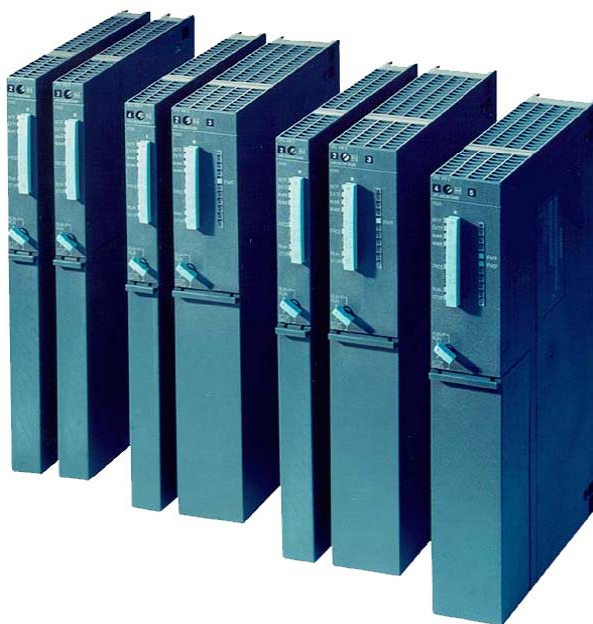


## **Программируемые контроллеры SIMATIC S7**

**2-й уровень профессиональной подготовки**

### **S7- PROF2**

**Учебник  
(раздаточный материал)**



## **Предисловие**

Данный учебник предназначен для слушателей курса S7-PROF2, а также может использоваться в качестве справочного материала для специалистов по разработке и обслуживанию систем управления на базе контроллеров SIMATIC S7-300/400.

Содержание учебника является 2-й частью материалов по программируемым контроллерам SIMATIC S7 (1-я часть – курс S7-PROF1).

Учебник содержит разделы по структурному программированию, работе с данными, системными функциями и S7-библиотеками, а также вопросам расширенной диагностики SIMATIC S7-300/400.

Большое количество иллюстративного материала и примеров решения различных задач позволит пользователю успешно освоить даже самые сложные темы.

При подготовке материалов использовалась версия STEP 7 Professional v5.4

Автор курса: руководитель отдела АСУ ООО "Промышленная Автоматизация"  
к.т.н. Альтерман И.З.

Об авторе:

Автор имеет 10-летний опыт преподавательской деятельности, работал старшим преподавателем в РГТА. Автор 17 изобретений и 12 научных статей, в 1995 г. в Санкт-Петербургском государственном университете аэрокосмического приборостроения защитил кандидатскую диссертацию по системам управления.

С 1996 по 2000 г. работал в ООО "СИМЕНС" г. Москва мастер-тренером по системам автоматизации SIMATIC, провел более 130 учебных курсов. Более 4 лет сотрудничал с компанией Siemens VAI ведущим консультантом. В последнее время успешно работает разработчиком и руководителем проектов по АСУТП на базе технических решений компании SIEMENS. С 2008 г. проводит авторские курсы по системам SIMATIC.

© Альтерман И.З., 2007-2011  
[igoraltreman@hotmail.com](mailto:igoraltreman@hotmail.com)

Без полученного разрешения копирование, распространение и использование содержания данного документа или его части в коммерческих целях запрещается. Нарушители понесут ответственность за причиненные убытки.

## Содержание

|   |    |
|---|----|
| Структурное программирование .....                        | 5  |
| Использование блоков FB и FC .....                        | 5  |
| Определение параметров блока.....                         | 6  |
| Фактические и формальные параметры .....                  | 6  |
| Локальные данные .....                                    | 6  |
| Статические локальные данные .....                        | 7  |
| Вызов блока: общая информация.....                        | 7  |
| Вызов FC с параметрами.....                               | 8  |
| Присвоение значения функции .....                         | 8  |
| Вызов FC без параметров .....                             | 8  |
| Вызов FB с параметрами.....                               | 9  |
| Вызов FB без параметров и статических данных.....         | 9  |
| Параметры блока EN/ENO.....                               | 10 |
| Корректировка вызовов блока.....                          | 10 |
| Изменение интерфейса блока FC/FB без остановки CPU.....   | 11 |
| Свойства блоков .....                                     | 12 |
| Создание исходных файлов .....                            | 13 |
| Пример исходного текста на STL .....                      | 13 |
| Установка защиты на блок.....                             | 14 |
| Блоки данных.....   | 15 |
| Создание DB .....   | 15 |
| Атрибуты блоков DB.....                                   | 15 |
| Адресация данных.....                                     | 16 |
| Открытие блока DB .....                                   | 17 |
| Размер блока данных и его номер.....                      | 18 |
| Оптимизация размера блока данных .....                    | 18 |
| Наблюдение и управление переменными блока данных .....    | 19 |
| Сохранение текущих значений блока данных в проекте .....  | 19 |
| Сброс данных в начальные значения.....                    | 19 |
| Отображение экземплярных DB .....                         | 20 |
| Отображение экземплярных DB .....                         | 20 |
| Блоки DB, связанные с типом данных UDT.....               | 20 |
| Организационные блоки.....                                | 21 |
| Типы OB.....  | 21 |
| Стартовая информация, временные локальные данные .....    | 21 |
| Циклические прерывания .....                              | 22 |
| Обновление образа процесса при вызове OB.....             | 23 |
| Прерывания по времени суток .....                         | 23 |
| Программные прерывания.....                               | 24 |
| Аппаратные прерывания .....                               | 25 |
| Различия в обработке прерываний в S7-300 и в S7-400 ..... | 25 |
| Использование локальных данных в блоке прерывания.....    | 25 |
| ОВ синхронных ошибок.....                                 | 26 |
| ОВ асинхронных ошибок.....                                | 27 |
| Стартовые блоки .....                                     | 28 |



|  |    |
|--|----|
| Программно-доступные регистры S7- CPU .....                          | 31 |
| Инструкции с аккумуляторами .....                                    | 31 |
| Регистр STW .....  | 34 |
| Адресные регистры .....  | 35 |
| Режимы адресации данных .....  | 36 |
| Косвенная адресация через ячейки памяти .....                        | 36 |
| Косвенная адресация через адресные регистры .....                    | 37 |
| Пример: копирование элементов из одного блока DB в другой DB .....   | 38 |
| Тестирование программ с косвенной адресацией .....                   | 40 |
| Расширенные средства диагностики и тестирования программ .....       | 40 |
| Точки останова .....   | 40 |
| Режим "Force" .....  | 43 |
| Комплексные и параметрические типы данных .....                      | 44 |
| Тип данных DT (DATE_AND_TIME) .....                                  | 45 |
| Тип данных STRING .....  | 46 |
| Тип данных ARRAY .....   | 46 |
| Тип данных STRUCT .....  | 47 |
| Передача параметров сложного типа .....                              | 47 |
| Тип данных POINTER .....   | 48 |
| Тип данных ANY .....   | 49 |
| Пример программы подсчета контрольной суммы .....                    | 50 |
| Системные блоки .....  | 51 |
| Краткий обзор системных функций .....                                | 52 |
| Вызов SFC и SFB .....  | 55 |
| Ошибки выполнения системной функции .....                            | 55 |
| Примеры использования системных функций .....                        | 56 |
| Установка часов реального времени (SFC 0) .....                      | 56 |
| Копирование DB из загрузочной памяти в рабочую память (SFC 20) ..... | 56 |
| Запись сообщения в диагностический буфер (SFC 52) .....              | 56 |
| Определение интервала времени (SFC 64) .....                         | 57 |
| Применение IEC-таймера SFB4 (ON Delay) .....                         | 58 |
| Библиотеки .....   | 59 |
| Конфигурация и содержание стандартной S7- библиотеки .....           | 59 |
| Пример использования функции FC6 из библиотеки IEC .....             | 60 |
| Масштабирование аналоговой величины - FC105 .....                    | 60 |
| Создание библиотек пользователя .....                                | 61 |
| Заключение .....   | 61 |
| Приложение. Система команд S7-CPU .....                              | 62 |

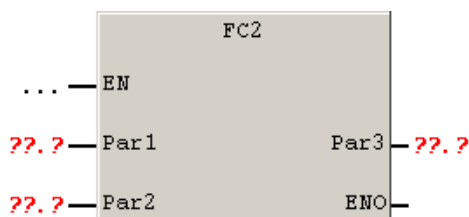
## Структурное программирование

Структурная программа содержит блоки FC (функции) и FB (функциональные блоки), которые предназначены для разбиения программы на функционально законченные части. Функциональное разделение всей программы имеет следующие преимущества:

- Улучшается читаемость программы. Сложная задача представляется как набор более конкретных частных задач.
- Уменьшается размер программы за счет многократного использования в одном цикле одних и тех же блоков (с параметрами).
- Сокращается время разработки программы за счет использования проверенных решений (блоков из библиотек или других проектов).
- Быстрая модернизация программы за счет замены отдельных блоков. Блоки могут загружаться в CPU во время исполнения программы.
- Поэтапный ввод в эксплуатацию за счет постепенного включения блоков в цикл программы. Блоки могут быть созданы и проверены независимо друг от друга.
- Сокращение времени цикла CPU за счет вызовов блоков в разные моменты времени (чередование вызовов).
- Привлечение к разработке программы нескольких авторов – параллельная разработка программы.

### Общие характеристики FB и FC

#### Функция (FC)



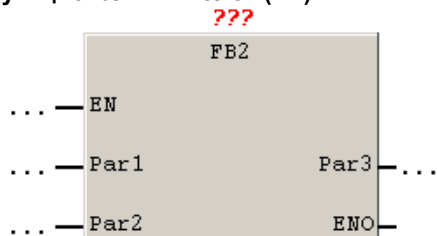
- Может иметь возвращаемое значение.
- Может иметь назначаемые параметры (IN, OUT, IN\_OUT).
- Обязательная инициализация (указание фактических данных) параметров в вызываемом блоке (для блоков с параметрами).
- Может иметь локальные данные (в L-стеке).

**Механизм передачи параметров** – передача по “ссылке” (через 32-разрядные указатели адреса, размещаемые после команды вызова).

Требования IEC 1131-3:

- Только один выходной параметр RET\_VAL. Он определяет тип функции (например, BOOL, INT и т.д.).
- Нельзя использовать доступ к глобальным переменным.

#### Функциональный блок (FB)

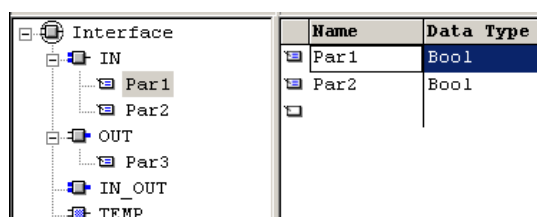


- Может иметь назначаемые параметры (IN, OUT, IN\_OUT). Параметры могут иметь начальные значения
- Необязательная инициализация параметров в вызываемом блоке
- Может иметь собственные статические данные (сохраняемые до следующего вызова)
- Может иметь локальные данные (в L-стеке)

**Механизм передачи параметров и статических данных** – передача через память (основан на использовании вспомогательного блока данных).

## Определение параметров блока

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

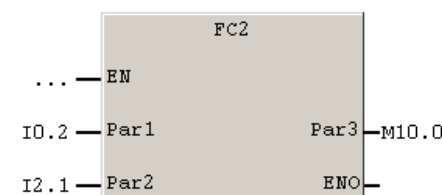


Параметр блока необходимо задавать в разделе "IN" интерфейсной части блока, если требуется только прочитать его значение в программе блока. Такие параметры называются "входными".

Если значение параметра только записывается, то Вы должны использовать параметр в разделе "OUT" ("выходной" параметр).

Если же параметр может быть, как записан, так и считан, то Вы имеете дело с параметром типа "входной/выходной" (IN\_OUT). Имя параметра может содержать до 24 символов. Символы в верхнем и нижнем регистрах не различаются. Имя не может совпадать ни с одним из ключевых слов.

## Фактические и формальные параметры



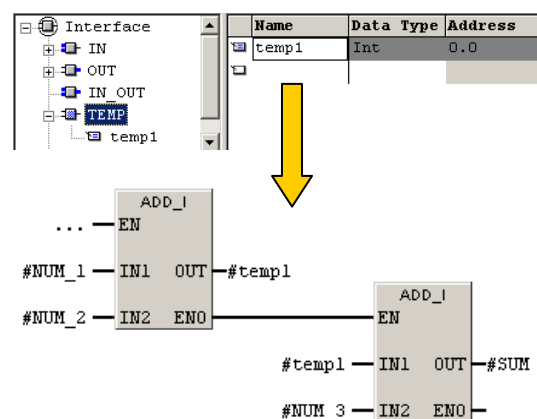
Параметры, объявленные внутри блока, называются *формальными параметрами*. Внутри блока формальные параметры указываются по имени (символьная адресация). Редактор добавляет к имени знак #.

Значения, которые передаются блоку при вызове, называются *фактическими*

*параметрами*. Они могут быть константами или переменными. Таким образом, при каждом вызове блока Вы можете передавать различные значения. Фактический параметр должен относиться к такому же типу данных, что и соответствующий параметр блока.

## Локальные данные

Временные локальные данные используются для промежуточного хранения результатов, получающихся при обработке программы блока. Локальные данные доступны только во время обработки блока. После завершения блока эти данные теряются.



Объявление локальных данных выполняется в разделе "TEMP" интерфейса блока.

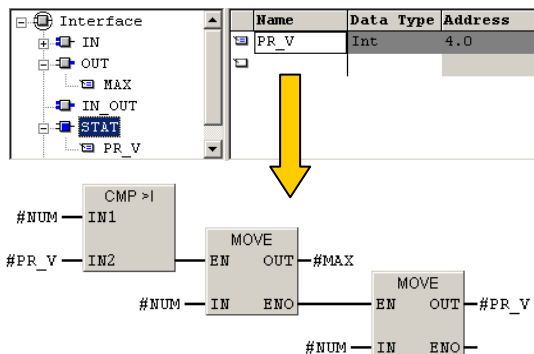
Временные локальные данные сохраняются в L-стеке (в системной памяти CPU) в порядке их объявления в соответствии с типами данных.

Объем локальных данных ограничен размером L-стека (для S7-300 не более 236 байт).

Адресацию временных локальных данных нужно проводить с использованием символьных имен. Допускается использование абсолютных адресов. Например, переменная "temp1" имеет адрес LW0.

## Статические локальные данные

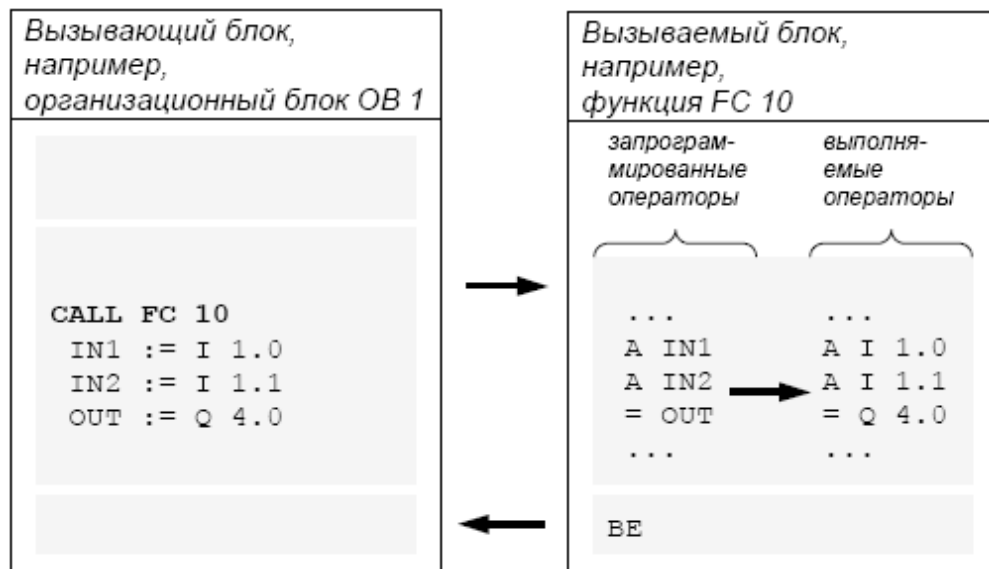
Статические локальные данные можно объявлять только для функциональных блоков.



Статические данные – это "память" функционального блока. Эти данные сохраняются до тех пор, пока программа не изменит их. При использовании статических данных FB требует обязательного использования экземплярного DB. С помощью него организуется хранение статических данных, а также параметров блока. Объем памяти, требуемый для размещения статических данных, ограничивается максимальным размером блока DB (для S7-300 – 16 Kb, S7-400 – 64 Kb).

## Вызов блока: общая информация

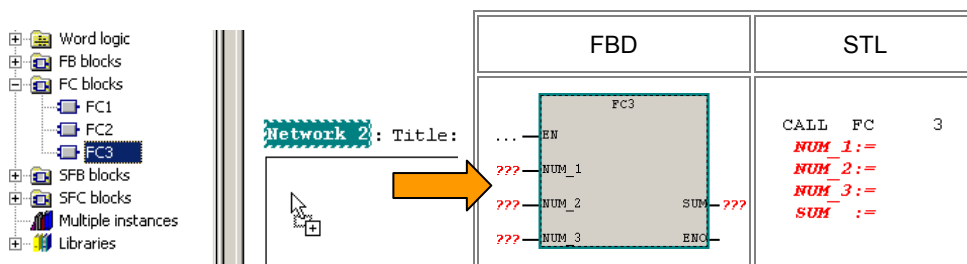
Вызов блока состоит из оператора вызова (в примере, CALL FC 10) и списка параметров. Если вызываемый блок не имеет параметров, инструкция вызова не будет иметь списка параметров.



После выполнения оператора вызова CPU продолжает выполнение программы в вызываемом блоке. Программа блока обрабатывается до оператора окончания блока. По окончании вызванного блока CPU возвращается к выполнению программы в вызывающем блоке; выполнение этой программы продолжается со следующего оператора после оператора вызова блока.

Информация, которая требуется CPU для возврата в вызывающий блок, сохраняется в стеке блоков (B-stack). При каждом вызове блока в B-стеке генерируется новый элемент стека, который содержит адрес возврата, содержимое регистра данных и адрес стека локальных данных вызывающего блока.

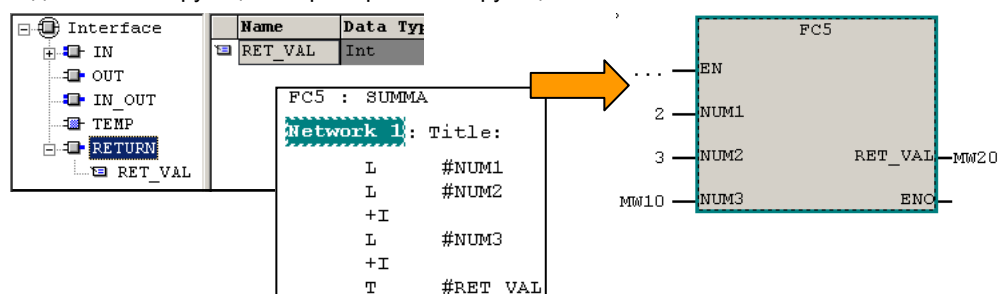
## Вызов FC с параметрами



Для вызова FC Вы можете использовать окно программных элементов редактора LAD/STL/FBD. Выделите блок и буксируйте его в выделенный сегмент (или просто кликните на нем мышью). Т.к. все при вызове функции все параметры должны быть назначены, то редактор отмечает все позиции для редактирования красным цветом. Командой вызова при таком способе является команда безусловного вызова "CALL".

## Присвоение значения функции

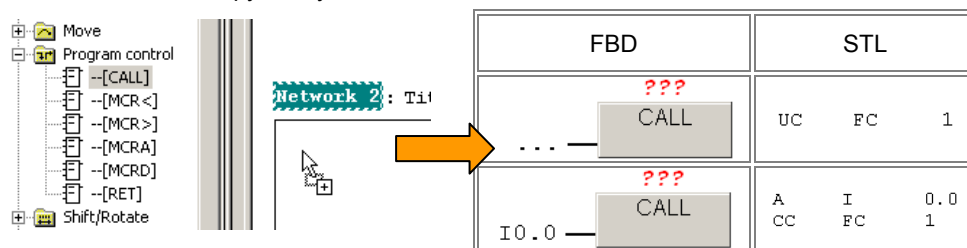
Такой параметр блока как значение функции является особым выходным параметром. Он имеет имя RET\_VAL и определяется как первый выходной параметр. На рисунке приведен пример для объявления функции ("сумматор") с параметром RET\_VAL, кодовая часть функции и пример вызова функции.



## Вызов FC без параметров

Если функция не имеет параметров, то Вы можете вызвать ее 3 способами:

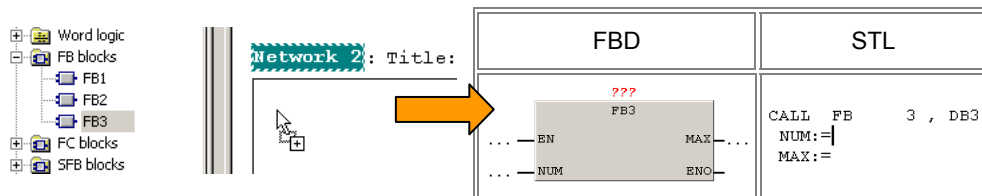
- Описанным выше способом (через инструкцию "CALL")
- С помощью инструкции безусловного вызова "UC"
- С помощью инструкции условного вызова "CC"



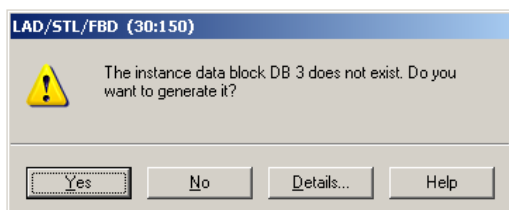
Инструкции UC и CC используются только для вызова блоков FC без параметров. Но редактор не проверяет выполнения этого условия. Оператор вызова UC является безусловным оператором. Оператор вызова CC является условным оператором (блок вызывается, если RLO= "1").  
Примечание: инструкции UC и CC могут использоваться для косвенного вызова блоков FC/FB. Например, команда UC FC[MW10] вызывает блок FC, номер которого вычисляется перед вызовом (в данном случае находится в ячейке MW10).



## Вызов FB с параметрами



При вызове FB с параметрами Вам не обязательно указывать актуальные значения параметров. Необходимым условием является указание блока данных (экземпляр DB). Если параметр ввода не передается или параметр выхода не записывается при вызове FB, сохраняется старая величина в экземпляре DB, которая и будет использоваться (экземпляр DB = память FB).



Если экземпляр DB для блока FB не был создан ранее, то редактор сам предлагает его создать. Одному блоку FB может быть назначено несколько экземплярных DB (для различных вызовов).

Экземплярные блоки данных содержат только те данные, которые заданы в интерфейсной части FB как параметры и/или статические данные.

|   | Address | Declaration | Name | Type | Initial value | Actual value | Comment |
|---|---------|-------------|------|------|---------------|--------------|---------|
| 1 | 0.0     | in          | NUM  | INT  | 0             | 0            |         |
| 2 | 2.0     | out         | MAX  | INT  | 0             | 0            |         |
| 3 | 4.0     | stat        | RR_V | INT  | 0             | 0            |         |

Структуру данных и адреса переменных экземплярного DB можно увидеть в окне утилиты "DB Param". Для этого нужно выделить блок DB в папке "Blocks" соответствующей S7-программы (в SIMATIC Manager) и кликнуть его мышью.

| @Actual value | Actual value |
|---------------|--------------|
| 2             | 4            |
| 2             | 2            |
| 10            | 10           |

Для наблюдения и модификации данных Вы должны перевести окно отображения блока в режим "Data View" (меню View->Data View).

В колонке "@Actual value" Вы наблюдаете за переменными.

В колонке "Actual value" Вы вводите новые значения. Значения, отличающиеся от текущих, указываются с оранжевым фоном.

Вы можете передать в блок DB (а значит и в FB!) новые значения параметров по команде меню PLC->Download Parameter Setting Data или по соответствующей кнопке.



Передать в блок DB весь набор данных из колонки "Actual value" можно по команде меню PLC->Download Entire Block или по соответствующей кнопке.



## Вызов FB без параметров и статических данных

FB без параметров и статических данных обычно не применяются. Для таких целей используют блоки FC.

Тем не менее, если Вы все-таки применяете такие блоки, то вызов их выполняется с помощью инструкций UC и CC (как при вызове FC без параметров). Такой вызов Вы можете организовать только на языке STL.

## Параметры блока EN/ENO

Вход разрешения "Enable" (EN) и выход "Enable output" (ENO) для блоков FC/FB доступны только на языке LAD или FBD. Это искусственные параметры, которые отображает редактор в "прямоугольниках" блоков.

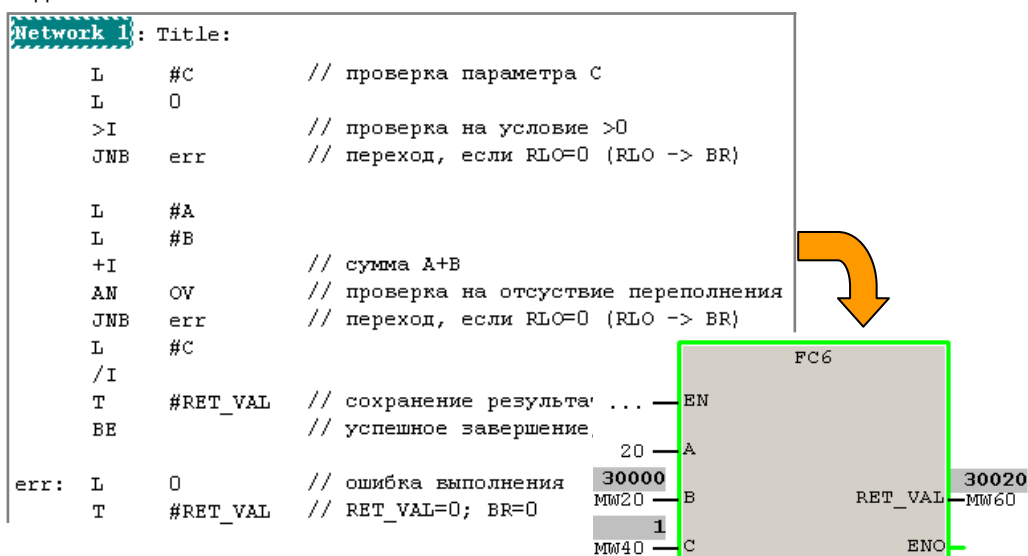
Входной параметр EN используется в качестве логического условия для вызова блока. Если он не задан, то блок вызывается без всяких условий.

Выходной параметр ENO можно рассматривать как логический признак выполнения блока. В действительности, ENO всегда отражает конъюнкцию признака BR слова состояния CPU (Status Word) и состояния входа EN:  $ENO := BR \& EN$ .

Вы можете передать информацию об ошибке выполнения блока в вызывающий блок с помощью установки признака BR в лог. "0".

### Пример программы с формированием признака BR


FC6: INT. Блок вычисляет выражение  $Y = (A+B)/C$ , где A, B, C - числовые параметры типа INT. Если сумма  $(A+B) > 32767$  или  $C \leq 0$ , то значение функции считать недействительным.



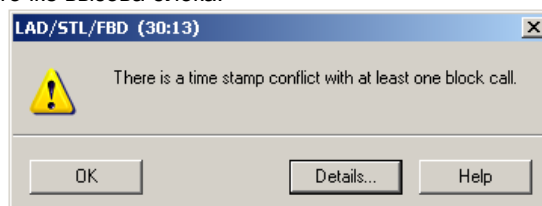
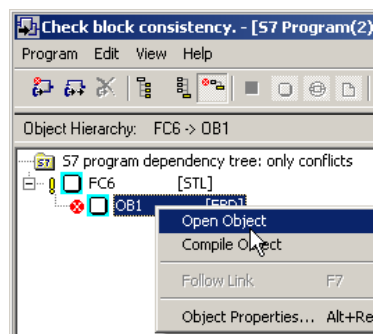
## Корректировка вызовов блока

Если Вы выполнили изменения в интерфейсной части блока, например, вставка новых параметров; удаление параметров; изменение типа параметров; изменение имен параметров или изменение порядка параметров, то в целях предотвращения ошибки Вы должны скорректировать все точки вызова данного блока.

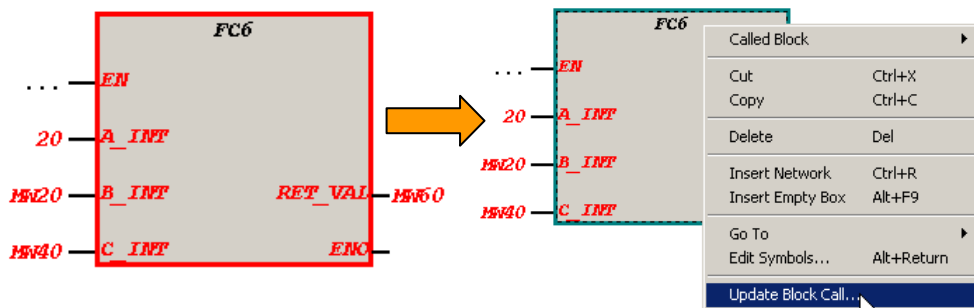
Выполните команду *Edit->Check Block Consistency...*

Блоки, которые необходимо скорректировать, указаны символом .

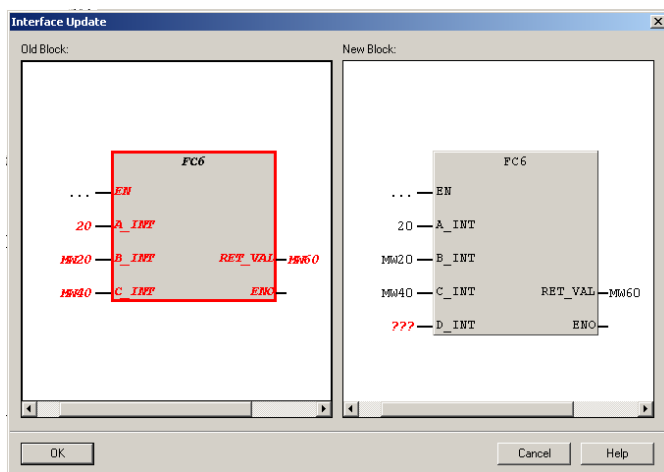
Функция *Open Object* открывает блок. В редакторе появляется информационное сообщение о конфликте в точке вызова блока.



Найдите в окне блока команду вызова блока, у которого был изменен интерфейс. Такой вызов отображается красным цветом. Вы можете использовать команду контекстного меню *Update Block Call* для автоматической корректировки вызова блока.



После выполнения функции *Update Block Call* появляется окно, показывающее блок со старым и с новым интерфейсом.



Подтвердив изменения по кнопке "OK", Вы должны решить вопрос по назначению параметров и сохранить блок.

Необходимо помнить, что при изменении интерфейса блоков FB, последующая корректировка необходима как для вызывающего блока, так и для экземпляра.

### Изменение интерфейса блока FC/FB в программе без остановки CPU

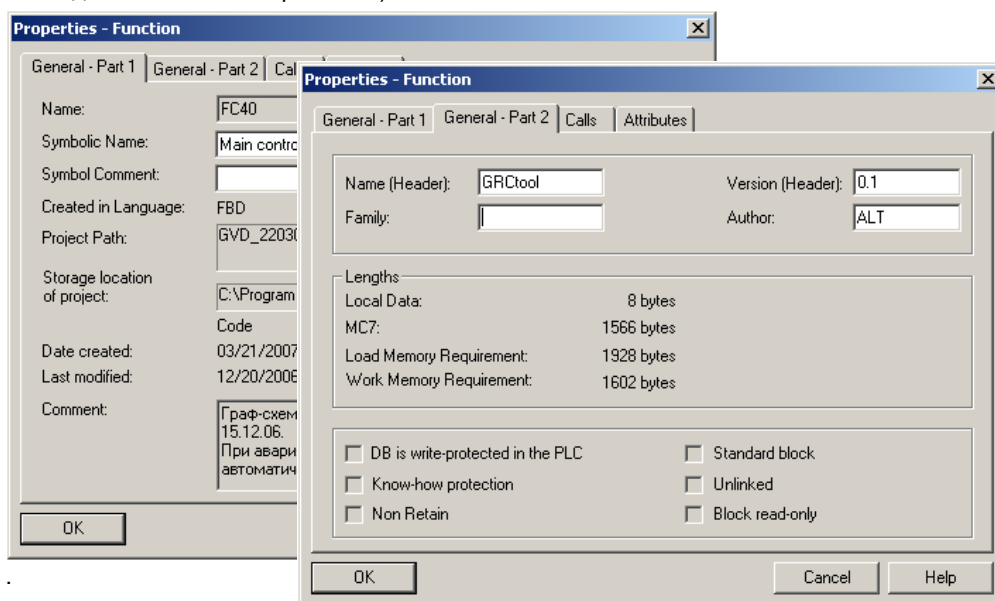
Если необходимо изменить интерфейс FC/FB в исполняемой программе без остановки CPU, то рекомендуется следующая последовательность действий:

1. Создать копию блока с новым номером.
2. Выполнить необходимые изменения в интерфейсе нового блока (копии).
3. Сохранить новый блок в проекте и в CPU.
4. В вызывающем блоке создать новый сегмент с командой вызова нового блока и удалить старый сегмент с командой вызова старого блока.
5. Только для FB ! Сохранить в CPU новый экземпляр DB.
6. Сохранить вызывающий блок в проекте и CPU.

Как можно заметить, описанная процедура основана на замене номеров блоков FC/FB и экземпляра DB. Для возврата к старым номерам (в целях сохранения документации) необходимо удалить старые блоки в проекте, а затем повторить описанную процедуру.

## Свойства блоков

В SIMATIC Manager выделите блок и выполните команду контекстного меню *Object Properties...* (аналогичное действие можно выполнить в редакторе блока с помощью команды меню *File-> Properties...*).

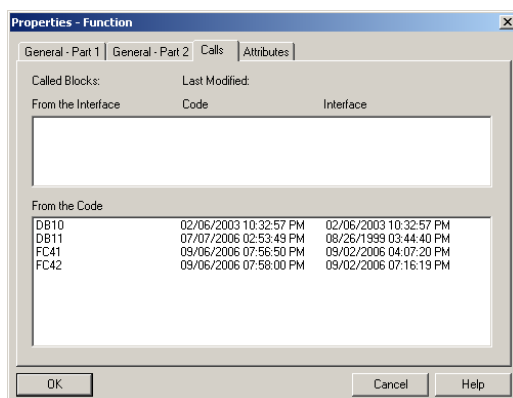


Окно блока “Properties” содержит 4 вкладки.

Вкладка “General – Part1” позволяет задать символьное имя (и комментарий) блока. Здесь можно увидеть информацию о дате создания блока и дате последних изменений.

Вкладка “General – Part2” позволяет увидеть или изменить 4 атрибутных параметра блока, загружаемых в CPU: “Name” (краткое, не более 8 символов, имя блока); “Family” (краткое имя для группы, к которой принадлежит блок); “Author” (краткое обозначения автора блока); “Version” (версия блока от 0.0 до 15.15).

Для справки приведена информация по занимаемому объему в загрузочной и рабочей памяти, а также “MC7” (размер кодовой части) и “Local Data” (размер локального стека). Атрибутами, расположенными в нижней части окна, Вы не можете здесь управлять. Для кодовых блоков (OB, FC, FB) с целью защиты от несанкционированного просмотра пользователь может установить атрибут “Know-how protection” (порядок установки этого атрибута см. в следующем разделе). Атрибут “Standard block” относится к блокам, включенным в библиотеки STEP7. Все остальные атрибуты относятся к блокам данных.



Вкладка “Calls” содержит информацию о блоках, вызываемых из данного блока непосредственно (“From the Code”) или как мультитекст ( “From the Interface”).

Примечание: концепция мультитекстов рассмотрена в разделе “Системные функции” (стр. 58).

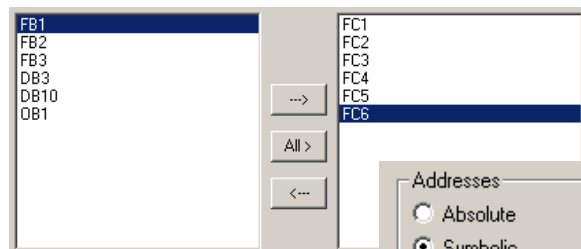
Вкладка “Attributes” позволяет увидеть/изменить системные атрибуты блока, которые используются при работе с дополнительными программными пакетами, такими как CFC, S7-GRAPH, S7-PDIAG, WinCC.

## Создание исходных файлов

Вы можете сгенерировать исходный файл STL из существующих блоков, который можно затем редактировать с помощью любого текстового редактора. Исходный файл размещается в папке "Source" S7-программы.

Для того чтобы сгенерировать исходный файл из блока, выполните следующее:

1. В программном редакторе выберите команду меню *File-> Generate Source File*.
2. В диалоговом окне выберите папку исходного файла, в которой Вы хотите сохранить новый исходный файл.
3. Введите имя для исходного файла в текстовом окне "Object name".



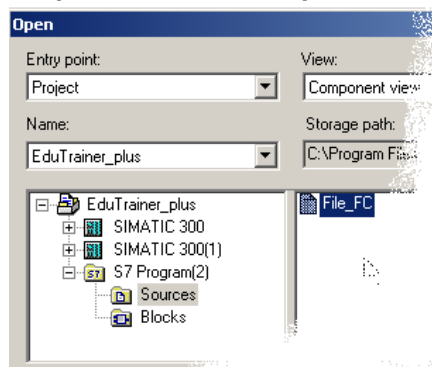
4. В диалоговом окне "Generate Source ...", выберите блок(и), по которым Вы желаете сгенерировать исходный файл. Выбранные блоки отображаются в правом списке.

В поле "Addresses" выберите, какая адресация (символьная или абсолютная) будет использоваться при создании исходного файла.

Примечание: символьная адресация удобна для автоматической замены адресов во всей программе (в соответствии с новой символьной таблицей).

5. Нажмите "OK". Из выбранных блоков создается один непрерывный исходный файл на языке STL.

## Открытие исходного файла



Открыть созданный исходный файл возможно из LAD/STL/FBD редактора по команде *File->Open...*

В появившемся окне необходимо выбрать проект, папку "Source", а затем и сам исходный файл.

Вы можете открыть исходный файл также из SIMATIC Manager (аналогично S7-блокам).

Текст исходного файла появляется в отдельном окне LAD/STL/FBD редактора.

Вы можете скопировать данный текст и перенести его в любой текстовый редактор.

## Пример исходного текста на STL

Справа приведен пример для блока FC5. Как любой кодовый блок, он состоит из таких разделов как: описание блока, включая атрибуты (1); описание параметров (2); кодовая часть (3).

```
FUNCTION FC 5 : INT
TITLE =SUMMA ①
VERSION : 0.1

VAR_INPUT
  NUM1 : INT ;
  NUM2 : INT ;
  NUM3 : INT ; ②
END_VAR
```

```
BEGIN
NETWORK ③
TITLE =

  L #NUM1;
  L #NUM2;
  +I ;
  L #NUM3;
  +I ;
  T #RET_VAL;
END FUNCTION
```

## Импорт исходных файлов

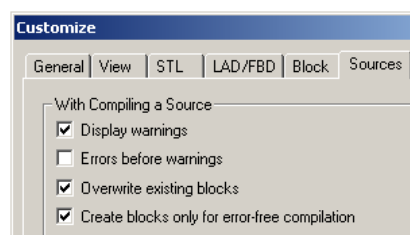
Для того чтобы импортировать исходный файл из любого каталога в проект:

1. В SIMATIC Manager, выберите папку исходного файла, в которую Вы хотите импортировать исходный файл.
2. Выберите команду меню *Insert New Object -> External Source File...*
3. В появившемся окне выберите определенный каталог и импортируйте исходный файл.
4. Нажмите кнопку "Открыть".

## Компиляция исходного файла



В результате компиляции заново создаются блоки, включенные в текст исходного файла программы. Они записываются в папку "Blocks", принадлежащей той же самой S7-программе, что и сам исходный файл.



Вы можете изменить настройки компиляции в LAD/STL/FBD редакторе через меню *Options -> Customize -> Sources*.

Настройки по умолчанию указывают, что новые блоки записываются "поверх" существующих, причем только те, которые не содержат ошибок.

## Последовательность действий в редакторе

1. Закройте в редакторе все блоки, которые включены в исходный файл.
2. Откройте исходный файл, который Вы хотите скомпилировать. Он должен находиться в папке исходных файлов S7 программы, в которой должны быть сохранены скомпилированные блоки.
3. Выберите команду меню *File -> Compile* или соответствующую кнопку в панели инструментов.
4. Нижнее окно "Details" покажет количество синтаксических ошибок (Errors) и предупреждений (Warnings). Каждая ошибка отображается в отдельной строке. Если дважды кликнуть на такой строке, то в исходном файле выделяется строка, содержащая ошибку.

## Установка защиты на блок



```
FUNCTION FC 5 : INT
TITLE =SUMMA
KNOW_HOW_PROTECT
VERSION : 0.1
```

```
VAR_INPUT
  NUM1 : INT ;
```

Если Вы хотите, чтобы редактор не отображал кодовую часть блока, Вам необходимо установить для блока атрибут "Know-how protection". Тогда в редакторе можно будет увидеть только раздел формальных параметров блока.

Атрибут "Know-how protection" может быть установлен только в исходном файле блока в виде строки KNOW\_HOW\_PROTECT, причем она должна стоять первой в списке остальных атрибутов.

Чтобы снять защиту, необходимо удалить в исходном файле блока строку KNOW\_HOW\_PROTECT и выполнить компиляцию блока.

## Блоки данных

Блоки данных содержат переменные (например, числовые величины), которые используются в программе пользователя. В отличие от логических блоков (OB, FC, FB), блоки DB не содержат команд STEP 7.

Существуют блоки трех видов: блоки глобальных данных (*Global DB*), экземплярные блоки (*Instance DB*), и блоки с привязкой к пользовательскому типу данных UDT (*DB of type*).

Блоки "Global DB" и "DB of type" может использовать любой логический блок. Разница между ними в том, что для блоков глобальных данных Вы сами определяете объем и структуру данных, а блоки "DB of type" повторяют структуру UDT непосредственно при программировании DB.

Блоки "Instance DB" данных содержат только те данные, которые использует связанный функциональный блок. Этот функциональный блок определяет структуру данных и их адреса.

Число и размер блоков данных определяются типом CPU. Например, для CPU314 размер DB – 16Kbyte, а максимальное количество – 511 (нумерация от 1 до 511); для CPU414 размер DB - 64 Kbyte, а максимальное количество – 3000.

Online функция PLC-> Module Information -> Performance Data сообщает сведения о количестве и длине блоков, которые Вы можете записать в CPU.

| Address Areas: |              |           |                  |
|----------------|--------------|-----------|------------------|
| Address type   | Quantity     | Area from | to / max. length |
| Timers         | 256          | T0        | T255             |
| Counters       | 256          | C0        | C255             |
| Local Data     | 3072 (Bytes) |           |                  |
| OB             | 16           |           | 16420 (Bytes)    |
| FC             | 2048         |           | 16420 (Bytes)    |
| FB             | 2048         |           | 16420 (Bytes)    |
| DB             | 511          |           | 16420 (Bytes)    |

## Создание DB

**Properties - Data Block**

General - Part 1 | General - Part 2 | Calls | Attributes

Name and type: DB2 Shared DB

Symbolic Name: Shared DB

Symbol Comment: Instance DB

DB of type

В SIMATIC Manager после выполнения функции *Insert New Object -> Data Block* Вы должны ввести номер блока его тип. В LAD/STL/FBD редакторе Вы также можете создать новый DB (*File->New*).

Для доступа к элементам блока данных необходимо присвоить имя самому блоку.

| Address | Name       | Type   | Initial | Comment  |
|---------|------------|--------|---------|----------|
| 0.0     |            | STRUCT |         |          |
| +0.0    | Seamer_Ref | DINT   | I#0     | Value on |
| +4.0    | Seamer_rol | DINT   | I#0     | Destinat |
| +8.0    | Seamer_des | DINT   | I#360   | Destinat |

При объявлении элементов глобального DB необходимо задать имя переменной (не более 24 символов) и тип данных. Также можно ввести начальное значение и комментарий.

## Атрибуты блоков DB

☐ DB is write-protected in the PLC

☐ Know-how protection

☐ Non Retain

☐ Standard block

☐ Unlinked

☐ Block read-only

Атрибуты блока DB Вы можете установить в окне свойств блока.

Атрибут "Know-how protection" можно установить только через исходный файл DB (аналогично кодовым блокам).





Атрибут "Data block is write-protected in the PLC" устанавливает на блок защиту от записи. В этом случае, данные блока не могут быть изменены во время выполнения программы. Вы сможете только считывать данные из этого DB. Данный атрибут полезен, например, для таблиц преобразования данных, размещенных в DB.

Блоки DB с атрибутом "Unlinked" передаются только в загрузочной памяти CPU. Для работы с таким блоком Вы должны перенести его в рабочую память из пользовательской программы (например, с помощью системной функции SFC20). Данный атрибут может использоваться, например, для однотипных наборов данных ("рецептур"), которые организованы в отдельных блоках DB и для размещения которых не достаточно места в рабочей памяти.

Атрибут "Non-Retain" эффективен тех CPU, которые поддерживают концепцию сохранения блоков DB. Блоки с таким атрибутом теряют свои текущие значения при выключении питания на CPU.

## Два регистра блоков данных

В CPU есть два регистра для адресации к блокам данных: DB и DI- регистры. DB-регистр ("Global data block register") используется для доступа к блокам глобальных данных. DI-регистр ("Instance data block register") используется преимущественно для доступа к экземплярным блокам данных.

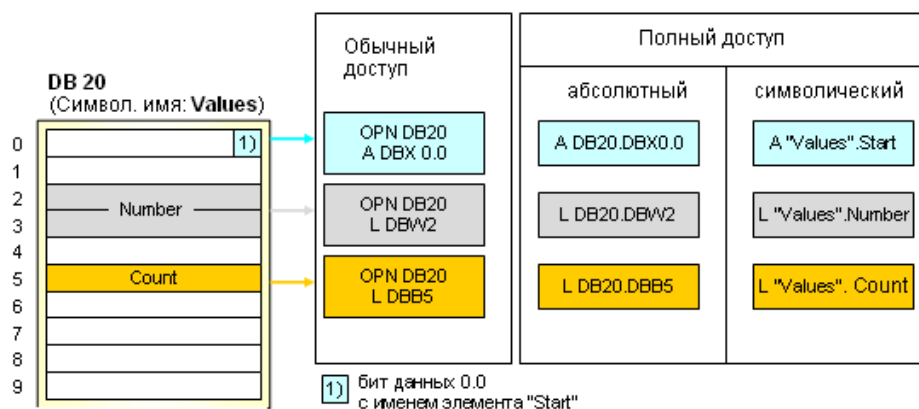
## Адресация данных

При загрузке в аккумулятор, например, слова данных, Вы должны указать с помощью какого регистра, DB или DI, открыт доступ к блоку DB (в него должен быть записан номер DB). Также как к области M или L, к ячейкам DB можно обратиться побитно, или как к байту, слову, двойному слову.

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

y – адрес байта; x – адрес бита

## Пример доступа к ячейкам блока данных



Адресация с использованием регистра DI не допускает режима полного доступа!



### Доступ к элементам данных с указанием полного адреса

При доступе к элементам блоков данных с указанием полного адреса Вы должны задавать адрес данных совместно с номером блока DB. Такой способ указания адреса может быть использован как с символьными, так и с абсолютными адресами.

**Network 3:** Title: "MOTOR" является символьным адресом блока DB50. SPEED\_ACT - это имя переменной, имеющей адрес DBD22.

|     |                   |            |
|-----|-------------------|------------|
| L   | "MOTOR".SPEED_ACT | DB50.DBD22 |
| T   | MD                | 30         |
| NOP |                   | 0          |

При выполнении команды с полной адресацией элементов блока данных сначала открывается блок данных с помощью DB-регистра (в DB-регистр заносится номер DB), а затем выполняется операция доступа к адресам данных.

Только полная адресация к данным дает возможность для редактора отображать данные в символьной адресации. В результате:

- Уменьшаются ошибки программирования (редактор сравнивает тэги, использованные в таблице символов и в программе)
- Более простым становится чтение и понимание программы
- Если символьная адресация имеет приоритет, то при добавлении (или удалении) элементов блока данных Вам не нужно заботиться об изменении адресов в программе.

### Доступ к элементам данных с указанием неполного адреса

Неполный адрес элементов данных – это адрес (точнее смещение от начала блока), который они имеют в блоке данных. Он отображается в колонке "Address".

| Address | Name            | Type |
|---------|-----------------|------|
| 0.0     | Rec1.GUN1_OP_SP | BOOL |
| 0.1     | Rec1.GUN2_OP_SP | BOOL |
| 0.2     | Rec1.GUN3_OP_SP | BOOL |

Для сложных типов данных, например, массив или структура, адрес компонентов отображается в режиме *View->Data View*. Серый фон таблицы означает, что в этом режиме состав и тип данных нельзя изменить.

Адресация с неполным доступом может быть выполнена с использованием регистров DB и DI. Перед доступом в регистры DB должен быть открыт.

### Открытие блока DB

Процедура открытия DB всегда предполагает запись номера DB в регистр DB или DI. Открываемый блок должен быть в рабочей памяти.

Как отмечено выше, в регистр DB всегда заносится номер блока при использовании полной адресации к элементам данных.

Существуют также специальные команды для открытия блоков данных: OPN DBx и OPN DIx, где x- номер DB.

### Примеры открытия DB:

**Network 4:** Пример открытия блоков данных

|     |                   |   |
|-----|-------------------|---|
| L   | "MOTOR".SPEED_ACT | // открытие совмещено с доступом (через DB-регистр) |
| OPN | "MOTOR"           | // открытие блока по имени (через DB-регистр)       |
| OPN | DI 50             | // открытие блока по номеру (через DI-регистр)      |

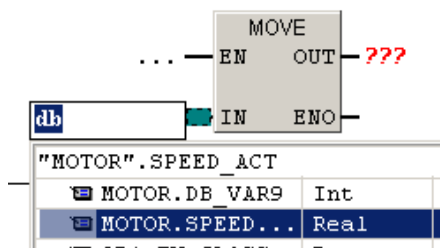
### Пример сравнения двух переменных из разных блоков

**Network 5:** Сравнение текущей скорости с заданной

|     |                   |            |
|-----|-------------------|------------|
| L   | "MOTOR".SPEED_ACT | DB50.DBD22 |
| L   | "REC".MODE_1      | DB2.DB12   |
| >=R |                   |            |

После выполнения данного фрагмента в регистре DB останется значение 2.

## Ввод элементов данных на LAD и FBD



Если установлен язык LAD или FBD, то ввод операндов из блоков DB, имеющих символьные имена выполняется очень удобно. Сразу после ввода символов "DB" или первой буквы имени блока появится выпадающее меню для дальнейшего выбора операнда.

## Размер блока данных и его номер

В STEP7 имеются специальные команды, позволяющие записать в ACCU1 номер открытых блоков данных:

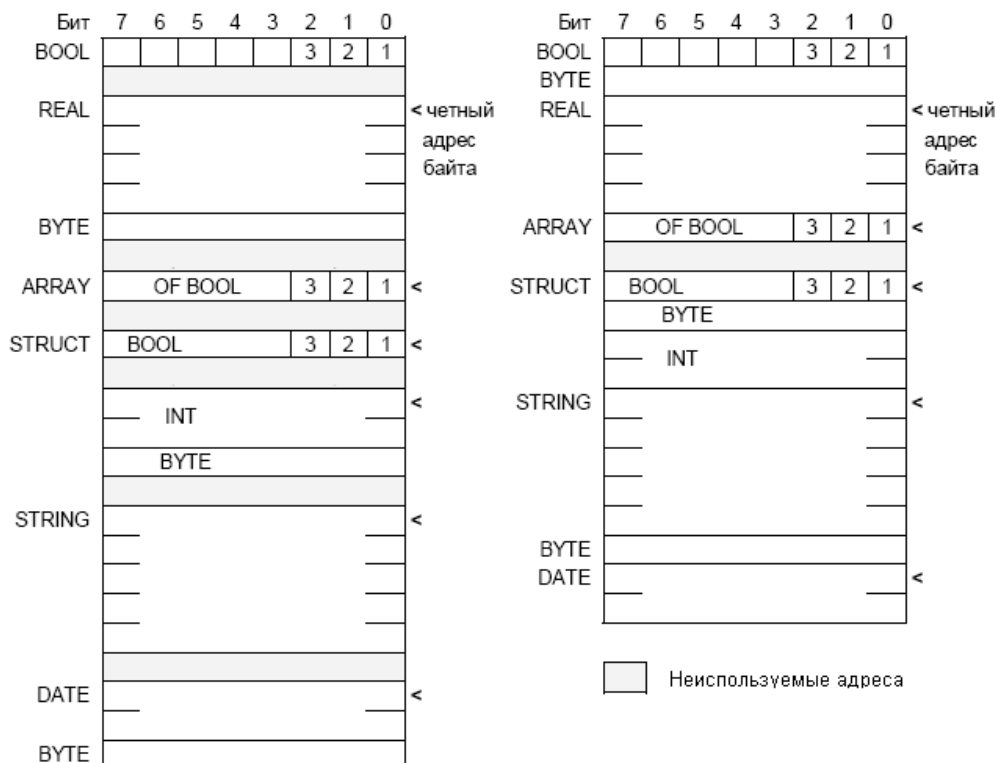
- **L DBNO** – чтение регистра DB
- **L DINO** – чтение регистра DI,

а также узнать размер в байтах блока данных:

- **L DBLG** – загрузка длины блока, открытого через DB-регистр
- **L DILG** – загрузка длины блока, открытого через DI-регистр.

## Оптимизация размера блока данных

Группируя отдельные битовые переменные и komponуя байтовые переменные, Вы можете оптимизировать размещение данных в блоках данных.



Переменные, имеющие размер одного слова и имеющие размер двойного слова, всегда размещаются, начиная с границы слова, то есть в байте с четным адресом. Любые данные со сложными типами данных, например, структура или массив, также “выравниваются” по четным адресам.

### Наблюдение и управление переменными блока данных

Для наблюдения и управления переменными в CPU используется режим отображения блоков данных Data View (*View->Data View*). При вызове функции *Debug ->Monitor* режим Data View установится автоматически.

| Address | Name      | Type | Initial value | Actual value | Comment |
|---------|-----------|------|---------------|--------------|---------|
| 0.0     | Seamer_Rd | DINT | L#0           | L#230        | Value o |
| 4.0     | Seamer_rd | DINT | L#0           | L#245        | Destina |
| 8.0     | Seamer_ds | DINT | L#360         | L#320        | Destina |

Actual value - текущее значение переменной.



Initial value - начальное значение, которое вводится при объявлении переменных.

Вы можете установить два различных значения для переменной из блока данных. В программе пользователя всегда используется только “Actual value”.

Когда блок данных создается в первый раз, текущие и начальные значения всегда совпадают.



При выключении CPU текущие значения блоков данных сохраняются в загрузочной либо энергонезависимой памяти, (кроме блоков с атрибутом “Not Retain”).

Для управления переменными из блока данных в CPU необходимо:

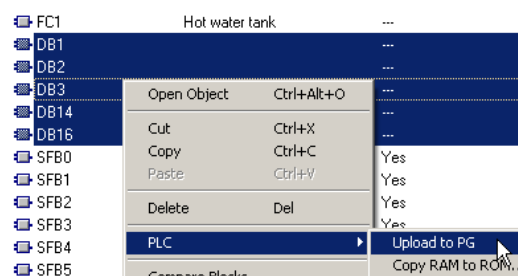
1. Открыть блок в режиме online (кнопка .
2. Изменить значение переменной в колонке “Actual value”.
3. Выполнить команду *PLC ->Download* или нажать кнопку .

### Сохранение текущих значений блока данных в проекте

Для сохранения текущих данных блока DB в проекте необходимо:

1. Открыть блок в режиме online (кнопка .
2. Сохранить блок по команде *File ->Save* или нажать кнопку .

Для обновления нескольких блоков данных в проекте рекомендуется следующий порядок:



1. Открыть проект в режиме online.
2. В папке Blocks вашей программы выделить блоки данных.
3. Выполнить функцию *PLC ->Upload to PG*.

### Сброс данных в начальные значения

Чтобы всем переменным блока данных вновь присвоить начальные значения необходимо:

1. Открыть блок в режиме Data View (*View->Data View*).
2. Выполнить команду меню *Edit > Initialize Data Block*.
3. Сохранить блок в CPU (*PLC ->Download*) и в проекте (*File ->Save*).



## Отображение экземплярных DB

В отображении экземплярного блока (в столбце "Declaration") Вы можете увидеть, как были заданы переменные в таблице описания функционального блока (параметры IN, OUT, IN\_OUT и статические данные STAT).

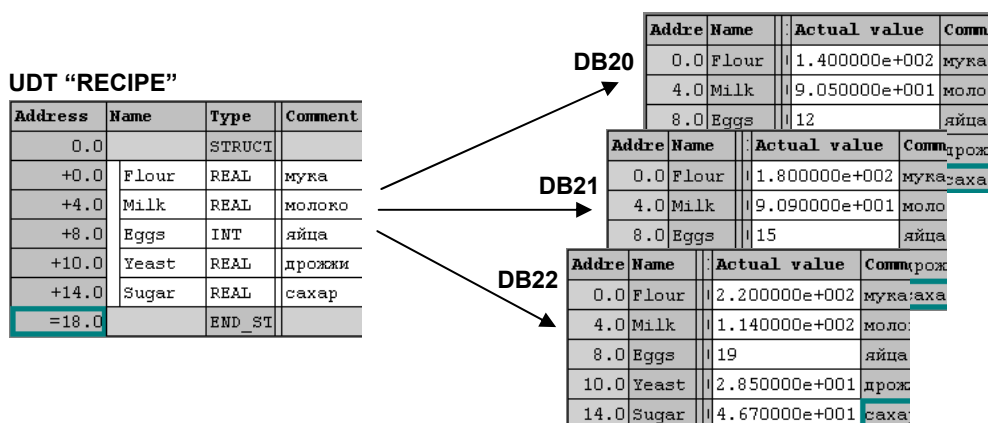
| Address | Declaration | Name | Type | Initial  | Actual v |
|---------|-------------|------|------|----------|----------|
| 0.0     | in          | NUM  | REAL | 0.000000 | 4.000000 |
| 4.0     | out         | MAX  | REAL | 0.000000 | 6.500000 |
| 8.0     | stat        | RR_V | REAL | 0.000000 | 6.500000 |

Вы можете редактировать только текущие значения переменных (в режиме Data View).

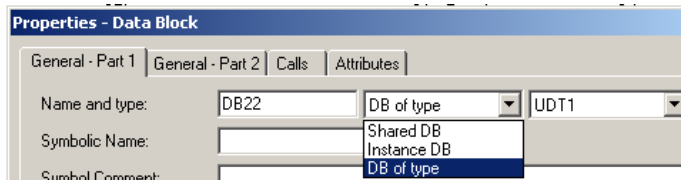
## Блоки DB, связанные с типом данных UDT

Типы данных UDT (User Data Type) - это специальные структуры данных, создаваемые пользователем. Они используются для облегчения программирования при работе с данными. UDT не загружаются в CPU.

UDT могут использоваться как шаблон для создания блоков данных с одинаковой структурой данных.



Например, Вы можете создать в папке "Blocks" новый UDT с именем "RECIPE" (команда *Insert New Object -> Data Type*). После редактирования UDT сохраните его.



Вы легко можете создать требуемое количество блоков DB (рецептов), повторяющих структуру UDT.

При изменении UDT Вы должны создать заново блоки данных, ссылающиеся на UDT. Эти изменения удобно проводить через исходную программу.

## Использование UDT при объявлении переменных

| Address | Name   | Type        |
|---------|--------|-------------|
| 0.0     |        | STRUCT      |
| +0.0    | recipe | ARRAY[1..5] |
| *20.0   |        | "RECIPE"    |
| =100.0  |        | END_STRUCT  |

Если Вам понадобится одна и та же структура несколько раз в одном и том же блоке данных, Вы можете использовать UDT, как тип данных для элементов массива.

Тогда команда **L "Cake".Recipe[2].Eggs** означает доступ к компоненту "Eggs" 2-го рецепта из блока данных с именем "Cake".

## Организационные блоки

Организационные блоки (ОВ) вызываются операционной системой CPU и управляют либо циклическим исполнением программы, либо являются обработчиками прерываний. Имеются также стартовые блоки, которые выполняются однократно только при запуске CPU.

|             |                   |
|-------------|-------------------|
| General     | Diagnostic Buffer |
| Time System | Performance Data  |

Organization Blocks:

| No.  | Function                                  |  |
|------|---|--|
| OB1  | Free scan cycle - start event: sta...     |  |
| OB10 | Time-of-day interrupt - start eve...      |  |
| OB20 | Time-delay interrupt - start event: ti... |  |
| OB35 | Cyclic interrupt - default clock...       |  |
| OB40 | Hardware interrupt - start event...       |  |
| OB80 | Timing error                              |  |
| OB82 | Diagnostic interrupt                      |  |
| OB85 | Program execution error                   |  |

Состав блоков ОВ зависит от типа в CPU. С помощью online- функции *Module Information - >Performance Data* Вы всегда можете увидеть список доступных для работы блоков ОВ.

Номер ОВ определяет его функцию. Когда Вы добавляете в программу новый ОВ (*Insert New Object ->Organization Block*), то Вы должны ввести тот номер блока, который означает требуемую для программы функцию CPU (событие-прерывание).

## Типы ОВ

Выделяют следующие основные типы ОВ:

- Запуска (ОВ 100, ОВ101, ОВ102)
- Основного цикла (ОВ1)
- Периодического выполнения: циклического прерывания (ОВ30.. ОВ38), прерывания по времени суток (ОВ10..ОВ17)
- Программных прерываний (ОВ20..ОВ23)
- Аппаратных прерываний (ОВ40..ОВ47)
- Ошибок: асинхронных (ОВ80..ОВ87), синхронных (ОВ121,122)

В S7-400 выделяют еще дополнительные типы ОВ:

- Мультипроцессорной обработки (ОВ60)
- Фоновой обработки (ОВ90)

## Приоритет организационного блока

Исполнение ОВ может быть прервано вызовом другого ОВ. Какому ОВ разрешается прервать другой ОВ, зависит от его приоритета или приоритетного класса (от 1 до 28). Самый низкий приоритет (1) имеет ОВ1. Блоки ОВ с более высоким приоритетом могут прерывать ОВ с более низким приоритетом. Приоритет ОВ в S7-300 нельзя изменить, а в S7-400 можно (для ОВ10 - ОВ47) при параметрировании CPU.

## Стартовая информация, временные локальные данные

| Name             | Data Type | Address | Comment    |
|------------------|-----------|---------|------------|
| OB1_EV_CLASS     | Byte      | 0.0     | Bits 0-3 = |
| OB1_SCAN_1       | Byte      | 1.0     | 1 (Cold re |
| OB1_PRIORITY     | Byte      | 2.0     | Priority o |
| OB1_OB_NUMBR     | Byte      | 3.0     | 1 (Organiz |
| OB1_RESERVED_1   | Byte      | 4.0     | Reserved   |
| OB1_RESERVED_2   | Byte      | 5.0     | Reserved   |
| OB1_RESERVED_3   | Byte      | 6.0     | Reserved   |
| OB1_RESERVED_4   | Byte      | 7.0     | Reserved   |
| OB1_RESERVED_5   | Byte      | 8.0     | Reserved   |
| OB1_RESERVED_6   | Byte      | 9.0     | Reserved   |
| OB1_RESERVED_7   | Byte      | 10.0    | Reserved   |
| OB1_RESERVED_8   | Byte      | 11.0    | Reserved   |
| OB1_RESERVED_9   | Byte      | 12.0    | Reserved   |
| OB1_RESERVED_10  | Byte      | 13.0    | Reserved   |
| OB1_RESERVED_11  | Byte      | 14.0    | Reserved   |
| OB1_RESERVED_12  | Byte      | 15.0    | Reserved   |
| OB1_RESERVED_13  | Byte      | 16.0    | Reserved   |
| OB1_RESERVED_14  | Byte      | 17.0    | Reserved   |
| OB1_RESERVED_15  | Byte      | 18.0    | Reserved   |
| OB1_RESERVED_16  | Byte      | 19.0    | Reserved   |
| OB1_RESERVED_17  | Byte      | 20.0    | Reserved   |
| OB1_RESERVED_18  | Byte      | 21.0    | Reserved   |
| OB1_RESERVED_19  | Byte      | 22.0    | Reserved   |
| OB1_RESERVED_20  | Byte      | 23.0    | Reserved   |
| OB1_RESERVED_21  | Byte      | 24.0    | Reserved   |
| OB1_RESERVED_22  | Byte      | 25.0    | Reserved   |
| OB1_RESERVED_23  | Byte      | 26.0    | Reserved   |
| OB1_RESERVED_24  | Byte      | 27.0    | Reserved   |
| OB1_RESERVED_25  | Byte      | 28.0    | Reserved   |
| OB1_RESERVED_26  | Byte      | 29.0    | Reserved   |
| OB1_RESERVED_27  | Byte      | 30.0    | Reserved   |
| OB1_RESERVED_28  | Byte      | 31.0    | Reserved   |
| OB1_RESERVED_29  | Byte      | 32.0    | Reserved   |
| OB1_RESERVED_30  | Byte      | 33.0    | Reserved   |
| OB1_RESERVED_31  | Byte      | 34.0    | Reserved   |
| OB1_RESERVED_32  | Byte      | 35.0    | Reserved   |
| OB1_RESERVED_33  | Byte      | 36.0    | Reserved   |
| OB1_RESERVED_34  | Byte      | 37.0    | Reserved   |
| OB1_RESERVED_35  | Byte      | 38.0    | Reserved   |
| OB1_RESERVED_36  | Byte      | 39.0    | Reserved   |
| OB1_RESERVED_37  | Byte      | 40.0    | Reserved   |
| OB1_RESERVED_38  | Byte      | 41.0    | Reserved   |
| OB1_RESERVED_39  | Byte      | 42.0    | Reserved   |
| OB1_RESERVED_40  | Byte      | 43.0    | Reserved   |
| OB1_RESERVED_41  | Byte      | 44.0    | Reserved   |
| OB1_RESERVED_42  | Byte      | 45.0    | Reserved   |
| OB1_RESERVED_43  | Byte      | 46.0    | Reserved   |
| OB1_RESERVED_44  | Byte      | 47.0    | Reserved   |
| OB1_RESERVED_45  | Byte      | 48.0    | Reserved   |
| OB1_RESERVED_46  | Byte      | 49.0    | Reserved   |
| OB1_RESERVED_47  | Byte      | 50.0    | Reserved   |
| OB1_RESERVED_48  | Byte      | 51.0    | Reserved   |
| OB1_RESERVED_49  | Byte      | 52.0    | Reserved   |
| OB1_RESERVED_50  | Byte      | 53.0    | Reserved   |
| OB1_RESERVED_51  | Byte      | 54.0    | Reserved   |
| OB1_RESERVED_52  | Byte      | 55.0    | Reserved   |
| OB1_RESERVED_53  | Byte      | 56.0    | Reserved   |
| OB1_RESERVED_54  | Byte      | 57.0    | Reserved   |
| OB1_RESERVED_55  | Byte      | 58.0    | Reserved   |
| OB1_RESERVED_56  | Byte      | 59.0    | Reserved   |
| OB1_RESERVED_57  | Byte      | 60.0    | Reserved   |
| OB1_RESERVED_58  | Byte      | 61.0    | Reserved   |
| OB1_RESERVED_59  | Byte      | 62.0    | Reserved   |
| OB1_RESERVED_60  | Byte      | 63.0    | Reserved   |
| OB1_RESERVED_61  | Byte      | 64.0    | Reserved   |
| OB1_RESERVED_62  | Byte      | 65.0    | Reserved   |
| OB1_RESERVED_63  | Byte      | 66.0    | Reserved   |
| OB1_RESERVED_64  | Byte      | 67.0    | Reserved   |
| OB1_RESERVED_65  | Byte      | 68.0    | Reserved   |
| OB1_RESERVED_66  | Byte      | 69.0    | Reserved   |
| OB1_RESERVED_67  | Byte      | 70.0    | Reserved   |
| OB1_RESERVED_68  | Byte      | 71.0    | Reserved   |
| OB1_RESERVED_69  | Byte      | 72.0    | Reserved   |
| OB1_RESERVED_70  | Byte      | 73.0    | Reserved   |
| OB1_RESERVED_71  | Byte      | 74.0    | Reserved   |
| OB1_RESERVED_72  | Byte      | 75.0    | Reserved   |
| OB1_RESERVED_73  | Byte      | 76.0    | Reserved   |
| OB1_RESERVED_74  | Byte      | 77.0    | Reserved   |
| OB1_RESERVED_75  | Byte      | 78.0    | Reserved   |
| OB1_RESERVED_76  | Byte      | 79.0    | Reserved   |
| OB1_RESERVED_77  | Byte      | 80.0    | Reserved   |
| OB1_RESERVED_78  | Byte      | 81.0    | Reserved   |
| OB1_RESERVED_79  | Byte      | 82.0    | Reserved   |
| OB1_RESERVED_80  | Byte      | 83.0    | Reserved   |
| OB1_RESERVED_81  | Byte      | 84.0    | Reserved   |
| OB1_RESERVED_82  | Byte      | 85.0    | Reserved   |
| OB1_RESERVED_83  | Byte      | 86.0    | Reserved   |
| OB1_RESERVED_84  | Byte      | 87.0    | Reserved   |
| OB1_RESERVED_85  | Byte      | 88.0    | Reserved   |
| OB1_RESERVED_86  | Byte      | 89.0    | Reserved   |
| OB1_RESERVED_87  | Byte      | 90.0    | Reserved   |
| OB1_RESERVED_88  | Byte      | 91.0    | Reserved   |
| OB1_RESERVED_89  | Byte      | 92.0    | Reserved   |
| OB1_RESERVED_90  | Byte      | 93.0    | Reserved   |
| OB1_RESERVED_91  | Byte      | 94.0    | Reserved   |
| OB1_RESERVED_92  | Byte      | 95.0    | Reserved   |
| OB1_RESERVED_93  | Byte      | 96.0    | Reserved   |
| OB1_RESERVED_94  | Byte      | 97.0    | Reserved   |
| OB1_RESERVED_95  | Byte      | 98.0    | Reserved   |
| OB1_RESERVED_96  | Byte      | 99.0    | Reserved   |
| OB1_RESERVED_97  | Byte      | 100.0   | Reserved   |
| OB1_RESERVED_98  | Byte      | 101.0   | Reserved   |
| OB1_RESERVED_99  | Byte      | 102.0   | Reserved   |
| OB1_RESERVED_100 | Byte      | 103.0   | Reserved   |
| OB1_RESERVED_101 | Byte      | 104.0   | Reserved   |
| OB1_RESERVED_102 | Byte      | 105.0   | Reserved   |
| OB1_RESERVED_103 | Byte      | 106.0   | Reserved   |
| OB1_RESERVED_104 | Byte      | 107.0   | Reserved   |
| OB1_RESERVED_105 | Byte      | 108.0   | Reserved   |
| OB1_RESERVED_106 | Byte      | 109.0   | Reserved   |
| OB1_RESERVED_107 | Byte      | 110.0   | Reserved   |
| OB1_RESERVED_108 | Byte      | 111.0   | Reserved   |
| OB1_RESERVED_109 | Byte      | 112.0   | Reserved   |
| OB1_RESERVED_110 | Byte      | 113.0   | Reserved   |
| OB1_RESERVED_111 | Byte      | 114.0   | Reserved   |
| OB1_RESERVED_112 | Byte      | 115.0   | Reserved   |
| OB1_RESERVED_113 | Byte      | 116.0   | Reserved   |
| OB1_RESERVED_114 | Byte      | 117.0   | Reserved   |
| OB1_RESERVED_115 | Byte      | 118.0   | Reserved   |
| OB1_RESERVED_116 | Byte      | 119.0   | Reserved   |
| OB1_RESERVED_117 | Byte      | 120.0   | Reserved   |
| OB1_RESERVED_118 | Byte      | 121.0   | Reserved   |
| OB1_RESERVED_119 | Byte      | 122.0   | Reserved   |
| OB1_RESERVED_120 | Byte      | 123.0   | Reserved   |
| OB1_RESERVED_121 | Byte      | 124.0   | Reserved   |
| OB1_RESERVED_122 | Byte      | 125.0   | Reserved   |
| OB1_RESERVED_123 | Byte      | 126.0   | Reserved   |
| OB1_RESERVED_124 | Byte      | 127.0   | Reserved   |
| OB1_RESERVED_125 | Byte      | 128.0   | Reserved   |
| OB1_RESERVED_126 | Byte      | 129.0   | Reserved   |
| OB1_RESERVED_127 | Byte      | 130.0   | Reserved   |
| OB1_RESERVED_128 | Byte      | 131.0   | Reserved   |
| OB1_RESERVED_129 | Byte      | 132.0   | Reserved   |
| OB1_RESERVED_130 | Byte      | 133.0   | Reserved   |
| OB1_RESERVED_131 | Byte      | 134.0   | Reserved   |
| OB1_RESERVED_132 | Byte      | 135.0   | Reserved   |
| OB1_RESERVED_133 | Byte      | 136.0   | Reserved   |
| OB1_RESERVED_134 | Byte      | 137.0   | Reserved   |
| OB1_RESERVED_135 | Byte      | 138.0   | Reserved   |
| OB1_RESERVED_136 | Byte      | 139.0   | Reserved   |
| OB1_RESERVED_137 | Byte      | 140.0   | Reserved   |
| OB1_RESERVED_138 | Byte      | 141.0   | Reserved   |
| OB1_RESERVED_139 | Byte      | 142.0   | Reserved   |
| OB1_RESERVED_140 | Byte      | 143.0   | Reserved   |
| OB1_RESERVED_141 | Byte      | 144.0   | Reserved   |
| OB1_RESERVED_142 | Byte      | 145.0   | Reserved   |
| OB1_RESERVED_143 | Byte      | 146.0   | Reserved   |
| OB1_RESERVED_144 | Byte      | 147.0   | Reserved   |
| OB1_RESERVED_145 | Byte      | 148.0   | Reserved   |
| OB1_RESERVED_146 | Byte      | 149.0   | Reserved   |
| OB1_RESERVED_147 | Byte      | 150.0   | Reserved   |
| OB1_RESERVED_148 | Byte      | 151.0   | Reserved   |
| OB1_RESERVED_149 | Byte      | 152.0   | Reserved   |
| OB1_RESERVED_150 | Byte      | 153.0   | Reserved   |
| OB1_RESERVED_151 | Byte      | 154.0   | Reserved   |
| OB1_RESERVED_152 | Byte      | 155.0   | Reserved   |
| OB1_RESERVED_153 | Byte      | 156.0   | Reserved   |
| OB1_RESERVED_154 | Byte      | 157.0   | Reserved   |
| OB1_RESERVED_155 | Byte      | 158.0   | Reserved   |
| OB1_RESERVED_156 | Byte      | 159.0   | Reserved   |
| OB1_RESERVED_157 | Byte      | 160.0   | Reserved   |
| OB1_RESERVED_158 | Byte      | 161.0   | Reserved   |
| OB1_RESERVED_159 | Byte      | 162.0   | Reserved   |
| OB1_RESERVED_160 | Byte      | 163.0   | Reserved   |
| OB1_RESERVED_161 | Byte      | 164.0   | Reserved   |
| OB1_RESERVED_162 | Byte      | 165.0   | Reserved   |
| OB1_RESERVED_163 | Byte      | 166.0   | Reserved   |
| OB1_RESERVED_164 | Byte      | 167.0   | Reserved   |
| OB1_RESERVED_165 | Byte      | 168.0   | Reserved   |
| OB1_RESERVED_166 | Byte      | 169.0   | Reserved   |
| OB1_RESERVED_167 | Byte      | 170.0   | Reserved   |
| OB1_RESERVED_168 | Byte      | 171.0   | Reserved   |
| OB1_RESERVED_169 | Byte      | 172.0   | Reserved   |
| OB1_RESERVED_170 | Byte      | 173.0   | Reserved   |
| OB1_RESERVED_171 | Byte      | 174.0   | Reserved   |
| OB1_RESERVED_172 | Byte      | 175.0   | Reserved   |
| OB1_RESERVED_173 | Byte      | 176.0   | Reserved   |
| OB1_RESERVED_174 | Byte      | 177.0   | Reserved   |
| OB1_RESERVED_175 | Byte      | 178.0   | Reserved   |
| OB1_RESERVED_176 | Byte      | 179.0   | Reserved   |
| OB1_RESERVED_177 | Byte      | 180.0   | Reserved   |
| OB1_RESERVED_178 | Byte      | 181.0   | Reserved   |
| OB1_RESERVED_179 | Byte      | 182.0   | Reserved   |
| OB1_RESERVED_180 | Byte      | 183.0   | Reserved   |
| OB1_RESERVED_181 | Byte      | 184.0   | Reserved   |
| OB1_RESERVED_182 | Byte      | 185.0   | Reserved   |
| OB1_RESERVED_183 | Byte      | 186.0   | Reserved   |
| OB1_RESERVED_184 | Byte      | 187.0   | Reserved   |
| OB1_RESERVED_185 | Byte      | 188.0   | Reserved   |
| OB1_RESERVED_186 | Byte      | 189.0   | Reserved   |
| OB1_RESERVED_187 | Byte      | 190.0   | Reserved   |
| OB1_RESERVED_188 | Byte      | 191.0   | Reserved   |
| OB1_RESERVED_189 | Byte      | 192.0   | Reserved   |
| OB1_RESERVED_190 | Byte      | 193.0   | Reserved   |
| OB1_RESERVED_191 | Byte      | 194.0   | Reserved   |
| OB1_RESERVED_192 | Byte      | 195.0   | Reserved   |
| OB1_RESERVED_193 | Byte      | 196.0   | Reserved   |
| OB1_RESERVED_194 | Byte      | 197.0   | Reserved   |
| OB1_RESERVED_195 | Byte      | 198.0   | Reserved   |
| OB1_RESERVED_196 | Byte      | 199.0   | Reserved   |
| OB1_RESERVED_197 | Byte      | 200.0   | Reserved   |
| OB1_RESERVED_198 | Byte      | 201.0   | Reserved   |
| OB1_RESERVED_199 | Byte      | 202.0   | Reserved   |
| OB1_RESERVED_200 | Byte      | 203.0   | Reserved   |
| OB1_RESERVED_201 | Byte      | 204.0   | Reserved   |
| OB1_RESERVED_202 | Byte      | 205.0   | Reserved   |
| OB1_RESERVED_203 | Byte      | 206.0   | Reserved   |
| OB1_RESERVED_204 | Byte      | 207.0   | Reserved   |
| OB1_RESERVED_205 | Byte      | 208.0   | Reserved   |
| OB1_RESERVED_206 | Byte      | 209.0   | Reserved   |
| OB1_RESERVED_207 | Byte      | 210.0   | Reserved   |
| OB1_RESERVED_208 | Byte      | 211.0   | Reserved   |
| OB1_RESERVED_209 | Byte      | 212.0   | Reserved   |
| OB1_RESERVED_210 | Byte      | 213.0   | Reserved   |
| OB1_RESERVED_211 | Byte      | 214.0   | Reserved   |
| OB1_RESERVED_212 | Byte      | 215.0   | Reserved   |
| OB1_RESERVED_213 | Byte      | 216.0   | Reserved   |
| OB1_RESERVED_214 | Byte      | 217.0   | Reserved   |
| OB1_RESERVED_215 | Byte      | 218.0   | Reserved   |
| OB1_RESERVED_216 | Byte      | 219.0   | Reserved   |
| OB1_RESERVED_217 | Byte      | 220.0   | Reserved   |
| OB1_RESERVED_218 | Byte      | 221.0   | Reserved   |
| OB1_RESERVED_219 | Byte      | 222.0   | Reserved   |
| OB1_RESERVED_220 | Byte      | 223.0   | Reserved   |
| OB1_RESERVED_221 | Byte      | 224.0   | Reserved   |
| OB1_RESERVED_222 | Byte      | 225.0   | Reserved   |
| OB1_RESERVED_223 | Byte      | 226.0   | Reserved   |
| OB1_RESERVED_224 | Byte      | 227.0   | Reserved   |
| OB1_RESERVED_225 | Byte      | 228.0   | Reserved   |
| OB1_RESERVED_226 | Byte      | 229.0   | Reserved   |
| OB1_RESERVED_227 | Byte      | 230.0   | Reserved   |
| OB1_RESERVED_228 | Byte      | 231.0   | Reserved   |
| OB1_RESERVED_229 | Byte      | 232.0   | Reserved   |
| OB1_RESERVED_230 | Byte      | 233.0   | Reserved   |
| OB1_RESERVED_231 | Byte      | 234.0   | Reserved   |
| OB1_RESERVED_232 | Byte      | 235.0   | Reserved   |
| OB1_RESERVED_233 | Byte      | 236.0   | Reserved   |
| OB1_RESERVED_234 | Byte      | 237.0   | Reserved   |
| OB1_RESERVED_235 | Byte      | 238.0   | Reserved   |
| OB1_RESERVED_236 | Byte      | 239.0   | Reserved   |
| OB1_RESERVED_237 | Byte      | 240.0   | Reserved   |
| OB1_RESERVED_238 | Byte      | 241.0   | Reserved   |
| OB1_RESERVED_239 | Byte      | 242.0   | Reserved   |
| OB1_RESERVED_240 | Byte      | 243.0   | Reserved   |
| OB1_RESERVED_241 | Byte      | 244.0   | Reserved   |
| OB1_RESERVED_242 | Byte      | 245.0   | Reserved   |
| OB1_RESERVED_243 | Byte      | 246.0   | Reserved   |
| OB1_RESERVED_244 | Byte      | 247.0   | Reserved   |
| OB1_RESERVED_245 | Byte      | 248.0   | Reserved   |
| OB1_RESERVED_246 | Byte      | 249.0   | Reserved   |
| OB1_RESERVED_247 | Byte      | 250.0   | Reserved   |
| OB1_RESERVED_248 | Byte      | 251.0   | Reserved   |
| OB1_RESERVED_249 | Byte      | 252.0   | Reserved   |
| OB1_RESERVED_250 | Byte      | 253.0   | Reserved   |
| OB1_RESERVED_251 | Byte      | 254.0   | Reserved   |
| OB1_RESERVED_252 | Byte      | 255.0   | Reserved   |
| OB1_RESERVED_253 | Byte      | 256.0   | Reserved   |
| OB1_RESERVED_254 | Byte      | 257.0   | Reserved   |
| OB1_RESERVED_255 | Byte      | 258.0   | Reserved   |
| OB1_RESERVED_256 | Byte      | 259.0   | Reserved   |
| OB1_RESERVED_257 | Byte      | 260.0   | Reserved   |
| OB1_RESERVED_258 | Byte      | 261.0   | Reserved   |
| OB1_RESERVED_259 | Byte      | 262.0   | Reserved   |
| OB1_RESERVED_260 | Byte      | 263.0   | Reserved   |
| OB1_RESERVED_261 | Byte      | 264.0   | Reserved   |
| OB1_RESERVED_262 | Byte      | 265.0   | Reserved   |
| OB1_RESERVED_263 | Byte      | 266.0   | Reserved   |
| OB1_RESERVED_264 | Byte      | 267.0   | Reserved   |
| OB1_RESERVED_265 | Byte      | 268.0   | Reserved   |
| OB1_RESERVED_266 | Byte      | 269.0   | Reserved   |

## Циклические прерывания

В S7-300 для обработки циклического прерывания служит OB35 с приоритетом 12.



При параметризации CPU Вы можете задать значение интервала времени для OB35 в диапазоне от 1 мс до 1 мин (стандартно 100 мс).

При отсутствии OB35 в памяти CPU операционная система его не вызывает.

В S7-400, в зависимости от типа CPU, можно использовать от 2-х до 9 циклических OB (OB30... OB38). С помощью нескольких таких блоков Вы проектируете свою программу как комбинацию циклических задач с разным циклом запуска ("многозадачность").

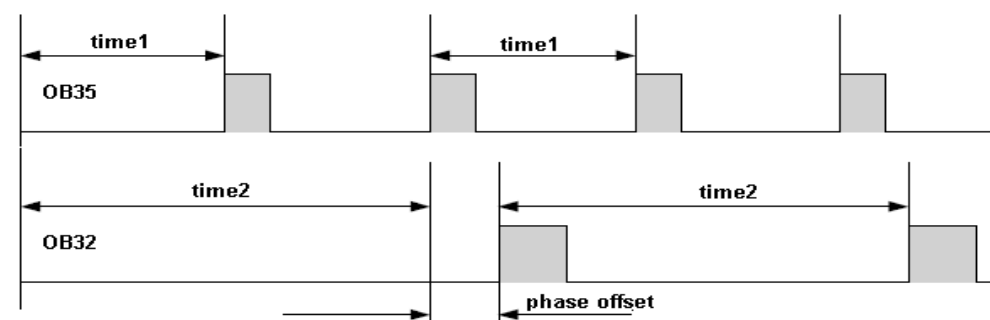
Properties - CPU 414-2 DP - (R0/S2)

|       | General  | Startup    | Synchronous Cycle Interrupts | Cycle/Clock Memory | Retentive Memory        |
|-------|----------|------------|------------------------------|--------------------|-------------------------|
|       | Memory   | Interrupts | Time-of-Day Interrupts       | Cyclic Interrupts  | Diagnostics/Clock       |
|       | Priority | Execution  | Phase offset                 | Unit               | Process image partition |
| OB30: | 0        | 5000       | 0                            | ms                 | ---                     |
| OB31: | 0        | 2000       | 0                            | ms                 | ---                     |
| OB32: | 9        | 1000       | 20                           | ms                 | PIP2                    |
| OB33: | 10       | 500        | 10                           | ms                 | ---                     |
| OB34: | 11       | 200        | 10                           | ms                 | ---                     |
| OB35: | 12       | 100        | 0                            | ms                 | ---                     |
| OB36: | 0        | 50         | 0                            | ms                 | ---                     |
| OB37: | 0        | 20         | 0                            | ms                 | ---                     |

В S7-400 циклическое прерывание имеет 4 параметра: интервал, фазовое смещение, приоритет и указание на обновление областей отображения PII/PIQ (образа процесса).

Значения для приоритета выбираются из диапазона значений от 2 до 24. Если приоритет приравнивается нулю, то циклическое прерывание не будет активно.

Параметр "Фазовое смещение" (Phase offset) может быть использован для обеспечения точности выдержки заданного временного интервала.



Как видно из рисунка, временной интервал для OB32 (time 2) больше в 2 раза времени запуска (time 1) для OB35. Если для OB32 не назначить фазовый сдвиг, то он будет всегда дожидаться выполнения OB35, как блока с более высоким приоритетом.

General Addresses

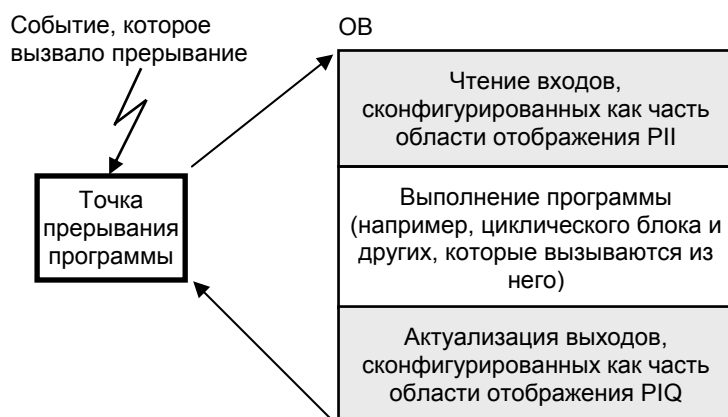
| Inputs |   |
|--------|---|
| Start: | 0 |
| End:   | 3 |

Process image: PIP 1

Параметр "Process image partition" позволяет связать вызов блока OB с функцией операционной системы по управлению определенной частью памяти PII/PIQ. При задании адреса модулям DI/DO, Вы также можете назначить им параметр "Process image", связав модуль с одной из 15 возможных внутренних областей (PIP1 – PIP15) памяти PII/PIQ.

## Обновление образа процесса при вызове OB

Если организационному блоку была указана связь с признаком PIPx, как части области отображения PII/PIQ, то операционная система CPU перед вызовом OB и после его завершения выполняет актуализацию PII/PIQ подобно тому, как при вызове OB1.



## Пример циклической программы

В отличие от OB1, блок OB35 имеет постоянное время цикла. Это позволяет строить на основе OB35 (либо других циклических блоков) программы, критичные к стабильности времени цикла. В качестве примера можно привести формирование интегральной составляющей выходного сигнала управления в задаче программного регулирования с обратной связью. Поэтому блоки регулирования (из библиотеки Standart Library->PID Control Block) необходимо всегда вызывать из OB циклических прерываний.

## Прерывания по времени суток

Прерывания по времени суток (Time-of-Day Interrupts) используются для выполнения определенных задач в заданное время суток однократно или периодически (каждую минуту, час, день, месяц, в конце месяца, или ежегодно). Для этого Вы можете использовать организационные блоки с OB 10 по OB 17. Какие из данных восьми OB доступны, зависит от типа CPU.

Предпосылкой правильного выполнения обработки прерываний по времени суток является корректная установка часов реального времени (PLC->Set Time of Day).

Если произошло прерывание по времени суток, а OB обработки не запрограммирован, то операционная система вызывает OB 85 (блок обработки ошибок). Если OB 85 не запрограммирован, то CPU переходит в режим STOP.

## Конфигурирование прерываний по времени суток

Вы можете произвести конфигурирование прерываний при параметрировании CPU.

| Properties - CPU 414-2 DP - (R0/S2) |            |                                     |                    |                   |             |     |
|-------------------------------------|------------|-------------------------------------|--------------------|-------------------|-------------|-----|
| General                             | Startup    | Synchronous Cycle Interrupts        | Cycle/Clock Memory | Retentive Memory  |             |     |
| Memory                              | Interrupts | Time-of-Day Interrupts              | Cyclic Interrupts  | Diagnostics/Clock | Protection  |     |
|                                     | Priority   | Active                              | Execution          | Start date        | Time of day | PIP |
| OB10:                               | 2          | <input checked="" type="checkbox"/> | Every minute       | 01/01/2008        | 00:00       | --- |
| OB11:                               | 3          | <input checked="" type="checkbox"/> | Every hour         | 01/01/1994        | 00:05       | --- |
| OB12:                               | 2          | <input type="checkbox"/>            | None               | 01/01/1994        | 00:00       | --- |

Опция "Active" разрешает запуск прерывания. В поле "Execution" выбирается режим: однократный или периодический. В поле "Start date" и "Time of day" вводится начальное время отсчета. Параметр "PIP" - см. конфигурирование циклических блоков.



### Системные функции для прерываний по времени суток

Вы можете управлять прерываниями по времени суток непосредственно из программы с помощью следующих системных функций:

- SFC 28 SET\_TINT (функция для установки времени прерывания и периодичности)
- SFC 29 CAN\_TINT (функция для отмены прерывания)
- SFC 30 ACT\_TINT (функция для активации прерывания)
- SFC 31 QRY\_TINT (функция для запроса о состоянии прерывания)

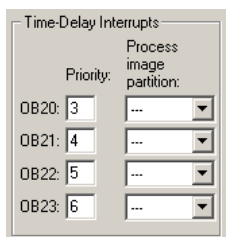
Управление прерываниями с помощью системных функций отменяет назначения, сделанные при параметрировании CPU.

### Программные прерывания

Программные прерывания или прерывания с задержкой обработки (Time-Delay Interrupts) позволяют запускать определенные подзадачи по событию, обнаруженному в программе (например, изменение ячейки памяти с 0 на 1).

Особенность данных прерываний в том, что запуск их осуществляется только через вызов системной функции (SFC 32). При этом системная функция запускает системный таймер для отсчета задержки времени, по истечении которой операционная система вызывает соответствующий организационный блок.

Время задержки вызова может составлять от 1 мс до 60000мс.



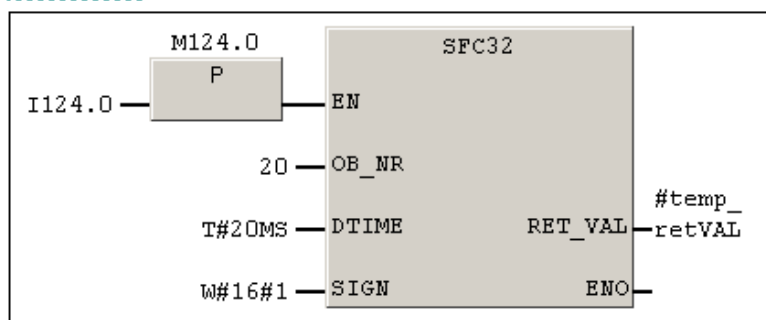
В S7-400, в зависимости от CPU, можно использовать до 4-х блоков программных прерываний (OB 20 - OB 23)

При параметрировании CPU пользователь может задать приоритеты для OB и область PIP с целью обновления части памяти PII/PIQ при вызове прерывания.

В S7-300 можно использовать только один блок (OB20) с приоритетом 3.

### Пример вызова блока OB20

**Network 5:** Вызов OB20 с задержкой через 20 ms



Параметр "SIGN" передается в локальные данные OB и может быть полезен для выбора различных подзадач внутри одного OB.

Если вызов функции SFC32 происходит в то время, когда не закончилось время с момента предыдущего вызова SFC32 для того же блока OB, то отсчет времени начинается сначала.

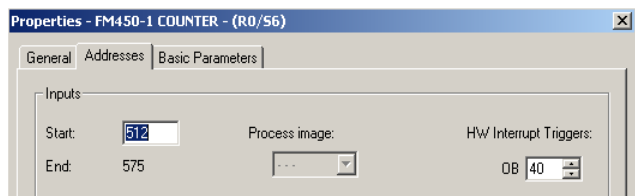
Примером использования блоков программных прерываний может быть задача "Подавление дребезга контактов". Такая типовая задача возникает, например, при подсчете срабатываний от конечного выключателя.



## Аппаратные прерывания

Аппаратные прерывания (Hardware Interrupts) используются для быстрой реакции пользовательской программы на события, происходящие в управляемом процессе (например, изменение сигнала с 0 на 1). Источниками прерываний служат внешние модули (например, DI-модули) с возможностью передачи на CPU сигналов прерывания. В ответ на сигнал прерывания CPU вызывает организационный блок обработки, имеющий, как правило, высокий приоритет.

В S7-300 только один блок (OB40) может быть запрограммирован для обработки прерывания.



В S7-400 в зависимости от типа CPU можно использовать от 2 до 8 блоков (с OB 40 по OB 47). Номер блока OB прерывания указывается при назначении адреса модуля.

Диаграмма показывает реакцию CPU на быстрое изменение сигналов прерывания



## Различия в обработке прерываний в S7-300 и в S7-400

В S7-300, любые аппаратные сигналы прерываний, поступающие во время выполнения OB40, в дальнейшем не обрабатываются.

В S7-400, все сигналы, поступившие либо с другого канала, либо от другого модуля обрабатываются после завершения текущего блока прерывания (производится перезапуск блока прерывания).

## Использование локальных данных в блоке обработки прерывания

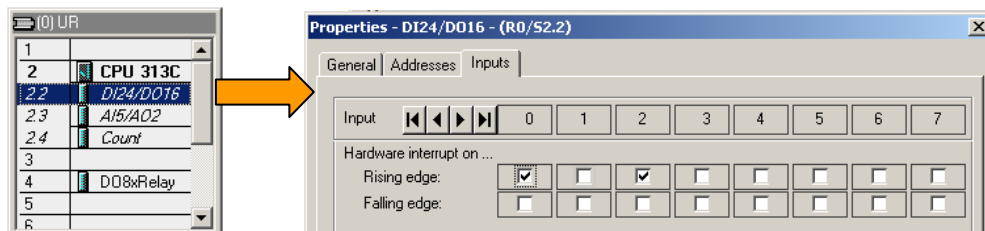
Если несколько событий назначены для вызова одного и того же организационного блока, то необходимо использовать локальные данные для запуска различных задач обработки прерывания внутри одного блока.

Так, локальная переменная OB40\_MD\_ADDR (LW6) передает информацию о базовом адресе модуля - источнике прерывания. Переменная OB40\_POINT\_ADDR (LD8) через битовый образ информирует о номере канала - источника прерывания.

## Пример конфигурирования и обработки аппаратного прерывания в OB40

Будем использовать метод аппаратных прерываний для сокращения времени реакции на отключение, например, мотора (Q4.0) при появлении аварийного сигнала (I0.2). Для этого воспользуемся встроенными цифровыми входами CPU 313C с поддержкой прерывания.

В утилите HW-Config выполним необходимые назначения.



Программа в блоке OB40:

OB40 : "Hardware Interrupt"

**Network 1:** Выделение сигналов прерывания

```
L      #OB40_POINT_ADDR  // чтение сигналов прерывания
L      W#16#4             // маска для выделения сигнала I0.2
AD
JP      m1                // переход по прерыванию I0.2
JU      m2                // переход по прерыванию I0.0
```

**Network 2:** Обработчик прерывания I0.2

```
m1:    L      QB      4           // чтение образа выходов
        L      2#11111110        // маска для сброса Q4.0
        AD
        T      QB      4           // обновление образа
        T      PQB     4           // установка выхода Q4.0=0
```

## ОВ ошибок

Операционная система CPU вызывает специальные организационные блоки для обработки обнаруженных ошибок. При этом всегда загорается индикатор "SF" (красного цвета) на лицевой панели CPU. Если ошибка возникнет, а соответствующий блок ОВ не загружен в CPU, то CPU перейдет в STOP.

## ОВ синхронных ошибок

Синхронными называются ошибки в программе, для которых можно точно указать место в программе. Для обработки синхронных ошибок применяются блоки OB121 и OB122. Приоритетный класс не назначается (такой же, как у прерванной задачи).

Ошибки программирования, для обработки которых вызывается блок OB 121:

- Ошибка при BCD-преобразовании
- Обращение к блокам (FC, FB, DB), которые не загружены в CPU
- Попытка записи в блок DB с защитой от записи ("write protected")
- Доступ к системной памяти (I, Q, M, T, C) за пределы зоны доступных адресов.
- Ошибка при выполнении косвенной адресации (адрес бита не равен 0 при обращении к байту/слову/двойному слову)

Блок OB 122 вызывается при прямом доступе к неисправному или к несуществующему модулю. Например, при выполнении команды L PIB8, произойдет вызов OB122, если в системе нет модуля с адресом 8.

## ОВ асинхронных ошибок

Асинхронные ошибки - это такие ошибки, которые могут произойти независимо от выполнения программы. Все ОВ асинхронных ошибок имеют высокий приоритет (по умолчанию 25). Асинхронные ошибки могут быть обработаны следующими блоками:

- ОВ 80 - ошибки времени (timing error)
- ОВ 81 - ошибки в блоке питания (power supply error)
- ОВ 82 - диагностические прерывания (diagnostic interrupt)
- ОВ 83 - ошибки при установке/удалении модуля
- ОВ 84 - обработки отказа аппаратной части CPU
- ОВ 85 - ошибки выполнения программы (program execution error)
- ОВ 86 - обработки отказа стойки (rack failure)
- ОВ 87 - обработки коммуникационной ошибки (communication error).

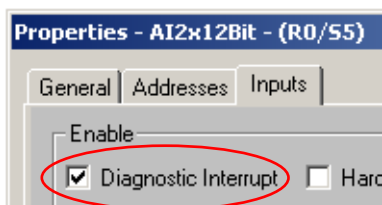
### Ошибки времени

Операционная система вызывает организационный блок ОВ 80, если происходит одна из следующих ошибок:

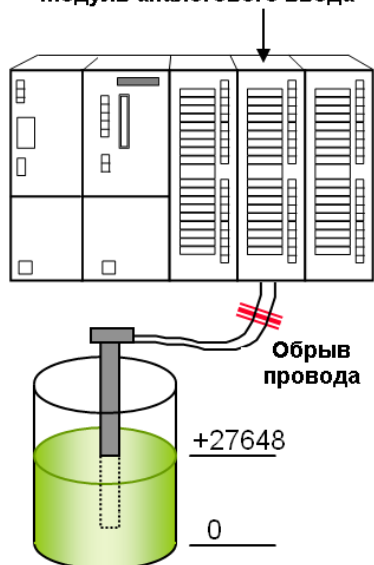
- превышает время мониторинга цикла сканирования программы;
- из-за перевода часов реального времени вперед (если время, установленное для прерывания по времени суток прошло).

CPU переходит в режим STOP, если организационный блок ОВ 80 вызывается второй раз в том же самом цикле сканирования программы.

### Диагностические прерывания



Модуль аналогового ввода



Операционная система вызывает организационный блок ОВ 82, когда модуль с поддержкой внутренней диагностики обнаруживает ошибку, а также при устранении ошибки. Пользователь должен установить параметр, разрешающий передачу диагностического прерывания на CPU.

Если Вы имеете модуль аналогового ввода с функцией диагностики, то ОВ82 вызывается, например, при обрыве провода или когда превышен диапазон измерения.

Вы можете запрограммировать ОВ82, чтобы получить детальную информацию об ошибке и сформировать передачу пользовательского сообщения в диагностический буфер или на панель оператора.

При возникновении ошибки ("приходящее" событие) в переменную ОВ82\_EV\_CLASS (LB0) записывается код В#16#39, при устранении ("уходящее" событие) - В#16#38.

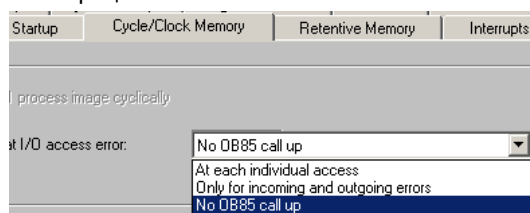
Переменная ОВ82\_MDL\_ADDR (LW6) из локальных данных блока содержит адрес модуля, который послал прерывание. В следующих четырех байтах содержится диагностическая информация, выдаваемая данным модулем.

В блоке ОВ 82 Вы можете использовать системную функцию SFC 59 RD\_REC, которая позволяет получить более детальную информацию об ошибке.

### Ошибки выполнения программы

Операционная система вызывает организационный блок OB 85, если происходит одна из следующих ошибок:

- поступает запрос на запуск организационного блока, который не был загружен
- при ошибке доступа к периферии во время автоматического обновления образа процесса.



При параметрировании CPU (в закладке "Cycle/Clock Memory") Вы можете задать режим вызова OB85 при ошибках доступа к периферии: при каждой ошибке; только при возникновении ошибки и ее устранении; запретить вызов.

### Отказ стойки (Rack Failure)

Операционная система вызывает организационный блок OB 86, если она обнаружила отказ стойки (авария в системе питания, обрыв провода, отказ интерфейсного модуля IM), отказ в подсети или в станции децентрализованной периферии.

### Стартовые блоки

Различают 3 вида старта CPU: полный или теплый (warm restart), горячий (hot restart) и холодный (cold restart).

В S7-300 возможен только теплый или холодный старт (только CPU 318-2).

При каждом виде старта однократно запускается блоки OB100, OB101 и OB102 соответственно (если они загружены в CPU). Назначение данных блоков – инициализация значений переменных перед выходом в RUN-режим.

#### Теплый старт (warm restart)

Это самый обычный вид старта. Одно из названий – полный (complete) рестарт.

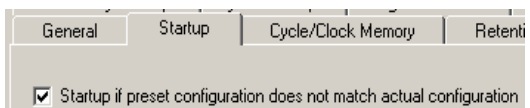
Различают ручной и автоматический старт. Ручной запуск инициируется посредством переключателя режимов или помощью коммуникационной функции (функции в SIMATIC Manager *PLC->Operating Mode*..или системной функции, например, SFB19, запущенной в другом CPU) .

Автоматический перезапуск инициируется при пропадании питания CPU и последующем его возобновлении, если переключатель режимов находился в положении "RUN" или "PUN-P".

При теплом старте выполняются следующая последовательность действий:

- Блокировка выходных модулей
- Сброс таблиц отображения PII /PIQ
- Очистка несохраняемых данных из области M, T, C
- Инициализация модулей
- Выполнение OB100 (если он загружен)
- Разблокирование выходных модулей
- Переход в RUN-режим.

Переменная OB101\_STRTUP (LB 1) локальных данных OB101 содержит информацию о виде старта (B#16#81: ручной старт, B#16#82: автоматический старт).



Примечание: в случае несоответствия текущей и заданной конфигурации старт может быть запрещен, если при параметрировании CPU это было указано (по умолчанию – разрешено).

### Холодный старт (cold restart)

Холодный старт похож на сброс CPU, за исключением того, что не происходит очистки загрузочной памяти. При таком сбросе очищается вся внутренняя память CPU (рабочая и системная), а затем производится загрузка программы из загрузочной памяти в рабочую. Все блоки данных, созданные в загрузочной памяти, например, с помощью SFC82 удаляются. CPU устанавливает свои собственные параметры и параметры модулей в запрограммированное исходное состояние.

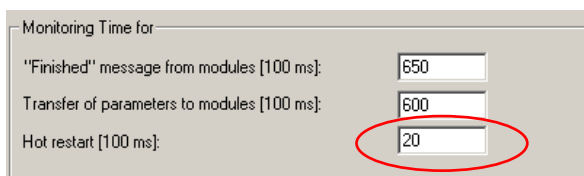
Холодный старт может быть выполнен только в новых моделях CPU S7-400 и в CPU318-2. Ручной старт возможен только от устройства программирования либо с помощью системной функции, исполняемой на другом CPU. При этом переключатель режимов должен быть в положении RUN.

### Горячий старт (hot restart)

Такой вид старта возможен только в системах S7-400.

При "горячем" старте возможно продолжение выполнения программы, начиная с точки, в которой ее выполнение было прервано. Кроме того, информация обо всех "старых" прерываниях сохраняется, поэтому все они будут обслужены.

"Горячий" перезапуск возможен только тогда, когда не было сделано никаких изменений в пользовательской программе в то время, пока CPU находился в состоянии STOP.

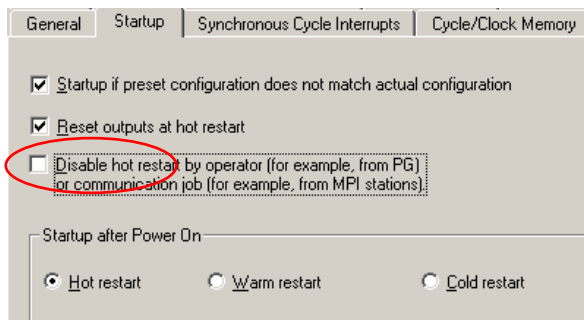


При параметризации CPU Вы можете определить отрезок времени, в течение которого сохраняется разрешение для CPU на выполнение "горячего" перезапуска после прерывания (в диапазоне от 100 мс до 1 часа).

Если прерывание длится большее время, то будет разрешен только "полный" перезапуск. Если задано значение 0, то время не контролируется.

"Горячий" перезапуск в ручном режиме выполняется в следующих случаях:

- если переключатель режима перезапуска находился в положении "WRST" и произведен перевод CPU в режим RUN
- с помощью коммуникационных функций от программатора PG или SFB.

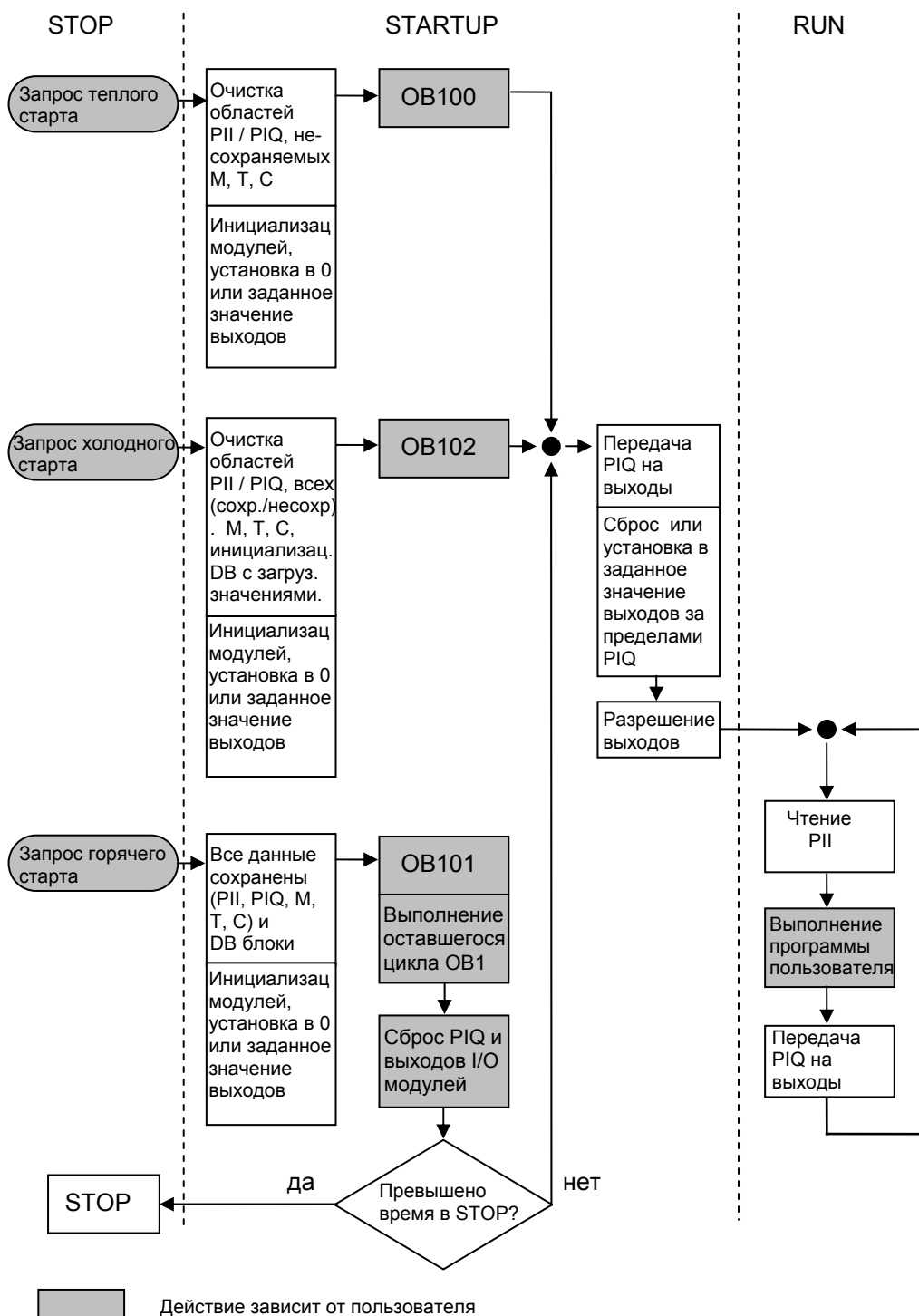


Ручной режим "горячего" перезапуска может быть инициирован, если на вкладке "Startup" окна параметризации CPU отменена блокировка "теплого" перезапуска.

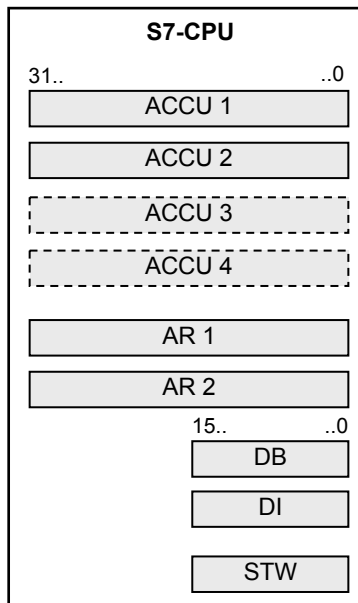
Автоматический "горячий" перезапуск выполняется при соблюдении условий:

- CPU не был в режиме STOP, когда было выключено питание
- переключатель режимов находится в положениях RUN или RUN-P, когда CPU был включен
- задан параметр "Hot restart" для функции "Startup after Power On" (горячий рестарт при включении питания)
- батарея резервного питания вставлена в батарейный отсек и находится в рабочем состоянии.

Следующий рисунок показывает действия CPU в режимах STARTUP и RUN.



## Программно-доступные регистры S7- CPU

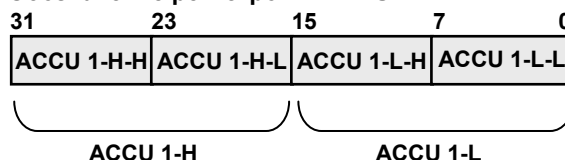


S7-300 CPU имеют только 2 аккумулятора (кроме CPU 318), тогда как S7-400 CPU имеют 4.

Разрядность аккумуляторов – 32.

ACCU 1 и ACCU 2 представляются комбинацией двух 16-разрядных регистров или четырех 8-разрядных программно доступных регистров.

**Обозначение регистров в ACCU1:**



Регистры AR 1 и AR 2 используются для косвенной адресации к данным. Разрядность AR 1 / AR 2 - 32.

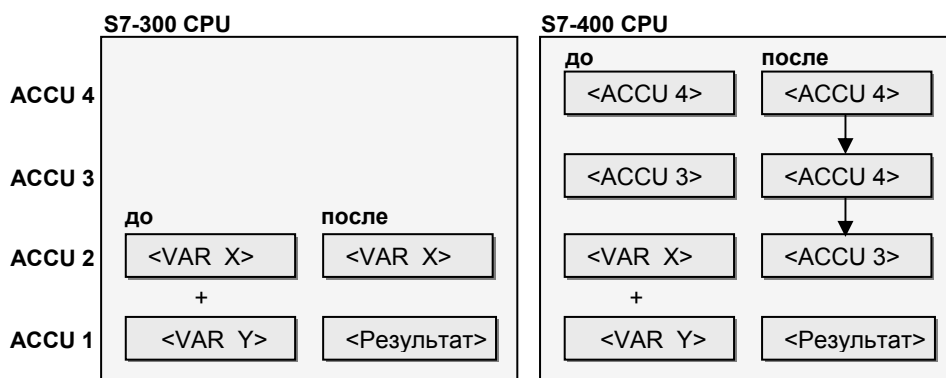
Регистры DB и DI хранят номер открытого DB для операций доступа к элементам блоков DB.

Регистр STW (слово состояния) содержит битовые признаки операций (/FC, RLO, STA, OR, OS, OV, CC1, CC0, BR).

## Инструкции с аккумуляторами

Все математические инструкции, такие как сложение, вычитание, умножение, деление, получение остатка от деления (например, +I, -D, \*R, /D) используют ACCU 1 и ACCU 2 для временного хранения чисел до операции и всегда помещают результат в ACCU 1.

Особенностью CPU с 4-мя аккумуляторами является то, что после выполнения операции операнд в ACCU2 не сохраняется (как в CPU с 2-мя ACCU), т.к. ACCU 2 принимает значение ACCU 3, а ACCU 3 – значение из ACCU 4.



Вы можете запрограммировать выполнение операций одну за другой. В этом случае результат выполнения первой операции будет использоваться для обработки в последующей операции.

Примечание: из-за различий выполнения арифметических команд (в поведении аккумуляторов) результат может отличаться в разных CPU при обработке одной и той же последовательности команд.



Пример выполнения программы в S7-400 и S7-300 и состояние аккумуляторов после выполнения каждой команды

| Программа<br>на STL: | S7-300 |        | S7-400 |        |        |        |
|----------------------|--------|--------|--------|--------|--------|--------|
|                      | ACCU 1 | ACCU 2 | ACCU 1 | ACCU 2 | ACCU 3 | ACCU 4 |
| L VAR1               | <VAR1> | ACCU 2 | <VAR1> | ACCU 2 | ACCU 3 | ACCU 4 |
| L VAR2               | <VAR2> | <VAR1> | <VAR2> | <VAR1> | ACCU 3 | ACCU 4 |
| +D                   | SUM1   | <VAR1> | SUM2   | ACCU 3 | ACCU 4 | ACCU 4 |
| L VAR3               | <VAR3> | SUM1   | <VAR3> | SUM2   | ACCU 4 | ACCU 4 |
| L VAR4               | <VAR4> | <VAR3> | <VAR4> | <VAR3> | ACCU 4 | ACCU 4 |
| -D                   | SUB1   | <VAR3> | SUB2   | ACCU 4 | ACCU 4 | ACCU 4 |
| *D                   | MUL1   | <VAR3> | MUL2   | ACCU 4 | ACCU 4 | ACCU 4 |
| T RESULT             | MUL1   | <VAR3> | MUL2   | ACCU 4 | ACCU 4 | ACCU 4 |

Промежуточный результат SUM1=SUM2 и равен VAR1+VAR2.

Промежуточный результат SUB1=SUB2 и равен VAR3-VAR4.

Окончательный результат MUL1=MUL2.

Действительно,  $MUL1 = VAR3 * (VAR3 - VAR4) = (VAR3)^2 - VAR4$ , а

$MUL2 = (VAR3 - VAR) * ACCU 4$ , где ACCU 4 не известно.

### Добавление констант к содержимому аккумулятора ACCU 1 (команда +)

Операция добавления константы более предпочтительна для вычислений с адресуемыми данными, поскольку по сравнению с арифметическими функциями она не влияет на содержимое остальных аккумуляторов, а также выполняется всего за одну команду.

Если записать десятичное число со знаком минус, то, таким образом, можно выполнить операцию вычитания константы из значения в ACCU 1.

Примеры: + -33 // вычитает число 33 из ACCU1 (результат = ACCU 1-L - 33)  
+ L#1 // прибавляет 1 к ACCU1 в формате DINT

### Операции декрементирования и инкрементирования

При выполнении команд декрементирования и инкрементирования содержимое ACCU 1-L-L уменьшается (декрементируется) или увеличивается (инкрементируется) на число (0-255), определенное в параметре инструкции. Старшие байты аккумулятора не изменяются.

Если в результате инкрементирования значение превышает величину 255, то расчет значения начинается с начала (с 0). Если в результате декрементирования значение становится меньше 0, то расчет значения начинается с максимально возможного значения (с 255).

Примеры: INC 5 // прибавляет 5 к ACCU 1-L-L число 5  
DEC 7 // уменьшает ACCU 1-L-L на 7



### Команды пересылки данных между аккумуляторами

| CAW  |       |       |       |
|--|-------|-------|-------|
| Байты аккумулятора 1 до выполнения операции    |       |       |       |
| n  | n + 1 | n + 2 | n + 3 |
| Байты аккумулятора 1 после выполнения операции |       |       |       |
| n  | n + 1 | n + 3 | n + 2 |

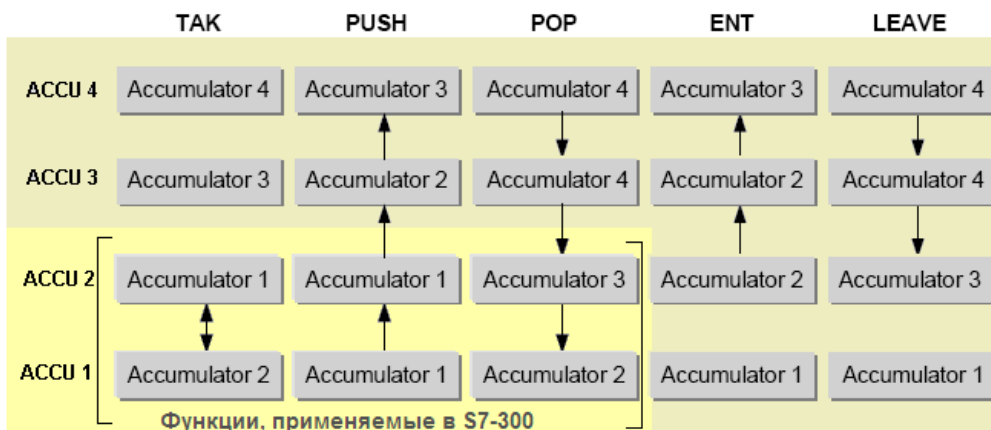
Команда CAW меняет местами два байта в младшем слове в ACCU 1.

| CAD  |       |       |       |
|--|-------|-------|-------|
| Байты аккумулятора 1 до выполнения операции    |       |       |       |
| n  | n + 1 | n + 2 | n + 3 |
| Байты аккумулятора 1 после выполнения операции |       |       |       |
| n + 3  | n + 2 | n + 1 | n     |

Команда CAD меняет местами все байты в аккумуляторе ACCU 1.

Имеется еще 5 команд прямой пересылки данных между аккумуляторами, причем последние 2 команды не применяются в S7-300:

- **PUSH** сдвиг содержимого аккумулятора "вперед"
- **POP** сдвиг содержимого аккумулятора "назад"
- **TAK** обмен содержимым между ACCU 1 и ACCU 2
- **ENT** сдвиг содержимого аккумулятора "вперед" (без ACCU 1)
- **LEAVE** сдвиг содержимого аккумулятора "назад" (без ACCU 1).



Рассмотрим пример программы для S7-400:

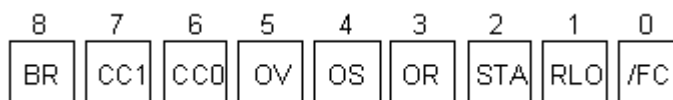
```

L VAR1
L VAR2
+D
L VAR3
ENT
L VAR4
-D
*D
T RESULT
    
```

С помощью инструкции ENT сохраняется промежуточный результат (сумма VAR1 + VAR2) в ACCU 3. После выполнения операции вычитания сумма переписывается из ACCU 3 в ACCU 2. Поэтому, после команды умножения операнд RESULT принимает следующее значение:  
 $RESULT := (VAR1 + VAR2) * (VAR3 - VAR4)$



**Регистр STW** (слово состояния) хранит 9 битовых признаков выполнения операций.



### Двоичные флаги:

/FC - первичный опрос (first check)  
RLO - результат логической операции  
STA - состояние сигнала ("status")  
OR - бит состояния OR (OR status bit)  
BR - двоичный результат (binary result)

### Числовые флаги:

OS - для сохранения информации о переполнении (stored overflow)  
OV - переполнение (overflow)  
CC0 - условный код (condition code)  
CC1 - условный код (condition code)

CPU использует "двоичные флаги" ("binary flags") для управления двоичными функциями; "числовые флаги" ("digital flags") используются, прежде всего, для индикации результатов математических функций.

Используя биты статуса, Вы можете выполнять оценку предыдущих операций. Например, с помощью логической инструкции A (или AN, O, ON, X, XN) можно проверить биты слова статуса, а иногда и проверить результат математической операции:

A ==0 проверка результата на 0 ((CC 1 = 0) AND (CC 0 = 0))  
A <>0 проверка на ≠ (((CC 1 = 1) AND (CC 0 = 0)) OR ((CC 1 = 0) AND (CC 0 = 1)))  
A >0 проверка результата на >0 ((CC 1 = 1) AND (CC 0 = 0))  
A <0 проверка результата на <0 ((CC 1 = 0) AND (CC 0 = 1))  
A >=0 проверка на >=0 (((CC 1 = 1) AND (CC 0 = 0)) OR ((CC 1 = 0) AND (CC 0 = 0)))  
A <=0 проверка на <=0 (((CC 1 = 0) AND (CC 0 = 1)) OR ((CC 1 = 0) AND (CC 0 = 0)))  
A UO проверка, что результат не верен ((CC 1 = 1) AND (CC 0 = 1))  
A OS проверка OS = 1  
A OV проверка OV = 1  
A BR проверка BR = 1

Вы можете проверять биты состояния с помощью соответствующих функций перехода.

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

Вы можете загружать слово состояния в ACCU 1, а также считать его значение из ACCU 1.

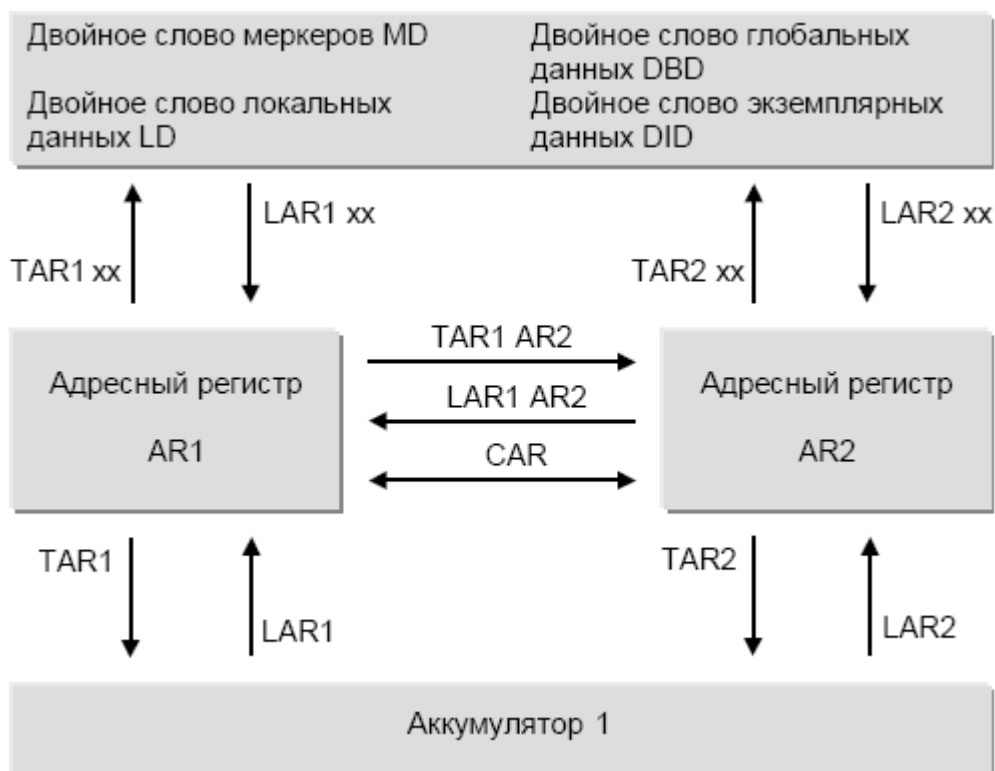
L STW // загрузить слово состояния  
T STW // загрузить в слово состояния

Замечание: S7-300 CPU (кроме CPU 318) не загружают биты состояния /FC, STA и OR в ACCU 1; аккумулятор в соответствующих этим битам позициях содержит "0".

### Адресные регистры

32-разрядные регистры AR 1 и AR 2 в основном используются для косвенной адресации к данным. В этом случае адресные регистры содержат указатели адреса. Иногда AR1 или AR2 используются для временных пересылок данных в ACCU1. Так, по команде LAR1 (LAR2) содержимое ACCU1 сохраняется в AR1 (AR2), а по команде TAR1 (TAR2) значение AR1 (AR2) пересылается в ACCU1.

Команды пересылки между ACCU1, AR1, AR2 и ячейками из области M, L и блоков данных представлены на рисунке.



### Примеры:

LAR1 "X\_SCALE" - загрузить в AR1 содержимое переменной X\_SCALE;  
LAR1 DB10.DBD22 - загрузить в AR1 двойное слово DBD22 из DB10;  
LAR1 LD22 - загрузить в AR1 содержимое переменной LD22;  
TAR1 "Y\_POINT" - переслать AR1 в переменную Y\_POINT;  
TAR1 LD22 - переслать AR1 в локальную переменную LD22;  
TAR1 DB10.DBD32 - переслать AR1 в двойное слово DBD32 из DB10.

## Режимы адресации данных

Виды адресации, доступные в STEP 7, изображены на рисунке.



При *непосредственной адресации* численное значение задается непосредственно в операторе. Примеры: L 5; SLW 2; SET; CLR.

С помощью *прямой адресации* Вы получаете прямой доступ к адресу, например, A I 1.2 или L MW 122. Вы адресуете данную ячейку памяти, указывая ее адрес прямо в команде.

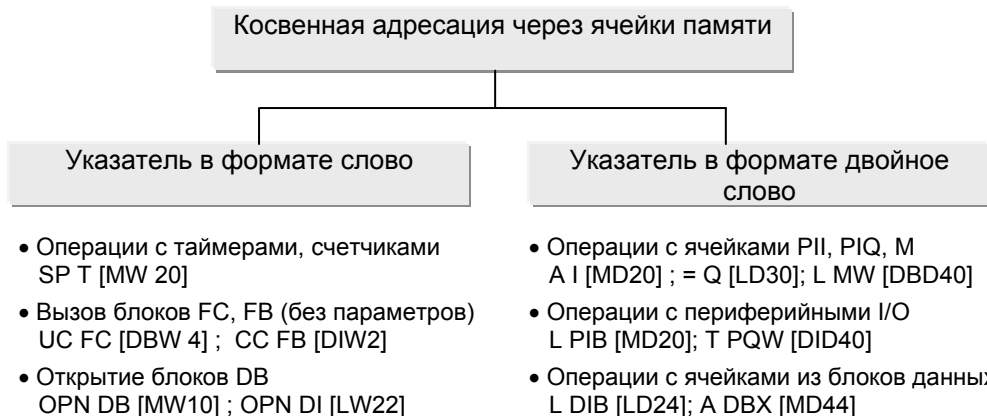
При использовании *косвенной адресации* Вы указываете, как адрес должен быть найден, вместо указания самого адреса. При косвенной адресации адреса переменных вычисляются каждый раз во время выполнения команды.

Пример: A I[MD4] – опрос входа, адрес которого находится по адресу MD4.

Адресация посредством параметров блока – особая форма косвенной адресации: назначая фактические параметры параметрам блока, Вы определяете адреса, которые будут обработаны во время выполнения программы.

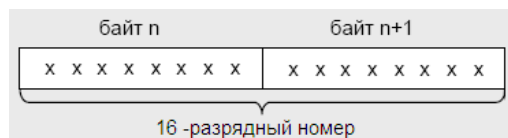
### Косвенная адресация через ячейки памяти

При косвенной адресации посредством памяти (memory-indirect addressing) адрес указывается с помощью адресованной ячейки памяти. Адрес должен иметь размер двойного слова, если требуется использовать указатель на область памяти, или же он должен иметь размер слова, если требуется использовать число в качестве указателя. Адрес операнда может находиться в одной из ниже перечисленных областей: область M (флаги), L-стек, блоки данных DB, открытые через DB или DI регистры.



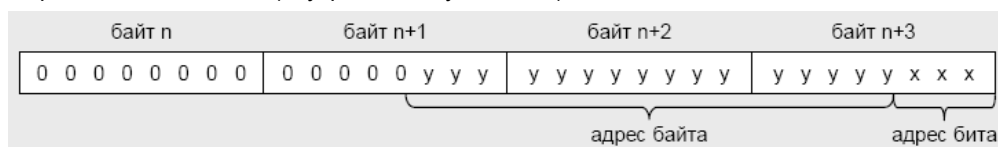
## Форматы указателей адреса

Формат слово:



Данный указатель имеет длину 16 бит и содержит номер, используемый для косвенной адресации таймеров, счетчиков или для вызова блоков FC, FB и DB.

Формат двойное слово (внутризонный указатель):



Данный указатель содержит адрес байта и адрес бита. Если Вы хотите адресовать числовые операнды, то Вы должны всегда в качестве адреса бита указывать 0.

### Константа–указатель

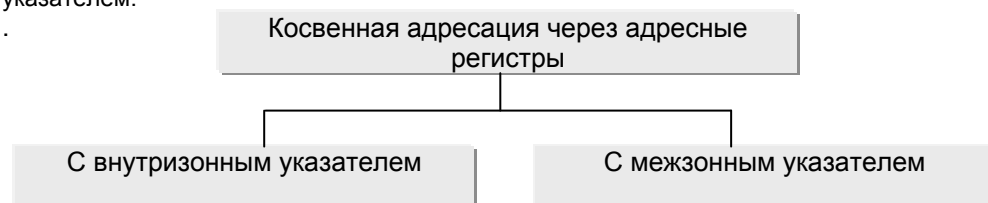
С помощью специальной константы-указателя вида P#y.x Вы можете загружать указатель адреса в ячейку памяти:

Пример: L P#30.0 // загрузка указателя в аккумулятор

T MD20 // сохранение указателя (MD 20 указывает на байт №30 и бит №0)

### Косвенная адресация через адресные регистры

При косвенной адресации посредством регистра (register indirect addressing) адрес указывается посредством одного из двух адресных регистров AR1 или AR2. Содержимое адресного регистра может быть внутризонным или межзонным указателем.



Примеры:

A I [AR1, P#2.1]

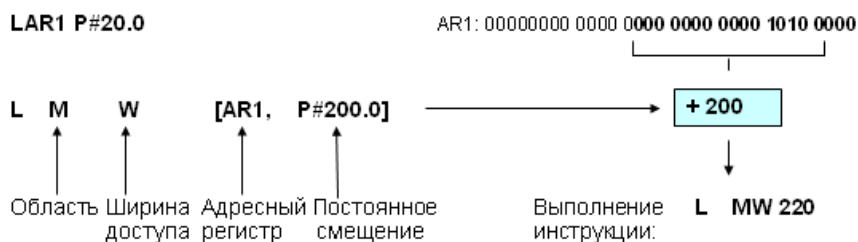
L MW [AR2, P#4.0]

A [AR1, P#2.1]

L W [AR2, P#4.0]

### Косвенная внутризонная адресация посредством регистра

При косвенной внутризонной адресации адрес операнда вычисляется как сумма содержимого адресного регистра и смещения в виде константы-указателя. При адресации числовых операндов смещение всегда должно иметь нулевой битовый адрес. Максимальное значение смещения составляет P#8191.7.

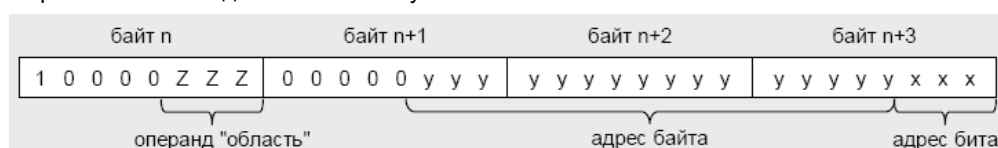


### Косвенная межзонная адресация посредством регистра

При косвенной межзонной адресации содержимое адресного регистра является межзонным указателем (area-crossing pointer).

Межзонный указатель содержит не только адрес области в формате адрес байта/адрес бита, но и указатель самой области и имеет вид: P#Zy.x, где Z – идентификатор области.

Формат константы для межзонного указателя:

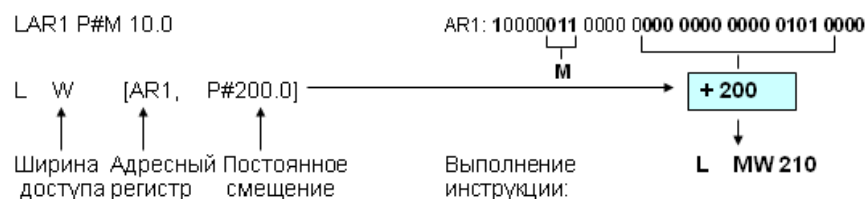


#### Кодирование адресной области Z:

|   |   |   |  |          |                 |
|---|---|---|--|----------|-----------------|
| 0 | 0 | 0 | Периферийные входы/выходы (P)                    | Примеры: | R# P 20.0       |
| 0 | 0 | 1 | Входы (I)  |          | R# I 20.1       |
| 0 | 1 | 0 | Выходы (Q)                                       |          | R# Q 20.2       |
| 0 | 1 | 1 | Меркеры (M)                                      |          | R# M 20. 4      |
| 1 | 0 | 0 | Глобальные данные (DBX)                          |          | R# DBX 20.0     |
| 1 | 0 | 1 | Экземплярные данные (DIX)                        |          | R# DIX 20.6     |
| 1 | 1 | 0 | Временные локальные данные (L)                   |          | R# L 20.0       |
| 1 | 1 | 1 | Временные локальные данные вызывающего блока (V) |          | Нет обозначения |

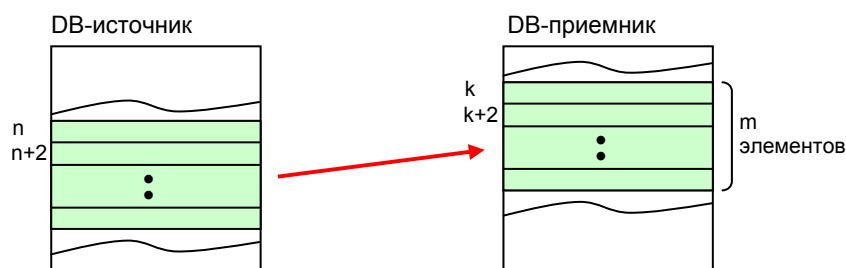
При косвенной межзонной адресации Вы задаете только идентификатор для спецификации ширины доступа: нет идентификатора для бита, "B" - для байта, "W" - для слова, "D" - для двойного слова.

Как и при внутризонной адресации, Вы используете в данном случае смещение P#y.x, которое прибавляется к значению адреса в адресном регистре.



#### Пример: копирование элементов из одного блока DB в другой DB

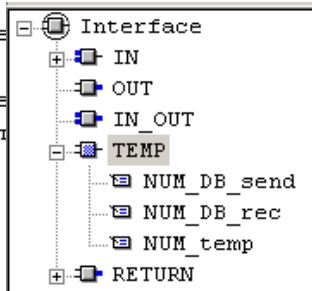
Пусть требуется создать блок FC10, выполняющий копирование элементов типа "WORD" из блока DB-источника в блок DB-приемник. Адреса блоков DB и количество элементов (m) передаются как параметры. Начальные адреса размещения областей памяти (адрес n, адрес k) также передаются в виде параметров.



Объявление входных параметров, временных переменных и текст программы FC10:

Contents Of: 'Environment\Interface\IN'

| Name      | Data | Comment            |
|-----------|------|--------------------|
| DB_send   | Int  | № блока источника  |
| ADDR_send | Int  | нач. адрес в блоке |
| DB_rec    | Int  | № блока приемника  |
| ADDR_rec  | Int  | нач. адрес в блоке |
| Number    | Int  | количество элемент |



FC10 : Title:

**Network 1:** Открытие блоков данных

```

L      #DB_send
T      #NUM_DB_send
OPN    DB [#NUM_DB_send]    // открыть блок DB-send

L      #DB_rec
T      #NUM_DB_rec
OPN    DI [#NUM_DB_rec]    // открыть блок DB-rec

```

**Network 2:** Подготовка к копированию

```

L      #ADDR_send           // чтение начального адреса
                           // области DB-send
SLD    3                   // преобразование адреса в формате
                           // указателя P#ADDR_send.0
LAR1                                // запись адреса в AR1

L      #ADDR_rec           // чтение начального адреса
                           // области DB-send
SLD    3                   // преобразование адреса в формате
                           // указателя P#ADDR_rec.0
LAR2                                // запись адреса в AR2

```

**Network 3:** цикл копирования

```

L      #Number             // чтение количества элементов
                           // для копирования
next: T      #NUM_temp      // запись в ячейку локального стека

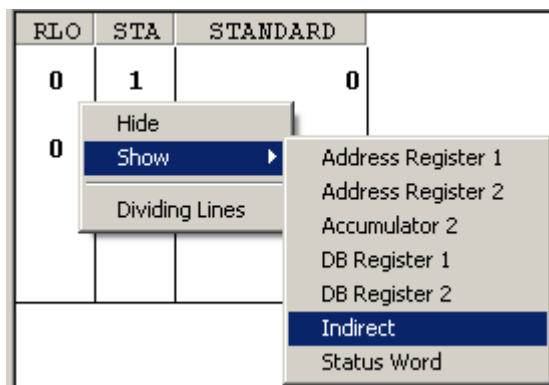
L      DBW [AR1,P#0.0]     // чтение ячейки из DB-send
T      DIW [AR2,P#0.0]     // запись в DB-rec

+AR1    P#2.0              // подготовка указателей
+AR2    P#2.0              // к следующему циклу

L      #NUM_temp           // проверка выхода из цикла
LOOP    next              // повторить цикл, если
                           // количество элементов <>0

```

## Тестирование программ с косвенной адресацией



По умолчанию в режиме "Monitor" отображаются признаки RLO, STA и значение ACCU 1 (STANDART).

С помощью функции "Show" Вы можете отобразить в формате указателей значение адресных регистров AR1, AR2 (Address Register 1/2), а также ячейки памяти, используемой для косвенной адресации (Indirect).

|                                |  | STANDARD | INDIRECT |
|--------------------------------|--|----------|----------|
| FC10 : Title:                  |  |          |          |
| Network 1: Открытие блоков дан |  |          |          |
| L #DB_send                     |  | 1        | --       |
| T #NUM_DB_send                 |  | 1        | --       |
| OPN DB [#NUM_DB_send]          |  | 1        | 1        |
| L #DB_rec                      |  | 2        | --       |
| T #NUM_DB_rec                  |  | 2        | --       |
| OPN DI [#NUM_DB_rec]           |  | 2        | 2        |

Здесь, колонка "INDIRECT" показывает, что выполняется косвенный доступ к открытию блоков DB1 и DB2.

|                             |  | AR 1 | AR 2 |
|-----------------------------|--|------|------|
| Network 3: цикл копирования |  |      |      |
| L #Number                   |  | 10.0 | 0.0  |
| next: T #NUM_temp           |  | 28.0 | 18.0 |
| L DBW [AR1,P#0.0]           |  | 28.0 | 18.0 |
| T DIW [AR2,P#0.0]           |  | 28.0 | 18.0 |
| +AR1 P#2.0                  |  | 30.0 | 18.0 |
| +AR2 P#2.0                  |  | 30.0 | 20.0 |
| L #NUM_temp                 |  | 30.0 | 20.0 |
| LOOP next                   |  | 30.0 | 20.0 |

Из колонок "AR1" и "AR2" видно, что перед выполнением цикла стартовыми адресами в блоках данных являются r#10.0 и r#0.0.

При значении адреса r#30.0 и r#20.0 происходит выход из цикла.

Примечание: для проверки выполнения действий внутри цикла необходимо включить пошаговый режим наблюдения.

## Расширенные средства диагностики и тестирования программ

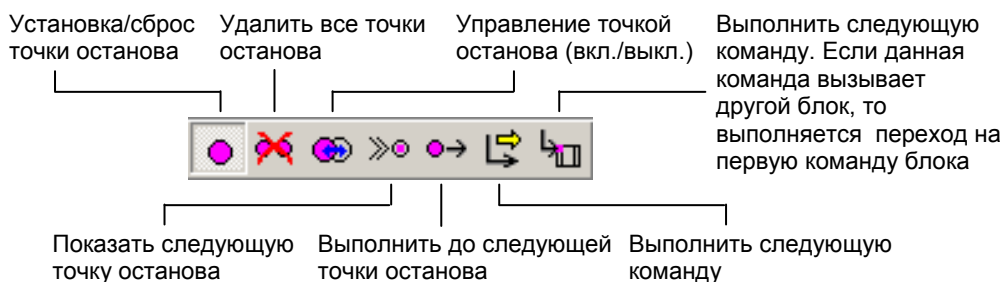
### Точки останова

Эта тестовая функция позволяет Вам тестировать вашу STL- программу шаг за шагом (покомандно). Вы можете установить несколько точек останова (в S7-300 не более 1). Для работы в этом режиме желательно включить отображение инструментальной панели через меню View -> Breakpoint Bar в редакторе LAD/STL/FBD.



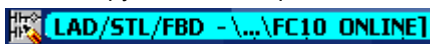
### Панель инструментов "Breakpoint Bar"

Все команды по управлению программой в пошаговом режиме можно выполнить с помощью кнопок, расположенных на панели "Breakpoint Bar".

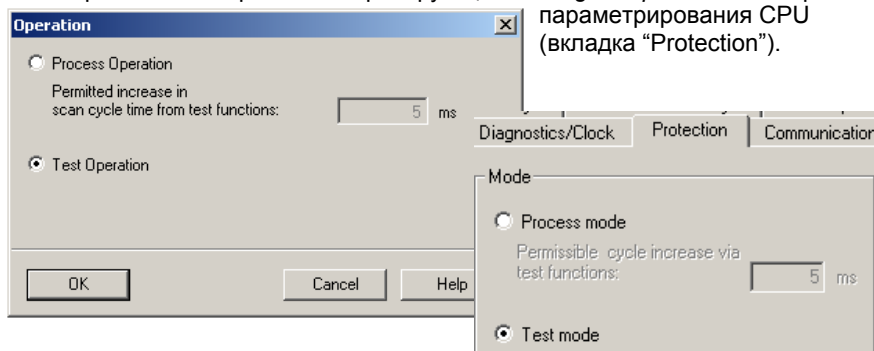


### Порядок работы в пошаговом режиме

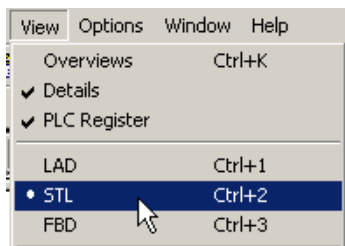
- 1) Открыть тестируемый блок в режиме online (*File -> Open ONLINE*).



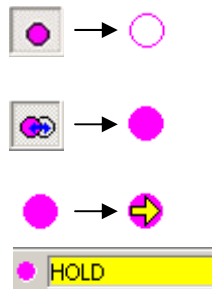
- 2) Установить режим "Test operation" через функцию *Debug -> Operation* или в режиме параметрирования CPU (вкладка "Protection").






- 3) Установить режим отображения "STL" (*View -> STL*).



- 4) Установить курсор на команду, на которой программа должна остановиться. Установить контрольную точку с помощью соответствующей кнопки или меню *Debug -> Set breakpoint*.
- 5) Активировать точку останова с помощью соответствующей кнопки или меню *Debug -> Breakpoints Active*.
- 6) Запустить CPU в режим RUN (или RUN-P). Когда программа достигнет точки останова, CPU перейдет в режим HOLD. На команде останова появится желтая стрелка. В окне "PLC register contents" отображаются значения регистров до выполнения данной команды.



7) Тестировать выполнение программы по одному из вариантов:

- продолжить выполнение программы до следующей точки останова (команда *Debug -> Resume* или кнопка  )
- выполнить текущую команду и остановиться на следующей команде (команда *Debug -> Execute Next Statement* или кнопка  ).  
Если текущей командой является переход на другой программный блок, то остановка происходит на команде, следующей за командой вызова блока
- выполнить текущую команду и остановиться на следующей команде (команда *Debug -> Execute Call* или кнопка  ).  
Если текущей командой является переход на другой программный блок, то остановка происходит на первой команде нового блока.

Рассмотрим пример тестирования цикла в блоке FC10.

Из рисунков ниже видно, что CPU остановился на команде проверки счетчика цикла (команда *LOOP next*). Следовательно, в окне регистров отображаются значения после выполнения предыдущей команды (*L# NUM\_temp* - чтение оставшихся элементов для копирования). Можно заметить, что следующий цикл копирования будет проводиться для пары элементов *DBW18 (AR1:= r#18.0)* и *DIW8 (AR1:= r#8.0)* из блоков данных *DB1* и *DB2 (GlobDB:=1, InstDB:=2)*.

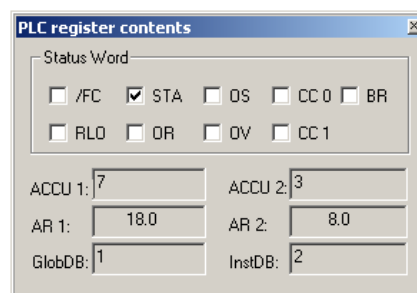
```

next: T      #NUM_temp

      L      DBW [AR1,P#0.0]
      T      DIW [AR2,P#0.0]



      +AR1   P#2.0
      +AR2   P#2.0

      L      #NUM_temp
      LOOP  next
  
```

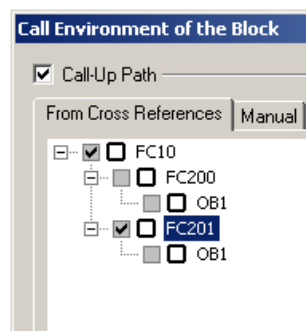


### Переход из пошагового режима в режим RUN

Для выхода из режима контрольных точек необходимо:

- Сбросить все контрольные точки по команде *Debug -> Delete All Breakpoints* или по кнопке .
- Выполнить команду *Debug -> Resume* или нажать на кнопку .

### Выбор режима тестирования через функцию "Call Environment"



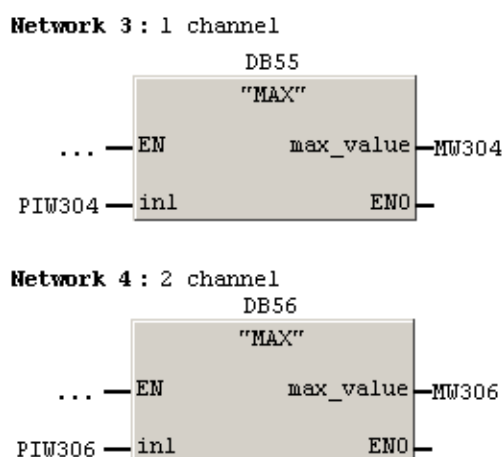
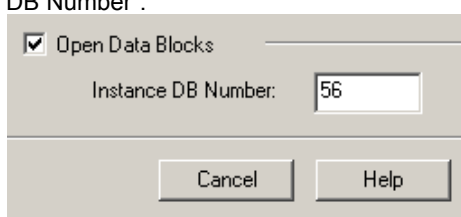
Команда меню *Debug -> Call Environment* позволяет задать условия для тестирования блока. Для активации этой функции должен быть включен режим "Test operation" (*Debug -> Operation*).

Данная функция используется тогда, когда блок вызывается в программе несколько раз, а Вы хотите контролировать только конкретный вызов.

Имеется 2 основных возможности для выбора точки наблюдения блока: 1) по указанному пути вызова блока (*Call-Up Path*) и 2) по открытому блоку данных (*Open Data Blocks*).

На рисунке слева приведен 1-й пример выбора пути для тестирования (только при вызове FC10 из блока FC201).

На рисунке справа приведен пример вызова блока FB (с именем "MAX") с разными параметрами. Чтобы тестировать FB с параметрами для 2-го вызова, необходимо воспользоваться условием "Open Data Blocks". Так как 2-й вызов блока FB выполняется с DB 56, то именно этот номер указан в поле "Instance DB Number".



### Режим "Force"

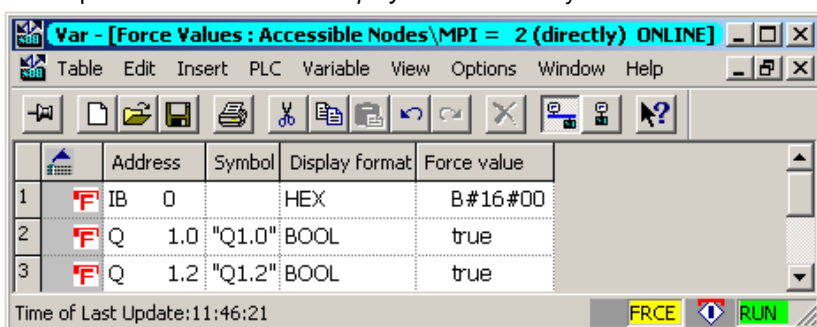
Режим "Force" может быть использован для принудительной установки переменных в заданное значение. За управление переменными в режиме "Force" отвечает операционная система CPU. Если режим "Force" установлен, то выключение питания на CPU не приводит к отмене такого управления.

! Необходимо помнить, что некорректное управление переменными в режиме "Force" может привести к повреждению оборудования, которым управляет CPU.

В S7-300 режим "Force" применяется только для I, Q областей памяти.

В S7-400 (и CPU 318) режим "Force" применяется не только для I, Q областей, но и для M (область флагов), а также PI и PQ (периферийные входы/выходы).

**Вызов режима "Force"** выполняется через online-функцию "PLC->Display Force Values" или через меню "Variable -> Display Force Values" утилиты "Monitor/Modify Variables".



В окне "Force Values" отображаются переменные, для которых задан режим принудительного управления. Вы можете изменить адреса переменных или их "форсированное" значение, а также добавить новые переменные.

**Активация режима "Force"**, а также актуализация всех проведенных изменений в таблице выполняется через функцию меню "Variable -> Force".

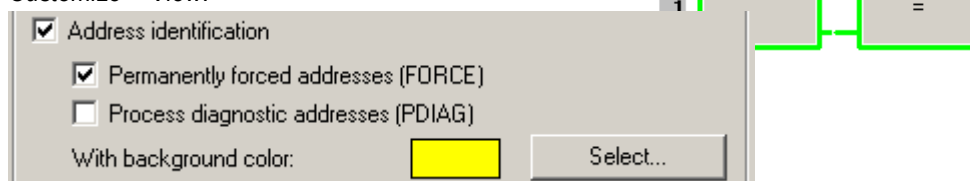
Если режим "Force" активен, то на CPU загорается желтый индикатор "FRCE".

**Отмена режима "Force"** возможна 2-мя способами:

- Через меню "Variable -> Stop Force" утилиты "Monitor/Modify Variables" (окно "Force Values" должно быть открыто!)
- Через сброс CPU.

В режиме "Monitor" редактор блоков отображает "форсированные" переменные со значком **F**.

Выбор фона для отображения адреса определяет пользователь в меню *Options -> Customize -> View*.



Примечание: если CPU не поддерживает режима "Force", то функция "Display Force Value" не может быть выполнена.

### Комплексные и параметрические типы данных

Тип данных – характеристика данных, которая определяет 1) множество значений, 2) допустимые операции, которые можно выполнять над этими значениями; 3) способ хранения этих значений в памяти.

В STEP 7 используются **простые, сложные и параметрические** типы данных.

К **простым** типам относятся данные, которые могут быть использованы в командах STEP 7. Длина простых типов не превышает 32 разряда.

Простые типы в STEP 7:

- Логический тип BOOL
- Числовые типы без знака: BYTE, WORD, DWORD
- Числовые типы знаковые: целочисленные INT, DINT и с плавающей точкой (вещественные) - REAL
- Символьный тип CHAR
- Типы для задания времени и даты: TIME, S5TIME, TOD, DATE

К **сложным** типам относятся данные, которые могут иметь длину более 32 бит. С помощью сложных типов удастся объединить простые данные в новую переменную.

Из-за нестандартной длины они не могут обрабатываться целиком одной командой STEP 7. Сложные типы данных используются для работы с переменными в блоках данных или в L-стеке, а также для работы с параметрами блока.

Сложные типы в STEP 7:

- Дата и время (8 байт) - DT. Пример: DT #2008-05-31-20:59:59.999
- Строка - STRING (до 254 символов). Пример: 'Simatic S7'
- Массив (набор компонентов одного типа данных) - ARRAY
- Структура (набор компонентов разного типа данных) – STRUCT

**Параметрические типы** - это типы данных, которые можно использовать только для параметров блоков FC или FB. К ним относятся:

- Таймеры и счетчики: TIMER, COUNTER
- Блок данных BLOCK\_DB
- Блоки FC и FB: BLOCK\_FC, BLOCK\_FB
- Указатель адреса POINTER
- Указатель области памяти ANY

## Тип данных DT (DATE\_AND\_TIME)

Тип DT используется для работы со временем суток и датой.

В STEP7 переменные типа DT занимают 8 байт памяти, все байты хранят информацию в BCD-коде и имеют следующий формат:

| байт | значение               |             | диапазон                |        |
|------|------------------------|-------------|-------------------------|--------|
| n    | год                    |             | 90-89 (от 1990 до 2089) |        |
| n+1  | месяц                  |             | 01 ... 12               |        |
| n+2  | день                   |             | 01 ... 31               |        |
| n+3  | час                    |             | 01 ... 23               |        |
| n+4  | минуты                 |             | 01 ... 59               |        |
| n+5  | секунды                |             | 01 ... 59               |        |
| n+6  | сотые миллисекунд      |             | 00 ... 99               |        |
| n+7  | единицы<br>миллисекунд | день недели | 0 ... 9                 | 1 .. 7 |
|      | 7                      | 4 3         | 0                       |        |

## Пример работы с данными типа DT

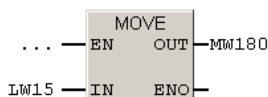
| Name           | Data Type     | Address |
|----------------|---------------|---------|
| OB1_EV_CLASS   | Byte          | 0.0     |
| OB1_SCAN_1     | Byte          | 1.0     |
| OB1_PRIORITY   | Byte          | 2.0     |
| OB1_OB_NUMBR   | Byte          | 3.0     |
| OB1_RESERVED_1 | Byte          | 4.0     |
| OB1_RESERVED_2 | Byte          | 5.0     |
| OB1_PREV_CYCLE | Int           | 6.0     |
| OB1_MIN_CYCLE  | Int           | 8.0     |
| OB1_MAX_CYCLE  | Int           | 10.0    |
| OB1_DATE_TIME  | Date_And_Time | 12.0    |

Все блоки OB вызываются операционной системой с набором данных в локальном стеке блока.

Дата вызова OB всегда записывается в локальный стек по адресам LB 12 - LB 19 в формате DT.

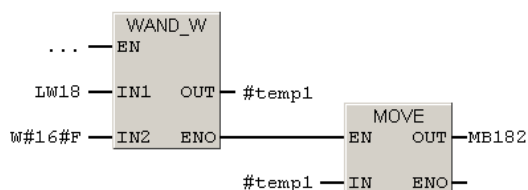
Чтение локальной переменной OB1\_DATE\_TIME в блоке OB1 позволит всегда иметь оперативную информацию о времени, которую можно отобразить, например, на панели оператора.

### Network 5: Чтение текущего часа и минут



Переменная LW15 содержит 2 байта с информацией о часе (LB15) и минутах (LB16).

### Network 6: Чтение дня недели (1-воскресенье)



4 младших разряда LW18 содержат информацию о дне недели. Маскирование старших разрядов с помощью операции "WAND" позволяет отделить день недели от миллисекунд.

Для обработки переменных типа DT можно использовать IEC-функции из стандартной библиотеки STEP 7: (FC1, FC3, FC6, FC7, FC8, FC9, FC12, FC14, FC34, FC35 и другие).



## Тип данных STRING

Данные типа STRING - символьная строка, в которой может быть до 254 символов.

| Address | Name   | Type       | Initial value |
|---------|--------|------------|---------------|
| 0.0     |        | STRUCT     |               |
| +0.0    | recipe | STRING[10] | 'Юнона'       |
| =12.0   |        | END_STRUCT |               |

Размер строки может быть ограничен числом, задаваемым при объявлении переменной.

Формат переменных типа STRING:

|            |               |                  |              |
|------------|---------------|------------------|--------------|
| Байт n     | Мак длина     | (k)              |              |
| Байт n+1   | Текущая длина | (m)              |              |
| Байт n+2   | 1-й символ    | Текущая<br>длина | Мак<br>длина |
| Байт n+3   | 2-й символ    |                  |              |
| Байт ...   | ...           |                  |              |
| Байт n+m+1 | m-й символ    |                  |              |
| Байт ...   | ...           |                  |              |
| Байт n+k+1 | ...           |                  |              |

Размещение строк в памяти выполняется, начиная с байта с четным адресом (на границе слов).

Как видно из рисунка, при размещении строки в памяти выделяется на 2 байта больше, чем число символов в строке.

Строка символов в формате ASCII располагается, начиная с 3-го по счету байта

Для обработки STRING-переменных можно использовать IEC-функции из библиотеки STEP 7: (FC2, FC4, FC10, FC11, FC13, FC15, FC17, FC19, FC20, FC21 и другие).

## Тип данных ARRAY

Данные типа ARRAY представляет собой массив, содержащий в себе комбинацию фиксированного числа переменных одинакового типа (в том числе и сложных, кроме массива).

| Address | Name  | Type         | Initial value      |
|---------|-------|--------------|--------------------|
| 0.0     |       | STRUCT       |                    |
| +0.0    | speed | ARRAY[1..10] | 10, 20, 30, 7 (50) |
| +2.0    |       | INT          |                    |
| =20.0   |       | END_STRUCT   |                    |

При задании массива указывается его размерность в виде нижнего и верхнего предела (индекса). Индексы задаются целыми числами типа INT.

Пример инициализации элементов массива:

5 (5,20) - следующие 10 компонентов (5x2) последовательно инициализированы значениями 5,20.

Массив может быть многомерным, например, ARRAY[x1..x2, y1..y2, z1..z2].

Максимальное число размерностей для массива составляет 6.

Доступ к элементу массива обеспечивается по имени массива и индексу в квадратных скобках, например, L speed [3,2].

Размещение массива в памяти выполняется на границе слов.

В многомерных массивах элементы хранятся в порядке следования размерностей, начиная с первой размерности.

| Address | Name  | Type               | Initial value |
|---------|-------|--------------------|---------------|
| 0.0     |       | STRUCT             |               |
| +0.0    | speed | ARRAY[1..10, 1..2] | 20 (50)       |
| +2.0    |       | INT                |               |
| =40.0   |       | END_STRUCT         |               |

| Address | Name        | Type | Initial | Actual v |
|---------|-------------|------|---------|----------|
| 0.0     | speed[1, 1] | INT  | 50      | 50       |
| 2.0     | speed[1, 2] | INT  | 50      | 50       |
| 4.0     | speed[2, 1] | INT  | 50      | 50       |
| 6.0     | speed[2, 2] | INT  | 50      | 50       |
| 8.0     | speed[3, 1] | INT  | 50      | 50       |
| 10.0    | speed[3, 2] | INT  | 50      | 50       |
| 12.0    | speed[4, 1] | INT  | 50      | 50       |
| 14.0    | speed[4, 2] | INT  | 50      | 50       |

Точное место хранения в памяти переменных ARRAY необходимо знать тогда, когда требуется обращение индивидуальным элементам с помощью косвенной адресации. Если переменная ARRAY хранится в блоке данных, то режим отображения через функцию View->Data View дает наглядное представление об адресах элементов массива.

### Тип данных STRUCT

Данные типа STRUCT представляет собой структуру данных, состоящую из фиксированного числа компонентов, которые могут относиться к различным типам данных.

|      |       |            |         |
|------|-------|------------|---------|
| +6.0 | MOTOR | STRUCT     |         |
| +0.0 | ON    | BOOL       | FALSE   |
| +0.1 | OFF   | BOOL       | FALSE   |
| +2.0 | Delay | S5TIME     | S5T#0MS |
| +4.0 | Speed | INT        | 0       |
| =6.0 |       | END_STRUCT |         |

На рисунке переменная "MOTOR" объединяет 4 различных переменных под одним именем.

Структура, как набор переменных, создается пользователем для наглядного доступа к данным, объединенных общим признаком (в примере, все 4 переменных используются для управления мотором).

|       |          |           |          |
|-------|----------|-----------|----------|
| +12.0 | Meas     | STRUCT    |          |
| +0.0  | Duaraion | TIME      | T#0MS    |
| +4.0  | Volume1  | STRUCT    |          |
| +0.0  | Width1   | INT       | 0        |
| +2.0  | Length   | INT       | 0        |
| +4.0  | Height   | INT       | 0        |
| +6.0  | MeasTi   | TIME_OF_D | TOD#0:0: |
| =10.0 |          | END_STRUC |          |

Применение структур обеспечивает компактный способ передачи параметров: множество элементов могут быть переданы в одном параметре.

Структура может быть вложенной. Вложенная структура - это такая структура, которая сама является компонентом другой структуры. Глубина вложения для структур может достигать 6.

Доступ к компоненту структуры обеспечивается с использованием имени структуры и имени компонента, разделенных точкой.

Например, L DB10.MOTOR.Speed или L DB10.Meas.Volume1.Lenght.

### Передача параметров сложного типа

**Network 1:** Доступ к элементам

```

L    #speed[1]
A    #motor.on

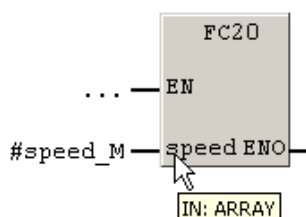
```

Прямой доступ к сложным параметрам типа ARRAY или STRUCT внутри блоков FC/FB реализуется "покомпонентном" режиме.

FB20 : Title:

**Network 1:** Сквозная передача массива

В блоках FB (не в FC!) Вы можете последовательно передавать параметры типов ARRAY и STRUCT в параметры вызываемых блоков.



## Тип данных POINTER

Для параметров блоков FC/FB может быть объявлен тип POINTER (указатель адреса). Это означает, что через данный параметр блок получает начальный адрес области памяти (а не содержимое переменной). При назначении актуальных параметров типу POINTER задается либо адрес или символьное имя переменной, либо межзонный указатель в формате P#Zy.x.

Примеры формата указателя для адресации данных:

- начинающихся с M 40.0: P#M40.0 ; M40.0 ; MW40 ; "START\_VALUE" (имя MD40).
- размещенных в DB1: P#DB1.DBX20.0 ; DB1. speed (speed – переменная в DB1)

### Формат типа POINTER

|          |                         |
|----------|-------------------------|
| Байт n   | Номер<br>блока данных   |
| Байт n+1 |                         |
| Байт n+2 | Указатель<br>на область |
| Байт n+3 |                         |
| Байт n+4 |                         |
| Байт n+5 |                         |

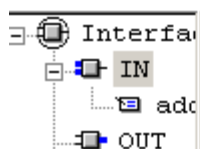
Параметр POINTER занимает в памяти 6 байт.

Номер блока данных содержится в первых двух байтах. Далее в 4 байтах хранится межзонный указатель в формате P#Zy.x.

Если межзонный указатель ссылается на области P, I, Q, M, L (не на блок данных), то номер DB равен 0.

Параметры типа POINTER в блоках FC/FB удобны для обработки различных, но однотипных областей памяти (одинакового типа и размера) нестандартной длины. Тогда одна процедура может быть использована для работы с разными данными.

В качестве примера приведен фрагмент программы замены первого символа у строки на символ '?' (в блоке FC1).



| Name    | Data Type |
|---------|-----------|
| address | Pointer   |

Address - параметр блока типа POINTER

|      |                 |  |
|------|-----------------|--|
| L    | P##address      | // чтение указателя на адрес параметра |
| LAR1 |                 | // загрузка указателя в AR1            |
| L    | W [AR1,P#0.0]   | // чтение номера DB                    |
| T    | #DB_number      | // передача в локальную переменную     |
| OPN  | DB [#DB_number] | // открытие доступа к DB               |
| L    | D [AR1,P#2.0]   | // чтение межзонного указателя         |
| LAR1 |                 | // загрузка указателя в AR1            |
| L    | '?'             | // загрузка символа '?'                |
| T    | B [AR1,P#2.0]   | // передача символа в 1 позицию строки |

### Network 1: Обработка строки

|      |                    |   |
|------|--------------------|---|
| CALL | FC                 | 1 |
|      | address:=DB1.names |   |

// вызов блока FC1 для передачи адреса строки с именем "names".



## Тип данных ANY

Вы можете применять тип ANY в качестве параметра блока, если необходимо задать область памяти. При назначении актуальных параметров типу ANY задается либо символьное имя переменной, либо константа в формате P#[Data Block]Zy.x Type Size. Константа для параметра ANY называется «указатель ANY» и используется в том случае, когда область данных не связана с переменной.

Примеры констант для адресации данных в формате ANY:

- P#DB10.DBX 30.0 INT 22 (Type – INT; Size – 22) - область из 22 чисел в формате INT в блоке DB10, начиная с DBB 30;
- P#I20.0 WORD 1 - слово входов IW 20;

| Указатель ANY<br>для типов данных |                    | Формат ANY для типов данных   |
|-----------------------------------|--------------------|---|
| Байт n                            | 16#10              | Для элементарных и составных типов данных формат ANY занимает в памяти 10 байт и содержит: <ul style="list-style-type: none"> <li>• 10h – идентификатор для S7.</li> <li>• Тип данных в соответствии с таблицей кодов для типов данных.</li> <li>• Размер области в виде коэффициента повторения элементов выбранного типа.</li> <li>• Номер DB (или 0, если адрес не связан с блоком данных).</li> <li>• Начальный адрес области данных в формате межзонного указателя.</li> </ul> |
| Байт n+1                          | Тип                |   |
| Байт n+2                          | Размер             |   |
| Байт n+3                          |                    |   |
| Байт n+4                          | Номер блока данных |   |
| Байт n+5                          |                    | Для массивов и структур, используемых в качестве параметра ANY, редактор STEP7 создает указатель ANY на область с типом BYTE с коэффициент повторения, равному количеству байт (исключение: для массива символов создается указатель с типом CHAR). -   |
| Байт n+6                          | Указатель          |   |
| Байт n+7                          | на область         |   |
| Байт n+8                          |                    |   |
| Байт n+9                          |                    |   |

## Простые и сложные типы, используемые с ANY-указателем:

| Код | Тип  | Код | Тип   | Код | Тип    |
|-----|------|-----|-------|-----|--------|
| 01  | BOOL | 06  | DWORD | 0B  | TIME   |
| 02  | BYTE | 07  | DINT  | 0C  | S5TIME |
| 03  | CHAR | 08  | REAL  | 0E  | DT     |
| 04  | WORD | 09  | DATE  | 13  | STRING |
| 05  | INT  | 0A  | TOD   | 0   | VOID   |

Код 0 (VOID) используется при использовании пустого указателя, имеющего вид: P#P 0.0 VOID 0.

Если для блока FB объявляется параметр с типом ANY, то в экземплярный блок DB записывается именно такой указатель в качестве начального значения (Initial value).



Тип ANY можно указывать для временных локальных данных блоков FC/FB.

В этом случае для переменной резервируется 10-байтная область для размещения указателя ANY. После занесения в эту область соответствующей информации, локальные переменные можно использовать в качестве входных параметров ANY.

Пример инициализации ANY-переменной в локальном стеке по адресу P#L0.0 :

```
L W#16#1002
T LW 0 . // запись ID и типа (BYTE)
L 26
T LW 2 // запись фактора повторения (размер области 26 байт)
L 60
T LW 4 // запись номера DB (=60)
L P#DBB0.0 // подготовка указателя адреса
T LD 6 // запись указателя адреса
```

После выполненного фрагмента программы переменная в локальном стеке будет соответствовать константе P#DB60.DBX0.0 BYTE 26.

## Пример программы подсчета контрольной суммы

|         |               |                                       |
|---------|---------------|---------------------------------------|
| L       | P##Area       | // чтение указателя на ANY-параметр   |
| LAR1    |               | // загрузка указателя в AR1           |
| L       | B [AR1,P#1.0] | // чтение типа данных                 |
| L       | 2             | // проверка на тип BYTE               |
| ==I     |               |                                       |
| JNB     | err           | // переход, если тип не BYTE          |
| L       | W [AR1,P#2.0] | // чтение фактора повторения          |
| T       | #quantity     | // передача в локальную переменную    |
| L       | W [AR1,P#4.0] | // чтение номера DB                   |
| T       | #num_DB       | // передача в локальную переменную    |
| OPN     | DB [#num_DB]  | // открытие доступа к DB              |
| L       | D [AR1,P#6.0] | // чтение межзонного указателя        |
| LAR1    |               | // загрузка указателя в AR1           |
| L       | 0             | // обнуление временной суммы          |
| T       | #sum_temp     |                                       |
| L       | #quantity     | // чтение фактора повторения          |
| next: T | #quantity     | // передача в локальную переменную    |
| L       | B [AR1,P#0.0] | // чтение байта из области данных     |
| L       | #sum_temp     | // сложение с временной суммой        |
| +I      |               |                                       |
| T       | #sum_temp     |                                       |
| +AR1    | P#1.0         | // подготовка указателя               |
| L       | #quantity     | // чтение фактора повторения          |
| LOOP    | next          | // проверка на выход из цикла         |
| L       | #sum_temp     | // передача результата в параметр SUM |
| T       | #SUM          |                                       |
| BE      |               | // конец блока                        |
| err: L  | 0             | // обработка ошибки                   |
| T       | #SUM          |                                       |

#### Комментарий к программе.

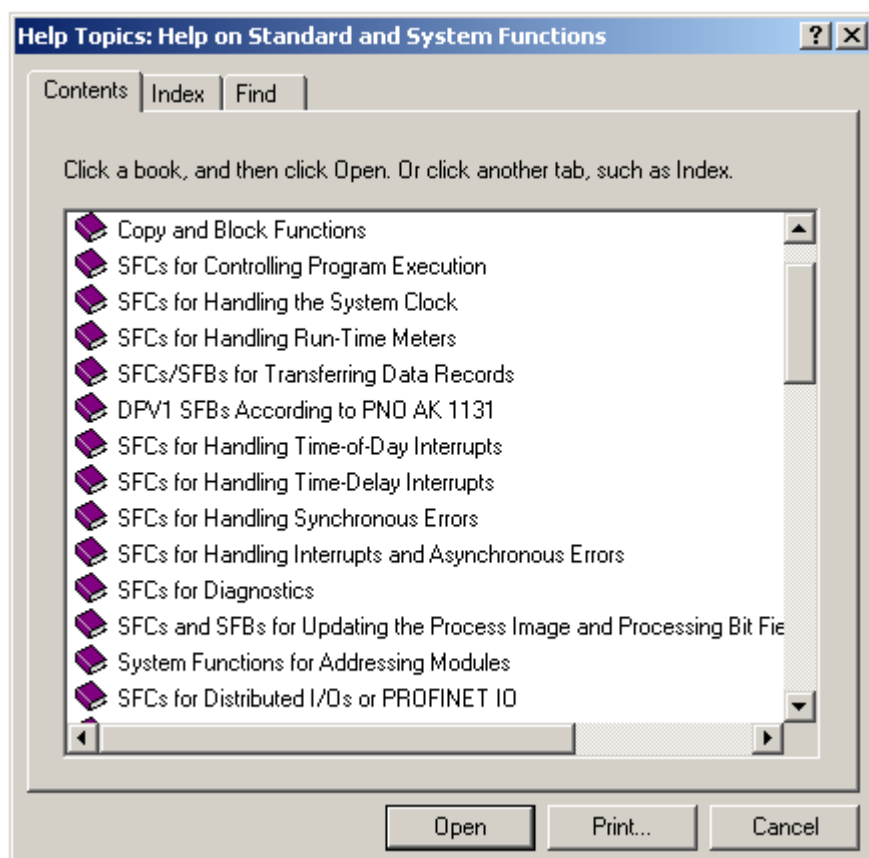
Контрольная сумма образуется путем простого сложения всех байтов области памяти без учета переполнения. Область памяти, для которой проводится подсчет суммы, задается через параметр "Area" в формате ANY. Если указатель ANY не указывает на тип BYTE, то контрольная сумма сбрасывается в 0.

### Системные блоки

Системные функции представляют собой часто используемые стандартные функции и функции, которые не могут быть выполнены инструкциями STEP 7 (как например, генерирование DB). Системные функции интегрированы в операционную систему CPU в виде блоков SFC и SFB. Системные блоки вызываются в программе пользователя инструкциями CALL SFC или CALL SFB. Для блоков SFB также необходимы связанные блоки данных.

Возможность использования системной функции зависит от типа CPU. Список доступных для CPU системных функций отображается через online-функцию *PLC->Module Information ->Performance Data*.

Все системные функции объединены по функциональным группам. Окно "Help on Standard and System Functions" показывает список групп. Детальное описание приводится в руководстве "System Software for S7-300 and S7-400, System and Standard Functions"



### Краткий обзор системных функций

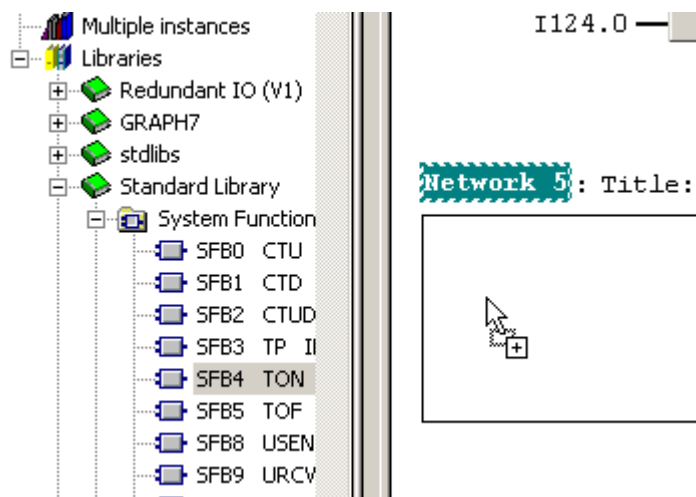
|  |  |
|--|--|
| Функции копирования и блоковые функции | <p><b>SFC 20</b> - копирует содержимое области памяти.</p> <p><b>SFC 21</b> - заполняет область памяти содержимым другой области.</p> <p><b>SFC 22</b> - создает блок данных в рабочей памяти.</p> <p><b>SFC 23</b> - удаляет блок данных в рабочей памяти. DB не должен быть открыт.</p> <p><b>SFC 24</b> - определяет имеется ли DB-блок в рабочей или загружаемой памяти, длину блока в байтах и защищен ли он от записи.</p> <p><b>SFC 25</b> - производит сжатие рабочей памяти. Если блоки корректировались, то занимаемые ими пространства в рабочей памяти ("дыры") удаляются.</p> <p><b>SFC 44</b> - передает при ошибках доступа в ACCU 1 заданное значение, с которым продолжается обработка основной программы (вызов только из OB 122 или OB121).</p> |
| Управление программой                  | <p><b>SFC 43</b> - заново запускает контроль времени цикла CPU.</p> <p><b>SFC 46</b> - переводит CPU в состояние останова (STOP).</p> <p><b>SFC 47</b> - вводит задержку времени от 1 до 32767 <math>\mu</math>s.</p> <p><b>SFC 35</b> - запускает синхронно OB 60 во всех CPU (для мультикомпьютинга).</p>  |
| Управление временем                    | <p><b>SFC 0</b> - устанавливает новую дату и время в CPU.</p> <p><b>SFC 1</b> - считывает текущую дату и время из CPU.</p> <p><b>SFC 48</b> - синхронизирует часы ведомых устройств по часам ведущего CPU (запуск только для ведущего CPU).</p>  |
| Управление счетчиками времени          | <p><b>SFC 2</b> - устанавливает счетчик рабочего времени (Run-Time Meter) на заданное значение (максимум. 32767). Счетчики рабочего времени хранятся в энергонезависимой памяти.</p> <p><b>SFC 3</b> - запускает /останавливает счетчик рабочего времени. После запуска счетчика операционная система CPU увеличивает счетчик на 1 через 1 час.</p> <p><b>SFC 4</b> - считывает текущее время из счетчика и его статус.</p> <p><b>SFC 64</b> - считывает системное время ((time tick) CPU).</p> <p><i>Системное время</i> - это автономный счетчик времени, который увеличивается каждые 10 ms (для S7-300) или 1 ms (для S7-400). Максимальное значение счетчика 2147483647, после которого счет снова начинается с 0.</p>  |
| Передача наборов данных                | <p><b>SFC 58</b> - передает набор данных "RECORD" в модуль I/O.</p> <p><b>SFC 59</b> - считывает набор данных "RECORD" в модуль I/O.</p> <p><b>SFC 54</b> - считывает сконфигурированные параметры модуля из системных данных</p> <p><b>SFC 55</b> - передает набор параметров модулю</p> <p><b>SFC 57</b> - передает все параметры модулю</p>   |
| Прерывания по времени                  | <p><b>SFC 28</b> - устанавливает новую дату и время для запуска прерывания (OB10-OB17).</p> <p><b>SFC 29</b> - отменяет прерывание по времени (OB10-OB17).</p> <p><b>SFC 30</b> - разрешает прерывание (OB10-OB17) в установленное время.</p> <p><b>SFC 31</b> - считывает информацию о статусе прерывания (загружен OB, активирован, ...)</p>   |

|   |  |
|---|--|
| Прерывания с задержкой времени                | Прерывания с задержкой (OB 20 - OB 23) вызываются спустя время после того как событие произошло.<br><b>SFC 32</b> - активирует отсчет времени для запуска прерывания.<br><b>SFC 33</b> - отменяет прерывание с задержкой.<br><b>SFC 34</b> - считывает статус прерывания (OB 20 - OB 23).  |
| Управление синхронными ошибками               | <b>SFC 36</b> - маскирование синхронных ошибок.<br><b>SFC 37</b> - отмена маскирования синхронных ошибок.<br><b>SFC 38</b> - чтение регистра ошибок (наличие ошибок, тип).   |
| Управление прерываниями                       | <b>SFC 39</b> - блокировка прерываний и асинхронных ошибок.<br><b>SFC 40</b> - деблокировка прерываний.<br><b>SFC 41</b> - задержка обработки новых прерываний.<br><b>SFC 42</b> - разрешение прерываний с высшим приоритетом  |
| Системная диагностика                         | <b>SFC 6</b> - чтение стартовой информации блока OB.<br><b>SFC 51</b> - чтение списка данных о состоянии системы (системные данные, данные диагностики, диагностический буфер).<br><b>SFC 52</b> - запись в диагностический буфер.   |
| Отображение процесса для области I/O          | <b>SFC 26</b> - актуализация области PII.<br><b>SFC 27</b> - актуализация области PIQ.<br><b>SFC 79</b> - установка в 1 битового массива для периферии PQ.<br><b>SFC 80</b> - установка в 0 битового массива для периферии PQ.   |
| Адресация модулей                             | <b>SFC 5</b> - определение логического адреса модуля<br><b>SFC 49</b> - определение слота модуля по логическому адресу.<br><b>SFC 50</b> - определение всех логических адресов модуля.   |
| Децентрализованная периферия                  | <b>SFC 7</b> - запускает процесс прерываний (OB40) на станции-мастере от интеллектуальной станции -slave (например, CPU 315-2DP).<br><b>SFC 11</b> - синхронизирует одну или несколько групп DP-slaves (поддержка режимов SYNC и FREEZE)<br><b>SFC 13</b> - считывает данные диагностики от DP slave.<br><b>SFC 14</b> - считывает консистентные данные (неделимая информация, как одно целое) из DP slave.<br><b>SFC 15</b> - записывает консистентные данные в DP slave.   |
| Передача глобальных данных                    | Глобальные данные передаются циклически (по умолчанию каждый 8-ой цикл) без использования SFC.<br><b>SFC 60</b> - передача GD пакета.<br><b>SFC 61</b> - прием GD пакета.  |
| Неконфигурируемые коммуникации (по MPI -сети) | <b>SFC 65</b> - передача данных партнеру по коммуникации (двусторонний обмен)<br><b>SFC 66</b> - прием данных от партнера по коммуникации (двусторонний обмен)<br><b>SFC 67</b> - прием данных от партнера (односторонний обмен)<br><b>SFC 68</b> - передача данных партнеру (односторонний обмен)<br><b>SFC 69</b> - отмена соединения<br><b>SFC 72</b> - прием данных от партнера внутри S7-станции<br><b>SFC 73</b> - передача данных партнеру внутри S7-станции<br><b>SFC 74</b> - отмена соединения внутри S7-станции |



|   |  |
|---|--|
| Конфигурируемые S7-коммуникации (по сети MPI, Profibus, Ethernet) | <p><b>SFB 8</b> - передача данных партнеру по коммуникации (двусторонний обмен, без подтверждения).</p> <p><b>SFB 9</b> - прием данных от партнера по коммуникации (двусторонний обмен, асинхронный прием).</p> <p><b>SFB 12</b> - передача сегментированных данных (двусторонний обмен, с подтверждением).</p> <p><b>SFB 13</b> - прием сегментированных данных (двусторонний обмен, с подтверждением).</p> <p><b>SFB 14</b> - прием данных от партнера по коммуникации (односторонний обмен, прием с подтверждением).</p> <p><b>SFB 15</b> - передача данных партнеру по коммуникации (односторонний обмен, с подтверждением).</p> <p><b>SFB 16</b> - передача данных на принтер.</p> <p><b>SFB 19</b> - выполнение рестарта в удаленном CPU.</p> <p><b>SFB 20</b> - перевод в стоп удаленный CPU.</p> <p><b>SFB 21</b> - выполнение горячего рестарта в удаленном CPU.</p> <p><b>SFB 22</b> - запрос состояния удаленного партнера</p> <p><b>SFB 23</b> - прием состояния удаленного CPU.</p> |
| IEC - счетчик и IEC таймер  | <p>Эти блоки реализуют функции времени и счетчика в соответствии со стандартом IEC 1131-3.</p> <p><b>SFB 0</b> - прямой счет счетчика.</p> <p><b>SFB 1</b> - обратный счет счетчика.</p> <p><b>SFB 2</b> - реверсивное управление счетчиком.</p> <p><b>SFB 3</b> - генерирование импульса.</p> <p><b>SFB 4</b> - задержка включения.</p> <p><b>SFB 5</b> - задержка выключения.</p>  |
| Интегрированные функции (для CPU31xC и CPU 31xIFM).               | <p><b>SFB 41</b> - непрерывное регулирование.</p> <p><b>SFB 42</b> - шаговое регулирование.</p> <p><b>SFB 43</b> - частотно- импульсное регулирование.</p> <p><b>SFB 44</b> - позиционирование с аналоговым выходом</p> <p><b>SFB 46</b> - позиционирование с дискретным выходом</p> <p><b>SFB 47</b> - управление счетчиками быстрого счета.</p> <p><b>SFB 48</b> - измерение частоты.</p> <p><b>SFB 48</b> -управление импульсом: широтно-импульсная модуляция.</p>  |
| Управление сообщениями  | <p><b>SFB 36</b> - передача сообщения, связанного с блоком, без квитирования.</p> <p><b>SFB 31</b> - передача до 8 сообщений, связанных с блоком, без квитирования.</p> <p><b>SFB 33</b> - передача сообщения, связанного с блоком, с квитированием.</p> <p><b>SFB 35</b> - передача до 8 сообщений, связанных с блоком, с квитированием.</p> <p><b>SFB 34</b> - передача до 8 сообщений, связанных с блоком, с квитированием (без передачи значений переменных).</p> <p><b>SFB 37</b> - передача архива.</p> <p><b>SFC 17 / SFC 18</b> - передача сообщения, связанного с блоком, с квитированием</p>   |

## Вызов системных функций и системных функциональных блоков



I124.0 —

**Network 5:** Title:

В режиме отображения FBD/LAD выбрать в каталоге элементов группу "Libraries" (Библиотеки) и библиотеку "Standard Library".  
Кликнуть мышью по выбранному блоку или буксировать блок в выделенный сегмент.

В режиме STL вызов системного блока выполняется через инструкцию CALL с указанием блока.

Когда в редакторе блоков выполняется вызов системной функции в первый раз, то она автоматически копируется в программу пользователя. В дальнейшем, вызов ее можно выполнять аналогично вызову FC/FB.

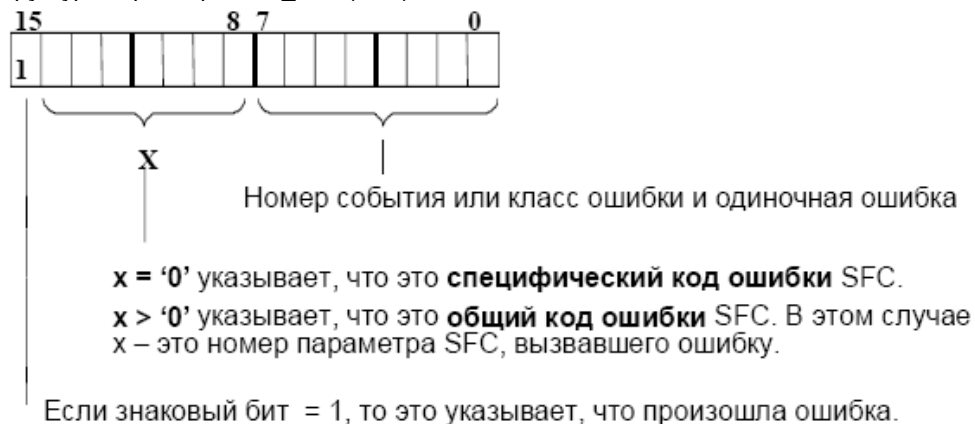
## Ошибки выполнения системной функции

Системная функция через нулевое значение бита BR слова состояния показывает, что при исполнении функции произошла ошибка.

Возвращаемое значение (RET\_VAL) системной функции предоставляет в распоряжение один из двух следующих типов кодов ошибки:

- общий код ошибки, относящийся к ошибкам, которые могут возникнуть в любой системной функции
- специфический код ошибки, который относится только к конкретной системной функции.

Структура параметра RET\_VAL (INT):





## Примеры использования системных функций

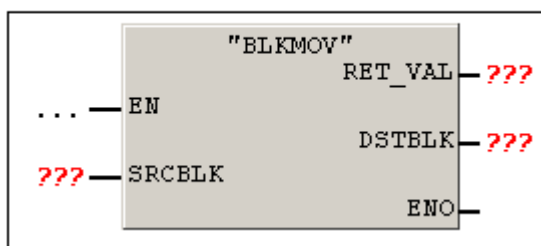
### Установка часов реального времени (SFC 0)

**Network 7:** set clock in CPU

```
CALL  "SET_CLK"           // вызов функции
PDT   :=#DATE_TIME       // выходной параметр DT
RET_VAL:=#retval
```

### Копирование DB из загрузочной памяти в рабочую память (SFC 20)

**Network 5:** Copy



Входные параметры:

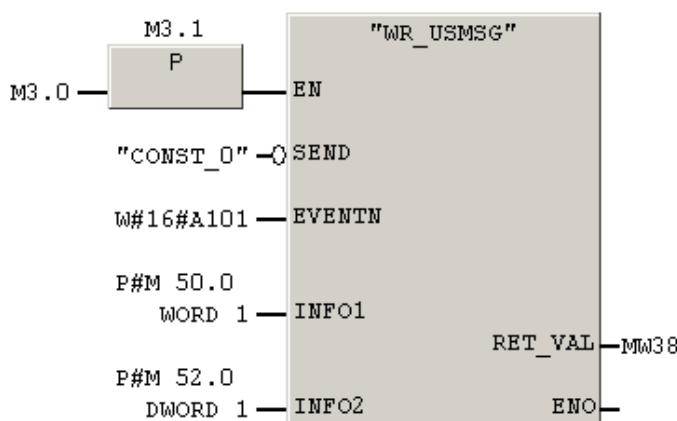
**SRCBLK** (тип ANY) - область памяти для передачи (I,Q,M,D,L). Блоки DB с атрибутом UNLINKED также могут быть указаны в качестве источника данных.

Выходные параметры

**DSTBLK** (тип ANY) - область памяти для приема (I,Q,M,D,L).

### Запись сообщения в диагностический буфер (SFC 52)

**Network 8:** Запись в диагн. буфер (прих. событие)



Входные параметры:

**SEND** (тип BOOL) - разрешение отправки на PG,OP

**EVENTIN** (WORD) - идентификатор события

**INFO1** (ANY) - 1-й параметр для записи в буфер (форматы WORD, INT, ARRAY[0..1] OF CHAR)

**INFO2** (ANY) - 2-й параметр для записи в буфер (форматы DINT, DWORD, REAL TIME, ARRAY[0..3] OF CHAR).

Выходные параметры

**RET\_VAL** (тип INT) - информация об ошибке.

Допустимые значения для идентификатора события: W#16#Axyz, W#16#Bxyz (определяемые пользователем тексты в утилите CPU Messages) или W#16#8xyz, W#16#9xyz (предопределенные тексты, например, W#16#9001 - Automatic mode). Список текстов приведен в документации по системным функциям.

Примечание:

Для включения в тест сообщения информации о параметрах INFO1 и INFO2 введите текст следующего типа: @< No. of associated value ><Format code>@.

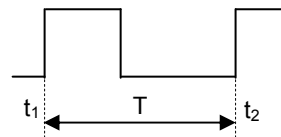
Пример сообщения: Температура превышена! T=@1W%5d@°. Максимум не должен превышать T=@2W%4d@°.



Для чтения диагностического буфера используется функция SFC51 с входными параметрами SZL\_ID :=W#16#01A0, INDEX:= количество записей (от 1 до 100). Размер приемного буфера (параметр DR) задается из условия: 20 байт на каждую запись.

### Определение интервала времени (SFC 64)

Пусть необходимо определить значение времени  $T$ , прошедшее с момента установки бита в 1 и его повторной установки в 1.



**SFC 64 "TIME\_TCK"** позволяет прочитать счетчик системного времени CPU, который считает циклически от 0 до 2147483647 мс (разрешение составляют 1 мс для S7-400 и CPU 318 и 10 мс для всех остальных CPU S7-300).

Период сигнала получим путем вычитания  $t_2 - t_1$ , т.е. разницы между 2 результатами чтения системного времени.

Ниже приведена программа вычисления периода в формате sec.ms для блока FB2.

FB2 : Вычисление периода

#### Network 1: Выделение + фронта

```

A      #event
FP     #help_edge
=      #pos_edge
AN     #pos_edge
BEC

// event - входной параметр (BOOL)
// help_edge - стат. переменная (BOOL)
// pos_edge - лок. переменная (BOOL)

```

#### Network 2 : Вычисление T

```

// чтение систем. времени
// curent_tick - лок. переменная (Time)
CALL  "TIME_TCK"
RET_VAL:=#curent_tick

// вычисление разницы отсчетов времени
// prev_time - стат. переменная (DINT)
L      #curent_tick
L      #prev_time
-D

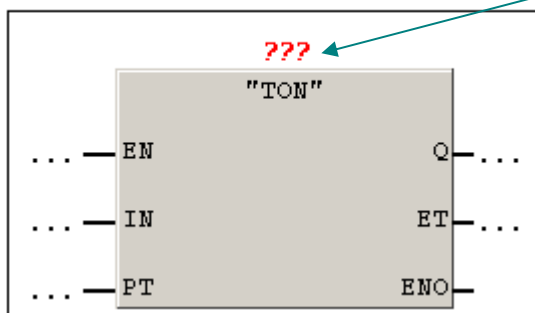
// перевод мс -> sec
// time_period - выходной параметр (REAL)
DTR
L      1.000000e+003
/R
T      #time_period

// запомнить время для след. измерения
L      #curent_tick
T      #prev_time

```

## Применение IEC-таймера SFB4 (ON Delay)

**Network 5:** ON Delay - таймер



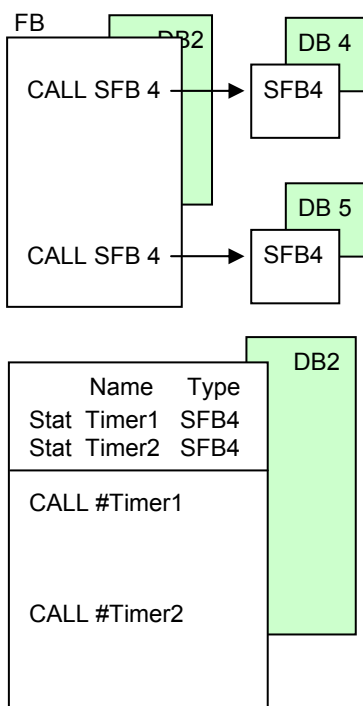
Т.к. SFB4 ("TON") - функциональный блок, то необходимо задать экземплярный блок DB.

### Входные параметры:

**IN** (BOOL) - запуск отсчета времени.  
**PT** (TIME) - время задержки в диапазоне 0 - +24d20h31m23s647ms.

### Выходные параметры

**Q** (BOOL) - состояние таймера.  
**ET** (TIME) - истекшее время с момента запуска.



Если внутри пользовательского FB выполняется вызов нескольких таймеров SFB4, то для каждого вызова необходимо указывать отдельный DB-экземпляр. При этом, если пользовательский FB имеет параметры и вызывается более одного раза, то при каждом вызове нужно передавать блоку FB дополнительные параметры в виде списка внутренних DB-экземпляров.

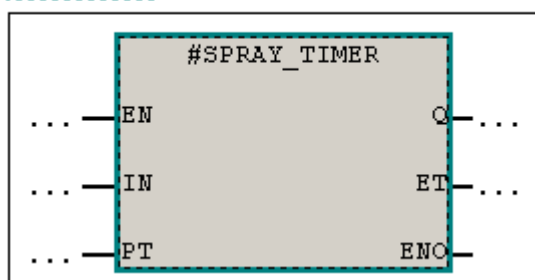
Для такого случая лучше использовать метод мультиэкземпляров.

Многоэкземплярная модель позволяет использовать только один DB экземпляр (родительский DB) для передачи параметров FB/SFB, вызов которых выполняется внутри родительского FB.

Мультиэкземпляры создаются путем объявления статических переменных с типом конкретного FB/SFB.

Для вызова блока FB/SFB в STL методом мультиэкземпляра в инструкции CALL необходимо указать имя статической переменной.

**Network 9:** Мультиэкземпляр



Для вызова блока FB/SFB в FBD/LAD методом мультиэкземпляра необходимо:

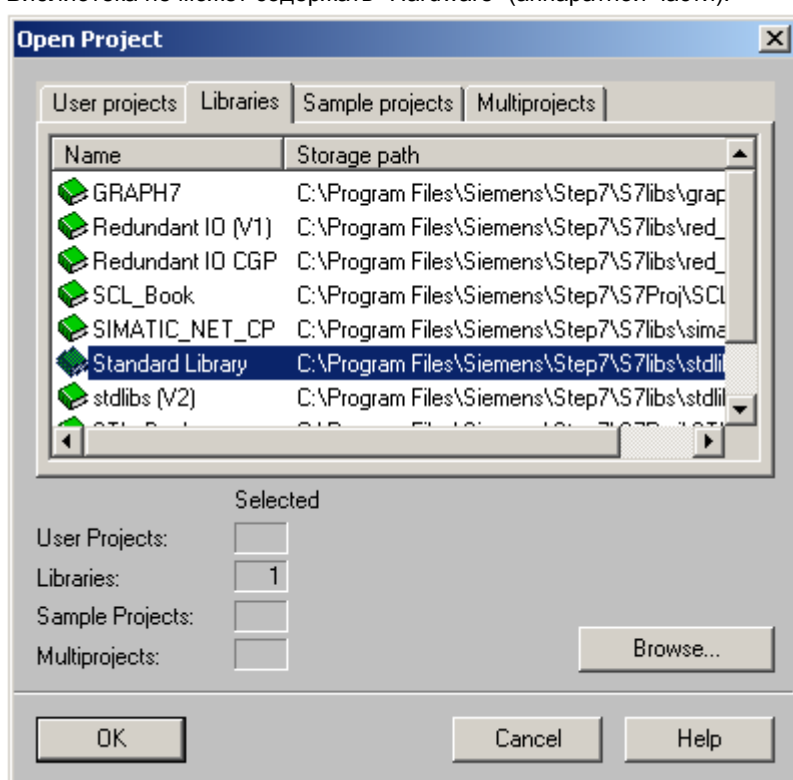
1. Вызвать блок обычным способом.
2. Выделить блок и вызвать функцию контекстного меню *Change to Multiple Instance Call..*
3. В окне "Define Multiple Instance Call" ввести имя статической переменной с типом данного FB/SFB (в примере SPRAY\_TIMER - статическая переменная с типом SFB4).

## Библиотеки

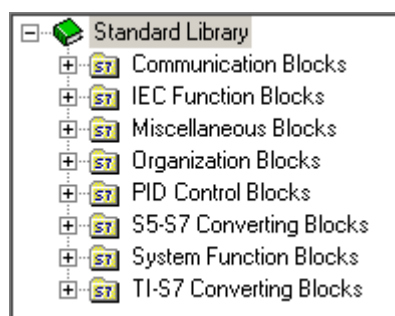
Библиотеки служат для хранения многократно используемых программных компонентов SIMATIC S7. Библиотека состоит из одной или нескольких папок S7-программ.

Программные блоки из библиотек не могут прямо передаваться в CPU и тестироваться. Это означает, что сначала Вы должны перенести блоки в свой проект.

Библиотека не может содержать "Hardware" (аппаратной части).



## Конфигурация и содержание стандартной S7- библиотеки



Communication Blocks: функции для передачи данных в сетях Profibus, Ethernet.

IEC Converting Blocks: блоки для функций IEC.

Organization Blocks: все организационные блоки S7-300/400.

PID Control Blocks: блоки PID-управления.

S5-S7 Converting Blocks: стандартные блоки, которые требуются при преобразовании S5-Programs к S7.

System Function Blocks: все системные функции S7-300/400.

TI-S7 Converting Blocks: различные стандартные функции, например, масштабирование аналоговых величин и т.д.

## Примеры использования функций библиотеки IEC

OB100 : "Complete Restart"

**Network 1:** Преобразование

```
CALL "DT_DATE"
IN :=#OB100_DATE_TIME
RET_VAL:=#data_local
NOP 0
```

FC6

Если запуск CPU был выполнен 01.04.2008, то блок FC1 не вызывается.

Функция FC6 "DT\_DATE" (IEC) преобразует информацию из переменной типа DT в переменную с типом Date.

Входной параметр IN – переменная OB100\_DATE\_TIME (время вызова OB).

Выходное значение RET\_VAL –: целое число, показывающее количество дней, прошедших с 1.1.1990 – локальная переменная OB100 с именем "data\_local" и типом Date.

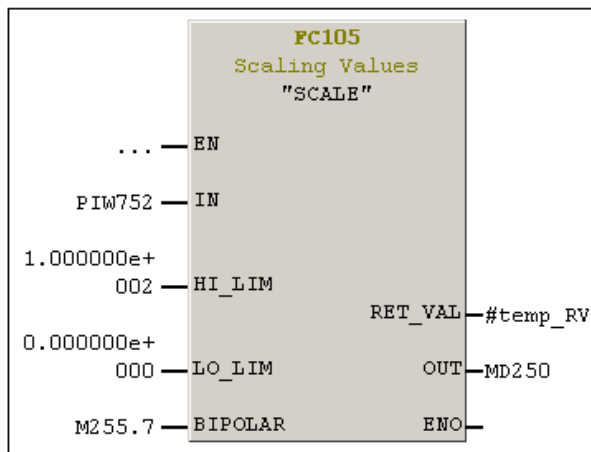
**Network 2:** Сравнение

```
L D#2009-4-1
L #data_local
==I
BEC
CALL "Hot water tank"
```

FC1

## Масштабирование аналоговой величины

**Network 3:** Масштабирование



Функция FC105 из библиотеки TI-S7 Converting Blocks может быть использована для масштабирования аналоговой величины.

**Входные параметры:**

**IN** (INT) - аналоговая величина.

**HI\_LIM** (REAL) - верхний предел физического диапазона, соответствующий номинальному значению аналоговой величины.

**LO\_LIM** (REAL) - нижний предел физического диапазона, соответствующий минимальному значению аналоговой величины.

**BIPOLAR** (BOOL) - лог. "1" задает двупольный диапазон.

**Выходные параметры**

**OUT** (REAL) – результат преобразования.

Формула преобразования:

$$OUT = [ ((FLOAT(IN) - K1) / (K2 - K1)) * (HI\_LIM - LO\_LIM) ] + LO\_LIM, \text{ где}$$

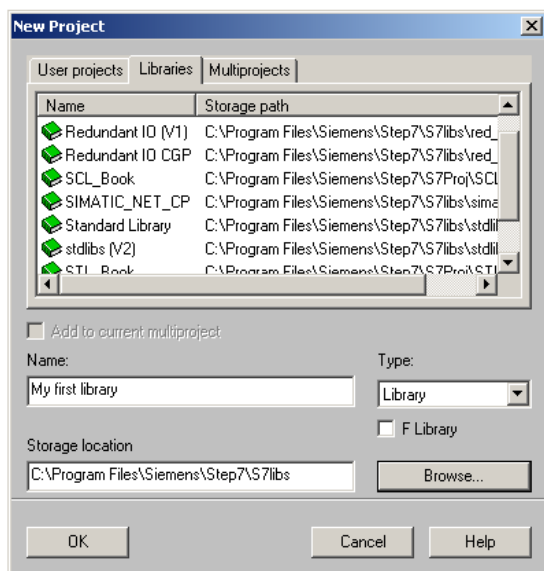
если  $BIPOLAR=1$ :  $K1 = -27648.0$ ;  $K2 = +27648.0$

$BIPOLAR=0$ :  $K1 = 0.0$ ;  $K2 = +27648.0$

Функция FC106 (UNSCALE) из библиотеки TI-S7 Converting Blocks выполняет обратное преобразование и может использоваться для выдачи аналогового сигнала через модуль аналогового вывода. Формула преобразования:

$$OUT = [ ((IN - LO\_LIM) / (HI\_LIM - LO\_LIM)) * (K2 - K1) ] + K1$$

## Создание библиотек пользователя



Для создания новой библиотеки выберите меню *File -> New -> Libraries* и введите новое имя.

Вставьте в библиотеку новый объект S7-программу : *Insert New Object -> S7 program*.

Теперь Вы можете скопировать любой блок в папку библиотеки или создать новый привычным образом.

## Заключение

Закончен обзор вопросов, включенных в программу учебного курса S7-PROF2. Участники данного курса подробно познакомились со следующими вопросами:

- Структурное программирование. Использование блоков FC и FB
- Блоки данных (DB). Доступ к элементам DB. Комплексные типы данных
- Организационные блоки (OB). Программные и аппаратные прерывания
- Типы адресации данных. Косвенная адресация
- Расширенные средства тестирования программ
- Системные функции (SFC/SFB)
- Стандартные библиотеки и библиотеки пользователя

Для тех, кто хотел бы получить знания по технологии проектирования S7 программ, предлагаем вашему вниманию наш уникальный курс S7-TECH.

В курсе S7-TECH рассматриваются методы формального проектирования программ. Что дает пользователю знание и умение пользоваться данными методами?

- Программирование без ошибок
- Простая реализация программы на основе алгоритма
- Легкость проведения изменений
- Встроенный контроль выполнения каждого шага алгоритма
- Сокращение издержек на разработку и отладку программ

Предлагаем вашему вниманию курс S7-BUS, в котором рассматриваются базовые принципы и практические задачи по коммуникационному обмену в сетях ASI, Profibus DP и Industrial Ethernet.

Желаем успехов в дальнейшем освоении “секретов” программирования и проектирования систем SIMATIC S7!

Всю информацию по семинарам Вы сможете найти в Интернете по адресу <http://www.promautomatic.ru/training.html>.



## Приложение

### Система команд S7-CPU (по категориям)

| Bit logic |                         |
|-----------|-------------------------|
| A         | And                     |
| AN        | And Not                 |
| O         | Or                      |
| ON        | Or Not                  |
| X         | Exclusive Or            |
| XN        | Exclusive Or Not        |
| FN        | Edge Negative           |
| FP        | Edge Positive           |
| ( )       | Nesting                 |
| =         | Assign                  |
| R         | Reset                   |
| S         | Set                     |
| NOT       | Negate RLO              |
| SET       | Set RLO (=1)            |
| CLR       | Clear RLO (=0)          |
| SAVE      | Save RLO in BR Register |

| Word logic |                          |
|------------|--------------------------|
| AW         | AND Word                 |
| AD         | AND Double Word          |
| OW         | OR Word                  |
| OD         | OR Double Word           |
| XOW        | Exclusive Or Word        |
| XOD        | Exclusive Or Double Word |

| Convert |                                     |
|---------|-------------------------------------|
| BTI     | BCD to Integer                      |
| ITB     | Integer to BCD                      |
| BTD     | BCD to Integer                      |
| ITD     | Integer to Double Integer           |
| DTB     | Double Integer to BCD               |
| DTR     | Double Integer to Floating-Point    |
| INVI    | Ones Complement Integer             |
| INVD    | Ones Complement Double Integer      |
| NEGI    | Twos Complement Integer             |
| NEGD    | Twos Complement Double Integer      |
| NEGR    | Negate Floating-Point Number        |
| CAW     | Change Byte Sequence in ACC1 Word   |
| CAD     | Change Byte Sequence in ACC1 Double |
| RND     | Round                               |
| TRUNC   | Truncate                            |
| RND-    | Round to Lower Double Integer       |
| RND+    | Round to Upper Double Integer       |

| Compare if true RLO = 1    |                                    |
|----------------------------|------------------------------------|
| <b>==I ==D</b>             | ACC2 is equal to ACC1              |
| <b>&lt;&gt;I &lt;&gt;D</b> | ACC2 is not equal to ACC1          |
| <b>&gt;I &gt;D</b>         | ACC2 is greater than to ACC1       |
| <b>&gt;=I &gt;=D</b>       | ACC2 is greater then equal to ACC1 |
| <b>&lt;I &lt;D</b>         | ACC2 is less then to ACC1          |
| <b>&lt;=I &lt;=D</b>       | ACC2 is less then equal to ACC1    |

| Math         |                                   |
|--------------|-----------------------------------|
| <b>+</b>     | Add Integer Constant (16, 32-Bit) |
| <b>+I +D</b> | Add ACC1 and ACC2                 |
| <b>-I -D</b> | Subtract ACC1 from ACC2           |
| <b>*I *D</b> | Multiply ACC1 and ACC2            |
| <b>/I /D</b> | Divide ACC2 by ACC1               |
| <b>MOD</b>   | Division Remainder Double Integer |

| Floating Point Math |                   |
|---------------------|-------------------|
| <b>ABS</b>          | Absolute Value    |
| <b>ACOS</b>         | Arc Cosine        |
| <b>ASIN</b>         | Arc Sine          |
| <b>ATAN</b>         | Arc Tangent       |
| <b>COS</b>          | Cosine of Angles  |
| <b>EXP</b>          | Exponential Value |
| <b>LN</b>           | Natural Logarithm |
| <b>SIN</b>          | Sine of Angles    |
| <b>SQR</b>          | Square            |
| <b>SQRT</b>         | Square Root       |
| <b>TAN</b>          | Tangent of Angles |

| Shift/Rotate |                            |
|--------------|----------------------------|
| <b>SSI</b>   | Shift Sign Integer         |
| <b>SSD</b>   | Shift Sign Double Integer  |
| <b>SLW</b>   | Shift Left Word            |
| <b>SRW</b>   | Shift Right Word           |
| <b>SLD</b>   | Shift Left Double Word     |
| <b>SRD</b>   | Shift Right Double Word    |
| <b>RLD</b>   | Rotate Left Double Word    |
| <b>RRD</b>   | Rotate Right Double Word   |
| <b>RLDA</b>  | Rotate ACC1 Left via CC 1  |
| <b>RRDA</b>  | Rotate ACC1 Right via CC 1 |

| Jumps       |                         |
|-------------|-------------------------|
| <b>JU</b>   | Jump Unconditional      |
| <b>JL</b>   | Jump to Labels          |
| <b>JC</b>   | Jump if RLO = 1         |
| <b>JCN</b>  | Jump if RLO = 0         |
| <b>JCB</b>  | Jump if RLO = 1 with BR |
| <b>JNB</b>  | Jump if RLO = 0 with BR |
| <b>JB</b>   | Jump if BR = 1          |
| <b>JNBI</b> | Jump if BR = 0          |
| <b>JO</b>   | Jump if OV = 1          |
| <b>JOS</b>  | Jump if OS = 1          |
| <b>JZ</b>   | Jump if Zero            |
| <b>JN</b>   | Jump if Not Zero        |
| <b>JP</b>   | Jump if Plus            |
| <b>JM</b>   | Jump if Minus           |
| <b>JPZ</b>  | Jump if Plus or Zero    |
| <b>JMZ</b>  | Jump if Minus or Zero   |
| <b>JUO</b>  | Jump if Unordered       |
| <b>LOOP</b> | Loop                    |



| Program Control             |                                  |
|-----------------------------|----------------------------------|
| <b>CALL</b>                 | Call FC, FB, SFC, SFB            |
| <i>Example</i>              |                                  |
| <b>CALL FC1 or FB1, DB1</b> |                                  |
| PARAM1 := I0.0              |                                  |
| PARAM2 := "Example".Test    |                                  |
| <b>CC</b>                   | Conditional Call                 |
| <b>UC</b>                   | Unconditional Call               |
| <b>BE</b>                   | Block End                        |
| <b>BEC</b>                  | Block End Conditional            |
| <b>BEU</b>                  | Block End Unconditional          |
| <b>MCR(</b>                 | Save RLO in MCR Stack, Begin MCR |
| <b>)MCR</b>                 | End MCR                          |
| <b>MCRA</b>                 | Activate MCR                     |
| <b>MCRD</b>                 | Deactivate MCR                   |

| Transfer              |   |
|-----------------------|---|
| <b>T</b>              | Transfer  |
| <b>T STW</b>          | Transfer ACC1 into Status Word                              |
| <b>TAR1</b>           | Transfer Address Register 1 to ACC1                         |
| <b>TAR1 &lt;D&gt;</b> | Transfer Address Register 1 to Destination (32-Bit Pointer) |
| <b>TAR1 AR2</b>       | Transfer Address Register 1 to Address Register 2           |
| <b>TAR2</b>           | Transfer Address Register 2 to ACC1                         |
| <b>TAR2 &lt;D&gt;</b> | Transfer Address Register 2 to Destination (32-Bit Pointer) |

| Load                  |  |
|-----------------------|--|
| <b>L</b>              | Load   |
| <b>L STW</b>          | Load Status Word into ACC1                                   |
| <b>LAR1</b>           | Load Address Register 1 from ACC1                            |
| <b>LAR1 &lt;D&gt;</b> | Load Address Register 1 with Double Integer (32-Bit Pointer) |
| <b>LAR1 AR2</b>       | Load Address Register 1 from Address Register 2              |
| <b>LAR2</b>           | Load Address Register 2 from ACC1                            |
| <b>LAR2 &lt;D&gt;</b> | Load Address Register 2 with Double Integer (32-Bit Pointer) |
| <b>CAR</b>            | Exchange Address Register 1 with Address Register 2          |

| Accumulator  |                                    |
|--------------|------------------------------------|
| <b>TAK</b>   | Toggle ACC1 with ACC2              |
| <b>POP</b>   | Pop accumulators                   |
| <b>PUSH</b>  | Push accumulators                  |
| <b>ENT</b>   | Enter ACC Stack                    |
| <b>LEAVE</b> | Leave ACC Stack                    |
| <b>DEC</b>   | Decrement ACC                      |
| <b>INC</b>   | Increment ACC                      |
| <b>+AR1</b>  | Add ACC1 to Address Register 1     |
| <b>+AR2</b>  | Add ACC1 to Address Register 2     |
| <b>BLD</b>   | Program Display Instruction (Null) |
| <b>NOP 0</b> | Null Instruction                   |



| Timers/Counters |   |
|-----------------|---|
| FR              | Enable Timer/Counter (Free)   |
| L               | Load Current Timer/Counter Value into ACC1 as Integer (i.e. L T 32) |
| LC              | Load Current Timer/Counter Value into ACC1 as BCD (i.e. LC T 32)    |
| R               | Reset Timer/Counter   |
| S               | Set Counter Preset Value (i.e. S C 15)                              |
| SD              | On-Delay Timer  |
| SS              | Retentive On-Delay Timer  |
| SP              | Pulse Timer   |
| SF              | Off-Delay Timer   |
| SE              | Extended Pulse Timer  |
| CD              | Counter Down  |
| CU              | Counter Up  |

| OBs   |  |
|-------|--|
| 1     | Main Program Scan                      |
| 10-17 | Time of Day                            |
| 20-23 | Time Delay                             |
| 30-38 | Cyclic (Periodic)                      |
| 40-47 | Hardware                               |
| 80    | Time Error                             |
| 81    | Power Supply Error                     |
| 82    | Diagnostic Interrupt                   |
| 83    | Insert/Remove Module Interrupt         |
| 84    | CPU Hardware Fault                     |
| 85    | Program Cycle Error                    |
| 86    | Rack Failure - Missing Profibus device |
| 87    | Communication Error                    |
| 100   | Warm restart                           |
| 101   | Hot restart                            |
| 102   | Cold restart                           |
| 121   | Programming Error                      |
| 122   | I/O Access Error                       |

| Data Blocks |                                    |
|-------------|------------------------------------|
| OPN         | Open a Data Block                  |
| CDB         | Exchange Shared DB and Instance DB |
| L DBLG      | Load Length of Shared DB in ACC1   |
| L DBNO      | Load Number of Shared DB in ACC1   |
| L DILG      | Load Length of Instance DB in ACC1 |
| L DINO      | Load Number of Instance DB in ACC1 |

