображения на on-line, выделить все программные блоки, нажать правую кнопку мышки и выбрать пункт delete, что приведет к удалению части блоков. Останутся только системные блоки, которые трогать нет необходимости. После удаления блоков on-line программы необходимо загрузить в CPU контроллера всю станцию проекта новой программы.

В режиме отображения on-line в папке проекта значок CPU отображается с диагностическим символом.

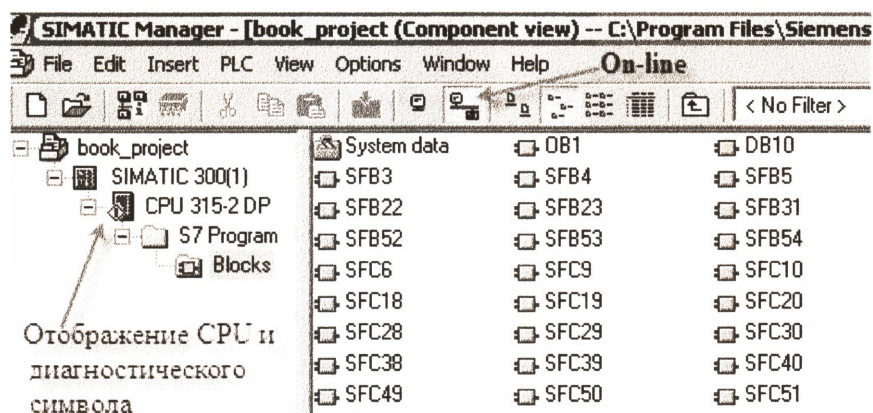


Рисунок 2.20.1: Отображение On-line программы

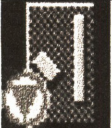






Вид значка	Режим работы CPU	Вид значка	Режим работы CPU
	Режим STOP (останов CPU)		режим STARTUP (запуск)
	режим RUN (выполнение)		режим HOLD (приостановка)
	Несоответствие спроектированной и фактической конфигурации.		Отказ, модуль неисправен: а) Диагностическое прерывание б) Ошибка ввода/вывода в) Активен светодиод ошибки
	Режим STOP, вызванный режимом STOP в другом CPU при многопроцессорной обработке		

Таблица 2.20.1: Диагностические символы CPU

2.21 Таблица символов Symbol Table

Таблица символов – утилита Simatic Manager, предназначенная для отображения переменных проекта и позволяющая присваивать переменным символьные имена. Допускается в качестве имени использовать кириллические символы и пробелы. Таблица символов глобальна и действует во всем проекте. В проекте может быть только одна таблица символов.

Для того чтобы создать таблицу символов необходимо открыть папку S7-programm, нажать правую кнопку мыши в пустом поле и выбрать пункты Insert New Object --> Symbol Table.

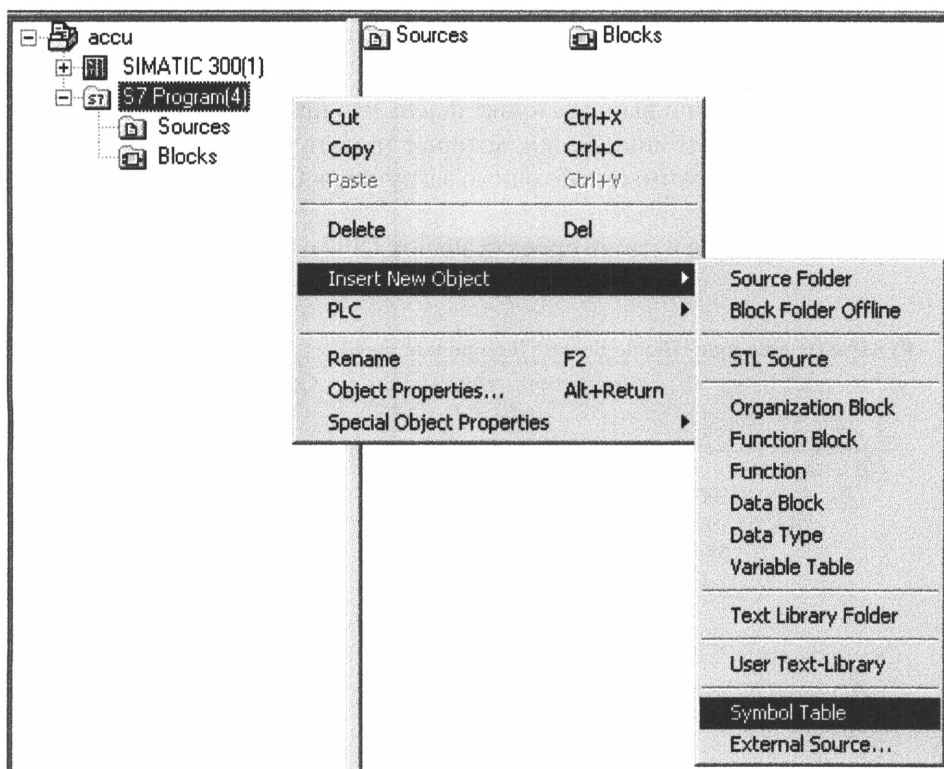


Рисунок 2.21.1: Добавление таблицы символов

Как правило, таблица символов имеет имя Symbols и находится в папке S7 Program. Её можно переименовать. Для этого необходимо щелкнуть по ней правой кнопкой мыши, выбрать пункт rename, ввести новое имя и нажать enter.

Другой вариант создания таблицы символов – через утилиты HW Config.

Данный способ позволяет быстро присвоить переменным из области отображения входов и выходов конкретные, символьные имена.

Для этого необходимо открыть утилиту Hw Config, выбрать блок входов либо выходов, нажать правую кнопку мыши и выбрать пункт Edit Symbols.

В появившемся после этого окне в поле Symbol пишутся символьные имена переменных, в поле Data Type – тип данных переменных, в поле Comment – комментарий. Текст, указанный в поле Symbol будет в дальнейшем, опционально отображаться в программном коде. После заполнения указанных полей следует последовательно нажать кнопку Add to Symbols и OK. Для того чтобы изменения вступили в силу после изменения таблицы символов, необходимо ее сохранить.

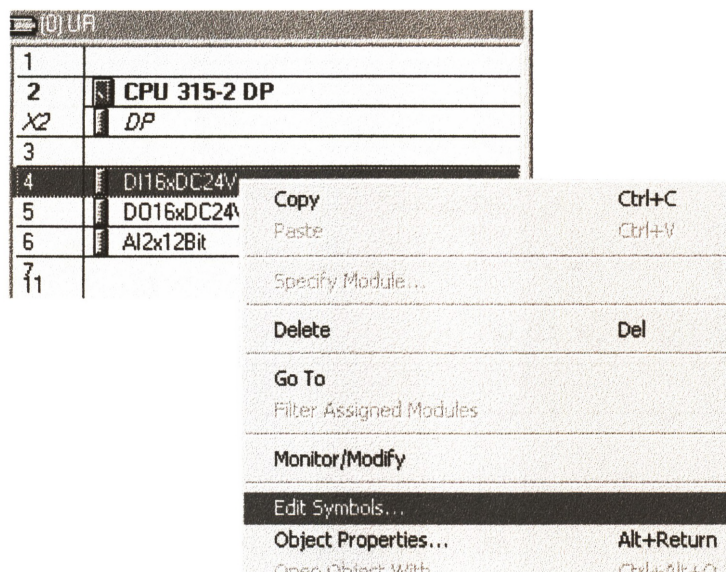


Рисунок 2.21.2: Меню свойств входного модуля

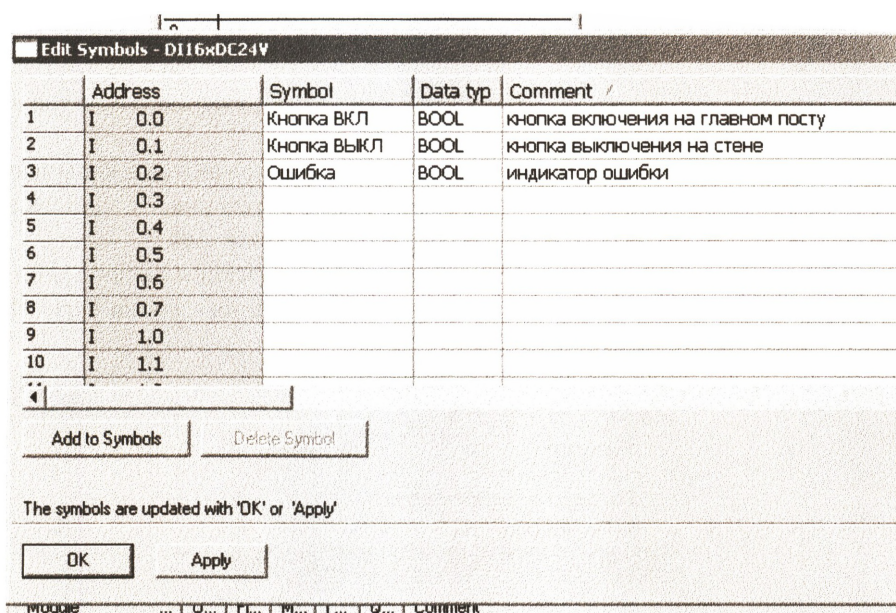


Рисунок 2.21.3: Редактирование символов

Network 18: Title:

A	"кнопка ВКЛ"	IO.0
AN	"кнопка ВЫКЛ"	IO.1
AN	"Ошибка"	IO.4
=	"Лампа1"	Q4.0
=	"Пуск Двигателя"	Q4.1

Network 18: Title:

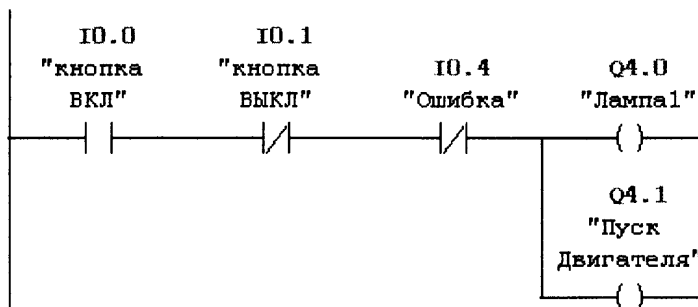


Рисунок 2.21.4: Отображение символов в программе

Символьные имена переменных из таблицы переменных отображаются в кавычках. Отображение символьной информации в коде отключается путем снятия галочек Symbolic Representation и Symbol Information меню View, подменю Display with. Таблица символов и утилите LAD/STL/FBD вызывается посредством выбора подменю Symbol Table в меню Options или посредством нажатия сочетания клавиш Ctrl+Alt+T

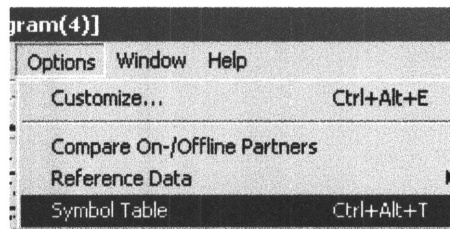


Рисунок 2.21.5: Запуск таблицы символов

При написании кода допускается использовать имена переменных. После ввода имени переменной без кавычек, её адрес пропишется автоматически.

A	"кнопка ВКЛ"	IO.0
AN	"кнопка ВЫКЛ"	IO.1
=	"лампа"	

Рисунок 2.21.6: Использование символьного имени переменных

2.22 Таблица переменных VAT (Variable Table)

Таблица переменных – утилита Simatic Manager предназначенная для мониторинга и отладки всех областей памяти кроме локального стека.

В проекте может быть несколько таблиц переменных. В каждой из них можно отслеживать переменные, относящиеся к отдельному объекту.

Если переменной было ранее присвоено символьное имя – оно будет отображаться в таблице переменных в поле Symbol.

Для того чтобы создать таблицу переменных необходимо открыть папку S7-programm, в ней открыть папку blocks, нажать правую кнопку мыши в пустом поле и выбрать пункты Insert New Object -->Variable Table.

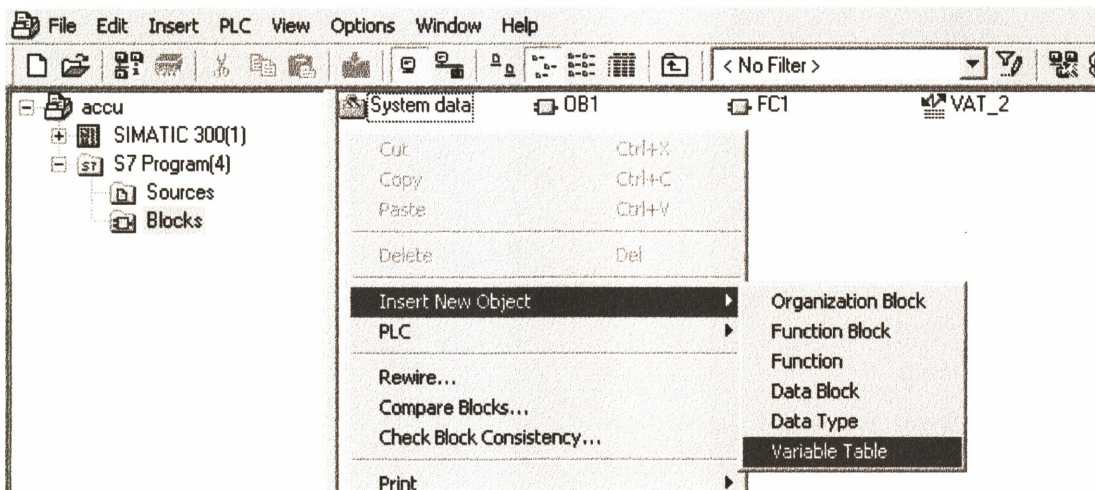


Рисунок 2.22.1: Добавление таблицы переменных

В таблице переменных в поле Address (адрес) вводится адрес отслеживаемой переменной. В поле Display format указывается формат отображения наблюдаемой переменной.

Table Edit Insert PLC Variable View Options Window Help						
VAT_2 -- accu\S7 Program(4)						
	Address	Symbol	Display format	Status value	Modify va	
1	MW 20		HEX	Default		
2	I 0.1	"кнопка ВЫКЛ"	BO	Define Default		
3	I 0.2		BO	Binary		
4	Q 4.0	"Лампа1"	BO	Boolean		
5	Q 4.1	"Пуск Двигателя"	BO	Decimal		
6	MW 20		HEX	Hexadecimal		
7	I 0.0	"кнопка ВКЛ"	BO	Floating-Point		
8				Character		
				Date		

Рисунок 2.22.2: Таблица переменных

2.23 Перекрестные ссылки (Reference data)

Перекрестные ссылки – утилита Simatic Manager предназначенная для навигации по данным проекта. Наглядно показывает, в каких блоках использована та или иная ячейка памяти.

Перекрестные ссылки в память ПЛК не загружаются.

Для запуска утилиты Reference data необходимо, чтобы тип проекта был Step7. Проверить параметр типа проекта можно, как показано на рисунке. При вызове утилиты из Simatic Manager курсор должен находиться на папке blocks.

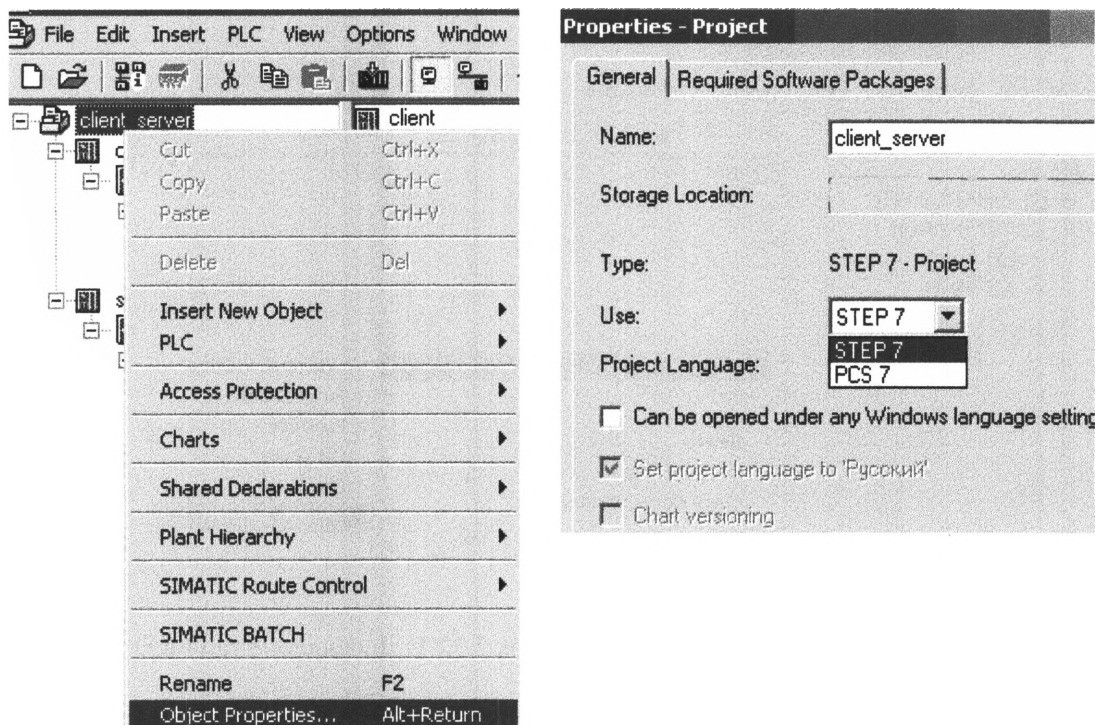


Рисунок 2.23.1: Изменение типа проекта

Утилита Reference Data вызывается из меню Options. В появившемся окне следует выбрать Cross-reference. В столбце Type буква W означает Write (запись в ячейку памяти), буква R – Read (считывание из ячейки памяти)

Address (symbol)	Block (symbol)	Type	Language	Location
DB1.DBDO (db_clen...	OB1	W	STL	NW 3 Sta 1 /CALL
DB1.DBDO4 (db_clen...	OB1	R	STL	NW 2 Sta 1 /CALL
DB1.DBW0	OB1	R	STL	NW 4 Sta 1 /L
DB2.DBX0.0	OB1	R	STL	NW 2 Sta 1 /CALL
DB2.DBX4.0	OB1	R	STL	NW 3 Sta 1 /CALL
I 0.0	OB1	R	STL	NW 2 Sta 1 /CALL
I 0.1	OB1	R	STL	NW 5 Sta 1 /CALL
IW 0	OB1	R	STL	NW 1 Sta 1 /L
MW 20	OB1	W	STL	NW 2 Sta 1 /CALL
MW 50	OB1	W	STL	NW 5 Sta 1 /CALL
Q 20.0	OB1	W	STL	NW 2 Sta 1 /CALL
Q 25.0	OB1	W	STL	NW 5 Sta 1 /CALL
QW 4	OB1	W	STL	NW 4 Sta 2 /T

Рисунок 2.23.2: Общий вид утилиты Cross References

2.24 Локальный стек

Для того чтобы переменные имели осмысленные имена удобно использовать локальный стек. В верхней части окна LAD/STL/FBD поле переменных, размещаемых в локальной области памяти ЦПУ. Переменные, используемые в локальном стеке, действуют только в пределах текущего блока. В программном коде перед именами переменных будет отображаться символ #. Локальный стек относится к RAM-памяти.

Для добавления локальной переменной, необходимо нажать на кнопку + в поле temp, выделить пустое поле в конец списка, ввести имя переменной, её формат данных и комментарий. Уже имеющиеся переменные типа OB1_DATE_TIME удалять не рекомендуется.

Contents Of: 'Environment\Interface\TEMP'				
	Name	Data Type	Address	Comment
[-] [-] Interface				
[-] [-] TEMP	OB1_MIN_CYCLE	Int	8.0	Minimum cycle t
	OB1_MAX_CYCLE	Int	10.0	Maximum cycle t
	OB1_DATE_TIME	Date...	12.0	Date and time C
	cos_phiR	Real	20.0	косинус угла фи
	powerR	Real	24.0	Мощность
	voltageR	Real	28.0	Напряжение
	curR	Real	32.0	Сила тока

Рисунок 2.24.1: Отображение области ввода локальных переменных

Name – имя переменной, Data Type – тип данных переменной, Comment – комментарий.

Рекомендуется давать переменным осмысленные имена и в названии переменных последний символ отводить типу данных переменной. (R-real, I-int, b-bool, D-Dint) .

При программировании «переходов» с одного уровня на другой на языке LAD, ЦПУ автоматически использует вспомогательные локальные переменные. Обращение к переменным - адресное. Адрес также назначается автоматически.

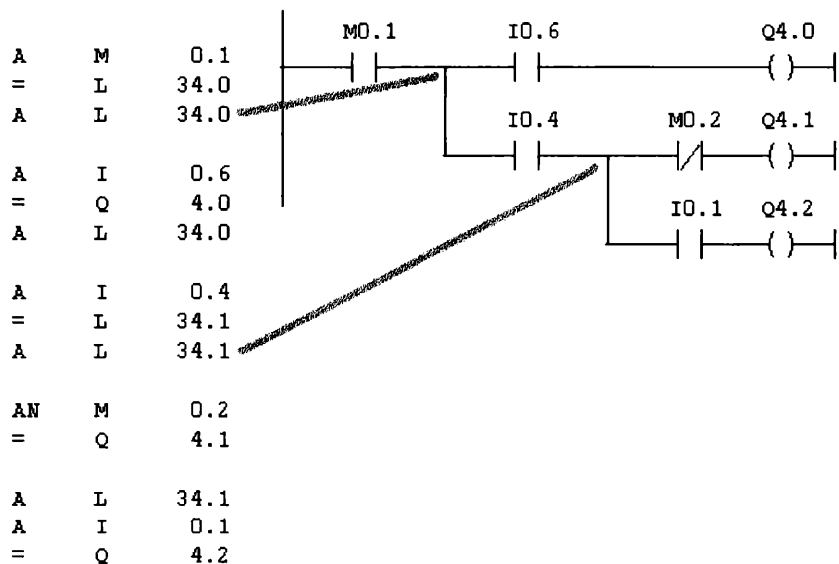


Рисунок 2.24.2: Использование вспомогательных локальных переменных

3. ЛОГИЧЕСКИЕ ОПЕРАЦИИ

3.1 Элементарные логические операции

В среде программирования Step-7 возможны следующие элементарные логические операции:

Название операции	A (And) «И»	AN (And not) «НЕ-И»	O (Or) «ИЛИ»	ON (Or Not) «ИЛИ-НЕ»																																																												
STL	A I 0.0 A I 0.1 = Q 4.0	AN I 0.0 AN I 0.1 = Q 4.0	O I 0.0 O I 0.1 = Q 4.0	ON I 0.0 ON I 0.1 = Q 4.0																																																												
LAD																																																																
Схема электрическая																																																																
FBD																																																																
Таблица истинности	<table border="1"> <tr><td>IO.0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>IO.1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>Q4.0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr> </table>	IO.0	0	1	0	1	IO.1	0	0	1	1	Q4.0	0	0	0	1	<table border="1"> <tr><td>IO.0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>IO.1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>Q4.0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr> </table>	IO.0	0	1	0	1	IO.1	0	0	1	1	Q4.0	1	0	0	0	<table border="1"> <tr><td>IO.0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>IO.1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>Q4.0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr> </table>	IO.0	0	1	0	1	IO.1	0	0	1	1	Q4.0	0	1	1	1	<table border="1"> <tr><td>IO.0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>IO.1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr> <tr><td>Q4.0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr> </table>	IO.0	0	1	0	1	IO.1	0	0	1	1	Q4.0	1	1	1	0
IO.0	0	1	0	1																																																												
IO.1	0	0	1	1																																																												
Q4.0	0	0	0	1																																																												
IO.0	0	1	0	1																																																												
IO.1	0	0	1	1																																																												
Q4.0	1	0	0	0																																																												
IO.0	0	1	0	1																																																												
IO.1	0	0	1	1																																																												
Q4.0	0	1	1	1																																																												
IO.0	0	1	0	1																																																												
IO.1	0	0	1	1																																																												
Q4.0	1	1	1	0																																																												
Комментарий	Выход Q4.0 установится в «1», если И IO.0 И IO.1 равны «1»	Выход Q4.0 установится в «1», если И IO.0 И IO.1 находятся в состоянии «0».	Выход Q4.0 установится в «1», хотя бы один из входов находится в состоянии «1».	Выход Q4.0 установится в «1», хотя бы один из входов находится в состоянии «0».																																																												

Таблица 3.1.1: Элементарные логические операции (начало)

Название операции	X (Exclusive OR) Исключающее или	XN (Exclusive OR NOT)	«И» перед «Или»																																																												
STL	X I 0.0 X I 0.1 = Q 4.0	XN I 0.0 XN I 0.1 = Q 4.0	A I 0.0 A M 10.0 O A I 0.2 A M 0.3 O M 10.1 = Q 4.0																																																												
LAD																																																															
Схема электрическая																																																															
FBD																																																															
Таблица истинности	<table><tr><td>I0.0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>I0.1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>Q4.0</td><td>0</td><td>1</td><td>1</td><td>0</td></tr></table>	I0.0	0	1	0	1	I0.1	0	0	1	1	Q4.0	0	1	1	0	<table><tr><td>I0.0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>I0.1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>Q4.0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr></table>	I0.0	0	1	0	1	I0.1	0	0	1	1	Q4.0	1	0	0	1	<table><tr><td>I0.0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>M10.0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>I0.2</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>M0.3</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>M10.1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>Q4.0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr></table>	I0.0	0	1	0	1	M10.0	0	0	1	1	I0.2	0	1	0	0	M0.3	0	1	0	1	M10.1	0	0	1	0	Q4.0	0	1	0	1
I0.0	0	1	0	1																																																											
I0.1	0	0	1	1																																																											
Q4.0	0	1	1	0																																																											
I0.0	0	1	0	1																																																											
I0.1	0	0	1	1																																																											
Q4.0	1	0	0	1																																																											
I0.0	0	1	0	1																																																											
M10.0	0	0	1	1																																																											
I0.2	0	1	0	0																																																											
M0.3	0	1	0	1																																																											
M10.1	0	0	1	0																																																											
Q4.0	0	1	0	1																																																											
Комментарий	Выход Q4.0 установится в «1»,если входы находятся в разных логических состояниях.	Выход Q4.0 установится в «1», если входы находятся в одинаковых логических состояниях	Q4.0 установится в «1» если: а) M1.0 =1 б) И I0.0 И M10.0 =1 в) И I0.2 И I0.3 =1																																																												

Таблица 3.1.2: Элементарные логические операции (продолжение)

Логические действия над битовыми ячейками памяти имеют следующий вид:

Операнд логической операции **Обращение к битовой переменной**

Пример:



В качестве переменных могут быть использованы данные типа BOOL из следующих областей памяти: I(область отображения входов), Q(область отображения выходов), M(меркерная область данных), L(область данных локального стека), D(блок данных), T(таймеры), C(счетчики).

Пример:

STL	FBD	LAD
A I0.0		
A I0.1		
= Q4.0		

Таблица 3.1.3: Использование переменных типа данных BOOL

В данном примере используется логическая операция «2И» с типом данных bool из области отображения входов «I».

Выход Q4.0 установится в логическую единицу, если **И** I0.0 **И** I0.1 равны логической единице.

Пример:

STL	FBD	LAD
A I 0.0		
O M 0.0		
O M 77.5		
= Q 4.0		

Таблица 3.1.3: Логическая операция «ИЛИ»

В данном примере используется логическая операция «3 ИЛИ». Выход Q4.0 установится в единицу, если хотя бы одна из переменных I0.0, M0.0, M77.5 находится в состоянии логической единицы.

Естественно, увидев такой пример, в голове возникает логичный вопрос: «почему же перед переменной I0.0 стоит A (And), а не O (Or)»?

А все дело в том, что в первой строчке происходит первичный опрос переменной (First check). И, теоретически, для контроллера нет никакой разницы, какая буква (A, O, X) будет стоять при первичном опросе. Код будет работать абсолютно одинаково. Однако, фирма Siemens рекомендует осуществлять первичный опрос булевой переменной, используя оператор A (And).

Итак, контроллер лишь опрашивает переменную, стоящую в первой строчке, а операнд логического действия, которое будет совершаться над данной и последующей переменной, контроллер узнает от операнда второй переменной.

При успешном выполнении первичного опроса бит FC (first check) устанавливается в «1» и выполняются дальнейшие логические действия.

Опрос переменной на состояние «1» осуществляется операторами A, O, X.

Опрос переменной на состояние «0» осуществляется операторами AN, ON, XN.

Битовые инструкции A, O, X, AN, ON, XN устанавливают бит FC в «1».

Вариант 1	Вариант 2	Вариант 3	LAD
A I 0.0	O I 0.0	X I 0.0	
A I 0.1	A I 0.1	A I 0.1	
A I 0.2	A I 0.2	A I 0.2	
= Q 4.0	= Q 4.0	= Q 4.0	

Таблица 3.1.4: Пояснение 1 к определению первичного опроса

Во всех трех вариантах код работает абсолютно одинаково!

Выход Q4.0 находится в состоянии логической единицы только тогда, когда все три входа I0.0 И I0.1 И I0.2 находятся в состоянии «1». Между переменными в первой и второй строки будет выполнена логика, операнд которой указан во второй строке.

Вариант 1	вариант 2	вариант 3	LAD
AN I 0.0	ON I 0.0	XN I 0.0	
A I 0.1	A I 0.1	A I 0.1	
A I 0.2	A I 0.2	A I 0.2	
= Q 4.0	= Q 4.0	= Q 4.0	

Таблица 3.1.5: Пояснение 2 к определению первичного опроса

Выход Q4.0 находится в состоянии «1» только тогда, когда I0.0 находится в состоянии «0», а I0.1 И I0.2 находятся в состоянии «1». (когда И I0.1 И I0.2 = «1» И I0.0=0»).

Рассмотри еще несколько примеров:

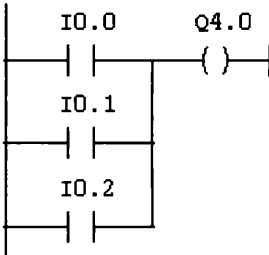
Строка 1	A I 0.0	Первичный опрос переменной I0.0 на состояние логической единицы. Если I0.0 находится в состоянии логической единицы, то бит FC (First check) слова состояния установится в единицу и будет осуществлено логическое действие с переменной из первой строки, операнд которого показан во второй строке.	
Строка 2	O I 0.1	Операнд логического действия между переменными в первой и второй строках. В данном случае выполнится логическая операция «2 ИЛИ» между переменными I0.0 и I0.1	
Строка 3	O I 0.2	третья переменная	
Строка 4	= Q 4.0	выход (выгрузка RLO в Q4.0)	
В данном примере: Выход Q4.0 установится в «1», если хотя бы один из входов I0.0, I0.1, I0.2 будет находится в состоянии «1».			

Таблица 3.1.5: Пояснение 3 к определению первичного опроса

Итак, для того, чтобы выполнить логическое действие между переменными нужно установить бит FC (first check) в единицу. Для этого в первой строке необходимо опросить первую переменную, а операнд логического действия, который необходимо сделать между первой и второй переменной нужно указывать во второй строке.

Пример:

A I 0.0	опрос переменной I0.0 на состояние логической единицы. Для опроса может использоваться любой операнд A, O, X. Фирма Siemens рекомендует использовать операнд «A». Если переменная находится в состоянии логической единицы, то бит FC установится в единицу и дальнейшие действия будут выполнены. Если переменная находится в состоянии логического нуля, бит FC в единицу не установится и дальнейшие действия выполнены не будут.
A I 0.1	Если результат первичного опроса положителен, т.е. переменная I0.0 находится в состоянии логической единицы, будет выполнено логическое действие между ней и переменной I0.1. Операнд логического действия (в нашем случае A – AND «И») указан во второй строке.
= Q 4.0	Выход Q4.0 установится в «1», если И I0.0 И I0.1 равны «1».

Таблица 3.1.6: Пример

Пример:

A I 0.0	опрос переменной I0.0 на состояние логической единицы. Для опроса может использоваться любой операнд A, O, X. Рекомендуется использовать операнд «A»
X I 0.1	Если результат первичного опроса положителен, т.е. переменная I0.0 находится в состоянии логической единицы, будет выполнено логическое действие между ней и переменной I0.1. Операнд логического действия (в нашем случае X – исключающее ИЛИ) указан во второй строке.
= Q 4.0	Выход Q4.0 установится в логическую единицу, если входы находятся в разных логических состояниях

Таблица 3.1.7: Пример

Опрос переменной на состояние «0» осуществляется операторами AN, ON, XN.

Пример:

AN I 0.0	опрос переменной I0.0 на состояние логического нуля. Для опроса может использоваться любой операнд AN, ON, XN. Рекомендуется использовать операнд «AN». Если переменная находится в состоянии «0», то бит FC установится в «1» и дальнейшие действия будут выполнены. Если переменная находится в состоянии «1», бит FC в «1» не установится и дальнейшие действия выполнены не будут.
X I 0.1	Если результат первичного опроса положителен, т.е. переменная I0.0 находится в состоянии «1», будет выполнено логическое действие между ней и переменной I0.1. Операнд логического действия (в данном случае X – исключающее ИЛИ) указан во второй строке.
= Q 4.0	Выход Q4.0 установится в «1», если входы находятся в одинаковых логических состояниях. Т.е. при I0.0=«0» и I0.1=«0» и при I0.0=«1»

Таблица 3.1.8: Пример

Естественно, что при программном описании датчиков и при переходе от релейно-контакторной схемы на ПЛК нормально-разомкнутые контакты чаще опрашиваются на единицу, а нормально-замкнутые – на ноль.

Составить программу для управления электрической тележкой. Имеются следующие входные данные:

Устройство	Адрес	Комментарий
Кнопка «вперед»	I0.0	подать напряжение на реле «вперед»
Выключатель концевой (SQ) движения тележки вперед	I0.1	прекратить движение тележки вперед
Реле (Q) запуска двигателя вперед	Q0.0	запустить двигатель в направлении «вперед»
Кнопка «стоп»	I0.2	прекратить движение тележки назад
Кнопка «назад»	I1.0	подать напряжение на реле «назад»
Выключатель концевой (SQ) движения тележки назад	I1.1	прекратить движение тележки назад
Реле (Q) запуска двигателя назад	Q1.0	запустить двигатель в направлении «назад»

Таблица 3.1.9: Исходные данные задачи

Решение:

движение тележки вперед при зажатой кнопке						
A	I	0.0	//нажата кнопка "вперед"			
AN	I	0.1	//не сработал SQ движения вперед			
AN	I	0.2	// не нажата кнопка "стоп"			
AN	I	1.0	// не нажата кнопка «назад»			
AN	Q	1.0	//не замкнуто Q тележки назад			
=	Q	0.0	// напряжение на Q движения вперед			
движение тележки назад при зажатой кнопке						
A	I	1.0	//нажата кнопка "назад"			
AN	I	1.1	//не сработал SQ движения назад			
AN	I	0.2	// не нажата кнопка "стоп"			
AN	I	0.0	// не нажата кнопка "вперед"			
AN	Q	0.0	//не замкнуто Q тележки вперед			
=	Q	1.0	//напряжение на Q движения назад			

Таблица 3.1.10: Решение задачи на движение тележки

В данном случае тележка будет ехать до тех пор, пока оператор держит нажатой кнопку «вперед» или «назад». По нажатию на кнопку «стоп» тележка остановится. Если требуется осуществлять движение тележки по нажатию кнопки и до тех пор, пока не сработает концевой выключатель, либо пока не будет нажата кнопка «стоп», то необходимо «зашунтировать» вход выходом, т.е. осуществить «самоподхват»

движение тележки вперед при однократном нажатии кнопки									
A(A I 0.0 //нажата кнопка "вперед" O Q 0.0 // «самоподхват») AN I 0.1 //не нажат SQ движения вперед AN I 0.2 // не нажата кнопка "стоп" AN I 1.0 // не нажата кнопка "назад" AN Q 1.0 //не замкнуто Q назад = Q 0.0 // напряжение на Q движения вперед									
движение тележки назад при однократном нажатии кнопки									
A(A I 1.0 //нажата кнопка "назад" O Q 1.0 // «самоподхват») AN I 1.1 // не нажат SQ движения назад AN I 0.2 // не нажата кнопка "стоп" AN I 0.0 // не нажата кнопка "вперед" AN Q 0.0 //не замкнуто Q назад = Q 1.0 // напряжение на Q движения назад									

Таблица 3.1.11: Решение задачи на движение тележки (продолжение)

Теперь, в коде появились, непонятные на первый взгляд, скобки. Пугаться их не надо – мы рассмотрим их в конце этой главы.

3.1.1 Инвертирование результата логической операции

Инвертирование итога логической операции осуществляется с помощью операнда NOT

Пример:

A	I	0.0	Опрос переменной I0.0 на состояние «1»
A	I	0.1	Если результат первичного опроса положителен, т.е. переменная I0.0 находится в состоянии «1», будет выполнено логическое действие между ней и переменной I0.1. Операнд логического действия (в нашем случае A – AND «И») указан во второй строке.
NOT			инвертирование результата логической операции
=	Q	4.0	Выход Q4.0 установится в «1», если ни одна из переменных I0.0 И I0.1 не равна «1». (Если И I0.0 И I0.1 не находятся в состоянии «1»).

Таблица 3.1.1.1: Пример инвертирования результата логической операции

3.1.2 Исключающее ИЛИ с тремя переменными

STL	FBD	LAD
A I 0.0 X I 0.1 X I 0.2 = Q 4.0		Не отобразится
Q4.0 установится в «1» если только один из входов (любой) =«1», а остальные =«0»		

Таблица 3.1.2.1: Исключающее ИЛИ с тремя переменными

3.1.3 Присвоение (функция Assign)

Присваивает биту значение результата логической операции. Допускается выполнять присвоение сразу нескольким переменным.

Пример:

STL	FBD	LAD
A I 0.0 AN I 0.1 A I 0.2 = Q 4.0 = Q 4.1		
В случае, когда I0.0 И I0.2 находятся в состоянии «1», а бит I0.1 находится в состоянии «0», битам Q4.0 и Q4.1 будет присвоено значение «1». В противном случае битам Q4.0 и Q4.1 будет присвоено значение «0»		

Таблица 3.1.2.2: Присвоение

3.1.4 Сброс и установка бита (функции Set и Reset), функции установки, сброса и выгрузки

Set – устанавливает текущее значение РЛО в «1», бит FC сбрасывается в «0».

(Принудительная установка бита RLO в «1»)

Clr – сбрасывает текущее значение РЛО в «0», бит FC сбрасывается в «0».

S – Записать «1» в адресуемый бит (при наличии 1 в бите RLO)

R – Записать «0» в адресуемый бит. (при наличии 1 в бите RLO)

= считать значение из бита РЛО и записать в указанную ЯП, старое значение в бите РЛО остается. Знак «=» является знаком выгрузки.

Пример:

Установка бита (Set)	Сброс бита (Reset)
Устанавливает бит, при РЛО=1	Сбрасывает бит, при РЛО=1
A I 0.0 AN I 0.1 A I 0.2 S Q 4.0	A I 0.0 AN I 0.1 A I 0.2 R Q 4.0
В случае, когда I0.0 И I0.2 находятся в состоянии «1», а бит I0.1 находится в состоянии «0», т.е. когда РЛО=1, значение бита Q4.0 будет установлено в «1» и останется в «1», даже если значение РЛО изменится на «0». Q4.0 будет находиться в состоянии «1» до тех пор, пока его не сбросят в состояние «0» функцией Reset	В случае, когда I0.0 И I0.2 находятся в состоянии «1», а бит I0.1 находится в состоянии «0», т.е. когда РЛО=1, значение бита Q4.0 будет сброшено в «0» и останется в «0», даже если значение РЛО изменится на «0».

Таблица 3.1.4.1: Использование функций Set и Reset

Пример:

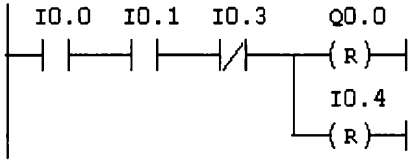
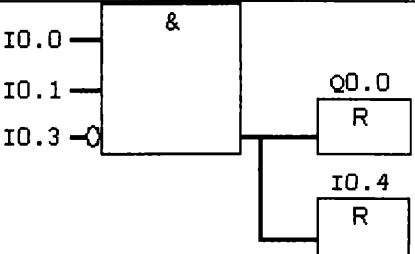
STL	Комментарий	
A I 0.0	//первичный опрос бита I0.0	
A I 0.1	// Логическое действие «2И» с битами I0.0 и I0.1	
AN I 0.3	//Логическое действие «2И-НЕ» результатом строки 2 и I0.3	
R Q 0.0	//сброс выходного бита Q0.0	
R I 0.4	//сброс входного бита I0.4	
LAD/FBD		

Таблица 3.1.4.2: Пример использования функции Reset

3.1.5 Триггеры

Триггер – функция, имеющая два входа S и R и один выход Q. При подаче сигнала на вход S, выход Q устанавливается в «1». При подаче сигнала на вход R, выход Q сбрасывается в «0». В зависимости от последовательности входов R и S различают триггеры с приоритетом установки и с приоритетом сброса.

Пример:

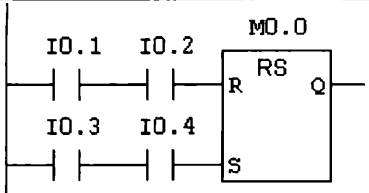
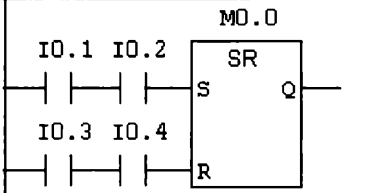
RS-триггер (с приоритетом установки)	SR-триггер (с приоритетом сброса)
Если подать на оба входа «1», выход установится в «1»	Если подать на оба входа «0», выход сбросится в «0»
A I 0.1 A I 0.2 R M 0.0 A I 0.3 A I 0.4 S M 0.0	A I 0.1 A I 0.2 S M 0.0 A I 0.3 A I 0.4 R M 0.0
	
<p>При I0.3=«1» и I0.4=«1» переменная M0.0 установится в «1» и будет = «1», даже если условие (I0.3=«1» и I0.4=«1») нарушится.</p> <p>При I0.1=«1» и I0.2=«1» переменная M0.0 сбросится в «0»</p> <p>При I0.3=«1» и I0.4=«1» и I0.1=«1» и I0.2=«1» переменная M0.0 установится в «1»</p>	<p>При I0.1=«1» и I0.2=«1» переменная M0.0 установится в «1» и будет = «1», даже если условие (I0.1=«1» и I0.2=«1») нарушится.</p> <p>При I0.3=«1» и I0.4=«1» переменная M0.0 сбросится в «0»</p> <p>При I0.3=«1» и I0.4=«1» и I0.1=«1» и I0.2=«1» переменная M0.0 сбросится в «0»</p>

Таблица 3.1.5.1: Пример использования триггеров

Пример:

По нажатию на кнопку «вкл» (I0.0) должна зажечься лампа Q4.0 и гореть, даже если кнопка «вкл» отжата. При нажатой кнопке «выкл» (I0.2) лампа должна потухнуть независимо от положения кнопки «вкл».

```

A I 0.1          //кнопка "выкл"
= M 0.2          //вспомогательный бит
R M 0.4          //вспомогательный бит

A I 0.0          //кнопка "вкл"
S M 0.4          //вспомогательный бит
A M 0.4          //вспомогательный бит
AN M 0.2         //вспомогательный бит
= Q 4.0          //лампа

```

3.1.6 Функции выделения фронтов

Существуют две битовые функции для выделения фронтов сигнала: **FP** (front positive) и **FN** (front negative).

Данные функции позволяют работать как с отдельными импульсами, так и с РЛО. При наличии фронта данные функции устанавливают бит памяти в «1». В качестве бита памяти локальные данные использовать нельзя, так как они актуальны только на момент выполнения текущего блока.

В каждом цикле сравнивается состояние бита RLO с его состоянием в предыдущем цикле.

При изменении уровня сигнала с «0» на «1» имеет место передний фронт импульса.

При изменении уровня сигнала с «1» на «0» имеет место задний фронт импульса.

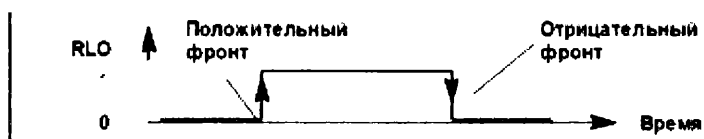


Рисунок 3.1.6.1: Положительный и отрицательный фронт импульса

Пример:

Выделение переднего фронта				Выделение заднего фронта			
A	I	0.0		A	I	0.0	
FP	M	0.0		FN	M	0.0	
=	Q	4.0		=	Q	4.0	
При изменении уровня RLO с «0» на «1» (в нашем случае бита I0.0), бит M0.0 установится в «1» на время равное времени цикла обработки OB1				При изменении уровня RLO с «1» на «0» (в нашем случае бита I0.0), бит M0.0 установится в «1» на время равное времени цикла обработки OB1.			

Таблица 3.1.6.1: Пример функции выделения фронтов

3.2 Сложные двоичные логические операции

Сложными двоичными операциями называют двоичные операции, содержащие вложенные выражения. Вложенные выражения используются для определения порядка выполнения логических операций. Первыми обрабатывают выражения, заключенные в скобки, то есть до выполнения выражений, находящихся за скобками

С целью временного сохранения результата логической операции (RLO), рассчитанного в некоторой точке программы используются вложенные выражения, которые обеспечивают выполнение одних команд раньше других.

При программировании на STL необходимо помнить о том, что приоритет оператора A (AND) выше приоритета операторов O (Or) и X (XOR), а приоритеты операторов Or и XOR одинаковы.

В языке программирования STL существуют следующие двоичные вложенные выражения:

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

Таблица 3.2.1: Сложные двоичные операции

3.2.1 A(: «И» с открывающей скобкой

Описание: A(:. Внутри скобок временно сохраняется результат логической операции. Бит FC (first check) скидывает в «0». Допускается максимум 7 вложений то есть допускается максимум 7 раз включать новые вложенные выражения в выражения, которые уже сами по себе являются вложенными, не завершая последних.

Пример:

STL	Релейная схема	LAD	FBD
<div>A(O I 0.0 O M 10.0) //-----// A(O I 0.2 O M 10.3) //-----// A M 10.1 = Q 4.0</div>			

Таблица 3.2.1.1: Пример использования открывающейся скобки

3.2.2): Закрывающая скобка

Инструкция) (закрытие вложения) - сопрягает ранее сохраненный RLO, с текущим RLO в соответствии с логической операцией, код которой был указан при открытии скобки. Бит FC (first check) скидывает в «0». Следующее после закрытия скобки действие будет являться первичным опросом.

3.2.3 Объединение AND-функций (И) в операторе OR (ИЛИ)

Комбинации функций OR(ИЛИ) и AND(И), можно записывать без скобок, при этом вначале выполняются функции AND (И). Затем результат, обрабатывается функцией OR

Пример:

STL	LAD	FBD
<div>A I 0.0 A I 0.1 O A I 0.2 A I 0.3 = Q 8.0</div>		

Таблица 3.2.1.1: Пример объединения функций 1

3.2.4 Объединение OR (ИЛИ) и XOR (Исключающее ИЛИ) в операторе AND (И)

Комбинации функций OR (ИЛИ) и AND (И), записываются с использованием скобок, с помощью которых указывается, что функции OR (ИЛИ) выполняются раньше функции AND (И).

Пример:

STL	LAD	FBD
<pre> A(O I 0.0 O I 0.1) A(O I 0.2 O I 0.3) = Q 1.0 </pre>		
<pre> A(X I 0.0 X I 0.1) A(X I 0.2 X I 0.3) = Q 1.0 </pre>	<p>На языке LAD не отобразится</p>	

Таблица 3.2.4.1: Пример объединения функций 2

В данном примере оператор открытой скобки объединен с функцией AND (И). Функция OR (ИЛИ) является вложенной. Закрывающая скобка связывает результат функции OR (ИЛИ) по логике функции AND (И).

3.2.5 Объединение функций AND (И) в операторе XOR (Исключающее ИЛИ)

Функции AND, выполняющиеся перед функцией XOR записываются в скобках. С помощью скобок сохраняются результаты функций AND (И) перед дальнейшим их комбинированием, по правилам функции XOR.

Пример:

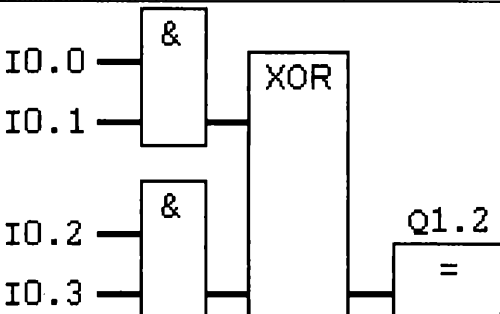
STL	LAD	FBD
X(A I 0.0 A I 0.1) X(A I 0.2 A I 0.3) = Q 1.2	На языке LAD не отобразится	
Выход Q1.2 установится в «1», если хотя бы в одной из скобок (IO.0 и IO.1) или (IO.2 и IO.3) выполнено условие функции And.		

Таблица 3.2.5.1: Пример объединения функций 3

3.2.6 Объединение функций OR (ИЛИ) в операторе XOR и наоборот

Функции OR (ИЛИ), выполняющиеся перед функцией XOR (Исключающее ИЛИ), записываются в скобках. С помощью скобок сохраняются результат функций OR (ИЛИ) перед дальнейшим их комбинированием, по правилам функции XOR.

Пример:

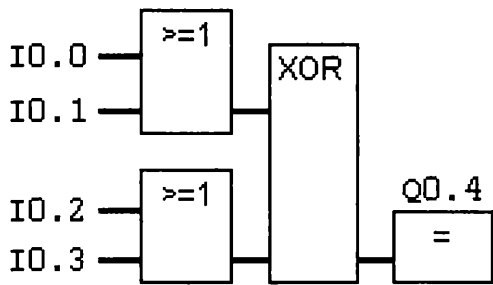
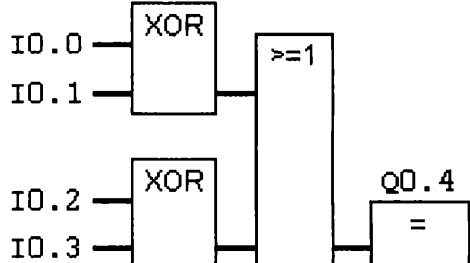
STL	LAD	FBD
X(O I 0.0 O I 0.1) X(O I 0.2 O I 0.3) = Q 0.4	На языке LAD не отобразится.	
O(X I 0.0 X I 0.1) O(X I 0.2 X I 0.3) = Q 0.4		

Таблица 3.2.6.1: Пример объединения функций 4

3.2.7 Инвертирование вложенных выражений

Для инвертирования результата вложенного выражения необходимо поставить дополнительный символ N в выражении открывающей скобки. Или используя, оператор NOT перед закрывающей скобкой.

Пример:

STL	LAD	FBD
<pre> AN(O I 0.0 O I 0.1) AN(X I 0.2 X I 0.3) = Q 1.6 </pre>		<p>В данном примере невозможно переключить отображения с языка STL на LAD или FBD</p>
<p>Выход Q1.6 установится в «1», если ни в одной из скобок (I0.0 и I0.1) и (I0.2 и I0.3) не выполнится условие вложенных функций. (Если И в первой скобке И во второй скобке условие не выполнится)</p>		

Таблица 3.2.7.1: Пример инвертирования вложенных выражений

3.2.8 Примеры

STL	FBD	LAD
A I 0.0 = Q 4.0		
A I 0.0 AN I 0.1 = Q 4.0		
A I 0.0 AN I 0.1 NOT A I 0.2 = Q 4.0		
A I 0.0 AN I 0.1 A I 0.2 O I 0.3 = Q 4.0		
A(O I 0.0 O I 0.3) AN I 0.1 A I 0.2 = Q 4.0		
A I 0.0 A(AN I 0.1 O I 0.3) A I 0.2 = Q 4.0		
A I 0.0 AN I 0.1 A(O I 0.2 O I 0.3) = Q 4.0		
A I 0.0 AN I 0.1 A I 0.2 = Q 4.0 = Q 4.1		

A(A I 0.0 AN I 0.1 O I 0.3) A I 0.2 = Q 4.0		<pre> graph LR IO0[IO.0] --- AND1[&] IO1[IO.1] --- AND1 AND1 --- OR1[>=1] IO3[IO.3] --- AND2[&] OR1 --- AND2 IO2[IO.2] --- AND2 AND2 --- Q40[Q4.0] </pre>
A I 0.0 A(AN I 0.1 A I 0.2 O I 0.3) = Q 4.0		<pre> graph LR IO1[IO.1] --- AND1[&] IO2[IO.2] --- AND1 AND1 --- OR1[>=1] IO0[IO.0] --- AND2[&] IO3[IO.3] --- AND2 OR1 --- AND2 AND2 --- Q40[Q4.0] </pre>
A(A I 0.0 O A I 0.3 A M 0.0) AN I 0.1 A I 0.2 = Q 4.0		<pre> graph LR IO3[IO.3] --- AND1[&] MO0[MO.0] --- AND1 AND1 --- OR1[>=1] IO0[IO.0] --- AND2[&] IO1[IO.1] --- AND2 IO2[IO.2] --- AND2 OR1 --- AND2 AND2 --- Q40[Q4.0] </pre>
A(A I 0.0 AN I 0.1 O A I 0.3 A M 0.0) A I 0.2 = Q 4.0		<pre> graph LR IO0[IO.0] --- AND1[&] IO1[IO.1] --- AND1 AND1 --- OR1[>=1] IO3[IO.3] --- AND2[&] MO0[MO.0] --- AND2 OR1 --- AND2 AND2 --- Q40[Q4.0] </pre>
A I 0.0 A(AN I 0.1 O A I 0.3 A M 0.0) A I 0.2 = Q 4.0		<pre> graph LR IO3[IO.3] --- AND1[&] MO0[MO.0] --- AND1 AND1 --- OR1[>=1] IO1[IO.1] --- AND2[&] IO0[IO.0] --- AND2 IO2[IO.2] --- AND2 OR1 --- AND2 AND2 --- Q40[Q4.0] </pre>
A I 0.0 A(AN I 0.1 A I 0.2 O A I 0.3 A M 0.0) = Q 4.0		<pre> graph LR IO1[IO.1] --- AND1[&] IO2[IO.2] --- AND1 AND1 --- OR1[>=1] IO3[IO.3] --- AND2[&] MO0[MO.0] --- AND2 OR1 --- AND2 AND2 --- Q40[Q4.0] </pre>

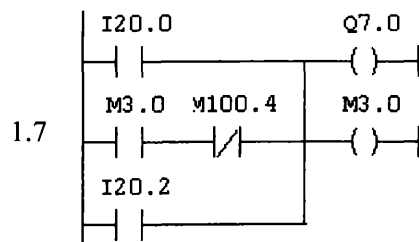
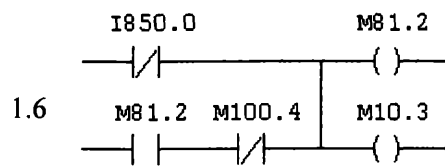
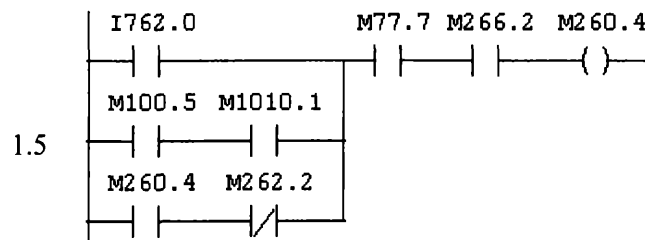
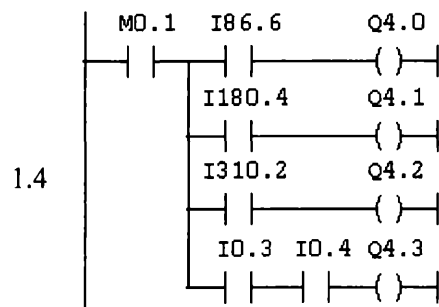
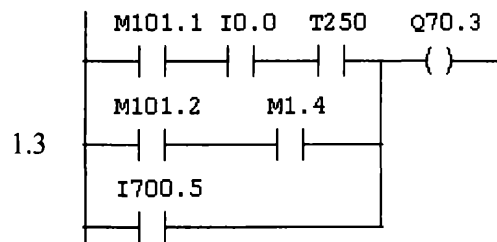
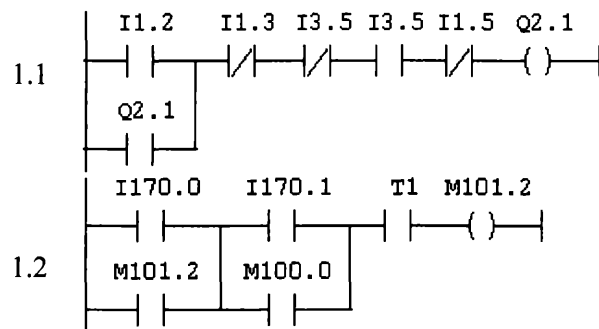
A I 0.0 A(A I 0.1 AN I 0.1 O M 0.1) A I 0.2 O I 0.3 A I 0.3 A M 0.0) = Q 4.0		
A I 0.0 A(AN I 0.1 A I 0.2 O A(O I 0.3 ON M 0.2) A M 0.0) = Q 4.0		
A I 0.0 A(AN I 0.1 A I 0.2 O A I 0.3 A M 0.0 ON M 0.2) = Q 4.0		
A I 0.0 A(AN I 0.1 A I 0.2 O A I 0.3 A M 0.0 O I 0.6) ON M 0.2) = Q 4.0		

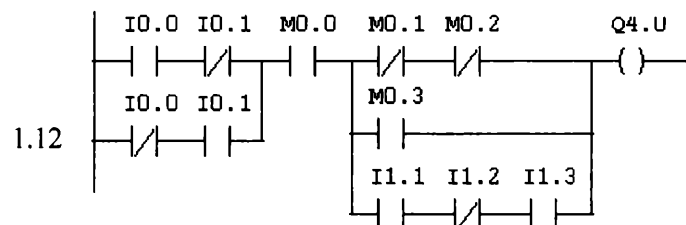
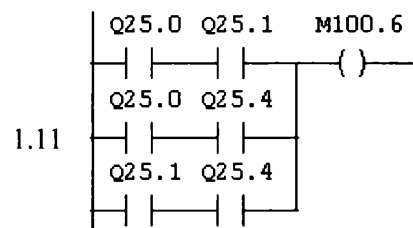
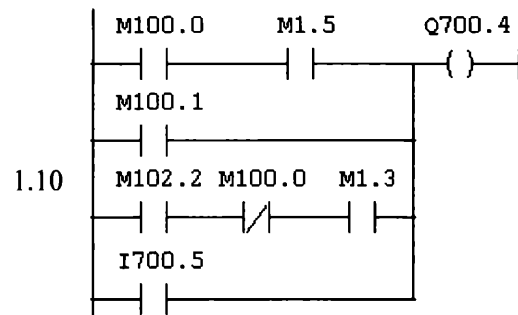
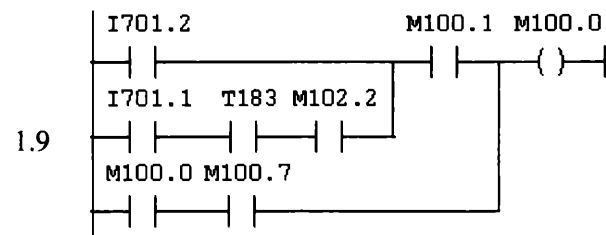
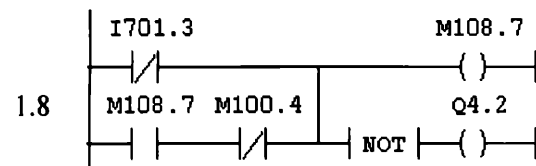
A I 0.0 A(AN I 0.1 A I 0.2 O A I 0.3 A(A M 0.0 O I 0.6) O AN M 0.2 A I 0.7) = Q 4.0		
AN I 0.1 A I 0.2 O A I 0.3 A(O M 0.0 O I 0.6) O AN M 0.2 A I 0.7 = Q 4.0 = Q 0.0 = Q 0.5 = M 2.4 = M 2.5		

Таблица 3.2.8.1: Примеры

3.2.9 Упражнения

1) Напишите код следующих логических действий на языке STL





2. Выполните следующие действия на языке STL:

2.1	В задаче 1.1 поставьте «перемычку» M111.1 параллельно I3.5 (в обоих случаях)
2.2	В задаче 1.2 поставьте «перемычку» M111.1 параллельно I701.1
2.3	В задаче 1.3 поставьте «перемычку» M111.1 параллельно M1.4
2.4	В задаче 1.4 поставьте «перемычку» M111.1 параллельно I180.4
2.5	В задаче 1.5 поставьте «перемычку» M111.1 параллельно M100.5
2.6	В задаче 1.6 поставьте «перемычку» M111.1 параллельно M81.2
2.7	В задаче 1.7 поставьте «перемычку» M111.1 параллельно I20.2
2.8	В задаче 1.8 поставьте «перемычку» M111.1 параллельно M100.4
2.9	В задаче 1.9 поставьте «перемычку» M111.1 параллельно M100.7
2.10	В задаче 1.10 поставьте «перемычку» M111.1 параллельно M100.0 (в двух местах)
2.11	В задаче 1.11 поставьте «перемычку» M111.1 параллельно Q25.4 (во всех местах)
2.12	В задаче 1.12 поставьте «перемычку» M777.7 параллельно I1.2

Таблица 3.2.9.1: Задачи

4) На языке STL напишите код следующих выражений

4.1		4.2	
4.3		4.4	
4.5		4.6	
4.7		4.8	

Таблица 3.2.9.2: Упражнения

3.2.10 Решение

1.1	A(O I 1.2 O Q 2.1) AN I 1.3 AN I 3.5 A I 3.5 AN I 1.5 = Q 2.1	1.2	A(A I 170.0 O M 101.2) A(A I 170.1 O M 100.0) A T 1 = M 101.2	1.3	A M 101.1 A I 0.0 A T 250 O A M 101.2 A M 1.4 O I 700.5 = Q 70.3
2.1	A(O I 1.2 O Q 2.1) AN I 1.3 A(ON I 3.5 O M 111.1) A(A I 3.5 O M 111.1) AN I 1.5 = Q 2.1	2.2	A(O I 170.0 O M 101.2) A(O I 170.1 O M 111.1 O M 100.0) A T 1 = M 101.2	2.3	A M 101.1 A I 0.0 A T 250 O A M 101.2 A(A M 1.4 O M 111.1) O I 700.5 = Q 70.3
1.4	A M 0.1 = #tem1 A #tem1 A I 86.6 = Q 4.0 A #tem1 A I 180.4 = Q 4.1 A #tem1 A I 310.2 = Q 4.2 A #tem1 A I 0.3 A I 0.4 = Q 4.3	1.5	A(O I 762.0 O A M 100.5 A M 1010.1 O A M 260.4 AN M 262.2) A M 77.7 A M 266.2 = M 260.4	1.6	ON I 850.0 O A M 81.2 AN M 100.4 = M 81.2 = M 10.3

2.4	A M 0.1 = #tem1 A #tem1 A I 86.6 = Q 4.0 A #tem1 A(A I 180.4 O M 111.1) = Q 4.1 A #tem1 A I 310.2 = Q 4.2 A #tem1 A I 0.3 A I 0.4 = Q 4.3	2.5	A(O I 762.0 O A(A M 100.5 O M 111.1) A M 1010.1 O A M 260.4 AN M 262.2) A M 77.7 A M 266.2 = M 260.4	2.6	ON I 850.0 O A(A M 81.2 O M 111.1) AN M 100.4 = M 81.2 = M 10.3
1.7	O I 20.0 O A M 3.0 AN M 100.4 O I 20.2 = Q 7.0 = M 3.0	1.8	A M 108.7 AN M 100.4 ON I 701.3 = M 108.7 NOT = Q 4.2	1.9	A(O I 701.2 O A I 701.1 A T 183 A M 102.2) A M 100.1 O A M 100.0 A M 100.7 = M 100.0
2.7	O I 20.0 O A M 3.0 AN M 100.4 A(O I 20.2 O M 111.1) = Q 7.0 = M 3.0	2.8	A M 108.7 A(AN M 100.4 O M 111.1) ON I 701.3 = M 108.7 NOT = Q 4.2	2.9	A(O I 701.2 O A I 701.1 A T 183 A M 102.2) A M 100.1 O A M 100.0 A(A M 100.7 O M 111.1) = M 100.0

1.10	A M 100.0 A M 1.5 O M 100.1 O A M 102.2 AN M 100.0 A M 1.3 O I 700.5 = Q 700.4	1.11	A Q 25.0 A Q 25.1 O A Q 25.0 A Q 25.4 O A Q 25.1 A Q 25.4 = M 100.6	1.12	A(X I 0.0 X I 0.1) A M 0.0 A(AN M 0.1 AN M 0.2 O M 0.3 O A I 1.1 AN I 1.2 A I 1.3) = Q 4.0
2.10	A(A M 100.0 O M 111.1) A M 1.5 O M 100.1 O A M 102.2 A(AN M 100.0 O M 111.1) A M 1.3 O I 700.5 = Q 700.4	2.11	A Q 25.0 A Q 25.1 O A Q 25.0 A(A Q 25.4 O M 111.1) O A Q 25.1 A(A Q 25.4 O M 111.1) = M 100.6	2.12	A(X I 0.0 X I 0.1) A M 0.0 A(AN M 0.1 AN M 0.2 O M 0.3 O A I 1.1 A(AN I 1.2 O M 777.7) A I 1.3) = Q 4.0

Таблица 3.2.10.1: Решения

4. АККУМУЛЯТОРЫ

Аккумулятор – специальный регистр в процессоре, выполняющий функции промежуточного буфера. При одновременной обработке двух численных значений требуется два промежуточных буфера. Данными буферами являются аккумулятор1 – ассу1 и аккумулятор2 – ассу2. Все CPU серии S7-300 (кроме S7-318 имеют 2 аккумулятора), CPU S7-400 и имеют 4 аккумулятора.

Разрядность каждого аккумулятора – 32бита. Обмен информацией может происходить побайтно, по 1 машинному слову и по 1 двойному машинному слову.

Загружаемые (Loading) данные – данные, направляемые из какой-либо области памяти в аккумулятор посредством команды L

Выгружаемые (Transferring) данные – данные, направляемые из аккумулятора в какую-либо область памяти посредством команды T.

Также, существуют отдельные инструкции для работы с аккумуляторами

4.1 Инструкции для работы с аккумуляторами

Приведенные ниже инструкции предназначены для работы с аккумуляторами ЦПУ. Данные инструкции не зависят от битов слова состояния и не меняют их.

Название	Описание
ТАК	обмен содержимым ассу1 и ассу2. На содержимое ассу3 и ассу4 не влияет.
POP	копирует содержимое ассу2 в ассу1. Содержимое ассу(2,3 и 4) не меняется. Для CPU с 4-мя аккумуляторами: копирует содержимое ассу2 в ассу1, ассу3 в ассу2, ассу4 в ассу3. Содержимое ассу4 не меняется.
PUSH	Копирует содержимое ассу1 в ассу2. Содержимое ассу1 не меняется. Для CPU с 4-мя аккумуляторами: Копирует содержимое ассу3 в ассу4, содержимое ассу2 в ассу3, и содержимое ассу1 в ассу2. Содержимое ассу1 не меняется.
BLD<№>	нулевая операция, действий не выполняет, используется в графических языках LAD и FBD. № присваивается программатором автоматически
NOP 0	Нулевая инструкция, никаких действий не выполняет, на биты слова состояния не влияет. Используется программатором в графических языках
NOP 1	Нулевая инструкция, никаких действий не выполняет, на биты слова состояния не влияет. Используется программатором в графических языках

Таблица 4.1.1: Инструкции для работы с аккумуляторами

4.2 Работа с крупными единицами данных

Крупные единицы памяти – единицы памяти, объем которых больше или равен 1 байту.

Работа с ними осуществляется с помощью команд L (load) и T (Transfer)

В общем виде, работа с крупными единицами памяти выглядит следующим образом:

<область памяти> <формат обращения> <адрес>
<I/Q/M> <B/W/D>.

Пример:

L MB10 //записать в аккумулятор меркерный байт 10

T QB 0 //отправить его на выходной байт 0

Пример:

L ID 0 // записать в аккумулятор входное двойное слово 0

T QD 0 //отправить его на выходное двойное слово

4.3 Инструкции сравнения

В среде программирования Step7 существует возможность сравнивать содержимое аккумуляторов. Для сравнения содержимого аккумуляторов используется следующий алгоритм:

- Загрузить данные в асс1 и асс2
- Указать знак сравнения и тип данных.

Для загрузки необходимо использовать функцию L (Load)

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.

Используются следующие типы данных:

I: Сравнение чисел типа Integer (16-bit), целых Пример: L 5 Пример: L-100.	сравнение содержимого младших слов аккумуляторов (ACCU2-L и ACCU 1-L)
D: Сравнение чисел типа Double Integer (32-bit), двойных целых Пример: L#5 Пример: L#-100	сравнение содержимого аккумуляторов (ACCU2 и ACCU1)
R: Сравнение чисел с плавающей точкой Пример: L 5.0 Пример: L 1.5	сравнение содержимого аккумуляторов (ACCU2 и ACCU1)

Таблица 4.3.1: Инструкции сравнения

При положительном результате сравнения RLO устанавливается в «1»

Инструкция сравнения	значение RLO при ACCU2>ACCU1	значение RLO при ACCU2=ACCU1	значение RLO при ACCU2<ACCU1
= (I/D/R) (ACCU2=ACCU1)	0	1	0
<> (I/D/R) (ACCU2≠ACCU1)	1	0	1
> (I/D/R) (ACCU2>ACCU1)	1	0	0
< (I/D/R) (ACCU2<ACCU1)	0	0	1
>= (I/D/R) (ACCU2≥ACCU1)	1	1	0
<= (I/D/R) (ACCU2≤ACCU1)	0	1	1

Таблица 4.3.2: Влияние результата сравнения на RLO

4.3.1 Примеры:

Загрузить в аккумуляторы два разных числа и сравнить их, используя несколько инструкций сравнения. При удачном сравнении – зажечь лампу Q4.0

№	STL	Комментарий
1	L 5	//загрузка числа 5 в формате Integer в ассu1
	L 2	// загрузка числа 2 в формате Int в ассu1, перемещение числа (5) в ассu2
	=I	//сравнение. (ассu1= ассu2)? то есть (2=5)?
	= Q 4.0	//лампа не зажжется, т.к. 2≠5
2	L 7	// загрузка числа 7 в формате Integer в ассu1
	L 3	// загрузка числа 3 в формате Int в ассu1, перемещение числа (7) в ассu2
	<>I	//сравнение. (ассu2 ≠ ассu1)? то есть (7≠ 3)?
	= Q 4.0	//лампа зажжется, т.к. 3≠7
3	L 8	// загрузка числа 8 в формате Int в ассu1
	L 5	// загрузка числа 5 в формате Int в ассu1, перемещение числа (8) в ассu2
	>I	//сравнение (ассu2 > ACCU1)? то есть (8>5)?
	= Q 4.0	//лампа зажжется, т.к. 8>5
4	L 5	// загрузка числа 5 в формате Int в ассu1
	L 8	// загрузка числа 8 в формате Int в ассu1, перемещение числа (5) в ассu2
	<=I	//сравнение. (ассu2 ≤ ассu1)? то есть (5≤ 8)?
	= Q 4.0	//лампа зажжется, т.к. 5≤ 8
5	L 500 000	// загрузка числа 500 000 в формате Dint в ассu1
	L 200 000	// загрузка числа 200 000 в формате Dint в ассu1, перемещение числа (500 000) в ассu2
	=D	//сравнение. (ассu1= ассu2)? то есть (200 000=500 000)?
	= Q 4.0	//лампа не зажжется, т.к. 200000≠500000
6	L 5.3	// загрузка числа 5.3 в формате Real в ассu1
	L 8.0	// загрузка числа 8.0 в формате Real в ассu1, а перемещение числа (5.3) в ассu2
	<=R	//сравнение (ассu2 ≤ ассu1)? то есть (5.3≤ 8.0)?
	= Q 4.0	//лампа зажжется, т.к. 5.3≤ 8.0

Таблица 4.3.1.1: Примеры использования функций сравнения

4.4 Математические операции

В среде программирования Step7 существует возможность осуществлять математические действия между содержимым аккумуляторов. Результат арифметического действия сохраняется в ассu1. После выполнения математических операций целесообразно проверять следующие биты слова состояния: CC0, CC1, OV, OS, так как по результатам их проверки можно судить о корректности результата.

Бит OS в случае ошибки установится в «1» и будет «1», пока не исполнится цикл OB, даже если ошибка ушла.

Бит OV – в случае ошибки установится в «1», при исчезновении ошибки - скинется в «0».

Биты CC0 и CC1 – флаги условия. Если операция не может быть использована, они устанавливаются в «1».

После выполнения математических операций на CPU S7-318 и CPU-400 содержимое ассu2 переписывается содержимым ассu3. А содержимое ассu4 заменяет прежнее значение ассu3. Старое содержимое ассu4 не меняется

Доступны следующие арифметические операции:

Арифметическая функция	Тип данных		
	Int	Dint	Real
Сложение	+I	+D	+R
Вычитание	-I	-D	-R
Умножение	*I	*D	*R
Деление	/I	/D	/R
Остаток от деления	-	Mod	-
Комментарий	Операции выполняются только над содержимым младших слов аккумуляторов	Mod – делит содержимое ассu2 на содержимое ассu1. Остаток от деления сохраняет в ассu1	

Таблица 4.4.1: Математические операции

При выполнении математических операций используется следующий алгоритм:

- Загрузить данные в ассu1 и ассu2
- Указать математический знак и тип данных
- Указать место выгрузки (передать данные в результирующую переменную)

Совершать математические действия можно только над числами с одинаковым типом данных.

4.5 Инкрементирование и декрементирование

При необходимости увеличить или уменьшить содержимое `assu1`, при условии нахождения в нем целого числа, на некоторое число, ожидая результат в диапазоне [0-255] можно воспользоваться функциями инкрементирования и декрементирования с помощью следующего алгоритма:

- Загрузить данные в аккумулятор
- Выполнить операцию декрементирования или инкрементирования (`DEC n` или `INC n`, где `n` – число, с которым производится математическое действие)
- Указать место выгрузки (передать данные в результирующую переменную)

Пример:

Обычная запись	Упрощенная запись	Инкрементирование
L 5 L 2 +I T MW6	L 5 + 2 T MW6	L 5 Inc 2 Если сумма >255, то скидывается в «0» и продолжает считать в большую сторону. Автоматически сохраняет результат в <code>assu1</code> . Работает только в пределах 1-го байта.
L 5 L 2 -I T MW6	L 5 - 2 T MW6	L 5 Dec 2 Если выйдет за «0», скинется в 255 и начнет считать в меньшую сторону. Сохраняет результат в <code>assu1</code> . Работает только в пределах 1-го байта.

Таблица 4.5.1: Пример инкрементирования и декрементирования

4.6 Типичные ошибки

При программировании математических инструкций и операций сравнения следует избегать следующих ошибок: переполнение, неверное указание типов данных операции, неверное указание формата ячеек памяти, операции над числами в разных форматах.

Пример:

№	STL	Комментарий
1	L 100000	//загрузка числа 100000 в ассу1
	L 5	//Загрузка 5 в ассу1, перемещение числа 100000 в ассу2
	+ I	// Сложение целых чисел. <u>ОШИБКА</u> – переполнение
	T MW 10	// максимального диапазона целых чисел
2	L 10.0	//загрузка числа 10.0 (Real) в ассу1
	L 5.7	//загрузка числа 5.7 (Real) в ассу1, перемещение 10.0 в ассу2
	+ D	//Сложение двойных целых чисел. <u>ОШИБКА</u> – неверно
	T MW 10	// указан тип данных. Для чисел Real нужно указывать букву R
3	L 10.0	//загрузка числа 10.0 (Real) в ассу1
	L 5	//загрузка числа 5 (Int/Dint) в ассу1, перемещение 10.0 в ассу2
	+ I	//Сложение целых чисел. <u>ОШИБКА</u> – числа находятся в разных
	T MW 10	//форматах данных
4	L 30000	//загрузка числа 30000 в ассу1
	L 50000	//Загрузка 50000 в ассу1, перемещение числа 30000 в ассу2
	+ D	// Сложение двойных целых чисел.
	T MW 10	// <u>ОШИБКА</u> – для формата Dint нужно двойное слово (32 байта)

Таблица 4.6.1: Типичные ошибки при выполнении математических операций

4.6.1 Примеры:

№	STL код	Комментарий
1	L 5	//загрузка числа 5 в младшее слово ассу1
	L 10	//Загрузка 10 в ассу1, перемещение числа 5 в ассу2
	+I	//сложение целых чисел
	T MW 10	//передача результата (15) в MW10
2	L 5	//загрузка числа 5 в ассу1
	L 10	//Загрузка 10 в ассу1, перемещение числа 5 в ассу2
	+D	//сложение двойных целых чисел
	T MD 10	//передача результата (15) в MD10
3	L 5.7	//загрузка числа 5.7 в ассу1
	L 10.0	//Загрузка числа 10 в ассу1, перемещение числа 5.7 в ассу2
	+R	//сложение чисел с плавающей точкой
	T MD 10	//передача результата (15.7) в MD10
4	L 10	//загрузка числа 10 в младшее слово ассу1
	L 7	// Загрузка числа 7 в ассу1, перемещение числа 10 в ассу2
	-I	//вычитание целых чисел
	T MW 10	//передача результата (3) в MW10
5	L 7	//загрузка числа 7 в младшее слово ассу1
	L 8	// загрузка «8» в ассу1, перемещение «7» в ассу2
	-I	//вычитание целых чисел
	T MW 10	//передача результата (65535) в MW10
	L 1	//загрузка единицы
	*I	//умножение целых чисел
6	T MW 20	//передача результата (-1) в MW 20
	L 7	//загрузка числа 7 в младшее слово ассу1
	L 9	// загрузка «9» в ассу1, перемещение «7» в ассу2
	-I	//вычитание целых чисел
	T MW 10	//передача результата (65534) в MW10
	L 1	//загрузка единицы
	*I	//умножение целых чисел
	T MW 20	//передача результата (-2) в MW 20
При вычитании из меньшего числа большего в формате данных int после нуля отображается не «-1», а 65535. Для корректного отображения результатов рекомендуется полученные данные умножить на единицу.		
7	L 10	//загрузка числа 10 в ассу1
	L 7	// Загрузка числа 7 в ассу1, перемещение числа 10 в ассу2
	-D	//вычитание двойных целых чисел
	T MD 10	//передача результата (3) в MD10
8	L 7	//загрузка числа 7 в ассу1
	L 8	// Загрузка числа 8 в ассу1, перемещение числа 7 в ассу2
	-D	//вычитание двойных целых чисел
	T MD 10	//передача результата (-1) в MD10
9	L 7	//загрузка числа 7 в ассу1
	L 9	// Загрузка числа 9 в ассу1, перемещение числа 7 в ассу2
	-D	//вычитание двойных целых чисел
	T MD 10	//передача результата (-2) в MD10

10	L 10.0 L 7.5 -R T MD 10	//загрузка числа 10.0 в ассu1 // Загрузка числа 7.5 в ассu1, перемещение (числа 10) в ассu2 //вычитание чисел с плавающей точкой //передача результата (2.5) в MD10
11	L 7.15 L 8.27 -R T MD 10	//загрузка числа 7.15 в ассu1 // Загрузка числа 8.27 в ассu1, перемещение числа 7.15 в ассu2 //вычитание чисел с плавающей точкой //передача результата (-1.12) в MD10
12	L 7.0 L 9.0 -R T MD 10	//загрузка числа 7.0 в ассu1 // Загрузка числа 9.0 в ассu1, перемещение числа 7.0 в ассu2 // вычитание чисел с плавающей точкой //передача результата (-2.0) в MD10
13	L 2 L 8 *I T MW 10	//загрузка числа 2 в младшее слово ассu1 // Загрузка числа 8 в ассu1, перемещение числа 2 в ассu2 //умножение целых чисел //передача результата (16) в MW10
14	L -2 L 8 *I T MW 10	//загрузка числа -2 в младшее слово ассu1 // Загрузка числа 8 в ассu1, перемещение числа -2 в ассu2 //умножение целых чисел //передача результата (-16) в MW10
15	L -2 L 8 *I T MD 10	//загрузка числа -2 в младшее слово ассu1 // Загрузка числа 8 в ассu1, перемещение числа -2 в ассu2 //умножение целых чисел //передача результата (-16) в MD10
16	L 2 L 8 *D T MD 10	//загрузка числа 2 в ассu1 // Загрузка числа 8 в ассu1, перемещение числа 2 в ассu2 //умножение двойных целых чисел //передача результата (16) в MD10
17	L 2.0 L 8.6 *R T MD 10	//загрузка числа 2.0 в ассu1 // Загрузка числа 8.6 в ассu1, перемещение числа 2.0 в ассu2 //умножение чисел с плавающей точкой //передача результата (17.2) в MD10
18	L -2.0 L 8.6 *R T MD 10	//загрузка числа -2.0 в ассu1 // Загрузка числа 8.6 в ассu1, перемещение числа -2.0 в ассu2 //умножение чисел с плавающей точкой //передача результата (-17.2) в MD10
19	L 500 L 10 /I T MW 10	//загрузка числа 500 в младшее слово ассu1 // Загрузка числа 10 в ассu1, перемещение числа 500 в ассu2 //деление целых чисел //передача результата (50) в MW10
20	L 5 L 10 /I T MW 10	//загрузка числа 5 в младшее слово ассu1 // загрузка числа «10» в ассu1, перемещение числа «5» в ассu2 //деление целых чисел //передача результата (327680) в MW10

21	L 2	//загрузка числа 2 в младшее слово ассu1
	L 4	// загрузка числа «4» в ассu1, перемещение числа «2»в ассu2
	/I	
	T MW 10	//деление целых чисел //передача результата (131072) в MW10
При делении меньшего числа на большее в формате данных int результат математической операции отобразится некорректно, а его фактическое значение будет равен нулю. При делении меньшего числа на большее, рекомендуется изменить тип данных.		
22	L 500.0	//загрузка числа 500.0 в ассu1
	L 10.0	// Загрузка числа 10.0 в ассu1, перемещение числа 500.0 в ассu2
	/R	
	T MD 10	// деление чисел с плавающей точкой //передача результата (50.0) в MD10
23	L 5.0	//загрузка числа 5.0 в ассu1
	L 10.0	// загрузка «10.0» в ассu1, перемещение. «5.0» в ассu2
	/R	//деление чисел с плавающей точкой
	T MD 10	//передача результата (0.5) в MD10
24	L 5	//загрузка числа 5 в ассu1
	L 10	// загрузка числа «10» в ассu1, перемещение числа «5»в ассu2
	/D	
	T MD 10	//деление двойных целых чисел //передача результата (0) в MD10
25	L 2	//загрузка числа 2 в ассu1
	L 4	// загрузка «4» в ассu1, перемещение числа «2»в ассu2
	/D	//деление двойных целых чисел
	T MD 10	//передача результата (0) в MD10
При делении меньшего числа на большее в формате данных Dint результатом математической операции будет ноль. При делении меньшего числа на большее, рекомендуется изменить тип данных.		
26	L 15	//загрузка числа 15 в ассu1
	L 2	// Загрузка числа 2 в ассu1, перемещение числа 15 в ассu2
	MOD	//Остаток от деления (15/2)
	T MD 10	//передача результата (1) в MD10
27	L 17	//загрузка числа 17 в ассu1
	L 3	// Загрузка числа 3 в ассu1, перемещение числа 17 в ассu2
	MOD	//Остаток от деления (17/3)
	T MD 10	//передача результата (2) в MD10

Таблица 4.6.1.1: Примеры использования математических операций

4.7 Упрощенная форма записи

При выполнении сложения или вычитания константы с содержимым асс1 допускается непосредственное добавление константы к асс1, минуя асс2. Это производится с помощью следующего алгоритма:

- Загрузить данные в аккумулятор
- Загрузить константу (с указанием знака)
- Указать место выгрузки (передать данные в результирующую переменную)

Добавление константы к содержимому асс1 производится следующим способом:

- +B#16#bb – добавление байтовой константы «byte»
- ±W – добавление константы формата «word»
- +L#±d – добавление константы формата двойное слово «doubleword»

Пример:

№	STL код	Комментарий
1	L 10	//загрузка числа 10 в асс1, автоматически переводит «10» (dec) в «a» (hex)
	+ B#16#21	// добавление константы 33. Выполняется сложение «a+21»=«2b» (hex)
	T MW 10	//передача результата «2b» (hex) или 43 (dec) в MW10
2	L 10	//загрузка числа 10 в асс1
	+ -7	// добавление константы «-7». Выполняется сложение «10+-7»
	T MW 10	//передача результата «3» в MW10
3	L 10	//загрузка числа 10 в асс1
	+ -11	// добавление константы «-11». Выполняется сложение «10+-11»
	T MW 10	
	L 1	//передача результата в MW10
	*I	//умножение на единицу
	T MW 20	//для корректного отображения
4	L 10	//загрузка числа 10 в асс1
	+ 11	// добавление константы 11». Выполняется сложение «10+11»
	T MW 10	//передача результата «21» в MW10
5	L 10	//загрузка числа 10 в асс1
	+ L#-11	// добавление константы "-11". Выполняется сложение "10+-11"
	T MD 10	//передача результата "-1" в MD10

Таблица 4.7.1: Примеры использования упрощенной формы записи

4.8 Математические функции

Среда программирования Step-7 также позволяет использовать следующие математические функции для 32-битовых чисел с плавающей точкой (real). Данные функции изменяют биты CC0, CC1, OS и OV слова состояния. Действия выполняется над ассu1. Содержимое асс2, ассu3 и ассu4 – не изменяется.

Исключение – ABS - Не зависит от слова состояния и не меняет его биты.

Название	ABS	SQR	SQRT
Описание	модуль числа.	Квадрат числа	квадратный корень
Пример	L -5.0 ABS T MD 10	L 4.84 SQR T MD 1	L 23.43 SQRT T MD 1
Комментарий	1//загрузка числа 5.0 в ассu1 2//вычисление модуля 3//выгрузка	1//загрузка 4.84 в ассu1 2// возведение в квадрат 3// выгрузка результата с округлением до сотых 23.43	1//загрузка 23.43 в ассu1 2// извлечение корня 3// выгрузка результата (4.84)
Название	EXP	LN	SIN
Описание	экспонента	натуральный логарифм	синус угла в радианной мере.
Пример	L 2.0 EXP T MD 10	L 7.389 LN T MD 10	L 1.570796 SIN T MD 10
Комментарий	1//загрузка числа 2.0 в ассu1 2//вычисление экспоненты 3//выгрузка 7.389	1//загрузка 7.389 в ассu1 2// вычисление логарифма 3// выгрузка (2.0)	1//загрузка ($\pi/2$) в ассu1 2// вычисление синуса 3// выгрузка результата (1.0)
Название	COS	TAN	ASIN
Описание	косинус угла в радианной мере.	Тангенс угла в радианной мере.	Арксинус угла в радианной мере.
Пример	L 3.1415 COS T MD 10	L 1. 570796 TAN T MD 10	L 0.5 ASIN T MD 10
Комментарий	1//загрузка π в ассu1 2// вычисление косинуса 3// выгрузка результата (-1.0)	1//загрузка ($\pi/2$) в ассu1 2// вычисление тангенса 3// выгрузка результата (3186000)	1//загрузка (0.5) в ассu1 2// вычисление арксинуса 3// выгрузка результата (0.5236)
Название	ACOS	ATAN	
Описание	арккосинус угла в радианной мере.	Арктангенс угла в радианной мере.	
Пример	L 1.0 ACOS T MD 10	L 1. 570796 ATAN T MD 10	
Комментарий	1//загрузка 1 в ассu1 2// вычисление арккосинуса 3// выгрузка результата (0)	1//загрузка ($\pi/2$) в ассu1 2// вычисление тангенса 3// выгрузка результата (1.004)	

Таблица 4.8.1: Математические функции

4.9 Функции преобразования

Функции преобразования конвертируют тип данных содержимого ассемблера из одного формата данных в другой.

В общем виде алгоритм преобразования типа данных следующий:

- Загрузка адреса
- Функция преобразования
- Передача результата

Название	Описание
BTI	Преобразование BCD (3-х значное) в Integer. Преобразует двоично-десятичное число формата BTI в Integer в ассемблере-L. Допустимый диапазон от -999 до +999.
ITB	преобразование Integer в BCD (3-х значное). Преобразует целое число формата Integer в двоично-десятичное формата BTI в ассемблере-L.
BDT	Преобразование BCD в Double Integer. Преобразует двоично-десятичное число формата BTI в Double Integer в ассемблере-L. Допустимый диапазон от -9 999 999 до +9 999 999.
ITD	преобразование Integer в Double Integer. Преобразует целое число (16 bit) в двойное целое число (32 bit). Результат сохраняется в ассемблере-L.
DTB	преобразование Double Integer в BCD. Преобразует двойное целое число в семизначное BCD число. Допустимый диапазон от -9 999 999 до +9 999 999. Результат сохраняется в ассемблере-L.
DTR	преобразование Double Integer в число с плавающей точкой. Результат сохраняется в ассемблере-L.
INVI	инверсия числа типа Integer (16 bit). Инвертирует ассемблере-L. То есть заменяет все «0» на «1», а «1» на «0». Результат сохраняется в ассемблере-L.
INVD	инверсия числа типа Double Integer (32 bit). Инвертирует содержимое ассемблере-L. То есть заменяет все «0» на «1», а «1» на «0». Результат сохраняется в ассемблере-L.
NEGI	инверсия числа типа Integer (16 bit). Инвертирует ассемблере-L. То есть заменяет все «0» на «1», а «1» на «0» с последующим прибавлением «1» к результату. Результат сохраняется в ассемблере-L.
NEGD	инверсия числа типа Double Integer (32 bit). Инвертирует содержимое ассемблере-L. То есть заменяет все «0» на «1», а «1» на «0» с последующим прибавлением «1» к результату. Результат сохраняется в ассемблере-L.
NEGR	инверсия числа с плавающей точкой типа Real (32 bit). Инвертирует содержимое ассемблере-L. То есть заменяет все «0» на «1», а «1» на «0». Результат сохраняется в ассемблере-L.
CAW	изменение местами байт в ассемблере-L. Результат сохраняется в ассемблере-L.
CAD	изменение местами байт в ассемблере-L. Результат сохраняется в ассемблере-L.
RND	округление до ближайшего целого числа с плавающей точкой типа Real (32 bit). Округление происходит не по математическим правилам. 2.5 → 2. 2.6 → 3. 2.50001 → 3.
TRUNC	выделение целой части числа с плавающей точкой типа Real (32 bit). Результат сохраняется в формате Dint
RND+	округление до ближайшего большего целого числа с плавающей точкой типа Real (32 bit).
RND-	округление до ближайшего меньшего целого числа с плавающей точкой типа Real (32 bit).

Таблица 4.9.1: Функции преобразования

4.9.1 Примеры

№	STL	Комментарий
1	L 400 ITD DTR T MD 20 L 6.5 /R T MD 40 RND- T MD 60	//загрузка числа 400 в младшее слово ассu1 //преобразование в двойное целое //преобразование в число с плавающей точкой //сохранение результата в MD20 //загрузка числа 6.5 //деление чисел с плавающей точкой //сохранение результата деления (61.54) в MD40 //округление до ближайшего меньшего //сохранение результата (61) в MD60 (111101 bin)
2	L 6.4999 RND T MD 20	//загрузка числа 6.4999 в ассu1 //округление до ближайшего целого //выгрузка результата (6) в MD20
3	L 6.4999 RND+ T MD 20	//загрузка числа 6.4999 в ассu1 //округление до большего целого //выгрузка результата (7) в MD20
4	L 6.4999 RND- T MD 20	//загрузка числа 6.4999 в ассu1 //округление до меньшего целого //выгрузка результата (6) в MD20
5	L 6.5 RND T MD 20	//загрузка числа 6.5 в ассu1 // округление до ближайшего целого //выгрузка результата (6) в MD20
6	L 6.5001 RND T MD 20	//загрузка числа 6.5001 в ассu1 //округление до ближайшего целого //выгрузка результата (7) в MD20
7	L 6.5001 RND- T MD 20	//загрузка числа 6.5001 в ассu1 //округление до меньшего целого //выгрузка результата (6) в MD20
8	L 6.5001 TRUNC T MD 20	//загрузка числа 6.5001 в ассu1 //выделение целой части //выгрузка результата (6) в MD20
9	L 2#11001100 INVI T MB 20 L 1 *I T MB 40	//загрузка числа (204) в ассu1 //побитовая инверсия в младшем слове ассu1 //выгрузка результата 2#110011 (51) //умножение на единицу //для корректного //отображения результата
Если число полей и единиц < 8 – выгрузку производить в байт. Если > 8 – в Word. При инверсии всего аккумулятора – в DWord. Для корректного отображения результата инверсии ассu1L, его необходимо умножить на 1. Для правильного отображения инвертирования аккумулятора – необходимо выполнить операцию маскирования.		
10	L 2#1111000011110000 INVI T MW 20 L 1 *I T MW 40	//загрузка числа (61680) в ассu1 //побитовая инверсия в младшем слове ассu1 //выгрузка результата 2#111100001111 (3855) //умножение на единицу //для корректного //отображения результата

Таблица 4.9.1.1: Примеры использования функций преобразования

4.10 Поразрядные логические инструкции со словами

В среде программирования Step-7 также доступны поразрядные логические инструкции со словами – попарное сопряжение бит машинных слов, в соответствии с булевой логикой.

В общем виде алгоритм выполнения поразрядных логических инструкций следующий:

- Загрузить данные в ассu1 и ассu2
- Указать операнд инструкции
- Указать место выгрузки (передать данные в результирующую переменную)

Существуют следующие поразрядные логические инструкции:

Операнд	Описание	Разрядность
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 бита
OD	Поразрядное «ИЛИ» над двойными словами	32 бита
XOD	Поразрядное «ИСКЛЮЧАЮЩЕЕ ИЛИ» над словами	32 бита

Таблица 4.10.1: Поразрядные логические операции

4.10.1 Примеры

№	STL	Комментарий
1	L 20 L 17 AW T MW 30	//загрузка константы 20 (10100) в младшее слово ассу1 //загрузка константы 17 (10001) в младшее слово ассу1, перемещение старого содержимого (10100) в младшее слово ассу2 //Поразрядное «И» между ассу1-L и ассу2-L //Выгрузка результата 16 (10000) в MW 30
2	L 20 L 17 OW T MW 30	//загрузка константы 20 (10100) в младшее слово ассу1 //загрузка константы 17 (10001) в младшее слово ассу1, перемещение старого содержимого (10100) в младшее слово ассу2 //Поразрядное «ИЛИ» между ассу1-L и ассу2-L //Выгрузка результата 21 (10101) в MW 30
3	L 20 L 17 XOW T MW 30	//загрузка константы 20 (10100) в младшее слово ассу1 //загрузка константы 17 (10001) в младшее слово ассу1, перемещение старого содержимого (10100) в младшее слово ассу2 //Поразрядное «Исключающее ИЛИ» между ассу1-L и ассу2-L //Выгрузка результата 5 (00101) в MW 30
4	L 203 L 172 AD T MD 30	//загрузка константы 203 (11001011) в ассу1 //загрузка константы 172 (10101100) в ассу1, перемещение старого содержимого (10001000) в ассу2 //Поразрядное «И» между ассу1 и ассу2 //Выгрузка результата 136 (10001000) в MD 30
5	L 203 L 172 OD T MD 30	//загрузка константы 203 (11001011) в ассу1 //загрузка константы 172 (10101100) в ассу1, перемещение старого содержимого (10001000) в ассу2 //Поразрядное «ИЛИ» между ассу1 и ассу2 //Выгрузка результата 239 (11101111) в MD 30
6	L 203 L 172 XOD T MD 30	//загрузка константы 203 (11001011) в ассу1 //загрузка константы 172 (10101100) в ассу1, перемещение старого содержимого (10001000) в ассу2 //Поразрядное «Исключающее ИЛИ» между ассу1 и ассу2 //Выгрузка результата 103 (01100111) в MD 30

Таблица 4.10.1.1: Примеры поразрядных логических операций

4.11 Выполнение операции маскирования

Иногда, необходимо выделить содержимое каких-либо конкретных бит из байта, слова или двойного слова. Для этого применяется операция маскирования, а именно побитовая операция «2И» между крупной единицей памяти, отдельные биты которой необходимо извлечь и маской.

Маска – константа, которую необходимо явно указать в двоичном (можно шестнадцатеричном) формате для выполнения последующей операции «2И». При этом в маске на позициях бит, которые необходимо выделить из константы должны стоять единицы, а на остальных позициях – нули.

Пример:

Существует некоторый байт, от содержимого первых трех бит которого необходимо избавиться. То есть необходимо выделить последние 5 бит данного байта.

Пусть, содержимое байта будет следующим: 1 0 0 1 1 0 1 0

Для того, чтобы выделить 5 последних бит из данного бита необходимо явно указать маску 0 0 0 1 1 1 1 1 и выполнить поразрядную операцию «2И»

Номер бита	7	6	5	4	3	2	1	0	STL
IB байт	1	0	0	1	1	0	1	0	L 2#10011010
Маска	0	0	0	1	1	1	1	1	L2#00011111 AW
Результат	0	0	0	1	1	0	1	0	T MB20

Таблица 4.11.1: Пример выполнения операции маскирования 1

Пример:

Существует машинное слово, содержимое пятого бита которого необходимо выделить.

Машинное слово приведено в таблице

Номер бита	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Слово	1	0	0	1	1	0	1	0	1	0	1	1	1	0	1	0

Таблица 4.11.2: Пояснение к примеру выполнения операции маскирования

Для выделения пятого бита необходимо явно указать маску – машинное слово, в котором содержимое пятого бита «1», а содержимое остальных бит «0» и произвести поразрядную операцию «2И»

Номер бита	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Слово	1	0	0	1	1	0	1	0	1	0	1	1	1	0	1	0
Маска	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
Результат	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Таблица 4.11.3: Пример выполнения операции маскирования 2

При выделении бит из двойного слова необходимо пользоваться функцией AD по аналогии.

5 СЛОВО СОСТОЯНИЯ (STATUS WORD)

Слово состояния – отдельный регистр CPU, представляющий собой набор битов состояния, располагаемый в энергозависимой (RAM) памяти ЦПУ.

В каждом новом цикле блока OB1 все биты слова состояния скидываются в «0».

Номер бита	9	8	7	6	5	4	3	2	1	0
Остальные биты автоматически используются для доступа к слову состояния, алгоритм доступа скрыт от программиста		BR	CC1	CC0	OV	OS	OR	STA	RLO	FC

Таблица 5.1: Расположение бит в слове состояния

Первичный опрос (First Check) – бит FC

Бит FC – флаг первичного опроса. До опроса находится в состоянии «0». После исполнения опроса =«1». Бит FC =«1» если выполняется другая логическая операция, кроме присвоения, скобки, S, R, команд перехода, указаний таймеров либо счетчиков.

Состояние бита FC управляет последовательностью битовых команд. Двоичное логическое действие всегда начинается с процедуры первичного опроса.

При успешном битовом первичном опросе переменной на состояние логической единицы (A/O/X), т.е. при нахождении опрашиваемой переменной в состоянии логической единицы – бит FC устанавливается в «1» и дальнейшие логические действия цепочки выполняются. При нахождении опрашиваемой переменной в состоянии «0» этого не происходит.

При успешном битовом первичном опросе переменной на состояние логического нуля (AN/ON/XN), т.е. при нахождении опрашиваемой переменной в состоянии логического нуля – бит FC устанавливается в «1» и дальнейшие логические действия цепочки выполняются.

При нахождении опрашиваемой переменной в состоянии «1» этого не происходит.

Значение бита FC сбрасывается в «0» после завершения выполнения цепочки битовых логических операций, а также, после выполнения команд присваивания (=, R,S), а также после команд переходов, анализирующих биты RLO или BR.

Результат логической операции (Result of logic operation) – бит RLO

В данный бит записывается итог первичного опроса и результат любой битовой логической операции.

Если логическая операция выполнена – в бит RLO записывается «1», а если не выполнена – записывается «0».

Бит статуса (Status bit) – бит STA

Используется сам, автоматически. (Булевы значения указанной битовой ячейки памяти до выполнения операции над ней). Т.е. физически, операции выполняются над битом STA.

Значение бита STA используется для отладки программ.

Бит состояния OR – флаг операции «ИЛИ»

Используется автоматически при выполнении операция «И» до операции «ИЛИ».

Устанавливается в «1», если после выполнения команды «И», перед выполнением команды «ИЛИ» в RLO находится «1». Сбрасывается в «0» любой другой командой для работы с битами.

Бит переполнения OV (Overflow)

В случае ошибки при выполнении математических операций или команд сравнения с числами формата real (переполнение/выход за границы диапазона/недопустимая операция) устанавливается в «1». При исчезновении ошибки сбрасывается в «0».

Бит переполнения OS (Overflow stored)

Сохраняет значение бита OV, но в отличие от него НЕ сбрасывается в «0» при исчезновении ошибки. То есть, отображает, были ли ошибки при выполнении предыдущих команд. Сбрасывается в «0» командой JOS (переход при OS=«1»), командами вызова блоков и на новом цикле блока, если ошибка уже не актуальна.

Флаги условия CC0 и CC1 (condition codes)

Используются для анализа результатов выполнения математических команд, команд сравнения, команд сдвига и логических команд над словами. (см таблицу «Состояние флагов условия после выполнения команд»)

Бит двоичного результата BR (binary result)

Устанавливается в «1» при успешном выполнении функций, системных функций, функциональных блоков. Является аналогом поля return языков высокого уровня.

5.1 Состояние флагов условия после выполнения команд

Тип команды	Результат выполнения команды	Состояние битов CC0 и CC1	
Команды сравнения	Содержимое аккумуляторов равно	0	0
	Содержимое ассу2 < ассу1	0	1
	Содержимое ассу2 > ассу1	1	0
	Операция сравнения не может быть выполнена (только для типа данных real)	1	1
Математические команды	Результат = 0	0	0
	Результат < 0	0	1
	Результат > 0	1	0
Команды целочисленной арифметики (с переполнением)	Переполнение отриц. Диапазона (для команд +I и +D)	0	0
	Переполнение отрицательного диапазона для команд *I и *D. Переполнение положительного диапазона для команд +I, -I, +D, -D, NEGI, NEGD	0	1
	Переполнение положительного диапазона для команд *I, *D, /I, /D Переполнение отрицательного диапазона для команд +I, -I, +D, -D	1	0
	Деление на ноль для команд /I, /D, Mod	1	1
	Очень малый (по модулю) результат	0	0
Команды вещественной арифметики (с переполнением)	Переполнение отрицательного диапазона	0	1
	Переполнение положительного диапазона	1	0
	Некорректное вещественное число	1	1
	Последний выдвинутый бит = «0»	0	0
Команды сдвига	Последний выдвинутый бит = «1»	1	0
	Результат равен нулю	0	0
Логические команды над словами	Результат не равен нулю	1	0

Таблица 5.1.1: Состояние флагов условия после выполнения команд

5.2 Анализ битов слова состояния через команды переходов

Для анализа битов слова состояния используются команды условных переходов, то есть переходов из одной точки программы в другую, при выполнении определенных условий.

Следует помнить правила использования команд перехода:

- После выполнения перехода бит RLO=«1»
- После любой команды перехода нужно указывать метку, на которую будет осуществляться переход
- Метка указывается латинскими символами, длина метки не может превышать 4 символа
- Имя метки не должно повторяться внутри блока и начинаться с цифры
- Переходы осуществляются как вперед, так и назад, но в пределах одного блока, максимальная длина перехода -32768 или +32768 слов программного кода
- Имя метки располагать на пустой строке не допускается. Для решения данной проблемы используются нулевые инструкции NOP 0 или NOP1
- Если при переходе на метку необходимо выйти из блока – используются команды: **ВЕС** – выход из блока при RLO=«1» и **ВЕС** – безусловный выход из блока

Существуют следующие команды переходов:

Команда	Условие выполнения	Комментарий
JU	-----	безусловный переход
JC	RLO=«1»	условный переход
JCN	RLO=«0»	
JN	результат≠0	(CC 1=1 и CC 0=0) или (CC 1=0 и CC 0=1)
JMZ	результат≤0	(CC 1=0 и CC 0=1) или (CC 1=0 и CC 0=0)
JPZ	результат≥0	(CC 1=1 и CC 0=0) или (CC 1=0 и CC 0=0)
JCB	RLO=«1»	RLO сохранится в бите BR
JNB	RLO=«0»	RLO сохранится в бите BR
JB	BR=«1»	
JNB	BR=«0»	
JO	OV=«1»	
JOS	OS=«1»	
JZ	результат =0	(CC 1=0 и CC 0=0)
JP	результат >0	(CC 1=1 и CC 0=0)
JM	результат <0	(CC 1=0 и CC 0=1)
JUO	недействительный результат	переход при делении на 0, недопустимой инструкции, неопределенном результате

Таблица 5.2.1: Команды переходов

Пример:

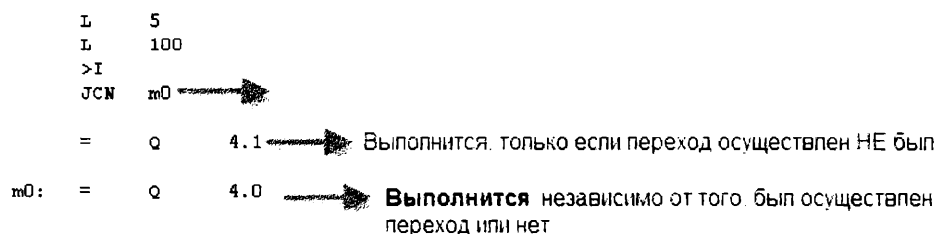


Рисунок 5.2.1: Пример условного перехода

5.3 Работа со словом состояния

В среде программирования STL предусмотрена возможность работать как со всем словом состояния, так и с отдельными его битами.

Биты RLO, OS, OV, BR, CC0, CC1 – открыты, то есть их можно как угодно изменять и осуществлять с ними логические операции. Также предусмотрена возможность явно скинуть биты OV и OS в «0» не дожидаясь окончания блока.

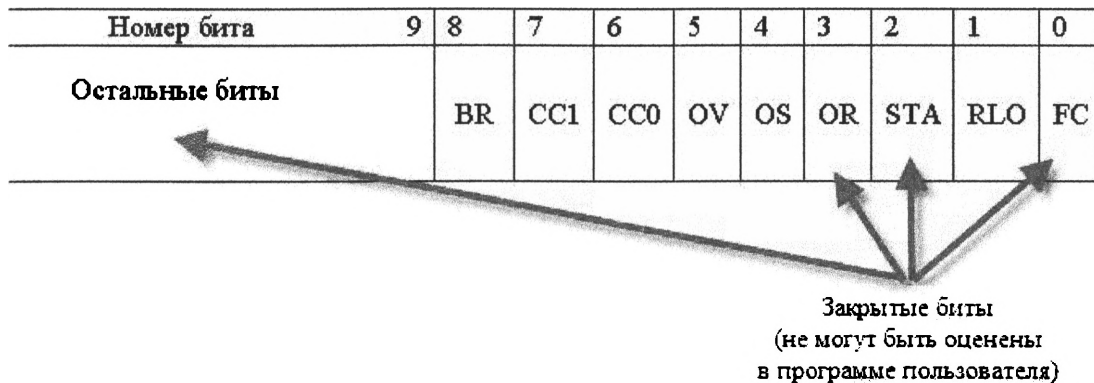


Рисунок 5.3.1: Пояснение возможности работы с битами слова состояния

Предусмотрены следующие команды для работы со словом состояния и его битами:

- **L STW** – загрузить слово состояния в accu1.
- **SAVE** – сохранить бит RLO в бите BR.
- **A/O/X (OS/OV/BR)** – сохранение значения соответствующего бита в бите RLO. (опрос бита слова состояния)
- **A/O/X (= =0, >0, <0, <>0, >=0, <=0)** – при истинном выполнении условия RLO устанавливается в «1»
- **T STW** – загрузка данных в STW. НЕЛЬЗЯ изменять биты FC, OR, STA и биты с 15-го по 9-й.

5.3.1 Примеры

№	STL	Комментарий
1	L 7.9	//загрузка числа 7.9 в ассu1
	L 3.4	//Загрузка 3.4 в ассu1, перемещение числа 7.9 в ассu2
	-R	// вычитание чисел с плавающей точкой
	A >=0	// Если результат >= 0, RLO установится в «1»
	JC m1	//переход на метку m1 при RLO=1
	L 2#11111111	//Зажечь все лампы
	T QB 4	//четвертого выходного байта
	m1: = Q 4.0	//зажечь лампу Q4.0
2	L 100	//загрузка числа 100 в ассu1L.
	L 0	//загрузка числа 0 в ассu1L, перемещение 100 в ассu2L
	/I	//Деление целого числа на 0. CC0=«1» и CC1=«1»
	JUO m1	// переход при недействительном результате
	L 2#11111111	//Зажечь все лампы
	T QB 4	//четвертого выходного байта
	m1: L 2#0	// скинуть, если работало то, что выше
	T QB 4	//
	SET	//RLO установить в «1»
	= Q 4.0	Выгрузка значения RLO в Q4.0
Умножить 2 произвольных числа формата Int, в случае превышения максимального диапазона – зажечь лампу Q4.0		
3	L 1000	//загрузка числа 100 в ассu1L
	L 5	// загрузка числа 5 в ассu1L, перемещение 100 в ассu2L
	*I	//умножение целых чисел.
	A OV	//опрос бита переполнения
	= Q 4.0	//выход
Умножить 2 произвольных числа в формате Int. При результате умножения больше 0 – зажечь лампу Q4.0, при результате умножения меньше либо равно 0 – зажечь лампу Q4.1, при переполнении отрицательного либо положительного диапазона – зажечь лампу Q4.2. При переполнении лампы Q4.0 и Q4.1 не зажигать		
4	L 1000	//загрузка первого числа в ассu1L
	L 5	//загрузка второго числа в ассu1L, первое в ассu2L
	*I	//умножение целых чисел
	T MW 20	//передача результата в MW20
	JO limt	//переход при переполнении
	JP pos0	//переход при результате больше 0
	JMZ les0	//переход при результате меньше или =0
	pos0: AN M 4.1	//результат не меньше нуля
	= Q 4.0	//выход при положительном результате
	= M 4.0	//вспомогательный меркер положительного результата
	les0: AN M 4.0	//результат<=0, не больше 0
	= Q 4.1	//выход при результате<=0
	= M 4.1	//вспомогательный меркер отрицательного результата
	limt: AN M 4.0	//переполнение, не горит лампа Q4.0
	AN M 4.1	//и Q4.1 – тоже не горит
	= Q 4.2	//выход при переполнении

Таблица 5.3.1.1: Примеры анализа бит слова состояния

6. КОМАНДЫ СДВИГА И ЦИКЛИЧЕСКОГО СДВИГА

Данные безусловные команды побитово сдвигают содержимое ассу1L или всего ассу1 влево либо вправо. Сдвигание на n бит влево эквивалентно умножению содержимого ассу1 на 2^n , а вправо - делению на 2^n .

Параметр инструкции $\langle n \rangle$ – число сдвигаемых разрядов. Если параметр не указан – в качестве параметра выступит содержимое младшего байта ассу2.

В качестве параметра $\langle n \rangle$ может выступать как константа, так и значение переменной.

При $\langle n \rangle = 0$ инструкции сдвига соответствуют нулевым инструкциям NOP 0.

При нециклическом сдвиге биты, которые ушли за область памяти – стираются.

Значение бита, сдвигаемого последним записывается в бит CC1 слова состояния. После выполнения команды сдвига при параметре $\langle n \rangle$ больше нуля биты OV и CC0 сбрасываются в «0». Для оценки состояния битов слова состояния используют команды переходов.

При циклическом сдвиге биты, вышедшие за пределы области памяти заново приходят в младшую часть младшего слова.

При ациклическом сдвиге освобождающиеся разряды заполняются нулями или состоянием знакового бита («0» - положительное число, «1» - отрицательное число).

Применяются следующие команды сдвига:

Команда	Описание	
SSI $\langle n \rangle$	Сдвиг вправо целого числа со знаком.	$n \in [0-15]$
SSD $\langle n \rangle$	Сдвиг вправо двойного целого числа со знаком	$n \in [0-32]$
SLW $\langle n \rangle$	Сдвиг значения в пределах слова влево	$n \in [0-15]$
SRW $\langle n \rangle$	Сдвиг значения в пределах слова вправо	$n \in [0-15]$
SRD $\langle n \rangle$	Сдвиг значения в пределах DW вправо	$n \in [0-32]$
SLD $\langle n \rangle$	Сдвиг значения в пределах DW влево	$n \in [0-32]$
RLD $\langle n \rangle$	Циклический сдвиг в пределах DW влево	$n \in [0-32]$
RRD $\langle n \rangle$	Циклический сдвиг в пределах DW вправо	$n \in [0-32]$
RLDA	Циклический сдвиг ассу1 влево через бит CC1 на 1 позицию	
RRDA	Циклический сдвиг ассу1 вправо через бит CC1 на 1 позицию	

Таблица 6.1: Команды сдвига и циклического сдвига

Пример: Использование команды RLDA

Содержимое	CC 1	Старшее слово ассу1				Младшее слово ассу1			
Бит №		31 16	15 0
До RLDA	X	0101	1111	0110	0100	0101	1101	0011	1011
После RLDA	0	1011	1110	1100	1000	1011	1010	0111	011X

X='0' или '1' - предыдущему состоянию Бита CC1 слова состояния

Таблица 6.2: Пример использования команды RLDA

Пример: Циклический сдвиг ассу1 влево через бит CC1 на 1 позицию

STL	Комментарий
L MD 400	//Загрузка содержимого MD400 в ACCU 1.
RLDA	//Циклический сдвиг ACCU 1 на 1 позицию влево через CC1
JP m1	//Перейти на метку m1, если последний выдвинутый в CC1 бит =1. (JP – переход, при результате >0, CC1 =«1», CC0 =«0»)

Таблица 6.3: Пример циклического сдвига влево

Пример: Ациклический сдвиг значения в пределах слова влево на 2 позиции

STL	Комментарий
L 2#101	//число, которое необходимо сдвинуть (5)
SLW 2	//команда сдвига и константа сдвига
T QW 4	//выгрузка результата (10100 bin) или 20 dec

Таблица 6.4: Пример ациклического сдвига влево

Пример: Ациклический сдвиг значения в пределах слова вправо на 3 позиции без указания параметра сдвига

STL	Комментарий
L 3	// запись в асс2 числа - порядок сдвига (3)
L 2#101	//число, которое необходимо сдвинуть (5)
SRW	//команда сдвига
T QW 4	//выгрузка результата (0)

Таблица 6.5: Пример ациклического сдвига вправо 1

Пример: Ациклический сдвиг числа вправо на 5 бит

Важно помнить, что SRW работает только над младшим словом, поэтому при выходе числа за диапазон возможна ошибка.

STL	Комментарий
L 2# 1111101000000000	//загрузка числа 32000
SRW 5	//сдвиг в пределах слова на 5 позиций вправо
T MW 2	//Передача результата 1000 в MW2

Сдвигаемое число	старший байт								младший байт							
№ бита	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Содержимое	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0
Комментарий	5 битов уйдут за область памяти и сотрутся															
Результат	старший байт								младший байт							
№ бита	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Содержимое	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0
Комментарий	На место стертых битов запишутся нули															

Таблица 6.6: Пример ациклического сдвига вправо 2

Пример: Циклический сдвиг числа влево на 3 бита

L 2#11100101001001111100011100101001 //загрузка числа 3844589353
 RLD 3 //сдвиг в пределах двойного слова на 3 позиции влево
 T MD 2 //Передача результата
 // 00101001001111100011100101001111 (691943759) в MD2

Старшее слово																Младшее слово															
1110010100100111																1100011100101001															
Старшее слово																Младшее слово															
0010100100111110																0011100101001111															

Рисунок 6.1: Пример циклического сдвига влево 2

Биты, вышедшие за пределы области памяти заново приходят в младшую часть младшего слова

6.1 Примеры

№	STL код	Комментарий
Задача: имеются две переменные: var1 и var2. Если #var1 > (#var2 + 120) – зажечь лампы Q4.3 Q4.4 и Q4.5. Если данное условие не выполняется – зажечь лампы Q4.0-----Q4.5		
1	L 328 T #var1 L 127 T #var2 L #var1 L #var2 + 100 >I JC NEXT = Q 4.0 = Q 4.1 = Q 4.2 NEXT: = Q 4.3 = Q 4.4 = Q 4.5	//загрузка значения //первой переменной //загрузка значения //второй переменной //загрузка переменной var1 в accu1-L //загрузка var2 в accu1-L, перемещение старого содержимого в accu2-L //Сложение ACCU 1-L с константой 100; сохранение результата в accu2-L //При ACCU 2 > ACCU 1 или #var1 > (#var2 + 120) //Выполнить условный переход на метку NEXT.
2	L 500 L 10 TAK -D T MW 30	//загрузка константы 500 в accu1 //загрузка константы 10 в accu1, перемещение старого содержимого (500) в accu2 //обмен содержимым между accu1 и accu2 //вычитание двойных целых чисел //выгрузка результата (-490) в MW30
Задача: имеются две переменных var1 и var в формате данных Int2. Записать большую из них в переменную max.		
3	L 333 T #var1_i L 500 T #var2_i L #var1_i L #var2_i >I JCN met1 TAK met1: T #max_i	//загрузка значения //первой переменной //загрузка значения //второй переменной //запись var1_i в accu1L //запись var2_i в accu1L, перемещение var1_i в accu2L // var1_i > var2_i? //если нет, уйти на метку met1 //обмен содержимым между accu1 и accu2. //записать содержимое accu1 в # max_i

На языке STL написать счетчик, который всякий раз, при нажатии на кнопку I0.0 будет увеличивать значение переменной var (формат Int) на 1		
4	A I 0.0 FP #bool_temp JCN m0 L #var1 + 1 T #var1 m0: NOP 0	//опрос входа I0.0 //выделение переднего фронта I0.0 //переход при отсутствии фронта //загрузка переменной var1 в accu1L //увеличение на 1 //передача результата
Задача: имеются три переменных var1 и var в формате данных Int2. Записать большую из них в переменную max.		
5	L 333 T #var1 L 5000 T #var2 L 740 T #var3 L #var1 L #var2 >I JCN met1 TAK met1: T #max L #max L #var3 >I JCN met2 TAK met2: T #max	//загрузка значения //первой переменной //загрузка значения //второй переменной //загрузка значения //третьей переменной //запись var1 в accu1L //запись var2 в accu1L, перемещение var1 в accu2L //var1>var2? //если нет, уйти на метку met1 //обмен содержимым между accu1 и accu2. //записать содержимое accu1 в #max //Теперь большая из переменных var1 и var2 хранится в переменной max //Осталось сравнить содержимое переменной max с var3 //запись переменной max в accu1L //запись var3 в accu1-L, перемещение max в accu2L //max>var3 //если нет – уйти на met2 //если да – поменять местами содержимое аккумуляторов //запись содержимого accu1L в переменную max
<p>Имеется кнопка, подключенная к входу I0.0. Если кнопка нажата (0-3) раз – ничего не происходит.</p> <p>(4-6) раз – горит лампа Q4.0 (7-10) раз – горят лампы Q4.0, Q4.1 и Q4.2.</p> <p>Если кнопка I0.0 нажата более 10 раз – ничего не происходит.</p> <p>По входу I0.1 количество нажатий на I0.0 обнуляется</p>		

6	A I 0.0	// подсчет
	FP M 0.0	//количества
	JCN m0	//нажатий
	L #var1i	//на кнопку
	L 1	// подключенную
	+I	// к входу I0.0
	T #var1i	//
	m0: NOP 0	//
	//от 3 до 6//	//диапазон счета от 3 до 6
	L #var1i	//загрузка var1i в accu1L
	L 3	//загрузка числа 3 в accu1L, перемещение var1i в accu2L
	>=I	//сравнение целых чисел
	= #temp1b	//если var1i больше или = 3, установить temp1b в «1»
		//
	L #var1i	// загрузка var1i в accu1L
	L 6	// загрузка числа 6 в accu1L, перемещение var1i в accu2L
	<=I	// сравнение целых чисел
	A #temp1b	// если var1i меньше или = 6, и temp1b = «1», т.е. в var1i в диапазоне [3-6]
	= Q 4.0	//то зажечь лампу Q4.0
		//
	//от 7 до 10//	//диапазон счета от 7 до 10
	L #var1i	//загрузка var1i в accu1L
	L 7	//загрузка числа 7 в accu1L, перемещение var1i в accu2L
	>=I	// сравнение целых чисел
	= #temp2b	// если var1i больше или = 7, установить temp2b в «1»
		//
	L #var1i	//загрузка var1i в accu1L
	L 10	//загрузка числа 10 в accu1L, перемещение var1i в accu2L
	<=I	// сравнение целых чисел
	A #temp2b	// если var1i меньше или = 10, и temp2b = «1», т.е. в var1i в диапазоне [7-10]
	JCN m1	//если нет – уйти на метку
		//
	L 2#111	//зажечь лампы
	T QB 4	// Q4.0, Q4.1 и Q4.2
		//
	m1: L #var1i	//загрузка var1i в accu1L
	L 10	//загрузка числа 10 в accu1L, перемещение var1i в accu2L
	>=I	// сравнение целых чисел
	= #temp3b	//если var1i больше или = 10, установить temp3b в «1»
	JCN m2	//иначе – уйти на метку m2
		//
	L 0	//потушить
	T QB 4	// все лампы
		//
	m2: NOP 0	//нулевая инструкция
	A I 0.1	//вход сброса
	= #temp4b	//установка вспомогательного бита в «1»
	JCN m3	//переход на метку
	L 0	//обнуление
	T #var1i	//значения переменной
	m3: NOP 0	//нулевая инструкция

Имеются пять целых чисел. Необходимо найти их среднее арифметическое и сохранить его в переменной sredR

7	L	5	//загрузка числа 5 в accu1L
	T	#var1i	//перемещение числа 5 в переменную var1i
	L	7	//загрузка числа 7 в accu1L
	T	#var2i	//перемещение числа 7 в переменную var2i
	L	4	//загрузка числа 4 в accu1L
	T	#var3i	//перемещение числа 4 в переменную var3i
	L	8	//загрузка числа 8 в accu1L
	T	#var4i	//перемещение числа 8 в переменную var4i
	L	5	//загрузка числа 5 в accu1L
	T	#var5i	//перемещение числа 5 в переменную var5i
			//
	L	#var1i	// загрузка переменной var1i в accu1L
	L	#var2i	//запись var2i в accu1L, перемещение var1 в accu2L
	+I		//сложение целых чисел
	T	#var1i	//передача результата в переменную var1i
			//
	L	#var3i	// загрузка переменной var3i в accu1L
	+I		// сложение целых чисел (содержимого accu1 с var3i)
	T	#var1i	// передача результата в переменную var1i
			//
	L	#var4i	// загрузка переменной var4i в accu1L
	+I		// сложение целых чисел
	T	#var1i	// передача результата в переменную var1i
			//
	L	#var5i	// загрузка переменной var5i в accu1L
	+I		// сложение целых чисел
	T	#var1i	// передача результата в переменную var1i
	ITD		//преобразование результата из Int в Dint
	DTR		//преобразование из Dint в Real
	L		// (Прямого из Int в Real не предусмотрено)
		5.000000e+000	//загрузка числа 5.0 в accu1
	/R		//деление чисел с плавающей точкой
	T	#sredR	//передача результата (5.8) в переменную sredR

Имеются два дробных числа. Необходимо отсечь от них целую часть и сравнить дробные части чисел; большую дробную часть записать в переменную #max

8	L 2.789	//загрузка числа 2.789 в accu1
	T #var1	//перемещение числа 2.789 в var1
	TRUNC	// отброс дробной части
	T #var1d	//передача результата (число 2) в формате Dint в var1d
	DTR	//преобразование числа 2 из Dint в Real (из 2 в 2.0)
	T #var1r	//передача числа 2.0 в var1r
	L #var1	//загрузка переменной var1
	TAK	// обмен содержимым между accu1 и accu2
	-R	//вычитание чисел с плавающей точкой
	T #var1	//передача результата (0.789) в var1
		//
		//
	L 7.345	//загрузка числа 7.345 в accu1
	T #var2	//перемещение числа 7.345 в var2
	TRUNC	// отброс дробной части
	T #var2d	//передача результата (число 7) в формате Dint в var2d
	DTR	//преобразование числа 7 из Dint в Real (из 7 в 7.0)
	T #var2r	//передача числа 7.0 в var2r
	L #var2	//загрузка переменной var2
	TAK	// обмен содержимым между accu1 и accu2
	-R	//вычитание чисел с плавающей точкой
	T #var2	//передача результата (0.3 45) в var2
		//
		//
	L #var1	////запись var1 в accu1
	L #var2	//запись var2 в accu1, перемещение var1 в accu2
	>R	//var1>var2?
	JCN met1	//если нет, уйти на метку met1
	TAK	//обмен содержимым между accu1 и accu2.
	met1: T #maxr	//записать содержимое accu1 в #max

Решить прошлую задачу, но с тремя числами		
9	L 2.789 T #var1r TRUNC T #var1d DTR T #var1rr L #var1r TAK -R T #var1r L 7.937 T #var2r TRUNC T #var2d DTR T #var2rr L #var2r TAK -R T #var2r L 5.988 T #var3r TRUNC T #var3d DTR T #var3rr L #var3r TAK -R T #var3r L #var1r L #var2r >R JCN met1 TAK met1: T #maxr L #maxr L #var3r >R JCN met2 TAK met2: T #maxr	//первая переменная// Буква в г в конце имени переменной – формат данных переменной (R- real. D- Dint, I – Int b-bool) //вторая переменная// //третья переменная// //сравнение первых двух переменных// //сравнение результата с третьей переменной
	<p>Имеются два выключателя и одна лампа. Каждый выключатель должен включать/выключать лампу, независимо от положения второго выключателя. Выключатели I0.0 и I0.1. Лампа Q4.0</p>	
	A I 0.0 X I 0.1 = Q 4.0	//первичный опрос переменной //самая элементарная функция XOR //выгрузка

Имеются две кнопки: I0.0 и I0.1. Определить, какая была нажата первой. Если первой была нажата I0.0 – зажечь лампы Q4.0, Q4.1, Q4.2, Q4.3. Если первой была нажата I0.1. – зажечь лампы Q4.4, Q4.5, Q4.6, Q4.7

11	<pre>//первый вход// AN #secondb A(O I 0.0 O #firstb) FP #firstb S #firstb JCN m1 L 2#1111 T QB 4 m1: NOP 0 //второй вход// AN #firstb A(O I 0.1 O #secondb) FP #secondb S #secondb JCN m2 L 2#11110000 T QB 4 m2: NOP 0</pre>	<pre>//описание вход1 и установки булевой firstb в «1» //второй вход не запущен раньше //самоподхват первого входа //первый вход //самоподхват первого входа //самоподхват первого входа //передний фронт //установка firstb в «1» //если firstb в состоянии «0» - уйти на метку, иначе //загрузить 1111 в accu1 //и передать на выходной модуль //нулевая инструкция //описание вход 2 и установки булевой secondb в «1» //первый вход не запущен раньше //самоподхват второго входа //второй вход //самоподхват второго входа //самоподхват второго входа //передний фронт //установка secondb в «1» //если secondb в состоянии «0» - уйти на метку, иначе //загрузить 11110000 в accu1 //и передать на выходной модуль //нулевая инструкция</pre>
необходимо найти ток двигателя, если известны следующие параметры: коэффициент мощности, мощность, напряжение		
12	<pre>L 380.0 T #voltageR L 0.9 T #cos_phiR L 3000.0 T #powerR L #voltageR L #cos_phiR *R T #voltageR L #powerR ТАК /R T #curR</pre>	<pre>//загрузка числа 380.0 в accu1 //перемещение числа 380.0 в переменную voltageR //загрузка числа 0.9 в accu1 //перемещение числа 0.9 в переменную cos_phiR //загрузка числа 3000.0 в accu1 //перемещение числа 3000.0 в переменную powerR //загрузка значения переменной voltageR в accu1 // загрузка переменной cos_phiR в accu1, перемещение voltageR в accu2 //умножение чисел с плавающей точкой //выгрузка результата в переменную voltageR // загрузка значения переменной powerR в accu1 //обмен содержимым аккумуляторов //деление чисел с плавающей точкой //Выгрузка результата 8.77 в переменную curR</pre>

Таблица 6.1.1: Примеры

6.2 Вопросы для самоконтроля

Что такое слово состояния? Назначение бита OS
Какая команда копирует содержимое ассу1 в ассу2?
Алгоритм выполнения операции маскирования
Команды, выполняющие округление чисел с плавающей точкой, разница между ними.
Какая команда копирует содержимое ассу2 в ассу1?
В чем разница между циклическим и ациклическим сдвигом?
Перечислите все биты слова состояния.
Назначение бита BR.
Назначение бита CC1.
Назначение бита CC0.
В чем разница между битами CC0 и CC1?
В чем разница между битами OV и OS?
Назначение бита OV.
Назначение бита OS.
При каких условиях оба бита CC0 и CC1 устанавливаются в логическую «1»

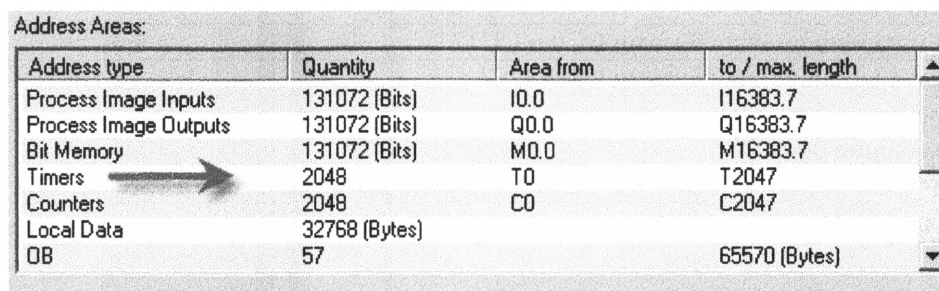
Таблица 6.2.1: Вопросы для самоконтроля

7. АППАРАТНЫЕ ТАЙМЕРЫ

Иногда, при автоматизации технологических процессов, бывает необходимо осуществить задержку времени либо выполнить какие-либо действия через заданные промежутки времени. В среде программирования Step-7 данные действия реализуются с помощью таймеров.

Таймеры имеют область памяти, зарезервированную в CPU. Максимальное количество доступных для использования таймеров различается в зависимости от модели CPU.

Для того чтобы узнать количество зарезервированного места под таймеры необходимо: Установить online-связь с PLC, запустить HW-Config, в верхнем меню выбрать вкладку PLC, выбрать подменю Module information, открыть вкладку Performance data и посмотреть параметр Quantity напротив адресного типа таймеров.



Address type	Quantity	Area from	to / max. length
Process Image Inputs	131072 (Bits)	I0.0	I16383.7
Process Image Outputs	131072 (Bits)	Q0.0	Q16383.7
Bit Memory	131072 (Bits)	M0.0	M16383.7
Timers	2048	T0	T2047
Counters	2048	C0	C2047
Local Data	32768 (Bytes)		
OB	57		65570 (Bytes)

Рисунок 7.1: Параметры CPU 414-2DP

В случае с CPU 414-2DP пользователю доступно 2048 аппаратных таймеров и допускается использовать таймеры с номерами от 0 до 2047. Изменить эти данные нельзя. Можно увеличить число аппаратных таймеров только заменой CPU на более мощный.

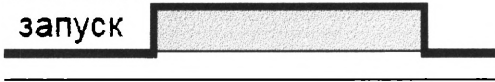
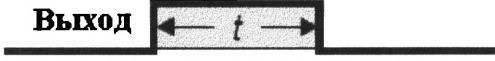
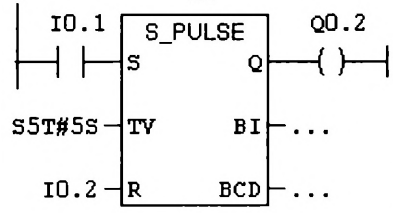


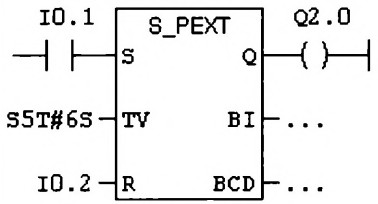


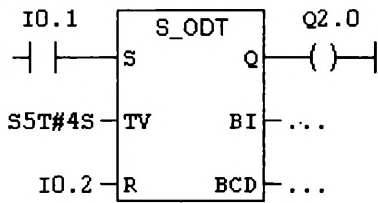

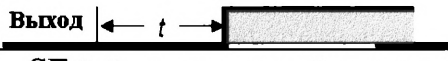
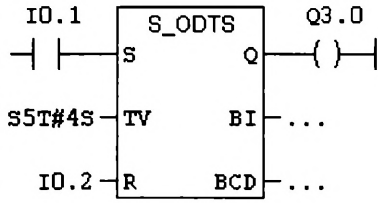


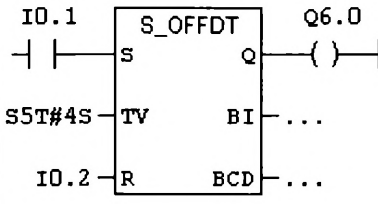
Наименование, описание диаграмма работы	LAD	STL
SP Импульсный таймер. Формирует импульсы с подтверждением по входу. Выход = «1» заданное время, если вход в состоянии «1» запуск  Выход 	<div style="text-align: center;">T0</div> 	<pre> A I 0.1 L S5T#5S SP T 0 A I 0.2 R T 0 NOP 0 NOP 0 A T 0 = Q 0.2 </pre>
SE Расширенный импульсный таймер. Формирует импульсы без подтверждения по входу. Выход = «1» заданное время, после появления «1» на входе. Даже если входной сигнал исчезнет. запуск  Выход 	<div style="text-align: center;">T2</div> 	<pre> A I 0.1 L S5T#6S SE T 2 A I 0.2 R T 2 NOP 0 NOP 0 A T 2 = Q 2.0 </pre>
SD Таймер с задержкой включения Таймер с подтверждением по входу. Сигнал на выходе будет ЧЕРЕЗ заданное время после того, как будет на входе запуск  Выход 	<div style="text-align: center;">T3</div> 	<pre> A I 0.1 L S5T#4S SD T 3 A I 0.2 R T 3 NOP 0 NOP 0 A T 3 = Q 2.0 </pre>
SS Таймер с задержкой включения с памятью. Сигнал на выходе будет ЧЕРЕЗ заданное время, после того, как будет на входе и останется там, даже если вход отключить. Таймер без подтверждения по входу запуск  Выход 	<div style="text-align: center;">T4</div> 	<pre> A I 0.1 L S5T#4S SS T 4 A I 0.2 R T 4 NOP 0 NOP 0 A T 4 = Q 3.0 </pre>
SF Таймер с задержкой выключения. На выходе есть сигнал сразу, после того, как появится на входе. Сигнал на выходе исчезнет через заданное время, после отключения сигнала с входа. запуск  Выход 	<div style="text-align: center;">T5</div> 	<pre> A I 0.1 L S5T#4S SF T 5 A I 0.2 R T 5 NOP 0 NOP 0 A T 5 = Q 6.0 </pre>

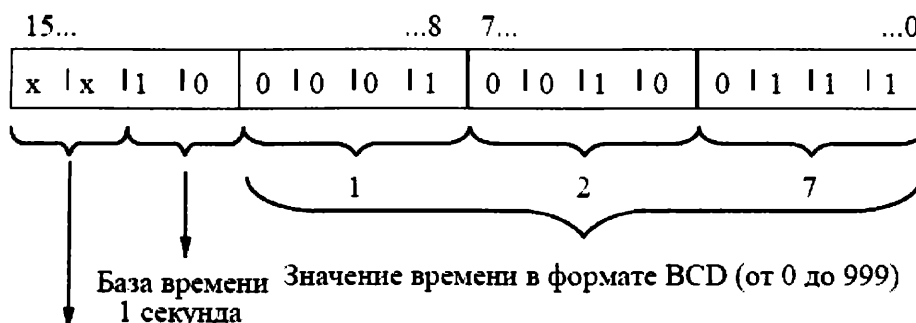
Таблица 7.1: Виды таймеров

Для задания времени работы аппаратного таймера резервируется одно 16-ти битовое слово.

Бит											
15	14	13	12	11	8	7	4	3			0
Не учитываются		б.м		Значение времени в формате BCD, диапазон [0-999]							

Таблица 7.2: Внутренняя организация таймерной памяти

Пример:



Несущественно: эти биты не учитываются при запуске таймера

Рисунок 7.2: Содержимое таймерной ячейки, загруженной значением времени 127 и базовым множителем 1 секунда.

Биты с 0 по 11 – значение времени в BCD-формате

Биты с 12 по 13 – база времени (базовый множитель)

Биты с 14 по 15 не учитываются при запуске таймера.

Значение времени – определяет количество временных отрезков. При актуализации таймера значение времени уменьшается на одну единицу через интервалы, установленные базой времени. Значит, чем меньше база времени – тем точнее таймер. Значение времени уменьшается до тех пор, пока не станет равным нулю.

Если значение времени работы таймера достигло «0» - отчитываемое таймером время истекло. При этом меняется битовое состояние (статус) таймера и он перестает быть активным до следующего запуска.

Таймеры обновляются асинхронно процессу сканирования программы пользователя, значит, состояние таймера в начале цикла сканирования может отличаться от состояния таймера в конце цикла сканирования. Это может привести к некорректной работе программы (несогласованности циклов). Поэтому, рекомендуется опрашивать таймер только 1 раз за программный цикл. Допускается задавать значение времени в двоично-десятичном (BCD), двоичном и шестнадцатеричном формате.

Пример:

Необходимо написать программу, реализующую следующую диаграмму

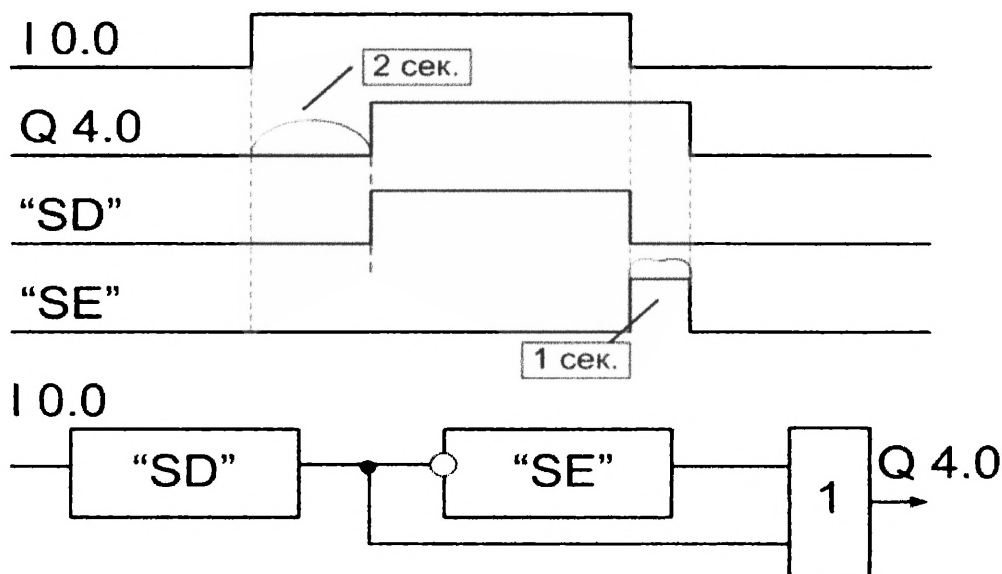


Рисунок 7.3: Пояснение к задаче

Предусмотреть запуск SE-таймера по событию окончания работы SD-таймера. Выходной сигнал должен образовываться путем сложения выходов двух таймеров.

Решение:

Неправильное	Правильное	Пояснение
A I 0.0	A I 0.0	При неправильном решении таймер T1 опрашивается несколько раз за программный цикл. При этом, результат опроса таймера в начале и в конце цикла может быть разным, что приведет к некорректной работе программы.
L S5T#2S	L S5T#2S	
SD T 0	SD T 0	Первый опрос
AN T 0	<u>A T 0</u>	
L S5T#1S	= M 0.0	Tc (время программного цикла)
SE T 1	<u>FN M 0.1</u>	
O T 1	L S5T#1S	Второй опрос
O T 0	SE T 1	
= Q 4.0	O T 1	Завершение работы таймера
	<u>O M 0.0</u>	
	= Q 4.0	

Таблица 7.3: Неправильное и правильное решение задачи

Возможны два варианта синтаксиса задания значения времени:

W#16#6DCD, где б - база времени, DCD- значение времени в BCD-формате

S5T#xHyMzSwMS,

x – количество часов, y-количество минут, z – количество секунд, w – количество миллисекунд.

Пример:

L S5T#1H22M30S //Загрузить значение времени 1ч 22мин 30сек в ассилL/

L S5TIME#10s //Длительность 10 с

L S5T#1m10ms //Длительность 1 м + 10 мс

Во втором случае происходит автоматический выбор базы времени. Максимальное время, которое можно задать составляет 2 часа 46 минут 30 секунд или 9990 секунд.

База времени – бинарная константа, определяющая интервал времени, через который значение времени уменьшается на единицу.

Содержимое 12 и 13 бита	База времени
00	10 ms
01	100 ms
10	1 s
11	10 s

Таблица 7.4: Связь базы времени с содержимым 12 и 13 бита

Необходимо помнить, что на разрешающую способность таймера накладываются некоторые ограничения. Значения, разрешающая способность которых слишком велика для требуемого диапазона округляются таким образом, что достигается требуемый диапазон, но не желаемая разрешающая способность.

Разрешающая способность	База времени
0.01 секунды	от 10 ms до 9s990ms
0.1 секунды	от 10 ms до 1m39s900ms
1 секунда	от 1s до 16m39s
10 секунд	от 10s до 2h46m30s

Таблица 7.5: Связь базы времени с разрешающей способностью

В форматах отображения FBD и LAD таймеры выглядят следующим образом:

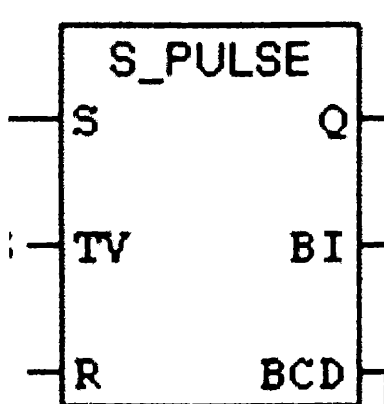
Отображение	описание		
	вход	тип данных	описание
	S	bool	вход запуска
	TV	s5time	длительность
	R	bool	вход сброса
	Q	bool	состояние таймера
	BI	word	текущее время в двоичном виде
	BCD	word	текущее время в BCD представлении

Таблица 7.6: Описание использования таймера в формате отображения LAD

При переводе на язык STL на месте неиспользованных входов/выходов будет написана нулевая инструкция пор 0, при её удалении код работает быстрее, но обратный переход на LAD/FBD становится невозможным.

7.1 Генератор импульсов на таймерах

В таблице приведен генератор импульсов на таймере с выходом q4.0 с периодом 3 секунды. Вспомогательный битовый меркер m0.0 перезапускает таймер каждые 3 секунды, выход устанавливается в соответствии с логикой «исключающее или». В первом цикле m0.0 = «0» и его инверсный опрос запустит таймер.

STL	Комментарий	LAD
<pre> A(AN T 0 L S5T#3S SE T 0) X M 0.0 = M 0.0 A M 0.0 = Q 4.0 </pre>	<pre> //инверсный опрос //RLO «1» //записать 3s в T0 // 2 XOR между //RLO и меркером //опрос меркера //выход </pre>	

Таблица 7.1.1: Генератор импульсов на таймерах

7.2 Таймерные инструкции

Для запуска таймера необходимо изменение уровня RLO с «0» на «1» до выполнения команды, описывающей таймер. Для таймера с задержкой выключения (off-delay timer) бит RLO должен измениться с «1» на «0».

Перед запуском таймера из ассу1 считывается длительность работы таймера. Рекомендуется загружать продолжительность работы таймера непосредственно перед заданием вида таймера в виде константы или переменной.

На языке STL поддерживаются следующие команды:

STL	Комментарий и пояснение
A T 1	Двоичный опрос таймера. Для двоичного опроса таймера допускается использовать те же команды, что и при опросе булевой ячейки памяти. A,O,X,AN,ON,XN. Например: A T 1 – опрос таймера и комбинирование результата опроса со значением бита RLO в соответствии с логической операцией AND (2И). Остальные команды опроса работают аналогично. Если таймер в момент опроса работает – результатом опроса на «1» будет «1».
FR T 1	Перезапуск (деблокировка) таймера. Скидывает таймер в 0 и заново его запускает. Перезапуск возможен только при высоком уровне сигнала на входе запуска. Для перезапуска таймера необходимо изменение RLO с «0» на «1» на входе перезапуска. Команда FR сбрасывает биты FC и OR слова состояния в «0».
L T 1	Загрузить двоичное значение текущего времени таймера в ассу1L. Старое содержимое ассу1 сохранится в ассу2.
LC T 1	Загрузить текущее время таймера в DCD формате (двоично-десятичном) в ассу1L. Старое содержимое ассу1 сохранится в ассу2. Точнее, записывает не само время в BCD, а только ту часть, которая без базового множителя
R T 1	Сброс таймера т.е. остановка текущей таймерной инструкции и обнуление значения времени с базовым множителем таймера при переднем фронте RLO. Скидывает бит FC в «0». После сброса таймера, пока в бите RLO находится «1» команда опроса таймера вернет значение «0»

Таблица 7.2.1: Таймерные инструкции

7.3 Аналоги аппаратных таймеров: (IEC-таймеры)

Также существуют аналоги аппаратных таймеров – IEC таймеры, т.е. таймеры, функционал которых соответствует международному стандарту IEC 1131.

Основные отличия от аппаратных таймеров:

Аппаратные таймеры считают от указанного времени в меньшую сторону, а аналоги – от нуля до указанного времени.

Аналоги таймеров работают синхронно циклу блока, из которого вызываются.

При первом вызове необходимо указывать номер аналога, при повторном – можно писать его имя.

Аналоги аппаратных таймеров представляют собой SFB-блоки.

Существуют следующие виды аналогов аппаратных таймеров:

SFB 3 (TP) генератор импульсов (аналог SE таймера)

SFB 4 (TON) генератор импульса с задержкой включения (аналог SD таймера)

SFB 5 (TOF) генератор импульса с задержкой выключения (аналог SF таймера)

Для того чтобы вызвать аналог таймера необходимо: открыть папку Libraries, выбрать пункт standard library и перетащить нужный элемент. При вызове потребуются через запятую после имени IEC таймера указать номер блока данных.

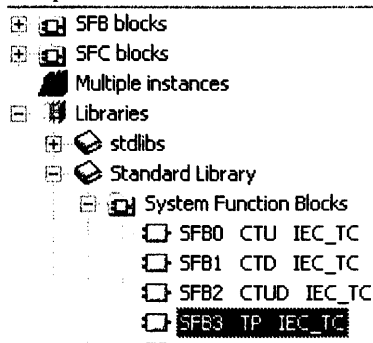


Рисунок 7.3.1: Доступ к IEC-таймерам.

При перетаскивании SFB-блока в рабочую область редактора вызов будет выделен красным цветом, и интерфейс блока будет недоступным

Для того чтобы появился интерфейс необходимо в вызове указать еще неиспользованный блок данных и в появившемся окне нажать кнопку yes. Simatic manager автоматически сгенерирует блок данных.

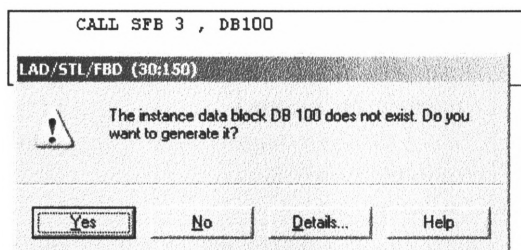


Рисунок 7.3.2: Генерация блока данных

После генерации блока данных автоматически появится доступ к интерфейсу блока

```
CALL "TP" , DB100
IN:=|
PT:=
Q :=
ET:=
```

Рисунок 7.3.4: Интерфейс аналога аппаратного таймера

Все три IEC-таймера имеют одинаковый интерфейс.

вход/выход	описание
IN	Булевый вход «запуск аналога таймера». Запуск будет осуществлен при изменении RLO с «0» на «1» на данном входе
PT	Время работы IEC-таймера. Диапазон: от 0 до 24d30h31m26s645ms
Q	Булевый выход. Логическая единица на данном выходе означает, что таймер запущен и работает. В противном случае на данном выходе находится логический ноль.
ET	Текущее значение IEC-таймера в формате time. Объем данных формата time – одно двойное машинное слово (DWord)

Таблица 7.3.1: Описание интерфейсов IEC-таймеров

Пример: На базе аналога SE-таймера получить аналог SP-таймера.

Решение:

При исчезновении единицы на входе – на выходе должен появиться «0». Или при отсутствии сигнала на входе, время работы таймера должно быть равным нулю. Решит данную задачу целесообразно через функциональный блок FB, дабы в OB1, из которого он вызывался создать схожий с изначальным интерфейс.

- 1) Создать функциональный блок FB1
- 2) Прописать в нем следующий код:

STL	Комментарий	Локальные данные
L #pt	//загрузить время работы	IN:
A #in	//опрос входа	in – bool
JC m1	//переход на метку при glo=1	pt – time
L T#0MS	//загрузка 0мс	OUT
m1: T #block.PT	//во время работы таймера	q – bool
	//	et – time
	//	STAT
CALL #block	//вызвать SFB3	block – SFB
IN:=#in	//параметры вызова	(<nr> стереть,
PT:=	//для удобства переменным	написать 3)
Q :=#q	//присвоены их же	
ET:=#et	// имена	

Таблица 7.3.2: Решение задачи на получение аналога SP-таймера

3) Вызвать функциональный блок FB1 из OB1

CALL FB 1, DB10

in:=I0.0
pt:=T#3S
q:=Q4.0
et:=MD300

4) Загрузить всю станцию в CPU. Так как, например, отсутствие блока данных может привести к ошибке.

5) Проверить результат и убедиться в том, что все работает.

Пример: На базе аналога SD таймера SFB 4 (TON) получить аналог SS-таймера.

Решение:

У SS-таймера сигнал на выходе будет через заданное время после того, как будет на входе и останется там, даже если таймер отключить.

Значит, при появлении сигнала на выходе, необходимо установить выход в «1» и оставить в положении «1» до тех пор, пока он не будет искусственно сброшен. С целью сброса можно добавить еще один вход – вход сброса r.

1) Создать функциональный блок FB2

2) Прописать в нем следующий код:

STL	Комментарий	Локальные данные
CALL #Timmer	//вызов SFB4	IN:
IN:=#in	//параметры	in – bool
PT:=#pt	// вызова	pt – time
Q :=	// SFB-4	r- bool
ET:=#ET	//	OUT
	//	q – bool
A #Timmer.Q	//опрос выхода	et – time
S #q	//установка выхода	STAT
	//	Timmer – SFB
A #r	//опрос сигнала сброса	(<nr> стереть,
R #q	//сброс выхода	написать 4)

Таблица 7.3.3: Решение задачи на получение аналога SS-таймера

3) Вызвать функциональный блок FB2 из OB1

CALL FB 2, DB12

in:=I0.0
pt:=T#2S
r:=I0.1
q:=Q4.0
ET:=MD400

4) Загрузить всю станцию в CPU. Так как, например, отсутствие блока данных может привести к ошибке.

5) Проверить результат и убедиться в том, что все работает.

7.3.1 Примеры

№	STL код	Комментарий
По переднему фронту I0.0 должна зажечься лампа Q0.0 и погаснуть через 3 секунды.		
1	A I 0.1 FP M 0.0 L S5T#3S SE T 0 A T 0 = Q 4.0	//опрос входа I0.1 // передний фронт //загрузка времени 3 секунды // запуск таймера //опрос таймера //выход
После перехода ЦПУ в режим RUN должна загореться лампа на 3 секунды.		
2	AN M 0.0 L S5T#3S SE T 0 A T 0 = Q 4.0	//инверсный опрос меркера, RLO=1 //загрузить 3 секунды //запуск таймера //опрос таймера //выход
По входу I0.0 должна загореться лампа на 3 секунды, если I0.0 остался в положении 1. Если входное воздействие исчезнет - лампа должна потухнуть. При повторном входном воздействии - должна заново запуситься на 3 секунды.		
3	A I 0.0 L S5T#3S SP T 0 A T 0 = Q 4.0	// опрос входа I0.0 // загрузить 3 секунды // запуск таймера // опрос таймера //выход
Через 3 секунды после перехода ЦПУ в режим RUN должна загореться лампа на 3 секунды.		
4	AN M 0.0 L S5T#3S SP T 0 AN T 0 L S5T#3S SP T 1 A T 1 = Q 4.0	//инверсный опрос меркера, RLO=1// // загрузить 3 секунды // запуск таймера // инверсный опрос таймера // // загрузить 3 секунды // запуск второго таймера // опрос таймера //выход
После перехода ЦПУ в режим RUN лампа должна включиться и моргать с частотой 1/3 Гц. Использовать только 1 таймер		
5	A(AN T 0 L S5T#3S SE T 0) X M 0.0 = M 0.0 A M 0.0 = Q 4.0	//опрос таймера с инверсией, RLO=1 //загрузить в таймер 3 секунды // скобки стоят т.к. инструкция задания вида таймера //скидывает бит «Fc» в «0» //без скобок был бы опрос. // 2XOR между RLO и меркером M0.0 //запись результата в m0,0 //опрос меркера //выход

После перехода ЦПУ в режим RUN лампа должна включиться и моргать с частотой 1/3 Гц. Использовать только 1 таймер. Необходимо посчитать число морганий

6	<pre> A(AN T 0 L S5T#3S SE T 0) X M 0.0 = M 0.0 A M 0.0 = Q 4.0 L S5T#3S LC T 0 ==I JCN m1 L 1 L MW 100 +I T MW 100 m1: NOP 0 </pre>	<pre> // опрос таймера с инверсией, RLO=1 //загрузить в таймер 3 секунды // скобки стоят т.к. инструкция задания вида таймера скидывает бит «Fc» в «0» //без скобок был бы опрос. // 2XOR между RLO и меркером M0.0 //запись результата в м0,0 //опрос меркера //выход //сравнение времени таймера с3-мя секундами //если время таймера не = 3с, уйти на метку //подсчет количества мерцаний </pre>
<p>После перехода ЦПУ в режим RUN лампа должна включиться и мерцать с частотой 1/3 Гц. Использовать только 1 таймер. Необходимо посчитать число мерцаний. После 7-го мерцания - прекратить мерцать. То есть оставить лампу зажженной</p>		
7	<pre> AN M 0.5 AN M 0.6 A(AN T 0 L S5T#3S SE T 0) X M 0.0 = M 0.0 A M 0.0 = Q 4.0 L S5T#3S LC T 0 ==I JCN m0 L 1 L MW 408 +I T MW 408 </pre>	<pre> //инверсный опрос меркера, RLO=1// //инверсный опрос меркера, RLO=1// //опрос таймера с инверсией, RLO=1 //загрузить в таймер 3 секунды // скобки стоят т.к. инструкция задания вида таймера скидывает бит «Fc» в «0» //без скобок был бы опрос. // 2XOR между RLO и меркером M0.0 //запись результата в м0,0 //опрос меркера //выход //сравнение времени таймера с3-мя секундами //подсчет числа мерцаний //сохранение результата в MW408 </pre>

	L MW 408 L 6 >I JCN m1 = M 0.5 = M 0.6 R T 0 L 2#0 T QB 4 m0: NOP 0 m1: NOP 0	//Как только результатом счета стало // 7 – убрать возможность самозапуска //так как при логике XOR возможны 2 варианта //значит – обнулить вход //если счет до 7 еще не завершен
По нажатию на I0.0 должна загореться лампа и "N" раз моргнуть с частотой "L" Гц. После этого другая лампа должна начать моргать с частотой "K" Герц.		
8	A I 0.0 S M 0.4 A M 0.4 AN M 0.5 AN M 0.6 A(AN T 0 A M 0.4 L S5T#3S SE T 0) X M 0.0 = M 0.0 A M 0.0 = Q 4.0 A M 0.4 L S5T#3S LC T 0 =I JCN m0 L 1 L MW 202 +I T MW 202 m0: L MW 202 L 10	//опрос входа //установка вспомогательного меркера //опрос вспомогательного меркера //дополнительное условие //запуска первого таймера ///<опрос таймера с инверсией, RLO=1 //вспомогательный меркер //загрузить в таймер 3 секунды // скобки стоят т.к. инструкция задания вида таймера скидывает бит «Fc» в «0» //без скобок был бы опрос. // 2XOR между RLO и меркером M0.0 //запись результата в m0,0 //опрос меркера //выход //условие того, что кнопка была нажата //сравнение времени таймера с 3с //каждые 3 секунды прибавлять 1 к mw202 //результат подсчета //если число мерцаний не равно 10

	<pre> ==I JCN m1 R M 0.4 = M 0.5 = M 0.6 AN M 0.4 A M 0.5 A(AN T 1 L S5T#5S SE T 1) X M 2.0 = Q 4.1 = M 2.0 m1: NOP 0 </pre>	<pre> //уйти на метку //иначе //убрать условие самозапуска //таймера //как только таймер досчитал до //условие запуска //второго таймера //таймер мерцания со второй частотой // меркер вспомогательный второй лампы //если число мерцаний не равно 10 </pre>
<p>По переднему фронту I0.0 должна загореться лампа и "N" раз моргнуть с частотой "L" Гц. После этого другая лампа должна начать моргать с частотой "K" Герц. Количество морганий необходимо посчитать и сохранять в меркерном слове. Счетчики не использовать.</p>		
9	<pre> A I 0.0 S M 0.4 A M 0.4 AN M 0.5 AN M 0.6 A(AN T 0 A M 0.4 L S5T#3S SE T 0) X M 0.0 = M 0.0 A M 0.0 = Q 4.0 A M 0.4 L S5T#3S LC T 0 ==I JCN m0 L 1 L MW 202 </pre>	<pre> //опрос входа //установка вспомогательного меркера //опрос вспомогательного меркера //дополнительное условие //запуска первого таймера // //опрос таймера с инверсией, RLO=1 //вспомогательный меркер //загрузить в таймер 3 секунды // скобки стоят т.к. инструкция задания вида таймера скидывает бит «Fc» в «0» //без скобок был бы опрос. // 2XOR между RLO и меркером M0.0 //запись результата в m0,0 //опрос меркера //выход //условие того, что кнопка была нажата //сравнение времени таймера с 3с //каждые 3 секунды прибавлять 1 к меркерному слову </pre>

	+I T MW 202	//результат подсчета
	m0: L MW 202	
	L 10	//если число мерцаний не равно 10
	=I	//уйти на метку
	JCN m1	//иначе
	R M 0.4	
	= M 0.5	//убрать условие самозапуска
	= M 0.6	
	AN M 0.4	//условие запуска
	A M 0.5	//второго таймера
	A(AN T 1	
	L S5T#5S	//таймер мерцания со второй частотой
	SE T 1	
)	
	X M 2.0	// меркер вспомогательный второй лампы
	= Q 4.1	
	= M 2.0	
	A M 2.0	//выполняется
	L S5T#5S	//подсчет
	LC T 1	//количества
	=I	//мерцаний
	JCN m20	//заданной
	L 1	//лампы
	L MW 302	//номер
	+I	//два
	T MW 302	//без счетчика
	m1: NOP 0	//если число мерцаний не равно 10
	m20: NOP 0	
По переднему фронту I0.0 должна загореться лампа и "N" раз моргнуть с частотой "L" Гц. После этого должна загореться вторая лампа и "M" раз моргнуть с частотой "K" Гц. По окончании цикла вторая лампа может оставаться зажженной. Количество морганий необходимо посчитать и сохранять в меркерном слове. Счетчики не использовать.		
10	A I 0.0	//опрос входа
	S M 0.4	//установка вспомогательного меркера
	A M 0.4	//опрос вспомогательного меркера
	AN M 0.5	//дополнительное условие
	AN M 0.6	//дополнительное условие
	A(AN T 0	//запуска первого таймера
		//опрос таймера с инверсией, RLO=1
	A M 0.4	//вспомогательный меркер

L S5T#3S SE T 0)	//загрузить в таймер 3 секунды // скобки стоят т.к. инструкция задания вида таймера скидывает бит «Fc» в «0» //без скобок был бы опрос.
X M 0.0 = M 0.0 A M 0.0 = Q 4.0	// 2XOR между RLO и меркером M0.0 //запись результата в m0,0 //опрос меркера //выход
A M 0.4 L S5T#3S LC T 0 ==I	//условие того, что кнопка была нажата //сравнение времени таймера с 3с
JCN m_0 L 1 L MW 202 +I T MW 202	//каждые 3 секунды прибавлять 1 к меркерному слову
m_0: L MW 202	//результат подсчета
L 10 ==I JCN m_1	//если число //мерцаний не равно 10 //уйти на метку
R M 0.4 = M 0.5 = M 0.6 AN M 0.4 A M 0.5	//полностью //исключить таймеру //возможность самозапуститься //условие запуска //второго таймера
A(AN T 1 L S5T#5S SE T 1) X M 2.0 = Q 4.1 = M 2.0	//таймер мерцания со второй частотой // меркер вспомогательный второй лампы
A M 2.0 L S5T#5S LC T 1 ==I	//выполняется //подсчет //количества //морганий
JCN m_20 L 1 L MW 302	//заданной //лампы //номер

	+I T MW 302 m_20: L 6 L MW 302 ==I JCN m7 R M 0.5 R T 1 R M 2.0 m_1: NOP 0 m7: NOP 0	//два //без счетчика //условие остановки //второго таймера //если число морганий не равно 10 //ничего не делать
По нажатию на кнопку I0.0 лампа должна моргать с частотой 2 Гц, по приходу импульса I0.1 лампа должна перестать моргать. При повторном приходе импульса I0.0 лампа должна возобновить моргание.		
11	A I 0.0 S M 0.4 A M 0.4 A(AN T 0 A M 0.4 L S5T#500MS SE T 0) X M 0.0 = M 0.0 A M 0.0 = Q 4.0 A I 0.1 R M 0.4	//опрос входа //установка вспомогательного меркера //опрос вспомогательного меркера // //опрос таймера с инверсией, RLO=1 //вспомогательный меркер //загрузить в таймер 3 секунды // скобки стоят т.к. инструкция задания вида таймера скидывает бит «Fc» в «0» //без скобок был бы опрос. // 2XOR между RLO и меркером M0.0 //запись результата в m0,0 //опрос меркера // //выход //сброс входа //сброс меркера
По входу I0.0 должна загореться лампа и моргать с частотой 2 Гц. По исчезновению входа - лампа должна потухнуть. По входу I0.1 частота мерцания должна измениться на 1Гц. По входу I0.3, мерцание должно прекратиться. При одновременном срабатывании входов I0.0 и I0.1 мерцание должно прекратиться. По входу I0.4 должно возобновиться мерцание с частотой 2Гц независимо от входа I0.3. А также должна заблокироваться возможность включения другими входами мерцания лампы. При решении задачи использовать только один таймер. Вести подсчет морганий лампы Q4.0 во всех режимах. Результаты сохранять в меркерной области памяти.		
12	A I 0.0 AN I 0.1 AN I 0.3 AN I 0.4 O I 0.4 JCN m0 L S5T#2S T #Vremya	/--вход I0.0--// //условия //работы //по //входу I0.0 //локальная переменная s5time

m0: FR T 0	//перезапуск таймера
A I 0.1	
AN I 0.0	//--вход I0.1--//
AN I 0.3	//условия работы
AN I 0.4	//по входу I0.1
JCN m1	
L S5T#4S	
T #Vremya	
m1: FR T 0	
A I 0.3	
AN I 0.4	//--сброс T0--//
JCN m2	
L S5T#0MS	
T #Vremya	
R Q 4.0	
R T 0	
R #zapusk	//локальная переменная
m2: A I 0.4	
JCN m3	
L S5T#2S	//--деблокировка T0//
T #Vremya	
R I 0.0	//осуществляется
R I 0.1	//сброс
R I 0.3	//входов
m3: AN I 0.3	
A(//--таймер--//
X I 0.0	//событие запуска//
X I 0.1	
)	
O I 0.4	
= #zapusk	
A(
AN T 0	//таймер//
A #zapusk	//условие запуска
L #Vremya	//загрузить #Vremya
SE T 0	//в таймер
)	//здесь
X M 4.0	//находится
= M 4.0	//такты
NOT	//генератор
A M 4.0	//с инверсией
A #zapusk	//перед
= Q 4.0	//выходом
L #Vremya	//подсчет импульсов//
LC T 0	//общее число //
=I	

	<pre> A T 0 JCN m_0 L 1 L MW 102 +I T MW 102 A I 0.0 JCN m_8 L S5T#2S L #Vremya =I A T 0 JC m4 m4: L #Vremya LC T 0 =I JCN m4_0 L 1 L MW 40 +I T MW 40 m_8: L MW 102 L MW 40 -I T MW 30 m_0: NOP 0 m4_0: NOP 0 </pre>	<pre> //--число по 2секунды--// //-----// //число морганий по 4 секунды// </pre>
Пока нажата кнопка I0.0 лампа Q4.0 должна мигать с определенной частотой		
13	<pre> A I 0.0 FN M 101.0 R T 0 R Q 4.0 A(AN T 0 A I 0.0 L S5T#1S SE T 0) X M 100.0 = M 100.0 A M 100.0 = Q 4.0 </pre>	<pre> //опрос входа //задний фронт //т.е. если отпустить кнопку ///произойдет сброс таймера и выхода // // опрос таймера с инверсией, RLO=1 //условие работы генератора //загрузить в таймер 1 секунду // скобки стоят т.к. инструкция задания вида таймера скидывает бит «Fc» в «0» //без скобок был бы опрос. // 2XOR между RLO и меркером M 100.0 //запись результата в M 100.0 //опрос меркера //выход </pre>

Таблица 7.3.1.1: Примеры использования таймеров

8. АППАРАТНЫЕ СЧЕТЧИКИ

Иногда, при автоматизации технологических процессов, бывает необходимо осуществить подсчет каких-либо импульсов. Например, деталей на сборочном конвейере.

В среде программирования Step-7 данные действия реализуются с помощью счетчиков.

Счетчик – функциональный элемент языка программирования

Step 7, занимающий одно 16-битное слово в зарезервированной области памяти.

Для использования доступны как аппаратные счетчики, так и их аналоги. Плюс аппаратных счетчиков в том, что они работают асинхронно блоку OB1 и работают быстрее блока OB1.

Счетчики имеют область памяти, зарезервированную в CPU. Максимальное количество доступных для использования счетчиков различается в зависимости от модели CPU.

Для того чтобы узнать количество зарезервированного места под счетчики необходимо:

Установить online-связь с PLC, запустить HW-Config, в верхнем меню выбрать вкладку PLC, выбрать подменю Module information, открыть вкладку Performance data и посмотреть параметр Quantity напротив адресного типа счетчиков.

Address Areas:

Address type	Quantity	Area from	to / max. length
Process Image Inputs	131072 (Bits)	I0.0	I16383.7
Process Image Outputs	131072 (Bits)	Q0.0	Q16383.7
Bit Memory	131072 (Bits)	M0.0	M16383.7
Timers	2048	T0	T2047
Counters	2048	C0	C2047
Local Data	32768 (Bytes)		
OB	57		65570 (Bytes)

Рисунок 8.1: Параметры CPU 414-2DP

В случае с CPU 414-2DP пользователю доступно 2048 аппаратных счетчиков и допускается использовать счетчики с номерами от 0 до 2047. Изменить эти данные нельзя. Можно увеличить число аппаратных счетчиков только заменой CPU на более мощный.

8.1 Инструкции счета

Предусмотрены следующие инструкции счета:

Инструкция	Описание
FR	Деблокировка счетчика. Передний фронт RLO перед командой деблокировки разблокирует счетчик, при этом команды установки и счета будут сброшены.
L	Загрузка текущего значения счетчика в двоичном коде (Integer) в ассу1-L. Старое содержимое ассу1 переместится в ассу2.
LC	Загрузка текущего значения счетчика в ассу1-L в BCD-коде. Старое содержимое ассу1 переместится в ассу2.
R	Сброс счетчика. Записывает значение ноль в счетчик, сбрасывает бит FC в «0».
S	Установка счетчика на заданное значение, то есть загрузка в счетчик содержимого ассу1-L при переднем фронте RLO. Значение ассу1L должно находиться в диапазоне от 0 до 999. Бит Fc сбрасывается в «0».
CU	Прямой счет. Увеличивает содержимое указанного счетчика на 1 при переднем фронте RLO и значении счетчика меньше 999. При достижении верхнего предела счета (999) увеличение и влияние переднего фронта на входе прямого счета прекращается. Бит переполнения OV в «1» не устанавливается, бит Fc сбрасывается в «0».
CD	Обратный счет. Уменьшает содержимое указанного счетчика на 1 при переднем фронте RLO и значении счетчика больше 0. При достижении нижнего предела счета (0) уменьшение и влияние переднего фронта на входе обратного счета прекращается. Бит переполнения OV в «1» не устанавливается, бит Fc сбрасывается в «0».

Таблица 8.1.1: Инструкции счета

8.2 Виды аппаратных счетчиков

Существует всего 3 вида аппаратных счетчиков: счетчик прямого счета, счетчик обратного счета и счетчик как прямого, так и обратного счета.

LAD	STL	Комментарий
<p>CU Счетчик прямого счета</p>	<pre> A I 0.0 CU C 0 BLD 101 A I 0.1 L C#30 S C 0 A I 0.2 R C 0 L C 0 T MW 100 LC C 0 T MW 200 A C 0 = Q 4.0 </pre>	<p>//запуск счетчика прямого счета //счетчик прямого счета //графический мусор //по входу I0.1 //загрузить число 30 //и установить счетчик на 30 //по входу I0.2 //сбросить счетчик //значение Int счетчика //загружать в mw100 //значение BCD счетчика //загружать в mw200 //опрос счетчика //Q =«0» если знач-е счетчика =0</p>
<p>CD Счетчик обратного счета</p>	<pre> A I 0.0 CD C 0 BLD 101 A I 0.1 L C#30 S C 0 A I 0.2 R C 0 L C 0 T MW 100 LC C 0 T MW 200 A C 0 = Q 4.0 </pre>	<p>//запуск счетчика обратного счета //счетчик обратного счета //графический мусор //по входу I0.1 //загрузить число 30 //и установить счетчик на 30 //по входу I0.2 //сбросить счетчик //значение Int счетчика //загружать в mw100 //значение BCD счетчика //загружать в mw200 //опрос счетчика //выход =«0» если значение счетчика =0</p>
<p>CUD Счетчик прямого и обратного счета.</p>	<pre> A I 0.0 CU C 11 A I 0.1 CD C 11 A I 0.2 L C#500 S C 11 A I 0.3 R C 11 L C 11 T MW 100 LC C 11 T MW 200 A C 11 = Q 4.0 </pre>	<p>//запуск счетчика вперед //счетчик прямого и обратного счета //по входу I0.1 //считать назад // по входу I0.2 //установить число 500 //в счетчик // по входу I0.3 //сбросить счетчик //значение Int счетчика //загружать в mw100 //значение BCD счетчика //загружать в mw200 //опрос счетчика //выход =«0» если значение счетчика =0</p>

Таблица 8.2.1: Виды аппаратных счетчиков

8.3 Примеры

№	STL	Комментарий
Осуществлять подсчет срабатываний входа I0.0. При срабатывании входа I0.1 – установить значение счетчика в 7.		
1	A I 0.0 CU C 1 A I 0.1 L C#7 S C 1	//Опрос состояния входа I0.0 //Счет вперед при переднем фронте опроса I0.0 //Опрос состояния входа I 0.1 //Загрузка константы 7 в ACCU 1-L. //Установка счетчика C1 на заданное значение при переходе RLO из 0 в 1.
При переходе CPU в режим RUN установить значение счетчика 10. При каждом срабатывании входа I0.0 – увеличивать значение счетчика на 1		
2	AN M 0.0 L C#10 S C 1 A I 0.1 CU C 1	//условие самозапуска //Загрузка константы 10 в ACCU 1-L. //Установка счетчика C1 на значение 7 при переднем фронте RLO //Опрос состояния входа I 0.1 //счет вперед при переднем фронте RLO
При переходе CPU в режим RUN установить значение счетчика 10. При каждом срабатывании входа I0.0 – уменьшать значение счетчика на 1		
3	AN M 0.0 L C#10 S C 1 A I 0.1 CD C 1	//условие самозапуска //Загрузка константы 10 в ACCU 1-L. //Установка счетчика C1 на значение 7 при переднем фронте RLO //Опрос состояния входа I 0.1 //счет назад при переднем фронте RLO
При срабатывании входов I0.0 И I0.7 счетчик считает вперед, при срабатывании входа I0.1 – счетчик считает назад, при срабатывании входа I0.2 – значение счетчика устанавливается =100, при срабатывании входов I0.3 И I0.0 – счетчик сбрасывается в 0. Если значение счетчика не равно нулю – горит лампа Q4.0.		
4	A I 0.0 A I 0.7 CU C 11 A I 0.1 CD C 11 A I 0.2 L C#100 S C 11 A I 0.3 A I 0.0 R C 11 L C 11 T MW 100 LC C 11 T MW 200 A C 11 = Q 4.0	//опрос входа I0.0 //операция 2И между итогом опроса I0.0 и I0.7 //запуск счетчика вперед // опрос входа I0.1 //запуск счета назад // опрос входа I0.2 // загрузка числа 100 в аккумулятор //установка счетчику значения 100 // опрос входа I0.3 // операция 2И между итогом опроса I0.3 и I0.0 //сбросить счетчик //значение Int счетчика //загружать в mw100 //значение BCD счетчика //загружать в mw200 //опрос счетчика //выход =«0» если значение счетчика =0

Таблица 8.3.1: Примеры использования аппаратных счетчиков

8.4 Аналоги аппаратных счетчиков: (IEC-счетчики)

Также существуют аналоги аппаратных счетчиков – IEC счетчики, т.е. счетчики, функционал которых соответствует международному стандарту IEC 1131.

Аналоги аппаратных счетчиков работают синхронно тому блоку, из которого их вызывают и представляют собой SFB-блоки:

SFB 0 (CTU) – прямой счет счетчика

SFB 1 (CTD) – обратный счет счетчика

SFB 2 (CTUD) – прямой обратный счет счетчика

При первом вызове необходимо указывать номер аналога, при повторном – можно писать его имя. Нижний предел счета аналогов таймеров -32767, а верхний - + 32767.

Для того чтобы вызвать аналог счетчика необходимо: открыть папку Libraries, выбрать пункт standard library и перетащить нужный элемент. При вызове потребуются через запятую после имени IEC таймера указать номер блока данных.

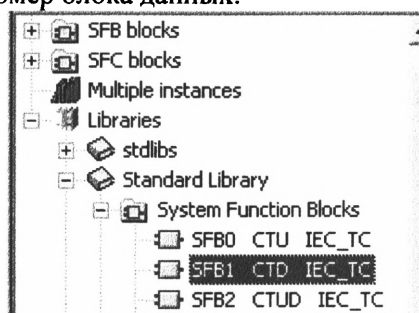


Рисунок 8.4.1: Доступ к IEC-счетчикам

При перетаскивании SFB-блока в рабочую область редактора вызов будет выделен красным цветом, и интерфейс блока будет недоступным

Для того чтобы появился интерфейс необходимо в вызове указать еще неиспользованный блок данных и в появившемся окне нажать кнопку yes. Simatic manager автоматически сгенерирует блок данных.

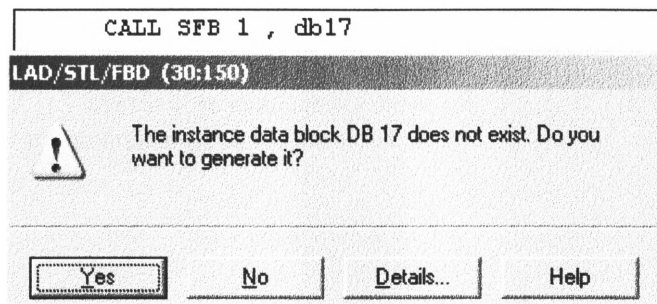


Рисунок 8.4.2: Генерация блока данных

После генерации блока данных автоматически появится доступ к интерфейсу блока

```

CALL "CTD" , DB17
CD :=
LOAD:=
PV :=
Q :=
CV :=

```

Рисунок 8.4.3: Интерфейс аналога аппаратного счетчика

Все три ИЕС-счетчика имеют схожий интерфейс и могут иметь следующие входа/выхода:

вход/выход	описание
CU	Булевый вход «запуск счета вперед». Запуск будет осуществлен при изменении RLO с «0» на «1» на данном входе
CD	Булевый вход «запуск счета назад». Запуск будет осуществлен при изменении RLO с «0» на «1» на данном входе
LOAD	Булевый вход «установка значения счетчика». При изменении RLO с «0» на «1» на данном входе в счетчик будет записано значение с входа PV формата Int.
PV	Вход формата Int. Может содержать значение в формате Int, которое будет передано в счетчик при положительном фронте на входе Load.
CV	Выход формата Int. В данный выход передается значение счетчика.
Q	Булевый выход. Для SFB0 =«1» если $CV > PV$, для SFB1 =«1» если $CV \leq 0$.
QU	Булевый выход. Для SFB2 =«1» если $CV \geq PV$.
QD	Булевый выход. Для SFB2 =«1» если $CV \leq 0$.

Таблица 8.4.1: Элементы интерфейса аналогов счетчиков

Пример: Аналог аппаратного счетчика по входу I0.1 получает значение 7, по входу I0.0 уменьшает свое значение на единицу. При достижении нуля – должна загореться лампа Q4.0

Решение:

STL	Комментарий
CALL "CTD" , DB11	//вызов аналога счетчика
CD :=I0.0	//счет назад по переднему фронту I0.0
LOAD:=I0.1	//установка счетчику значения с входа PV
PV :=7	//загружаемое значение
Q :=Q4.0	//Булевый выход
CV :=MW100	//Int-выход, передача результата

Таблица 8.4.2: Пример использования ИЕС-счетчика

9. РАБОТА С АНАЛОГОВЫМИ МОДУЛЯМИ

Аналоговые модули ввода/вывода, в зависимости от конкретной модели модуля могут быть как с гальванической развязкой, так и без нее. Её наличие указывается в документации к сигнальному модулю. Гальваническая развязка представляет собой микросхему, способную выдержать напряжение до 500 Вольт, при ее отсутствии возрастает опасность электрического пробоя.

Цена модуля с гальванической развязкой выше цены модуля без нее.

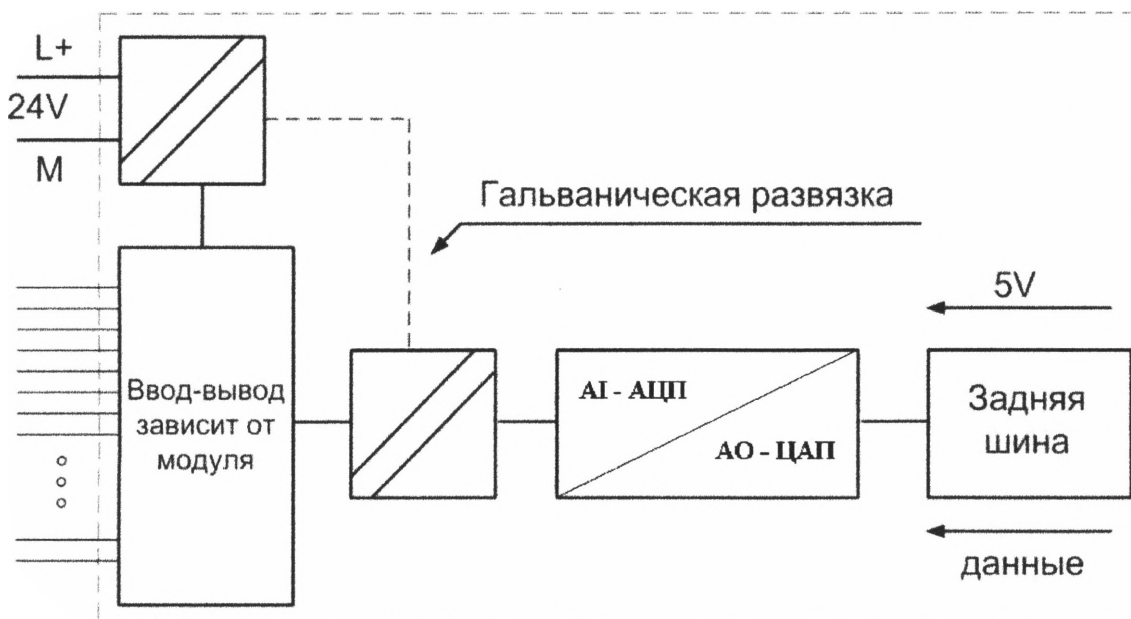


Рисунок 9.1: Обобщенная структурная схема аналогового модуля

Работа аналогового модуля зависит от типа подключенного к нему устройства. Изначально аналоговые модули спроектированы для работы с датчиком напряжения.

Некоторые модели аналоговых модулей поддерживают работу с датчиками различных типов. Для работы с аналоговым модулем необходимо осуществить аппаратную и программную настройку блока. Для аппаратной настройки модуля, некоторые из них имеют на боковой стороне блок переключателей, настройка осуществляется путем изменения угла поворота блока переключателей.

Программная настройка осуществляется через утилиту HW-Config.

В общем случае настройка аналоговых модулей осуществляется в соответствии с таблицей:

Тип датчика	Аппаратно	Программно
Датчик напряжения до 1В	А	Е
Датчик напряжения свыше 1В	В	Е
Активный 4-х проводный датчик тока	С	4DMV
Пассивный 2-х проводный датчик тока	Д	2MV
Активный 2-х проводный датчик тока	С	4DMV
3-х и 4-х проводные датчики сопротивления	А	R/RD/RCD (зависит от датчика)
Термопара	А	ТС

Таблица 9.1: Аппаратная и программная настройка аналогового модуля

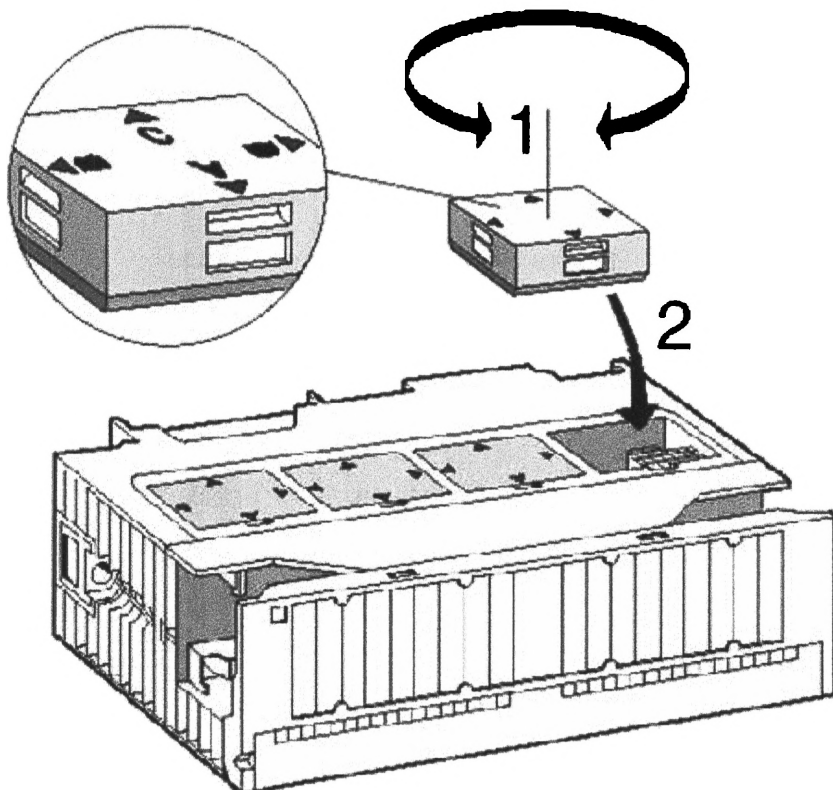


Рисунок 9.2: Аппаратная настройка аналогового модуля

Для осуществления программной настройки аналогового модуля необходимо запустить утилиту HW-Config, дважды щелкнуть левой кнопкой мыши по аналоговому модулю и настроить параметры на вкладке Inputs.

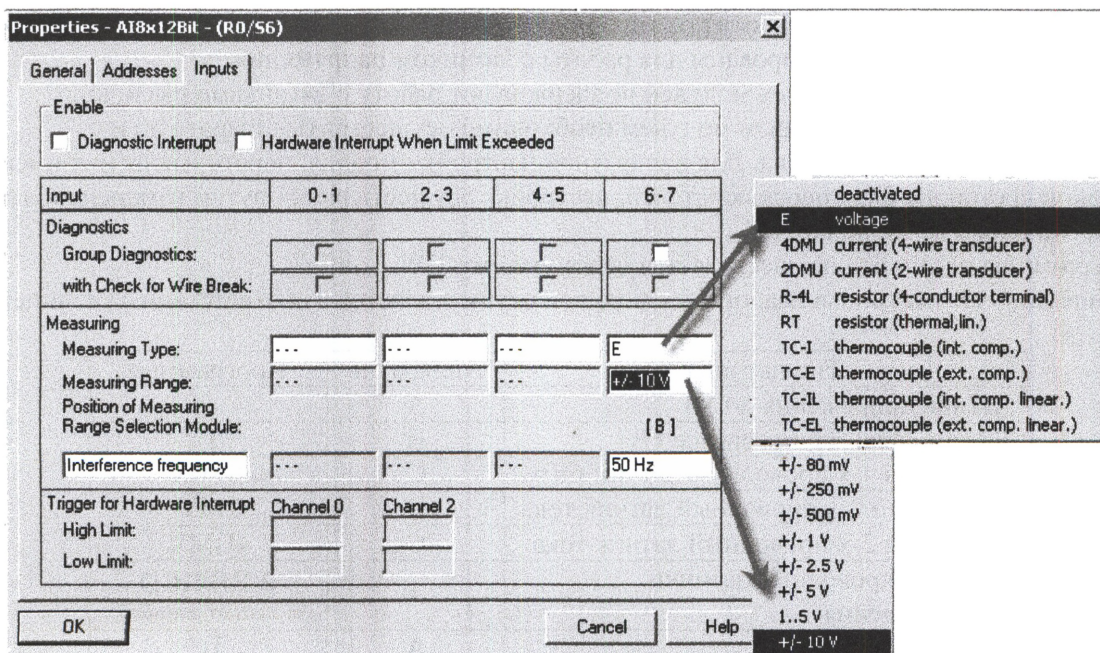


Рисунок 9.3: Программная настройка аналогового модуля

9.1 Адресация аналоговых модулей

АЦП, как и ЦАП, могут быть либо одноканальными, либо многоканальными.

Рассмотрим адресацию на примере входного аналогового двухканального модуля 6ES7 331-7KB02-0AB0

Slot	Module	Order number	Fi...	M...	I address	Q...	Comment
1	PS 307 2A	6ES7 307-1BA00-0AA0					
2	CPU 315-2 DP	6ES7 315-2AG10-0AB0	V2.6	2			
3	DP				204,7"		
4	DI16xDC24V	6ES7 321-1BH82-0AA0			0...1		
5	DI16xDC24V	6ES7 321-1BH82-0AA0			2...3		
6	AI2x12Bit	6ES7 331-7KB02-0AB0			288...291		
7	DO16xDC24V/0.5A	6ES7 322-1BH10-0AA0				4...5	

Рисунок 9.1.1: Отображение аналогового модуля в HW-Config

Каждый из каналов занимает 2 байта, следовательно, при конечном адресе 291, то канал Ch0 имеет адрес Piw288, а Ch1 – Piw290. (Peripheral Input Word). В случае использования выходных модулей перед адресом ставится область памяти PQW.

Адреса, которые поставляет система на этапе настройки аппаратной части модулей AI/AO выходят за диапазон области отображения входов/выходов CPU. Сделано это для того чтобы была возможность получать одни и те же данные при считывании данных из одной и той же ячейки памяти.

Аналоговые величины преобразуются в модуле в число из диапазона [0 – 27648] или, в случае симметричных значений (например +/-10В) в число из диапазона [-27648 – +27648].

При превышении аналоговой величины на входе максимального значения модуль может выдать значение в диапазоне [27648 – 32767], что может привести к переполнению.

Для преобразования аналогового сигнала в численное значение физической величины необходимо осуществить масштабирование.

При масштабировании физической величины в диапазоне от 0 до N необходимо составить пропорцию.

$$\frac{N}{27648} = \frac{\text{Величина}}{\text{текущ. знPIW}} \Rightarrow \text{Величина} = \frac{N \cdot \text{тек. знPIW}}{27648}$$

При необходимости масштабирования входного сигнала от нижнего до верхнего предела физической величины следует воспользоваться формулой:

$$\text{Выход} = \left(\frac{\text{Вход}}{27648} \cdot (\text{Верхний_предел} - \text{Нижний_предел}) \right) + \text{Нижний_предел}$$

Для обеспечения точности рекомендуется производить вычисления с числами формата данных Real.

Пример: Аналоговый сигнал с выхода лазерного дальномера должен быть преобразован в численное значение расстояния в диапазоне 0-15 метров. Результат сохранить в MD100.

$$\frac{15 \text{ метров}}{27648} = \frac{X_{\text{метров}}}{\text{выход из PIW}} \Rightarrow X_{\text{метров}} = \frac{15 \cdot \text{выход из PIW}}{27648}$$

STL	Комментарий
L PIW 288	//Загрузка значения аналогового входа в аккумулятор
ITD	//Преобразование Int в Dint
DTR	//Преобразование Dint в Real
L 15.0	//Загрузка числа 15.0 (real) в аккумулятор
*R	//Умножение чисел с плавающей точкой
L 27648.0	//Загрузка числа 27648.0 (real) в аккумулятор
/R	//Деление чисел с плавающей точкой
T MD 100	//Передача результата в MD100

Таблица 9.1.1: Пример преобразования 1

Пример: Аналоговый сигнал 0-5В с адресом PIW290 должен быть преобразован в физическую величину из диапазона 100-500. Результат сохранить в MD200.

$$\text{Выход} = \left(\frac{\text{PIW}290}{27648} \cdot (500 - 100) \right) + 100$$

STL	Комментарий
L PIW 290	//Загрузка значения аналогового входа в аккумулятор
ITD	//Преобразование Int в Dint
DTR	//Преобразование Dint в Real
L 27648.0	//Загрузка числа 27648.0 (real) в аккумулятор
/R	//Деление чисел с плавающей точкой
L 400.0	// Загрузка числа 400.0 (real) в аккумулятор (400=500-100)
*R	//Умножение чисел с плавающей точкой
L 100.0	//Загрузка числа 100.0 (real) в аккумулятор
+R	//Сложение чисел с плавающей точкой
T MD 200	//Передача результата в MD100

Таблица 9.1.2: Пример преобразования 2

ЗАКЛЮЧЕНИЕ

Вы внимательно прочитали учебное пособие и Вам в нем все понятно? Считайте, что Вы достаточно хорошо усвоили основы программирования ПЛК. Теперь, если Вам это интересно или нужно, Вы можете смело браться за самостоятельное изучение программируемых логических контроллеров. Объем учебного пособия не позволил включить все функции и возможности ПЛК. Не затронуты также вопросы по сетевым интерфейсам, функциям, обслуживанию автоматизированной техники. Автор не ставил перед собой такой цели. Главной задачей было – дать основные представления о программировании ПЛК.

Автор будет благодарен за любые предложения, направленные на улучшение структуры и качества учебного пособия.

ПРИЛОЖЕНИЕ А
(справочное)
ОБЗОР STL ИНСТРУКЦИЙ

Английская мнемоника	Описание
+	Прибавление целой константы к аккумулятору 1
+AR1	Прибавление содержимого аккумулятора 1 к адресному регистру AR1 Прибавление смещения к адресному регистру AR1
+AR2	Прибавление содержимого аккумулятора 1 к адресному регистру AR2 Прибавление смещения к адресному регистру AR1
+D	Сложение длинных целых чисел
+I	Сложение целых чисел
+R	Сложение вещественных чисел
-D	Вычитание длинных целых чисел
-I	Вычитание целых чисел
-R	Вычитание вещественных чисел
*D	Умножение длинных целых чисел
*I	Умножение целых чисел
*R	Умножение вещественных чисел
/D	Деление длинных целых чисел
/I	Деление целых чисел
/R	Деление вещественных чисел
=	Операция присваивания Операция промежуточного присваивания
==D	Сравнение на "равно" длинных целых чисел
==I	Сравнение на "равно" целых чисел
==R	Сравнение на "равно" вещественных чисел
<D	Сравнение на "меньше" длинных целых чисел
<I	Сравнение на "меньше" целых чисел
<R	Сравнение на "меньше" вещественных чисел
<=D	Сравнение на "меньше или равно" длинных целых чисел
<=I	Сравнение на "меньше или равно" целых чисел
<=R	Сравнение на "меньше или равно" вещественных чисел
<>D	Сравнение на "не равно" длинных целых чисел
<>I	Сравнение на "не равно" целых чисел
<>R	Сравнение на "не равно" вещественных чисел
>D	Сравнение на "больше" длинных целых чисел
>I	Сравнение на "больше" целых чисел
>R	Сравнение на "больше" вещественных чисел
>=D	Сравнение на "больше или равно" длинных целых чисел
>=I	Сравнение на "больше или равно" целых чисел
>=R	Сравнение на "больше или равно" вещественных чисел
A	И
A(И с открывающейся скобкой

ABS	Нахождение абсолютной величины вещественного числа
OPN	Открытие блока данных
BE	Безусловное окончание блока
BEC	Условное окончание блока по RLO="1"
BTD	Преобразование двоично-десятичного в длинное целое
BTI	Преобразование двоично-десятичного в целое
CALL	Безусловный вызов блока
CD	Счет назад
CU	Счет вперед
CC	Условный вызов блока без фактических параметров
CLR	Сброс флага логического результата
COS	Вычисление косинуса вещественного числа
DEC	Декрементирование аккумулятора 1
DTB	Преобразование длинного целого в двоично-десятичное
DTR	Преобразование длинного целого в вещественное число
EXP	Вычисление экспоненты вещественного числа
FN	Выделение отрицательного фронта флага RLO
FP	Выделение положительного фронта флага RLO
INC	Инкрементирование аккумулятора 1
INVD	Инвертирование всех битов длинного целого
INVI	Инвертирование всех битов целого
ITB	Преобразование целого в двоично-десятичное
ITD	Преобразование целого в длинное целое
L	Загрузка в аккумулятор 1
LAR1	Загрузка содержимого аккумулятора 1 в адресный регистр 1 Загрузка указателя в адресный регистр 1
LAR2	Загрузка содержимого аккумулятора 1 в адресный регистр 2 Загрузка указателя в адресный регистр 2
LAR1 AR2	Копирование содержимого адресного регистра AR2 в AR1
LC	Загрузка в аккумулятор в двоично-десятичном формате
LN	Вычисление натурального логарифма вещественного числа
LOOP	Цикл
L STW	Загрузка в аккумулятор слова состояния
MOD	Нахождение остатка от деления (деление по модулю)
MCR(Начало MCR-области
)MCR	Конец MCR-области
MCRA	Объявление MCR-области
MCRD	Деактивирование MCR-области
NEGD	Инвертирование знака длинного целого числа
NEGI	Инвертирование знака целого числа
NEGR	Инвертирование знака вещественного числа
NOP 0	Пустой оператор

NOT	Инверсия флага логического результата
O	Опрос на единицу (для операции "ИЛИ")
OD	Операция "ИЛИ" над двойными словами
ON	Опрос на ноль (для операции "НЕ-ИЛИ")
OW	Операция "ИЛИ" над словами
POP	Копирование содержимого аккумулятора 2 в аккумулятор 1
PUSH	Копирование содержимого аккумулятора 1 в аккумулятор 2
R	Сброс бита в ноль
RLD	Циклический сдвиг влево двойного слова
RLDA	Циклический сдвиг влево через флаг CC1
RND	Округление вещественного числа
RND+	Округление до ближайшего большего целого
RND-	Округление до ближайшего меньшего целого
RRD	Циклический сдвиг вправо двойного слова
RRDA	Циклический сдвиг вправо через флаг CC1
RS	RS-триггер (LAD)
S	Установка бита в единицу
SAVE	Сохранить RLO в регистре BR
SD	Таймер – формирователь задержки включения
SE	Таймер удлиненного импульса
SF	Таймер – формирователь задержки выключения
SIN	Вычисление синуса вещественного числа
SLD	Сдвиг влево двойного слова
SLW	Сдвиг слова влево
SP	Таймер – формирователь импульса
SQR	Вычисление квадрата числа типа данных Real
SQRT	Вычисление квадратного корня числа типа данных Real
SRD	Сдвиг вправо двойного слова
SRW	Сдвиг слова вправо
SS	Таймер задержки включения с запоминанием
SSD	Сдвиг вправо длинного целого
SSI	Сдвиг вправо целого числа
T	Пересылка из аккумулятора 1
T STW	Пересылка содержимого аккумулятора 1 в слово состояния
TAK	Обмен содержимым аккумуляторов
TAN	Вычисление тангенса вещественного числа
TAR	Обмен содержимым адресных регистров
TAR1	Загрузка содержимого адресного регистра AR1 в аккумулятор 1 Загрузка содержимого адресного регистра AR1 в переменную
TAR2	Загрузка содержимого адресного регистра AR2 в аккумулятор 1 Загрузка содержимого адресного регистра AR2 в переменную
TAR1 AR2	Копирование содержимого адресного регистра AR1 в AR2

T STW	Пересылка содержимого аккумулятора 1 в слово состояния
TRUNC	Округлить до целого, отбросом дробной части (для Real)
UC	Безусловный вызов блока без фактических параметров
X	Исключающее ИЛИ
X(Исключающее ИЛИ с открытием скобки
XN	Исключающее ИЛИ-НЕ
XN(Исключающее ИЛИ-НЕ с открытием скобки
XOD	Поразрядное исключающее ИЛИ с двойными словами (32 бита)
XOW	Поразрядное исключающее ИЛИ со словами (16 бита)

СПИСОК ЛИТЕРАТУРЫ

1. Шалыто А.А. Логическое управление. Методы аппаратной и программной реализации алгоритмов
2. Бесекерский В.А., Изранцев В.В. Системы автоматического управления с микроЭВМ. М.: Наука. 1987-320с.
3. Гросс В.Ю. Введение в микропроцессорную технику [текст]: учеб. пособие/ В.Ю. Гросс, Б.З. Кузнецов. – Новосибирск: Новосиб. гос. акад. водного трансп., 2006. – 174 с.
4. Полещенко Д.А., Кривоносов В.А. Автоматизация технологических процессов и производств. Методические указания к выполнению лабораторных работ. Старый Оскол, СТИ МИСиС, 2008. – 60 с.
5. Деменков Н.П. Языки программирования промышленных контроллеров: Учебное пособие/ Под ред. К.А Пупкова. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2004. – 172 с.: ил.

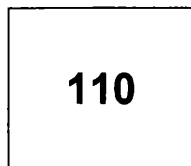
ОГЛАВЛЕНИЕ

Глава	Условные обозначения	Страница
	Предисловие	3
	Список принятых сокращений	4
1	ОБЩИЕ СВЕДЕНИЯ О ПЛК SIEMENS	5
1.1	Обзор аппаратной части ПЛК	5
1.1.1	Виды перезапусков CPU	9
1.2	Карты памяти	10
1.2.1	Защита карты памяти паролем	12
1.3	Концепция памяти ПЛК	14
1.3.1	Настройка реманентной (сохраняемой) памяти	15
1.4	Влияние внешних воздействий на работу CPU	16
1.5	Адаптер для связи ПЛК с ПК	17
2	ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ SIMATIC MANAGER	18
2.1	Описание Simatic Manager	18
2.2	Установка Simatic Manager	19
2.3	Запуск Simatic Manager	20
2.4	Общие настройки проекта (Options-Customize)	22
2.5	Создание проекта в Simatic Manager	27
2.6	Адресация слотов и установка начальных адресов модулей	34
2.7	Форматы данных	36
2.8	Адресное пространство CPU	37
2.9	Simatic S7 проект	38
2.10	Программные блоки	39
2.11	Назначение организационных блоков	41
2.12	Выбор адаптера	42
2.13	Создание первой программы	45
2.14	Пример программы на LAD	47
2.15	Пример программы на FBD	48
2.16	Пример программы на STL	49
2.17	Симуляция работы программы	50
2.18	Загрузка данных на карту памяти	51
2.19	Настройка параметров архивирования	54
2.20	On-line программа	57
2.21	Таблица символов Symbol Table	58
2.22	Таблица переменных VAT (Variable Table)	61
2.23	Перекрестные ссылки (Reference data)	62
2.24	Локальный стек	63
3	ЛОГИЧЕСКИЕ ОПЕРАЦИИ	64
3.1	Элементарные логические операции	64
3.1.1	Инвертирование результата логической операции	71
3.1.2	Исключающее ИЛИ с тремя переменными	71
3.1.3	Присвоение (функция Assign)	71
3.1.4	Сброс и установка бита (функции Set и Reset), функции установки, сброса и выгрузки	72
3.1.5	Триггеры	73
3.1.6	Функции выделения фронтов	74
3.2	Сложные двоичные логические операции	75
3.2.1	A(: И с открывающей скобкой	76
3.2.2): Закрывающая скобка	76

3.2.3	Объединение AND-функций (И) в операторе OR (ИЛИ)	76
3.2.4	Объединение OR (ИЛИ) и XOR (Исключающее ИЛИ) в операторе AND (И)	77
3.2.5	Объединение функций AND (И) в операторе XOR (Исключающее ИЛИ)	78
3.2.6	Объединение функций OR (ИЛИ) в операторе XOR и наоборот	78
3.2.7	Инвертирование вложенных выражений	79
3.2.8	Примеры	80
3.2.9	Упражнения	84
3.2.10	Решение	88
4	АККУМУЛЯТОРЫ	91
4.1	Инструкции для работы с аккумуляторами	91
4.2	Работа с крупными единицами данных	92
4.3	Инструкции сравнения	93
4.3.1	Примеры	94
4.4	Математические операции	95
4.5	Инкрементирование и декрементирование	96
4.6	Типичные ошибки	97
4.6.1	Примеры	98
4.7	Упрощенная форма записи	101
4.8	Математические функции	102
4.9	Функции преобразования	103
4.9.1	Примеры	104
4.10	Поразрядные логические инструкции со словами	105
4.10.1	Примеры	106
4.11	Выполнение операции маскирования	107
5	СЛОВО СОСТОЯНИЯ (STATUS WORD)	108
5.1	Состояние флагов условия после выполнения команд	110
5.2	Анализ битов слова состояния через команды переходов	111
5.3	Работа со словом состояния	112
5.3.1	Примеры	113
6	КОМАНДЫ СДВИГА И ЦИКЛИЧЕСКОГО СДВИГА	114
6.1	Примеры	118
6.2	Вопросы для самоконтроля	123
7	АППАРАТНЫЕ ТАЙМЕРЫ	124
7.1	Генератор импульсов на таймерах	129
7.2	Таймерные инструкции	130
7.3	Аналоги аппаратных таймеров: (IEC-таймеры)	131
7.3.1	Примеры	134
8	АППАРАТНЫЕ СЧЕТЧИКИ	143
8.1	Инструкции счета	144
8.2	Виды аппаратных счетчиков	145
8.3	Примеры	146
8.4	Аналоги аппаратных счетчиков: (IEC-счетчики)	147
9	РАБОТА С АНАЛОГОВЫМИ МОДУЛЯМИ	149
9.1	Адресация аналоговых модулей	151
	ЗАКЛЮЧЕНИЕ	153
	Приложение А. ОБЗОР STL-ИНСТРУКЦИЙ	154
	СПИСОК ЛИТЕРАТУРЫ	158



3 2 7 4 8 1 7



1 3 2 9 4 5 0 3 - 0 2 6 7



1 3 2 9 4 5 0 3 - 0 2 6 7 - 1



0001220132014